



中国科学技术大学

《网络安全》

实验报告

姓名：吴佳明

学号：SA19221027

年级：硕士研究生一年级

本实验基于 ubuntu18 64bit

首先，写一个使用 `execve` 调用 `/bin/ls` 的 c 语言程序。

```
get_ls_machir  get_ls_machinecode2.c  shellcode_from_net.c  get_ls_assembly.c  x  x
1  #include <stdio.h>
2  #include <unistd.h>
3
4  void call_ls(){
5      char* argv[2];
6      argv[0] = "/bin/ls";
7      argv[1] = NULL;
8
9      execve(argv[0],argv,NULL);
10 }
11
12 int main(){
13     call_ls();
14     return 0;
15 }
```

然后使用 `gcc` 进行编译，并用 `gdb` 进行调试。

先在 `call_ls` 函数入口处设置断点，反汇编。

```
Breakpoint 2, call_ls () at get_ls_assembly.c:4
warning: Source file is more recent than executable.
4      void call_ls(){
(gdb) disas
Dump of assembler code for function call_ls:
=> 0x000055555555464a <+0>:      push    %rbp
    0x000055555555464b <+1>:      mov     %rsp,%rbp
    0x000055555555464e <+4>:      sub     $0x10,%rsp
    0x0000555555554652 <+8>:      lea     0xc(%rip),%rax          # 0x555555
5554724
    0x0000555555554659 <+15>:     mov     %rax,-0x10(%rbp)
    0x000055555555465d <+19>:     movq    $0x0,-0x8(%rbp)
    0x0000555555554665 <+27>:     mov     -0x10(%rbp),%rax
    0x0000555555554669 <+31>:     lea     -0x10(%rbp),%rcx
    0x000055555555466d <+35>:     mov     $0x0,%edx
    0x0000555555554672 <+40>:     mov     %rcx,%rsi
    0x0000555555554675 <+43>:     mov     %rax,%rdi
    0x0000555555554678 <+46>:     callq   0x555555554520 <execve@plt>
    0x000055555555467d <+51>:     nop
    0x000055555555467e <+52>:     leaveq
    0x000055555555467f <+53>:     retq
End of assembler dump.
(gdb) |
```

接着在 `execve` 函数入口处设置断点，反汇编。

```

(gdb) c
Continuing.

Breakpoint 1, execve () at ../sysdeps/unix/syscall-template.S:78
78      ../sysdeps/unix/syscall-template.S: No such file or directory.
(gdb) disas
Dump of assembler code for function execve:
=> 0x00007ffff7ac8e30 <+0>:      mov     $0x3b,%eax
0x00007ffff7ac8e35 <+5>:      syscall
0x00007ffff7ac8e37 <+7>:      cmp     $0xffffffffffff001,%rax
0x00007ffff7ac8e3d <+13>:     jae     0x7ffff7ac8e40 <execve+16>
0x00007ffff7ac8e3f <+15>:     retq
0x00007ffff7ac8e40 <+16>:     mov     0x306021(%rip),%rcx      # 0x7ffff7dcee68
0x00007ffff7ac8e47 <+23>:     neg     %eax
0x00007ffff7ac8e49 <+25>:     mov     %eax,%fs:(%rcx)
0x00007ffff7ac8e4c <+28>:     or      $0xffffffffffffffff,%rax
0x00007ffff7ac8e50 <+32>:     retq
End of assembler dump.
(gdb)

```

可以看到，将寄存器 `eax` 的值设置为 `0x3b` 后，进行系统调用 `syscall`，此时查看寄存器的值。

```

(gdb) i r a
rax      0x555555554724      93824992233252
rbx      0x0                0
rcx      0x7fffffffdee0     140737488346848
rdx      0x0                0
rsi      0x7fffffffdee0     140737488346848
rdi      0x555555554724     93824992233252
rbp      0x7fffffffdef0     0x7fffffffdef0
rsp      0x7fffffffded8     0x7fffffffded8
r8       0x7ffff7dd0d80     140737351847296
r9       0x7ffff7dd0d80     140737351847296
r10      0x0                0

```

```

(gdb) x/s $rdi
0x555555554724: "/bin/ls"

```

可以看到，此时，寄存器 `rdx` 内容为 `0`，寄存器 `rdi` 指向字符串 `"/bin/ls"`。

由此，可以写出如下汇编代码。

```
get_ls_machinecode.c x  get_ls_machinecode2.c x
1  #include <stdio.h>
2
3  int main(){
4      asm (
5          "mov $0x736c2f6e69622f,%rax;"
6          "push %rax;"
7          "push %rsp;"
8          "pop %rdi;"
9
10         "xor %rdx,%rdx;"
11
12         "mov $0x3b,%rax;"
13         "syscall;");
14     return 0;
15 }
```

使用 gcc 进行编译，使用 objdump 反汇编。

```
000000000000005fa <main>:
5fa: 55                push    %rbp
5fb: 48 89 e5          mov     %rsp,%rbp
5fe: 48 b8 2f 62 69 6e 2f movabs  $0x736c2f6e69622f,%rax
605: 6c 73 00
608: 50                push    %rax
609: 54                push    %rsp
60a: 5f                pop     %rdi
60b: 48 31 d2          xor     %rdx,%rdx
60e: 48 c7 c0 3b 00 00 00 mov     $0x3b,%rax
615: 0f 05            syscall
617: b8 00 00 00 00 00 mov     $0x0,%eax
61c: 5d                pop     %rbp
61d: c3                retq
61e: 66 90            xchg    %ax,%ax
```

于是，写出如下文件。

```
get_ls_machinecode.c x get_ls_machinecode2.c x get_ls_assembly.c x le_from_net.c x
1  #include <stdio.h>
2
3  char badshellcode[] = "\x48\xb8\x2f\x62\x69\x6e\x2f\x6c\x73\x00"
4                          "\x50"
5                          "\x54"
6                          "\x5f"
7                          "\x48\x31\xd2"
8                          "\x48\xc7\xc0\x3b\x00\x00\x00"
9                          "\x0f\x05";
10
11 int main(){
12     (*(void (*)())badshellcode)();
13     return 0;
14 }
```

使用 gcc 编译，并执行，结果如下。

```
wjm@ubuntu:~/Desktop/networksecurity/chapter9$ gcc -fno-stack-protector -z execstack -g -o shellcode shellcode.c
wjm@ubuntu:~/Desktop/networksecurity/chapter9$ ./shellcode
get_ls_assembly      get_sh_assembly      shellcode
get_ls_assembly.c    get_sh_assembly.c    shellcode.c
get_ls_machinecode.c machinecode            shellcode_from_net
get_ls_machinecode2.c machinecode2           shellcode_from_net.c
wjm@ubuntu:~/Desktop/networksecurity/chapter9$
```

下面完善 shellcode，将 shellcode 中的\x00 去除。

```
File Edit Selection Find View Goto Tools Project Preferences
g get_ls_machinecode2.c x get_ls_assembly.c x she
1  #include <stdio.h>
2
3  int main(){
4      asm (
5          "xor %rax,%rax;"
6          "push %rax;"
7          "mov $0x736c2f2f6e69622f,%rax;"
8          "push %rax;"
9          "push %rsp;"
10         "pop %rdi;"
11
12         "xor %rdx,%rdx;"
13
14         "xor %rax,%rax;"
15         "mov $0x3b,%al;"
16         "syscall;");
17     return 0;
18 }
```

使用 gcc 编译，objdump 进行反汇编。

```

000000000000005fa <main>:
 5fa: 55                push    %rbp
 5fb: 48 89 e5          mov     %rsp,%rbp
 5fe: 48 31 c0          xor     %rax,%rax
 601: 50                push    %rax
 602: 48 b8 2f 62 69 6e 2f movabs  $0x736c2f2f6e69622f,%rax
 609: 2f 6c 73
 60c: 50                push    %rax
 60d: 54                push    %rsp
 60e: 5f                pop     %rdi
 60f: 48 31 d2          xor     %rdx,%rdx
 612: 48 31 c0          xor     %rax,%rax
 615: b0 3b            mov     $0x3b,%al
 617: 0f 05            syscall
 619: b8 00 00 00 00    mov     $0x0,%eax
 61e: 5d                pop     %rbp
 61f: c3                retq

```

写出如下文件。

```

get_ls_assembly.c x shellcode_from_net.c x shellcode.c x shellcode_without0x00.c x
1  #include <stdio.h>
2
3  char badshellcode[] = "\x48\x31\xc0"
4                        "\x50"
5                        "\x48\xb8\x2f\x62\x69\x6e\x2f\x2f\x6c\x73"
6                        "\x50"
7                        "\x54"
8                        "\x5f"
9                        "\x48\x31\xd2"
10                       "\x48\x31\xc0"
11                       "\xb0\x3b"
12                       "\x0f\x05";
13 int main(){
14     (*(void (*)())badshellcode)();
15     return 0;
16 }

```

使用 gcc 编译，并运行，结果如下。

```

wjw@ubuntu:~/Desktop/networksecurity/chapter9$ gcc -fno-stack-protector
-z execstack -g -o shellcode_without0x00 shellcode_without0x00.c
wjw@ubuntu:~/Desktop/networksecurity/chapter9$ ./shellcode_without0x00
get_ls_assembly      get_ls_machinecode2.c  machinecode  shellcode.c
                    shellcode_without0x00
get_ls_assembly.c    get_sh_assembly        machinecode2  shellcode_fr
om_net              shellcode_without0x00.c
get_ls_machinecode.c get_sh_assembly.c      shellcode    shellcode_fr
om_net.c

```