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

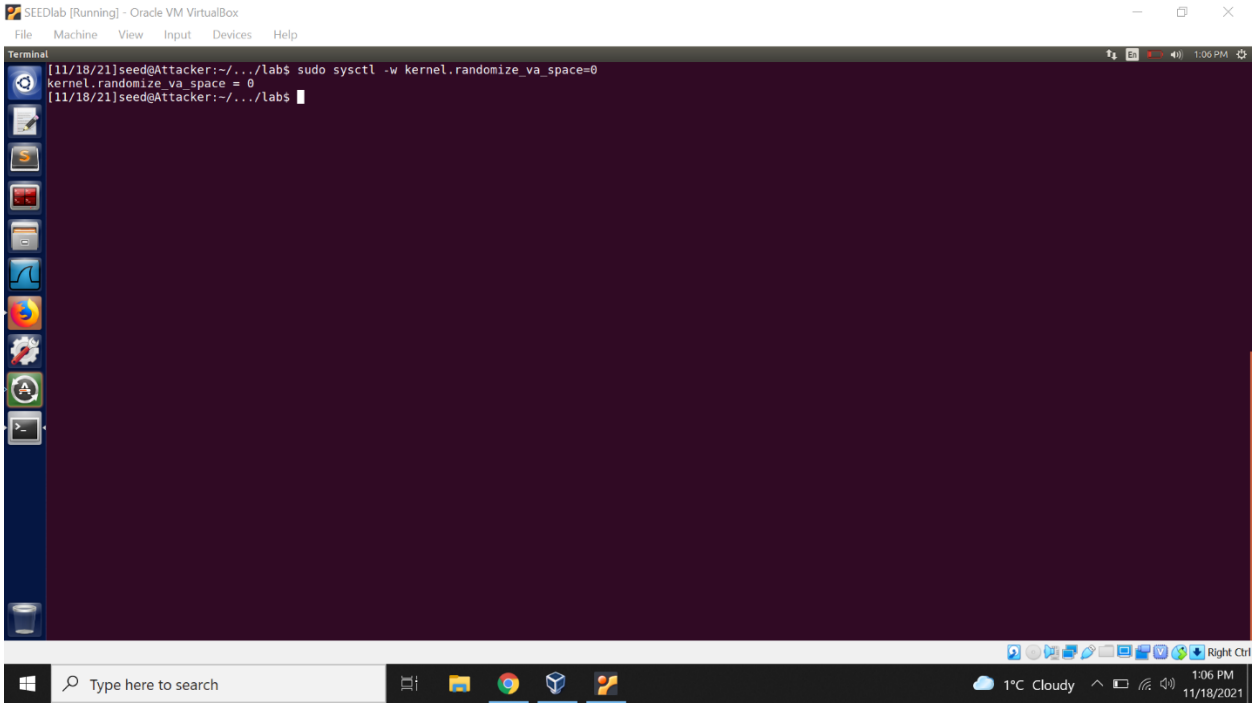
Date Submitted:

November 19, 2021

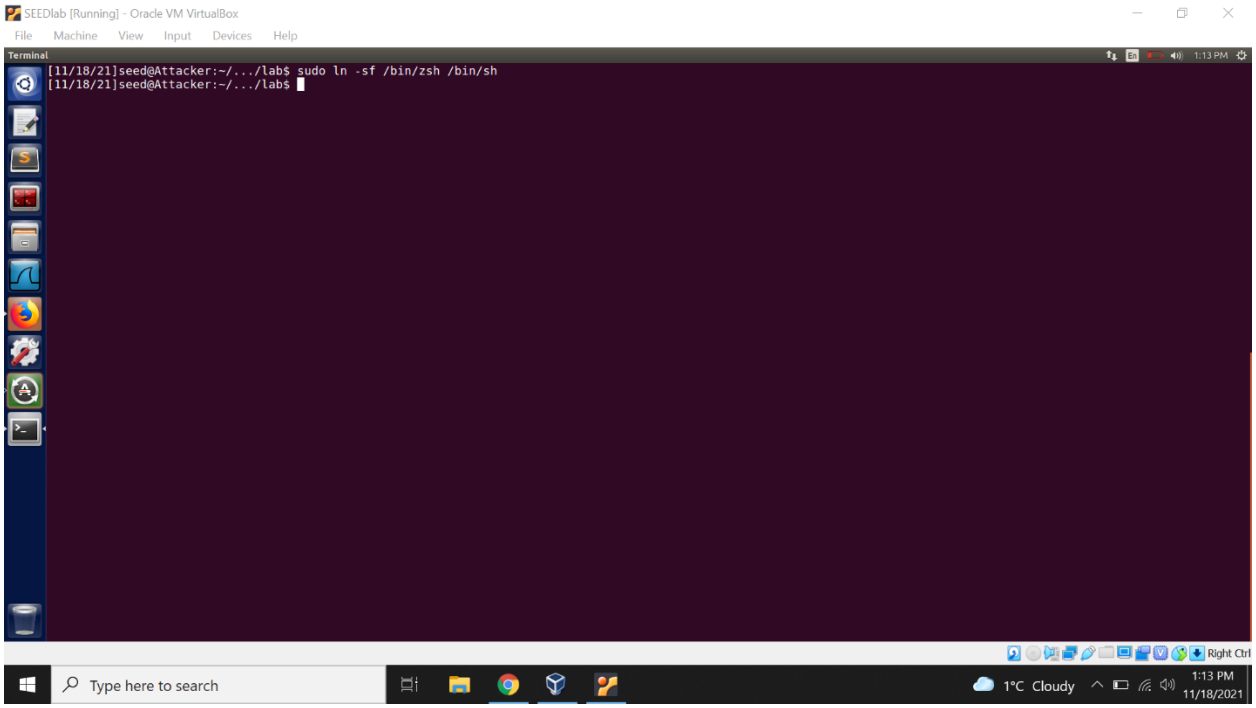
Lab: Buffer Overflow Vulnerability Lab

Turning off the Countermeasures

Turn off Address Space Randomization

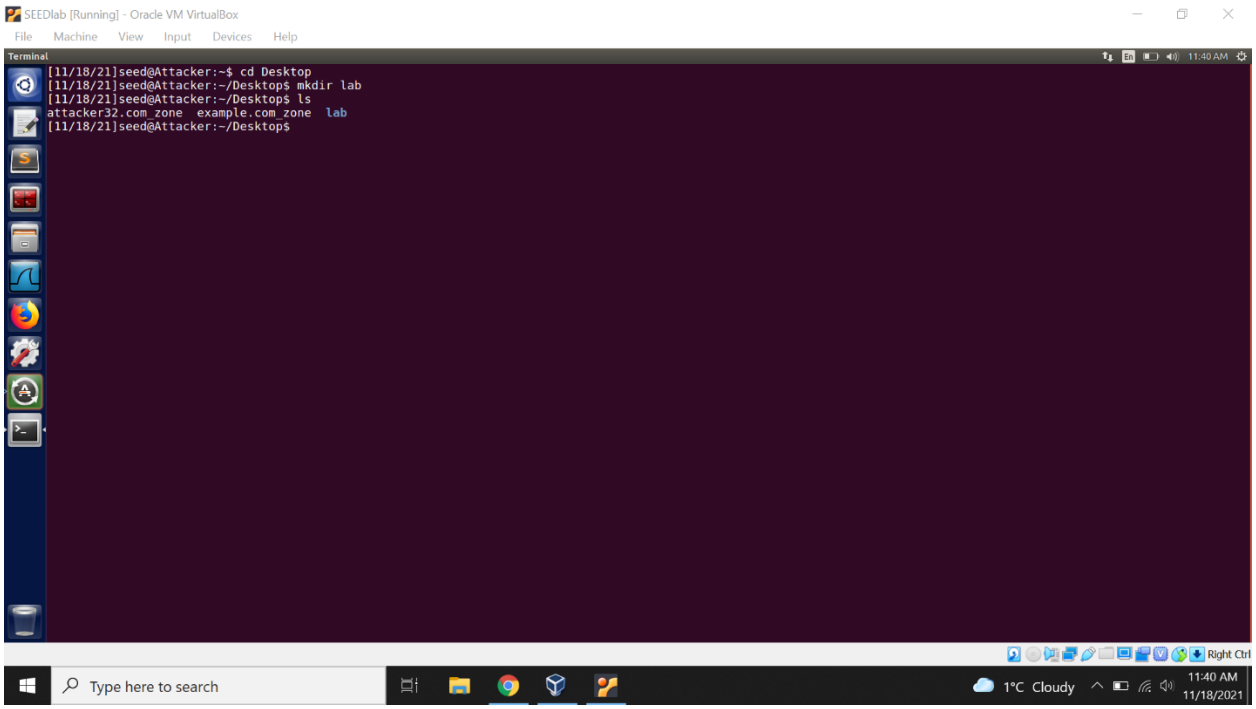


Configuring /bin/sh (Ubuntu 16.04 VM only)

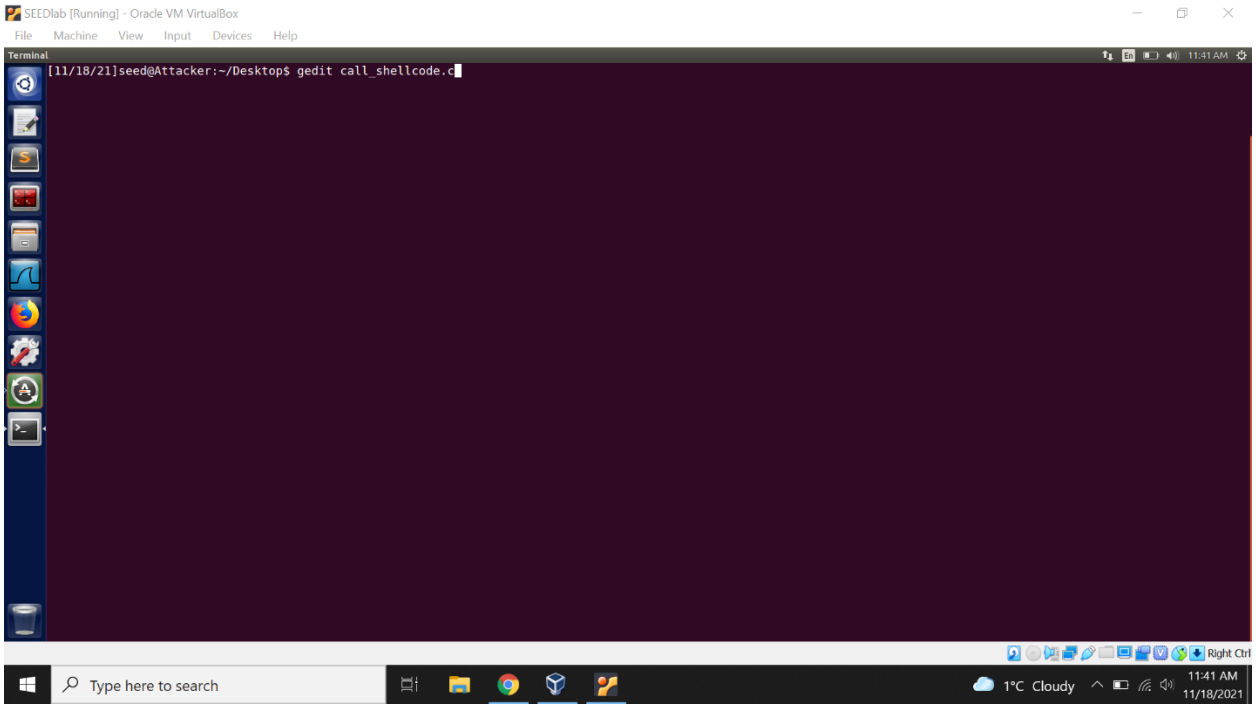


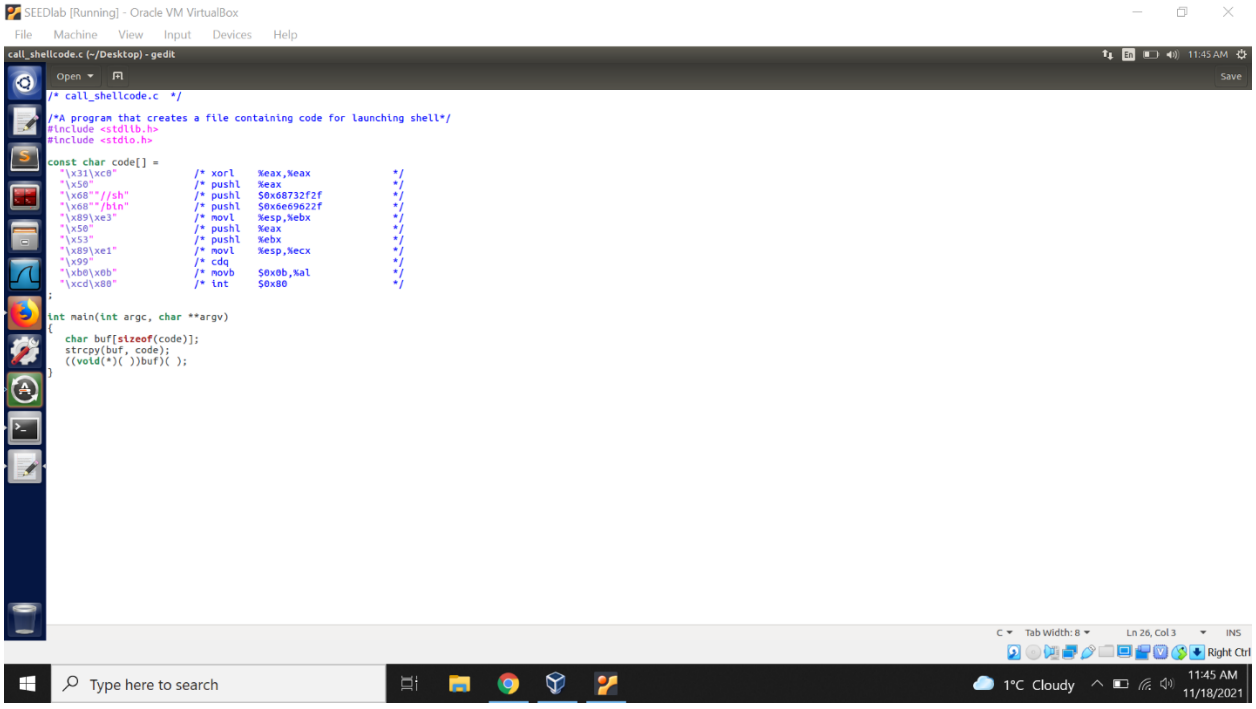
Task 1: Running Shellcode

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

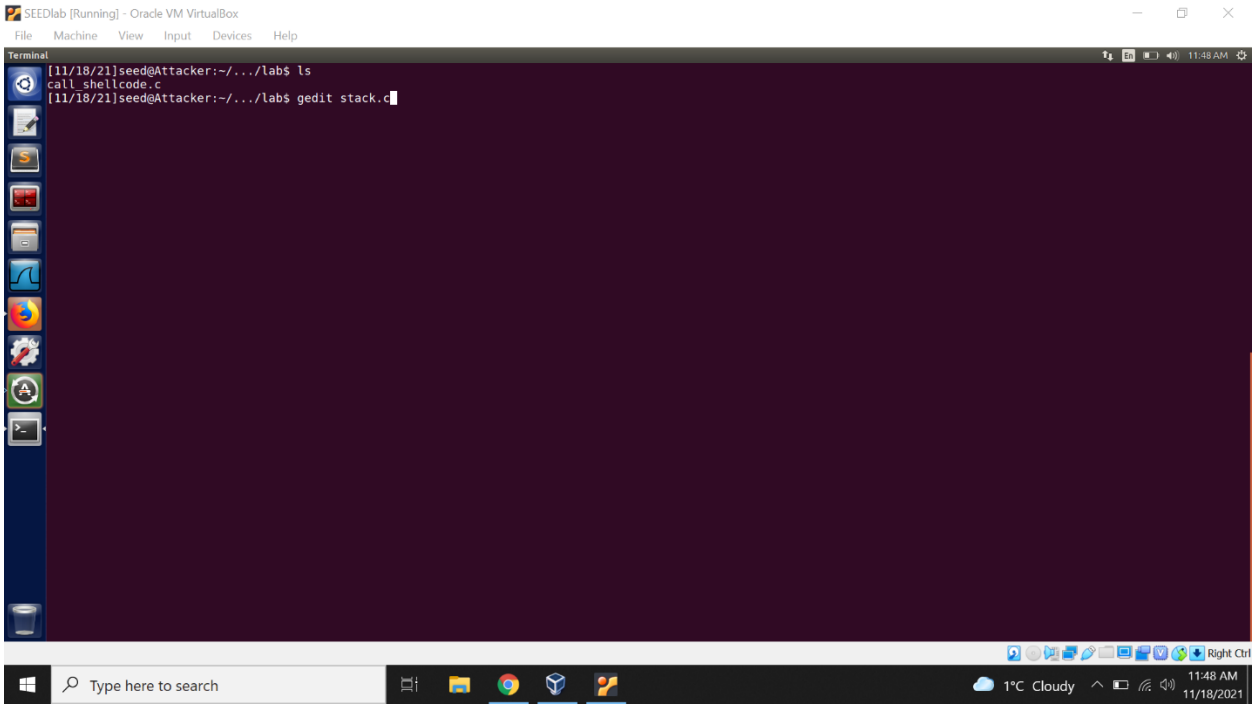


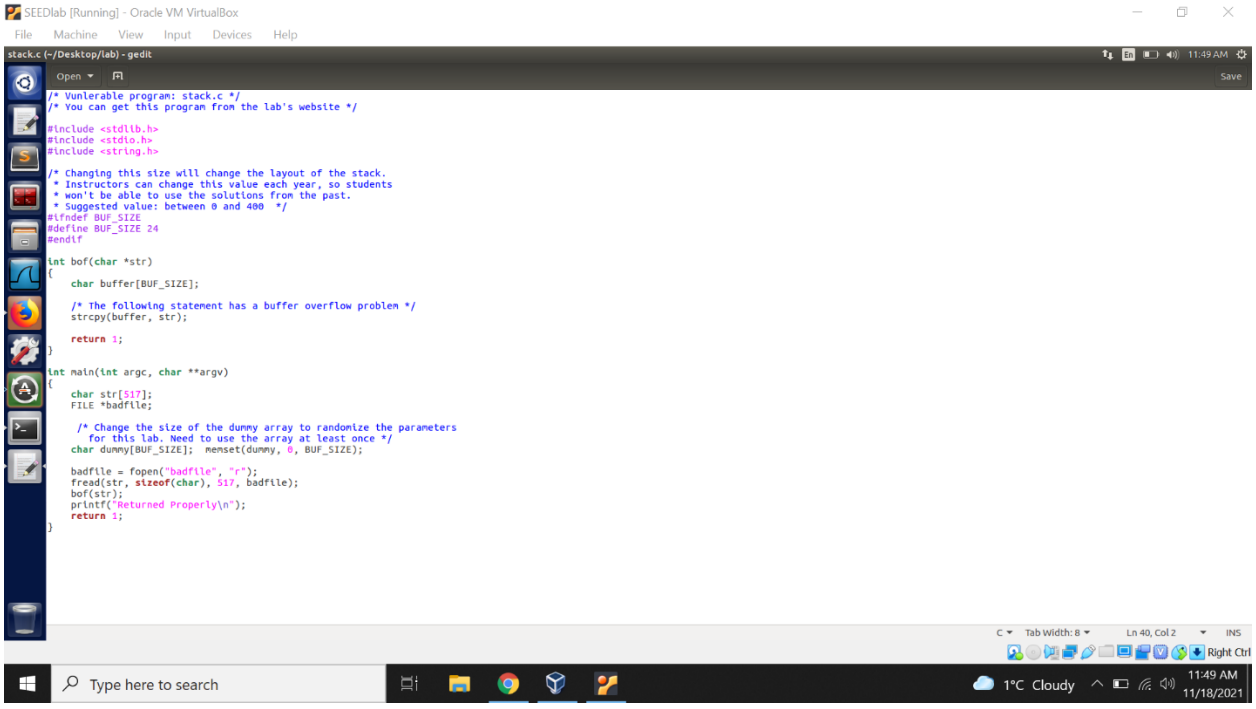
The call_shellcode.c file.



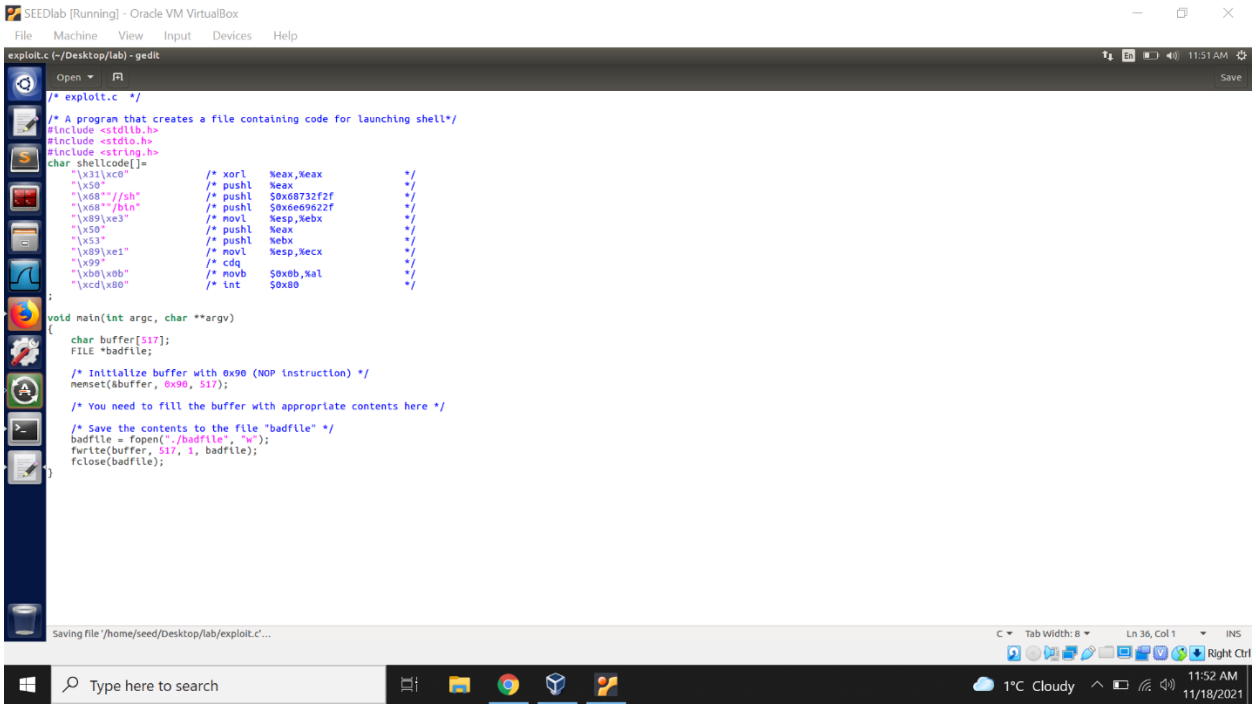


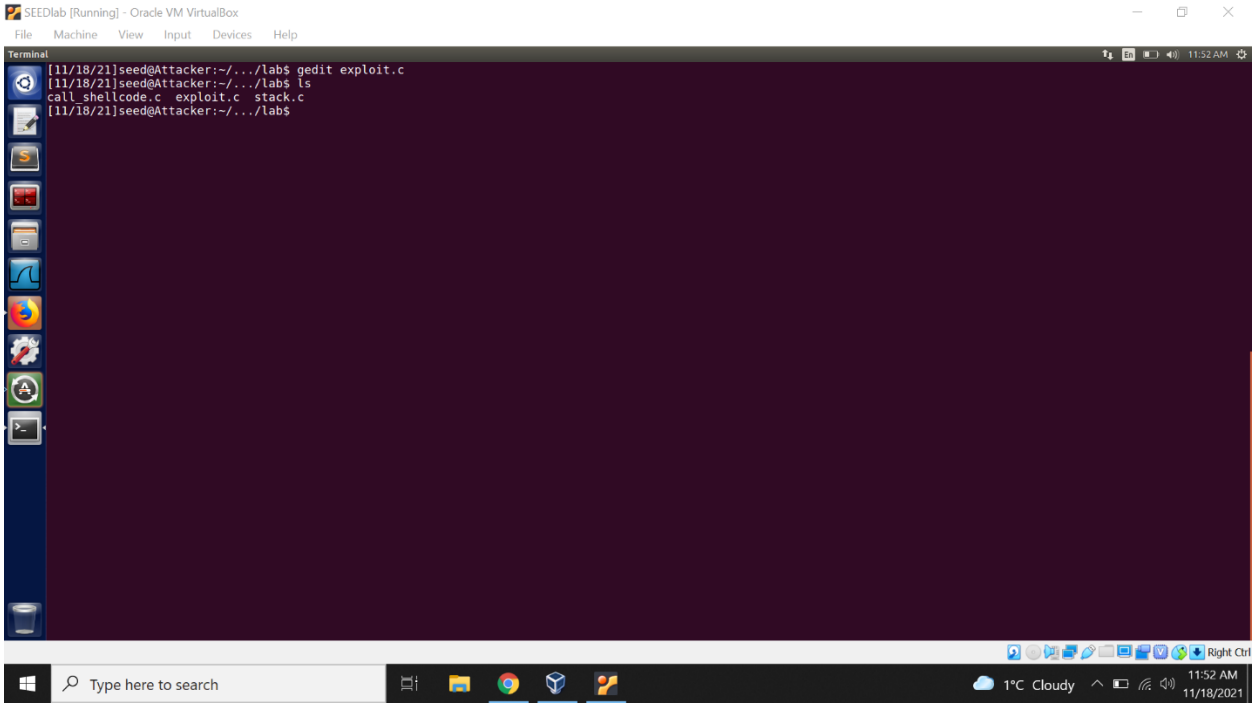
The stack.c file.



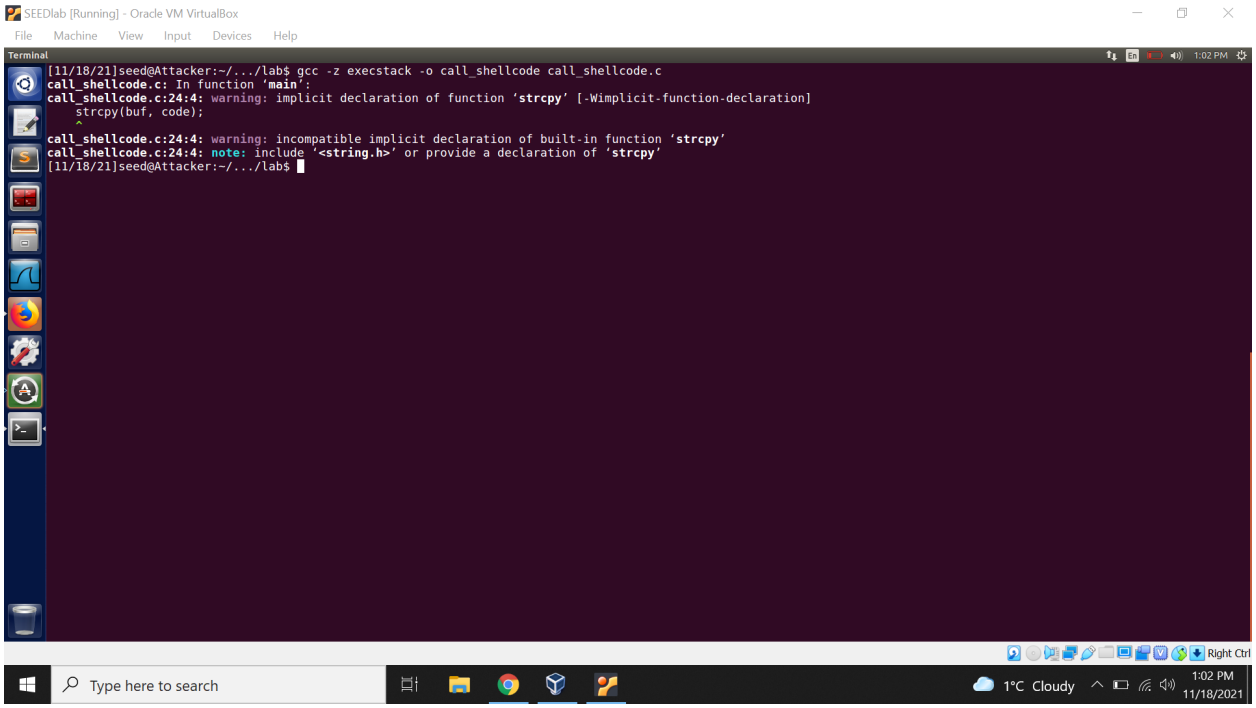


The exploit.c file.

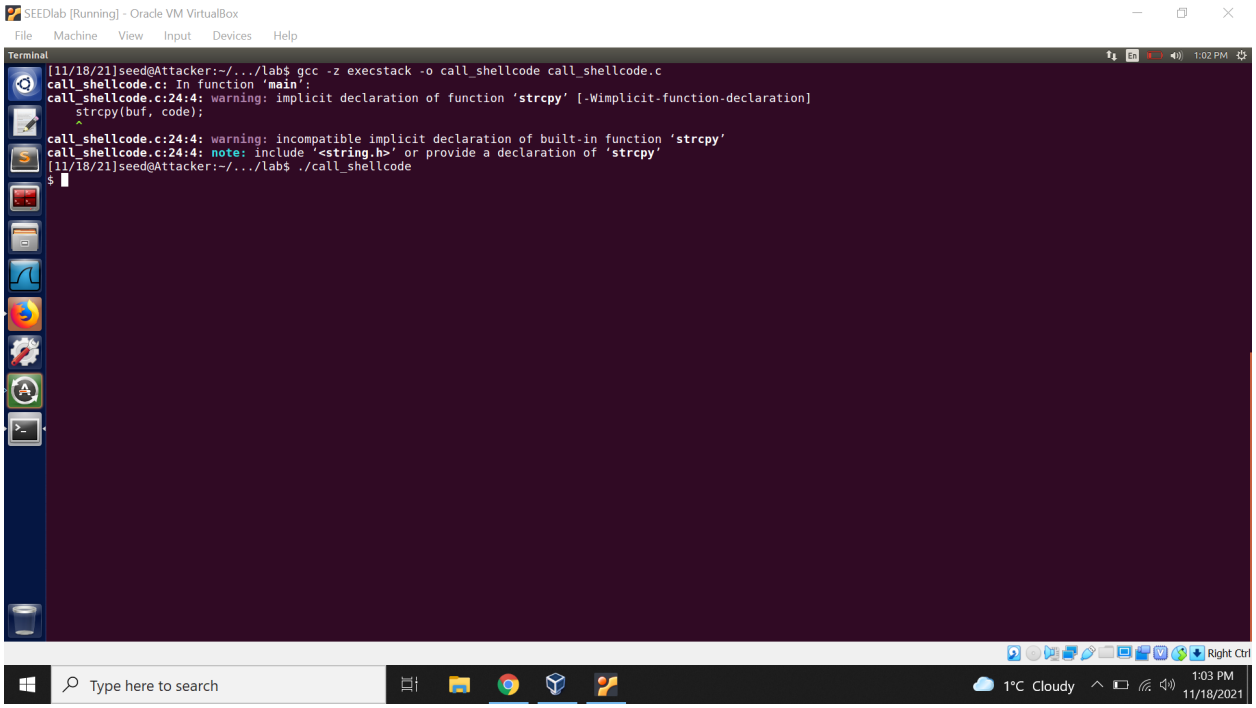




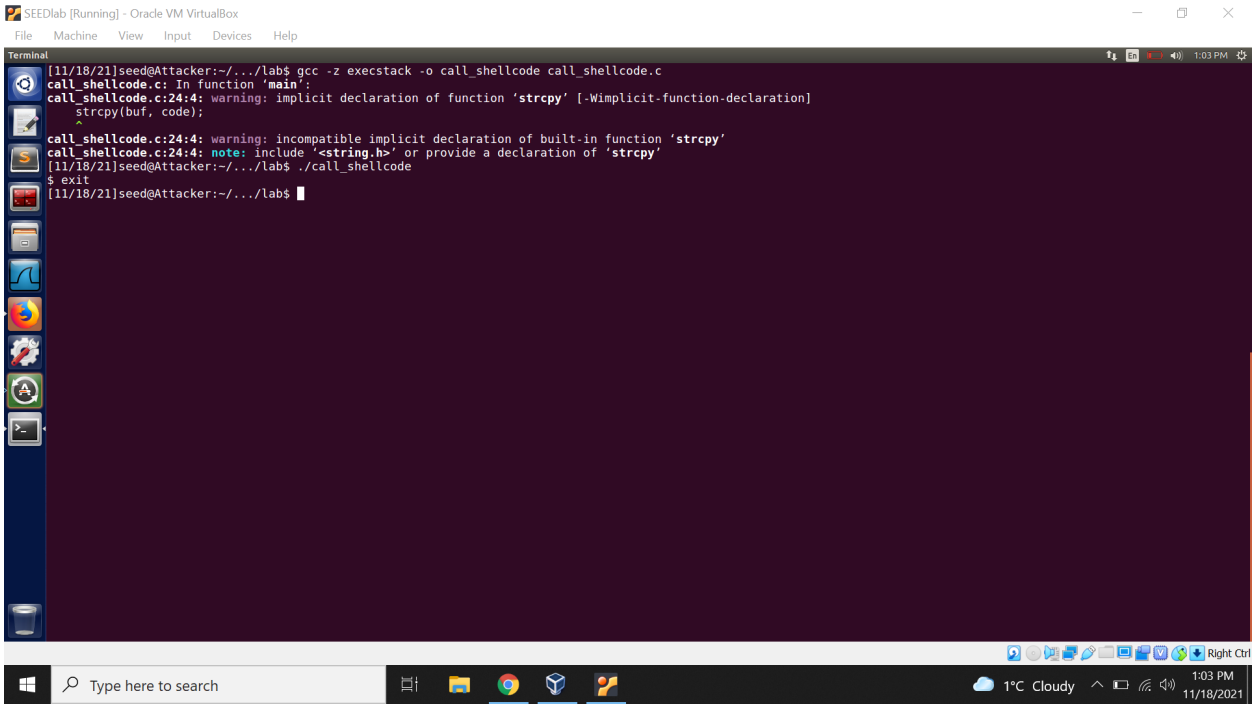
We, firstly, compile the call_shellcode.c file to check whether our shellcode is called.



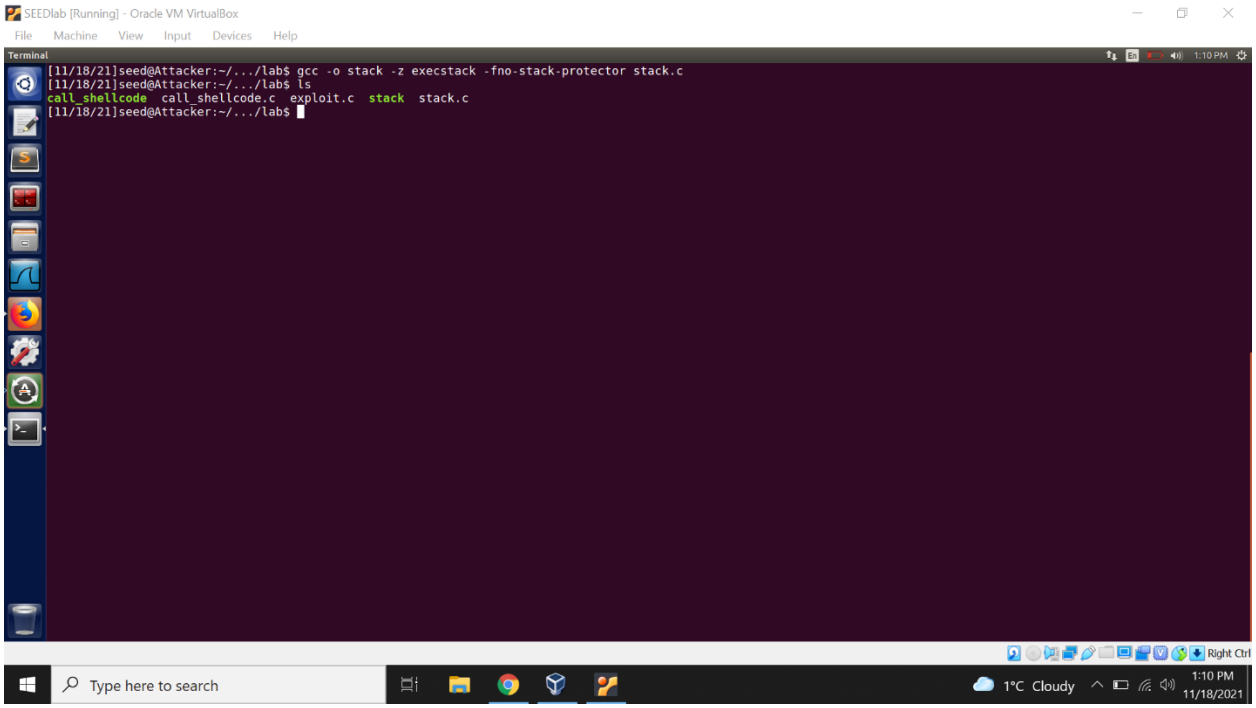
Now, we execute the call.shellcode.c code.



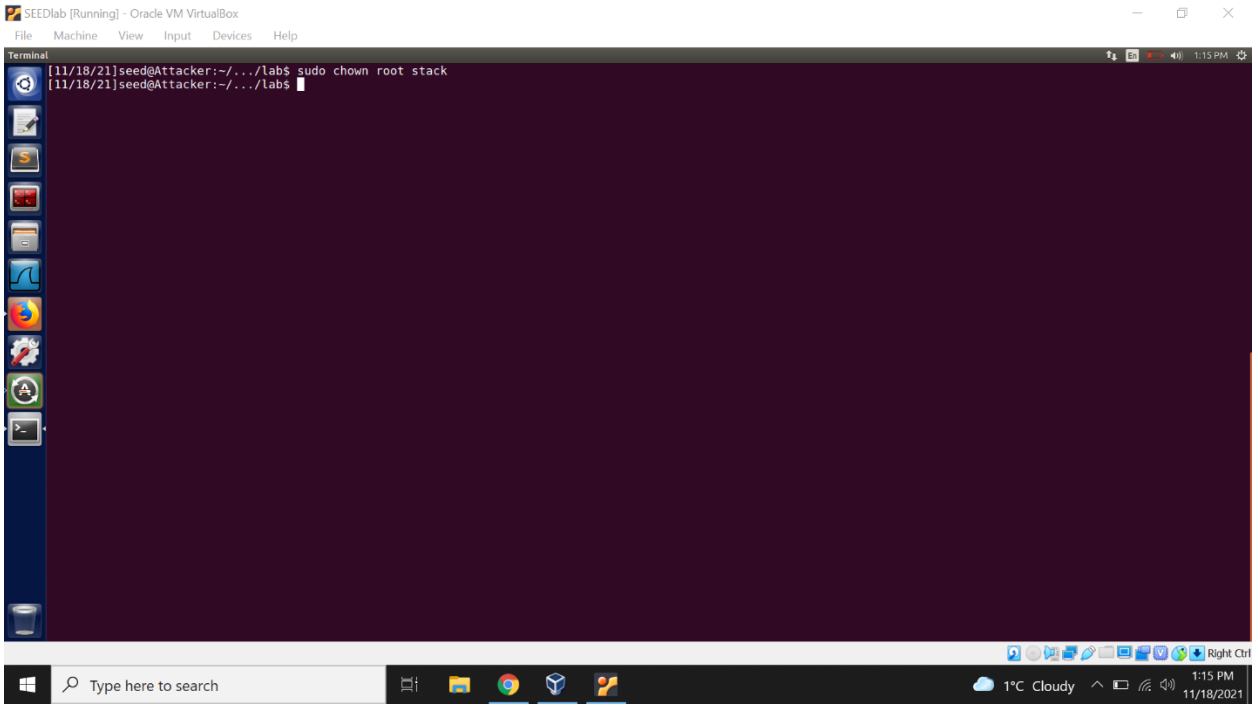
Observation: The shellcode is called and we are able to enter into the shell.



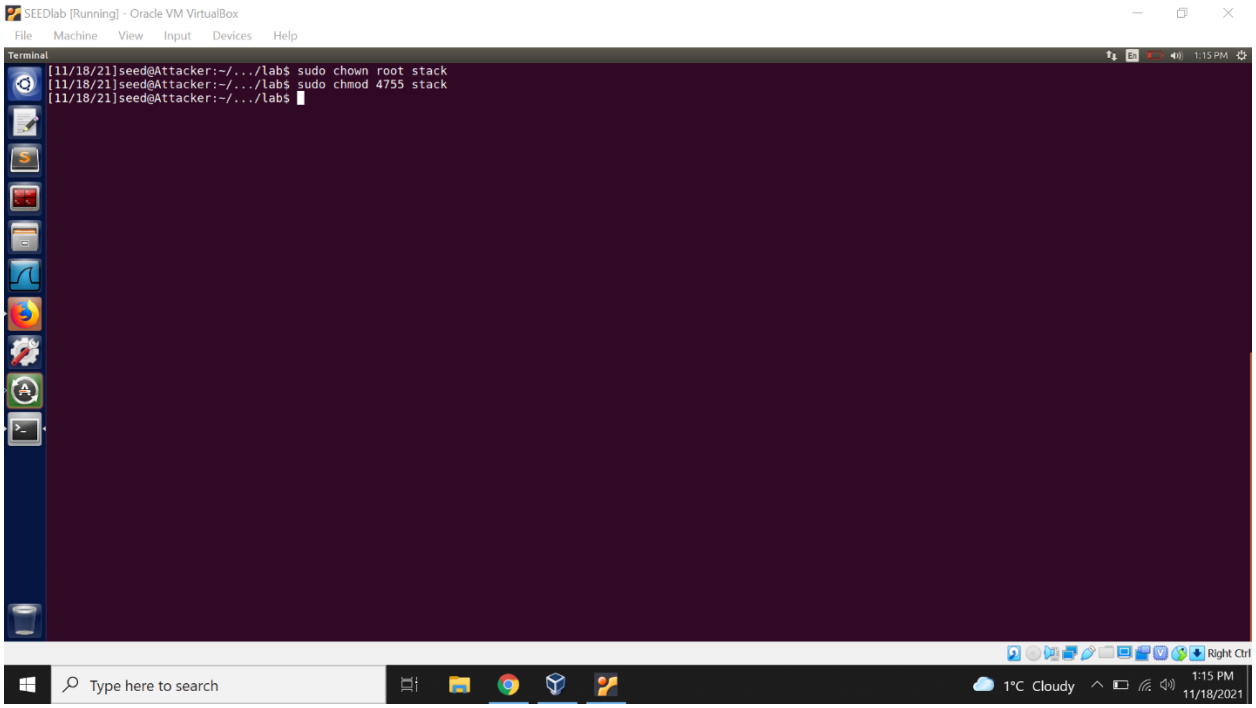
Compile the stack.c file by disabling the StackGuard Protection Scheme and using an executable stack.



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

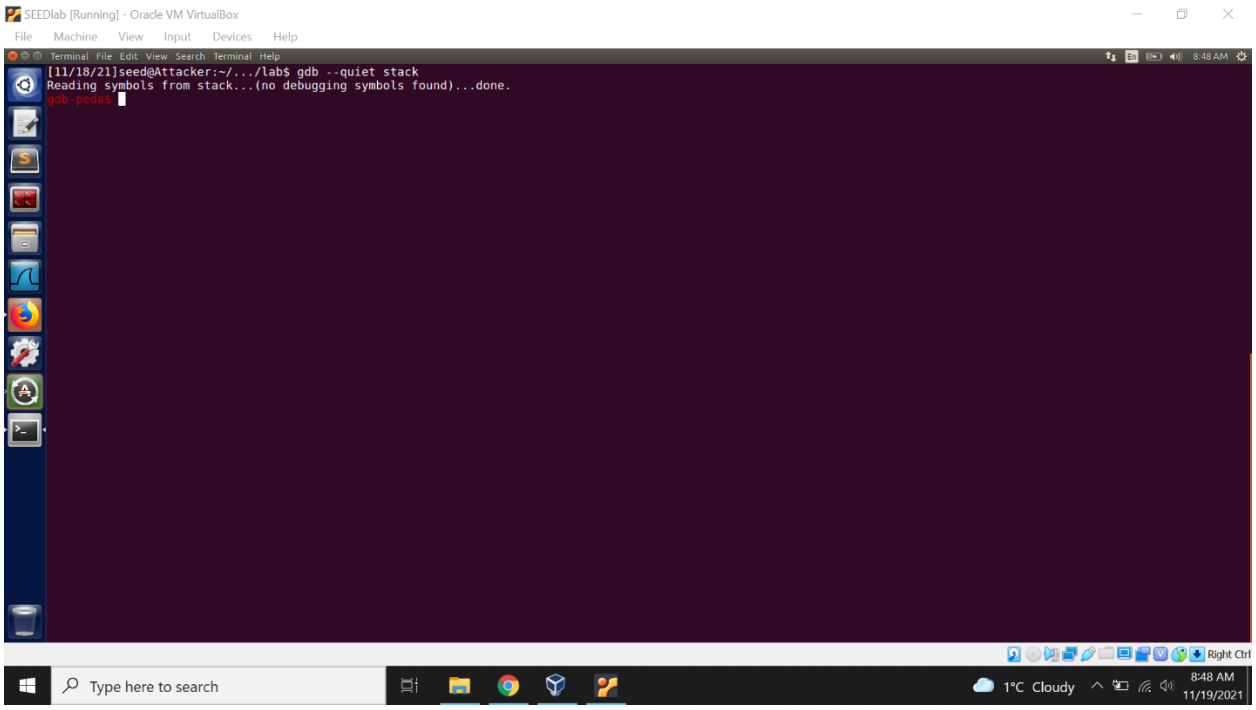


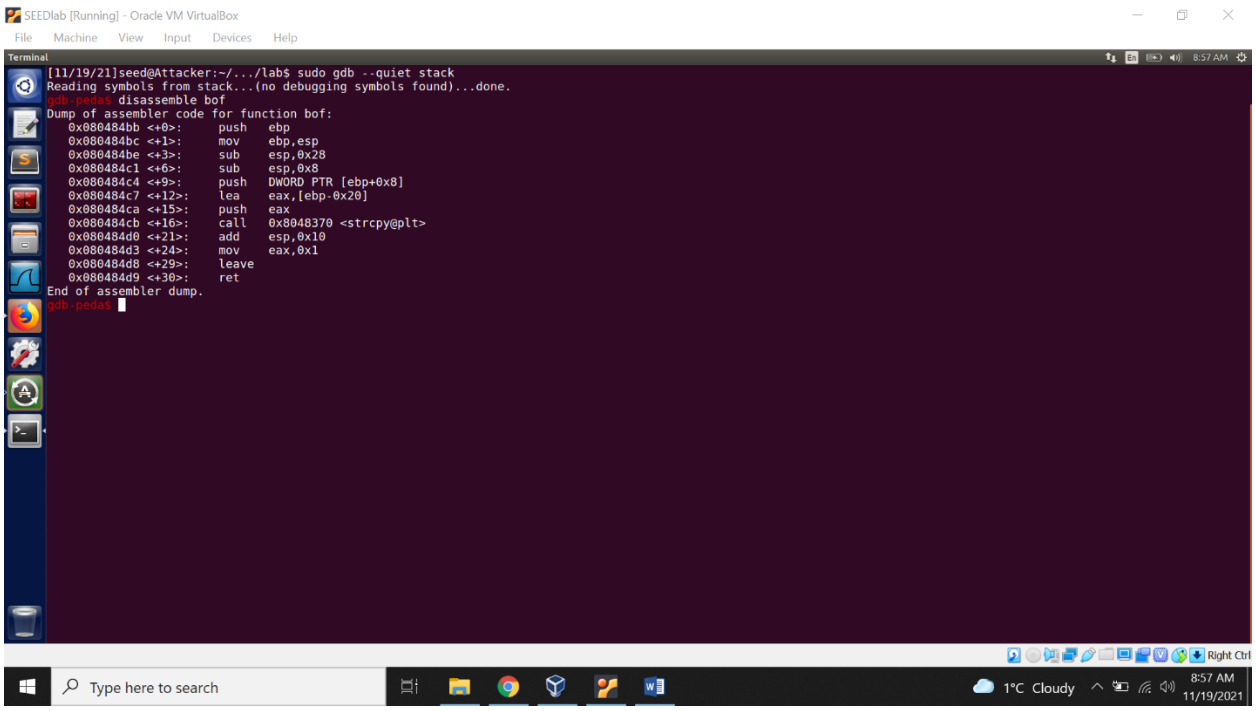
We change the permission to 4755 to enable the Set-UID bit.



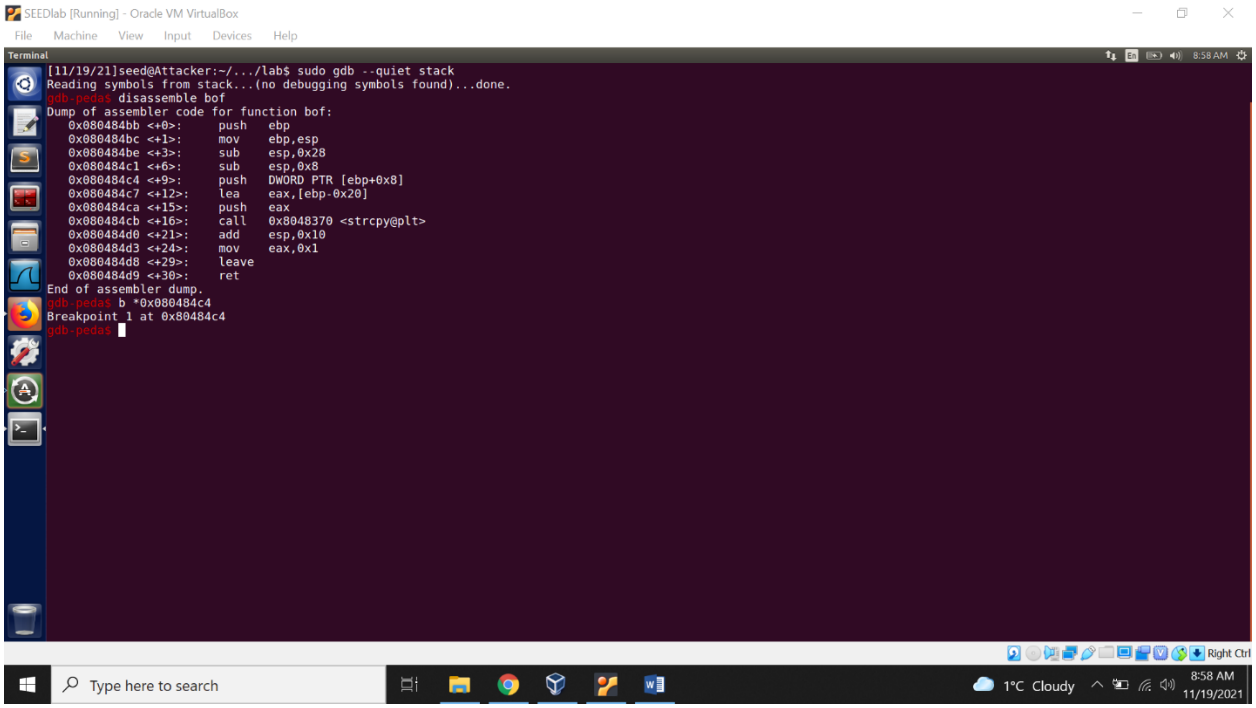
Task 2: Exploiting the Vulnerability

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

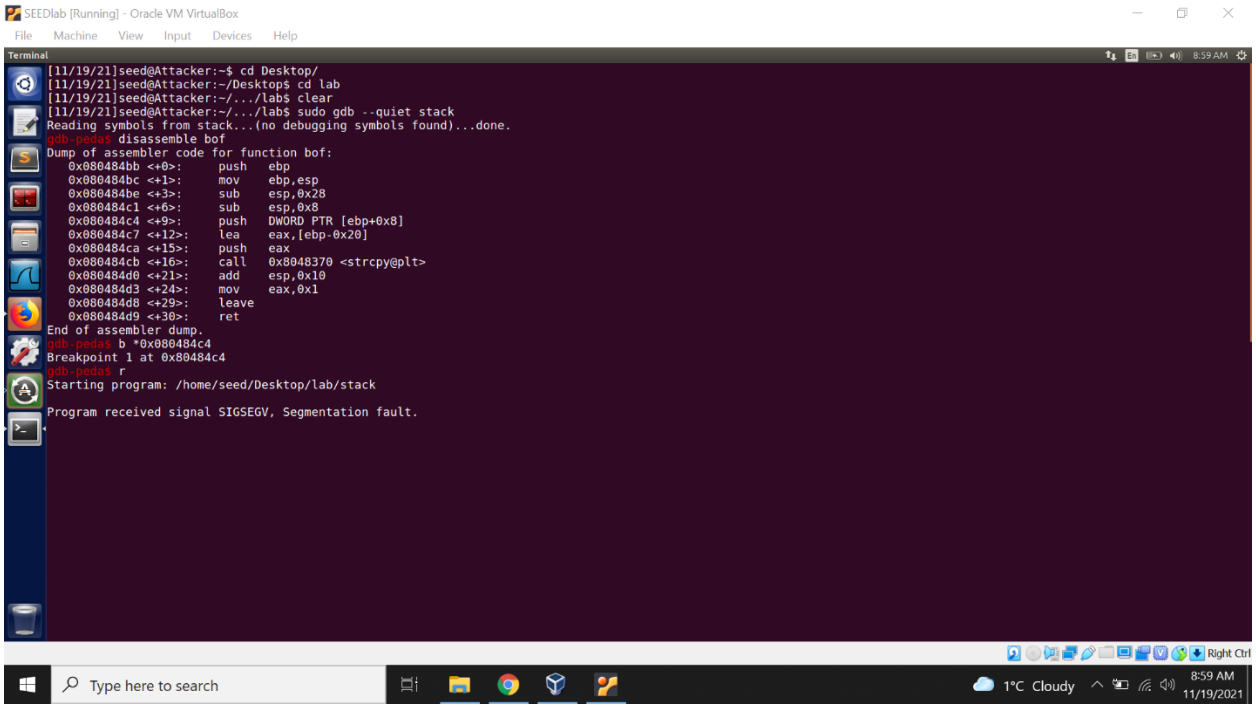


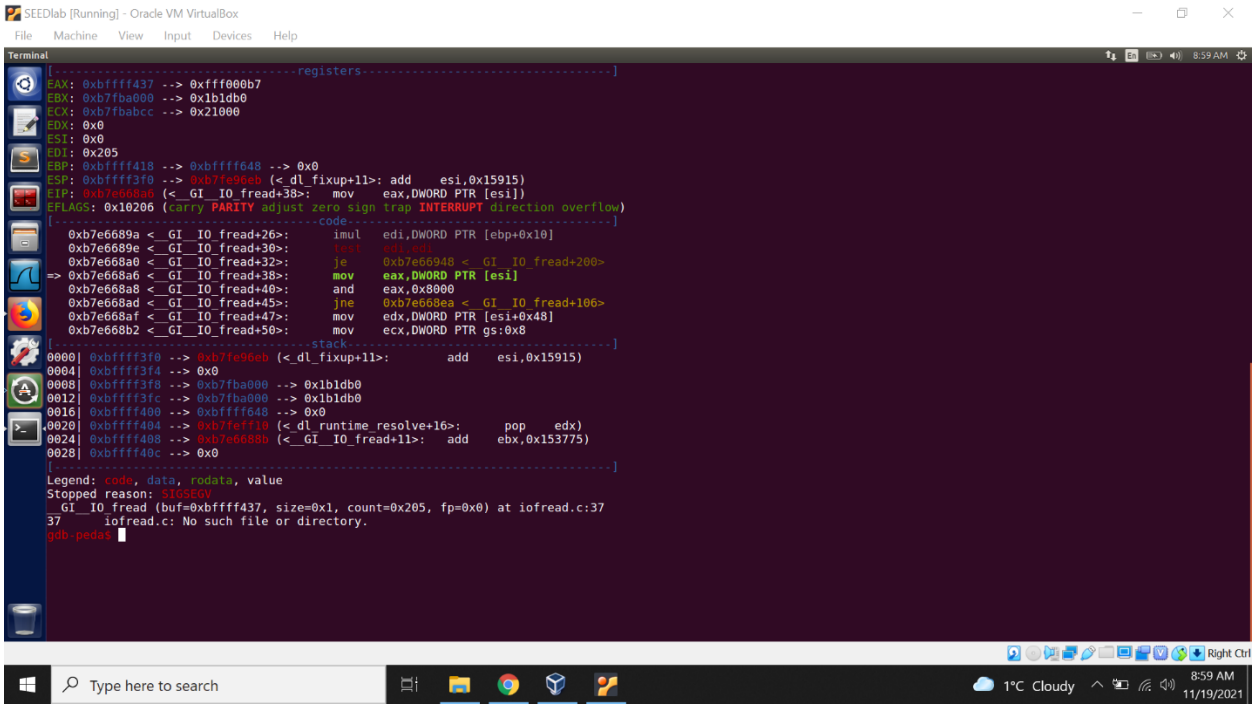


We create a breakpoint at the location of creation of the base pointer.

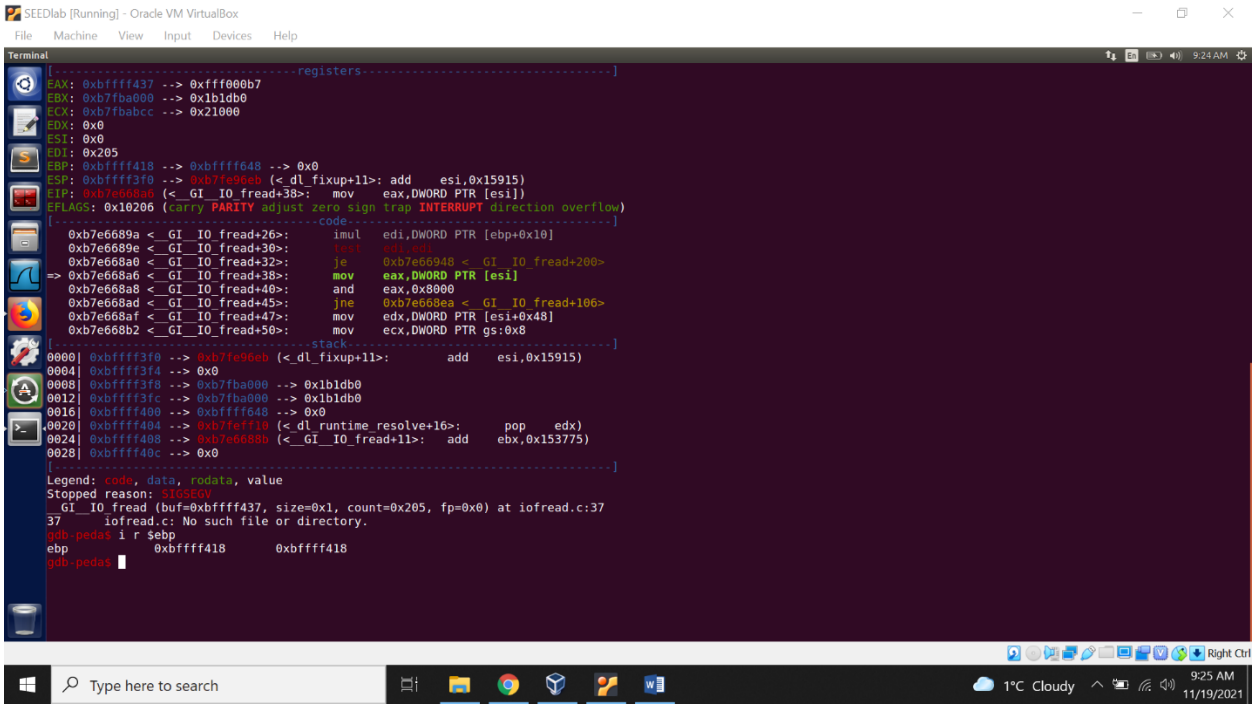


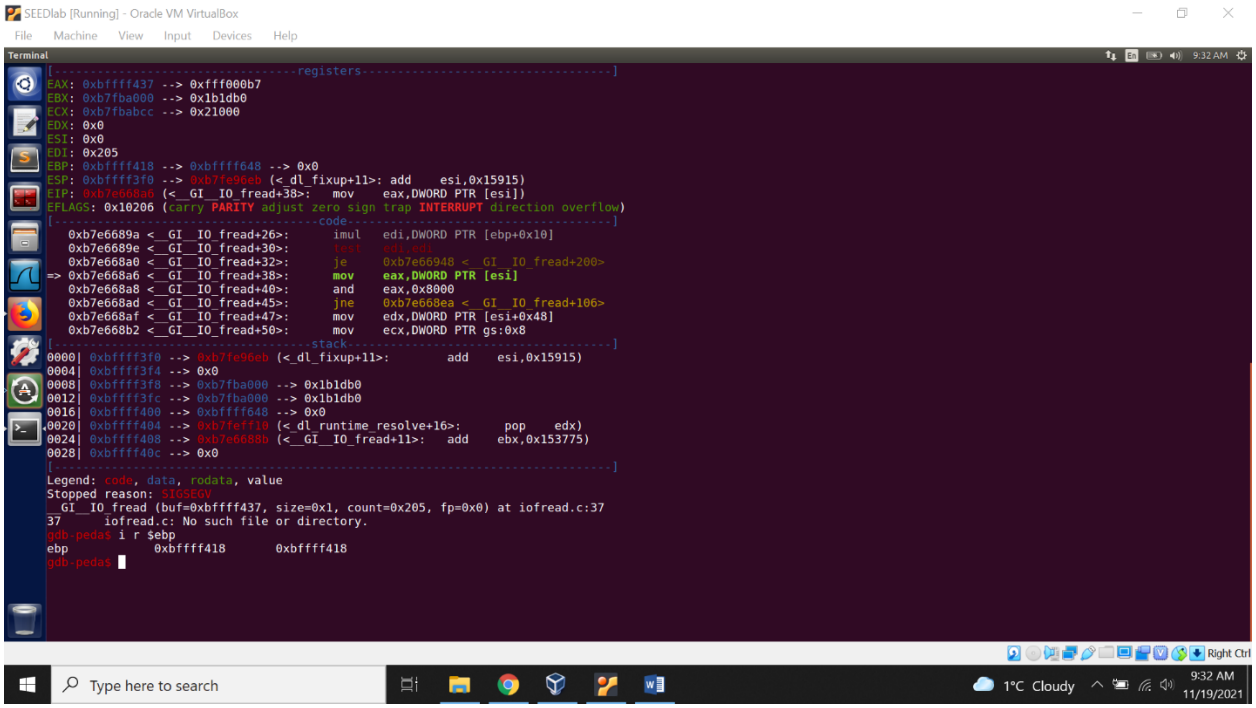
We run the rest of the program.



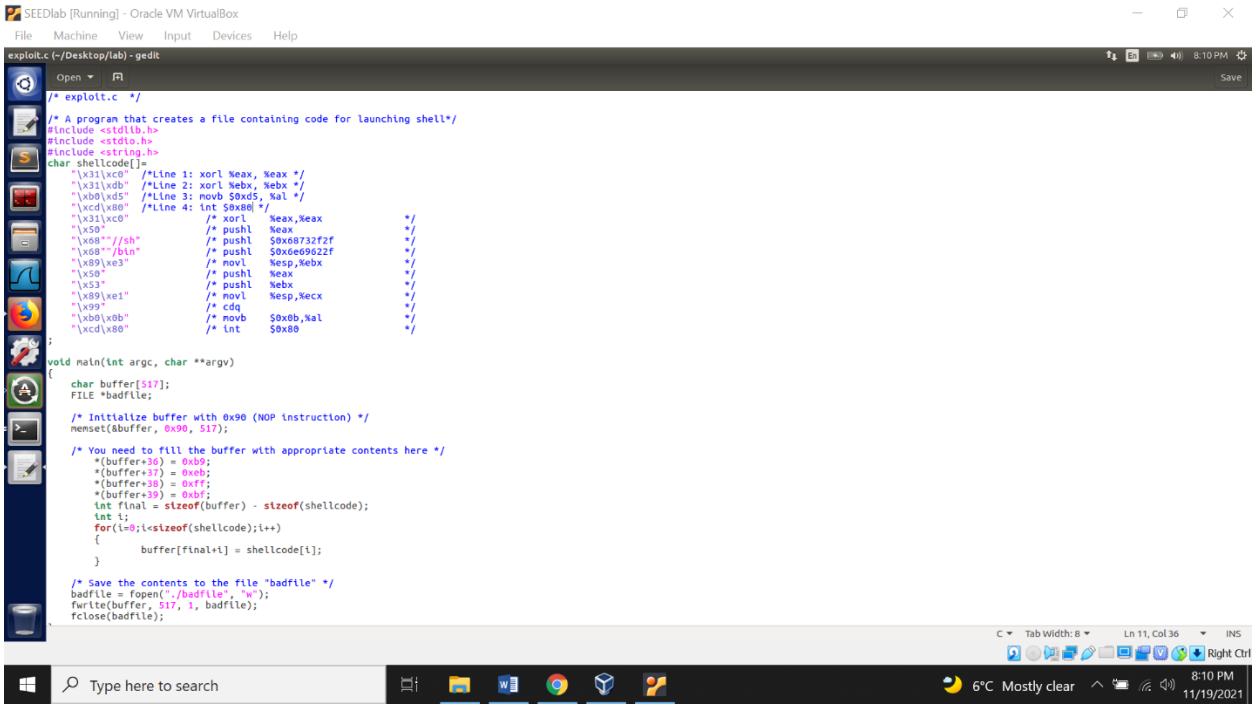


We find out the address of the base pointer ($ebp).

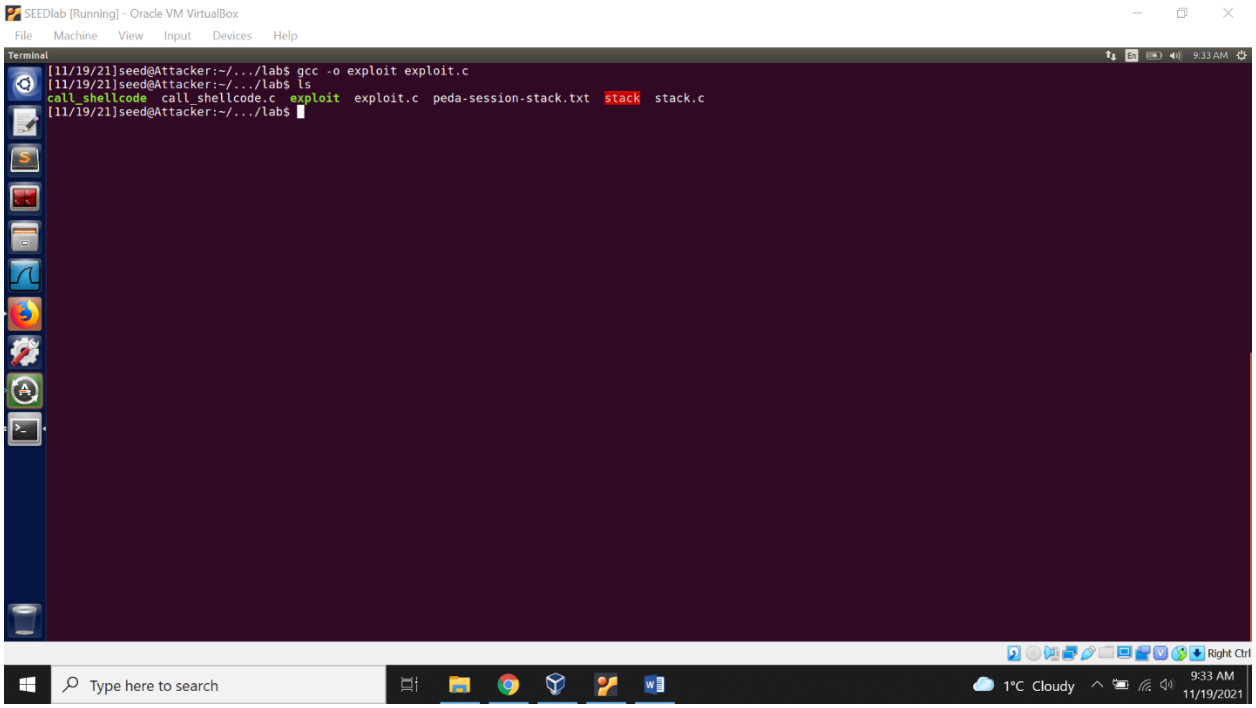




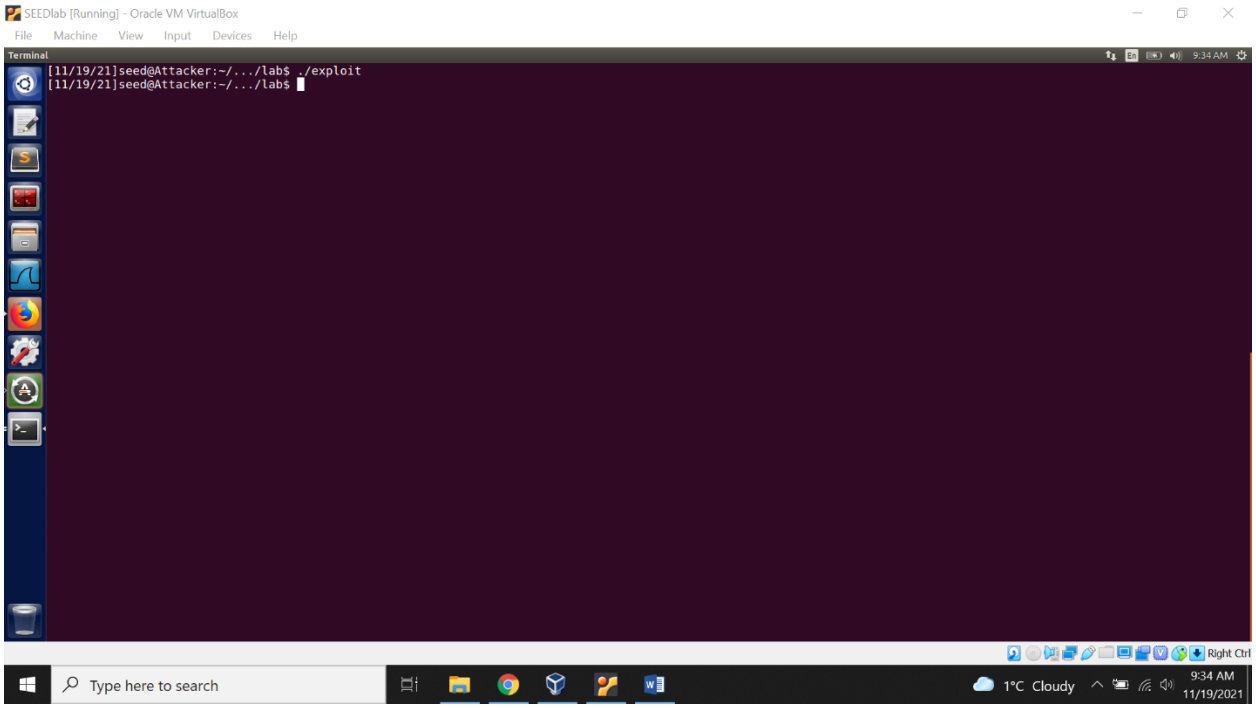
We change the exploit.c file as shown in the following screenshot.



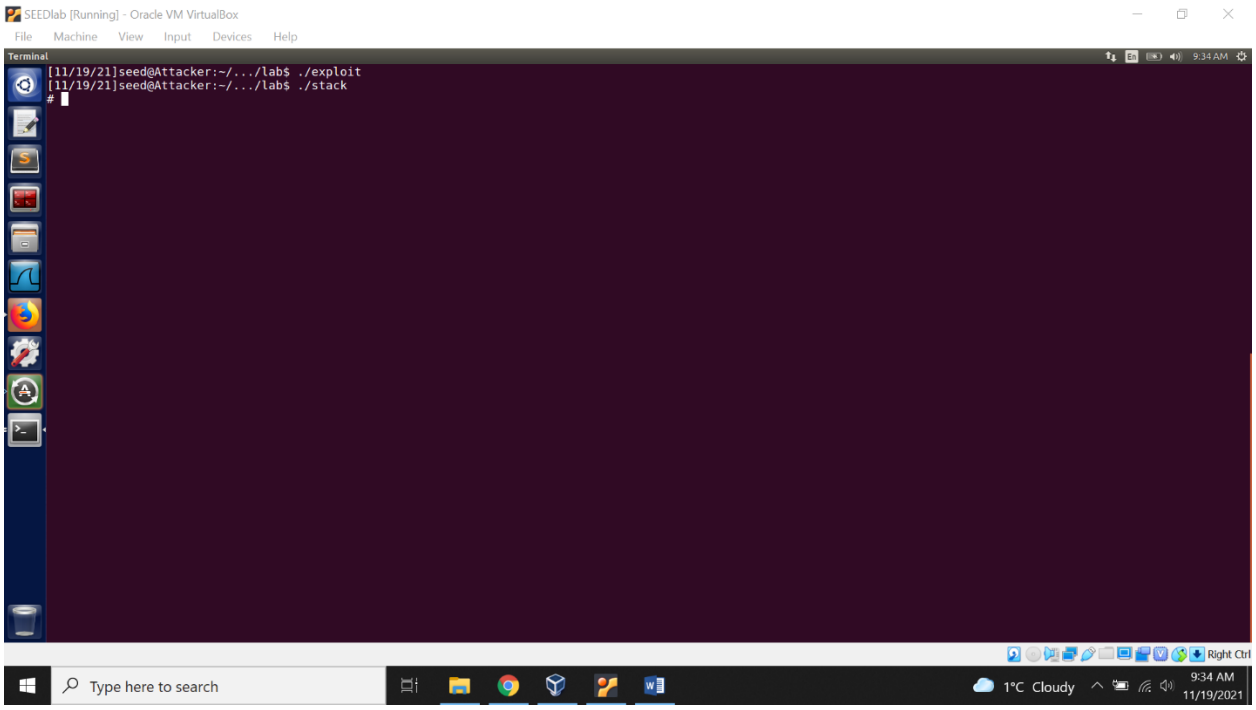
Compile the exploit.c file



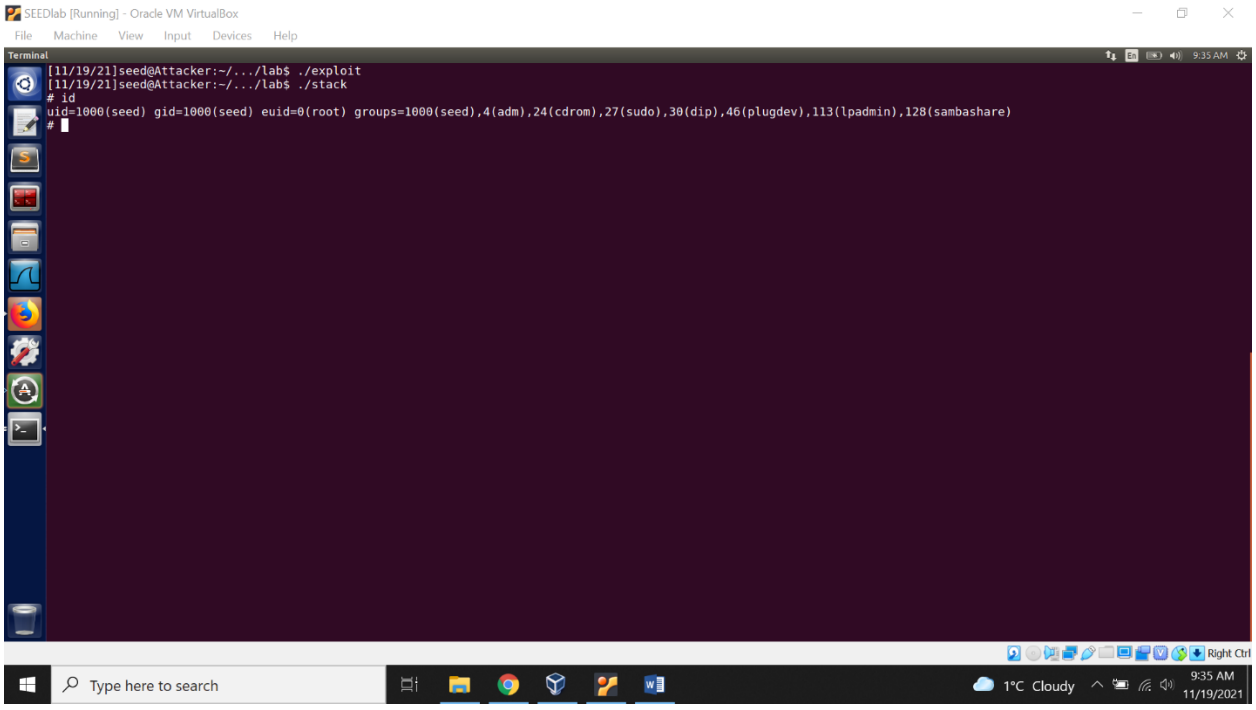
Execute the exploit.c file.



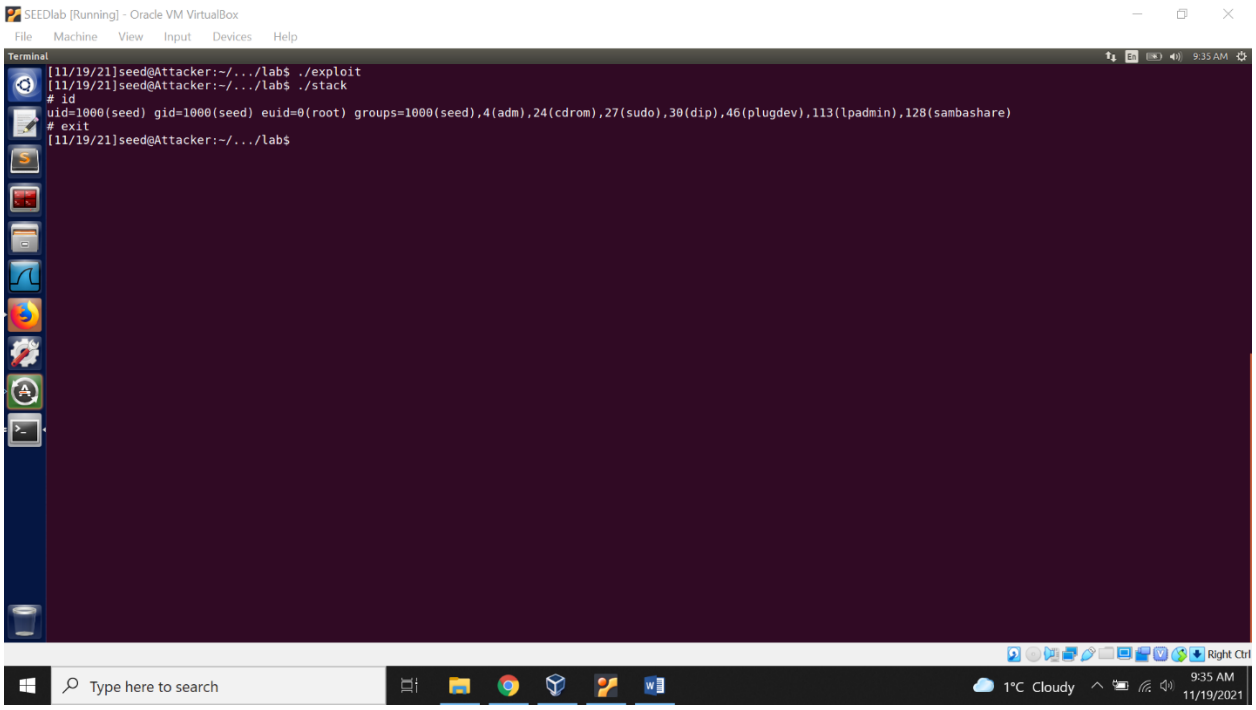
Execute the stack.c file.



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

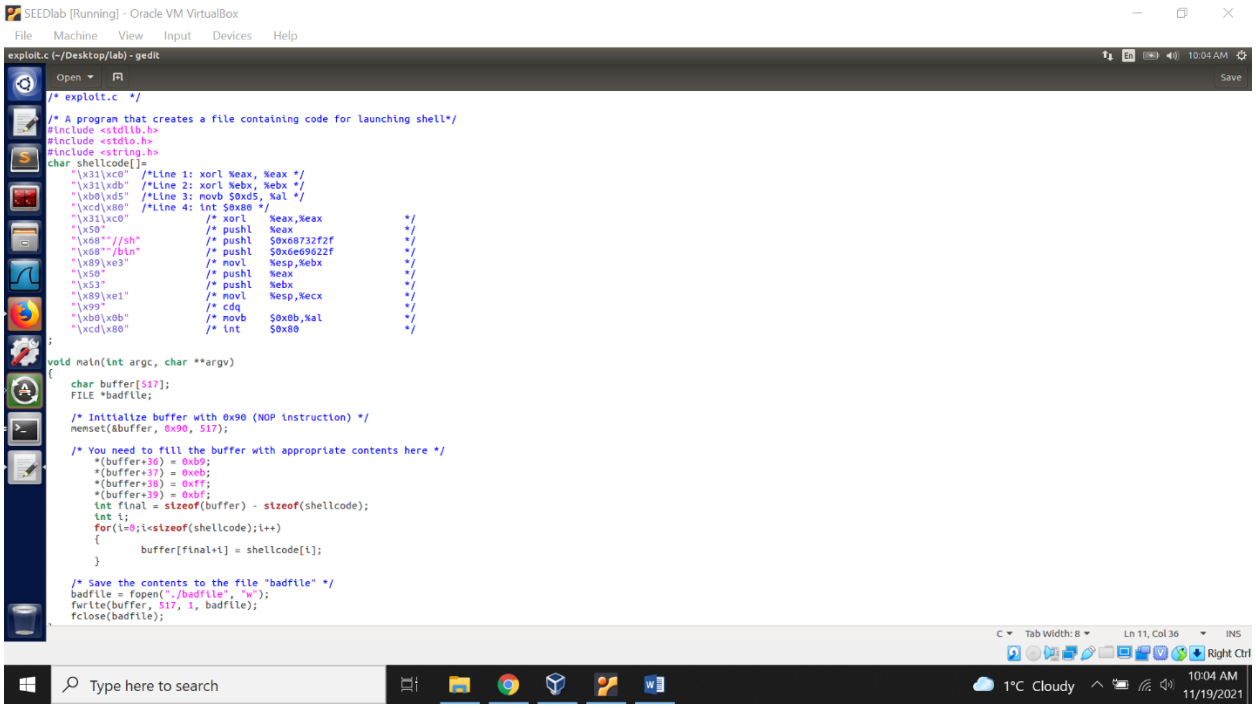


Observation: We check the UID. It is still the local UID, not root.

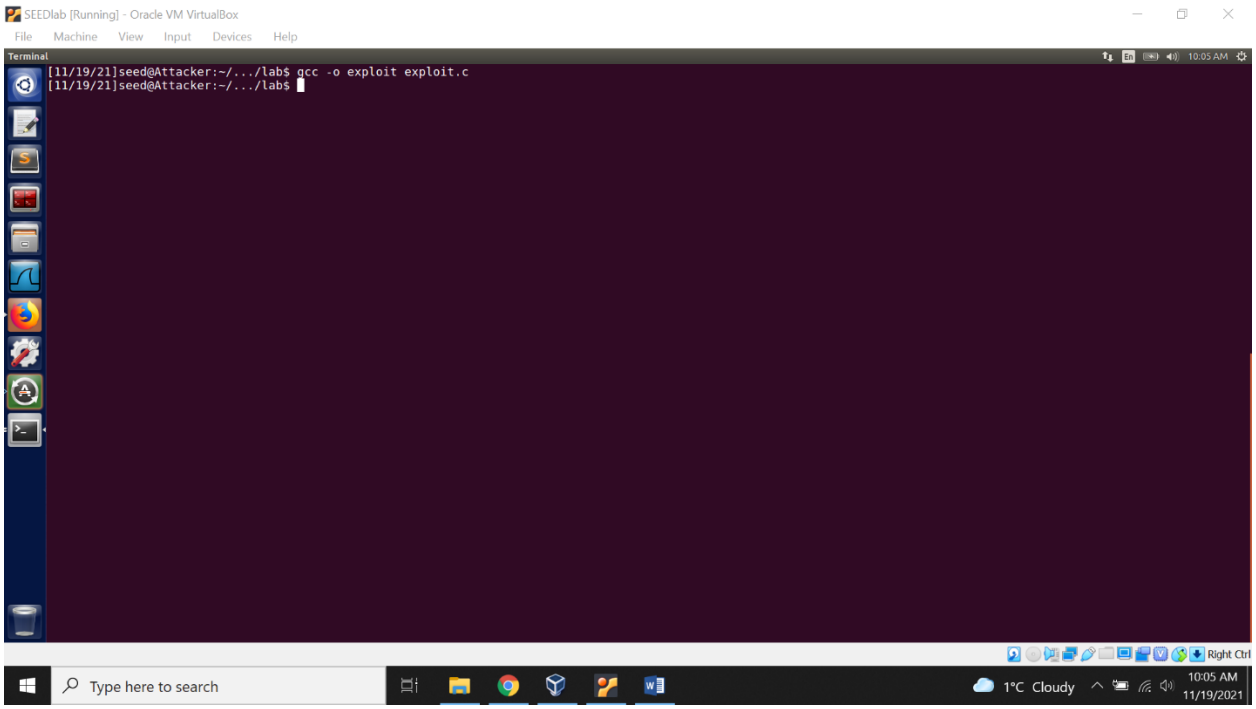


Task 3: Defeating dash's Countermeasure

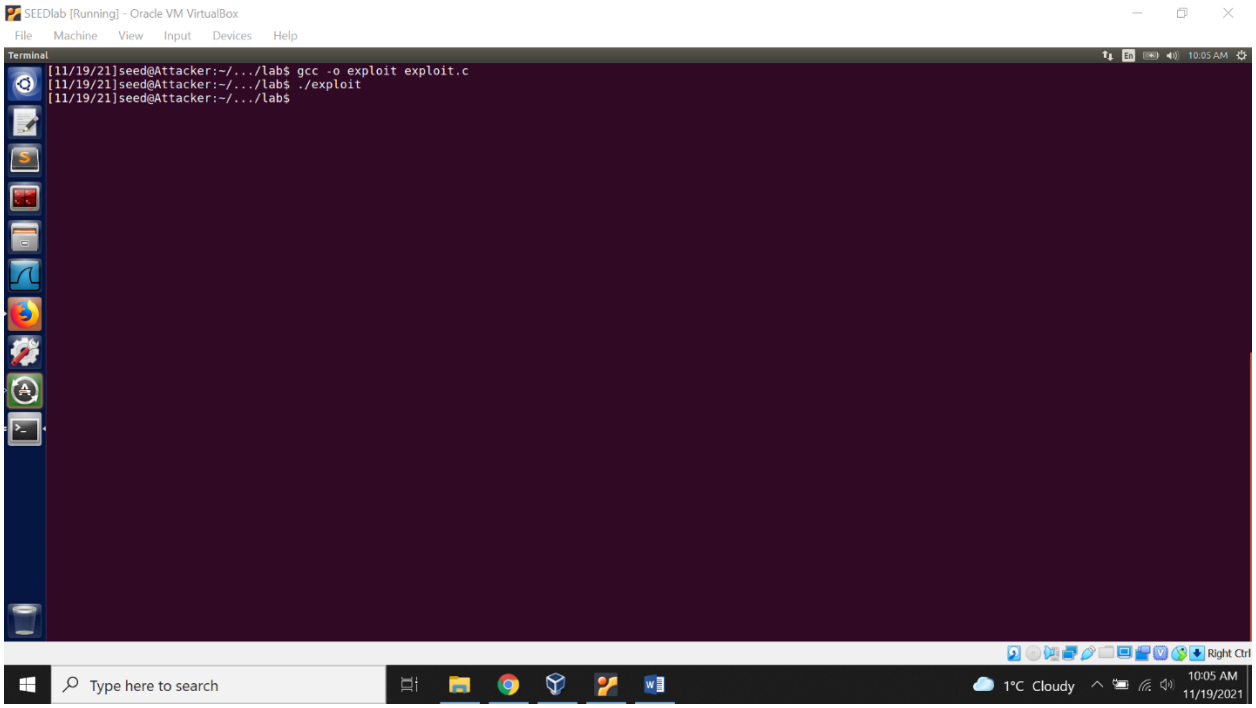
Now, we change our exploit.c make changes in the shellcode so that the UID is root.



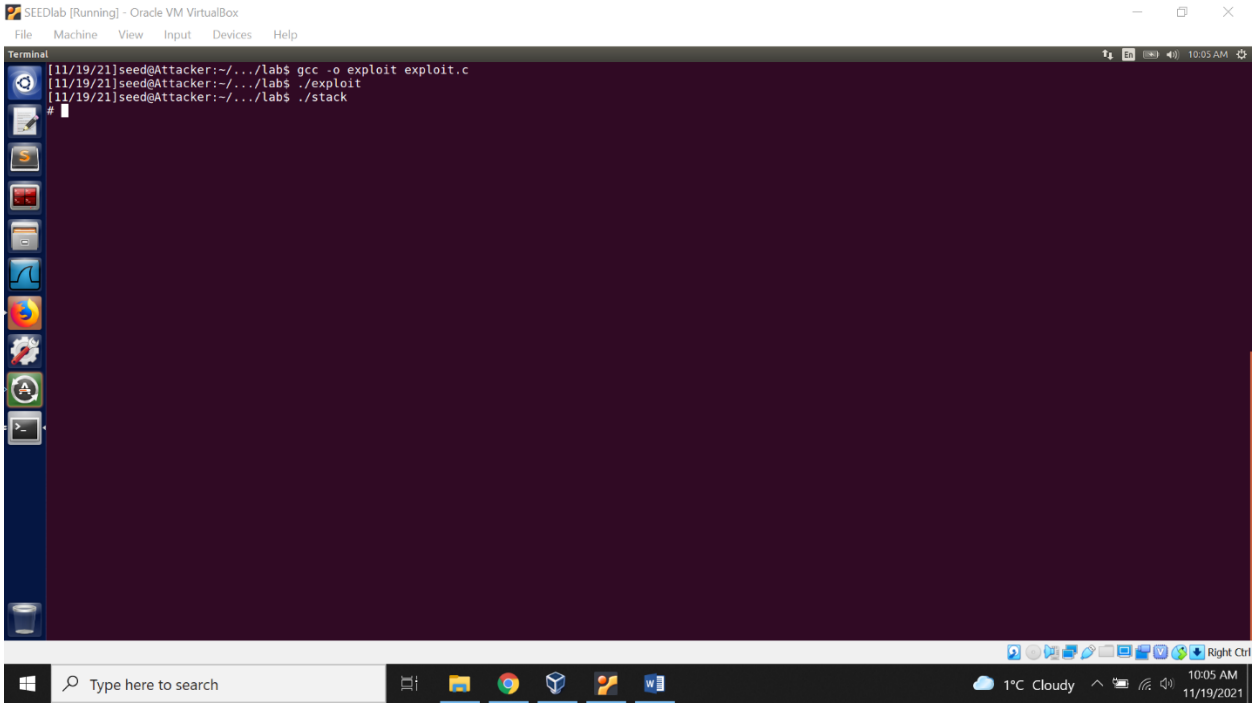
Compile the exploit.c file again.



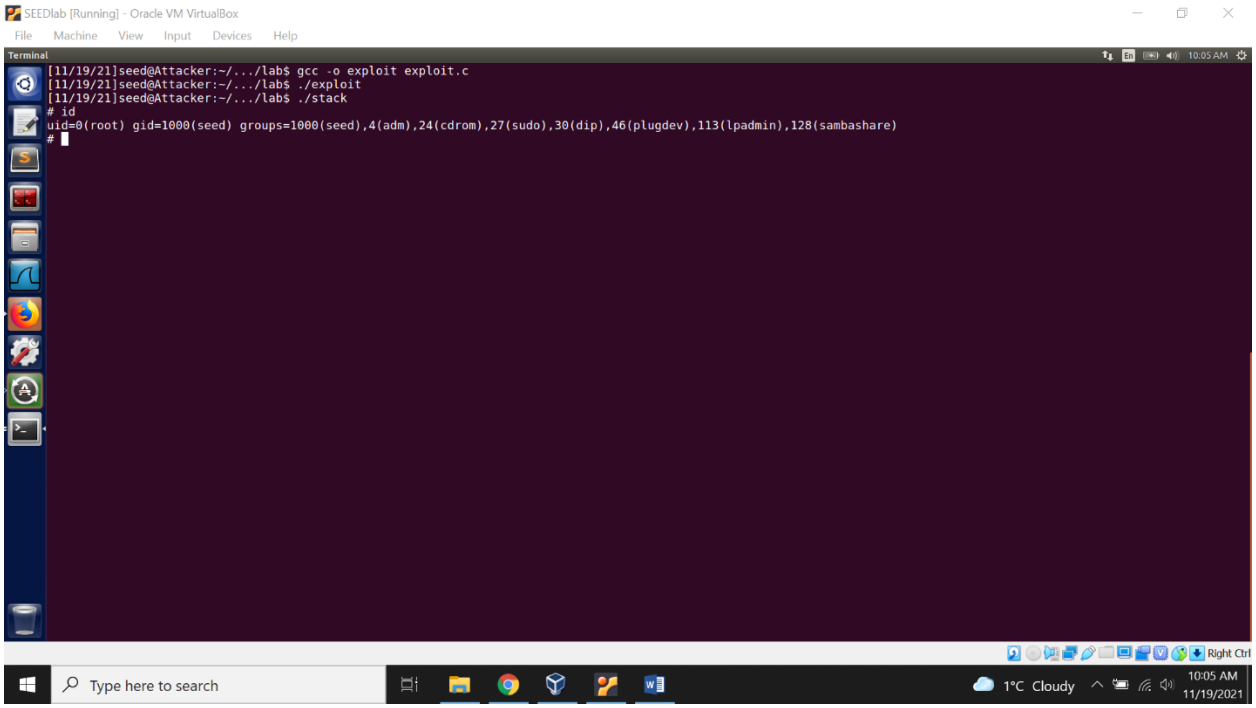
Execute the exploit.c file.



Execute the stack.c file

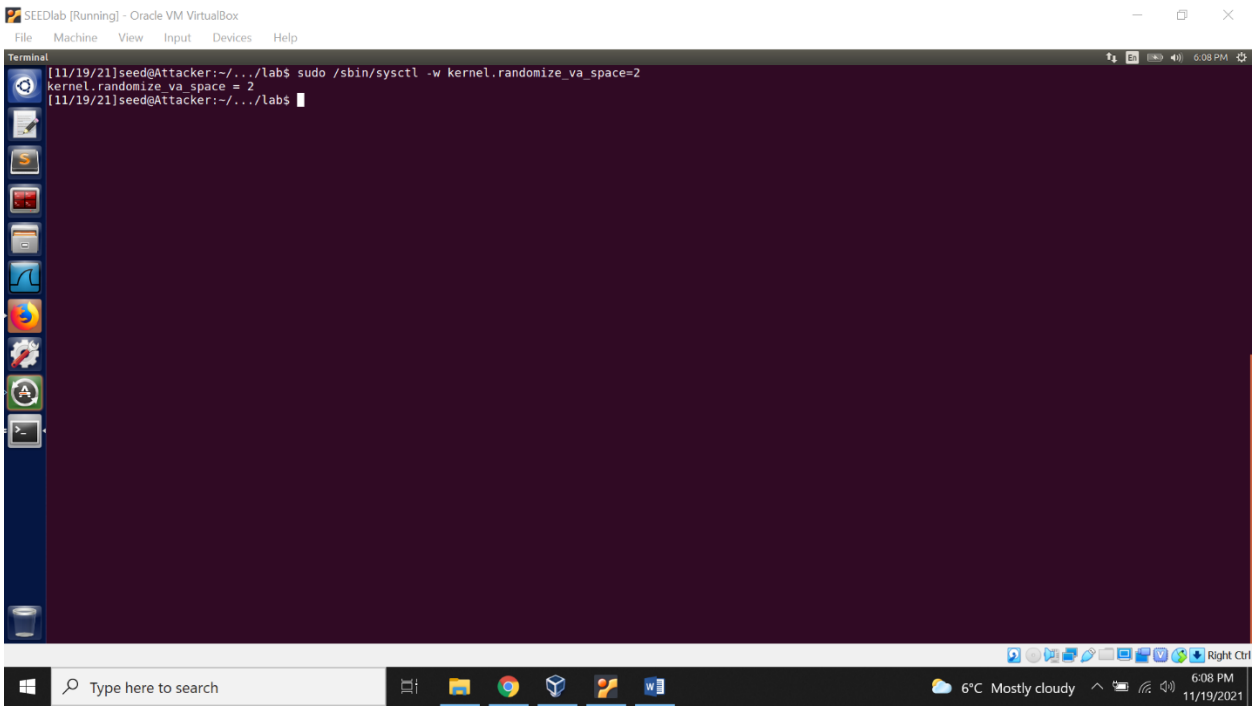


Observation: We entered the shell and the UID is root now.

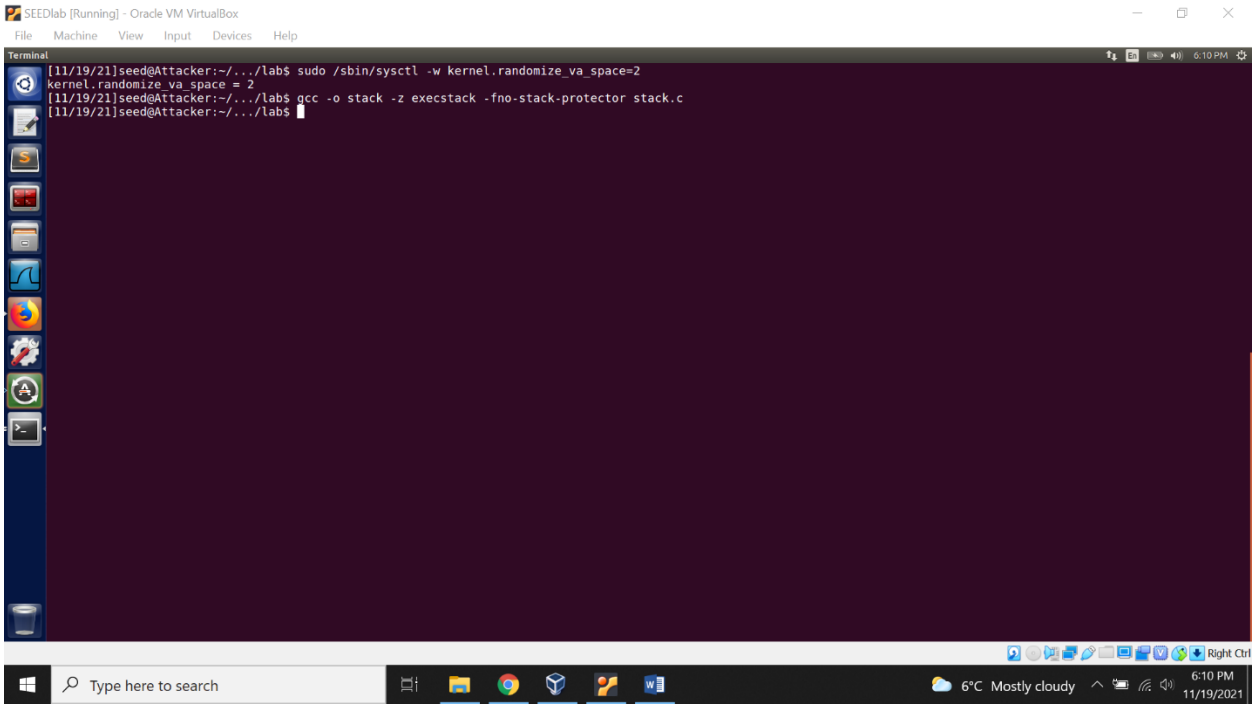


Task 4: Defeating Address Randomization

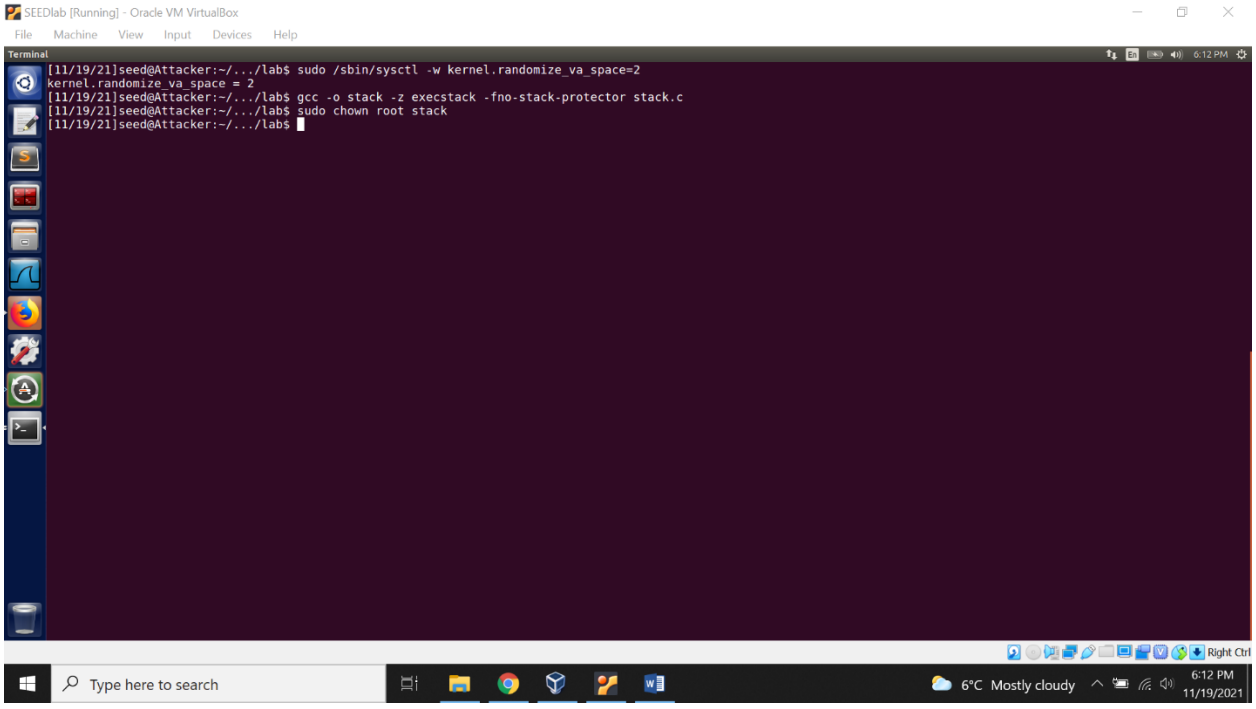
We turn on the Address Space Randomization using the command shown in the screenshot.



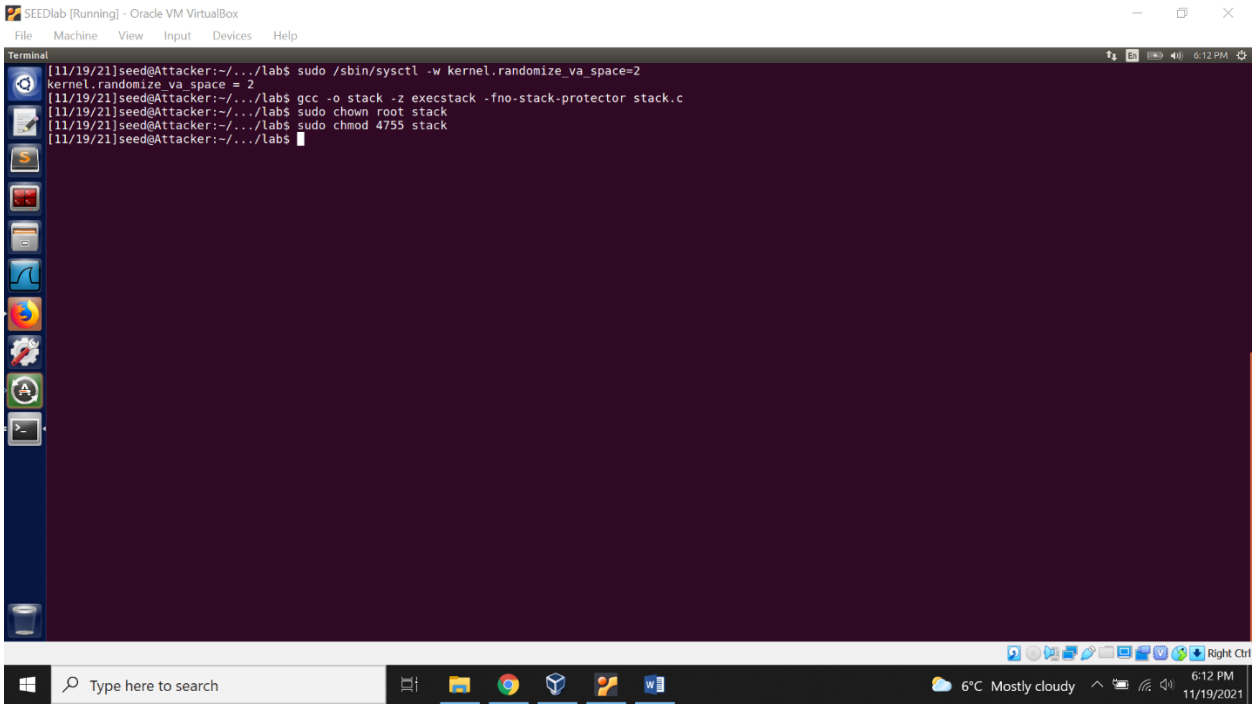
We again compile the stack.c file.



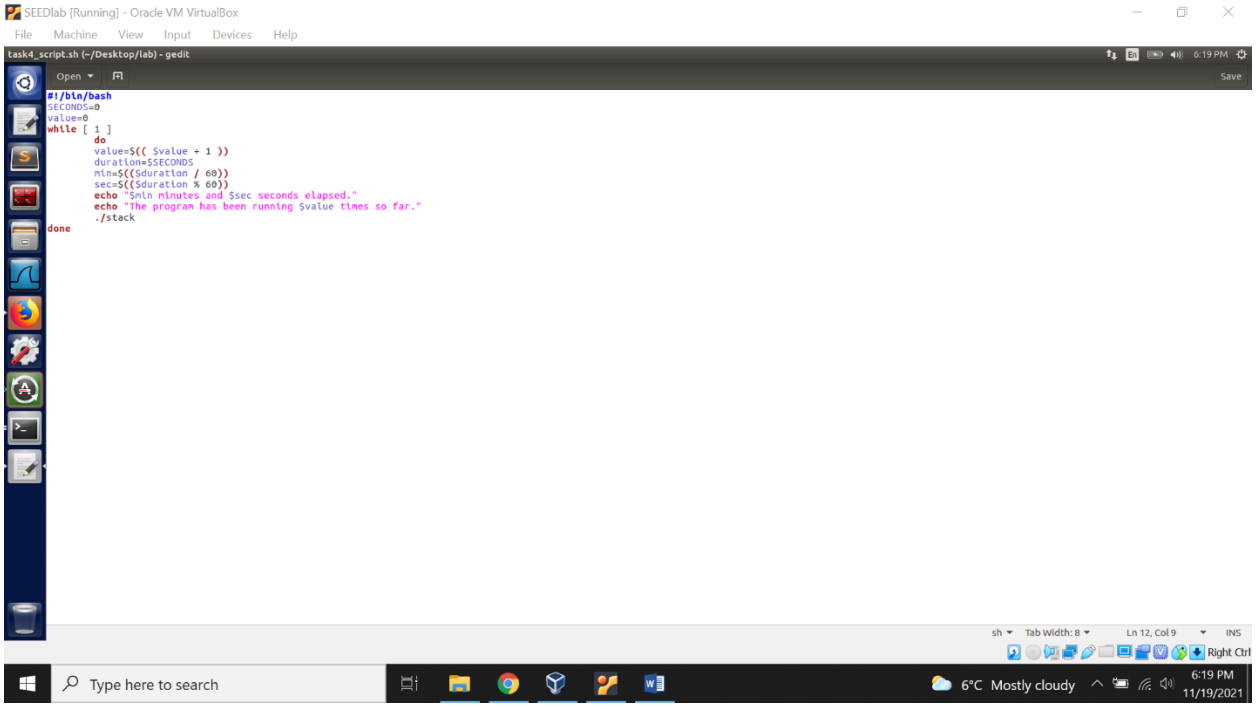
We change the ownership of the stack file to root.

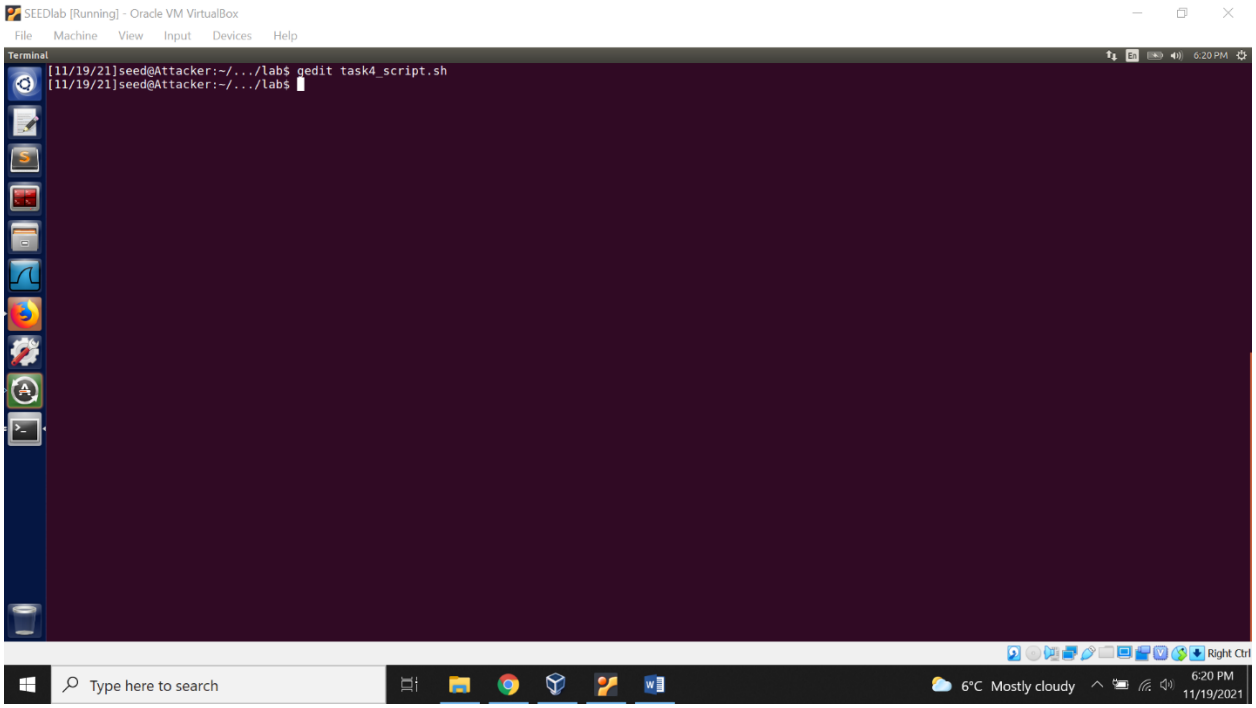


We change the mode of the stack.

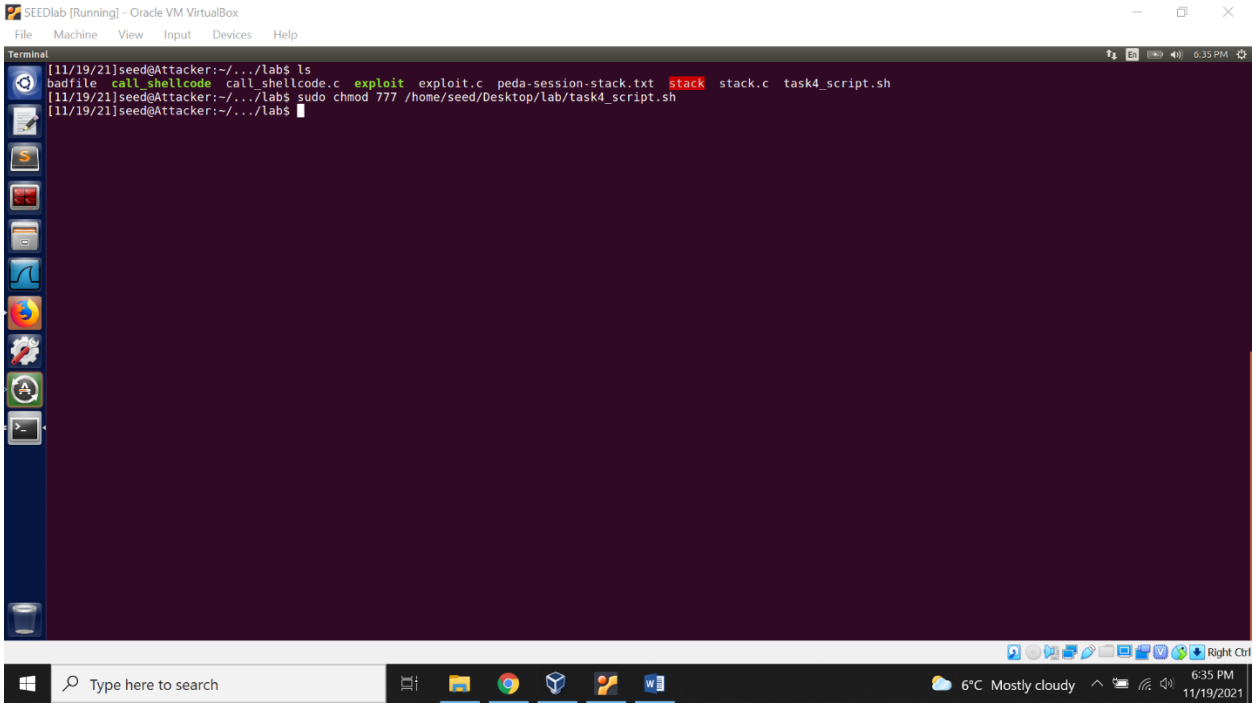


This is the shell script to run the vulnerable program in an infinite loop. The file name is task4_script.sh.

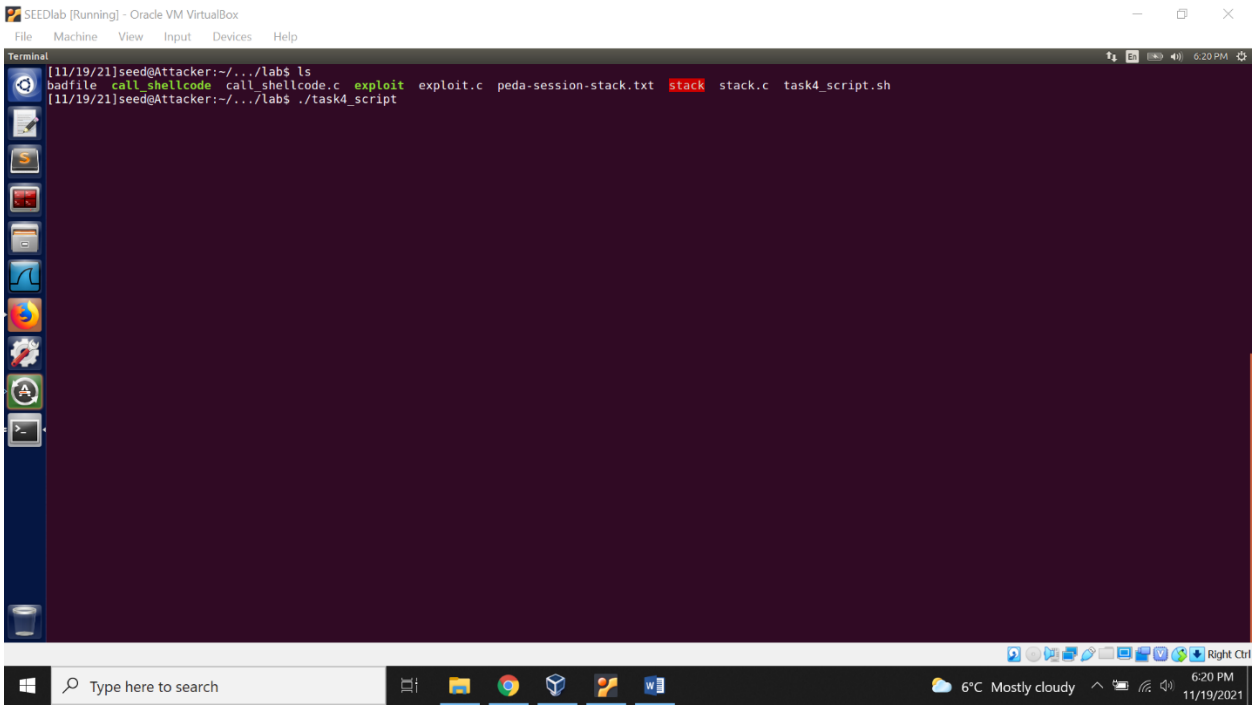




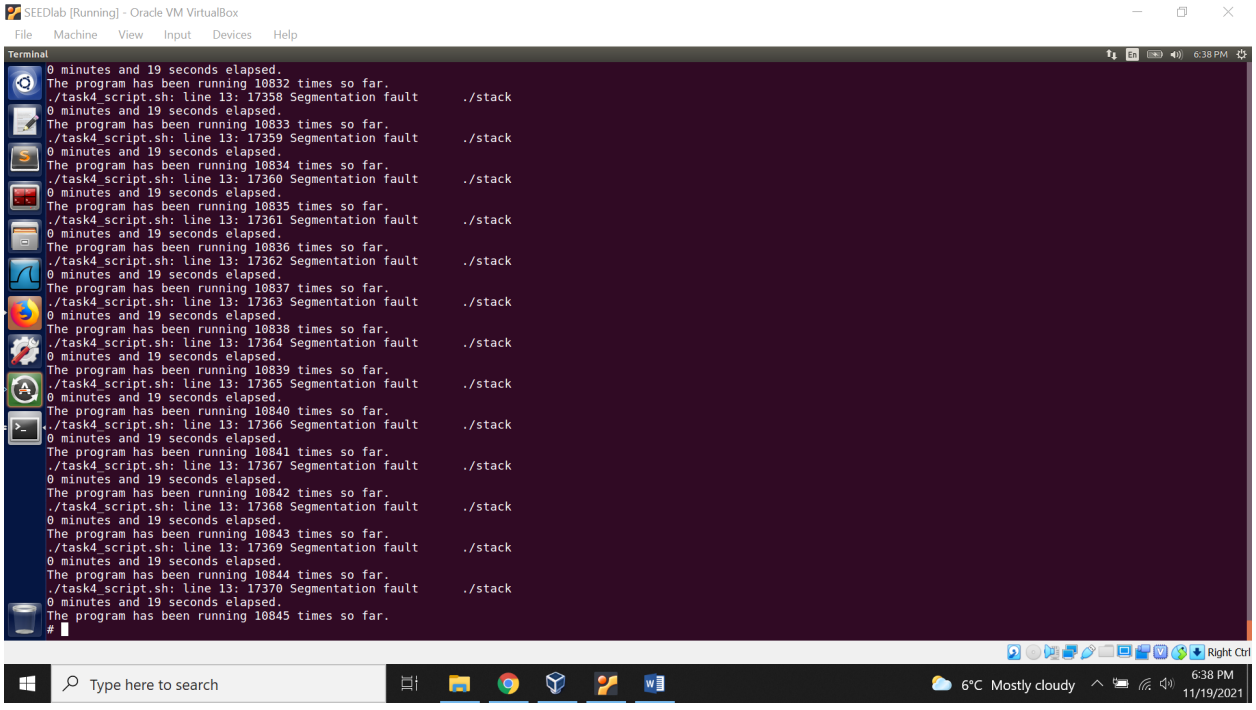
We make the task4_script.sh file executable.

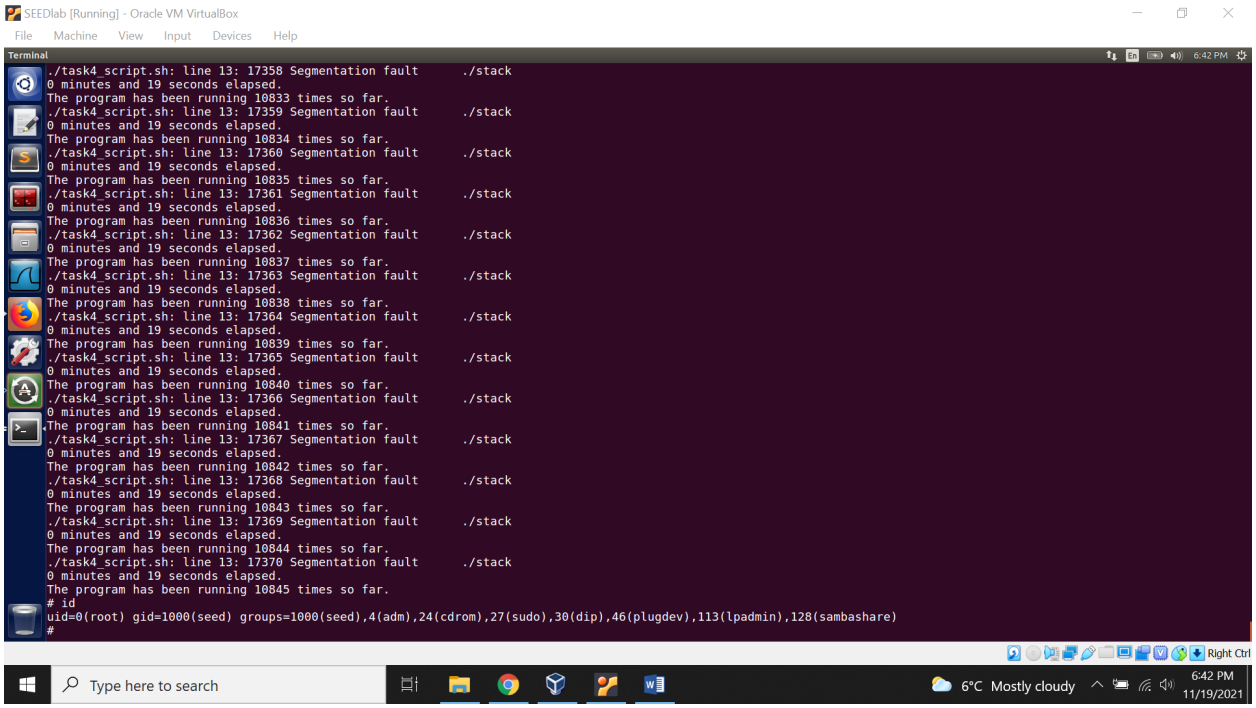


We execute the task4_script file.



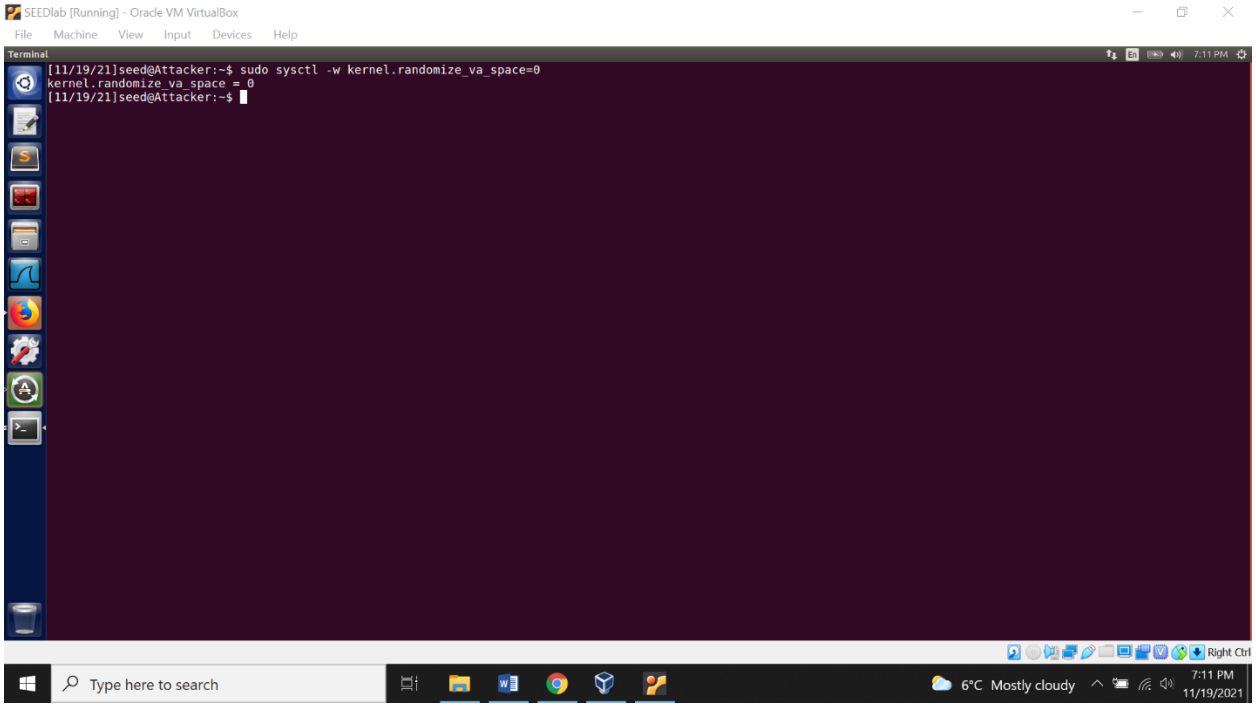
Observation: We run the program infinite times and defeated the Address Space Randomization as we have entered the shell as shown in the following screenshots.



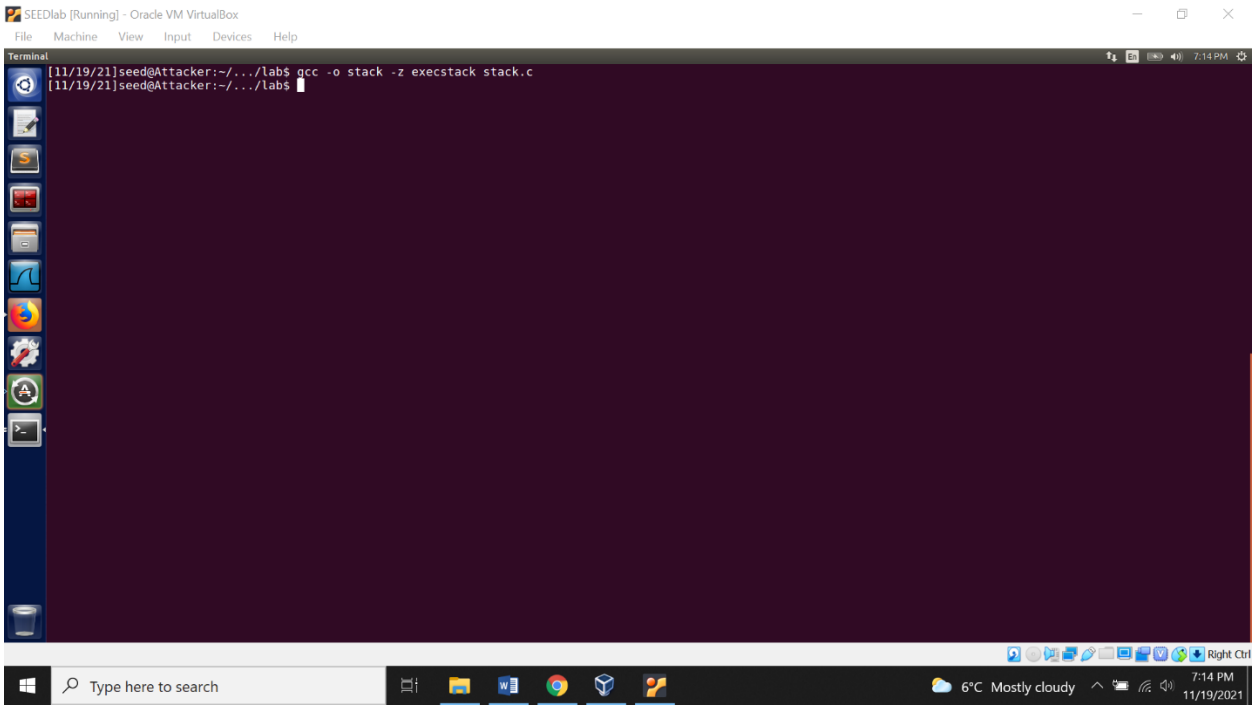


Task 5: Turn on the StackGuard Protection

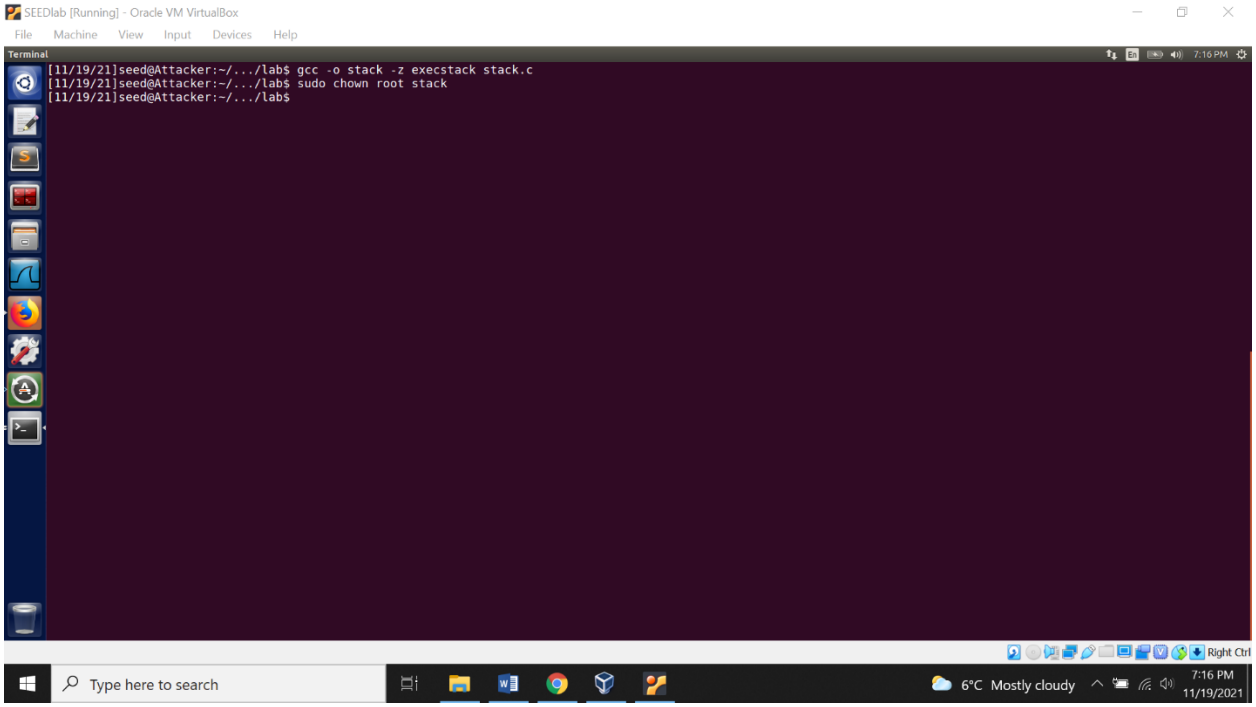
We firstly turn off the Address Space Randomization.



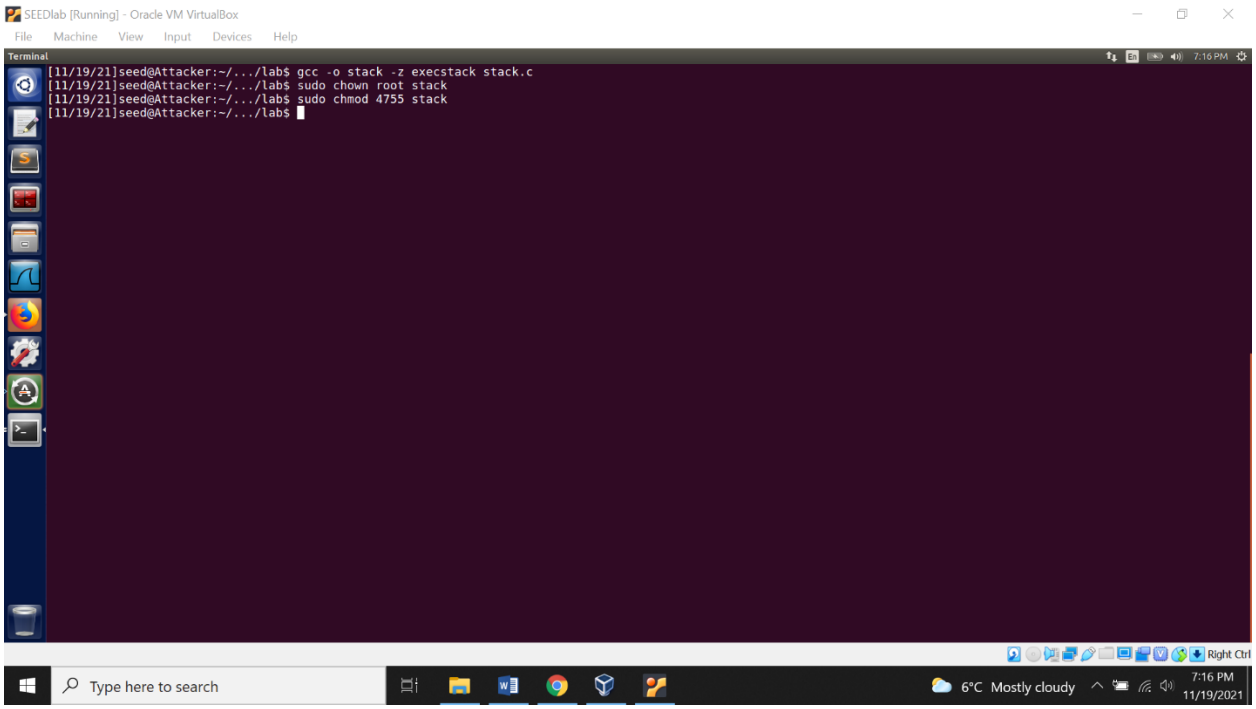
We compile the stack.c without the -fno-stack-protector option.



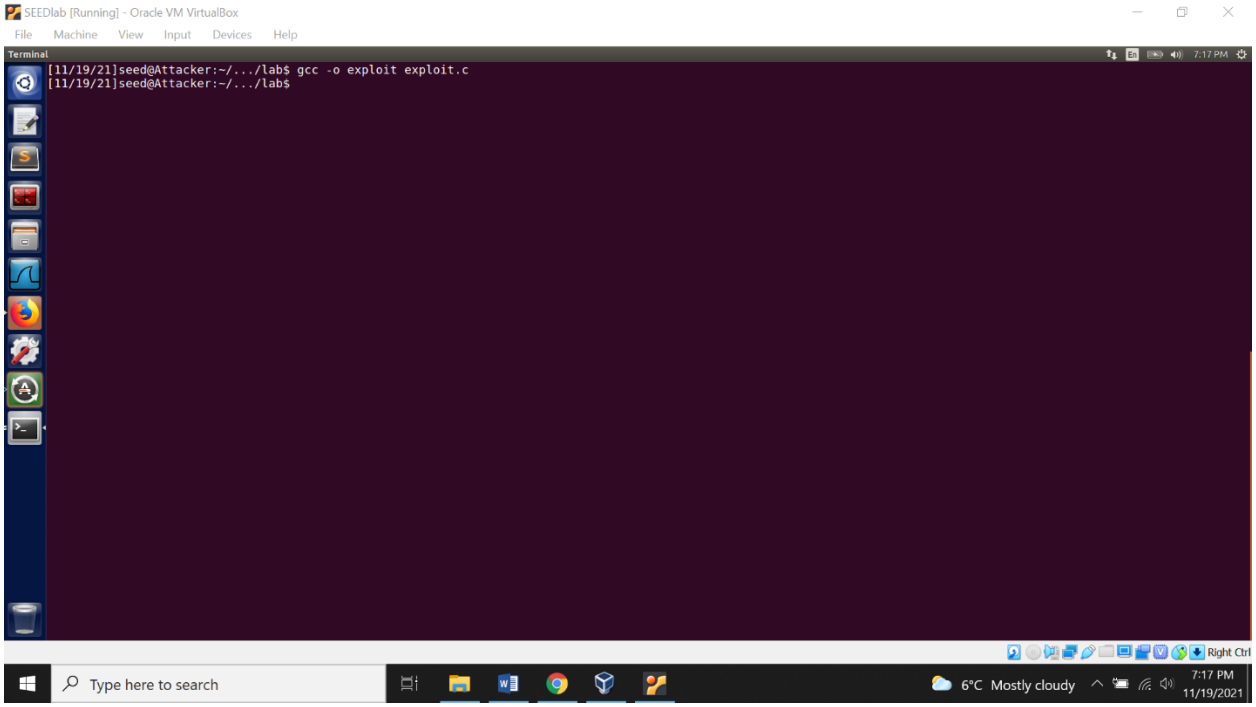
We change the ownership of the stack to root.



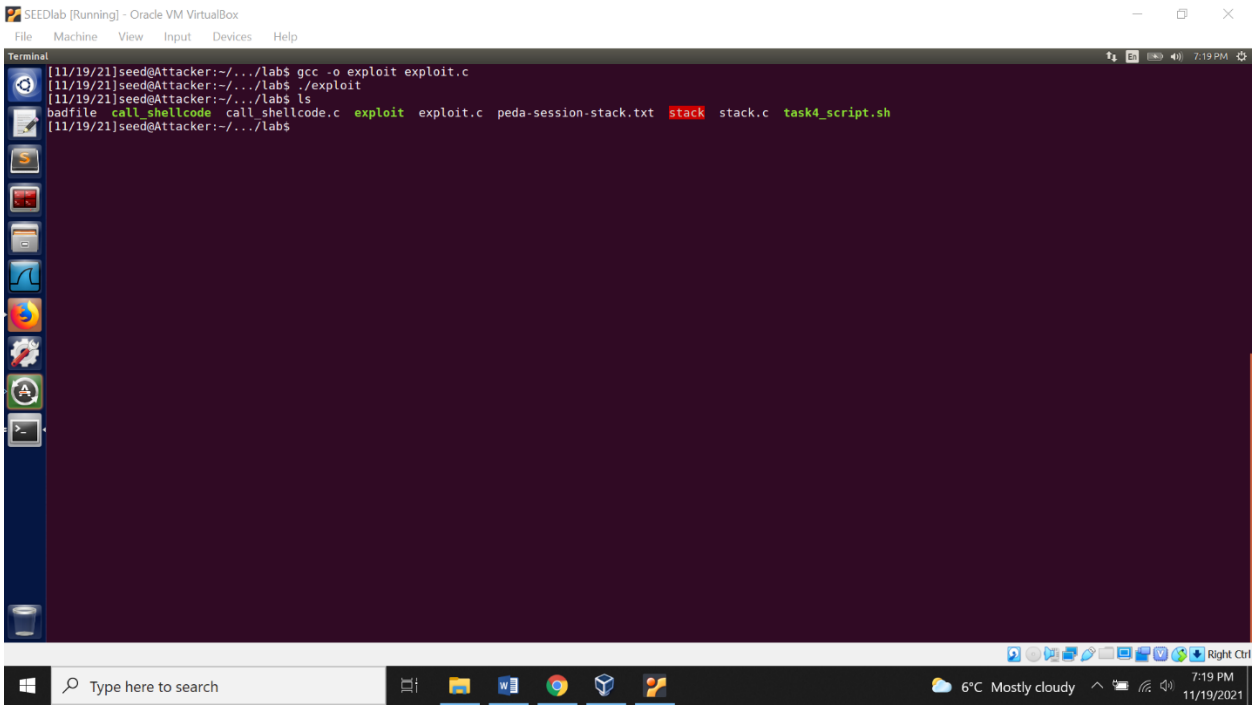
Again, we change the mode of the stack.



Compile the exploit.c file.



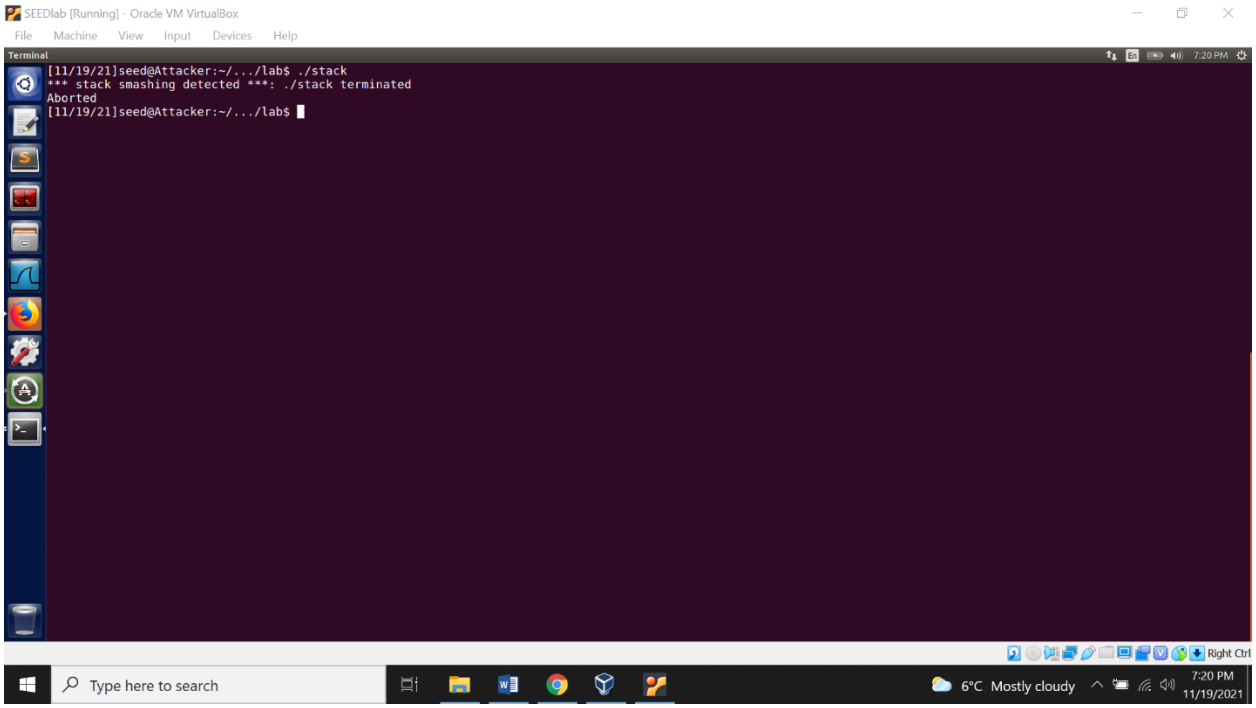
Execute the exploit.c file.



Execute the stack.c file.

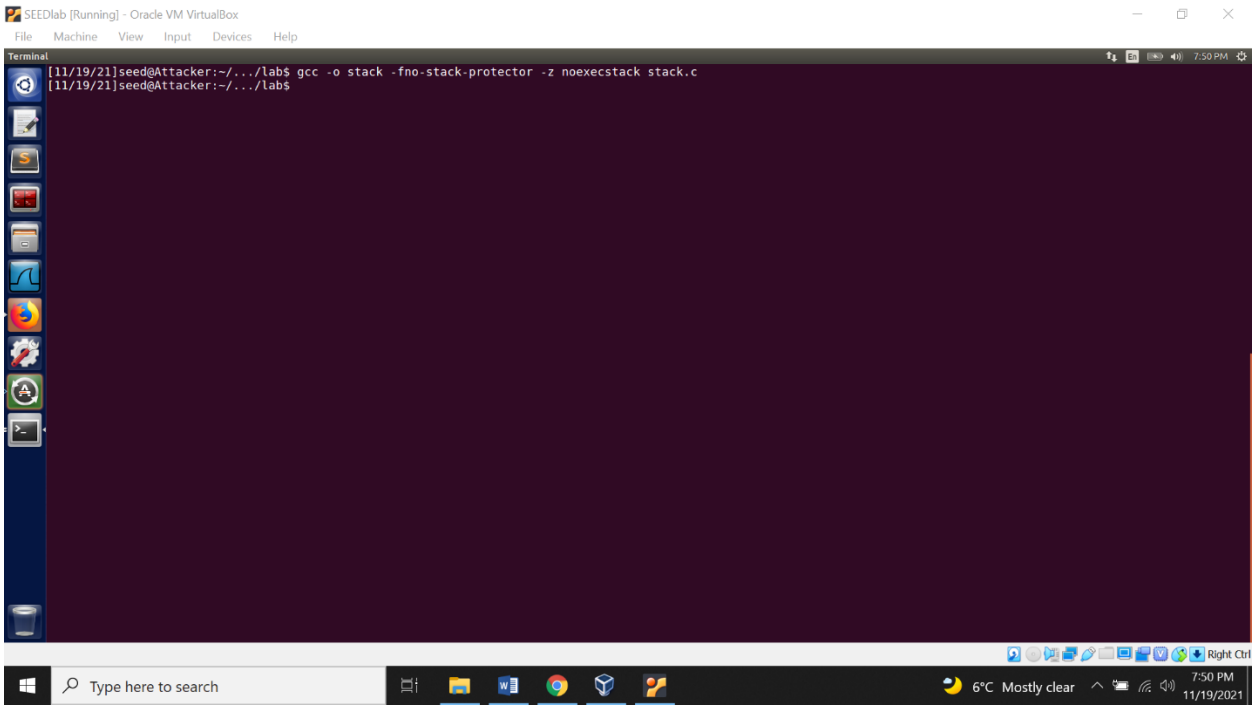
Observation: We got an error saying the "./stack is terminated" because the Stack Guard places a random integer value before the return address and when we try to overwrite the return address, that random value is also being overwritten. Since, the StackGuard know the value of the random integer, and

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

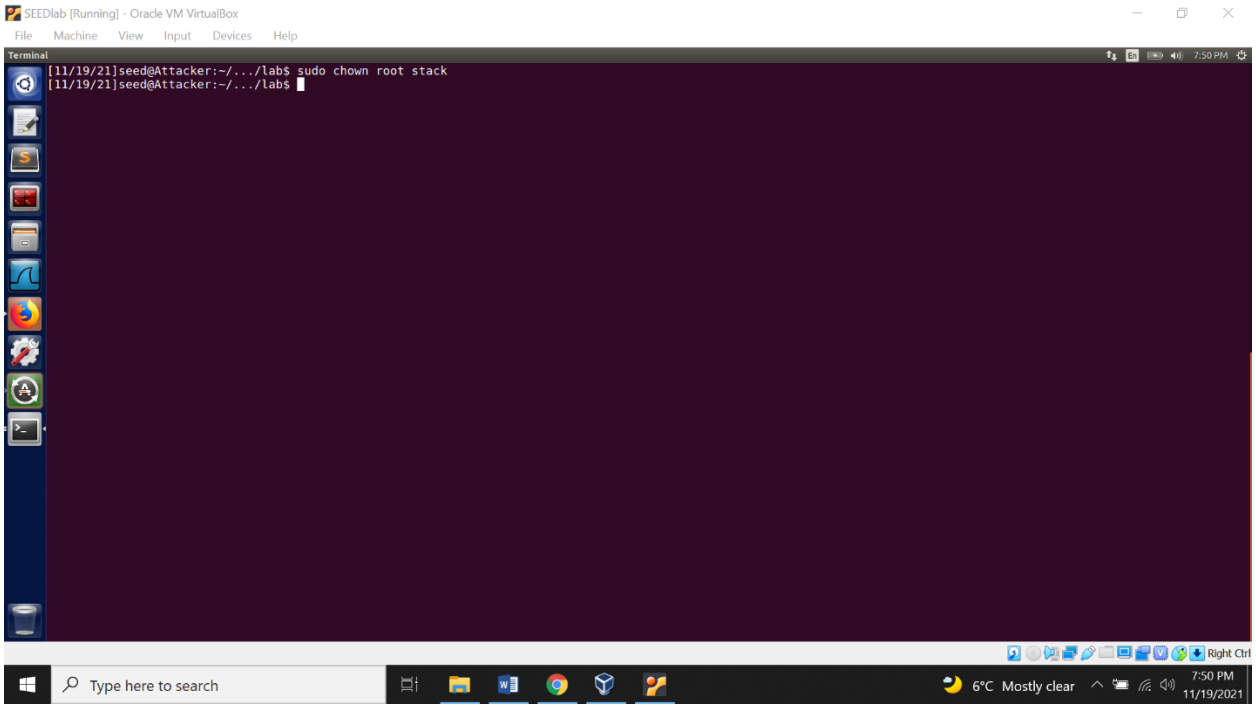


Task 6: Turn on the Non-executable Stack Protection

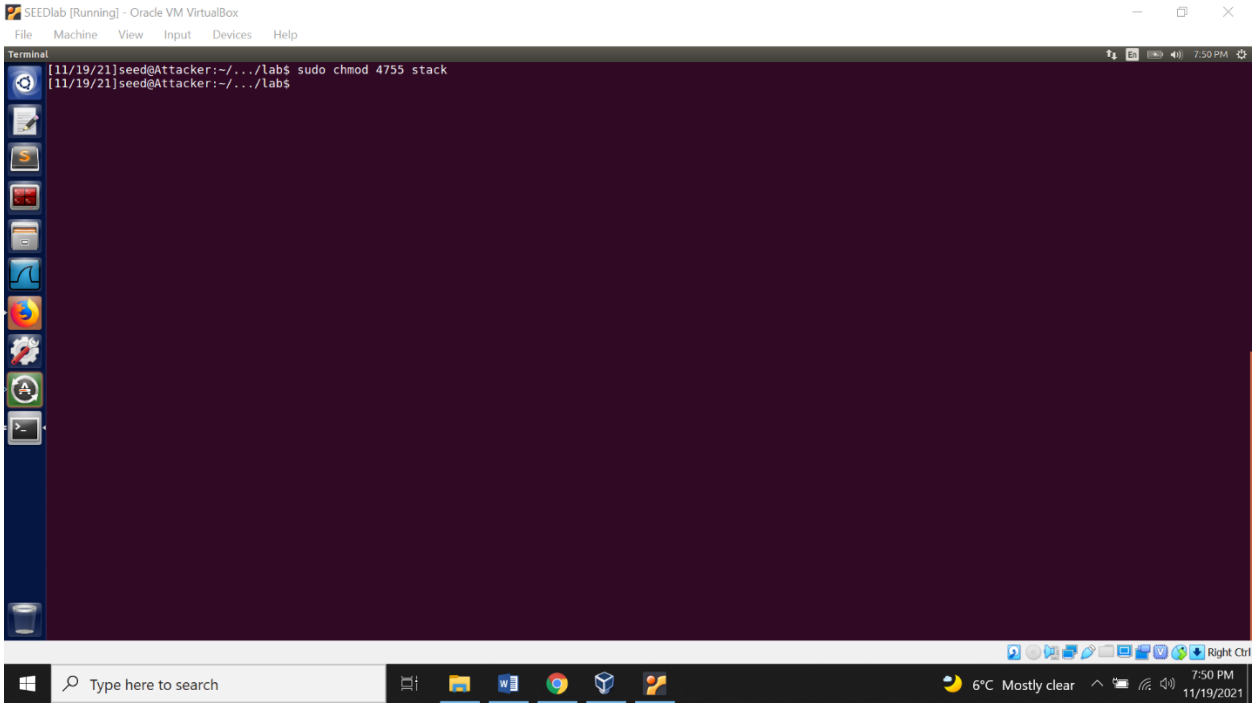
We compile the stack.c file using the non-executable stack protection



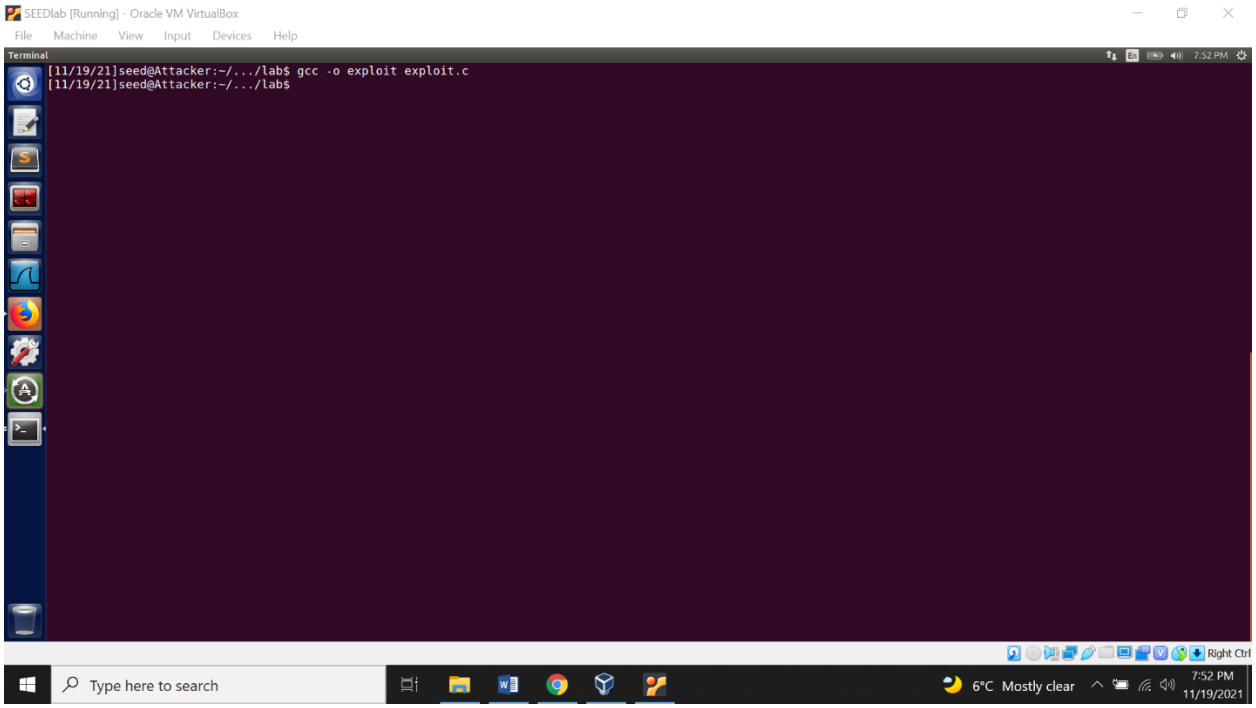
Change the ownership of the stack to root.



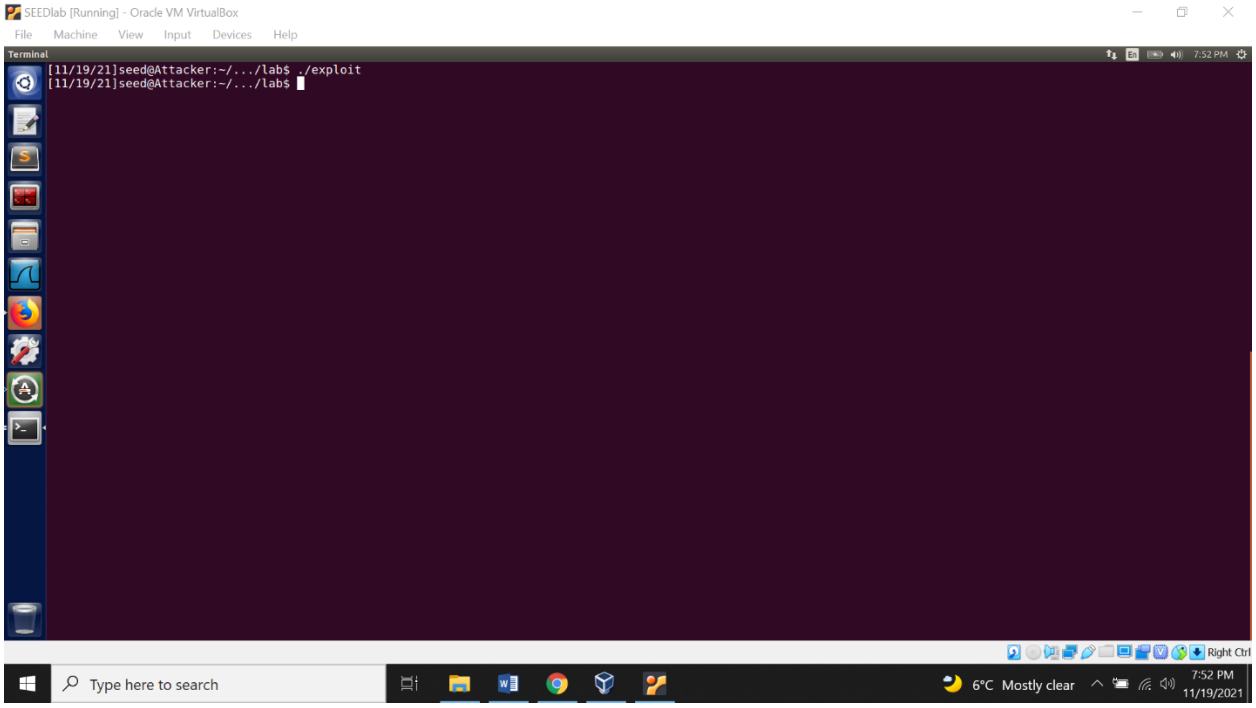
Changing the mode of the stack again.



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