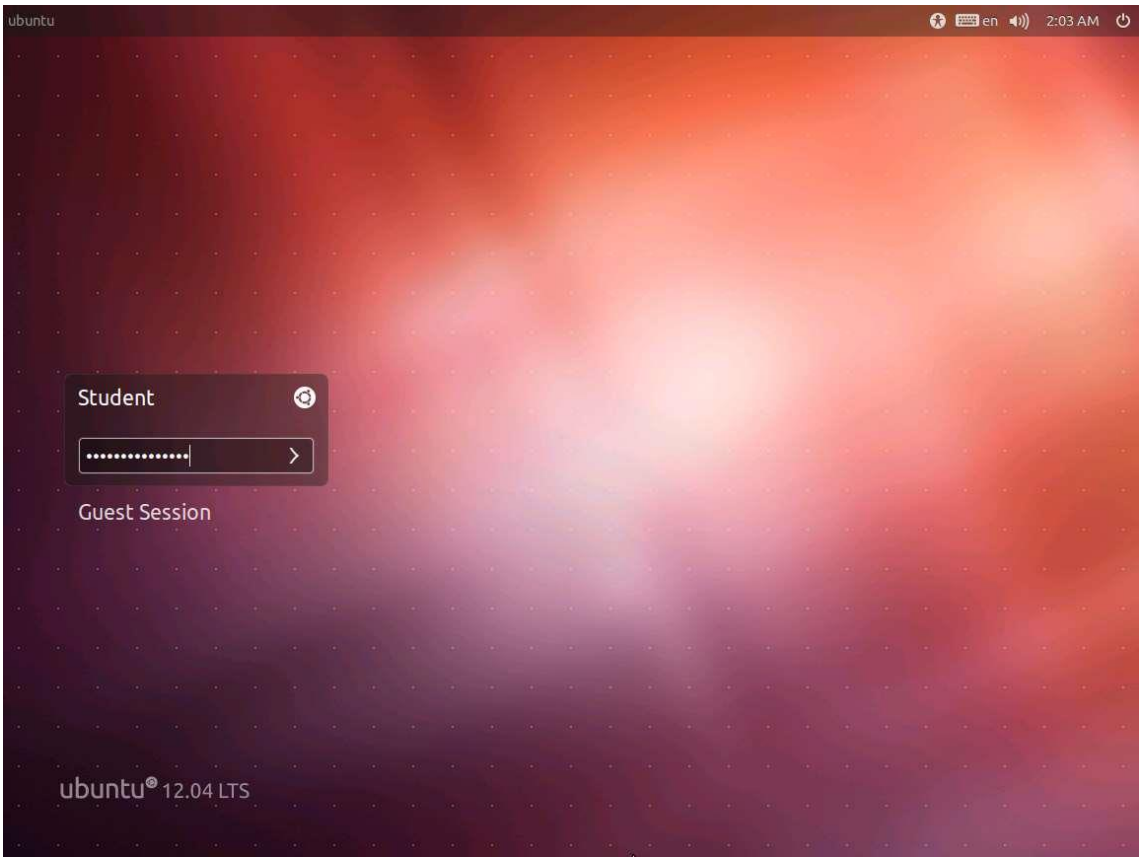
Exercise 9: Linux Kernel ROP and ROPgadget

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。

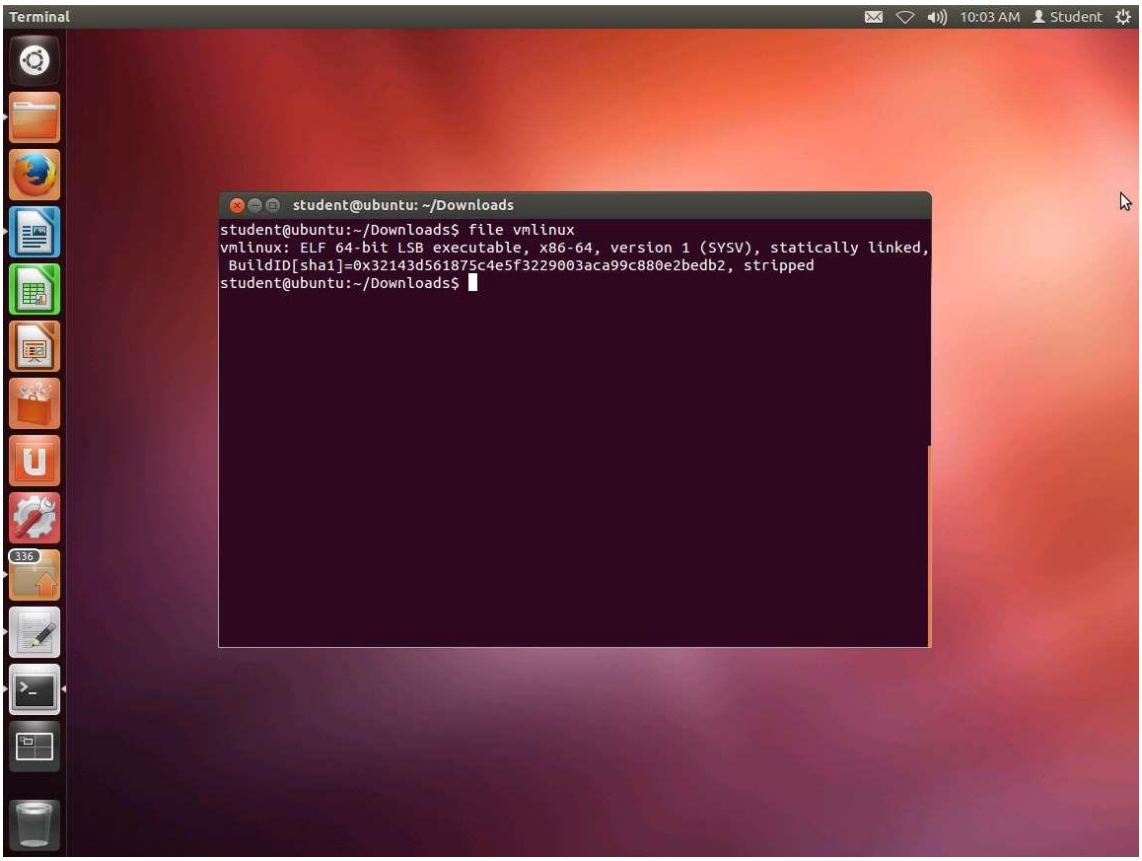
```
7. !/bin/sh
8. # SPDX-License-Identifier: GPL-2.0-only
9. # ------------------------------------------------
   --------------------
10.  # extract-vmlinux - Extract uncompressed vmlinux
   from a kernel image
11.  #
12.  # Inspired from extract-ikconfig
13.  # (c) 2009,2010 Dick Streefland
   <dick@streefland.net>
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

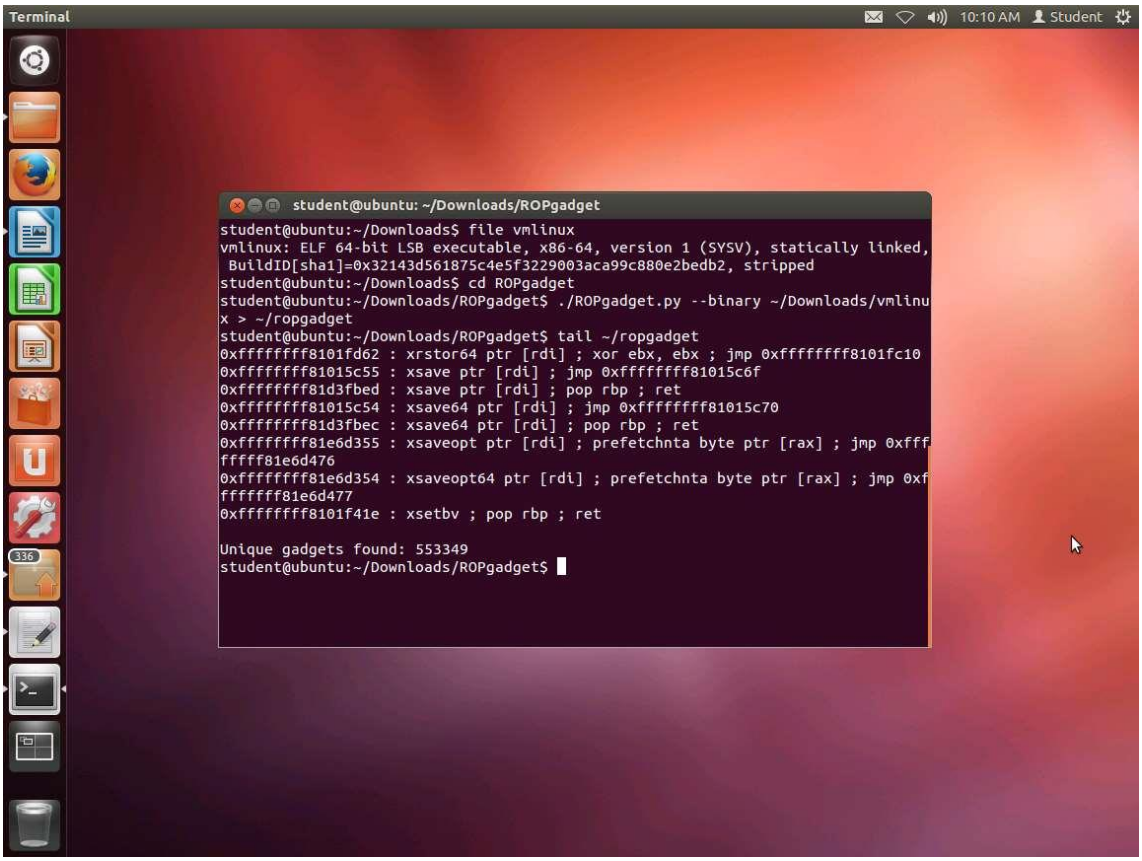
71. ☐ If we run objdump against the uncompressed kernel image, and then grep for gadgets, it will not provide that many, since we are working with aligned addresses, which do suffice in many cases.

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**.

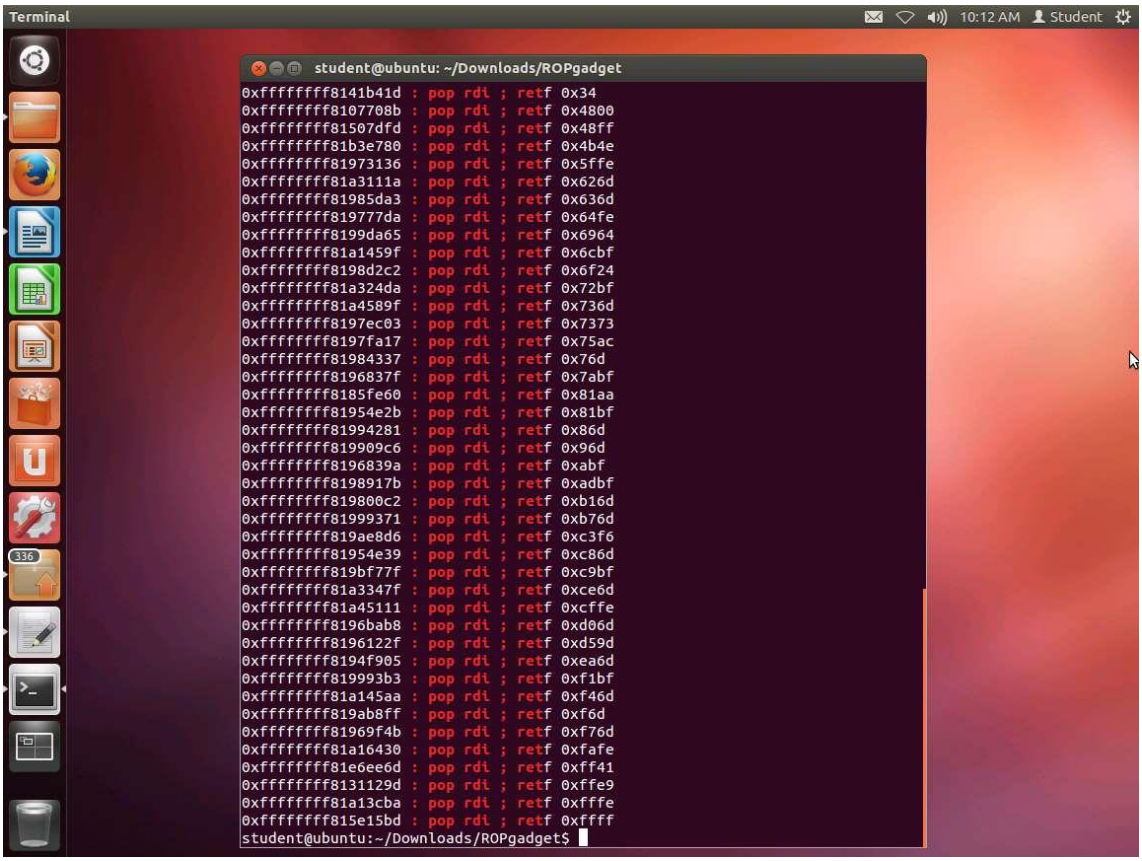
75. ☐ Now, enter **tail ~/ropgadget**. An example of the output of this command is shown in the following screenshot.



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.

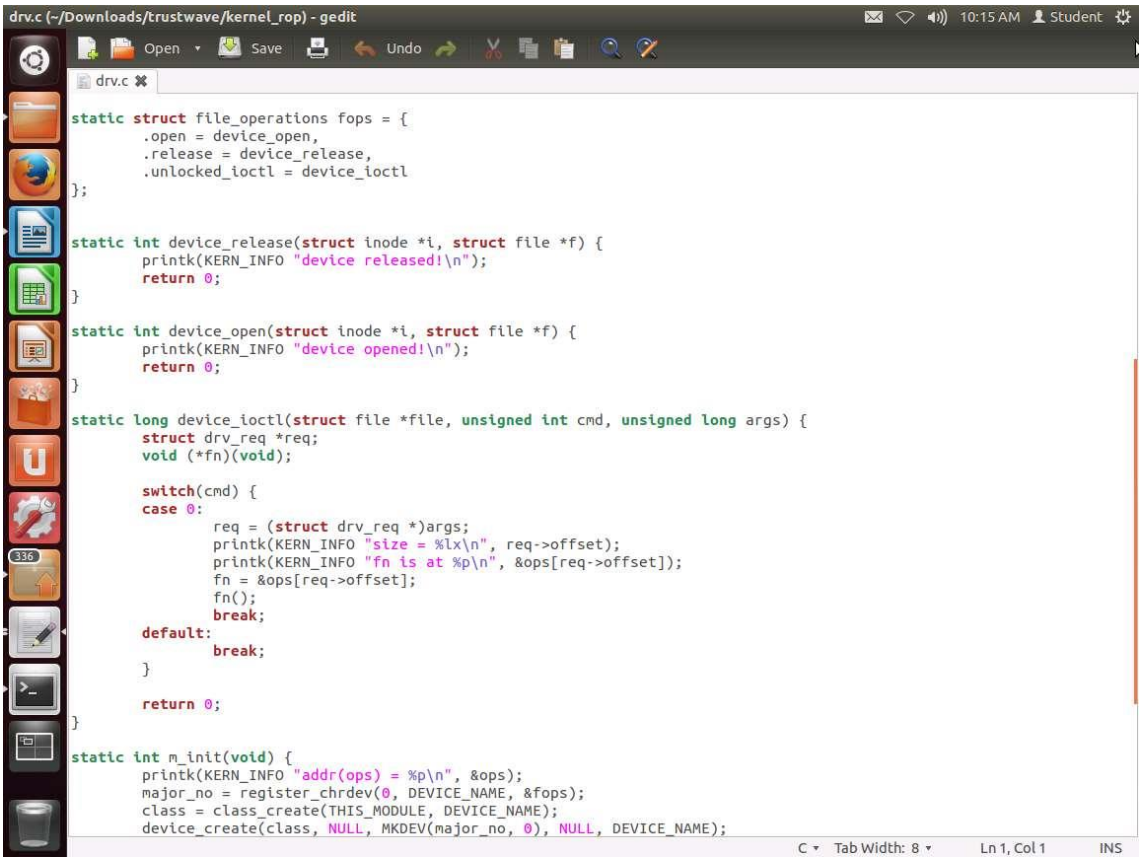


79. 明らかに、これらのいずれかをガジェットとして使用できますが、何を選択しても、スタックポインタがその番号で高いメモリから低いメモリに移動するため、その番号を使用してリターンを構築する必要があります。

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ープログラムを使用できます。ドライバーのコードを次のスクリーンショットに示します。



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。

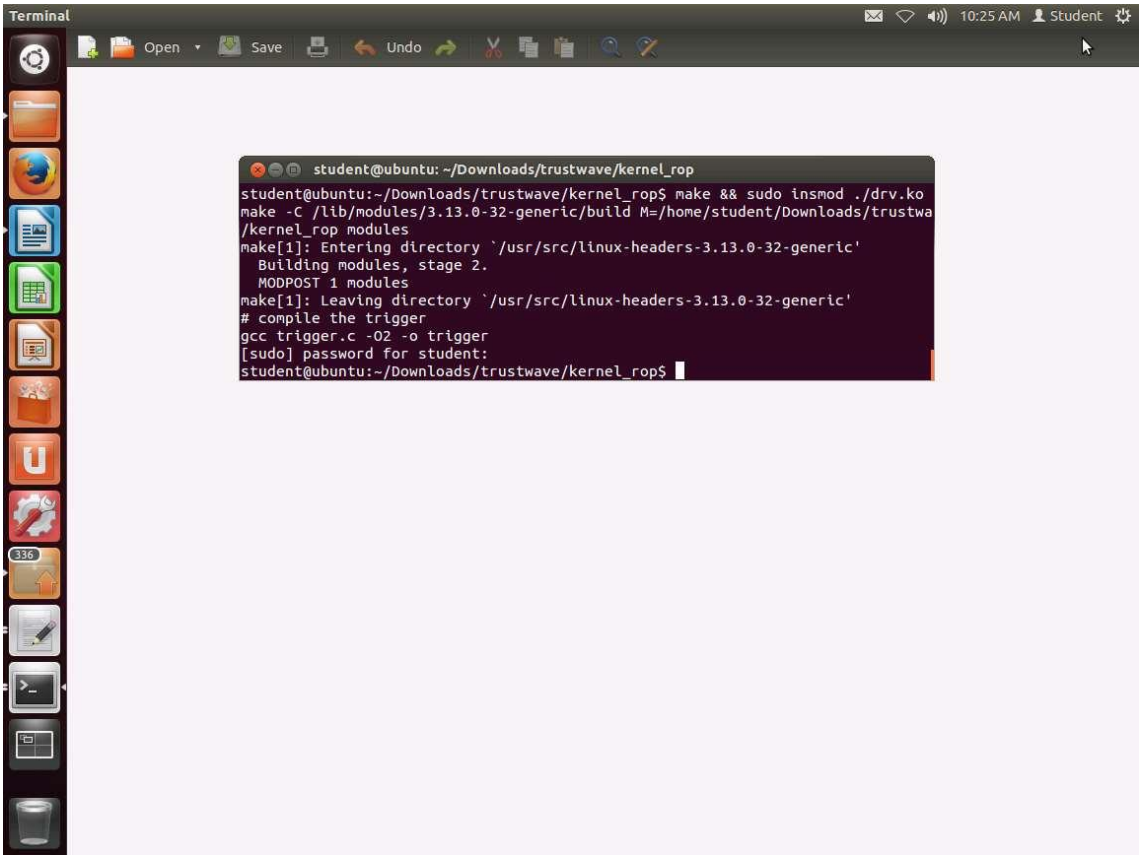
85. ☐ 次に、コードが既にコンパイルされている場合に備えて、コードを削除します。

    A.rm Drv.ko

    B. エルスモード |grep drv

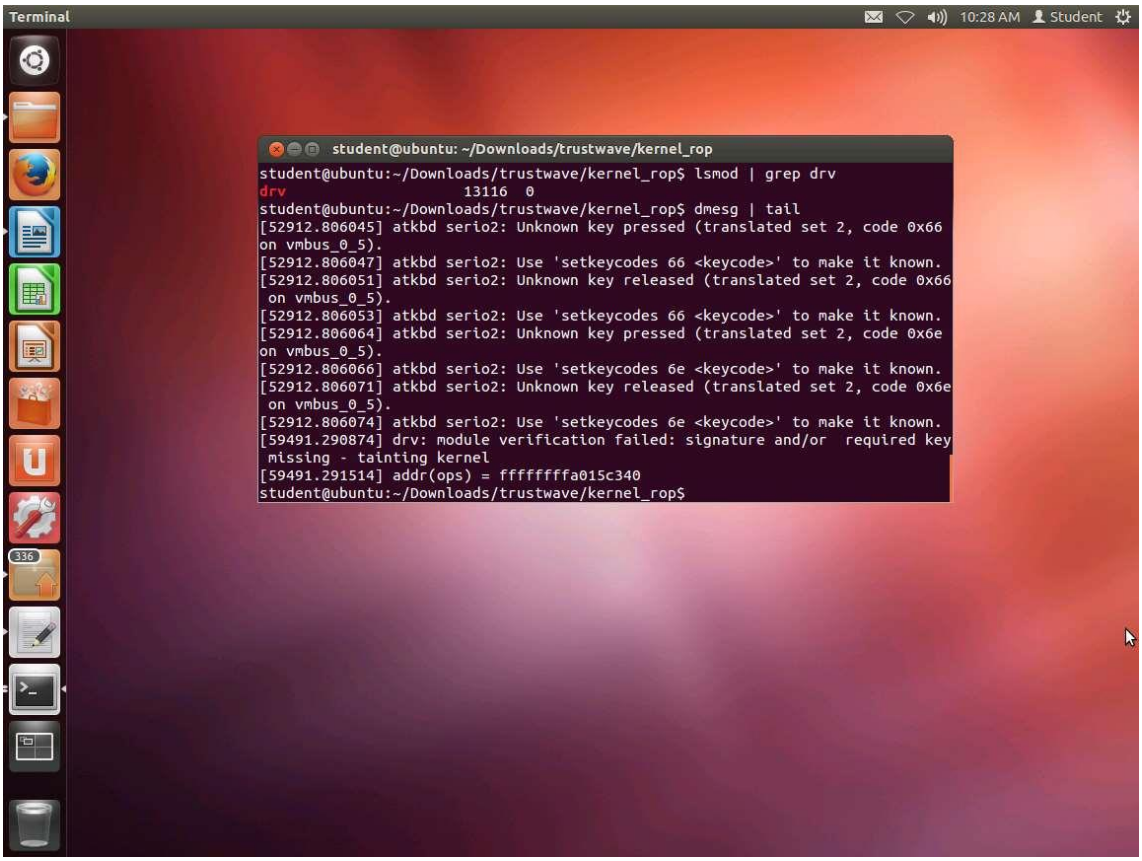
86. ☐ 出力がある場合は、drv 汚染されたモジュールがロードされていることを意味します。その場合は、アンロードする必要があります。Enter **sudo rmmod drv**.これにより、カーネルモジュールが削除されます。

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。



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.



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.

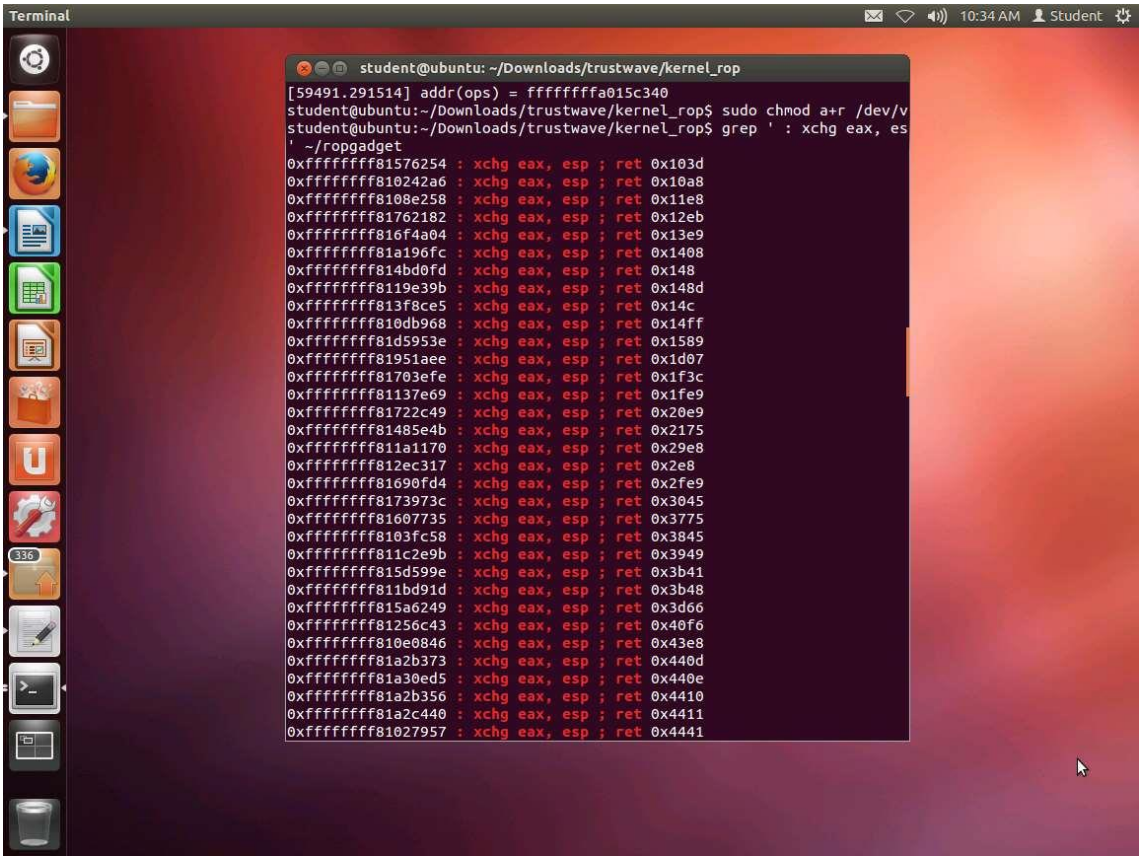
93. ☐ So far, we have demonstrated how to find useful ROP gadgets and build a privilege escalation ROP chain.

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., 0xffffffffXXXXXXXX), this stack pivot instruction will set the lower 32 bits of $rXx (0xXXXXXXXX which is a user-space address) as the new stack pointer. Since the **$rax** value is known right before executing **fn()**, we know exactly where our new user-space stack will be and mmap it accordingly.

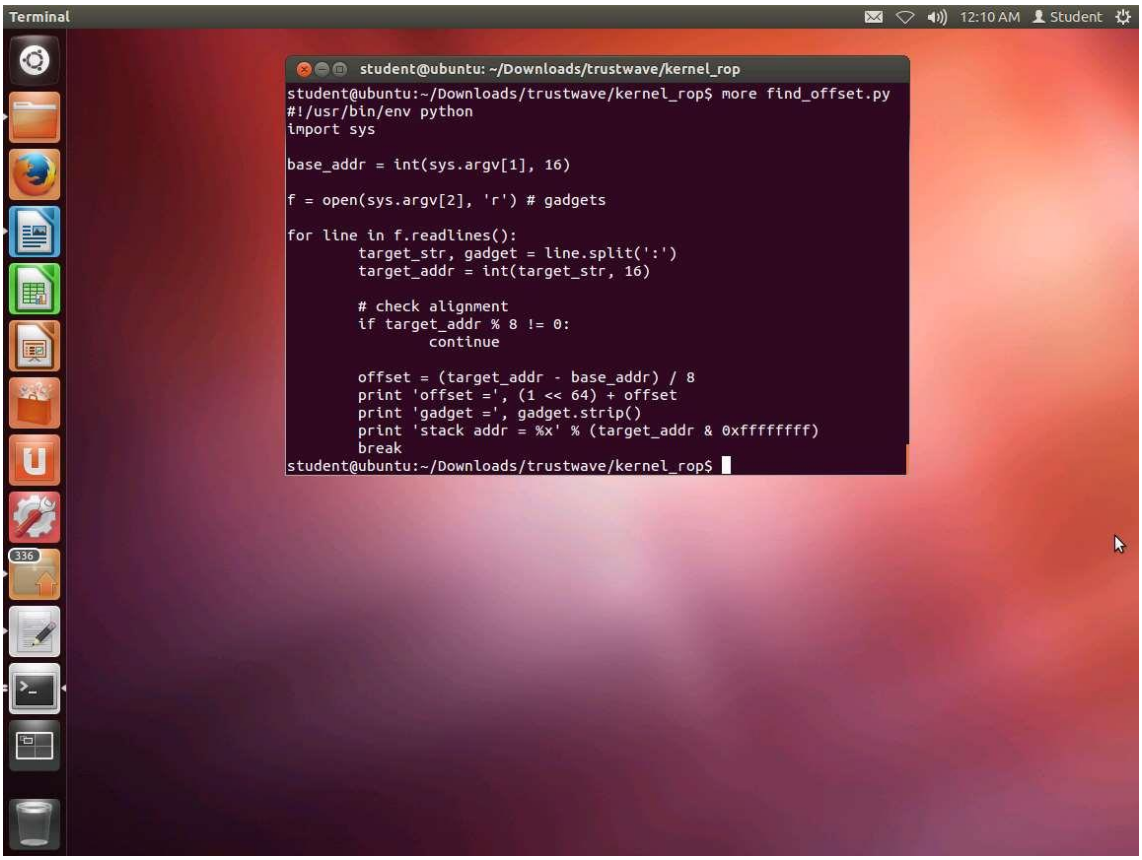
96. ☐ We are looking for the **xchg** instruction to select as our ROP gadget.

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



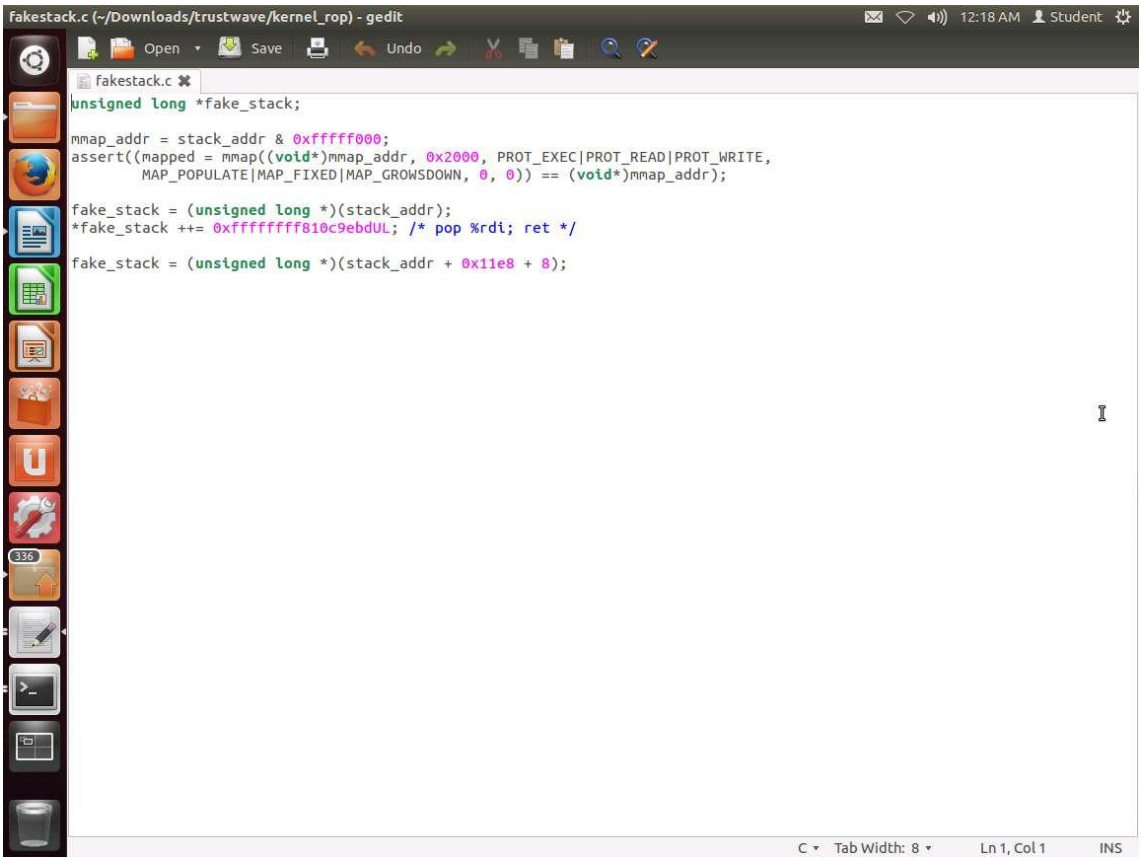
97. ☐ Please note when choosing a stack pivot gadget that it needs to be aligned by 8 bytes (since the ops is the array of 8-byte pointers and its base

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



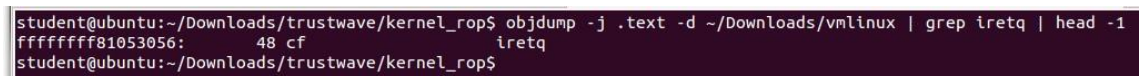
98. Let us run the script. Enter **cat ~/ropgadget | grep ': xchg eax, esp ; ret' > gadgets**. An example of the output of this command is shown in the following screenshot.

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.



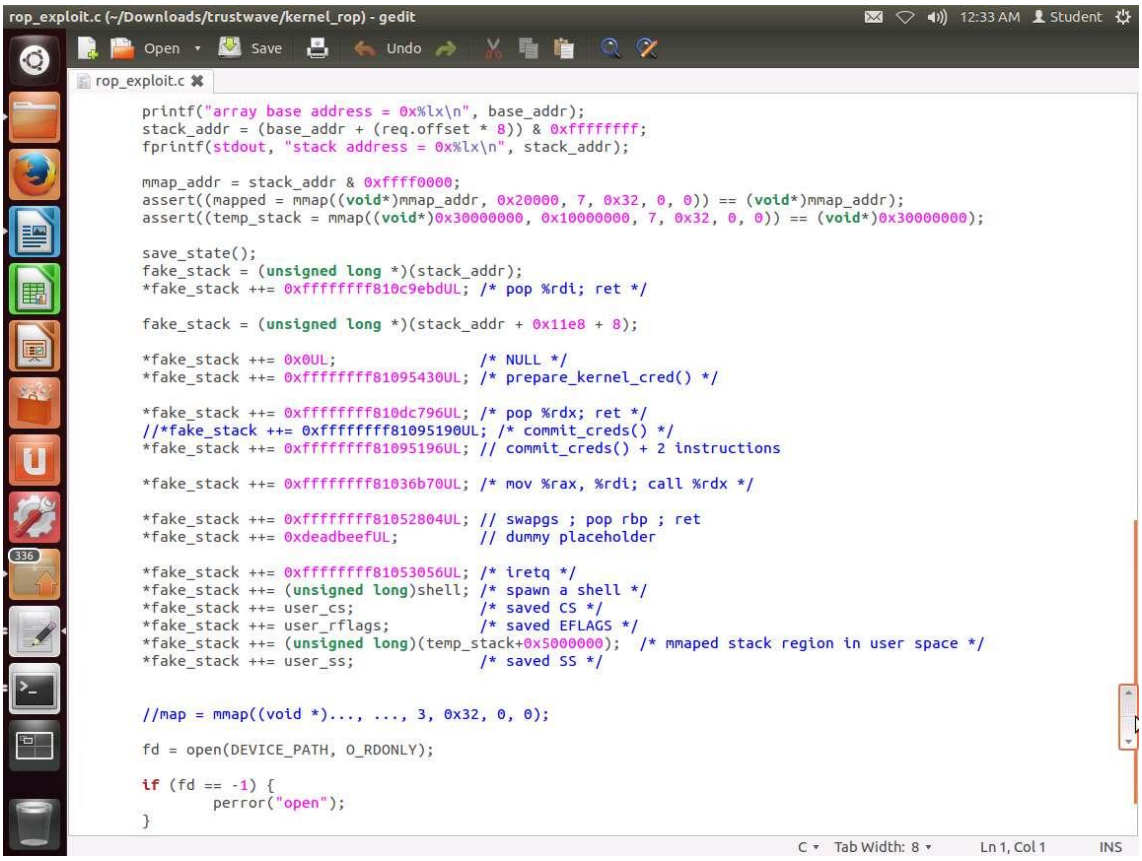
100. ☐ The RET instruction in the stack pivot has a numeric operand; since there is no argument; it pops the return address off the stack and jumps to it. The second ROP gadget is for cleaning up properly.

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.

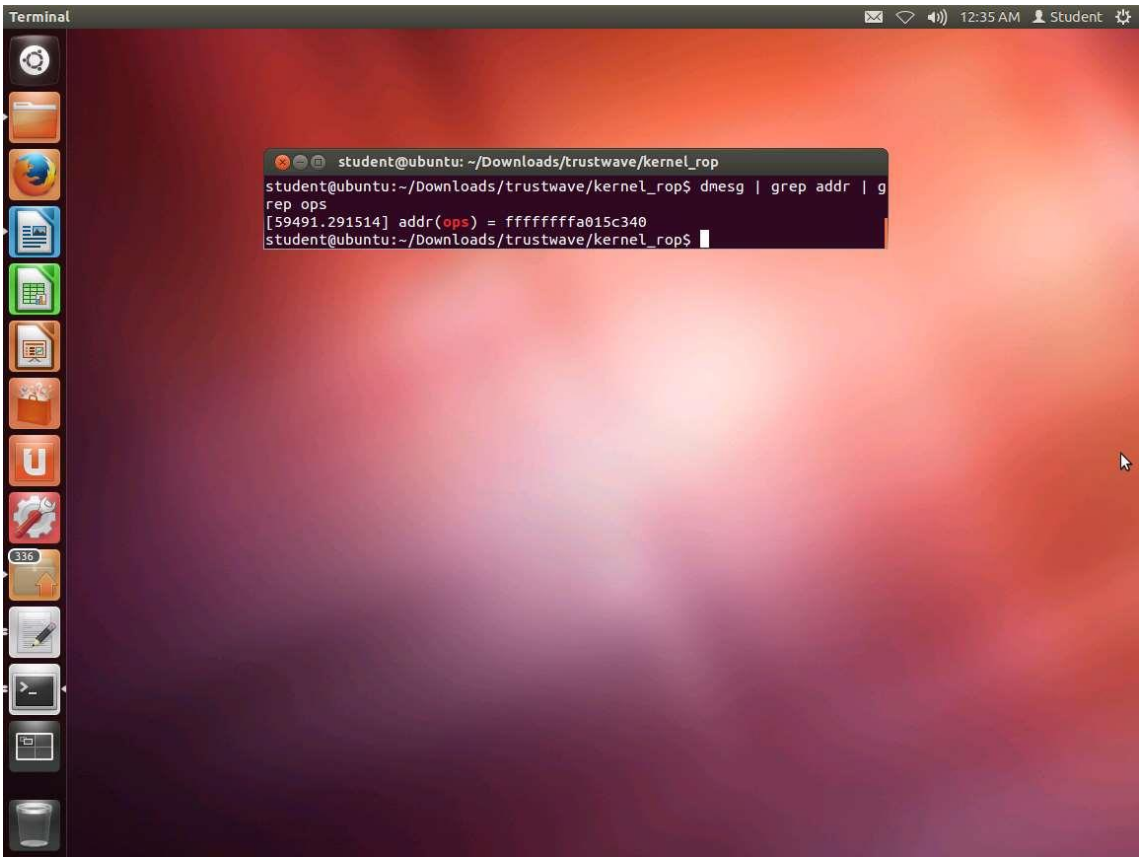


102. ☐ We need more since we are on a 64-bit system. We need swapgs since this is executed at the entry to a kernel space routing and is required before returning to user space.

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.

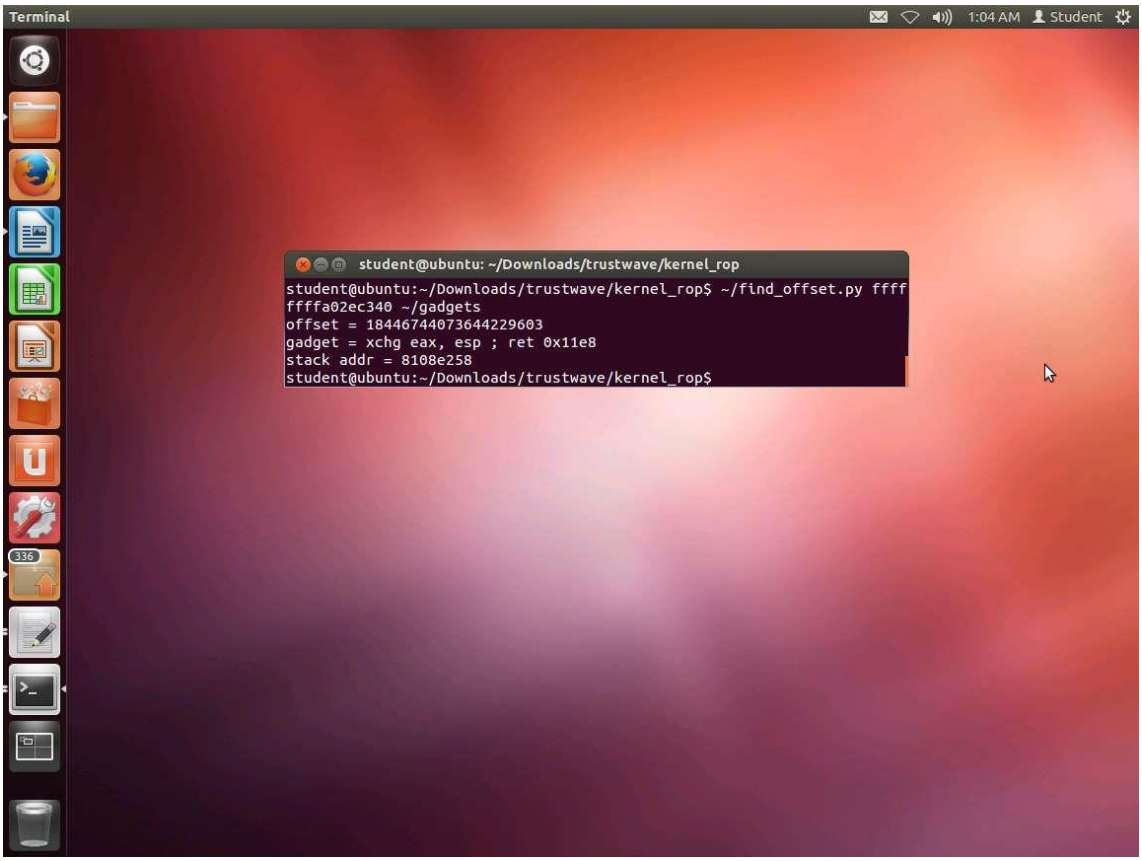


104. ☐ It is now time to check. Enter **dmesg | grep addr | grep ops**. An example of this is shown in the following screenshot.



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 ffffffffa02e9340 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 = 0xffffffffa02e9340
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。