SNS Assignment - 4 Report

Course: System and Network Security

Course Instructor: Dr. Ashok Das

Date of Submission: 30th March 2021

Name	**Roll Number**
Mohammad Mohsin Husain Rizvi	2020201014
Danial Kafeel	2020201069
Husain Ali Mistry	2020201039
Janmejay Singh	2020201089

Stack Overflow Exploit

- Overwriting return address.

  Firstly a C program is created to build a badfile which is being read by the vulnerable program. So, with the hit and trial method we found the number of bytes after which the return address gets overwritten by us. This is detected when segmentation fault arises and we check the EIP register with our given return address.

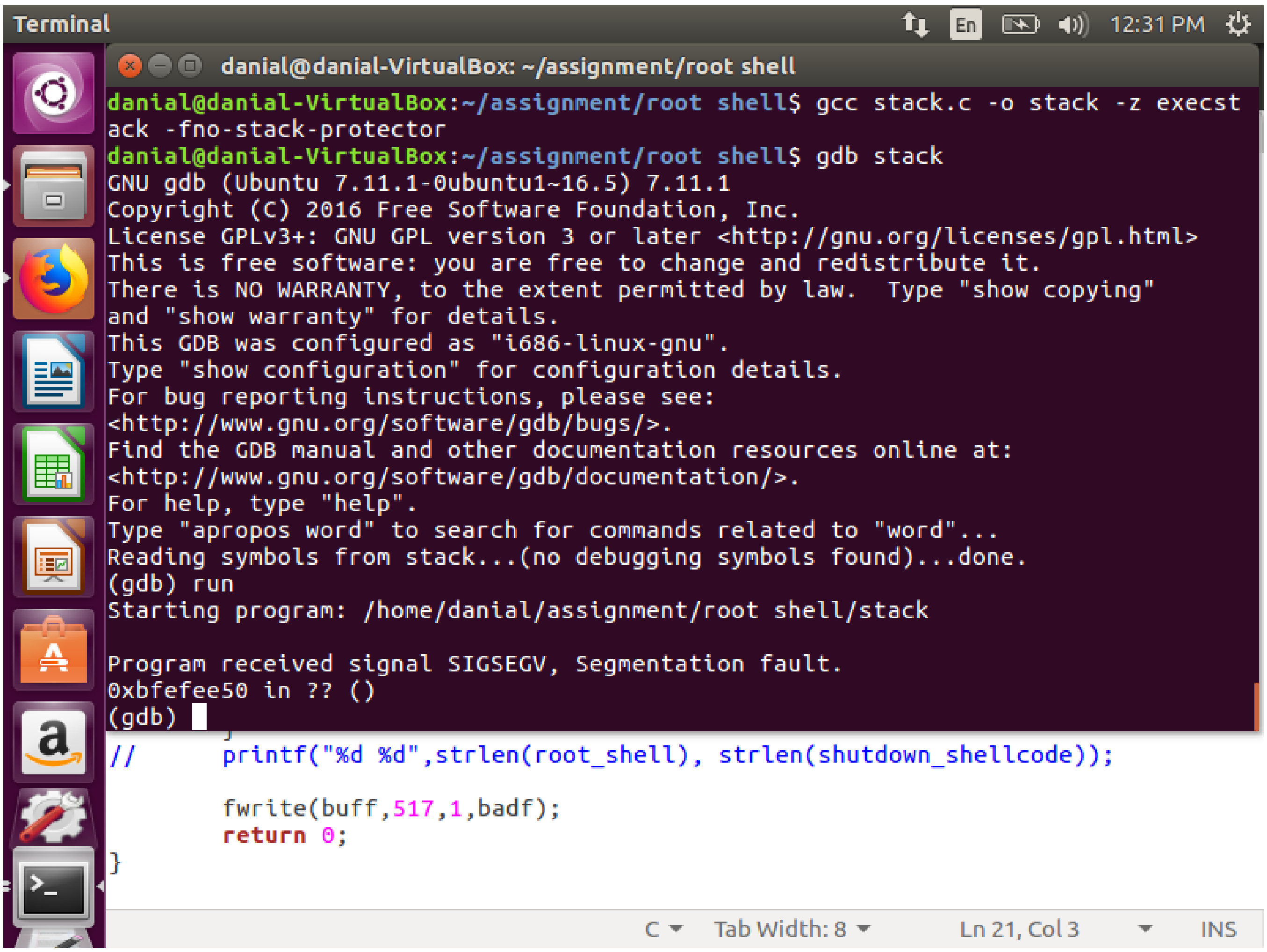


Fig.1 Overwriting return address (0xbfefee50)

- Inserting ShellCode

Now the badfile is created by firstly putting the NOP instructions (i.e. 0x90) 'N' times, where 'N' is calculated as follows:

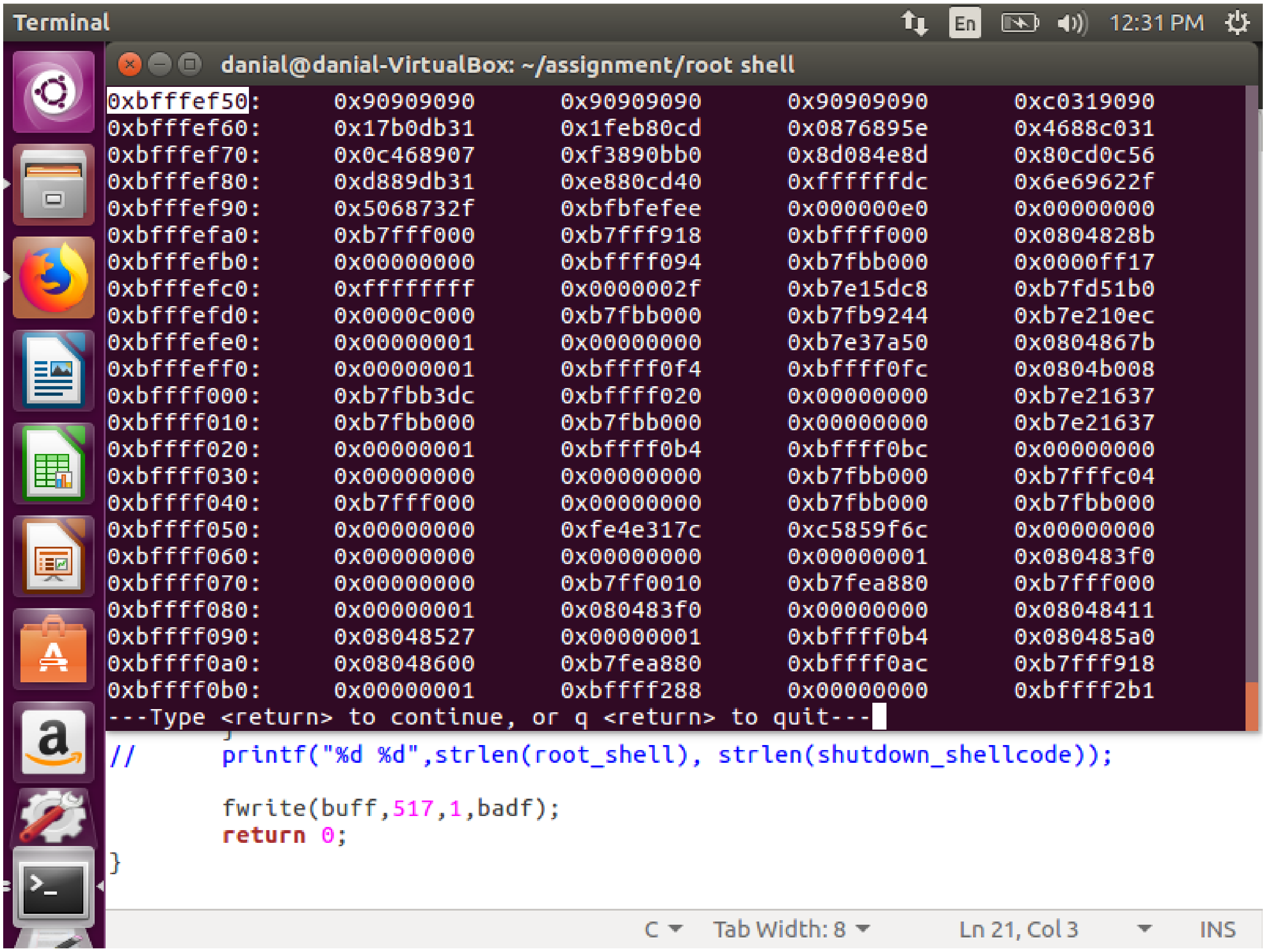
N = No. of Bytes after which the return address gets overwritten - No. of Bytes in shellcode.

Followed by NOPs we put our shell code in the badfile and then the 4 Bytes long return address. This completes our process of creation of badfile which is ready to be fed to vulnerable code.

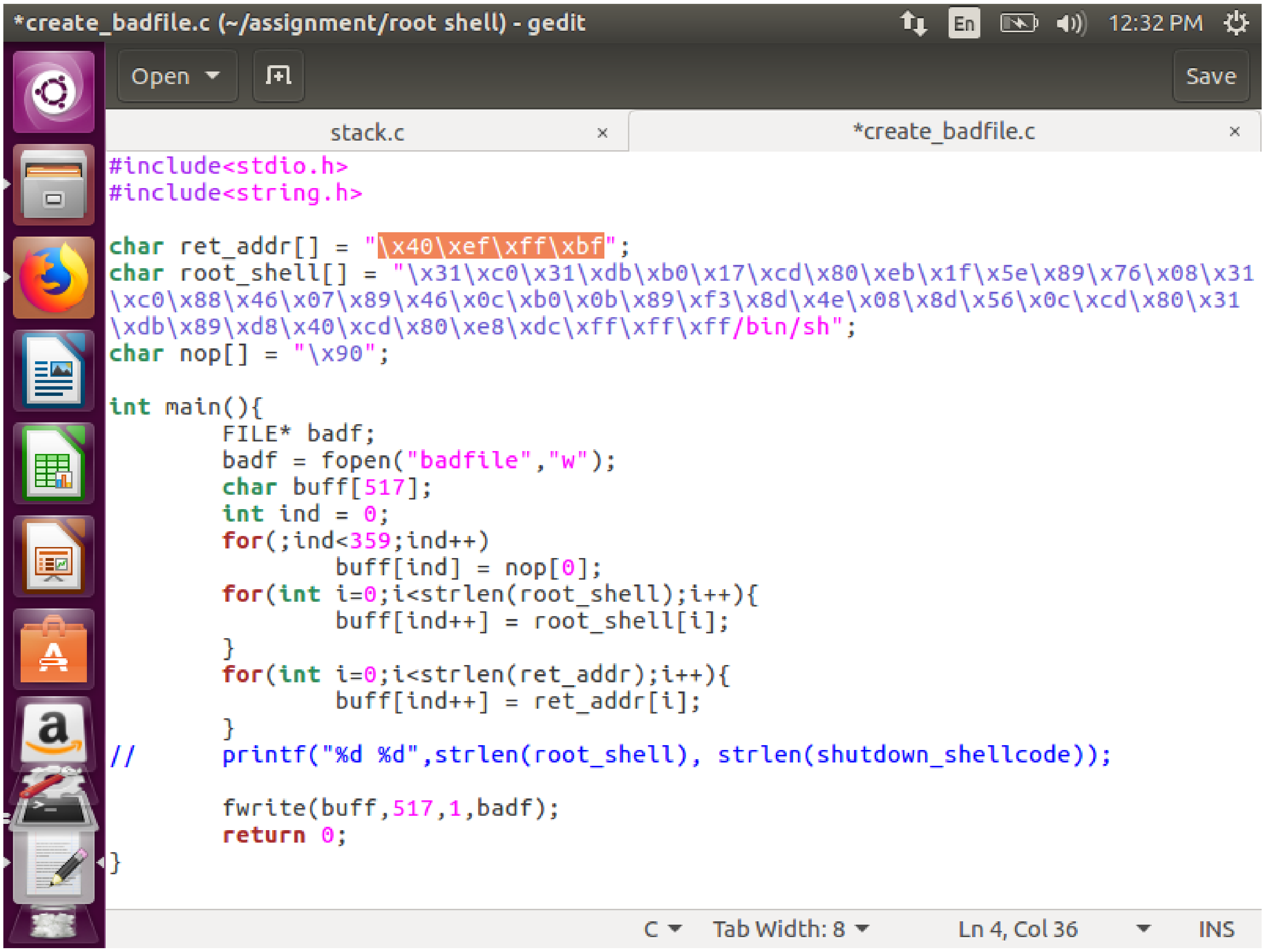
- Updating the Return Address

Now we find the desired return address where our malicious code resides. We do this using the ESP register. So, when our program hits a segmentation fault by returning to our dummy return address, at that point we check the contents of the ESP register and find the memory address from where our shellcode resides. And the shellcode is preceded by a trail of NOPs which implies that we could take any of the preceding memory addresses as the return address. Thus the code will return to this address, slides through the NOPs and executes our desired code..

As evident in the image below, the malicious code starts at address *0xbfffef50*. This same address is then updated as the return address in the badfile.



Terminal
En
12:31 PM
danial@danial-VirtualBox: ~/assignment/root shell
0xbfffef50: 0x90909090 0x90909090 0x90909090 0xc0319090
0xbfffef60: 0x17b0db31 0x1feb80cd 0x0876895e 0x4688c031
0xbfffef70: 0x0c468907 0xf3890bb0 0x8d084e8d 0x80cd0c56
0xbfffef80: 0xd889db31 0xe880cd40 0xffffffdc 0x6e69622f
0xbfffef90: 0x5068732f 0xbfbfefee 0x000000e0 0x00000000
0xbfffefa0: 0xb7fff000 0xb7fff918 0xbffff000 0x0804828b
0xbfffefb0: 0x00000000 0xbffff094 0xb7fbb000 0x0000ff17
0xbfffefc0: 0xffffffff 0x0000002f 0xb7e15dc8 0xb7fd51b0
0xbfffefd0: 0x0000c000 0xb7fbb000 0xb7fb9244 0xb7e210ec
0xbfffefe0: 0x00000001 0x00000000 0xb7e37a50 0x0804867b
0xbfffeff0: 0x00000001 0xbffff0f4 0xbffff0fc 0x0804b008
0xbffff000: 0xb7fbb3dc 0xbffff020 0x00000000 0xb7e21637
0xbffff010: 0xb7fbb000 0xb7fbb000 0x00000000 0xb7e21637
0xbffff020: 0x00000001 0xbffff0b4 0xbffff0bc 0x00000000
0xbffff030: 0x00000000 0x00000000 0xb7fbb000 0xb7fffc04
0xbffff040: 0xb7fff000 0x00000000 0xb7fbb000 0xb7fbb000
0xbffff050: 0x00000000 0xfe4e317c 0xc5859f6c 0x00000000
0xbffff060: 0x00000000 0x00000000 0x00000001 0x080483f0
0xbffff070: 0x00000000 0xb7ff0010 0xb7fea880 0xb7fff000
0xbffff080: 0x00000001 0x080483f0 0x00000000 0x08048411
0xbffff090: 0x08048527 0x00000001 0xbffff0b4 0x080485a0
0xbffff0a0: 0x08048600 0xb7fea880 0xbffff0ac 0xb7fff918
0xbffff0b0: 0x00000001 0xbffff288 0x00000000 0xbffff2b1
---Type <return> to continue, or q <return> to quit---
// printf("%d %d",strlen(root_shell), strlen(shutdown_shellcode));
fwrite(buff,517,1,badf);
return 0;
}
C
Tab Width: 8
Ln 21, Col 3
INS



- Final Step

Before executing our vulnerable C file *stack* , we need to need to give required permissions and ownership to our file using *chmod* and *chown* commands.

```
sudo chown root stack
sudo chmod 4755 stack
```

Before Executing , Randomizing is turned off using the following command -

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

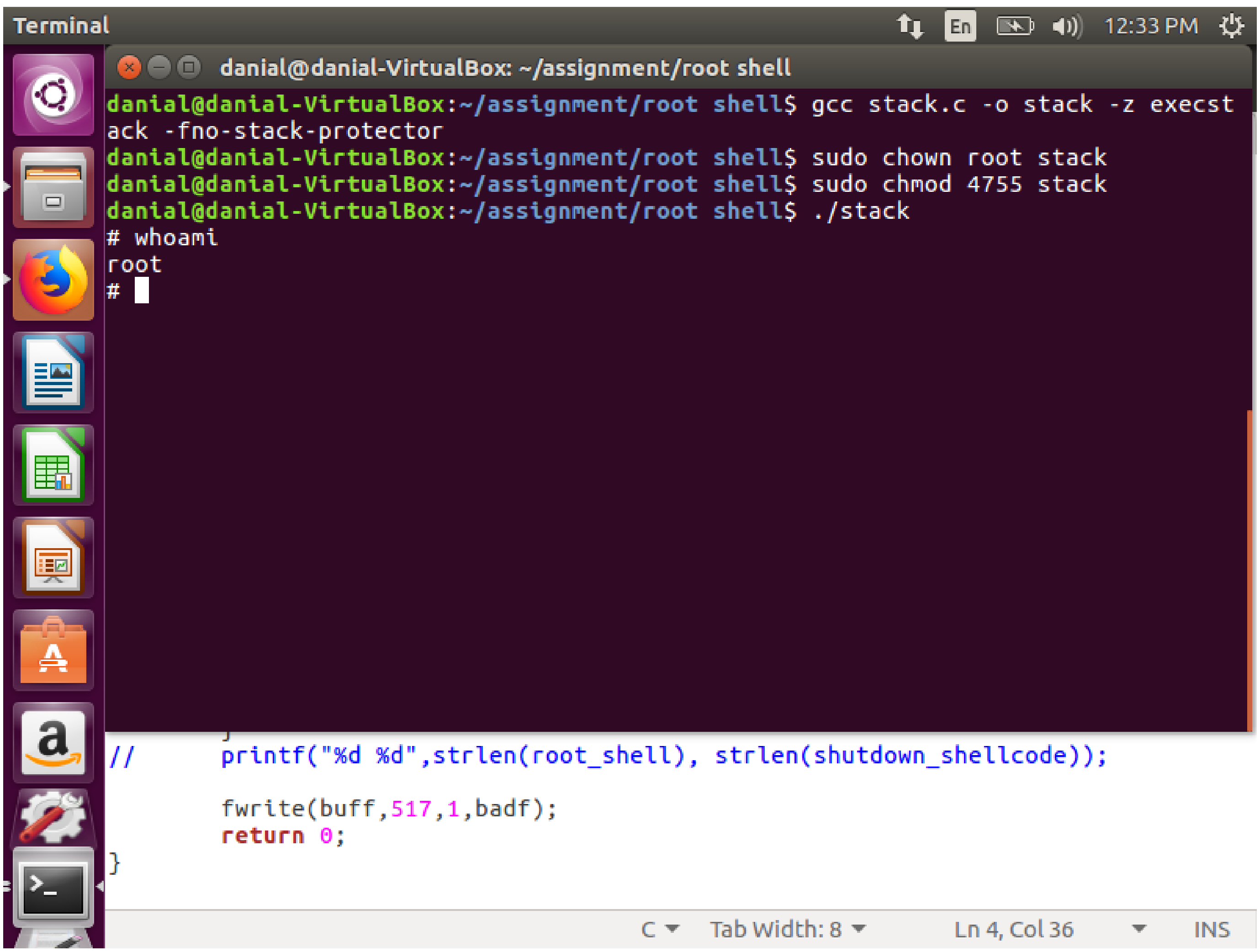
The program is then compiled by turning off the stack *protector.* We use the following command
to do that -

```
gcc -g stack.c -o stack -z execstack -fno-stack-protector
```

Execution of Different Exploits

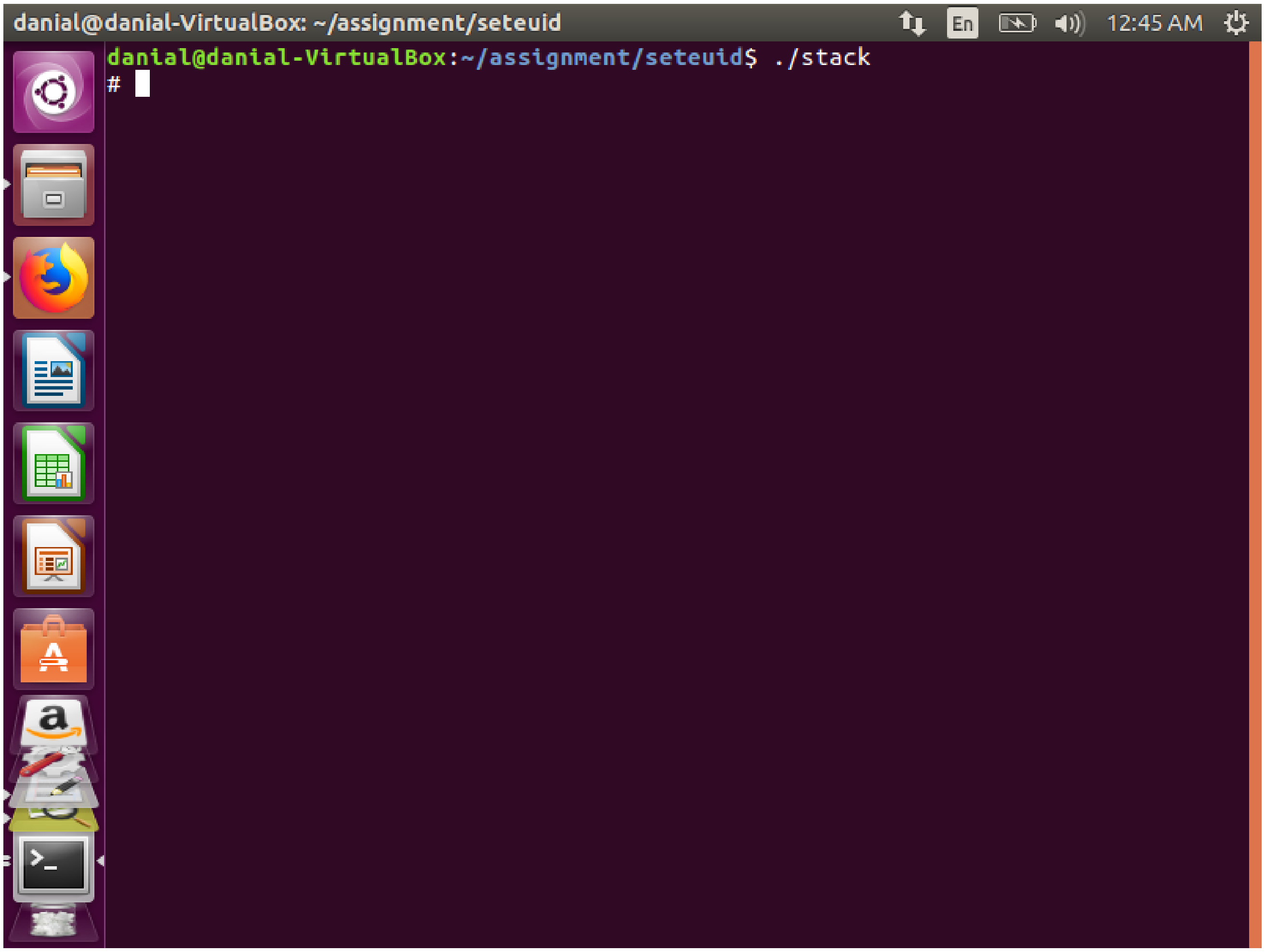
1. Launching Shell as Root

The malicious C file is executed with the shell code that launches the shell as *root*.



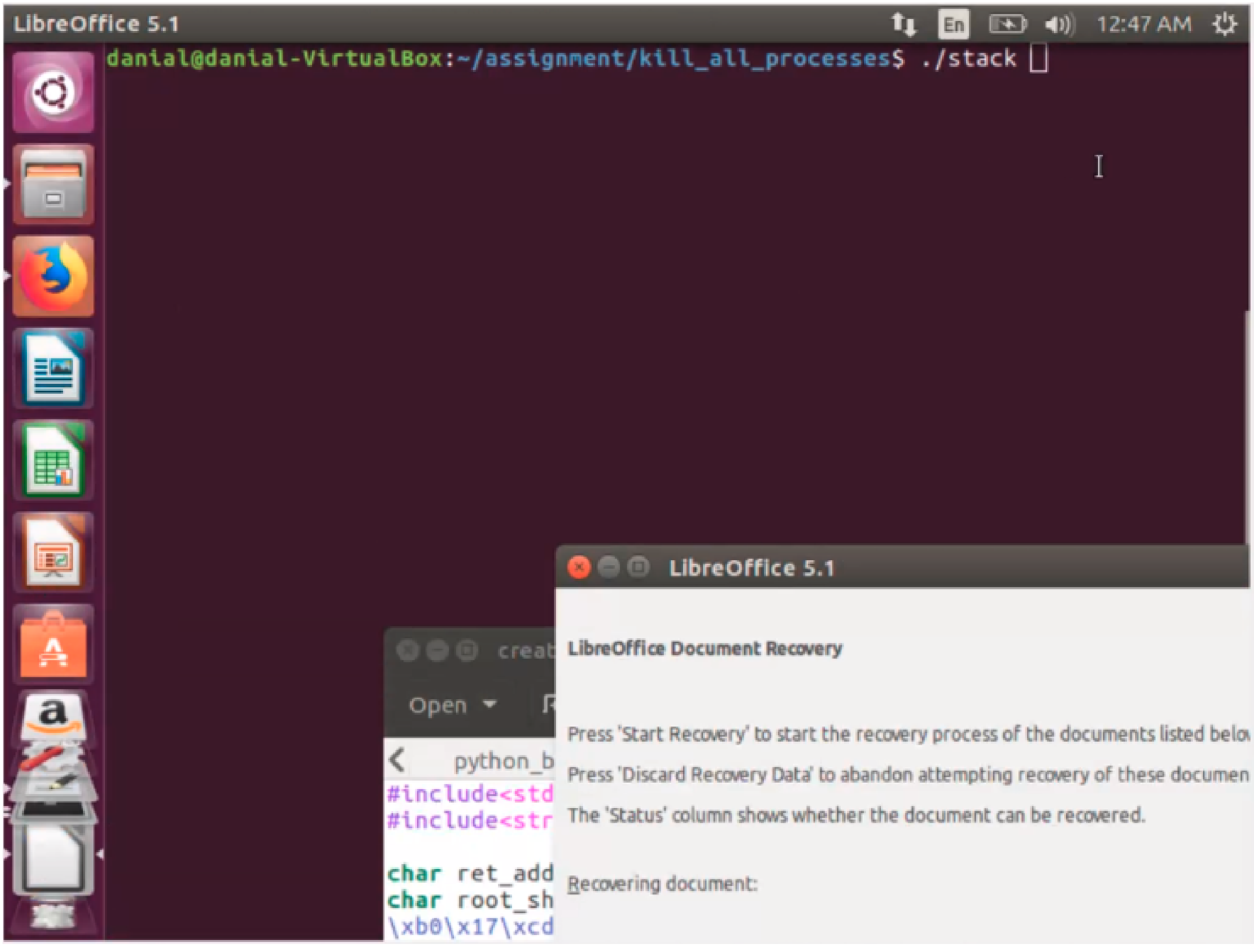
Root Shell accessed

2. Setting uid



3. Killing All Processes

The malicious C file is executed with the shell code that kills all the processes that are running. As seen in the image below, many processes are running like Libreoffice , text editor, etc.



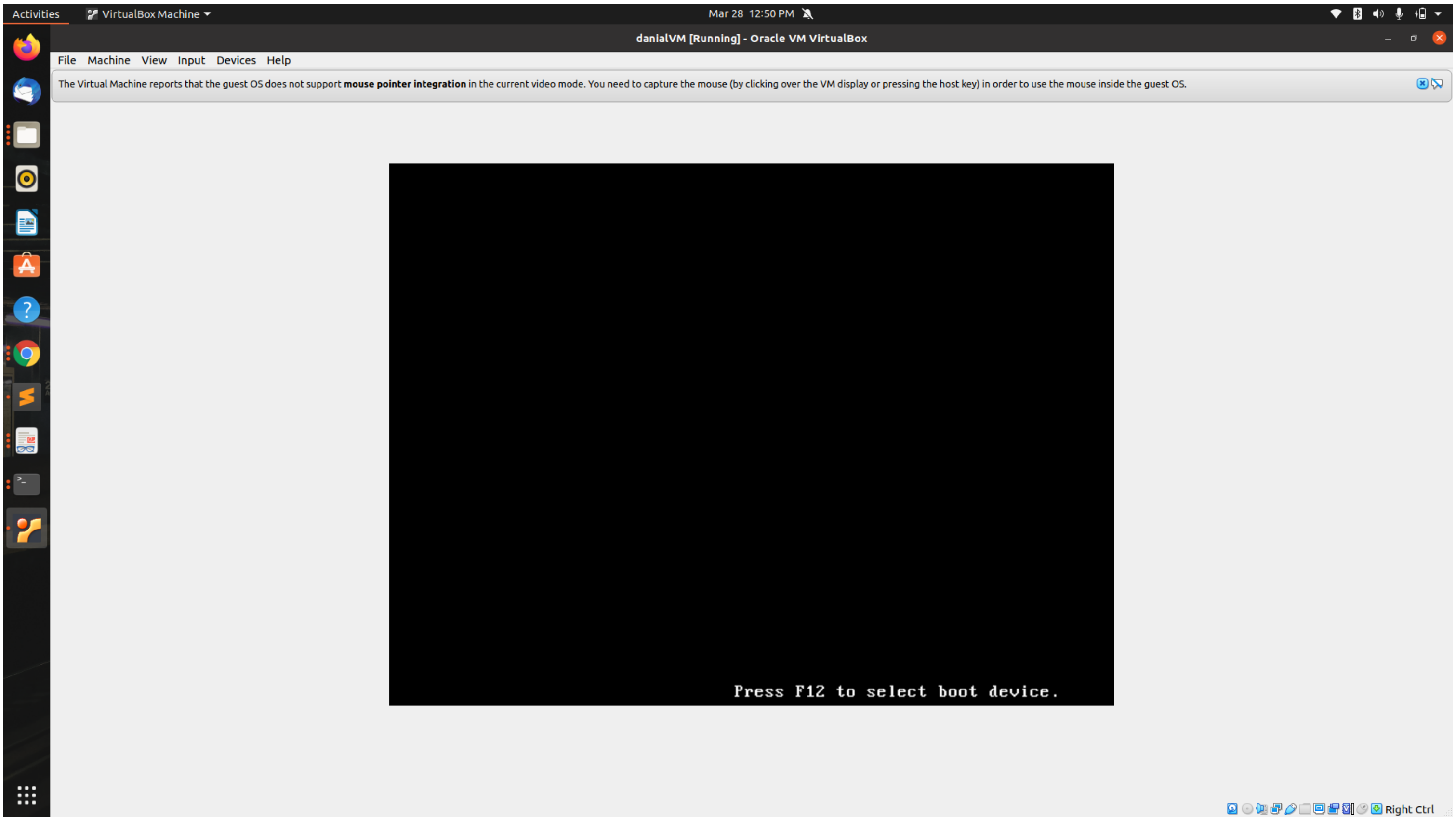
Before executing the executable

After execution of the malicious code, all the processes are killed as seen in the below image.

All Processes destroyed

4. Rebooting the System

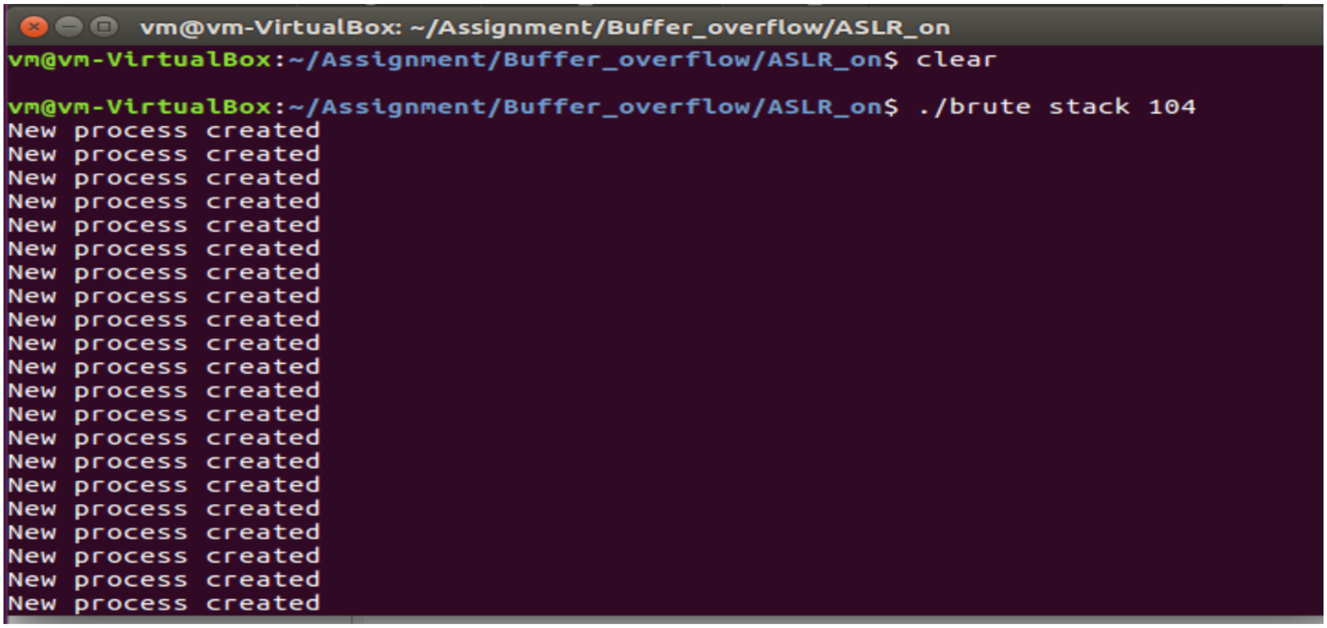
The malicious C file is executed with the shell code that reboots the system.



Device restarts after execution of vulnerable program

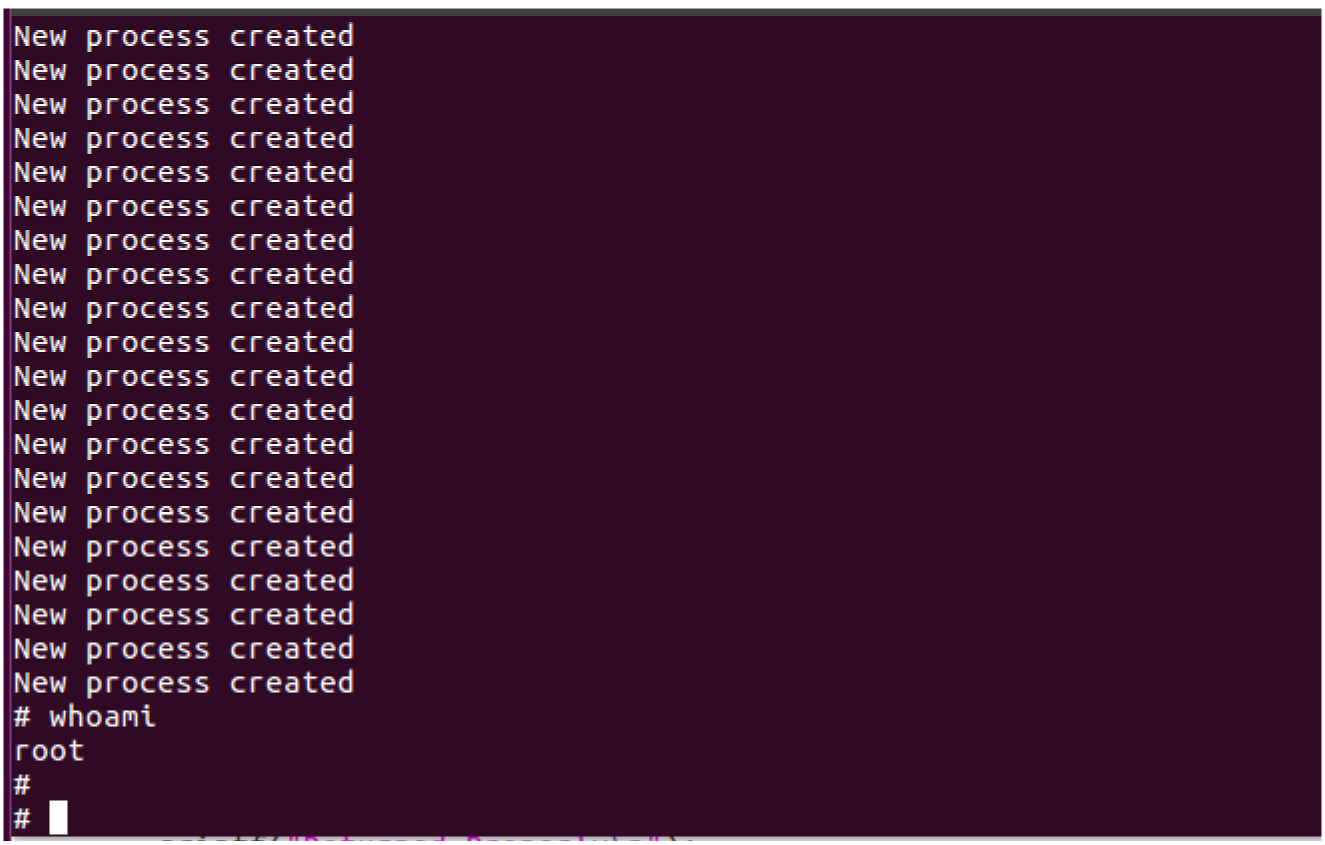
Execution of exploits with ASLR On

We followed the brute force strategy to bypass the ASLR feature. So, we created multiple processes responsible for performing hit and trial of return address to execute our desired shellcode.



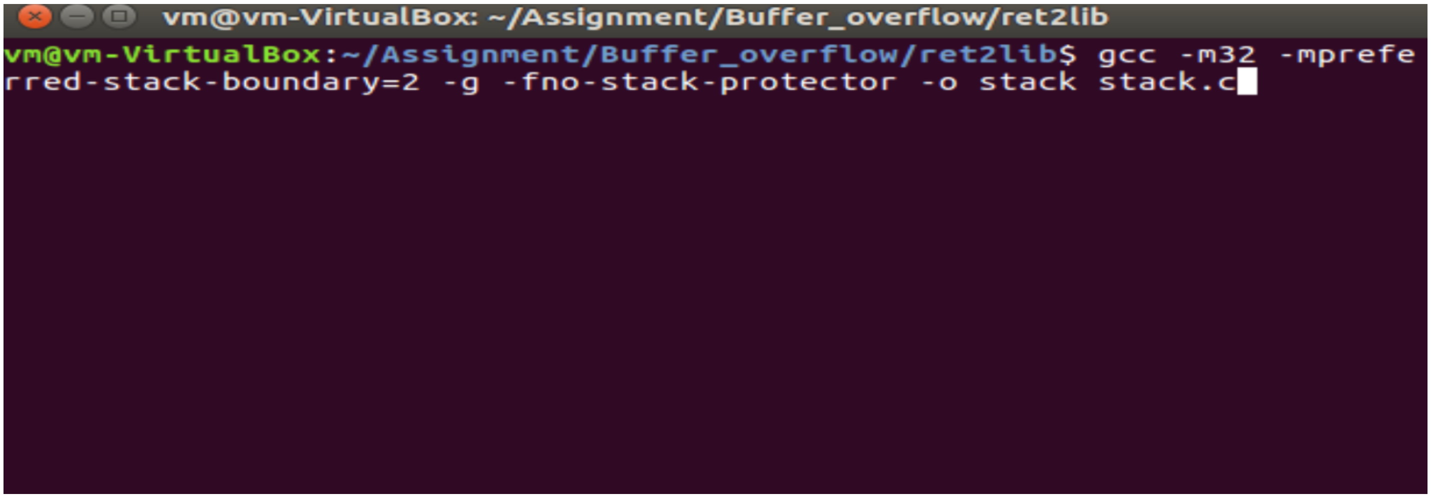
,

After some hits we made the successful attack.

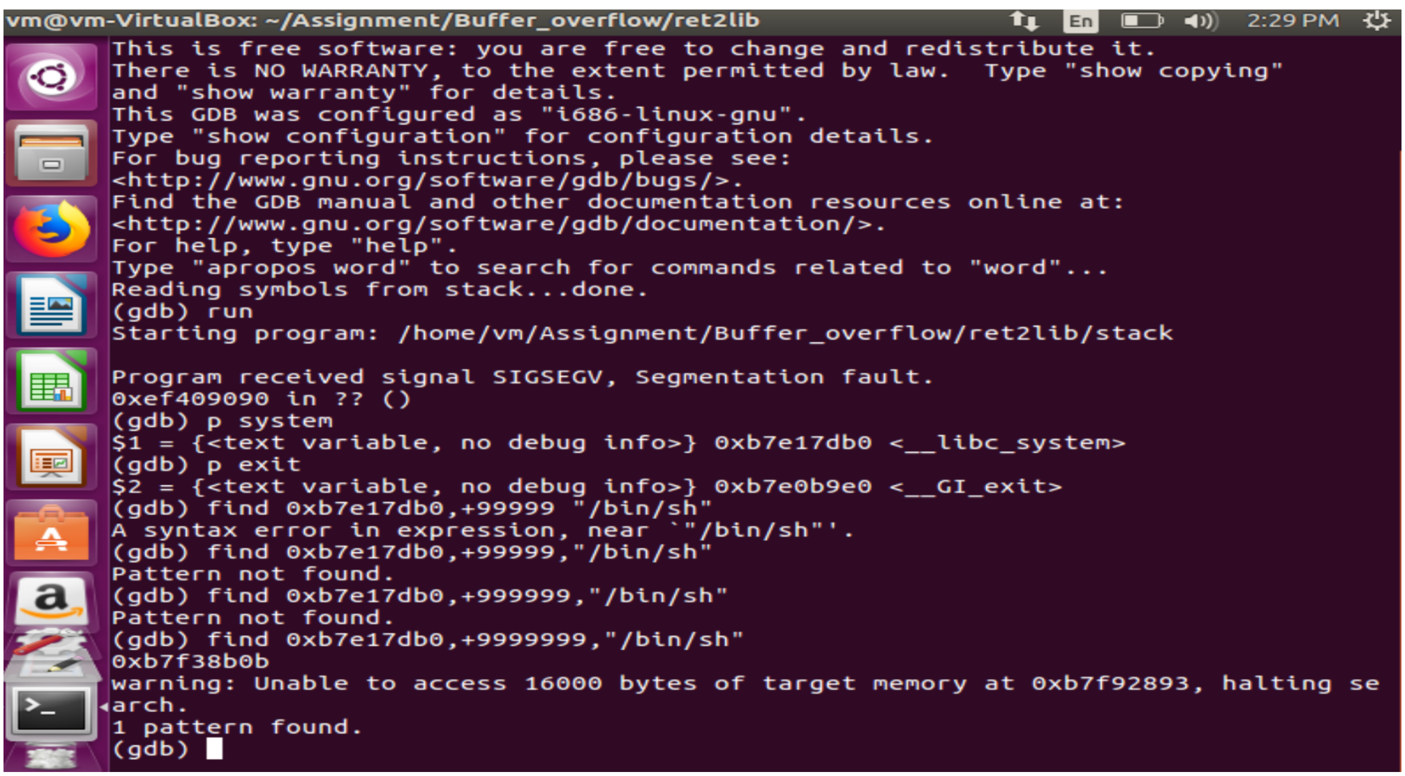


Ret2libc Attack

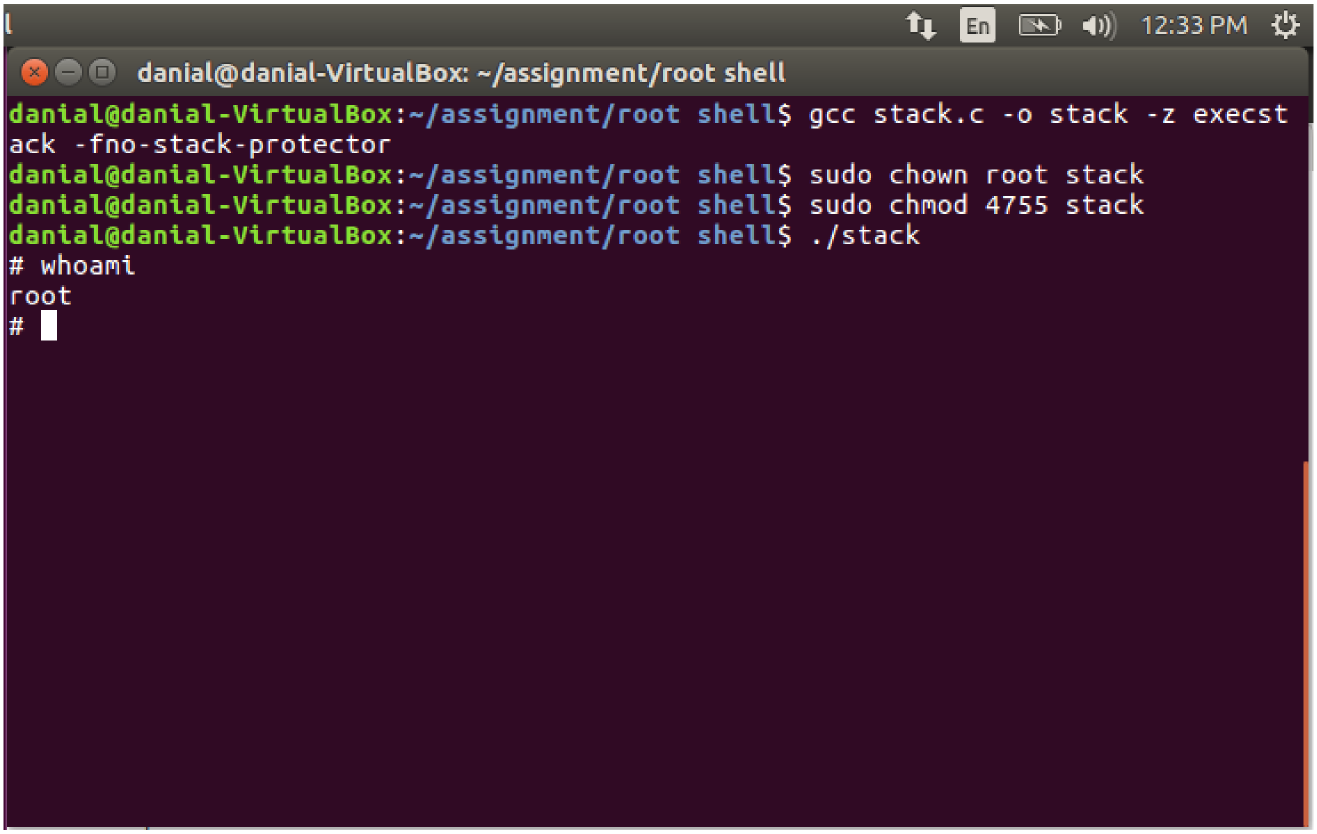
This attack is performed with a non-executable stack. So, here instead of putting our shellcode directly into the stack, we made the program execution to return to the address of libc.



Compilation of vulnerable program with non-executable stack



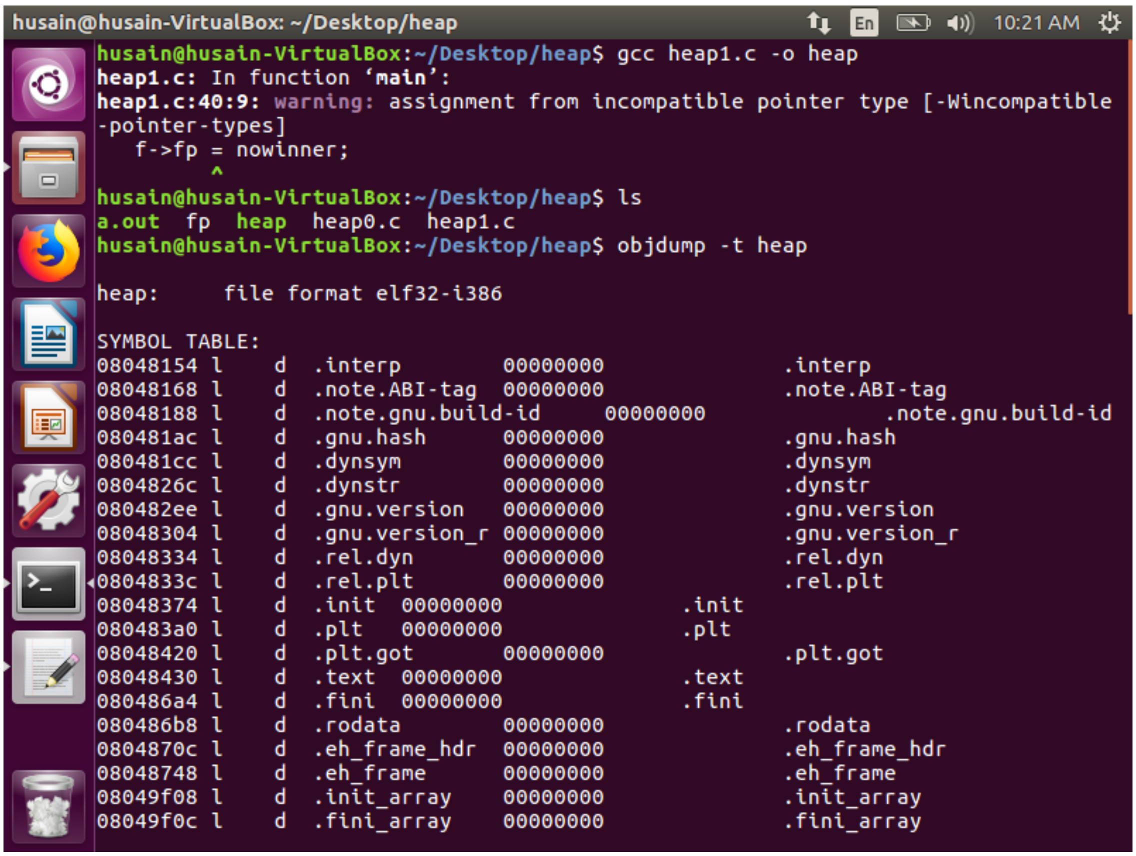
Finding the address of libc where we wish to return



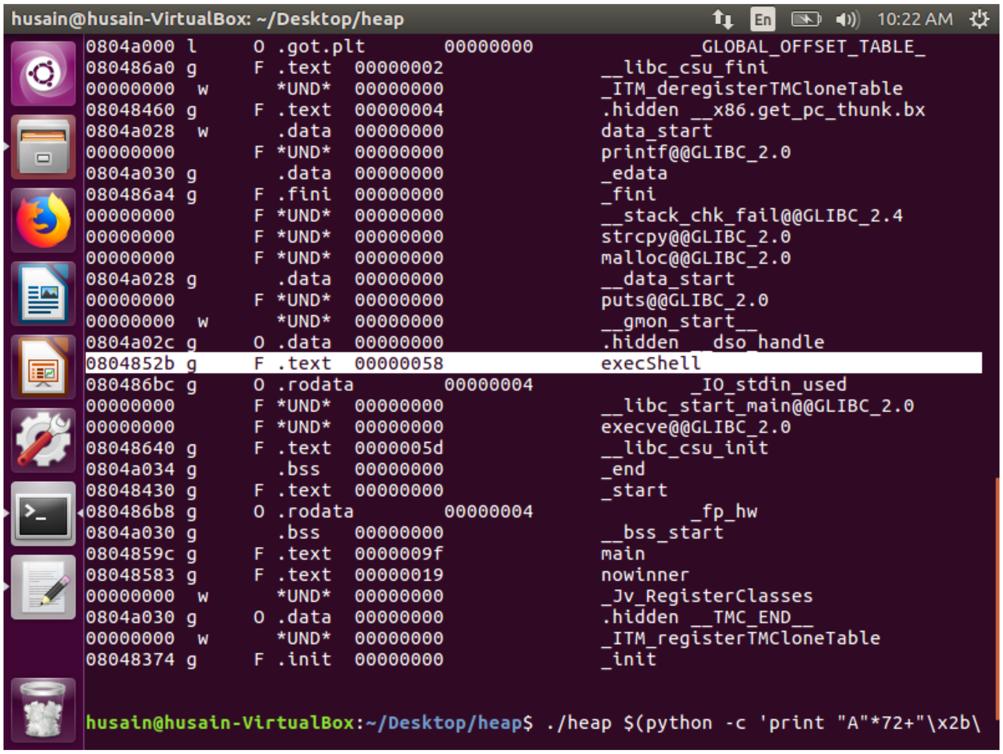
Execution of Shell

HEAP OVERFLOW

_____To execute the heap overflow attack , we assign two heaps (using malloc).First heap is for data pointer d and the next heap is for fp pointer f. Next we use objdump to get addresses of all functions in the program.



We get the address where fp has Failed function address .We see the data in memory address of fp and see where the Failed function pointer is stored. After that we calculate the offset required to overwrite Failed function with the execShell function .We see that Failed is at offset of 72 characters from start of d . Thus we write 72 dummy characters and at next address we fill with the address of execShell.



When we run the program we see that the heap exploit has worked and a new shell prompt is opened.