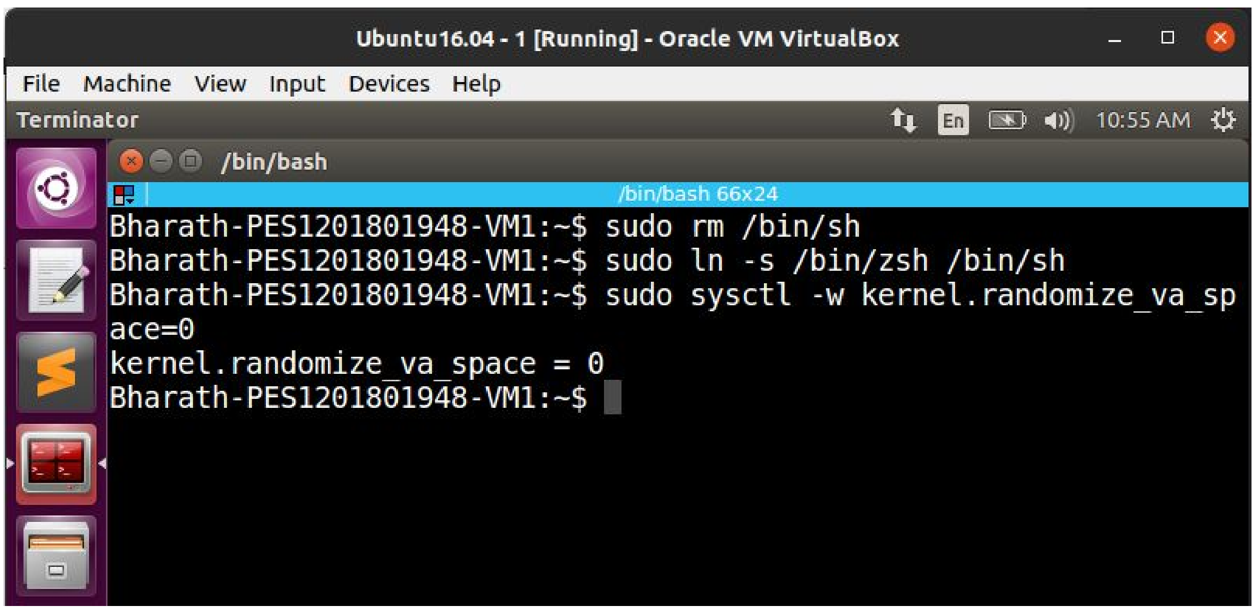
Information Security
Lab 4
Return - to - Libc attack

PES1201801948
Bharath S Bhambore
Section H

Task 1: Address Space Randomization

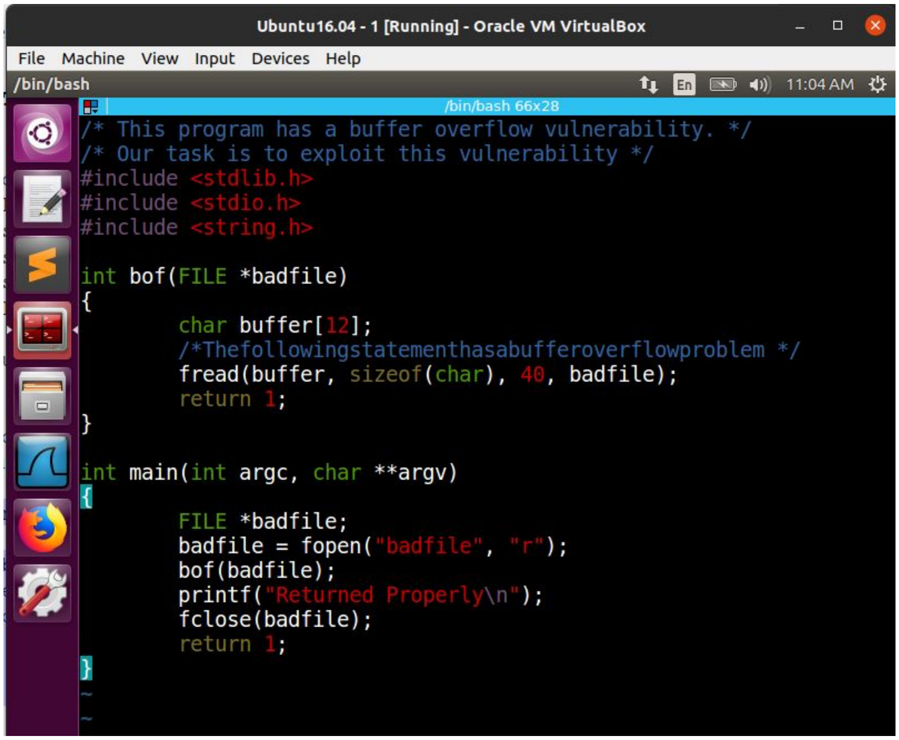
Initial Setup :



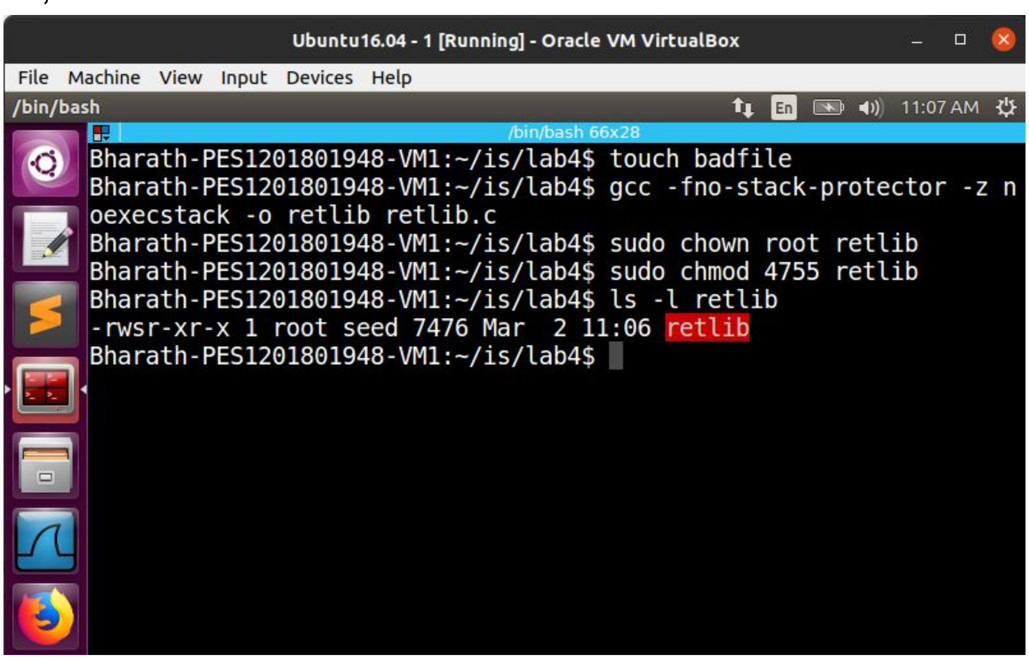
Point /bin/sh to /bin/zsh shell to overcome the countermeasure in dash shell drops the privileges of a set uid program, therefore we will never be able to pop a root shell.

And also disabled the address space randomization, so that it is easier to guess the starting stack address

Retlib.c : The vulnerable program contains a buffer overflow vulnerability in the function bof().

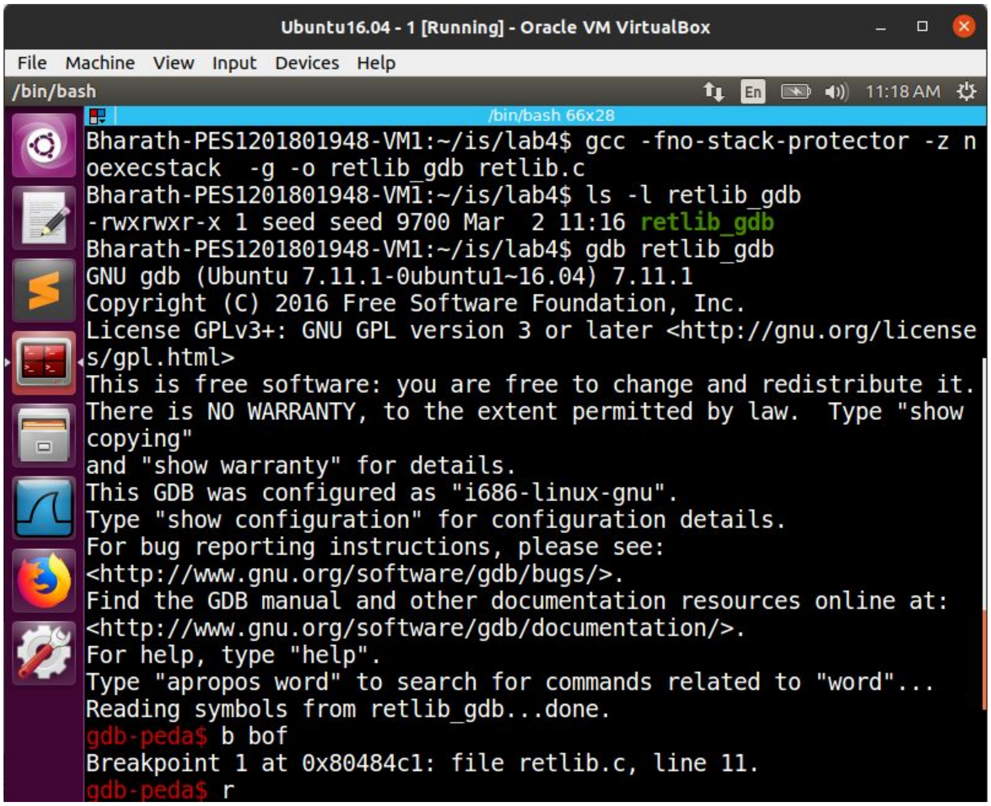


A badfile is created, the vulnerable file is compiled while the stack guard is off, but the stack is non-executable.

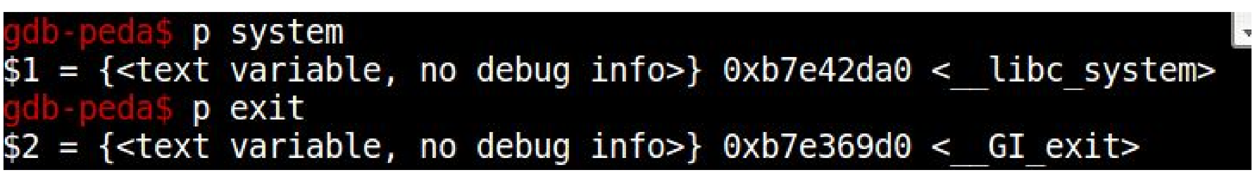


The executable is then made into a set uid program, so that when our exploit works, we will be able to generate a root shell.

Task 2: Finding out the address of the lib function

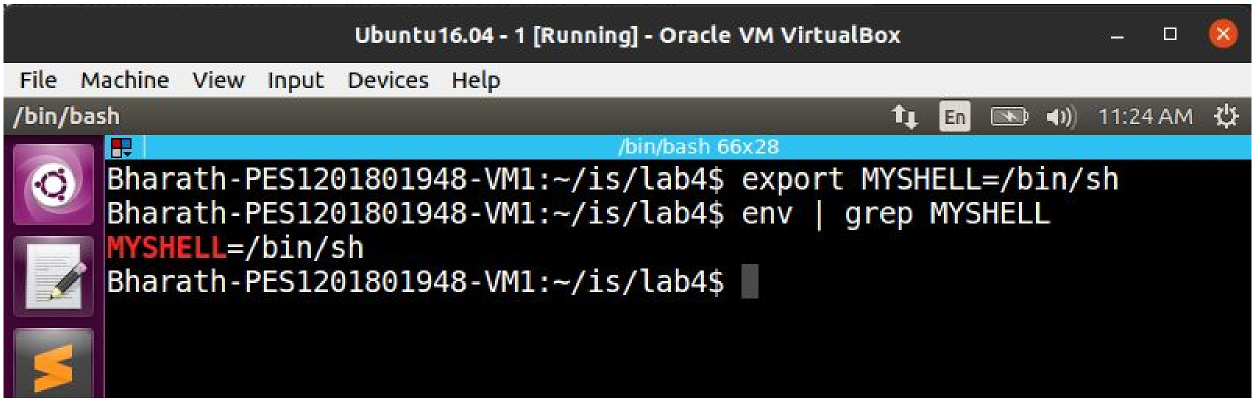


Re-compiling the program with the debugging option enabled in gcc, we run the executable through gdb, setting a breakpoint at the function bof.

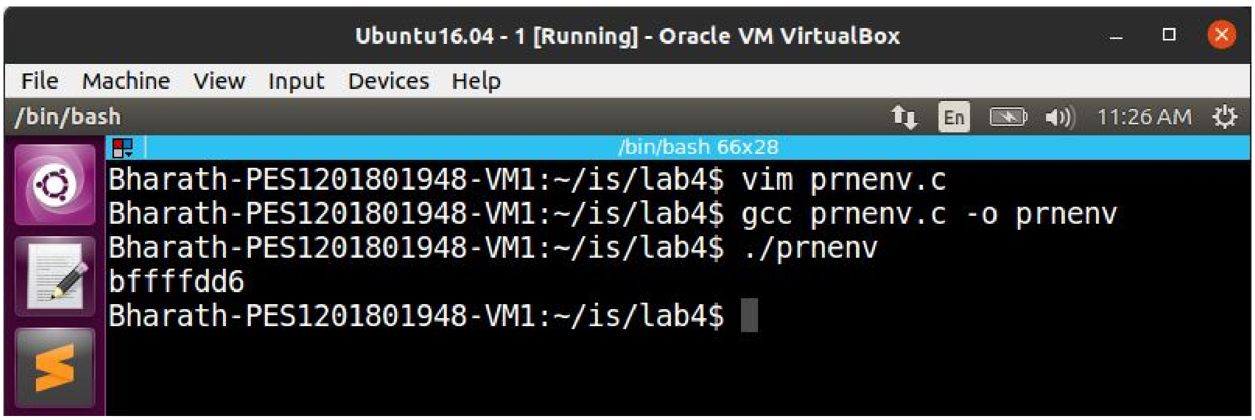


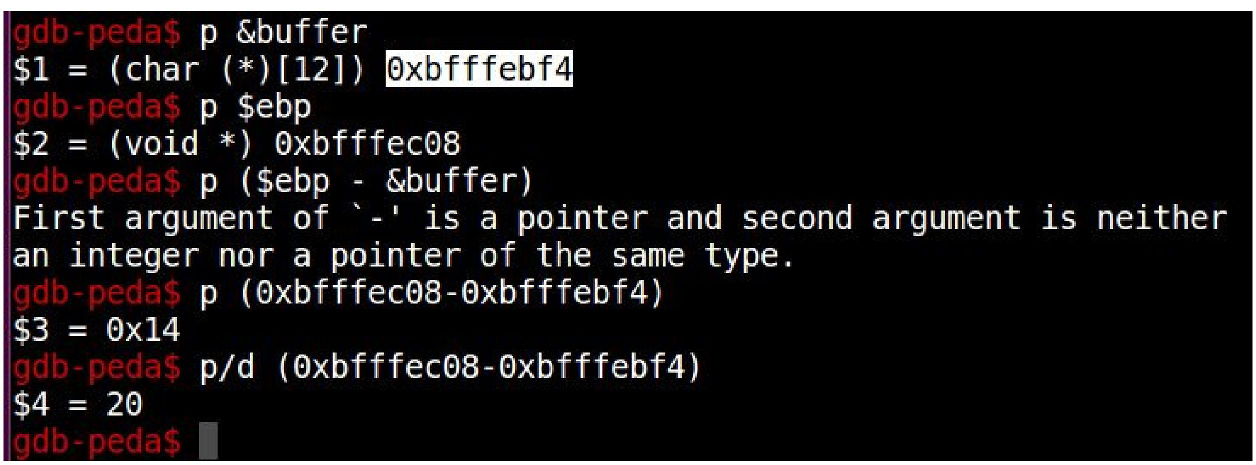
Therefore, we can print out the address of the system(), as well as the exit() function.

Task 3 : Putting the shell string in the memory

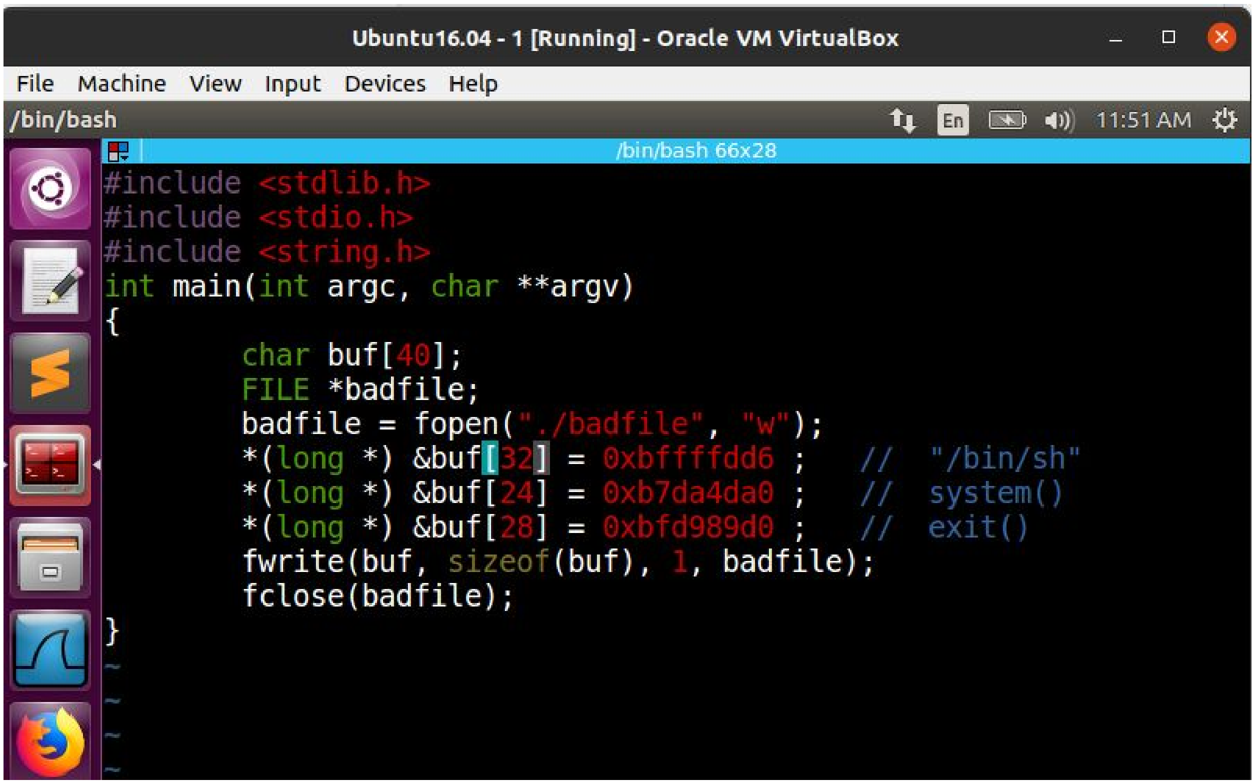


Exporting the shell variable which points to dash, in-turn it will point to zsh MYSHELL is passed as an environment variable to the C program, which is stored in the stack, therefore we can find out the address of it.

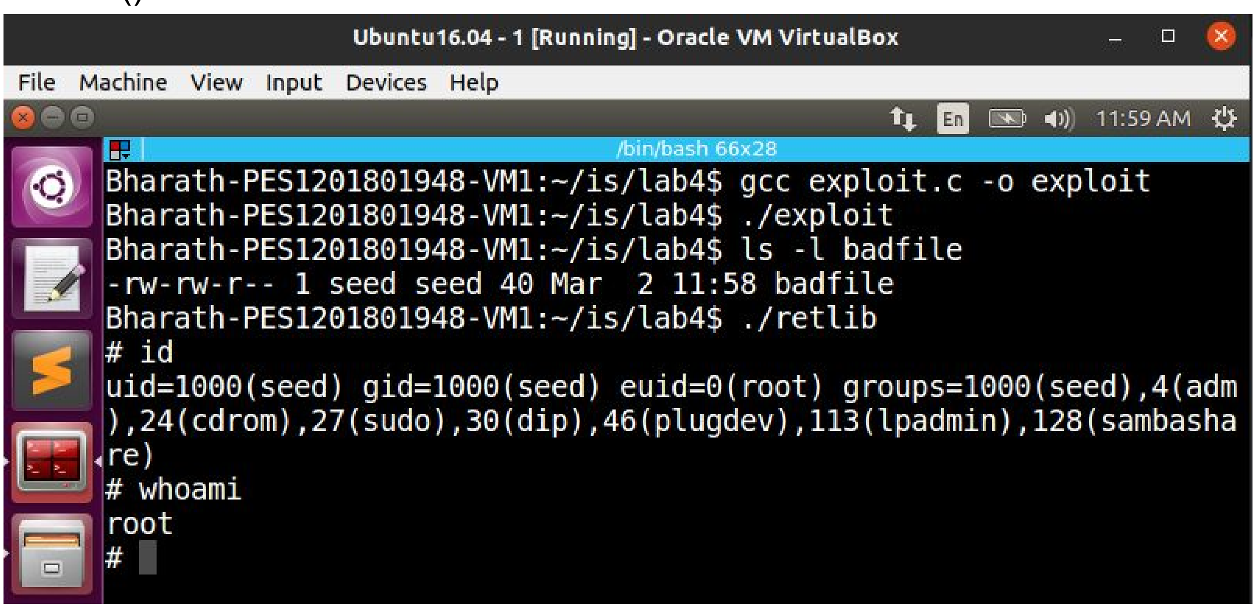




Using the ebp address, we can find out where each of the commands are placed appropriately in the buffer and point that to the address of each command we found before.

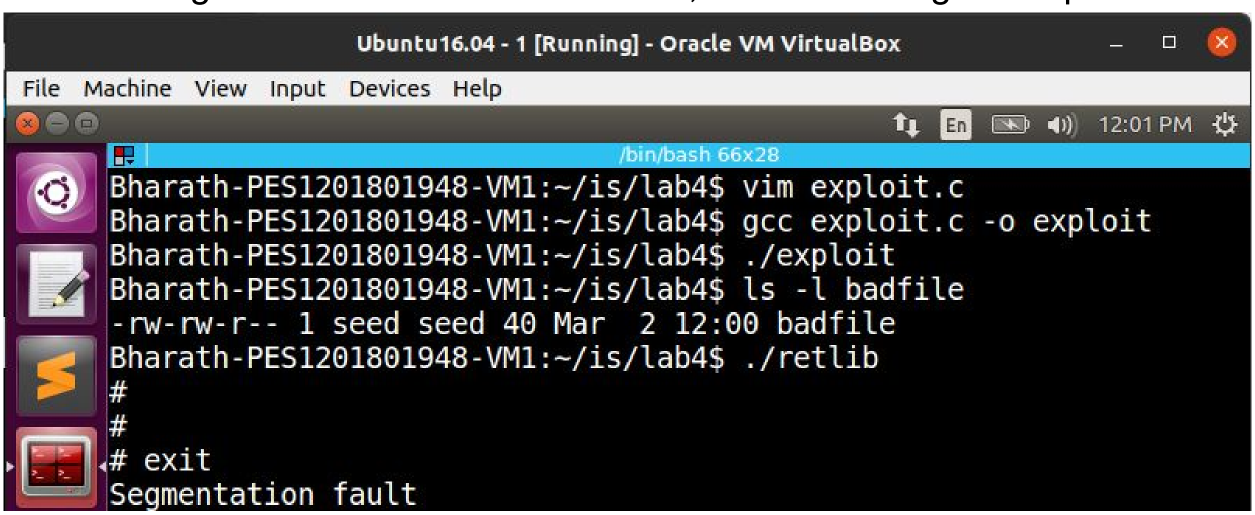


Ebp + 4 is treated as system() return address, ebp + 12 is the address at which /bin/sh is placed in the buffer. Subsequently ebp+8 is the address of the exit() function address in libc.



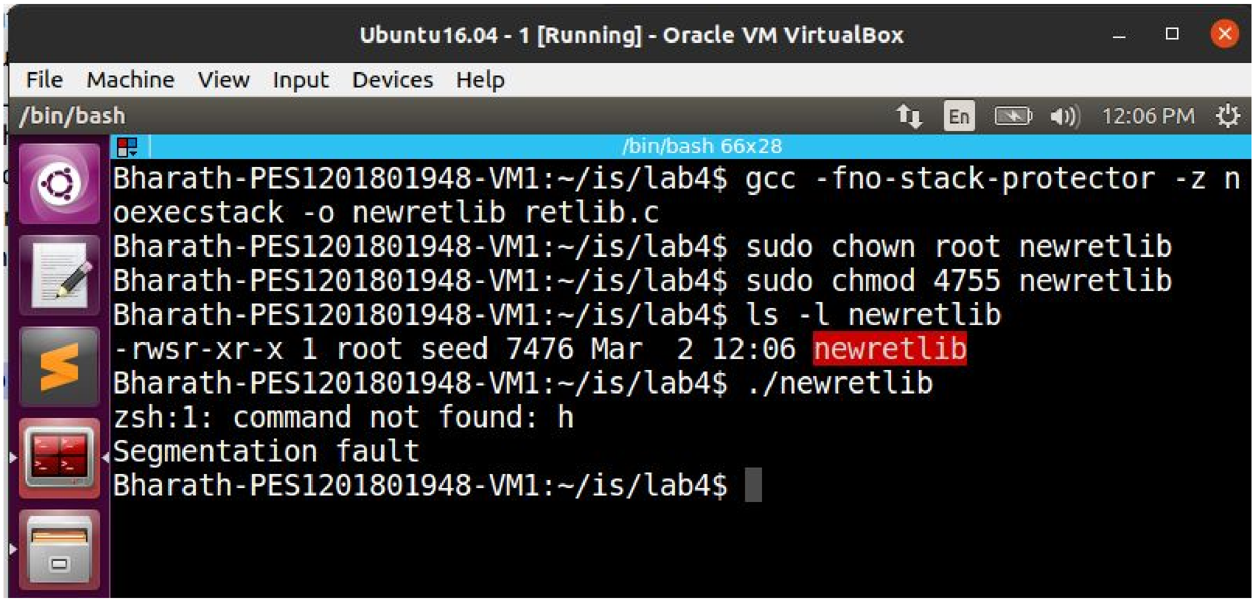
Running the exploit, we cann that we successfully executed system("/bin/sh"). Thereby popping a root shell since the executable was a SetUID program.

Commenting out the exit function address, and re-running the exploit.

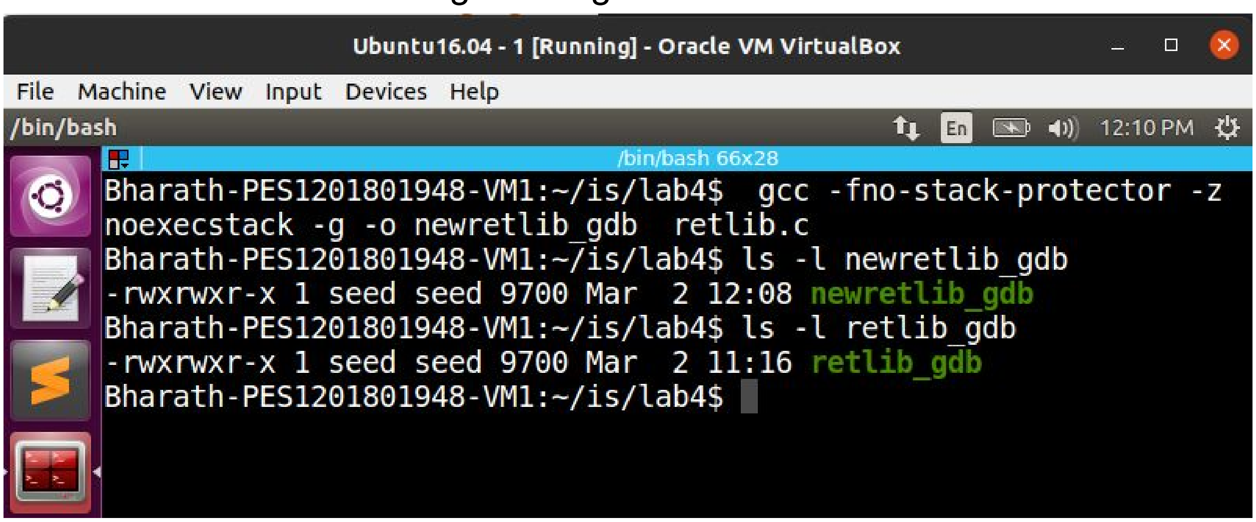


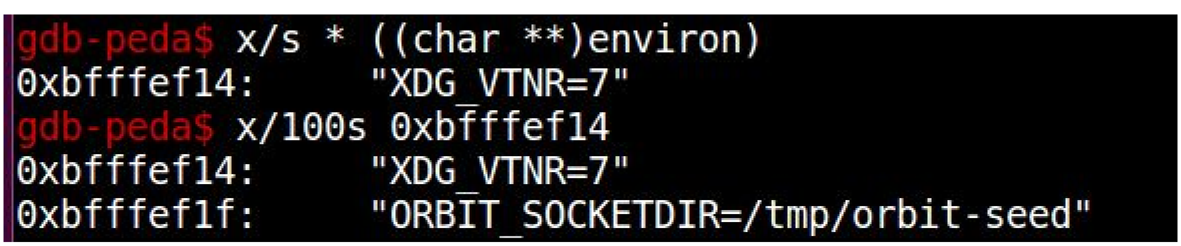
We get a seg fault, after exiting the shell.

Task 4: Changing length of the file name



Recompiling the vulnerable file, but this time with a different executable name, but the code doesnt give us the root shell since the address of /bin/sh varies with the change in length of filename.

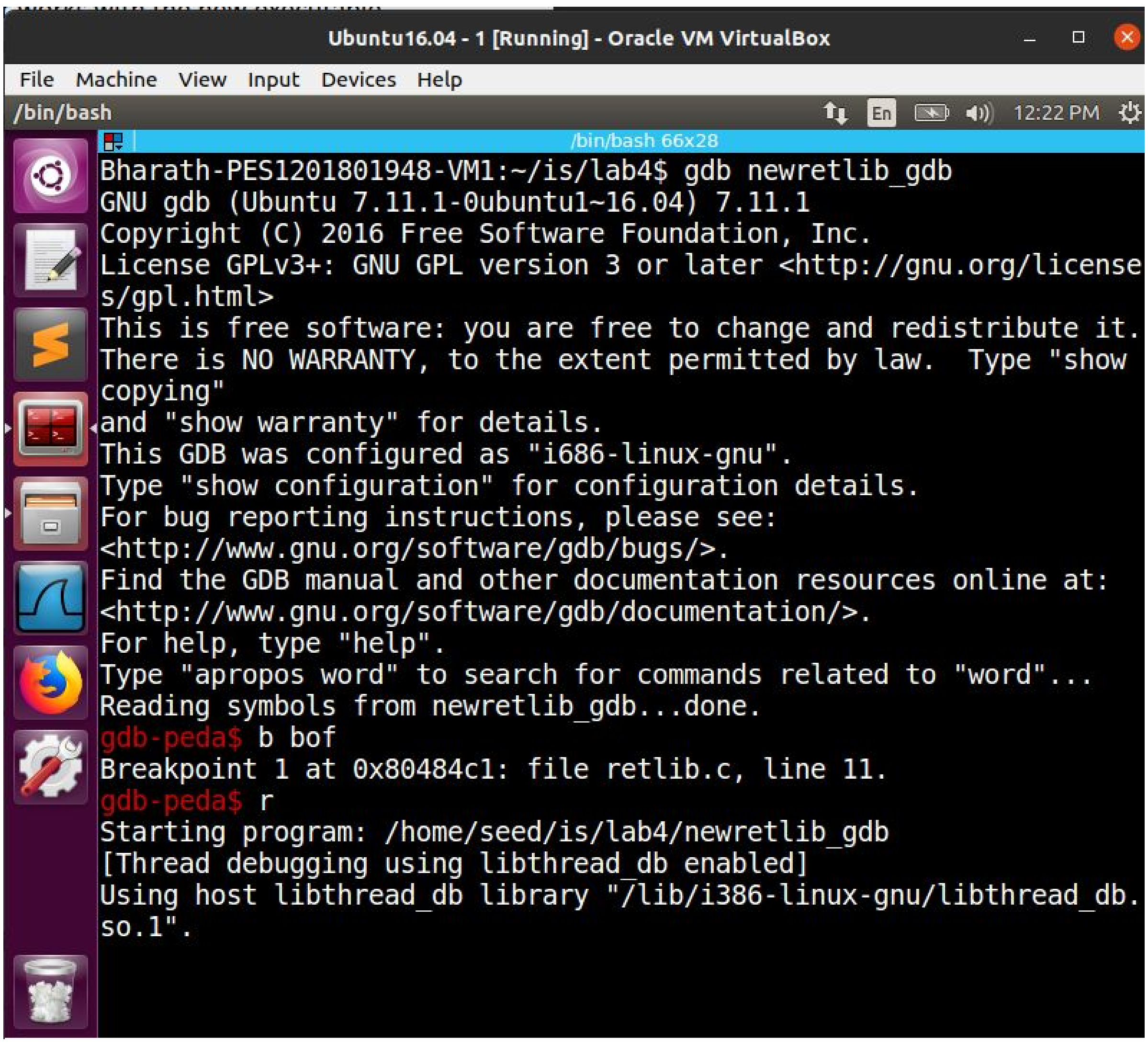




```
0xbfffff67:      "XDG_CURRENT_DESKTOP=Unity"
0xbfffff81:      "LESSCLOSE=/usr/bin/lesspipe %s %s"
0xbfffffa3:      "COLORTERM=gnome-terminal"
0xbfffffbc:      "XAUTHORITY=/home/seed/.Xauthority"
0xbfffffde:      "/home/seed/is/lab4/retlib_gdb"
0xbffffffc:      ""
```

```
0xbfffdc4:      "COMPIZ_BIN_PATH=/usr/bin/"
0xbffffdde:      "MYSHELL=/bin/sh"
0xbffffdee:      "QT4_IM_MODULE=xim"
0xbffffe00:      "XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:
```

We can see the environment variables are loaded in the bottom of the stack




```
Bharath-PES1201801948-VM1:~/is/lab4$ gdb newretlib_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/license
s/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 newretlib_gdb...done.
gdb-peda$ b bof
Breakpoint 1 at 0x80484c1: file retlib.c, line 11.
gdb-peda$ r
Starting program: /home/seed/is/lab4/newretlib_gdb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.
so.1".
```

```
gdb-peda$ x/s * ((char **)environ)
0xbfffef11:     "XDG_VTNR=7"
gdb-peda$ x/100s 0xbfffef11
0xbfffef11:     "XDG_VTNR=7"
0xbfffef1c:     "ORBIT_SOCKETDIR=/tmp/orbit-seed"
0xbfffef3c:     "XDG_SESSION_ID=c1"
```

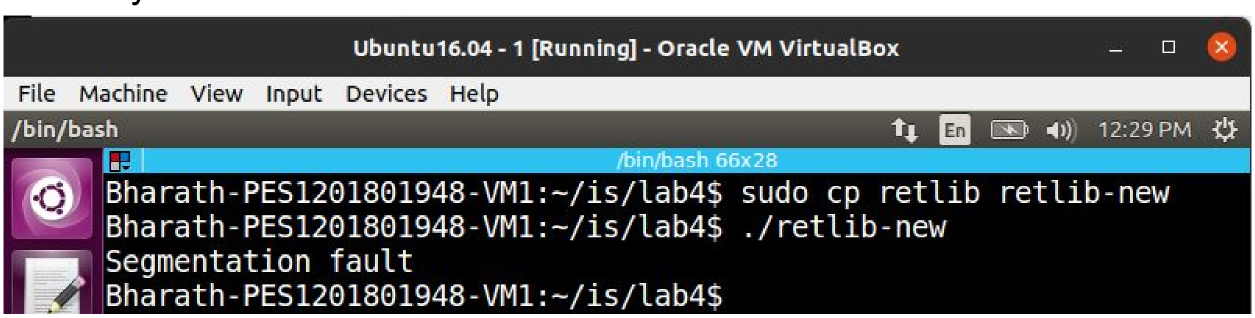
```
0xbffffdc1:     "COMPIZ_BIN_PATH=/usr/bin/"
0xbffffddb:     "MYSHELL=/bin/sh"
0xbffffdeb:     "QT4_IM_MODULE=xim"
```

```
0xbfffffa0:     "COLORTERM=gnome-terminal"
0xbfffffb9:     "XAUTHORITY=/home/seed/.Xauthority"
0xbfffffdb:     "/home/seed/is/lab4/newretlib_gdb"
0xbffffffc:     ""
```

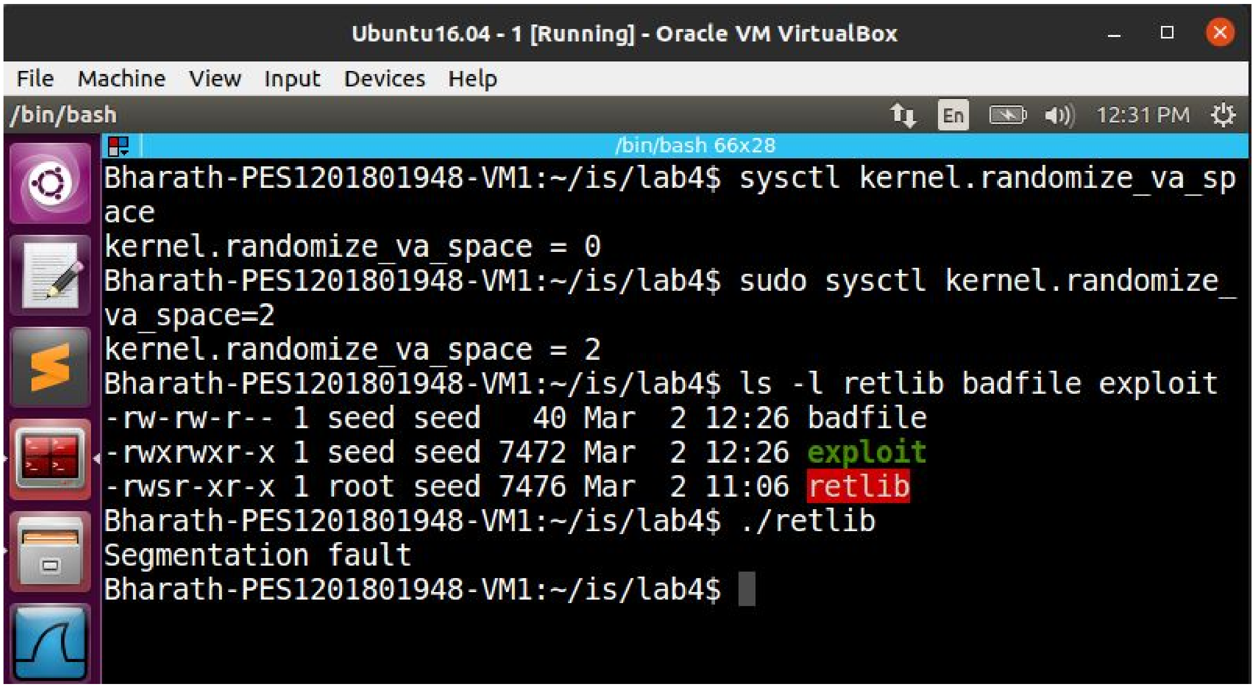
Running gdb for the new executable, we can see that the address of our environment variable MYSHELL is changed. Hence why our attack was not successful.

```
Bharath-PES1201801948-VM1:~/is/lab4$ ./retlib
# whoami
root
# exit
```

Showing our old executable is running, but even copying its contents into a new file, just with a longer filename, the exploit fails since the address internally switches.

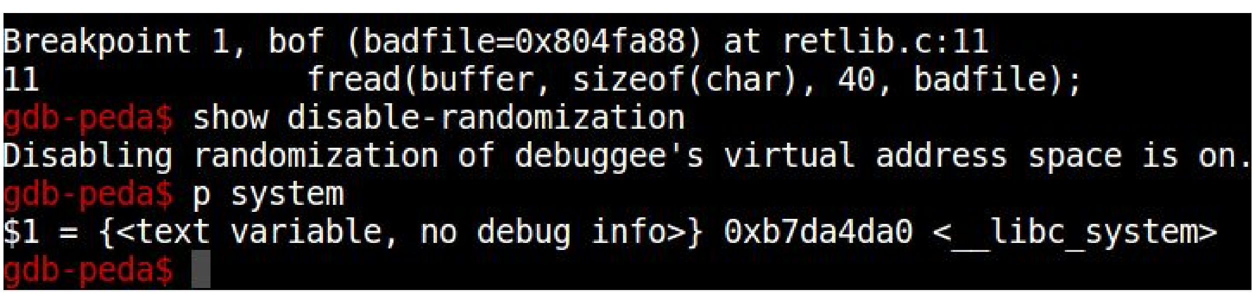


Task 5: Address Randomization



Enabling the address randomization, and running our executable doesnt work anymore since the addresses have changed.

Entering into gdb in quiet mode.



Disable randomization is on, we get the same address, but turning it off, the address of system() changes.

```
gdb-peda$ b main
Breakpoint 1 at 0x80484ec: file retlib.c, line 18.
gdb-peda$ r
Starting program: /home/seed/is/lab4/retlib_gdb
```

```
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is on.
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7da4da0 <__libc_system>
gdb-peda$
```

Since the address randomization is on, running our code to print the address of the environment variable MYSHELL, we can see that the address is randomized for every call. This shows why our exploit doesnt work and we arent able to pop a root shell.