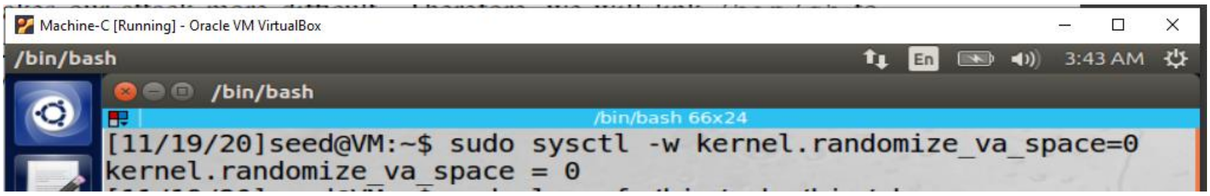
LAB: Buffer Overflow Vulnerability Lab

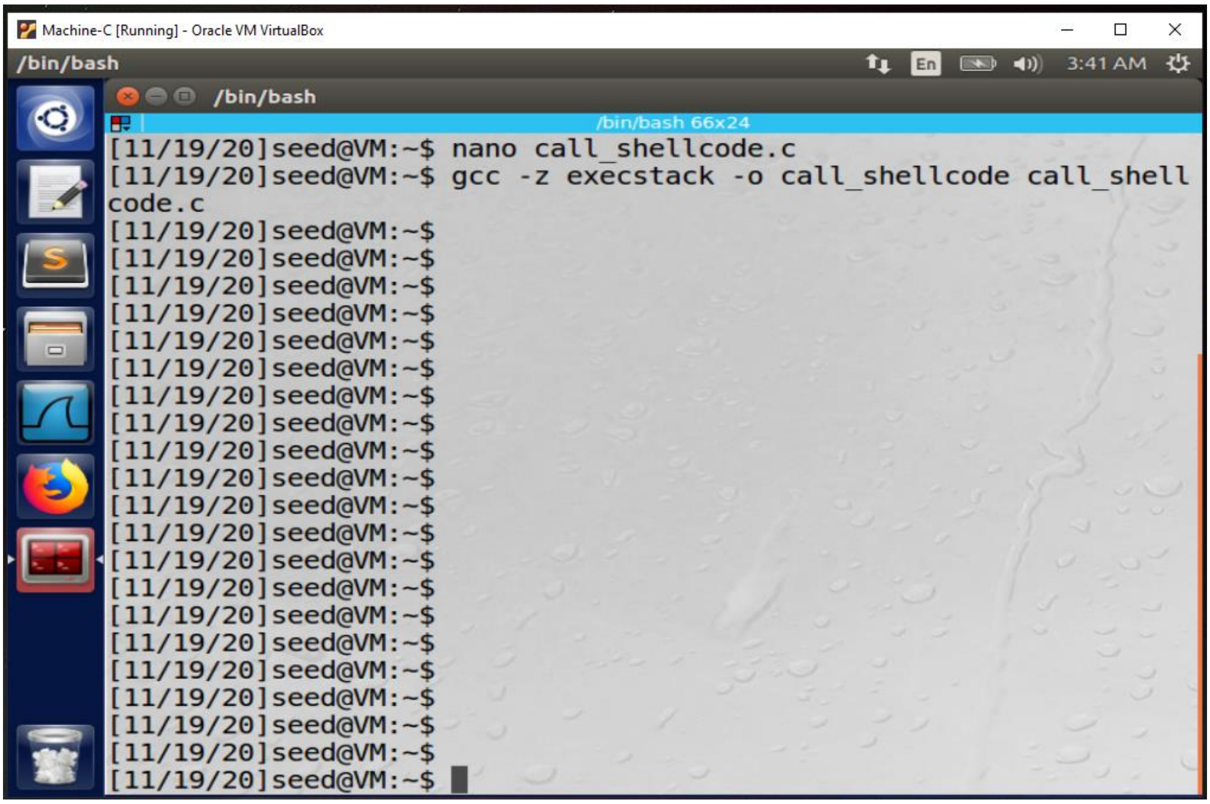
This lab is about buffer-overflow vulnerability. Buffer overflow is defined as the state in which a program attempts to write data outside the limits of fixed length buffers pre-allocated. A malicious user may use this vulnerability to modify the program's flow control, which leads to malicious code execution.

Task-1: Running Shellcode

- Executing below command to disable the address space randomization in ubuntu system.
  sudo sysctl -w kernel.randomize_va_space=0



- Compiled and executed the code **call_shellcode.c** which is launching a shell stored in the buffer of this code. The **execstack** in the command allows to execute stack in the code.
  gcc -z execstack -o call_shellcode call_shellcode.c

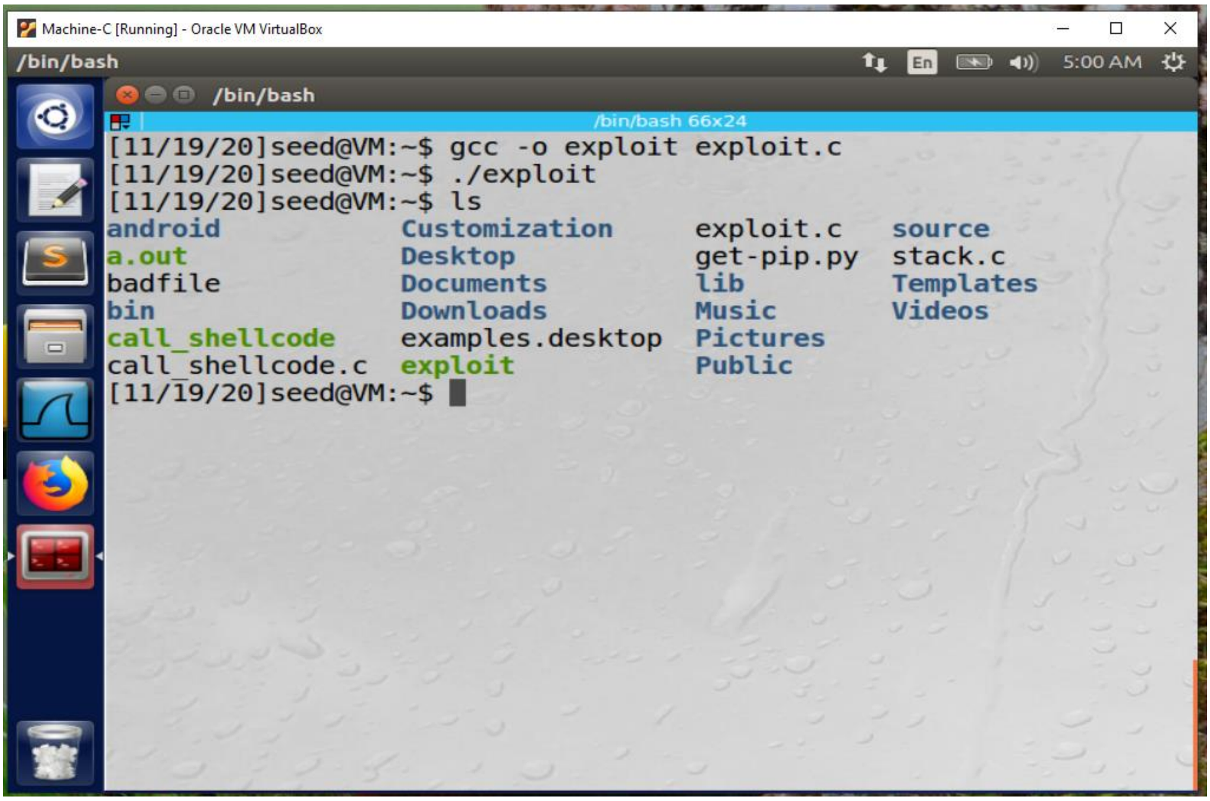


- I observed that call_shellcode file is generated after compiling and running call_shellcode.c program. Refer below snapshot:

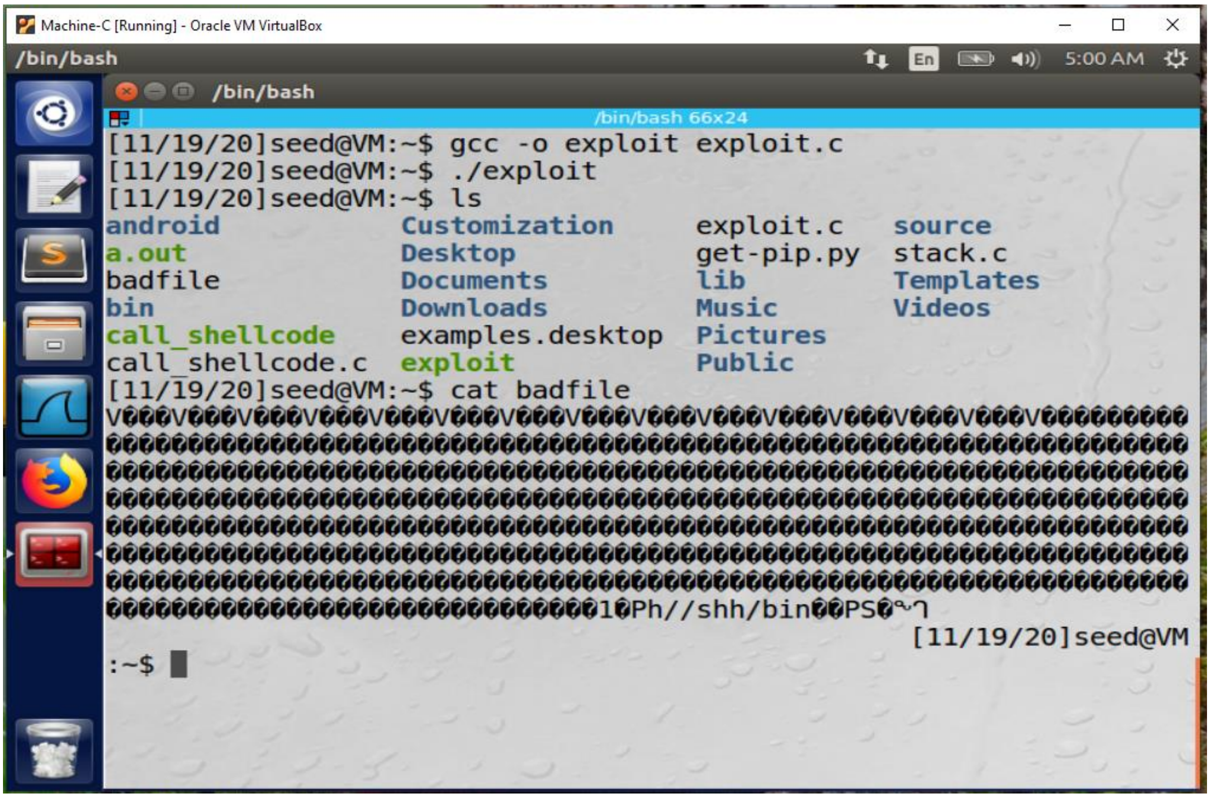
Task-2: Exploiting the Vulnerability

In the vulnerable program, a badfile with content is generated. This badfile input its content when the program executes. The maximum length of original input size is 517 bytes, and the buffer size is 24 bytes which is way too less. As given in code, strcpy() will not check the size and copy the content from badfile. Hence, it constitutes to bufferoverflow vulnerability.

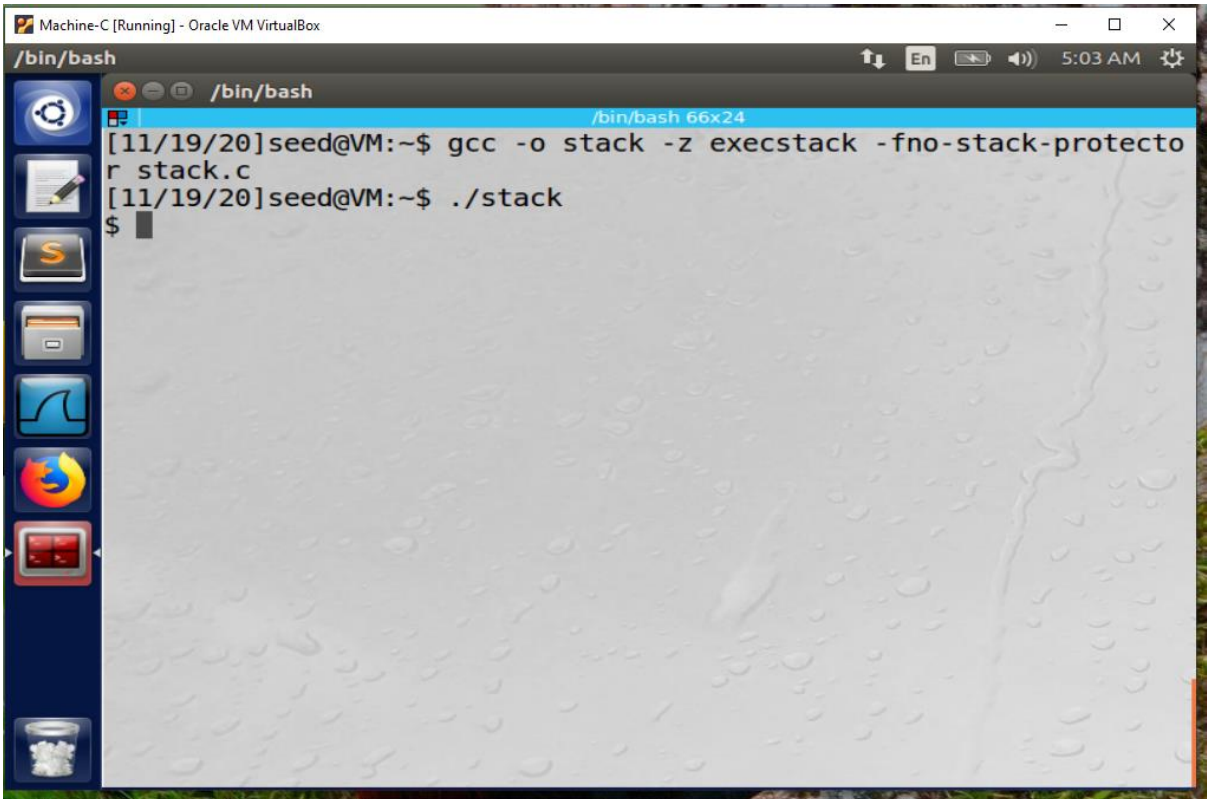
- To exploit the buffer overflow vulnerability and to gain root access, I have compiled and executed exploit.c program. This program will generate compiled file called exploit. Refer below snapshot:



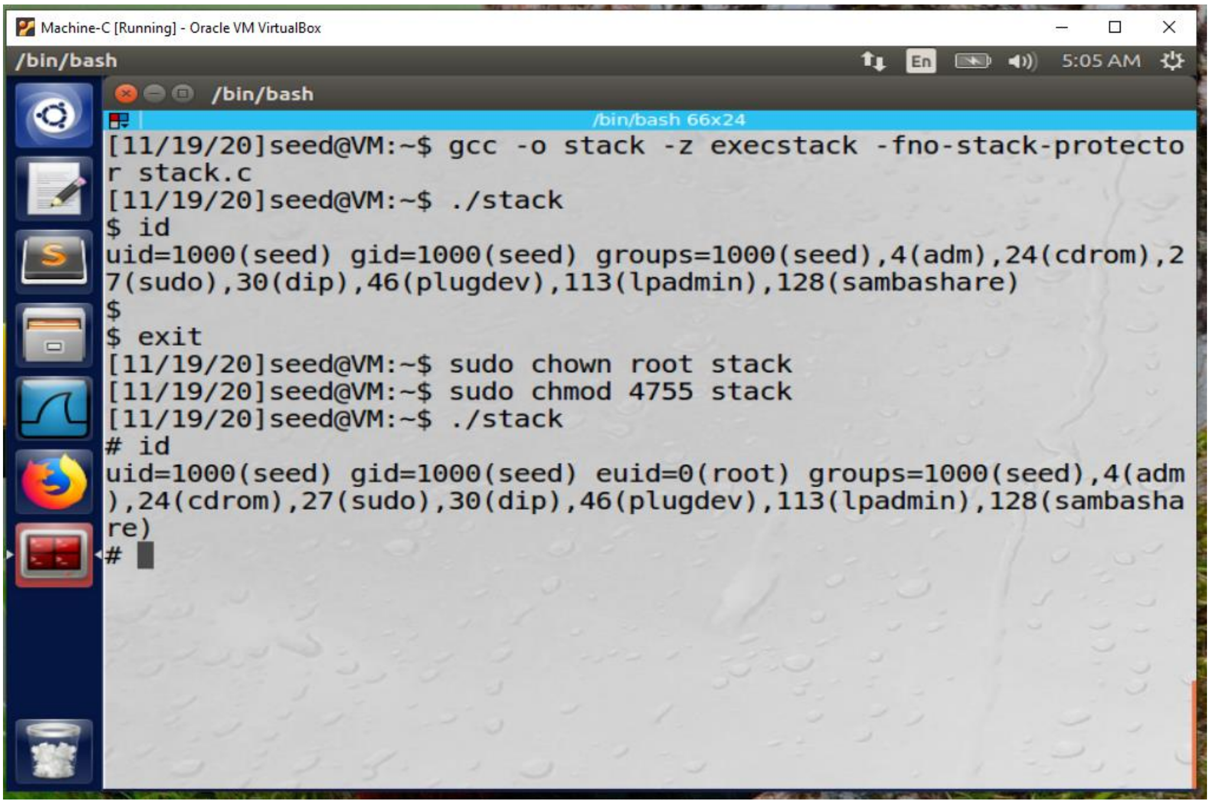
- After the execution of exploit.c program, a badfile was also created. Refer below snapshot showing badfile created and the content in it.



- The given command will turn off the StackGuard and the non-executable stack protections using the -fno-stack-protector and "-z execstack" options. Then launch the attack using vulnerable program called: ./stack

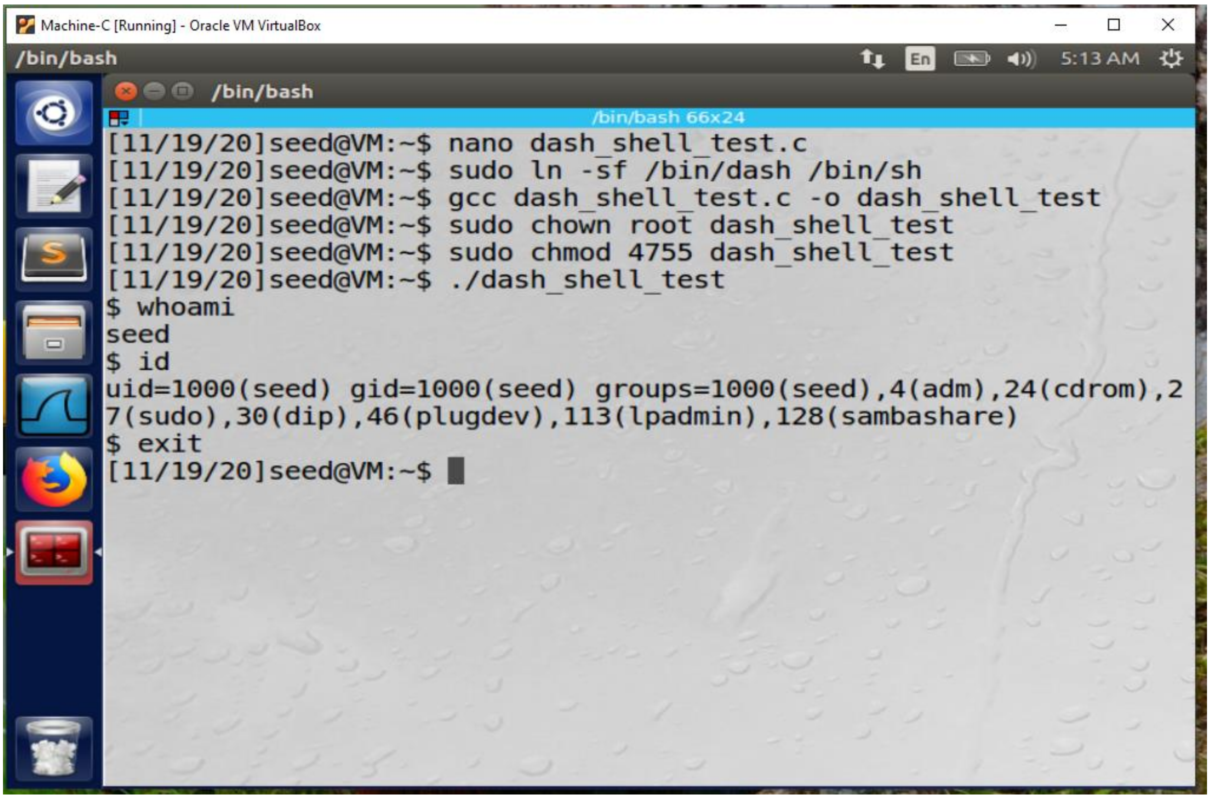


- I observed that the real user id is still "seed" but the effective user id (euid) has changed to root. Refer below snapshot:



Task 3: Defeating dash's Countermeasure

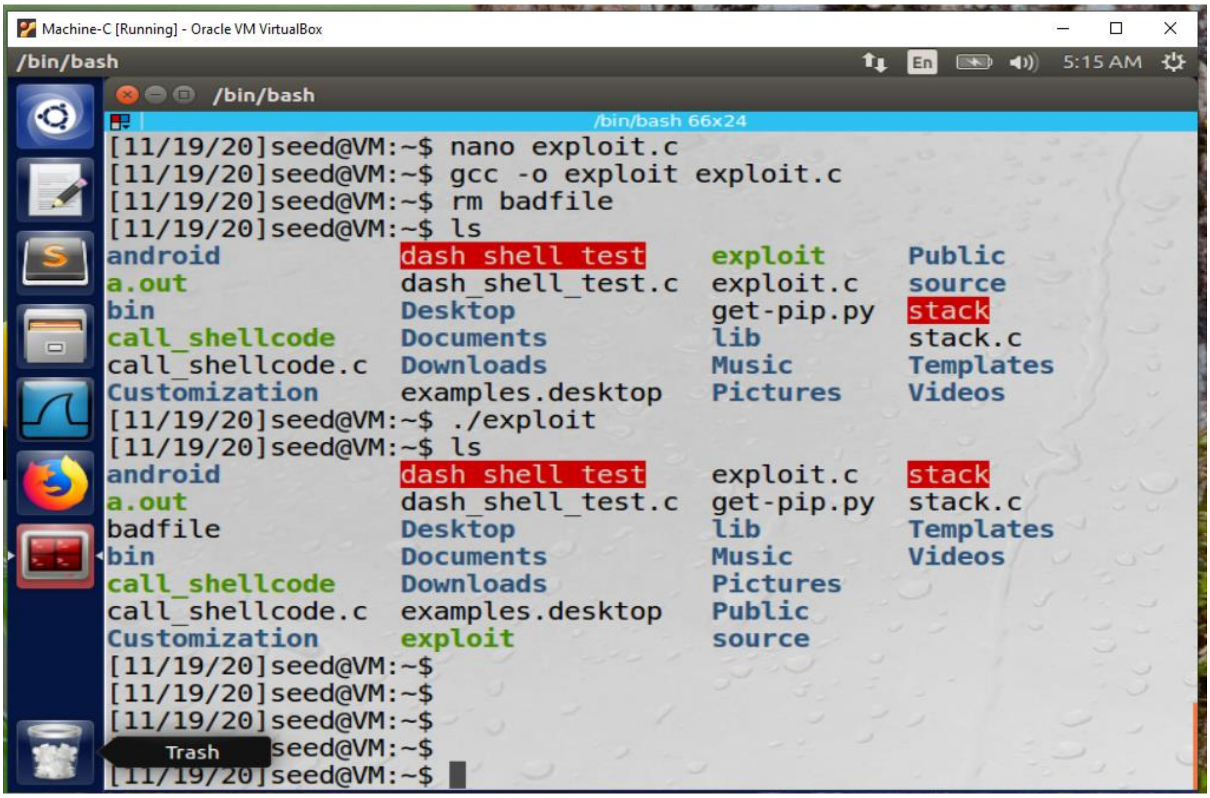
- The dash shell drops the privileges when it detects that effective UID does not equal to the real UID.
- Changing **/bin/sh** symbolic link to point back to **/bin/dash**
- The program dash_shell_test.c can be compiled and set up root-owned by using following commands:
  gcc dash_shell_test.c -o dash_shell_test
  sudo chown root dash_shell_test
  sudo chmod 4755 dash_shell_test
- Observed that after executing dash_shel_test.c program, the uid is **seed**. Refer below snapshot:



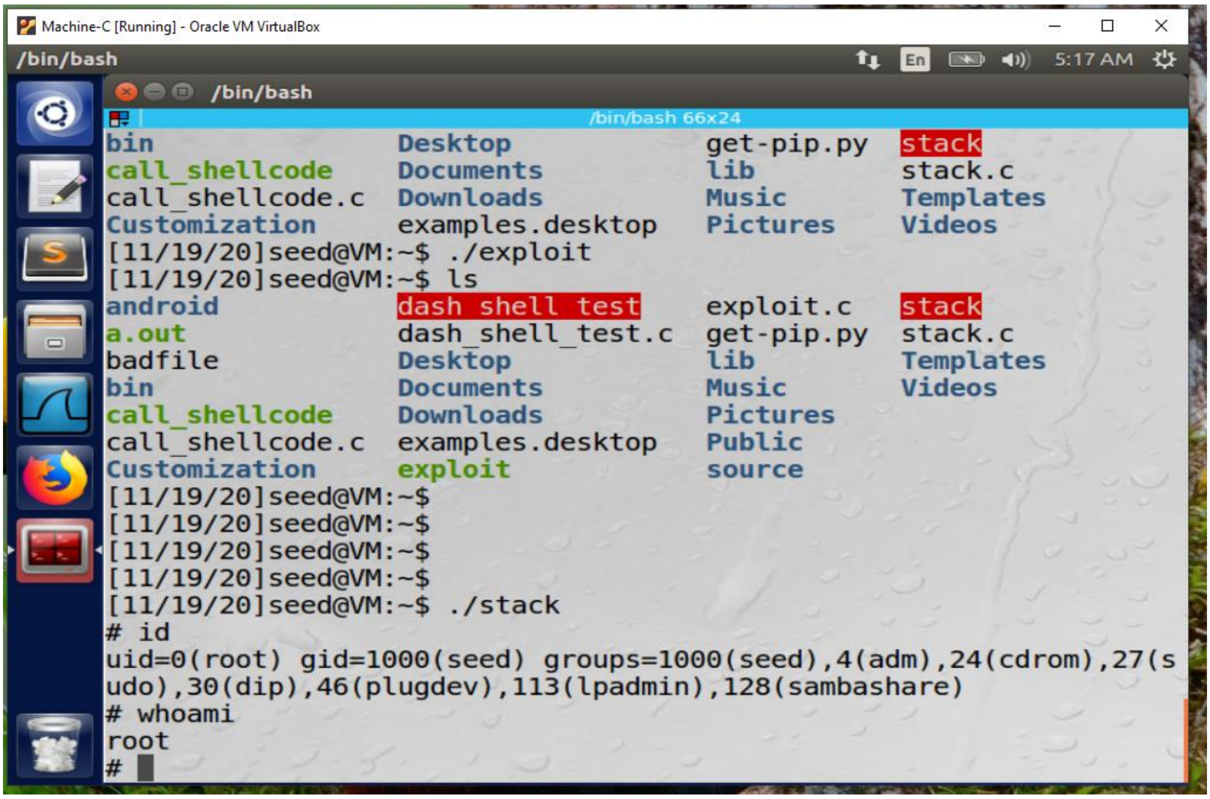
- Now removing the previous badfile generated with/bin/sh and again creating badfile with exploit.c after adding below changes in exploit.c program.

```
"\x31\xc0" /* Line 1: xorl %eax,%eax */
"\x31\xdb" /* Line 2: xorl %ebx,%ebx */
"\xb0\xd5" /* Line 3: movb $0xd5,%al */
"\xcd\x80" /* Line 4: int $0x80 */
```

- The attempt of this attack on the vulnerable program is to observe the effect when /bin/sh is linked to /bin/dash.

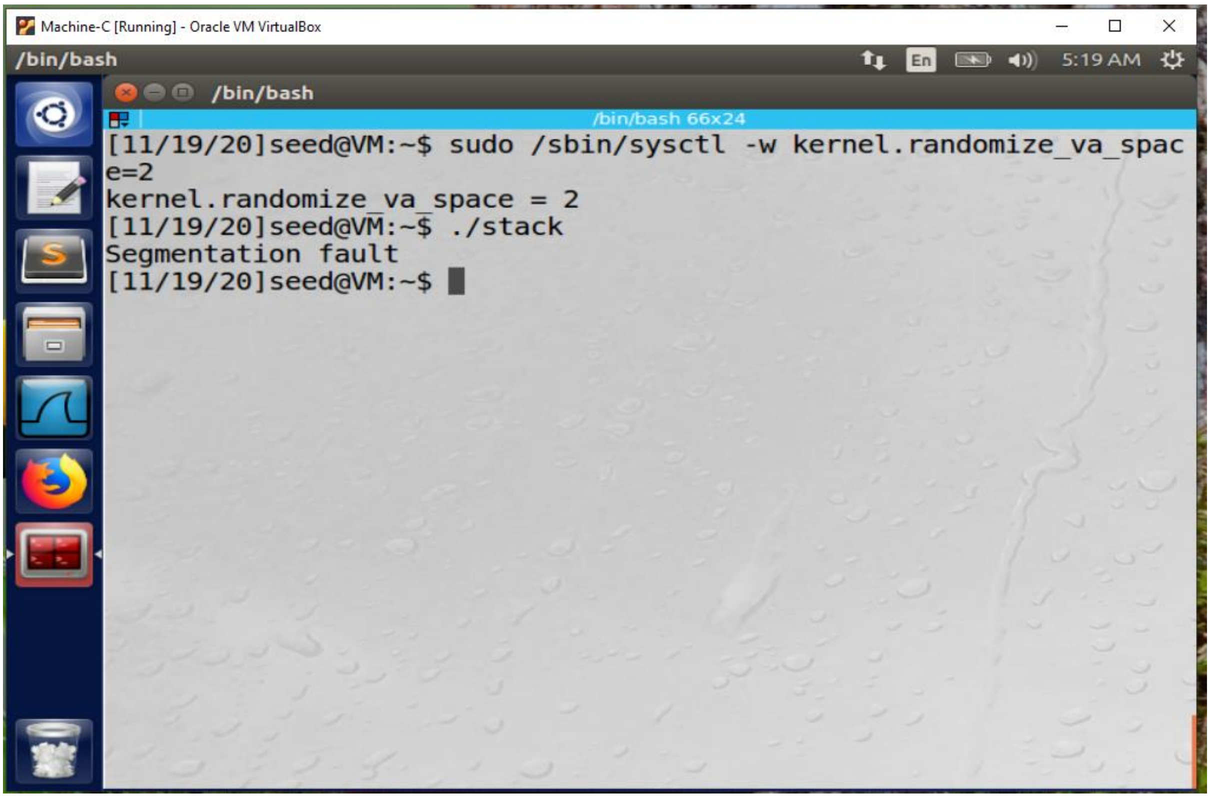


- After executing the vulnerable program ./stack, observed that uid has changed to root. Refer below snapshot:

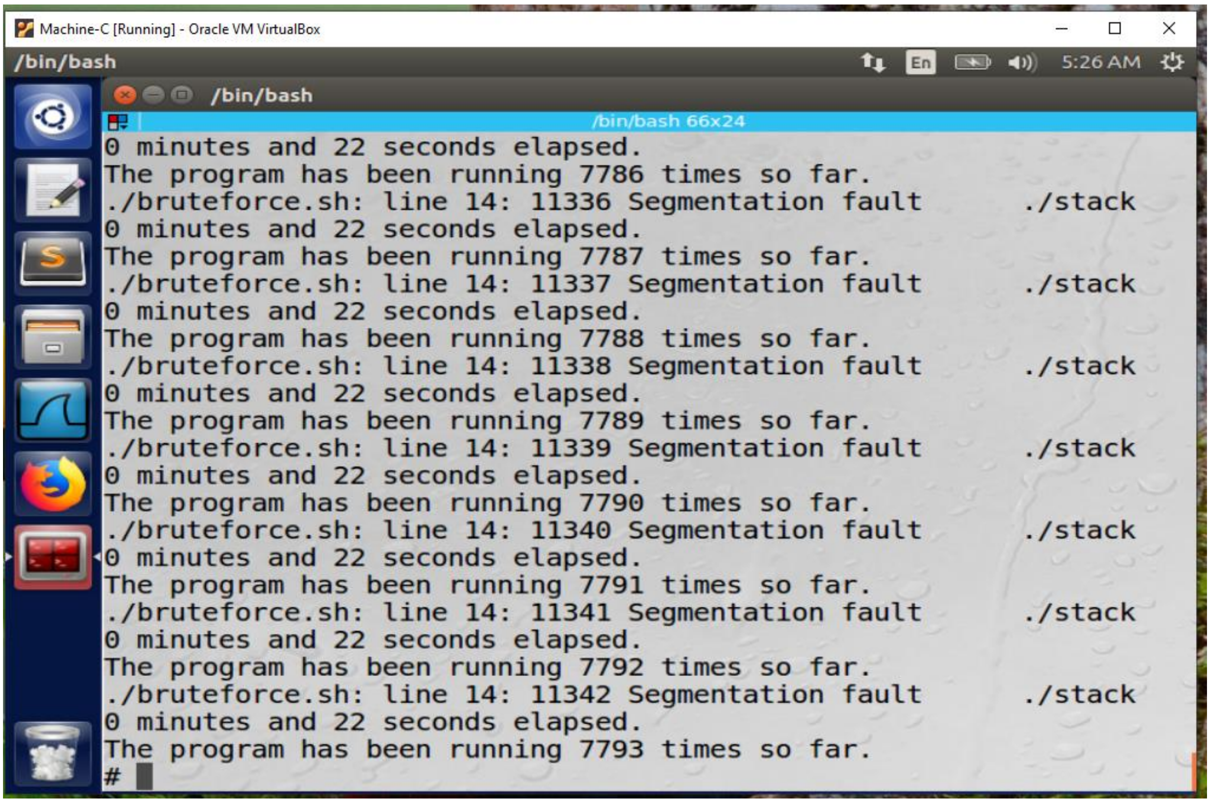


Task 4: Defeating Address Randomization

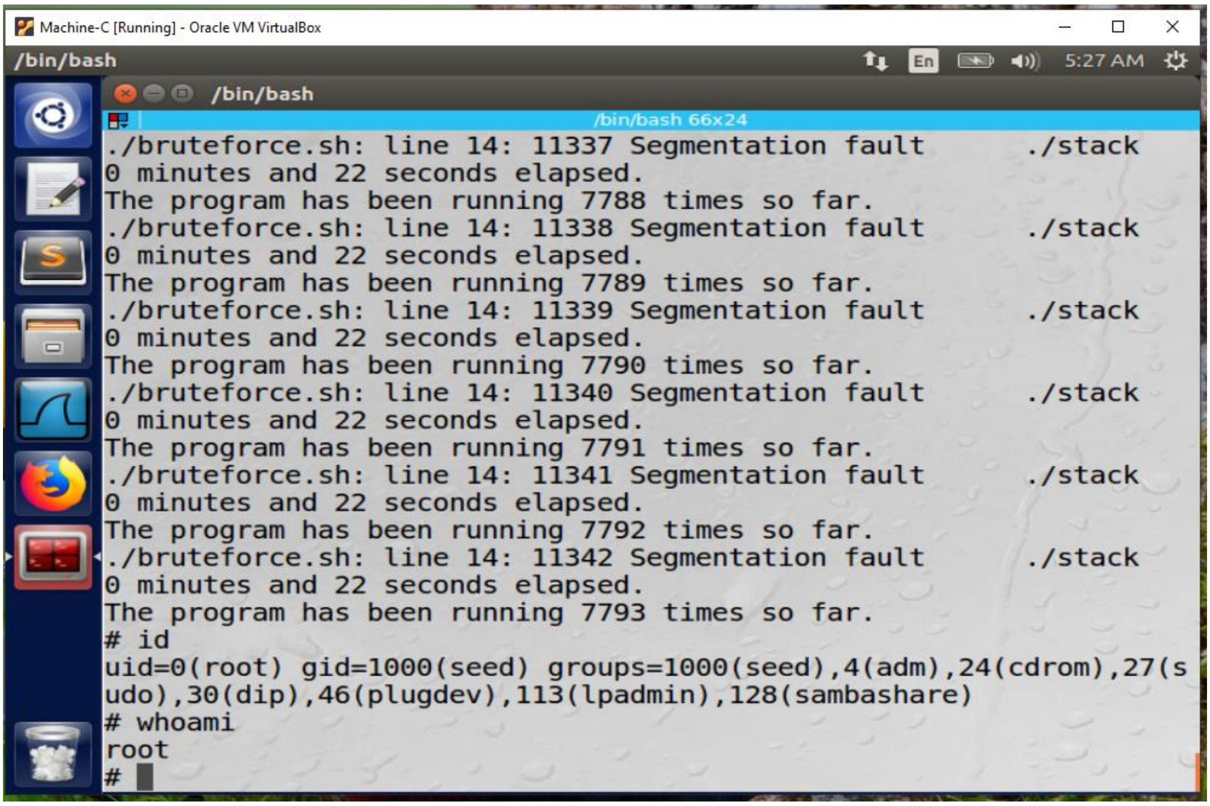
- Executing below command to defeat the address randomization countermeasure on 32-bit VM
  sudo /sbin/sysctl -w kernel.randomize_va_space=2
- After executing vulnerable program, observed **Segmentation fault**



- Now repeating the vulnerable program (./stack) by using brute force approach. It will run until address in the badfile is not correct. It will run in infinite loop and the attack will succeed once the script stop.



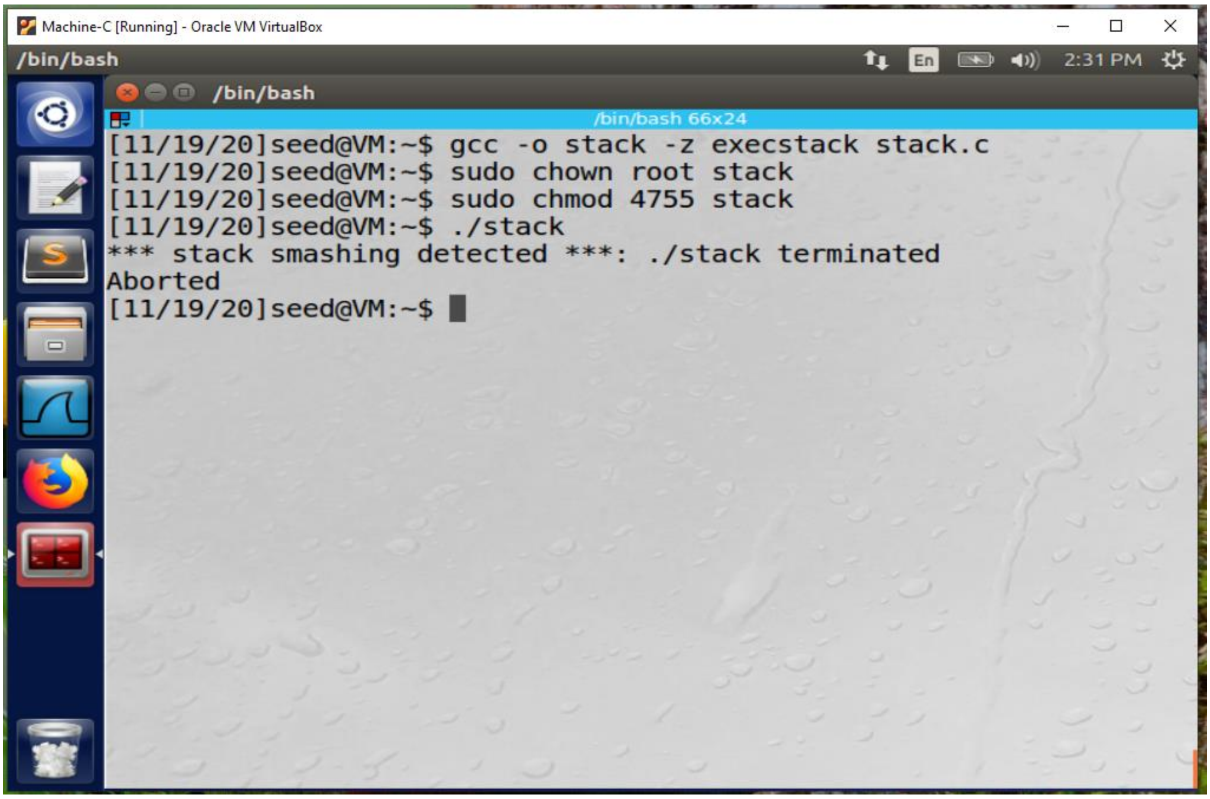
- After the successful brute force attack approach, observed that uid has changed to root.



Task 5: Turn on the StackGuard Protection

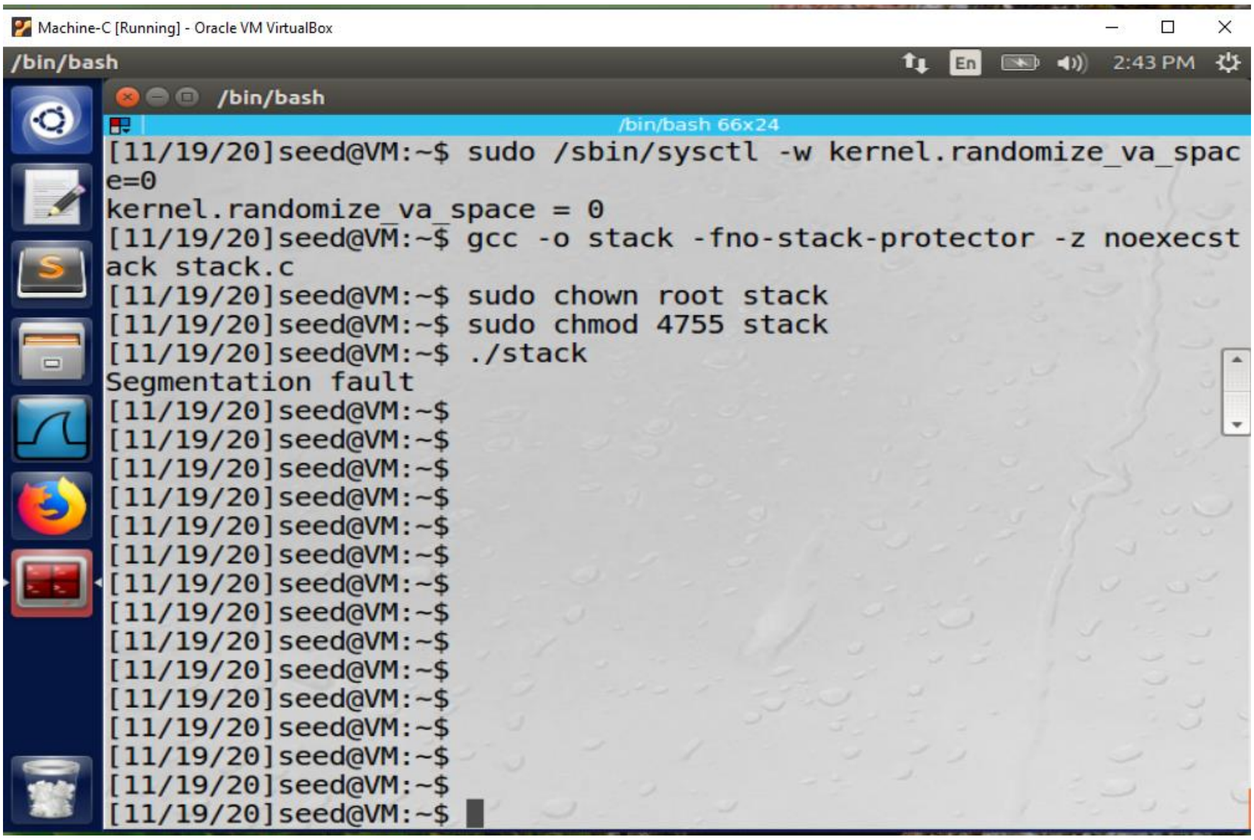
- Turn off the address randomization and compile the program without the -fno-stack-protector. Changing the permissions to root as shown in snapshot.

- After running the vulnerable program, observed that **stack terminated** or the program has been aborted.



Task 6: Turn on the Non-executable Stack Protection

- Now turn on the nonexecutable stack protection using command:
  gcc -o stack -fno-stack-protector -z noexecstack stack.c
- The non-executable stack only run shellcode on the stack, but it does not prevent buffer-overflow attacks.
- Hence, I observed segmentation fault after running the vulnerable program.
  Refer below snapshot:


Machine-C [Running] - Oracle VM VirtualBox
/bin/bash
En
2:43 PM
/bin/bash
/bin/bash 66x24
[11/19/20]seed@VM:~$ sudo /sbin/sysctl -w kernel.randomize_va_spac
e=0
kernel.randomize_va_space = 0
[11/19/20]seed@VM:~$ gcc -o stack -fno-stack-protector -z noexecst
ack stack.c
[11/19/20]seed@VM:~$ sudo chown root stack
[11/19/20]seed@VM:~$ sudo chmod 4755 stack
[11/19/20]seed@VM:~$ ./stack
Segmentation fault
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$
[11/19/20]seed@VM:~$