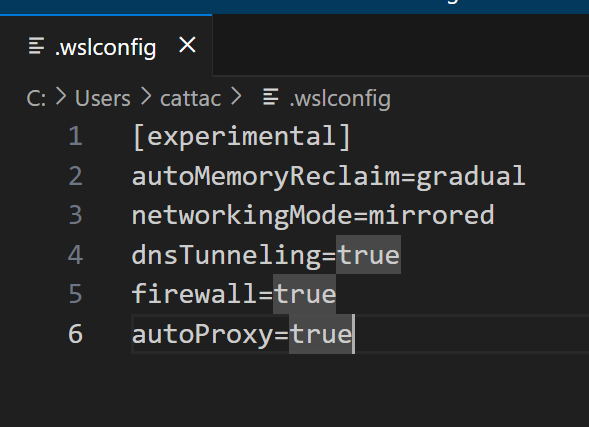
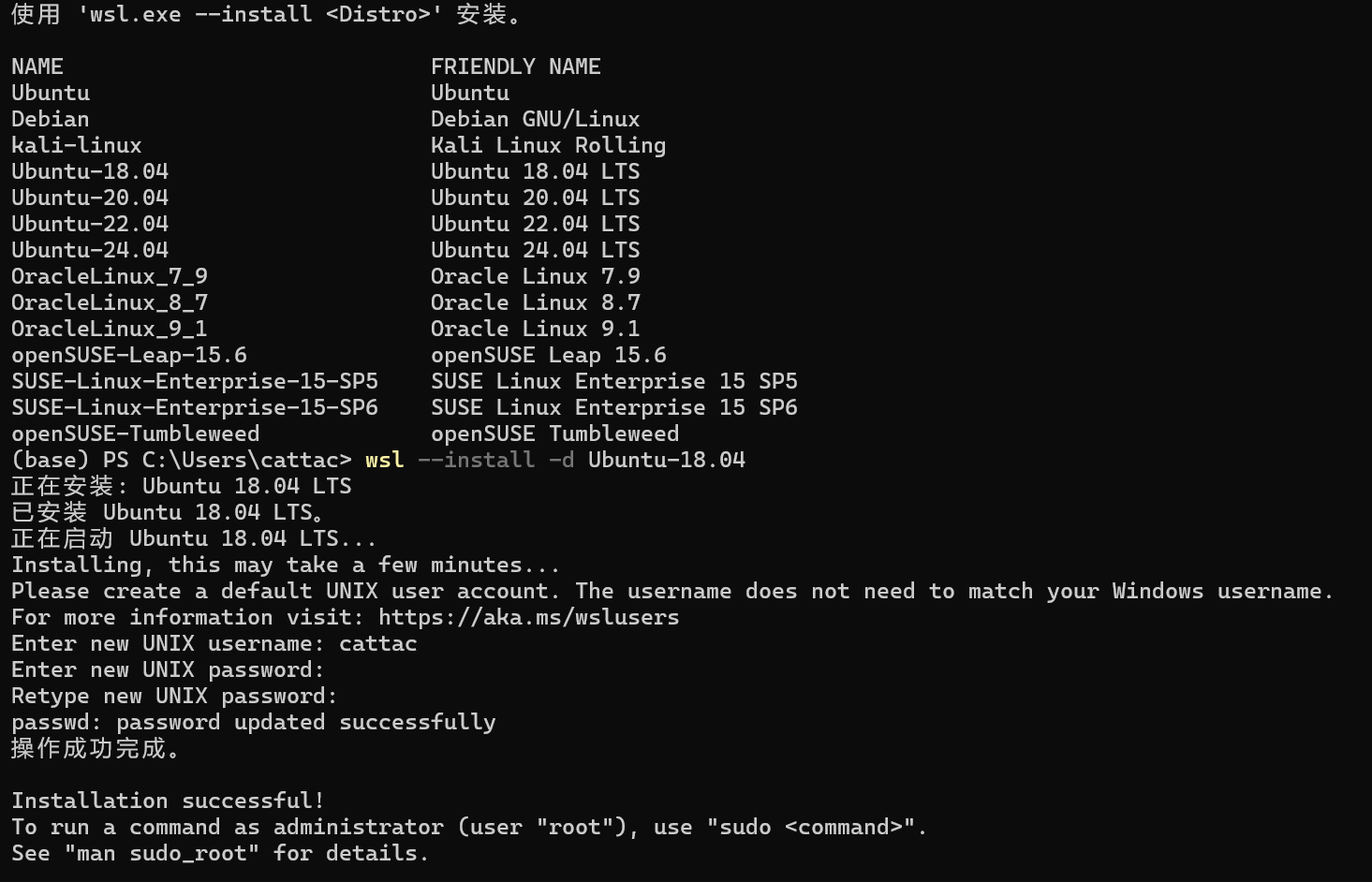
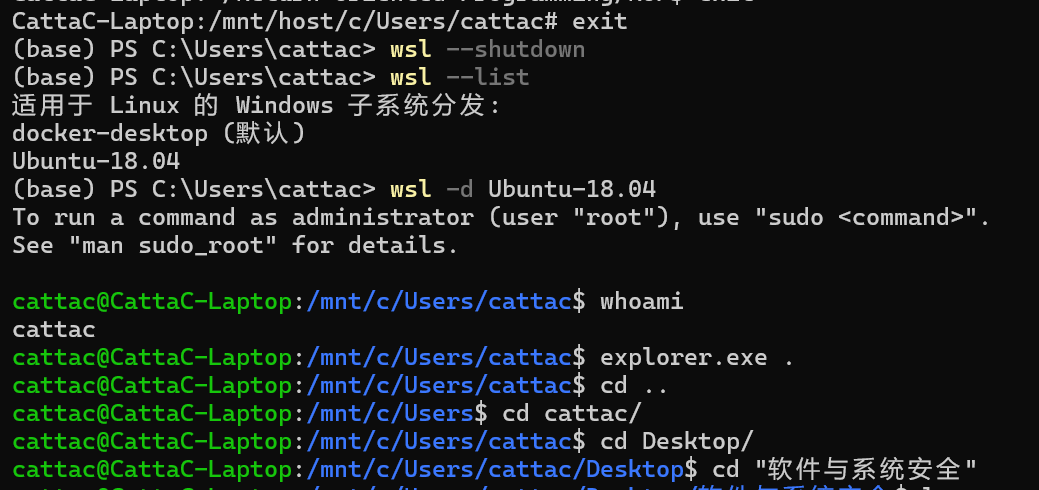
**配置WSL2的NAT代理**



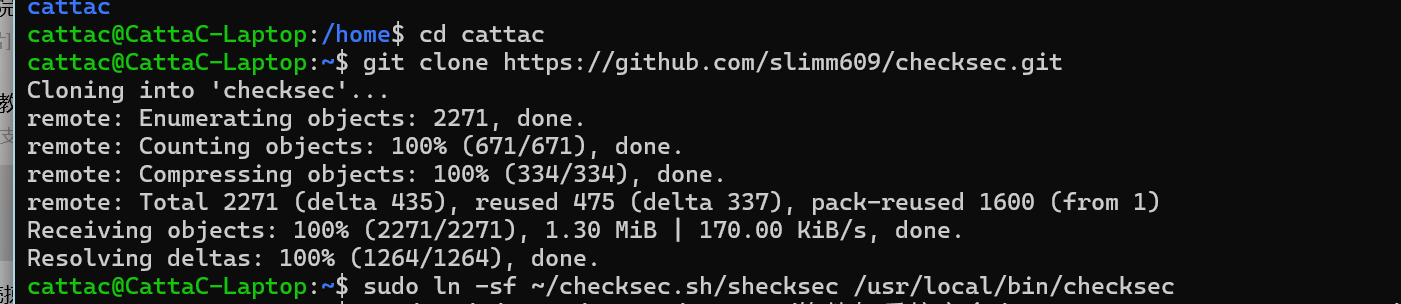
**安装Ubuntu18.04**



**启动正确的子系统**

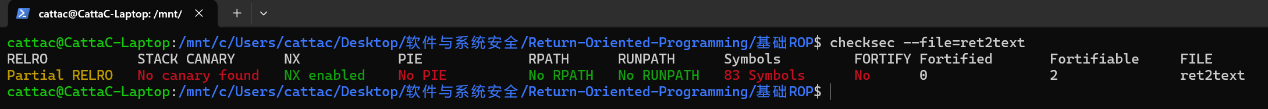
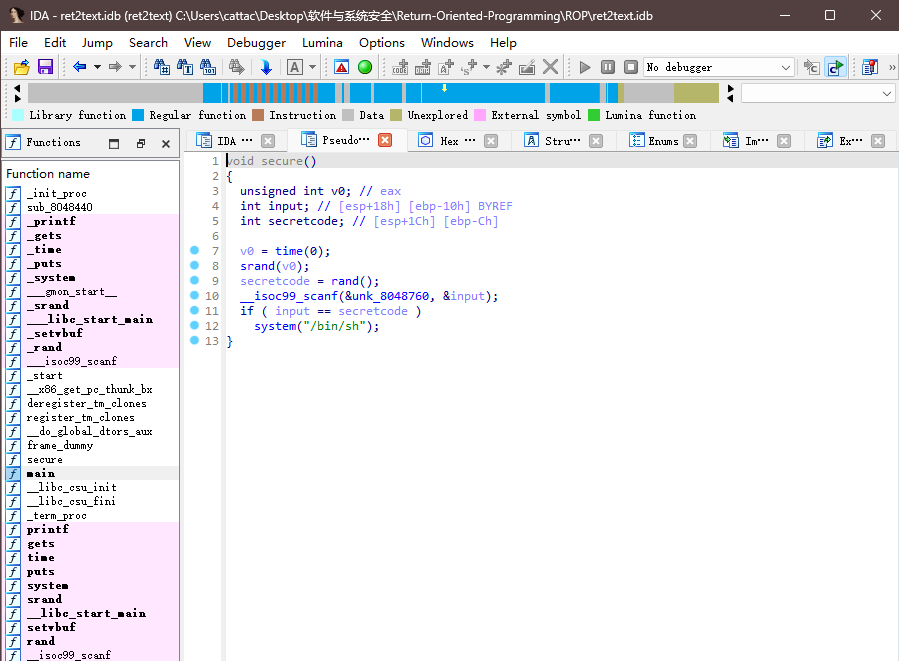


**下载并链接checksec**

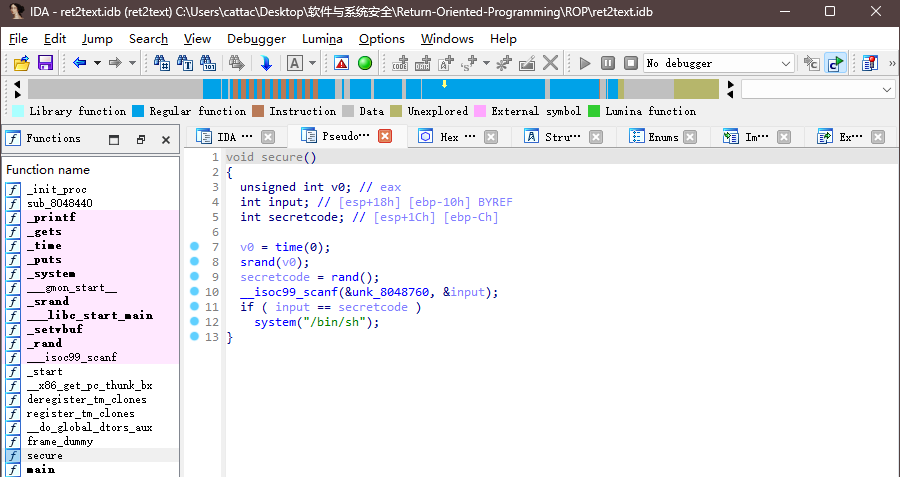


**Ret2text:**

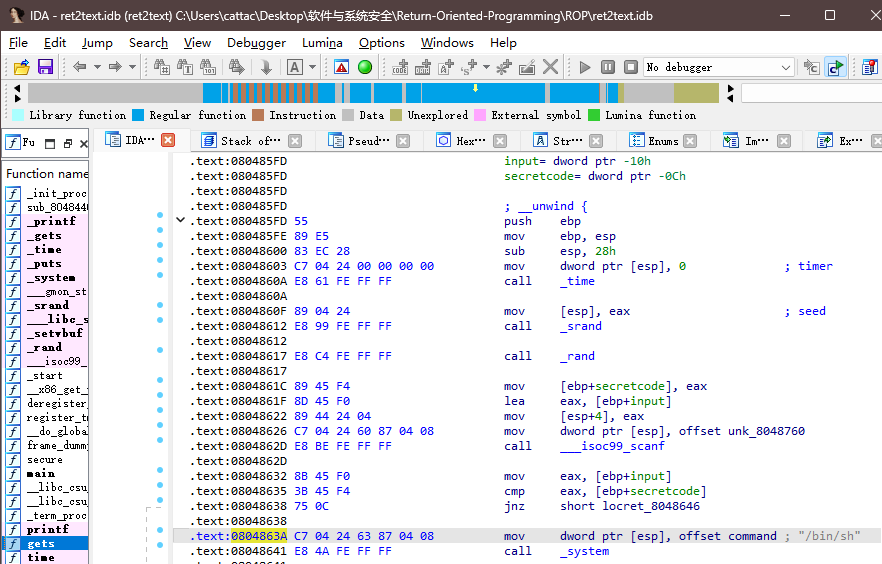
**运行checksec**

**使用IDA** **Pro分析代码**

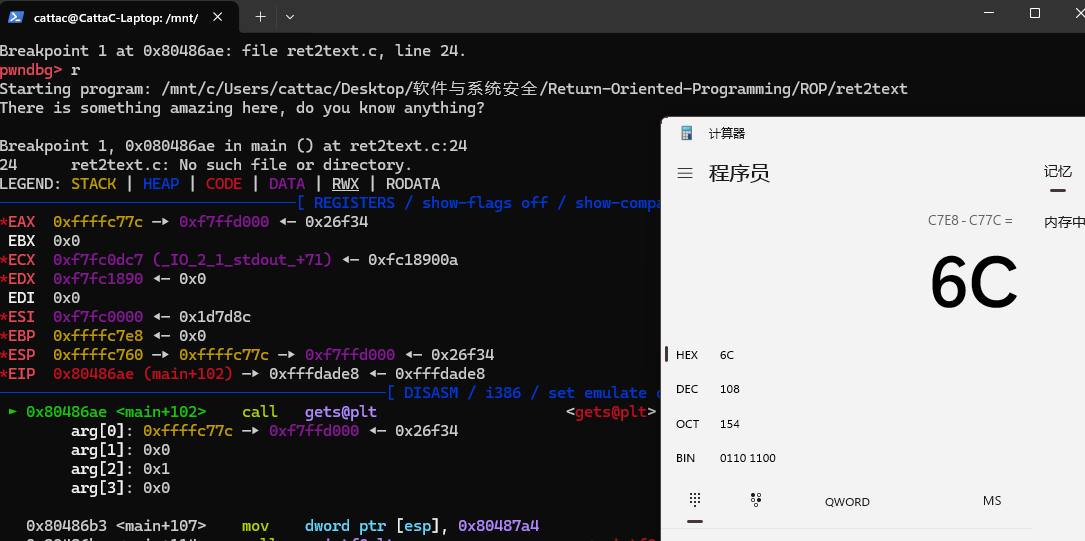
**发现调用shell的方式：使input和生成的随机数相同即可**



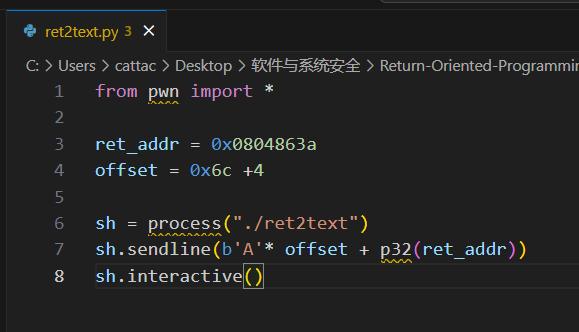
**查看汇编的地址内容，计算出input和secretcode的相对距离**



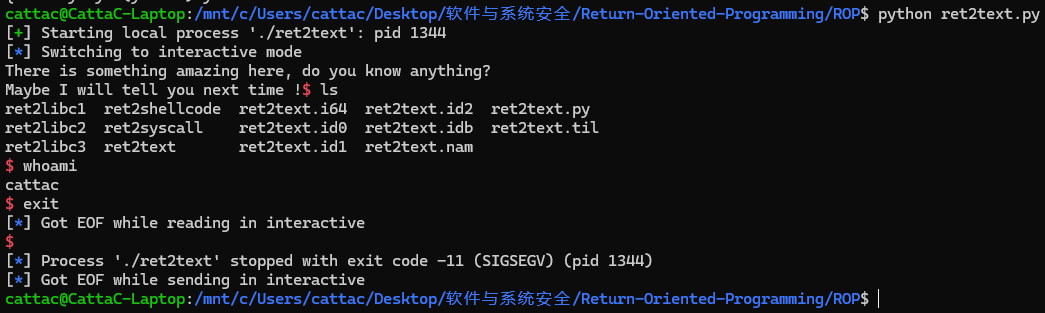
**查看栈长如图：**



**编写得到交互代码如图**

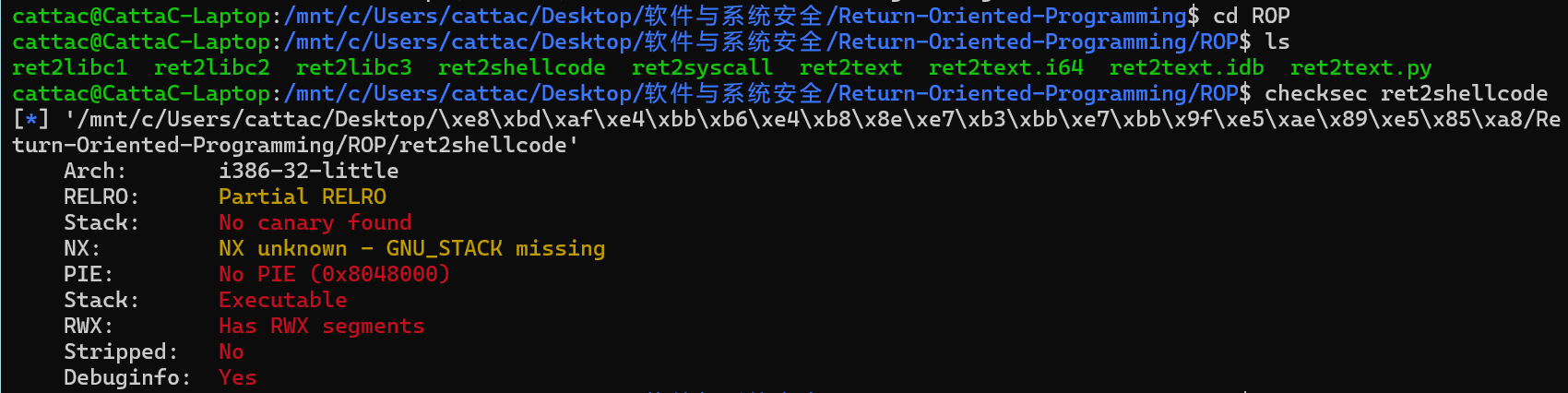


**运行结果如下：**

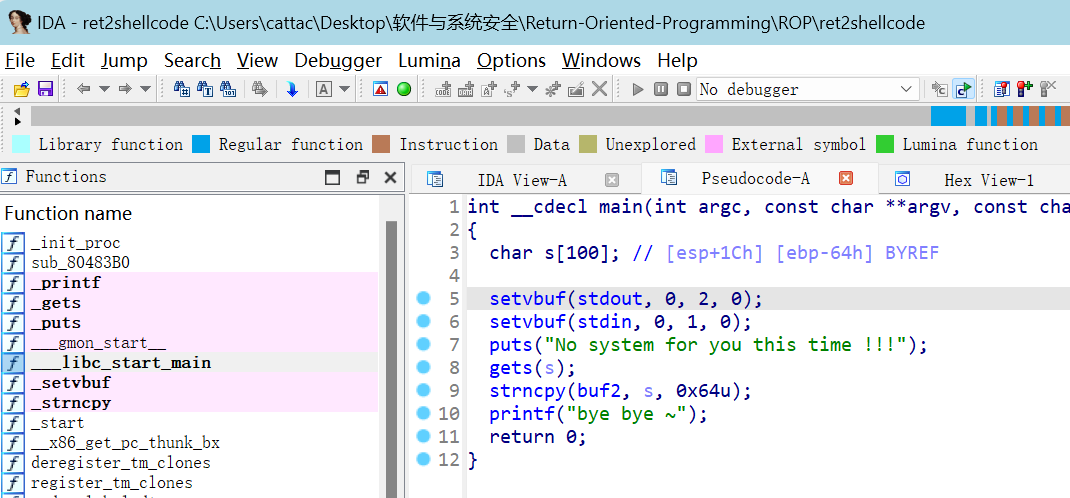


**Ret2shellcode**

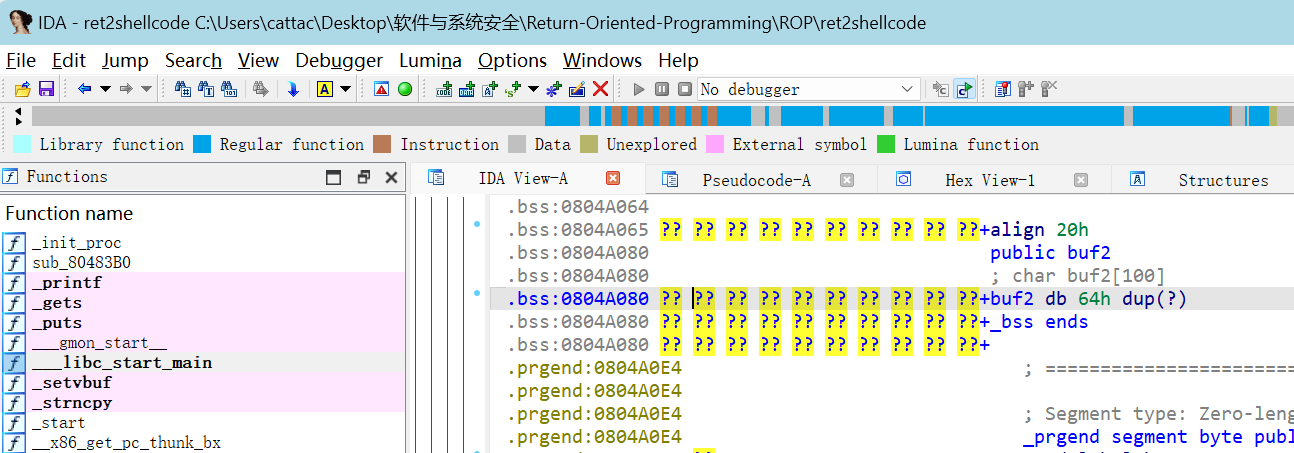
**使用checksec分析**



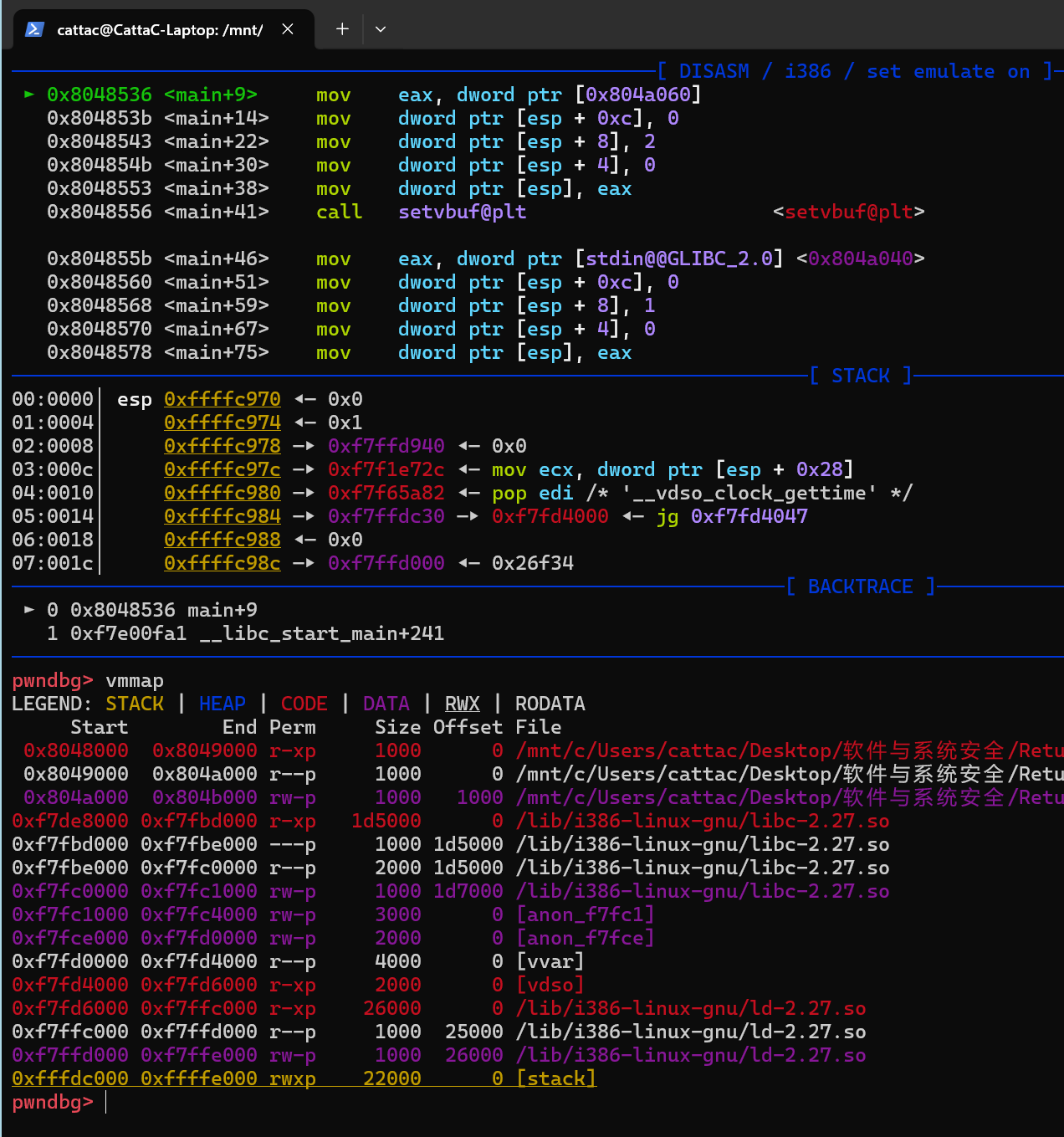
**查看反编译代码**



**可见buf2的位置**

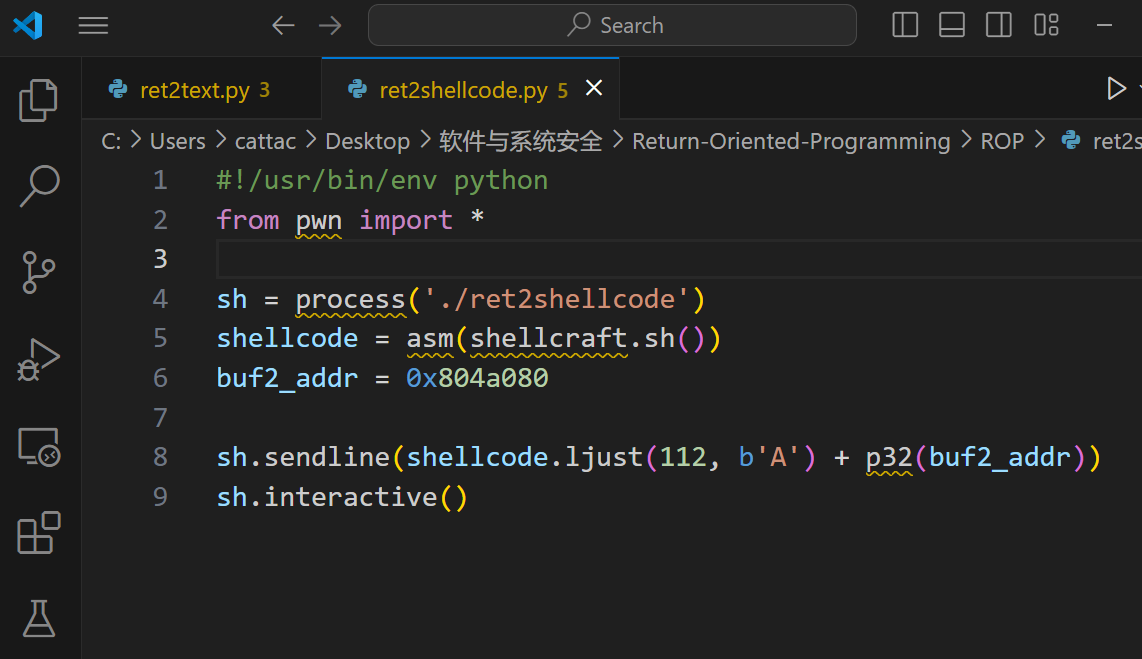


**使用pwndbg后，可见buf2所在位置没有shell的执行权限**

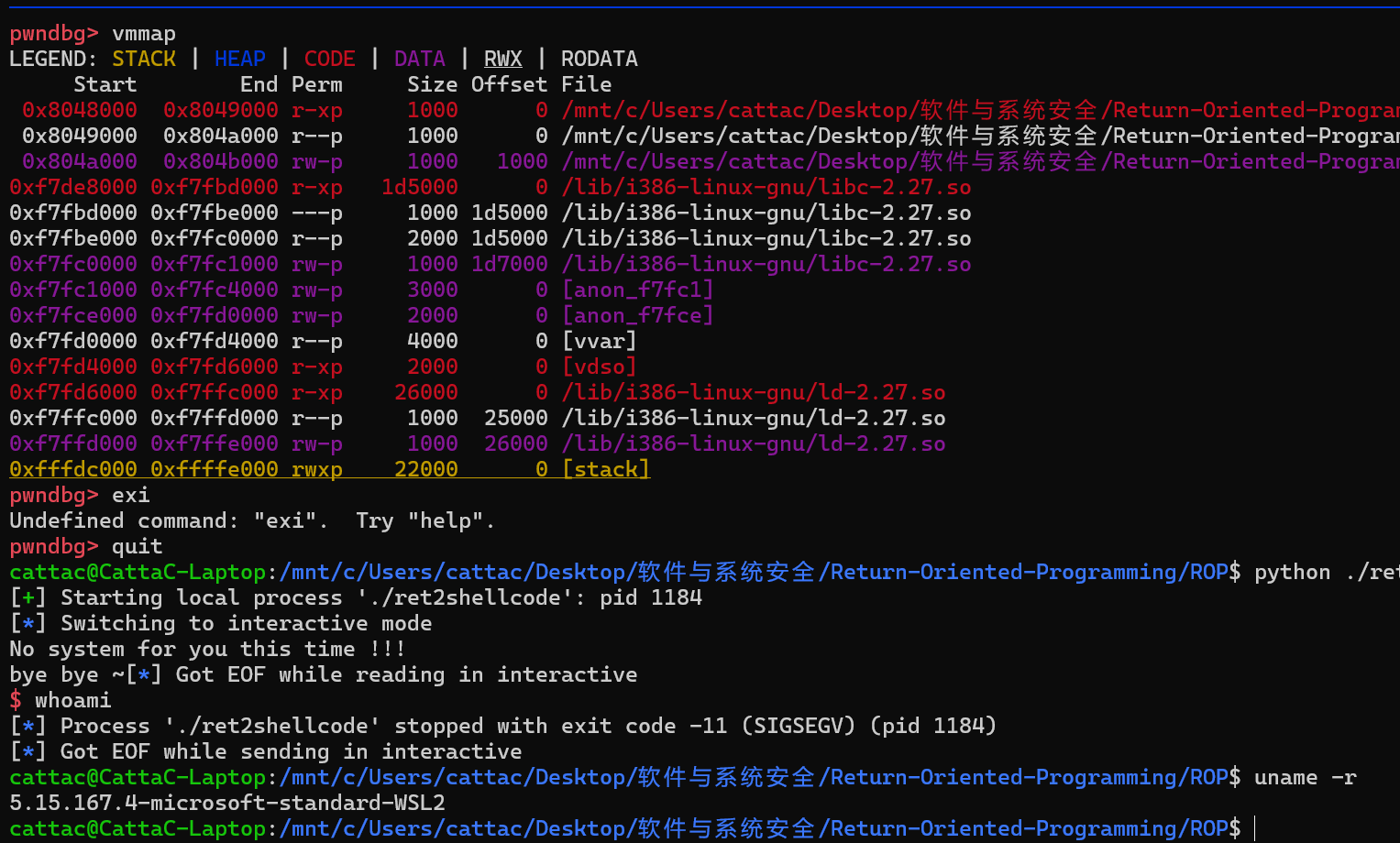


**（这里就没办法直接从.bass段注入了）**

**构建注入代码：**



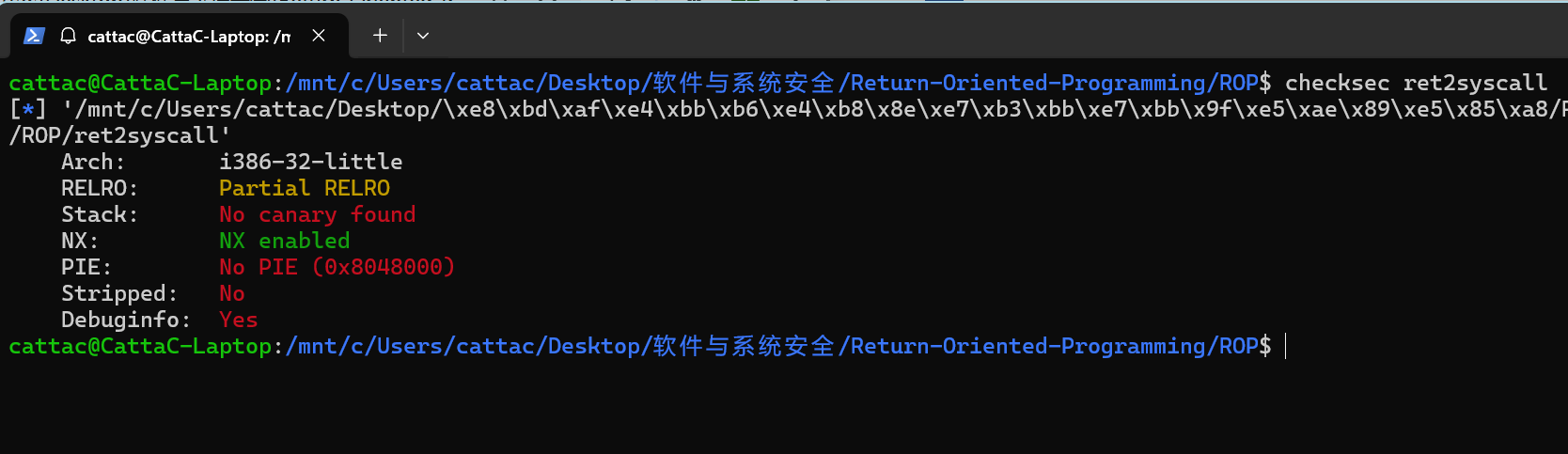
**仍然尝试使用.bass注入的方式，结果如下**



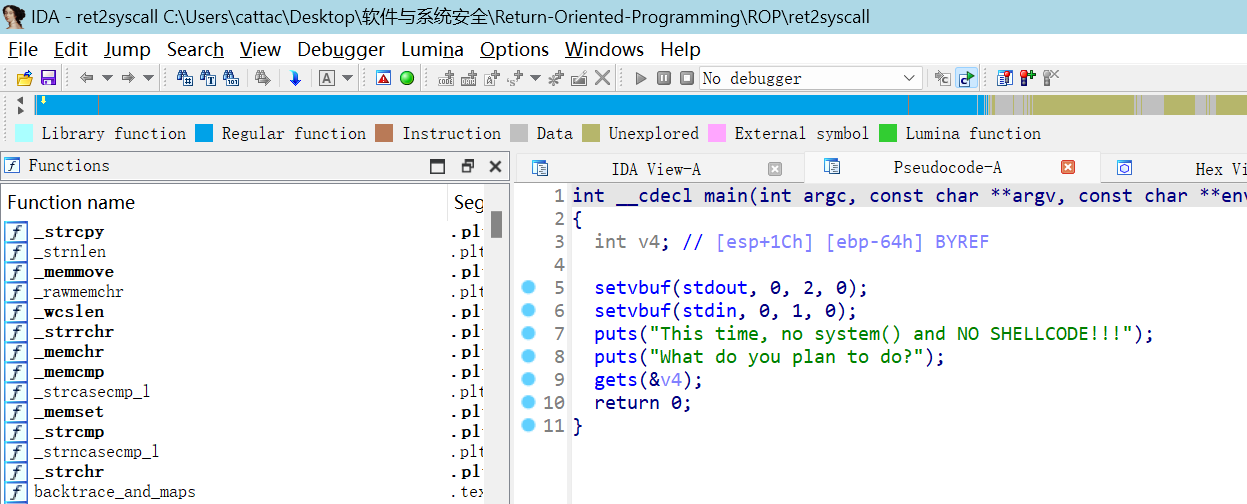
**可见失败的主要原因就是使用了WSL2 的系统内核，自动将.bss段保护了，因此攻击失败**

**Ret2syscall**

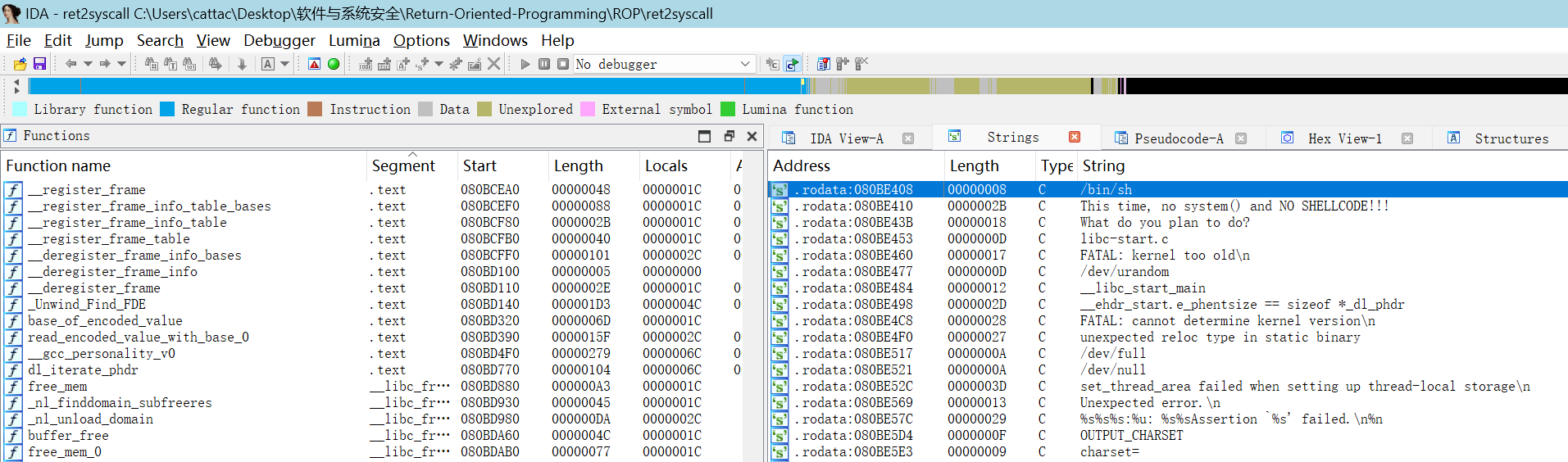
**首先进行checksec分析，可见NX保护开启**



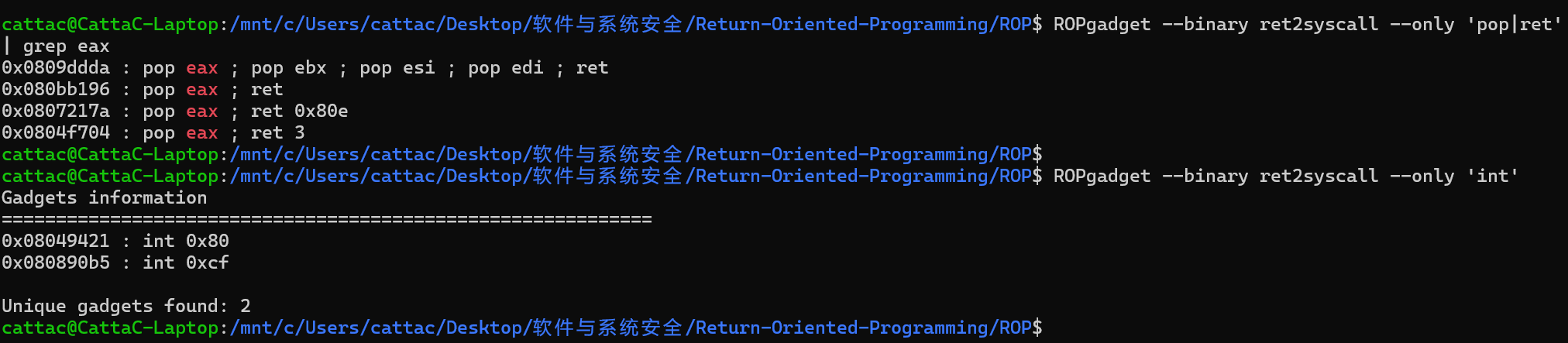
**反编译可执行文件，可见gets漏洞可利用：**

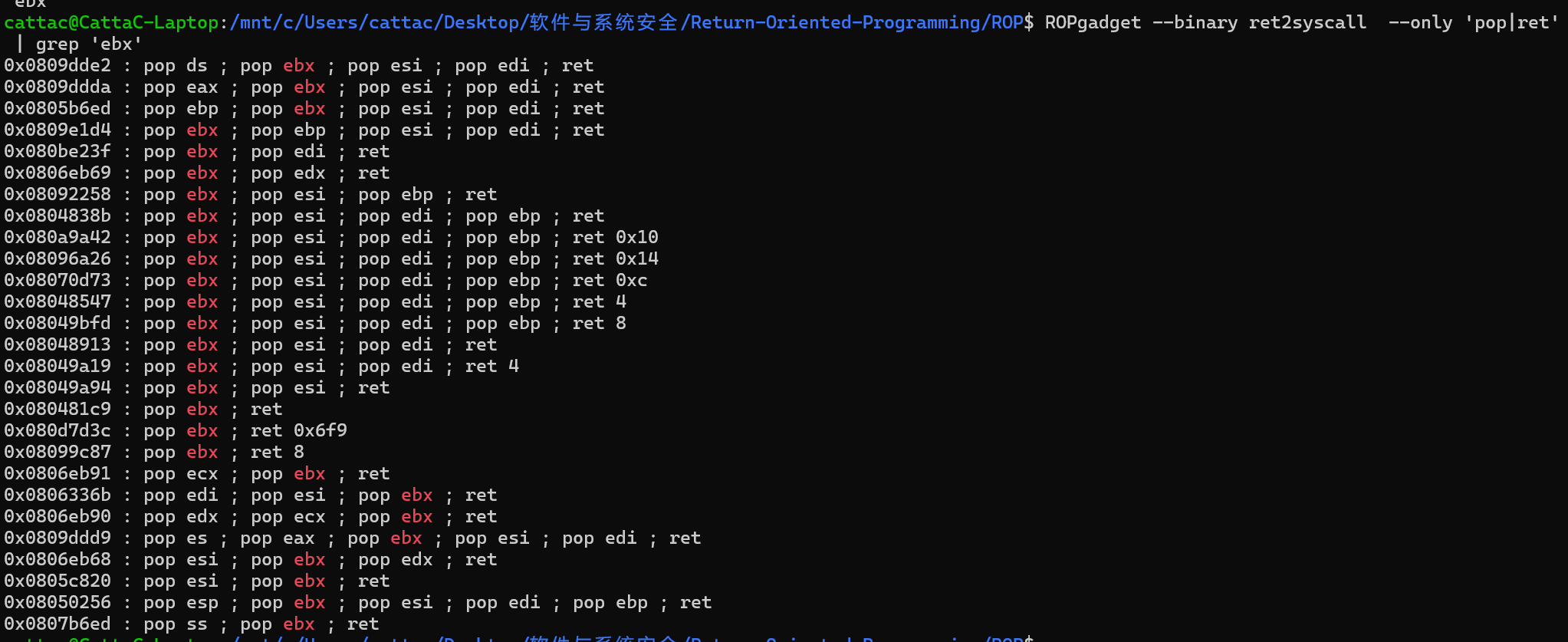


**在string的subview中可见/bin/sh：**



**使用ROPgadget寻找可用代码片段**





**可以获得如下寄存器和data的地址：**

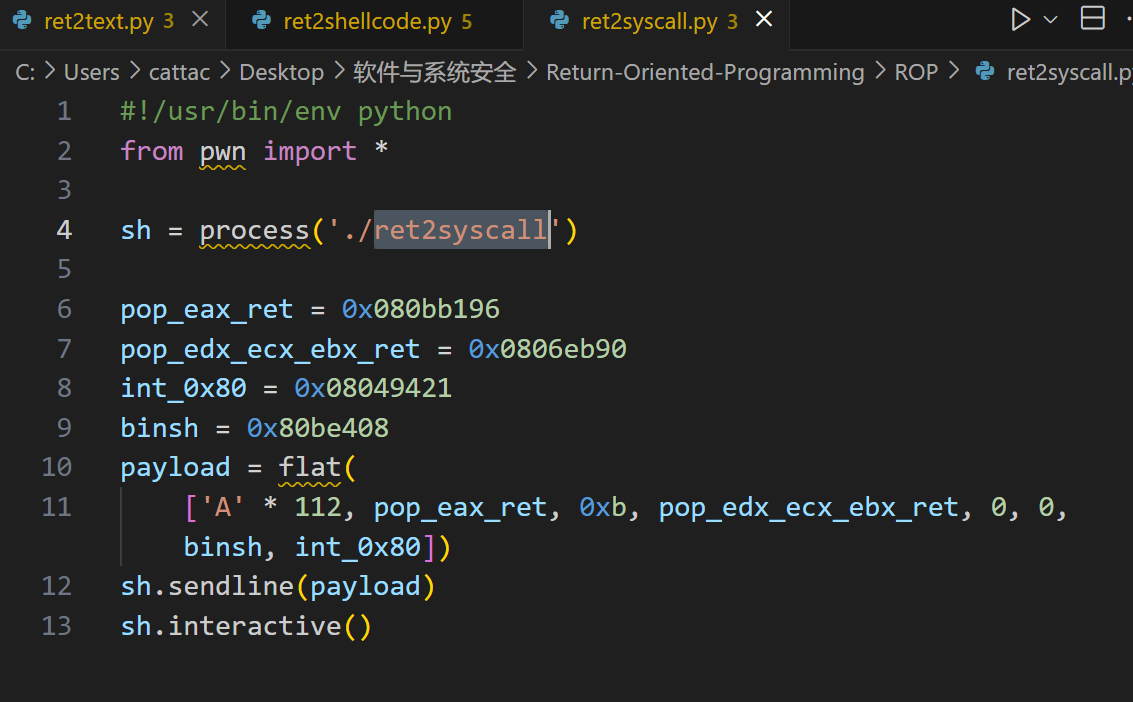
**0x080bb196 : pop eax ; ret**

**0x0806eb90 : pop edx ; pop ecx ; pop ebx ; ret**

**0x080be408 : /bin/sh**

**0x08049421 : int 0x80**

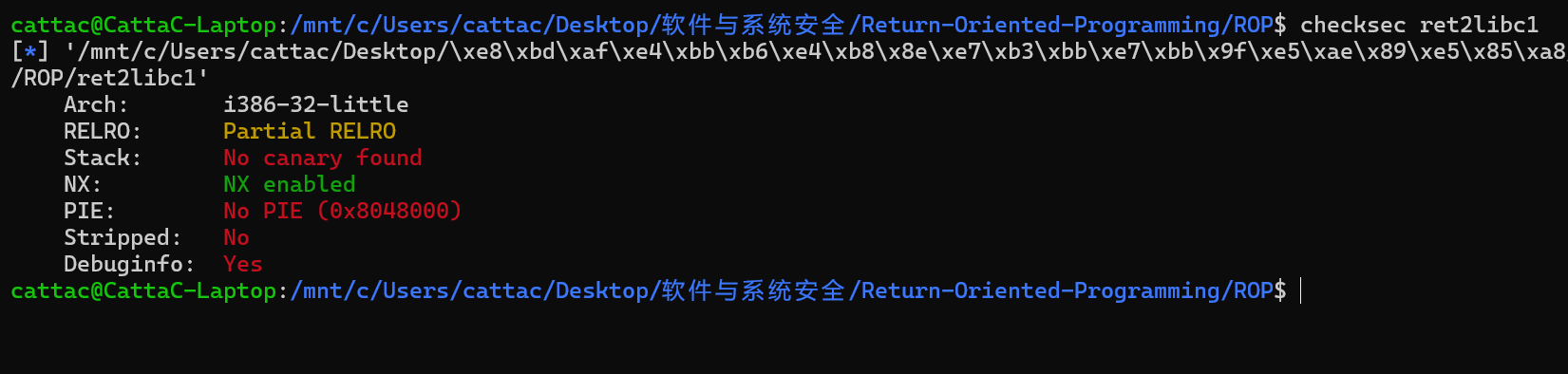
**编写代码**



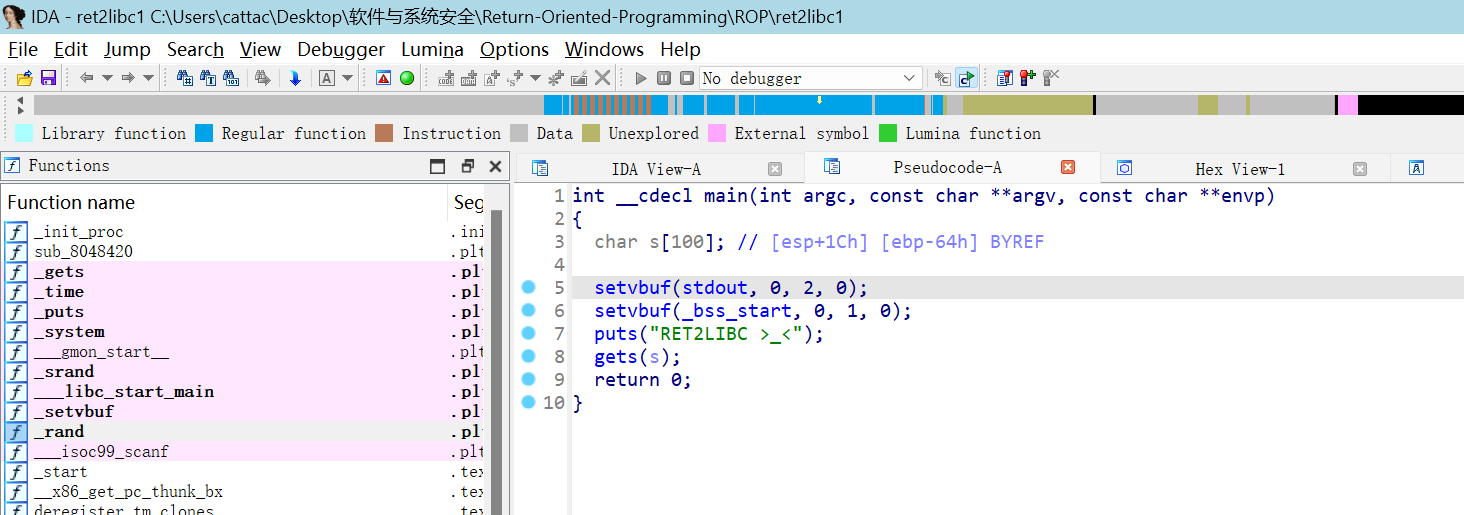
**运行结果**



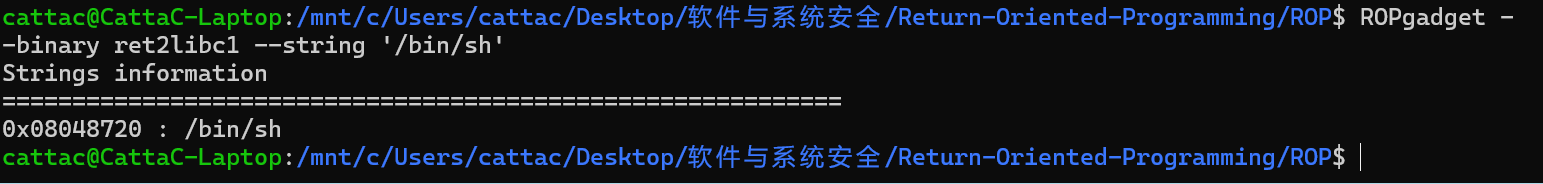
**ret2libc1同样是首先使用checksec进行保护模式的检测**



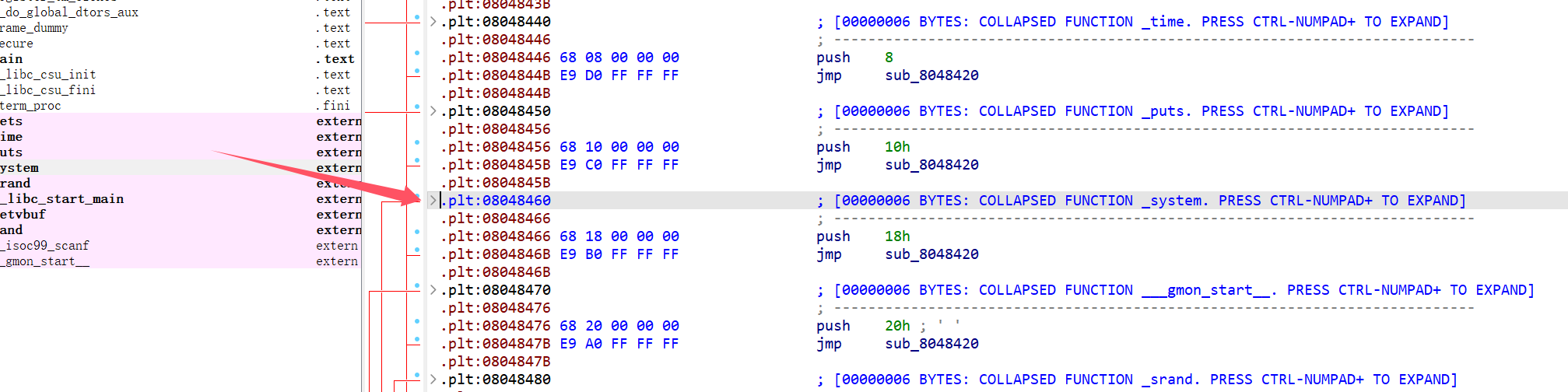
**接着使用IDA Pro进行反编译**



**这次实验首先使用ROPgadget来寻找’/bin/sh’**



**此外，本次实验比较幸运的地方在于可以直接找到调用system函数的位置：**

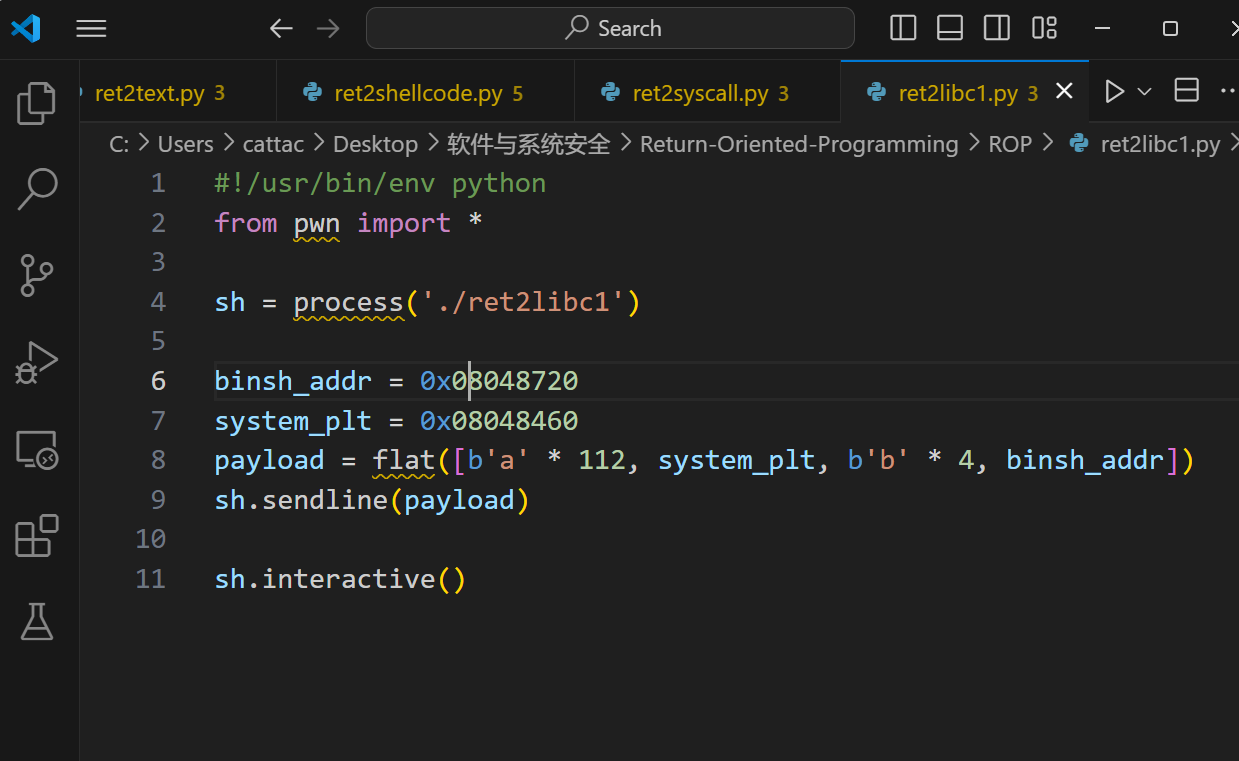


**可见，所需内容的位置如下：**

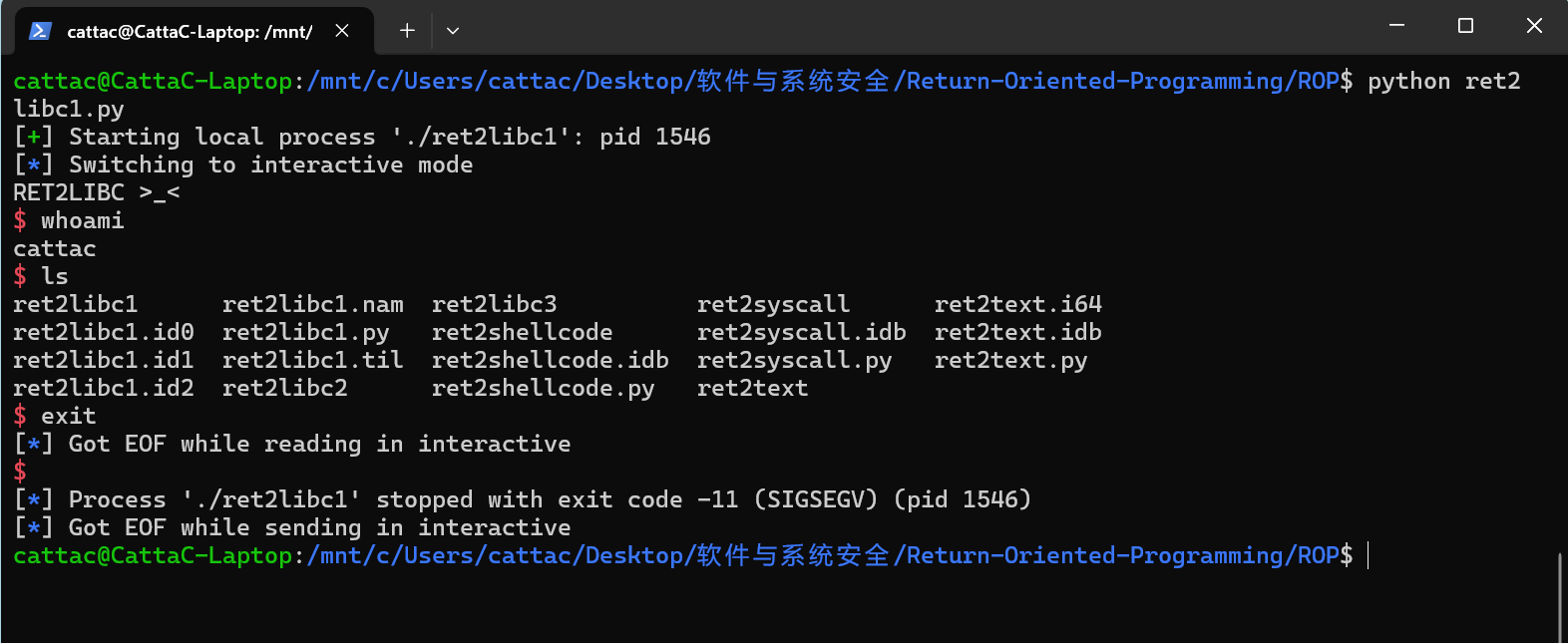
**0x08048720 : ‘/bin/sh’**

**0x08048460：system**

**编写代码如下：**

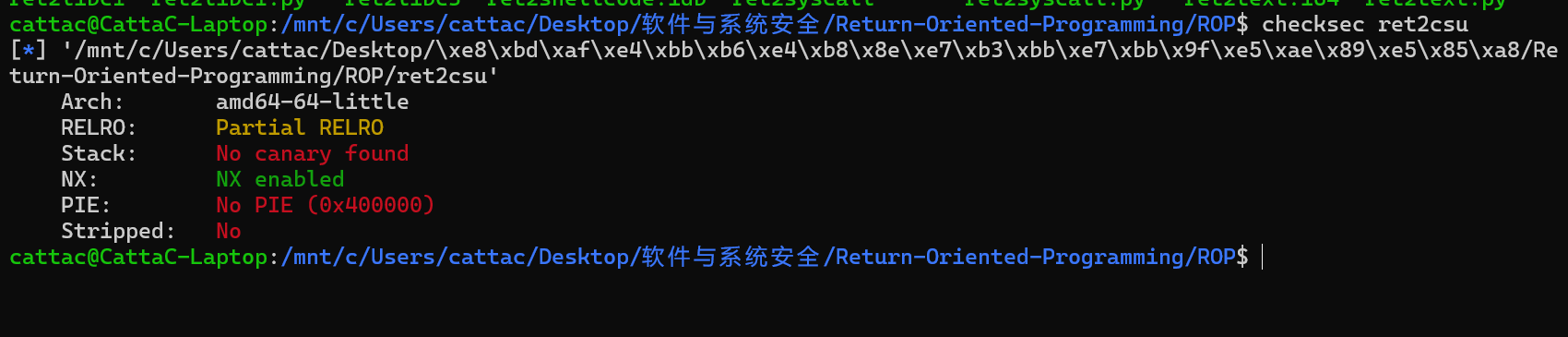


**运行结果如图：**

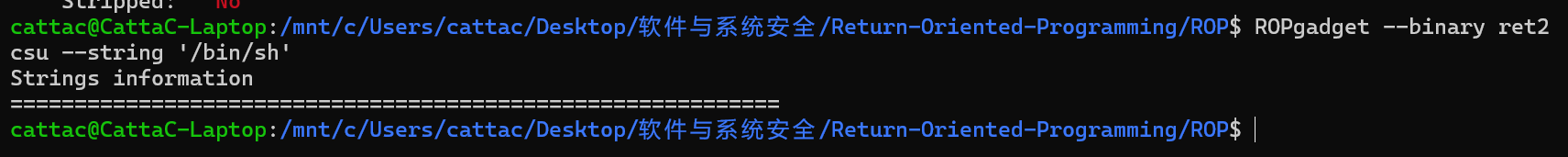


**Ret2csu**

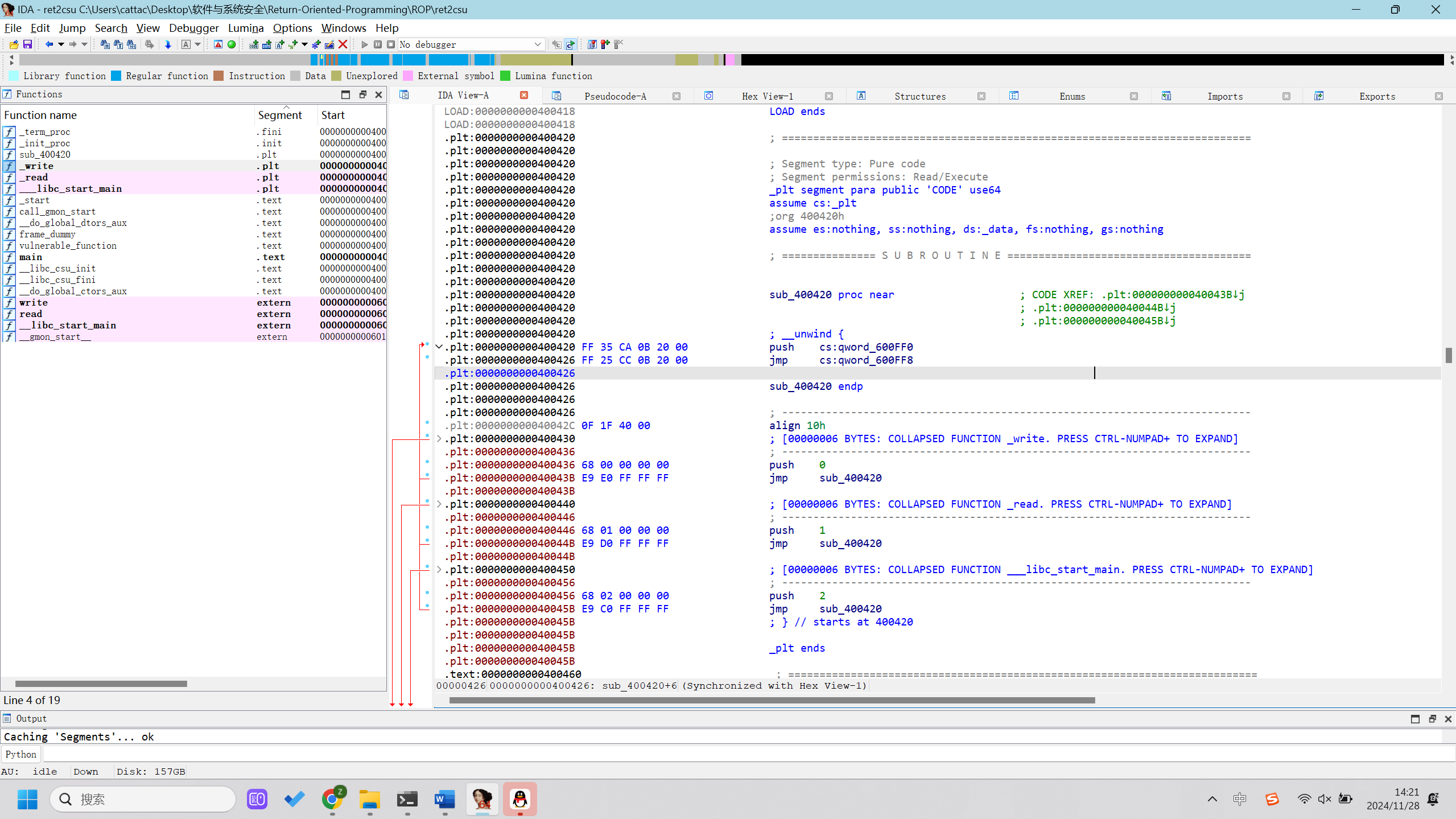
**首先查看程序信息：**



**没有发现’/bin/sh’:**



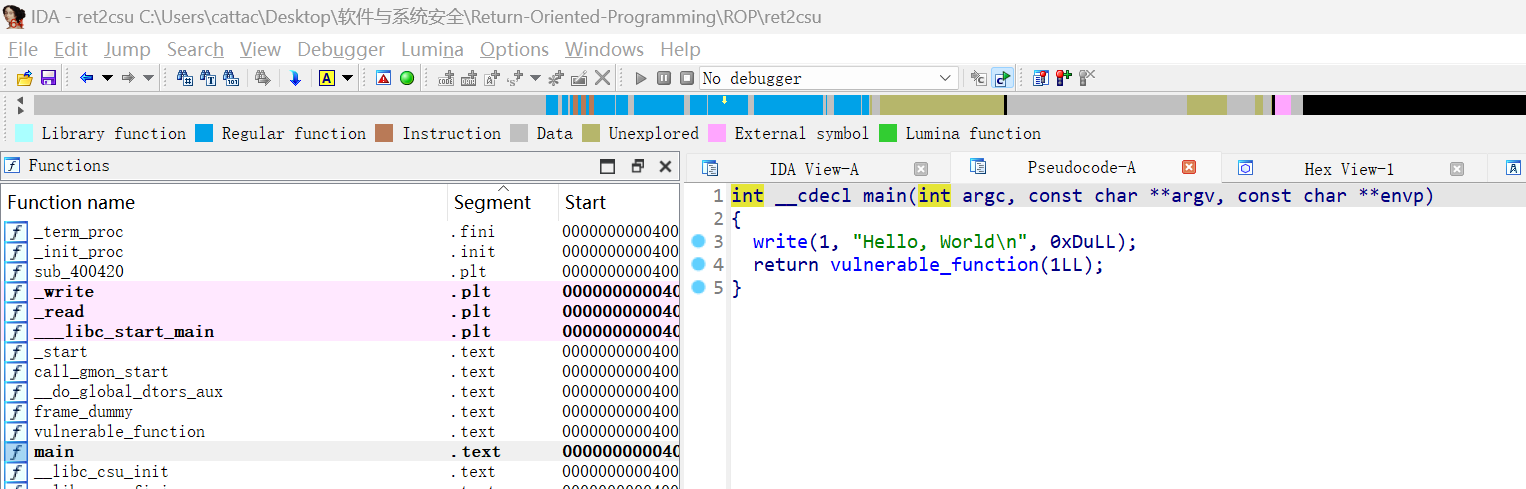
**同时，在.plt段中也没有发现system函数的调用情况**



因此，考虑到攻击所需的常用手段，我们需要在程序代码之外的合适地方寻找到system和‘/bin/sh’这两个关键的gadgets，也即寻找通用gadget。

1. 程序分析

首先，我们还是使用IDA Pro来对ret2csu的可执行文件进行分析。通过checksec的分析，我们已知该文件是64位的可执行文件，因此需要使用IDA Pro x64版本。

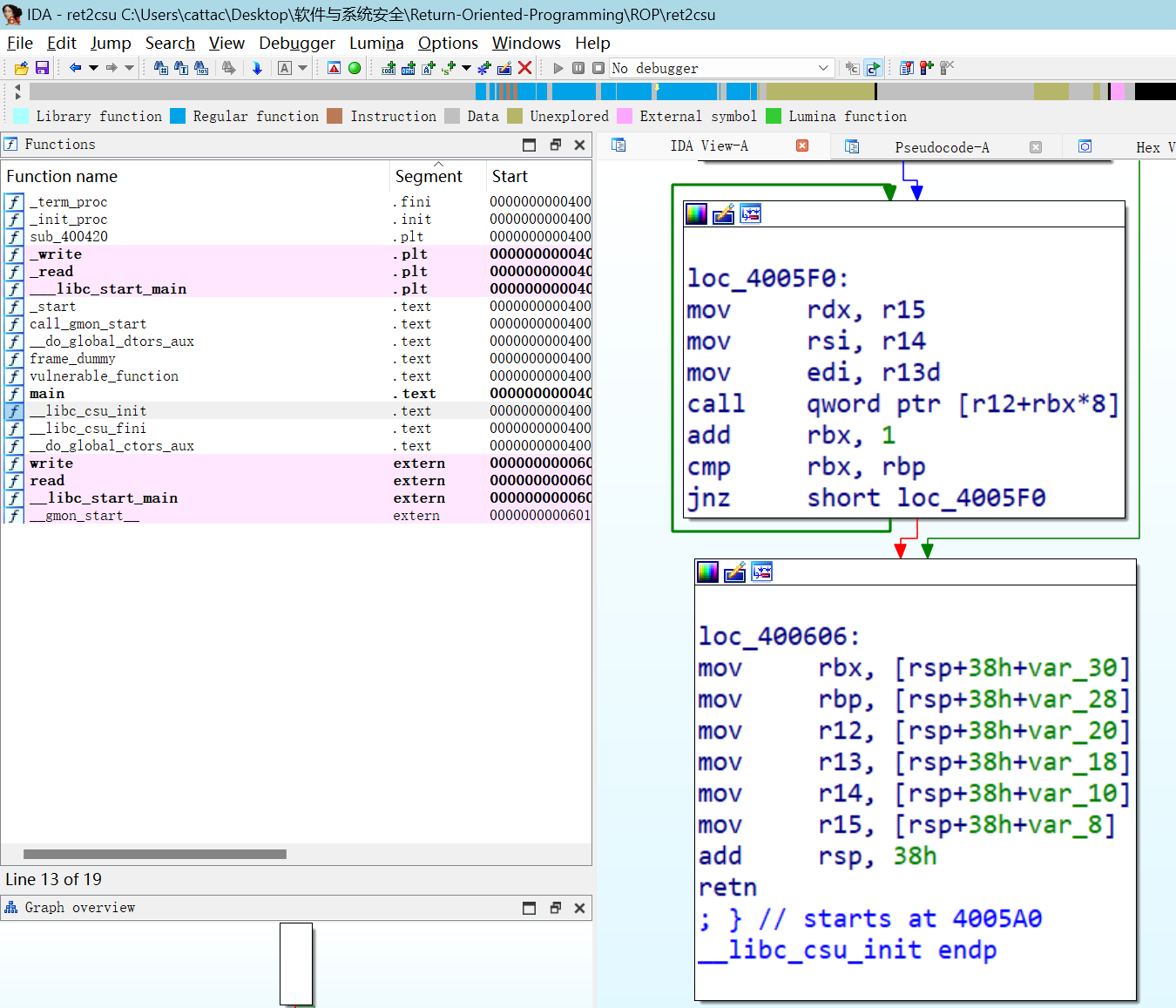


主函数可见，在进行简单的消息输出之后调用了vulnerable\_function这一函数，因此我们只能到该函数中序寻找可能的漏洞并发起攻击。因为程序在编译过程中会加入一些通用函数用来进行初始化操作（比如加载libc.so的初始化函数），所以虽然很多程序的源码不同，但是初始化的过程是相同的，因此针对这些初始化函数，我们可以提取一些通用的gadgets加以使用，从而达到我们想要达到的效果。

在通过查询网络资料加以学习后，了解了利用x64编译下的\_\_libc\_csu\_init方式，简称为CSU。

1. 漏洞原理

csu实际上是基于libc的一种泄露函数基地址的做法，其原理还是基于rop链的构造。大多时候我们难以找到每个寄存器对应的gadgets，但是当我们遇到wirte函数泄露的时候却必须要控制三个寄存器（64位首先存入数据的rdi，rsi，edi），这时候，我们就可以利用 x64 下的 \_\_libc\_csu\_init 中的 gadgets。



在IDA Pro中可见，csu中有如下两块代码值得我们注意和利用——前者记gadget1，后者记为gadget2。可见我们可以利用gadget1对rdx，rsi和edi进项赋值，而它们的数值来源，即r14，r13和r12是可以通过gadge2进行栈的溢出来注入数值的。

在完成以上三个数值的修改后，我们将会调用**call qword ptr [r12+rbx\*8]**。这时候我们只要再将rbx的值赋值为0，再通过构造栈上的数据，我们就可以控制pc指针去调用我们想要调用的函数了。执行完**call qword ptr [r12+rbx\*8]**之后，程序会对rbx+=1，然后对比rbp和rbx的值，如果相等就会继续向下执行并ret到我们想要继续执行的地址。