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

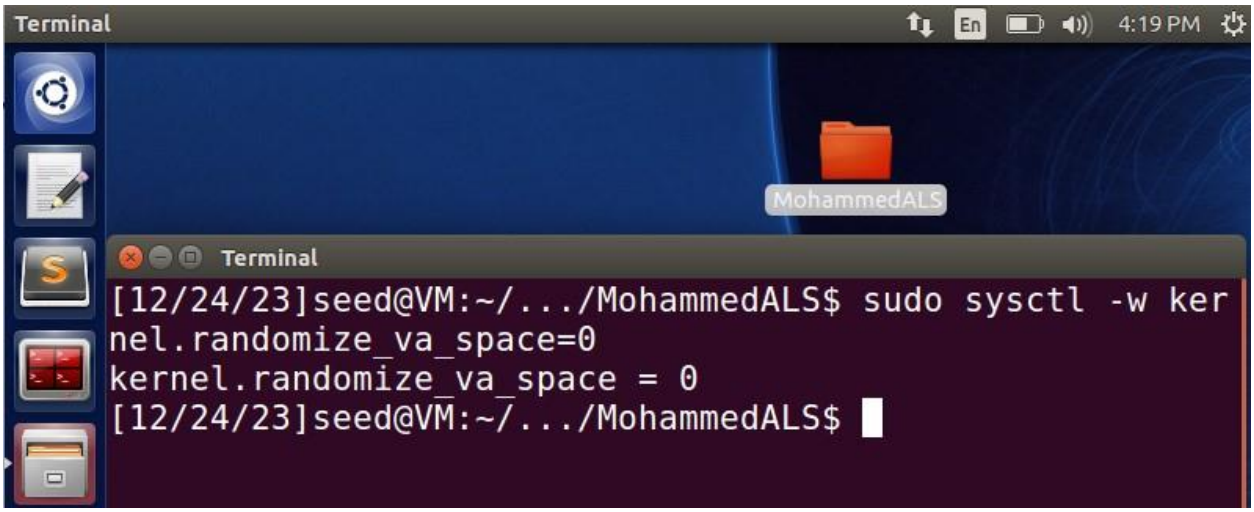
Student:

Mohammed Wahaq Alsahli

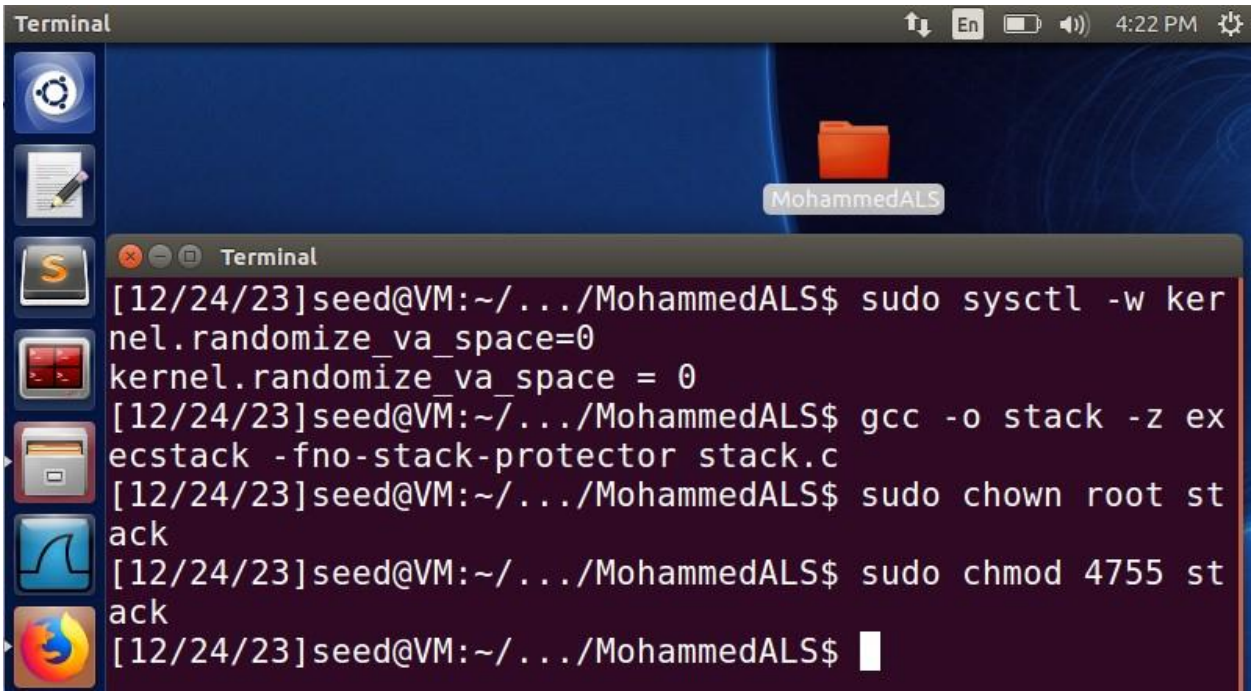
Task 1: You need to adjust the variables of the program accordingly and fill in any missing code to fulfill the buffer overflow attack.

We have to set up the environment

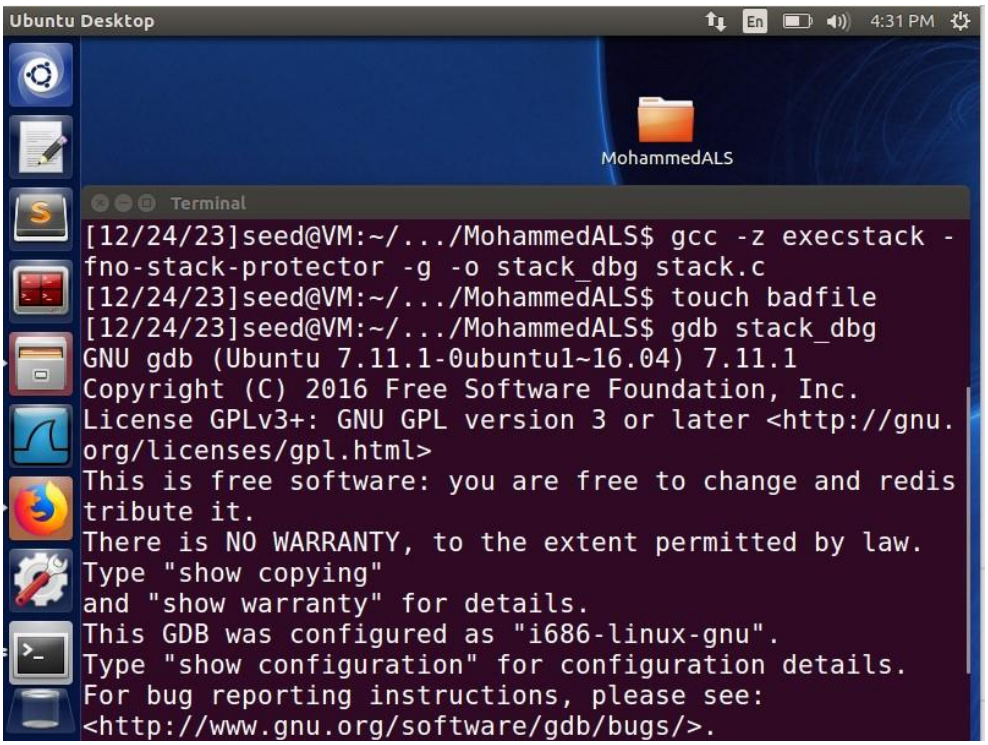
Turning off address randomization:



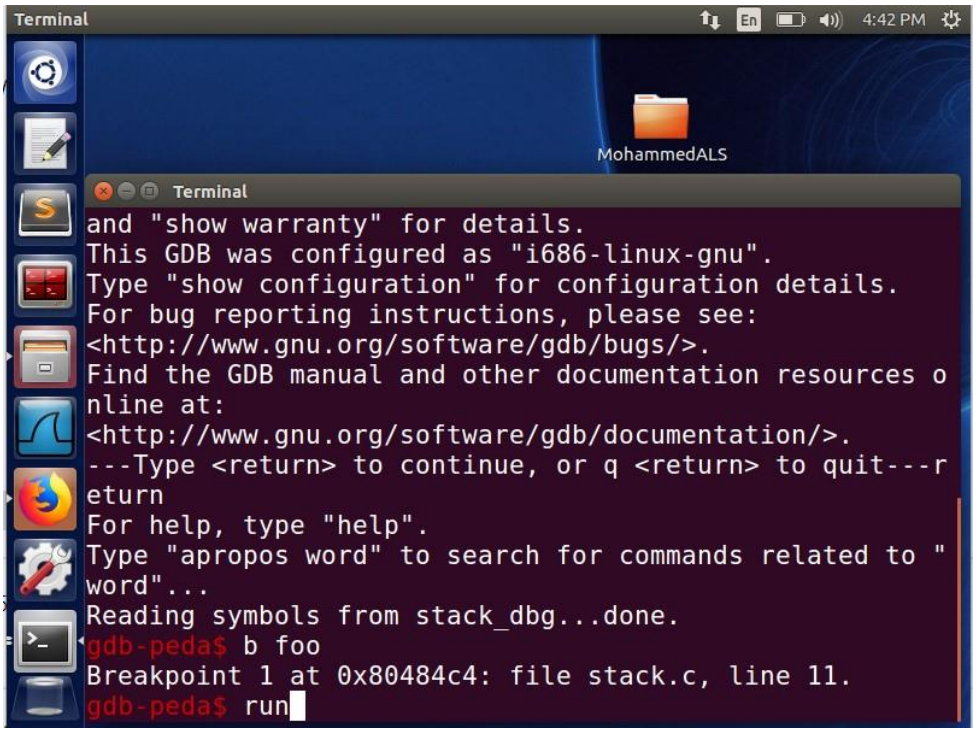
Compile set-uid root version of stack.c:

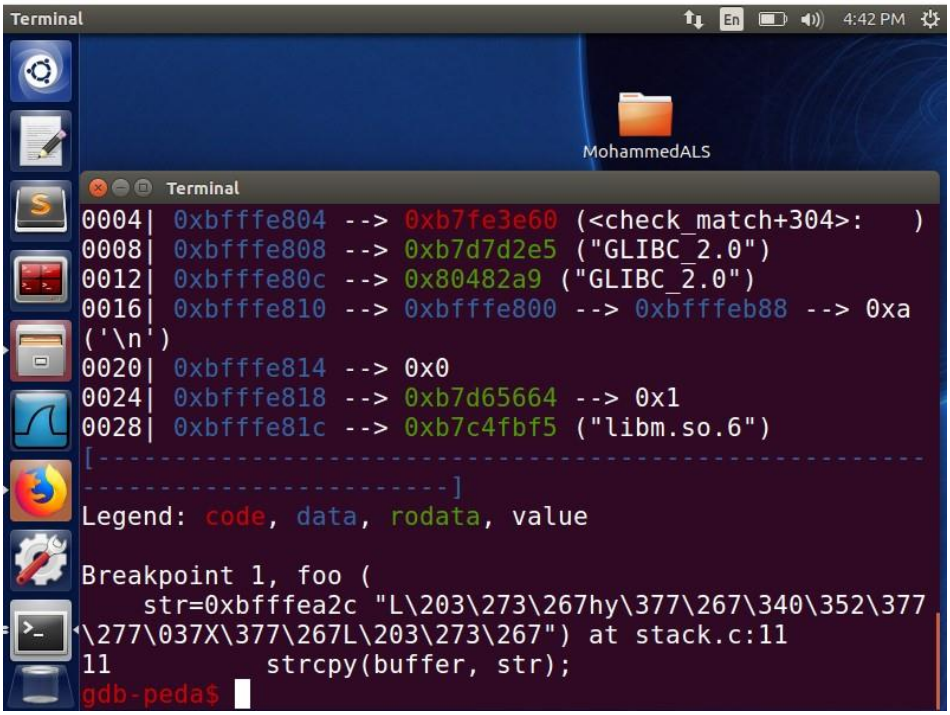


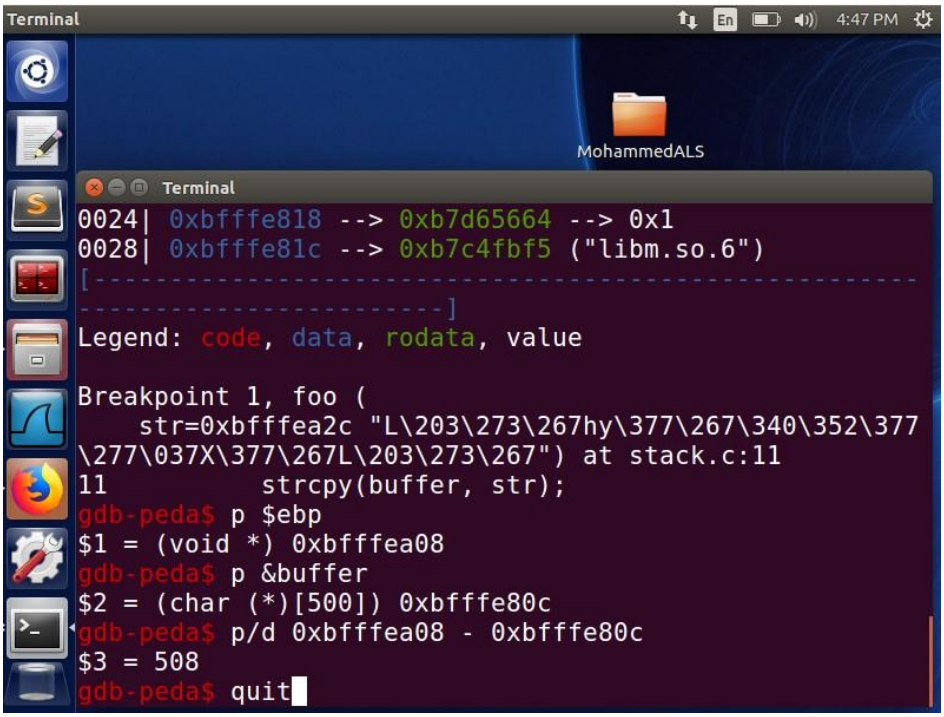
distance between the base of the buffer and return address:



setting a breakpoint for function foo() using gdb:



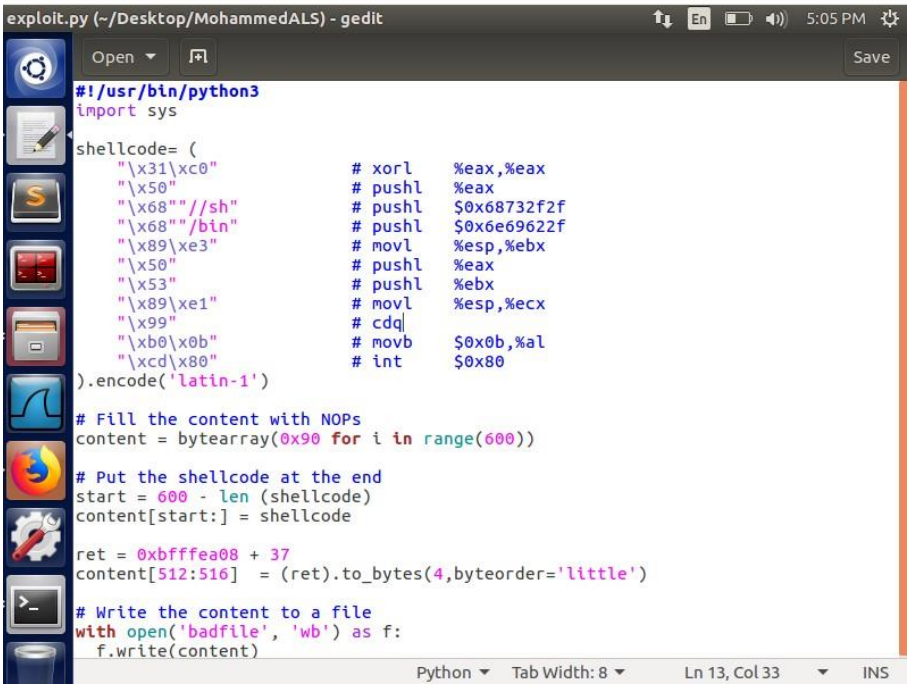




p $ebp = prints the value of register ebp, = 0xbfffea08.

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

the distance is 508 + 4 = 512.



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

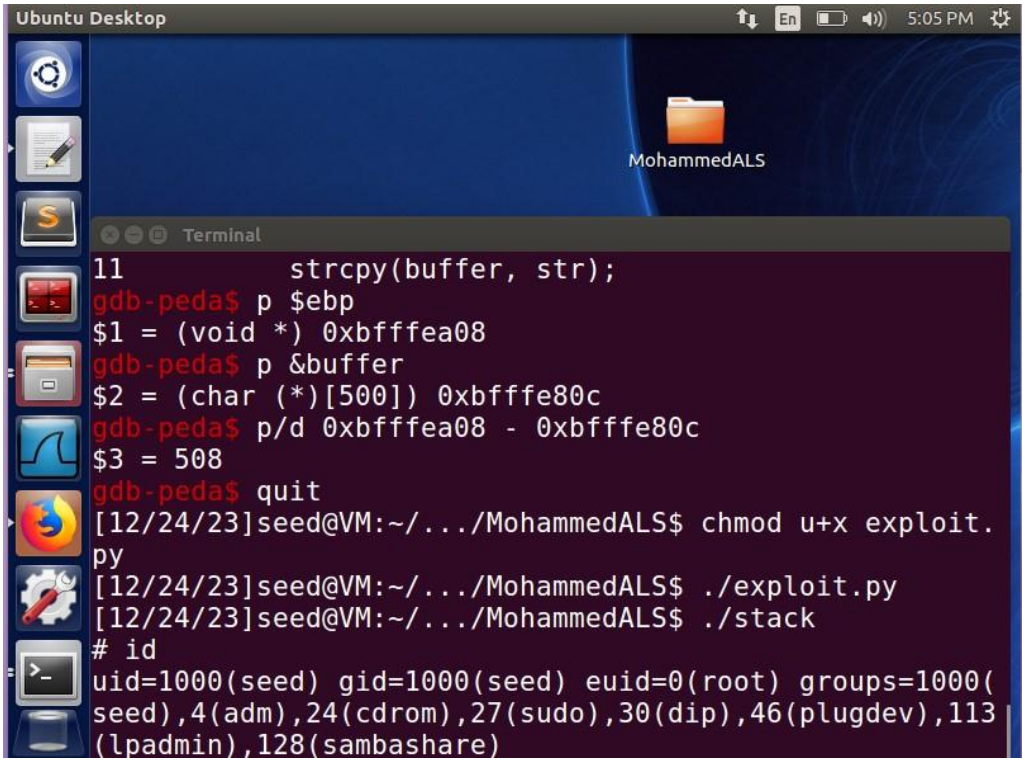
Ret ==0xbfffea08 + 37

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.



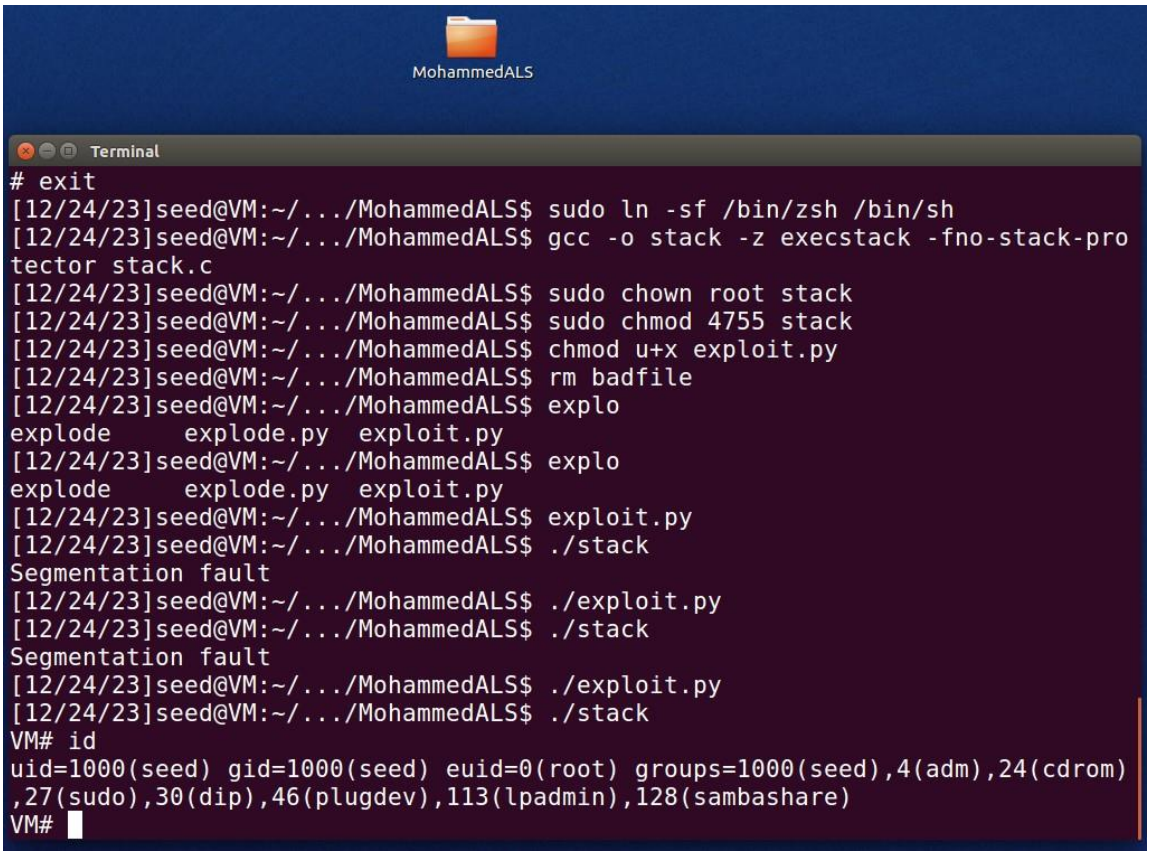
Task Two: Find a way to obtain root shell (read book/slides).

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

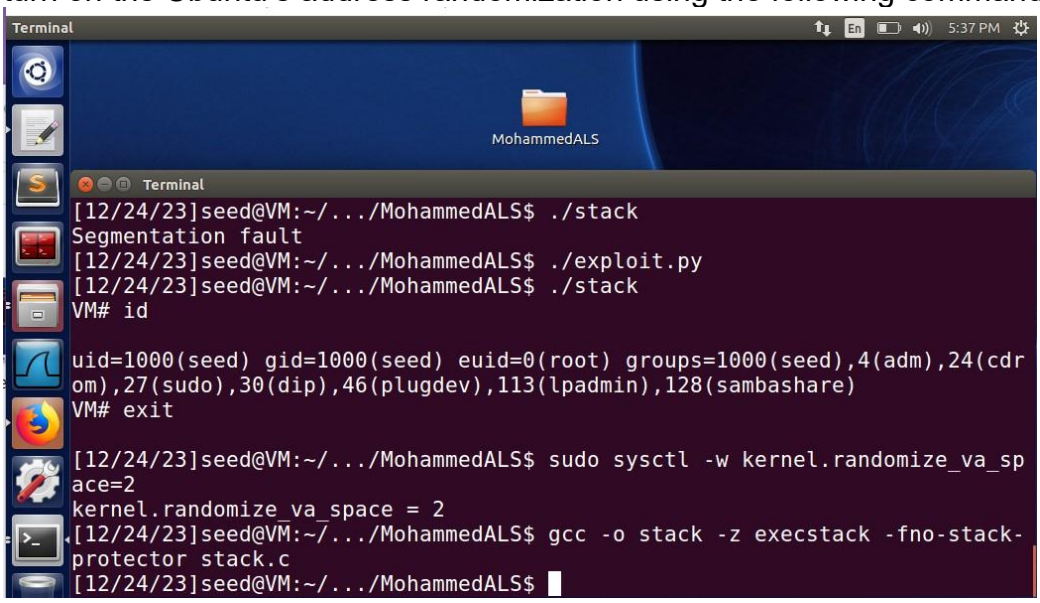


Task3: On 32-bit Linux machines, stacks only have 19 bits of entropy, which means the stack base address can have 2^19 = 524, 288 possibilities. This number is not that high and can be

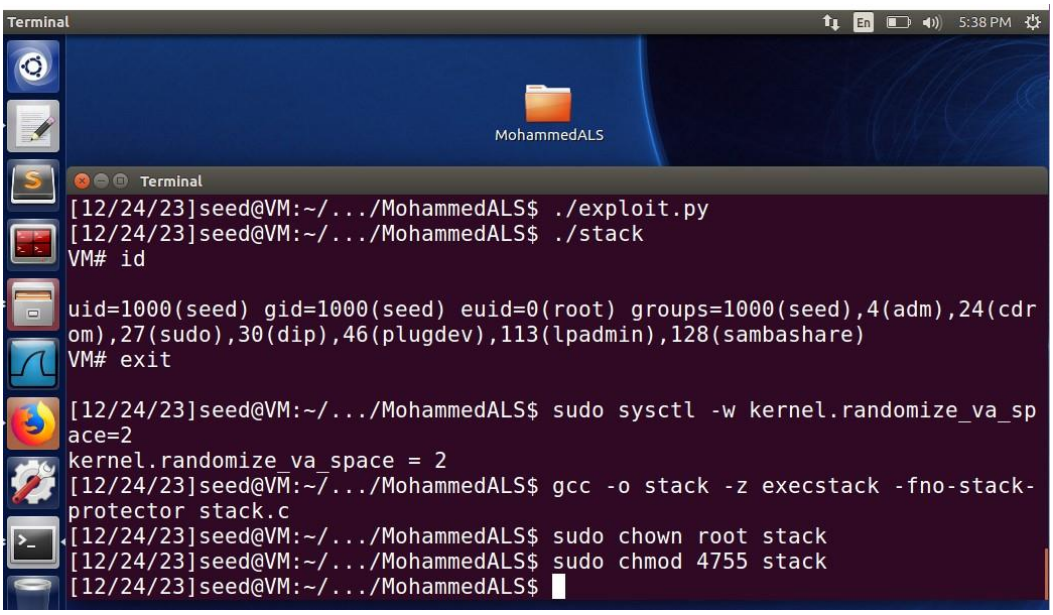
exhausted easily with the brute-force approach. In this task, we use such an approach to defeat the address randomization countermeasure on our 32- bit VM. First, we turn on the Ubuntu's address randomization using the following command:

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

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



Compile set-uid root version of stack.c



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