# 2021年春秋杯网络安全联赛秋季赛 勇者山峰 re部分wp(snake)

原创

雪月三十 于 2021-12-01 17:49:56 发布 2271 收藏 1

文章标签： python c语言



本文链接：https://blog.csdn.net/weixin_49764009/article/details/121660322



## 题目链接

> 链接：https://pan.baidu.com/s/1VpEXzkKM6D2EDd1ZhUeeog 提取码：fn2l

## 运行

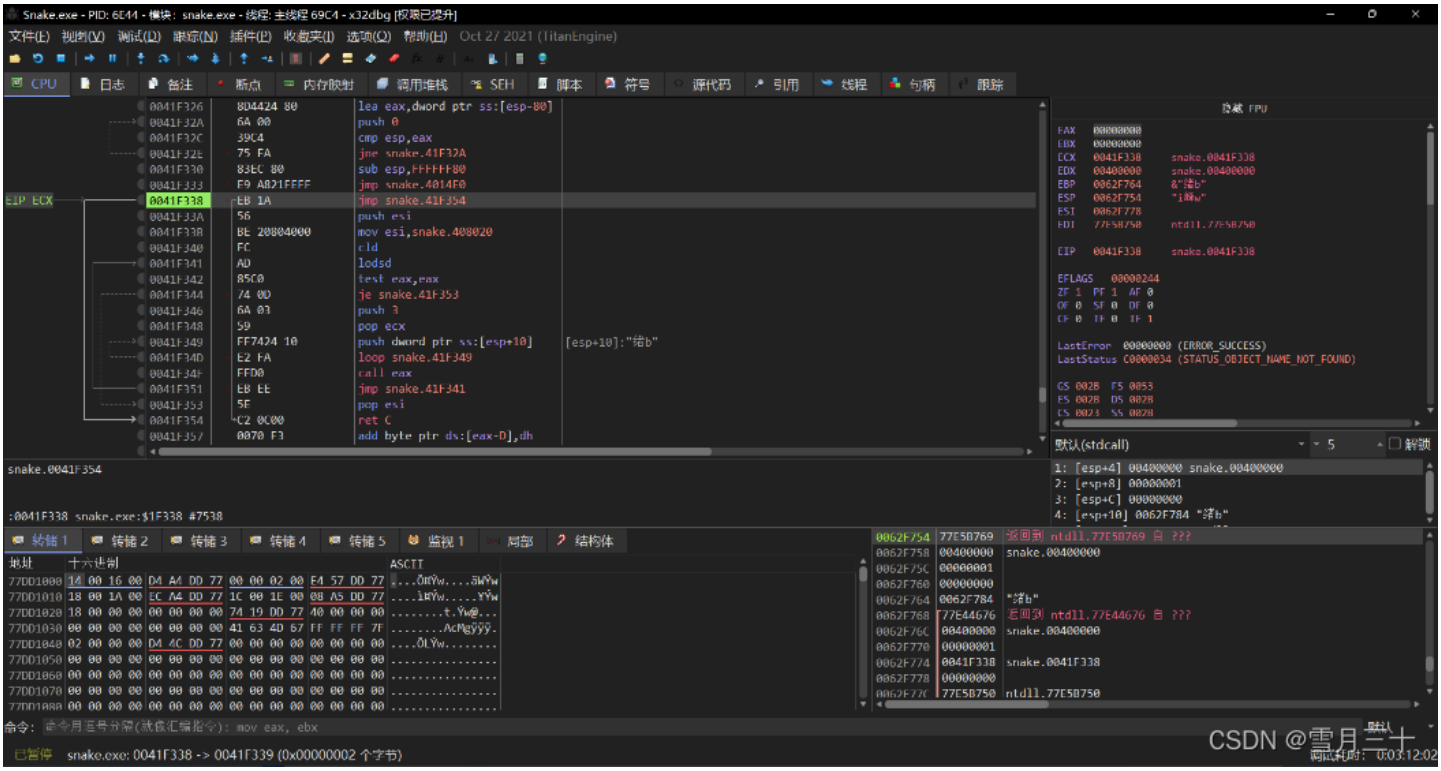

贪吃蛇 典中典 ~~一开始猜测是py文件打包起来的来着~~ 后来发现并不是

## 基本面


32位 UPX(-)[modified]壳 并不是upx壳

## 手动脱壳

去除了系统断点之后会停在下面这个位置




既然我们知道是有壳的

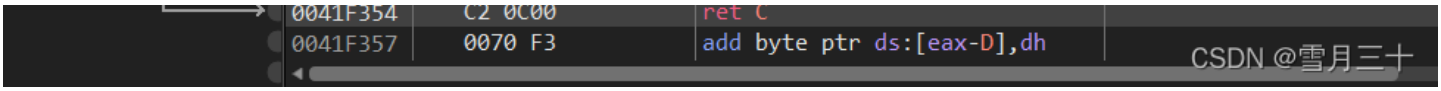
所以这些代码还不是程序真正的代码我们需要手动来脱一下这个upx壳

**手动脱upx我们只需要一直 单步步过 找到ret函数（返回值）**

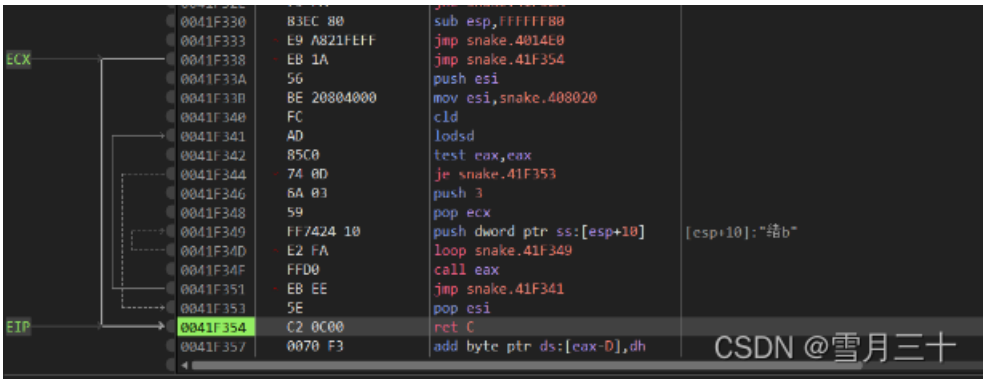
**然后步入返回值中 继续单步步过 再次寻找ret函数 步入**

**依次重复 直到找到popad 或者 明显的程序入口（lea rax,qword ptr ss:[rsp-80])**

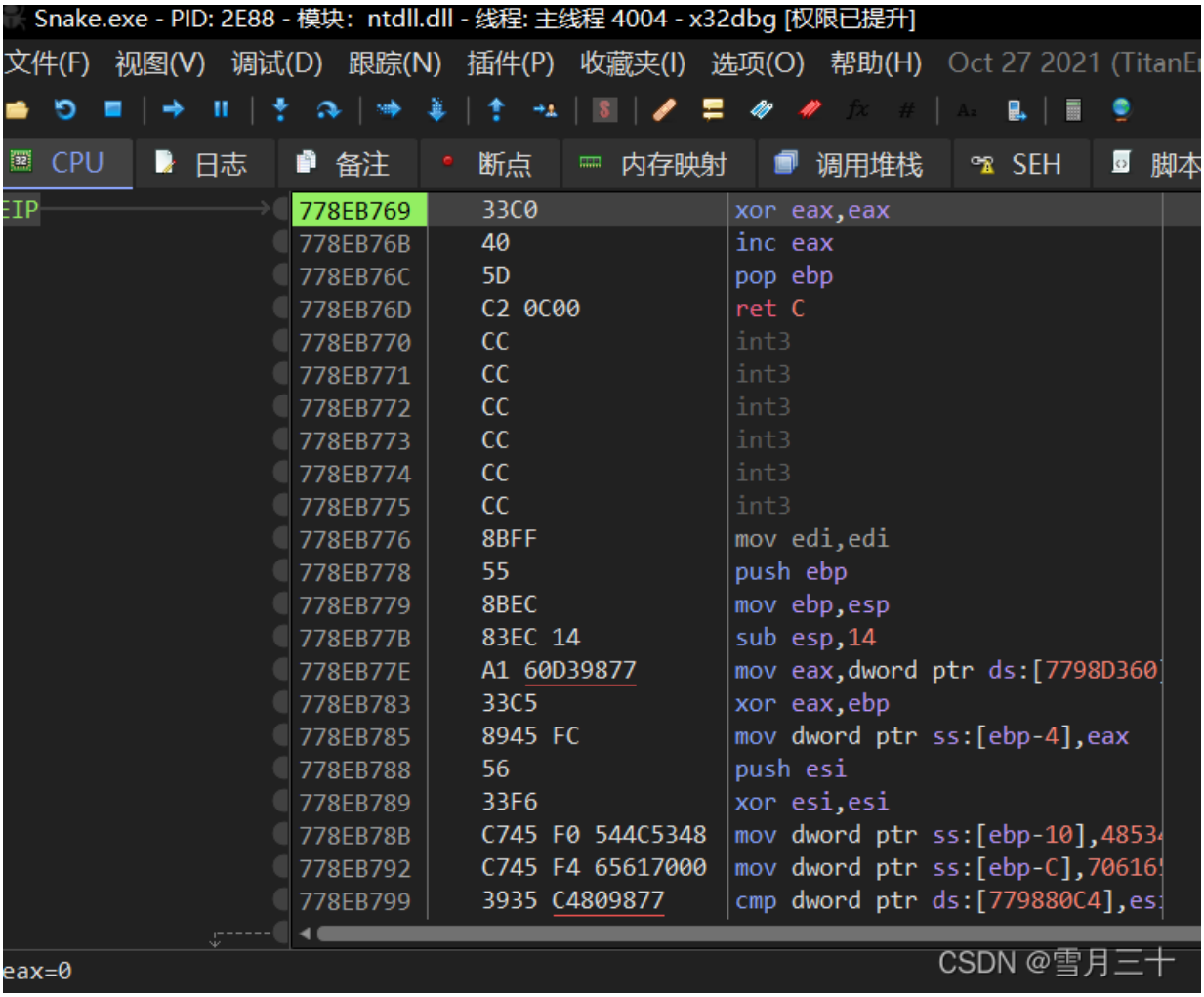
流程图：

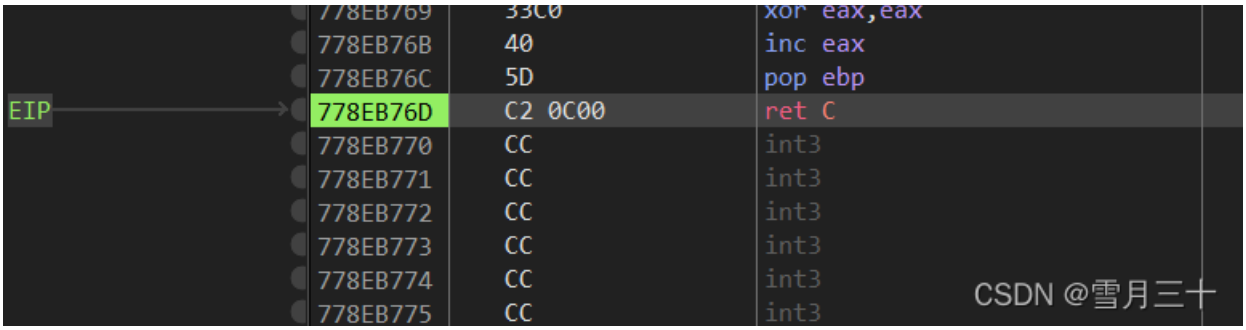


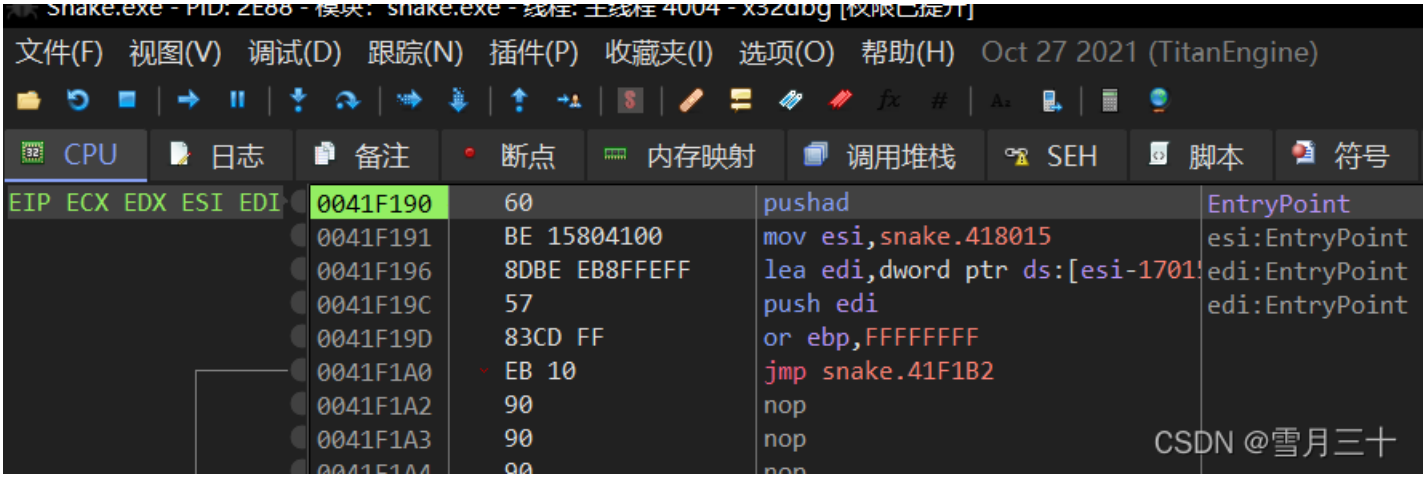

像一开始这种一下子就看到了ret函数我们就可以直接F4 运行到这个位置
这样可以节省时间

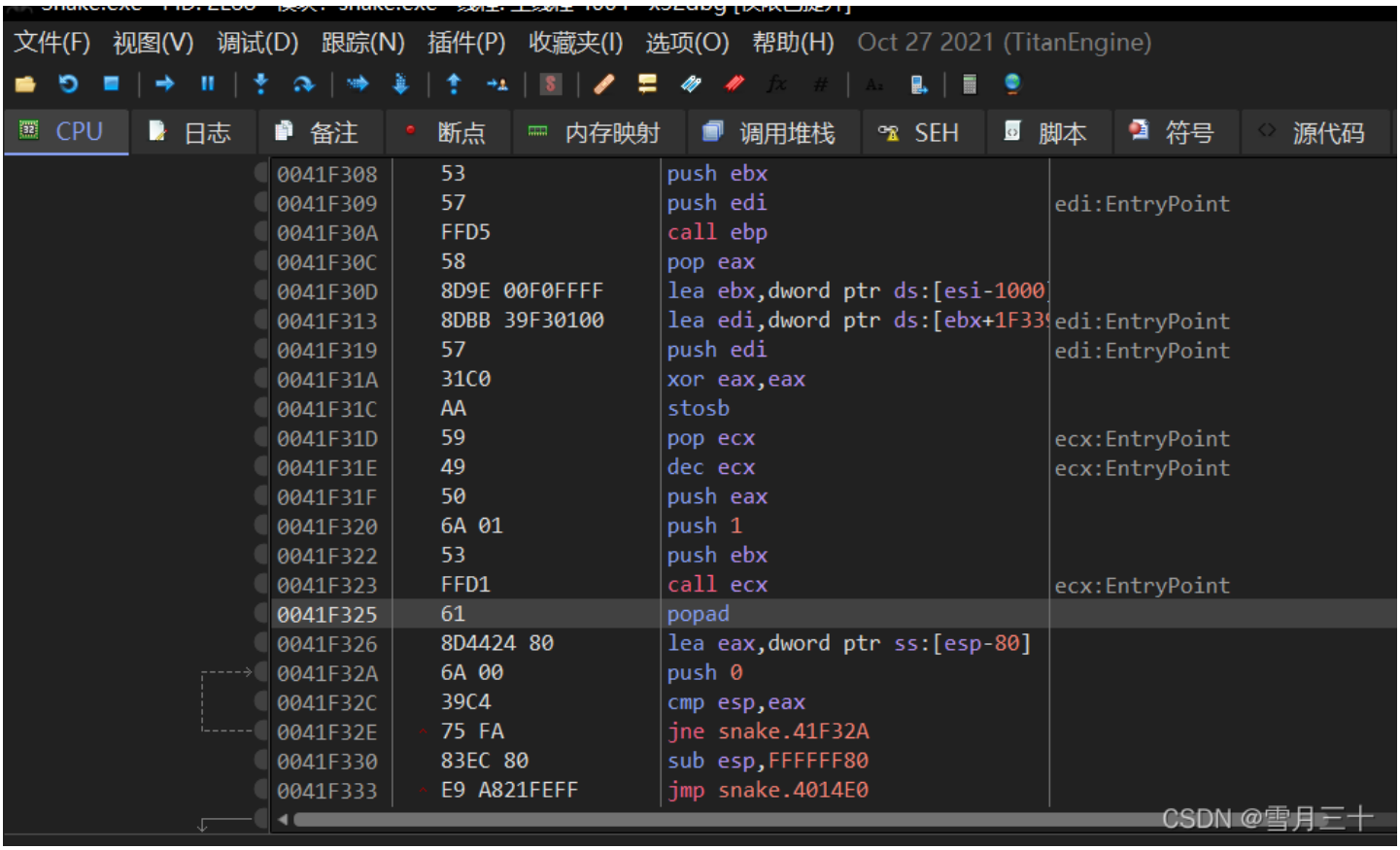



**运行到ret C后** 直接单步步入ret C函数内
然后重复执行这样的步骤就行了



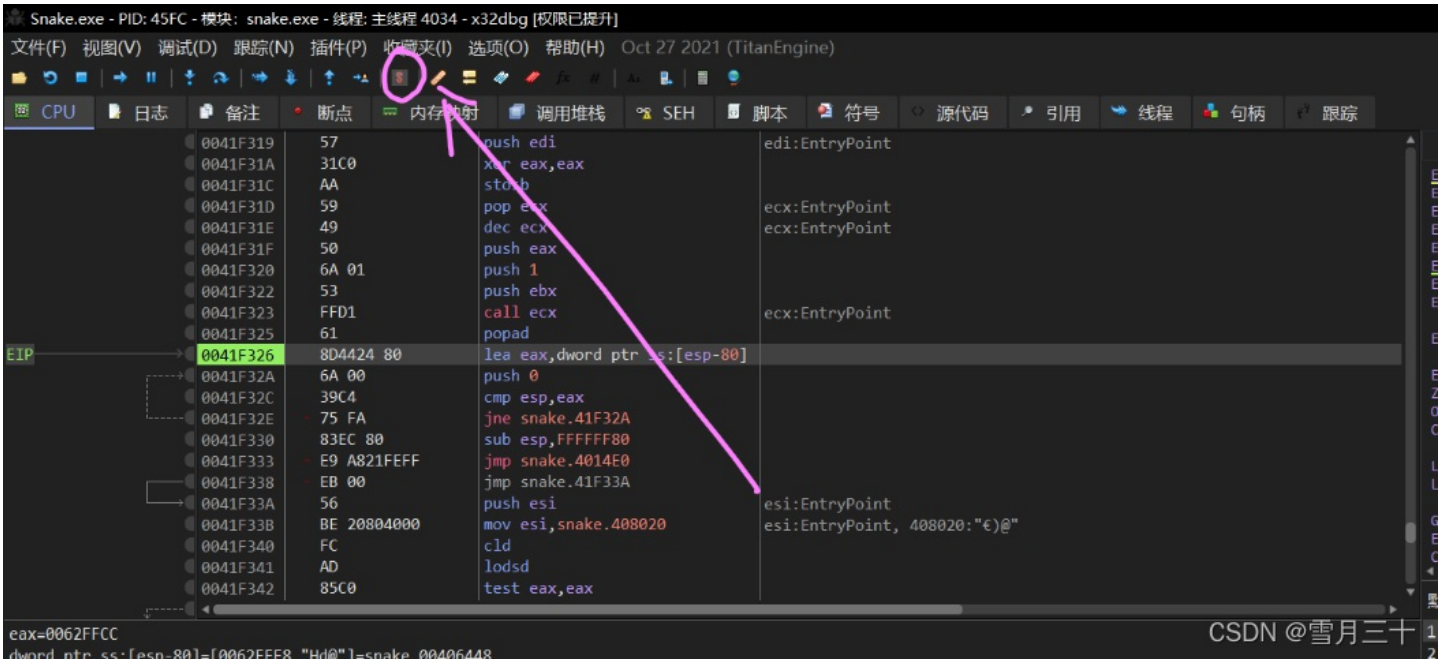




重复两三次就可以看到popad指令也就是结束了壳的程序的、

而且也看到了**（lea rax,qword ptr ss:[rsp-80]）**其实这个就是程序的入口点
我们在单步步过一下让eip来到我们的入口
开始来dump我们的程序

点击我们dbg自带的插件

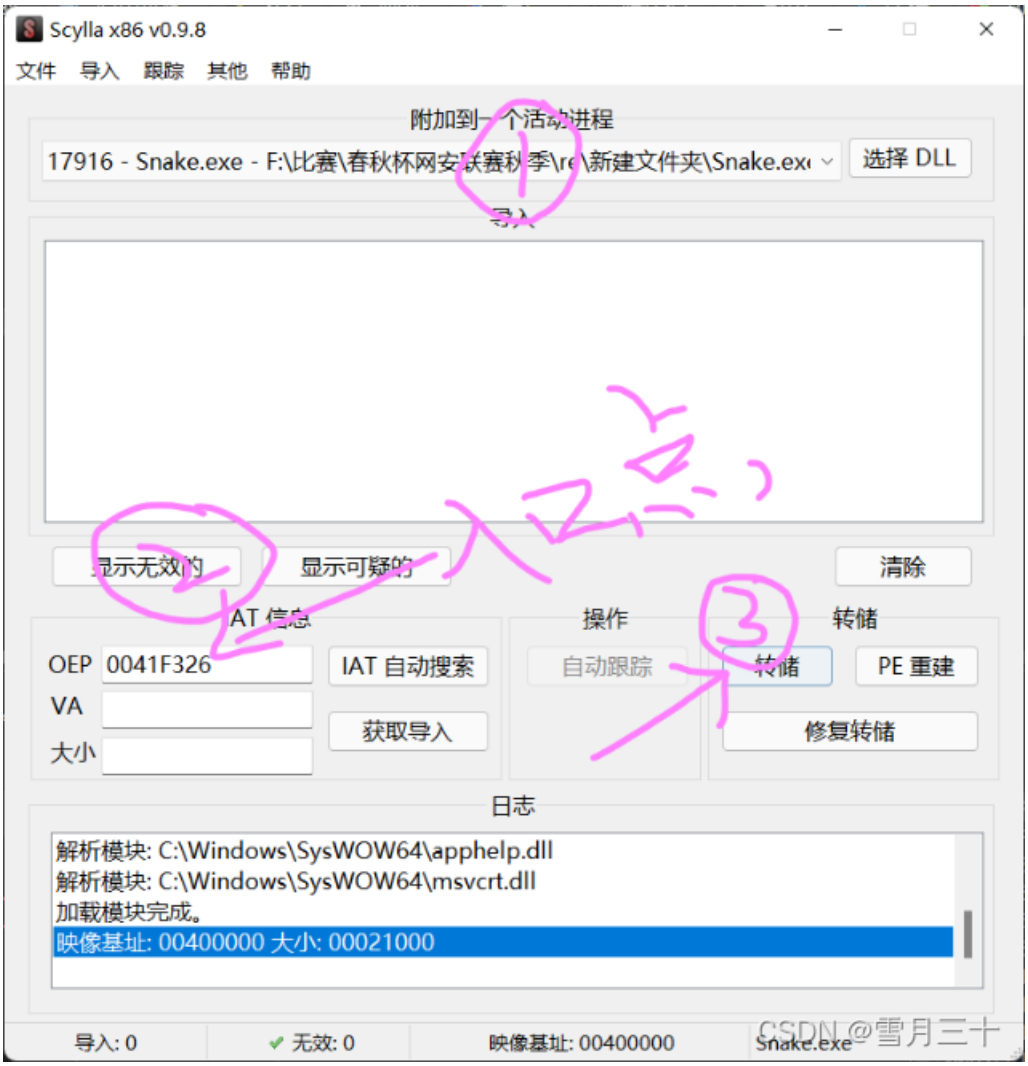



可以看到我们插件已经自动帮助我们调好了信息

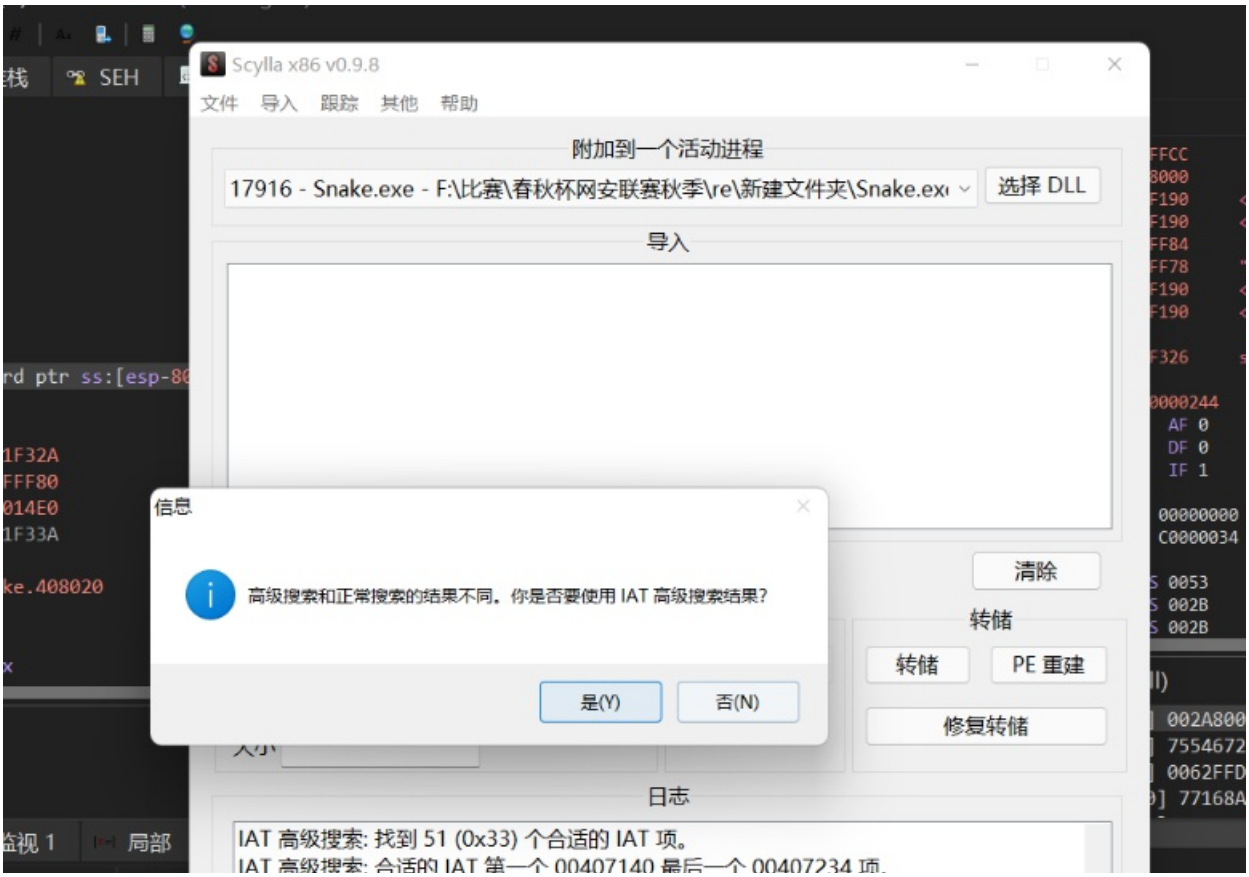
1、当前的进程 可以看到插件已经帮助我们找到了

2、程序的入口点（也就是我们手动运行到的点）

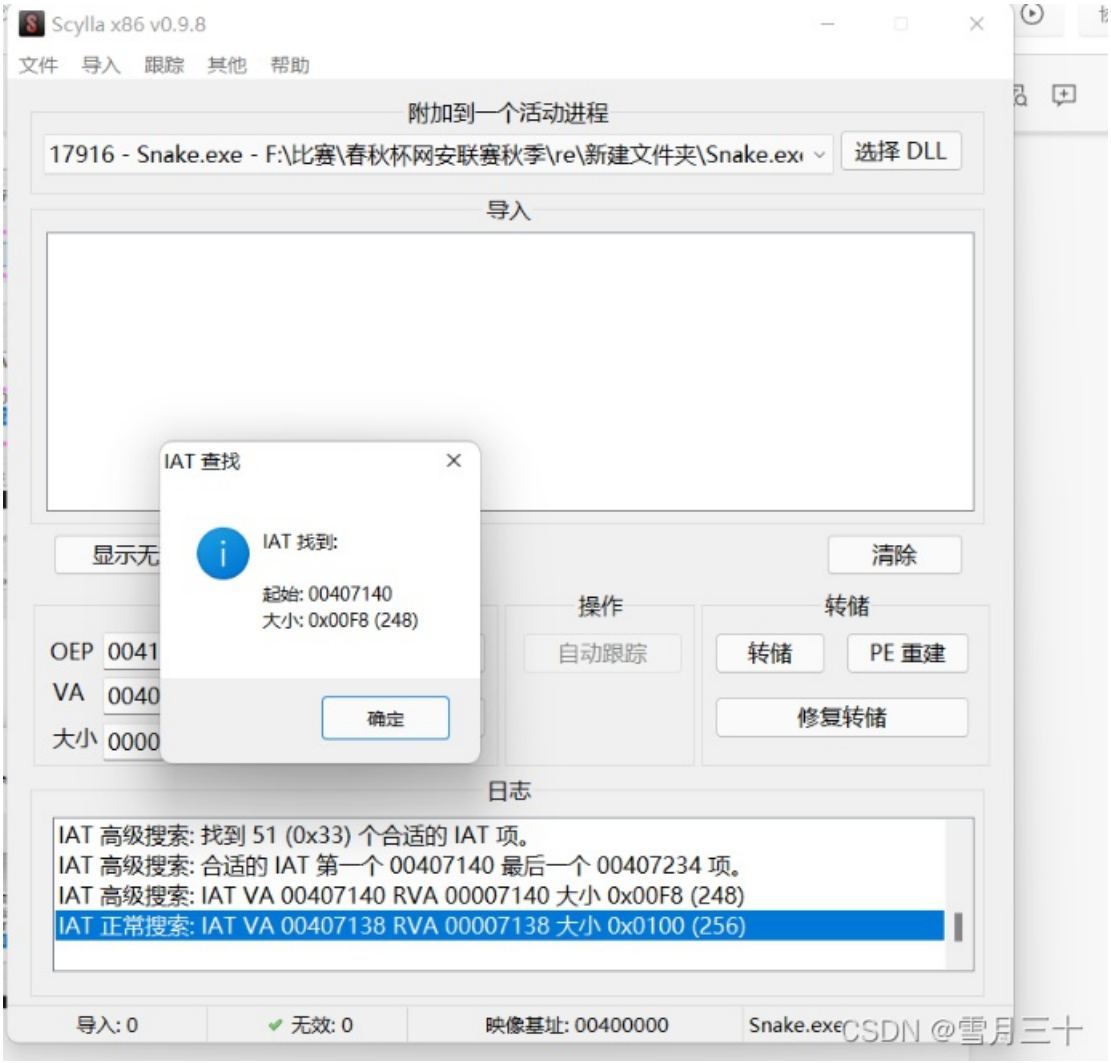
3、其实上面的的信息插件都已经帮我们配置好 我们只需点击转储即可



保存文件后再单击IAT自动搜索，修复IAT

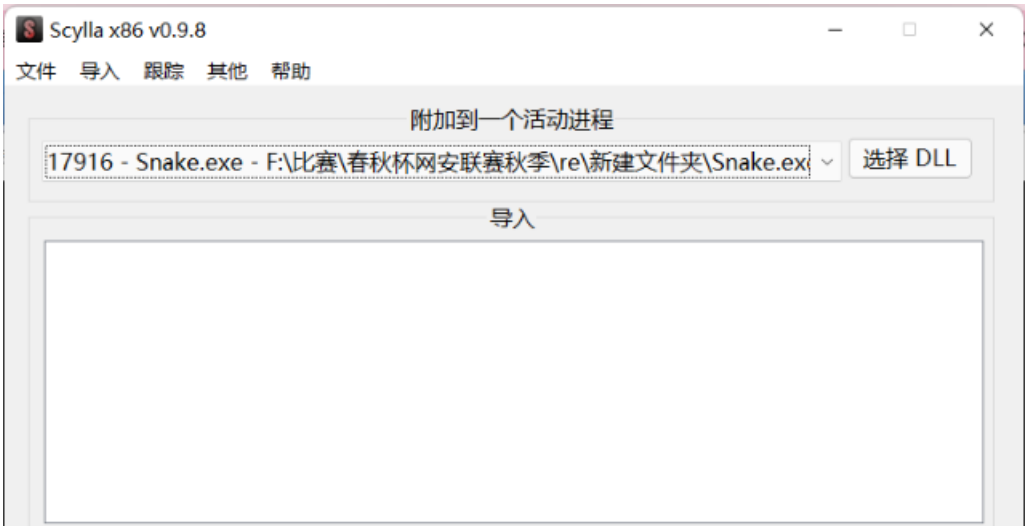



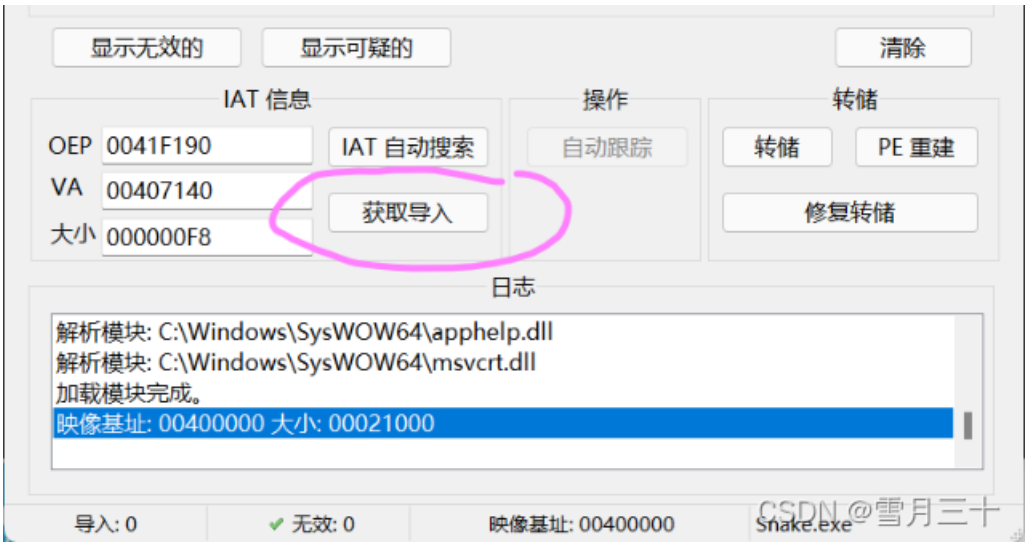

确定



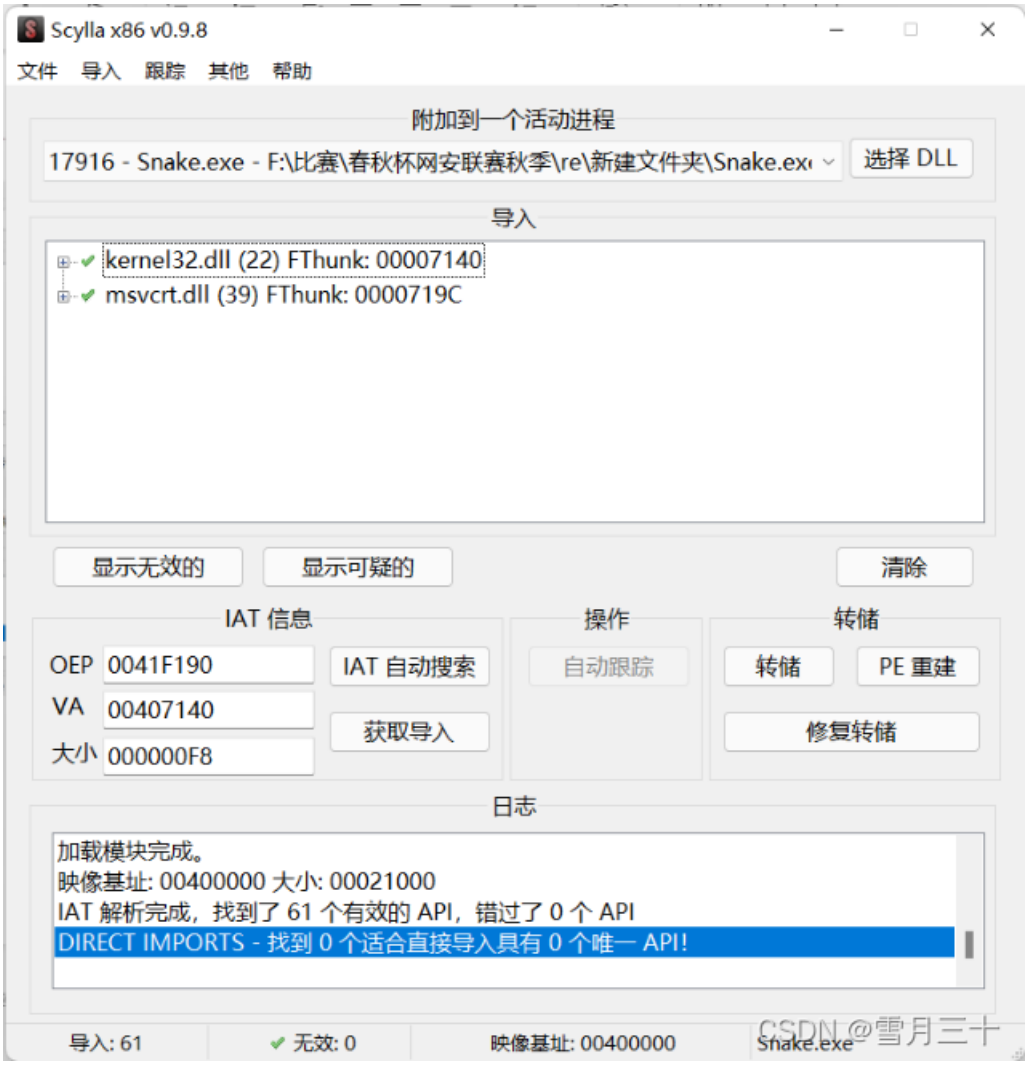

可以看到很幸运找到了IAT的地址

点击获取导入



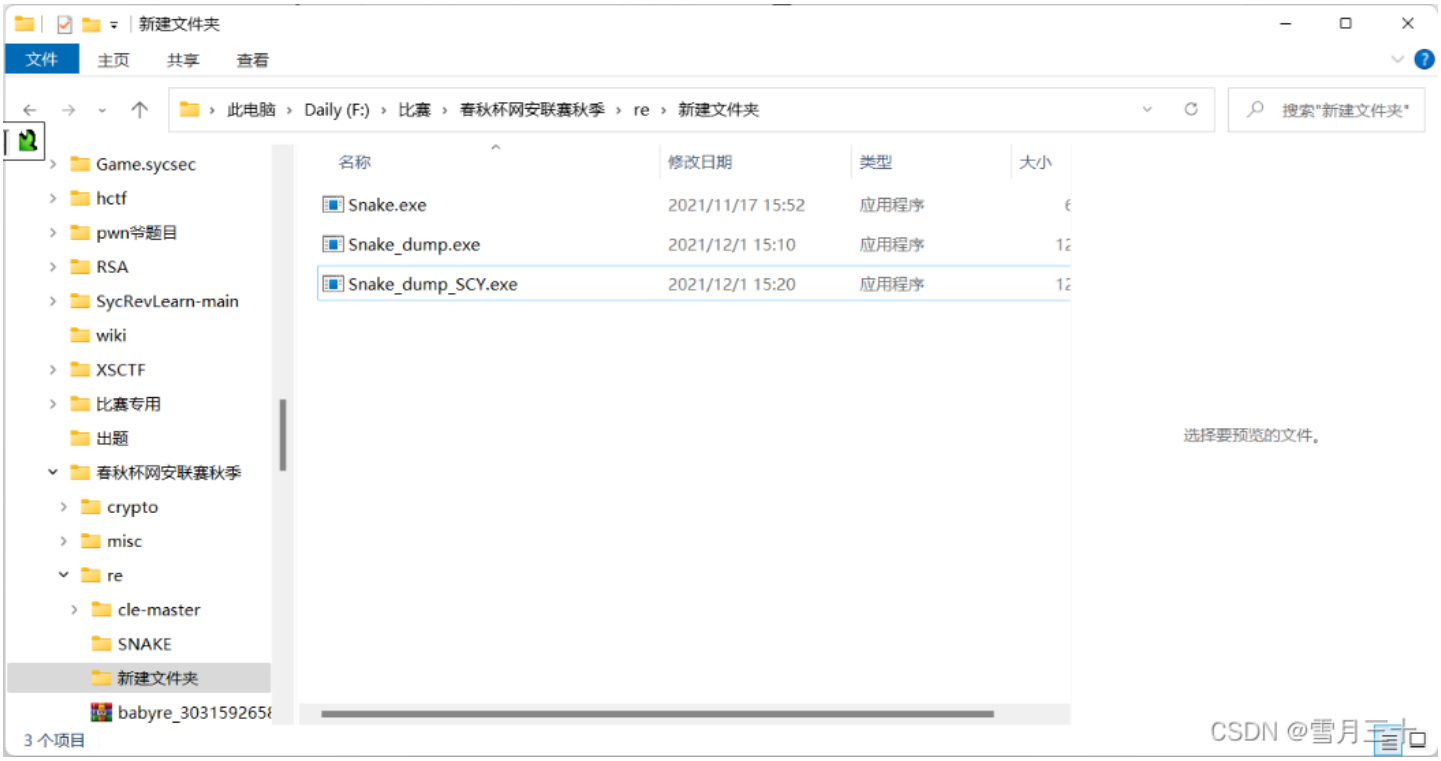



可以看到函数没有异常

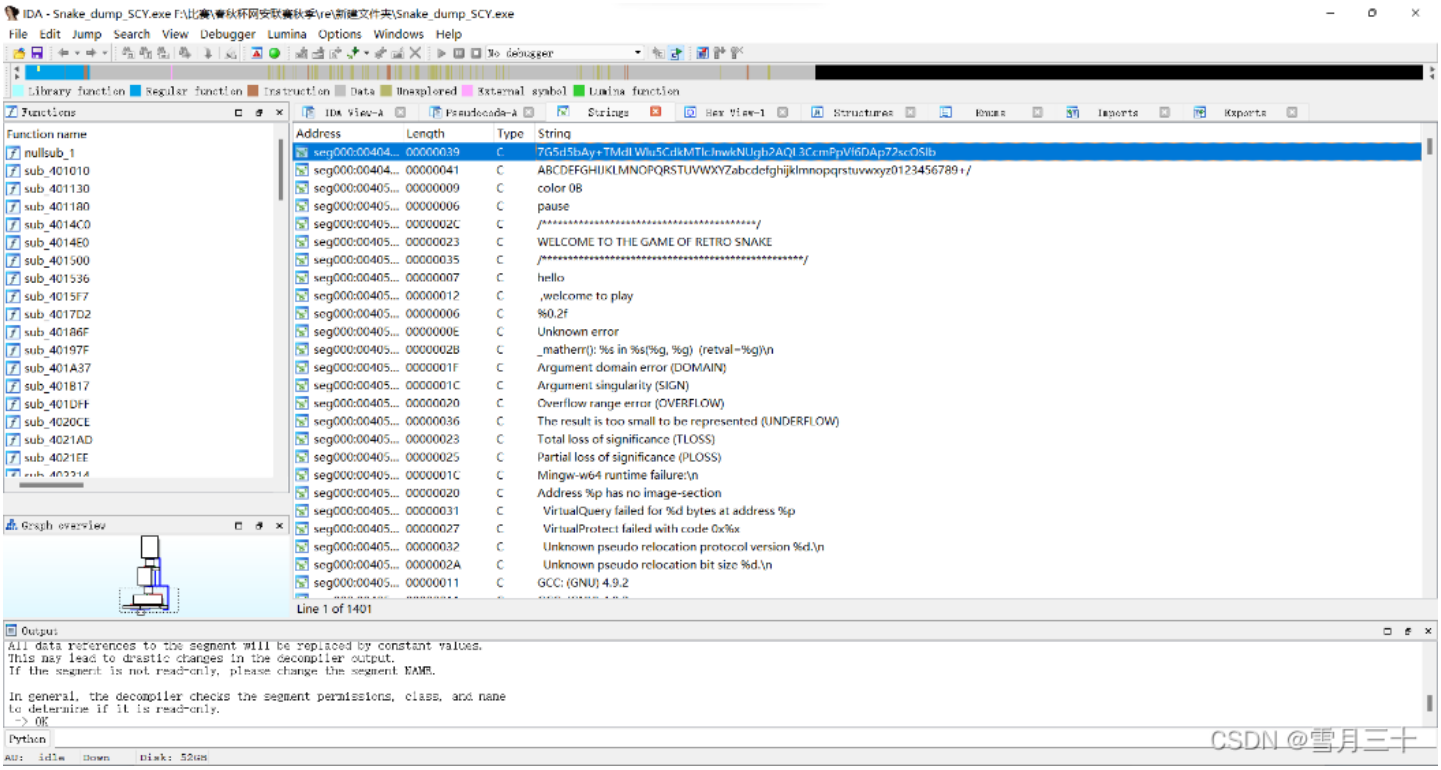



选择我们之前刚存储的文件点击打开

到这里会生成一个SCY文件也就完成脱壳而且修复之后的文件了

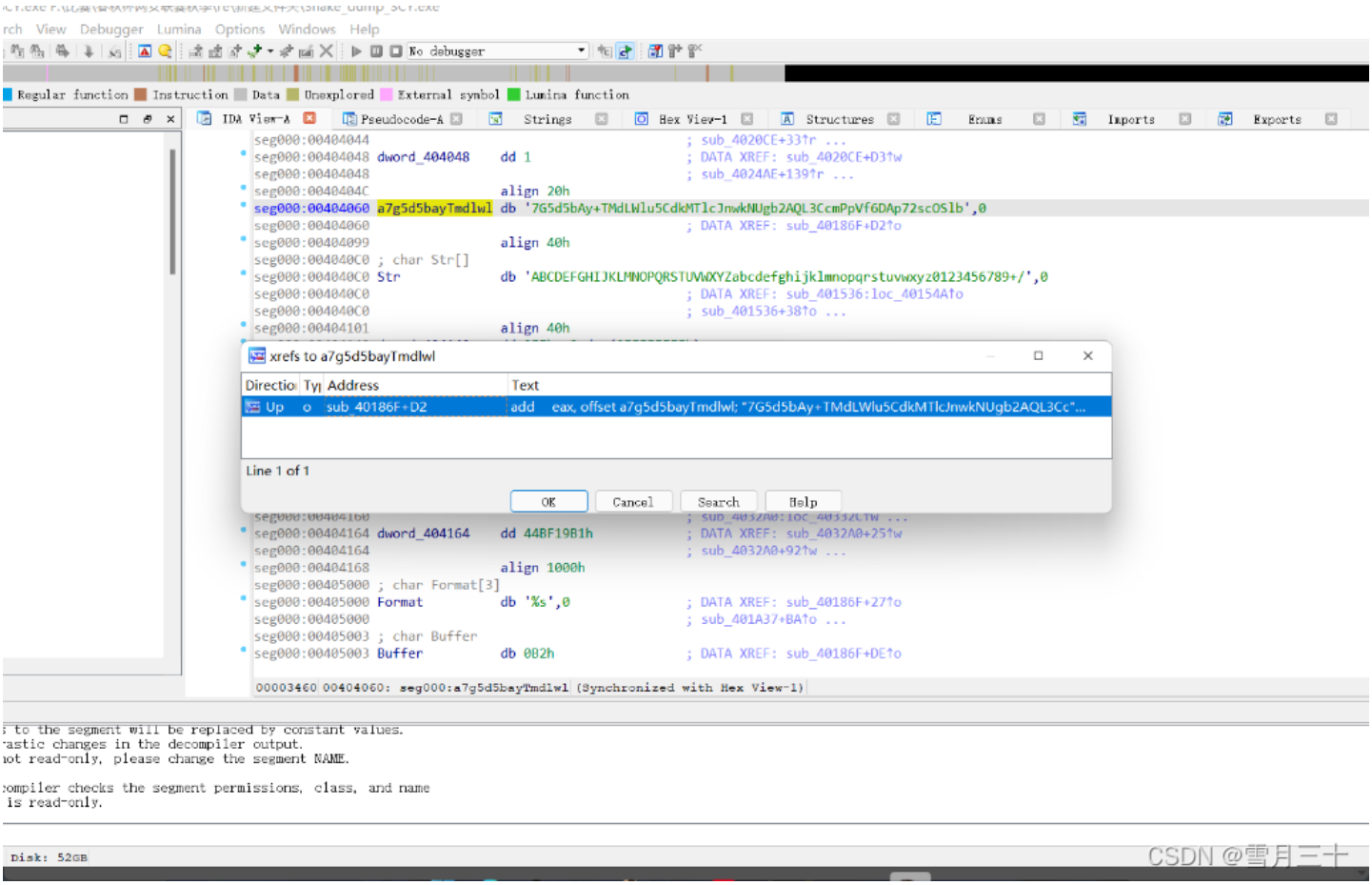



## 静态分析



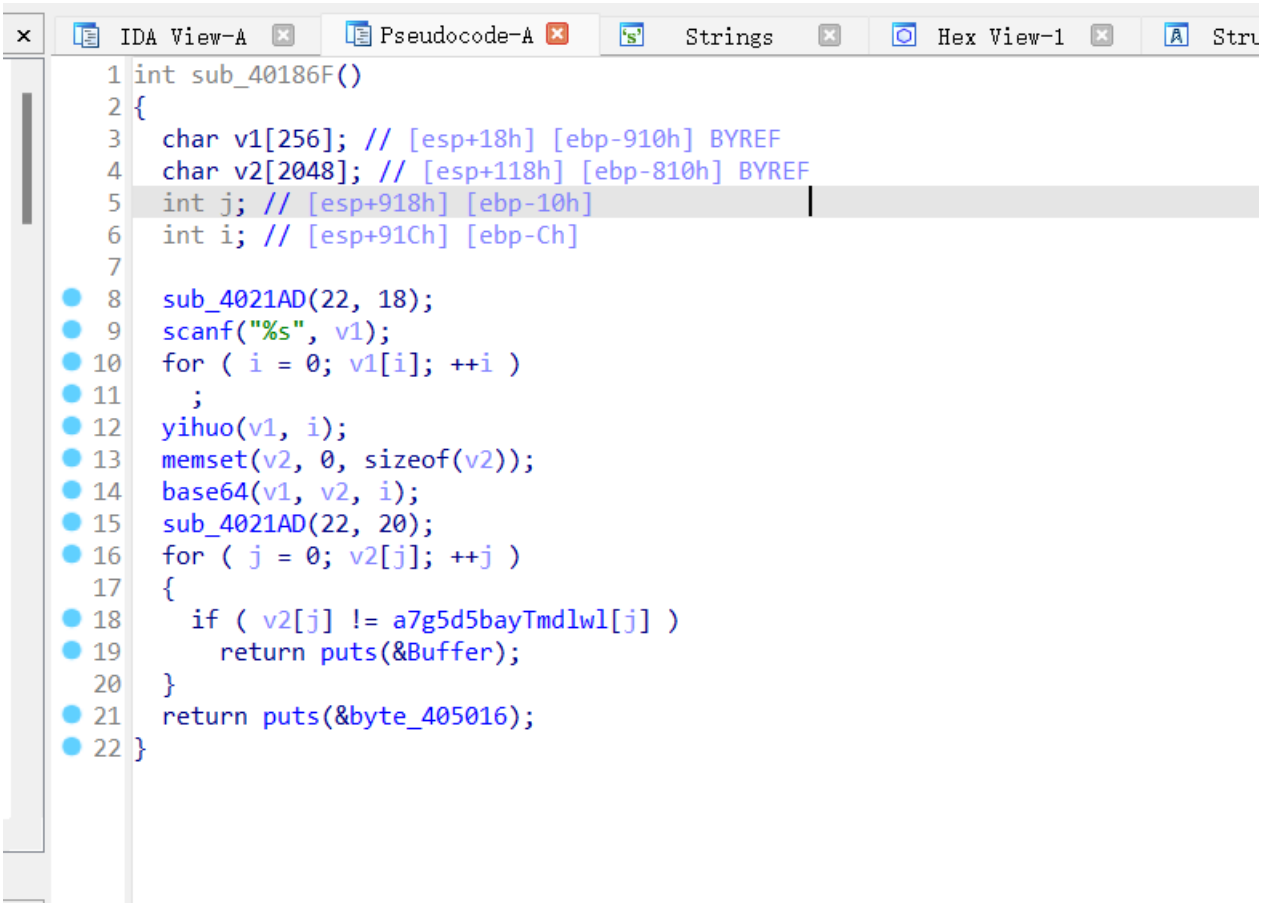

搜索字符串 可以看到第一行有可疑的字符串出现
第二行是明显的base表可以猜测是有base解密的

双击查看 按下x查看引用




进入之后反编译可以看到我们的主函数所在位置

分析函数



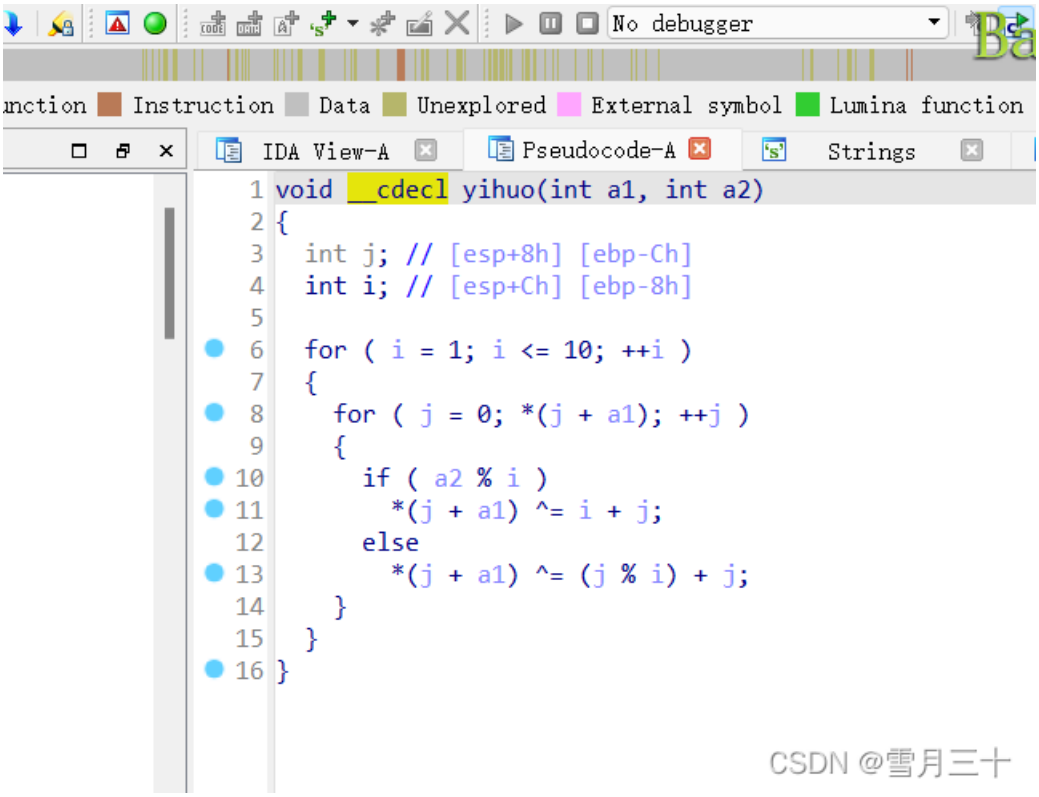

```
int sub_40186F()
{
  char v1[256]; // [esp+18h] [ebp-910h] BYREF
  char v2[2048]; // [esp+118h] [ebp-810h] BYREF
  int j; // [esp+918h] [ebp-10h]
  int i; // [esp+91Ch] [ebp-Ch]

  sub_4021AD(22, 18);
  scanf("%s", v1);
  for ( i = 0; v1[i]; ++i )
    ;
  yihuo(v1, i);
  memset(v2, 0, sizeof(v2));
  base64(v1, v2, i);
  sub_4021AD(22, 20);
  for ( j = 0; v2[j]; ++j )
  {
    if ( v2[j] != a7g5d5bayTmdlwl[j] )
      return puts(&Buffer);
  }
  return puts(&byte_405016);
}
```

其实主要是就是一个异或的加密其他的都是很简单的

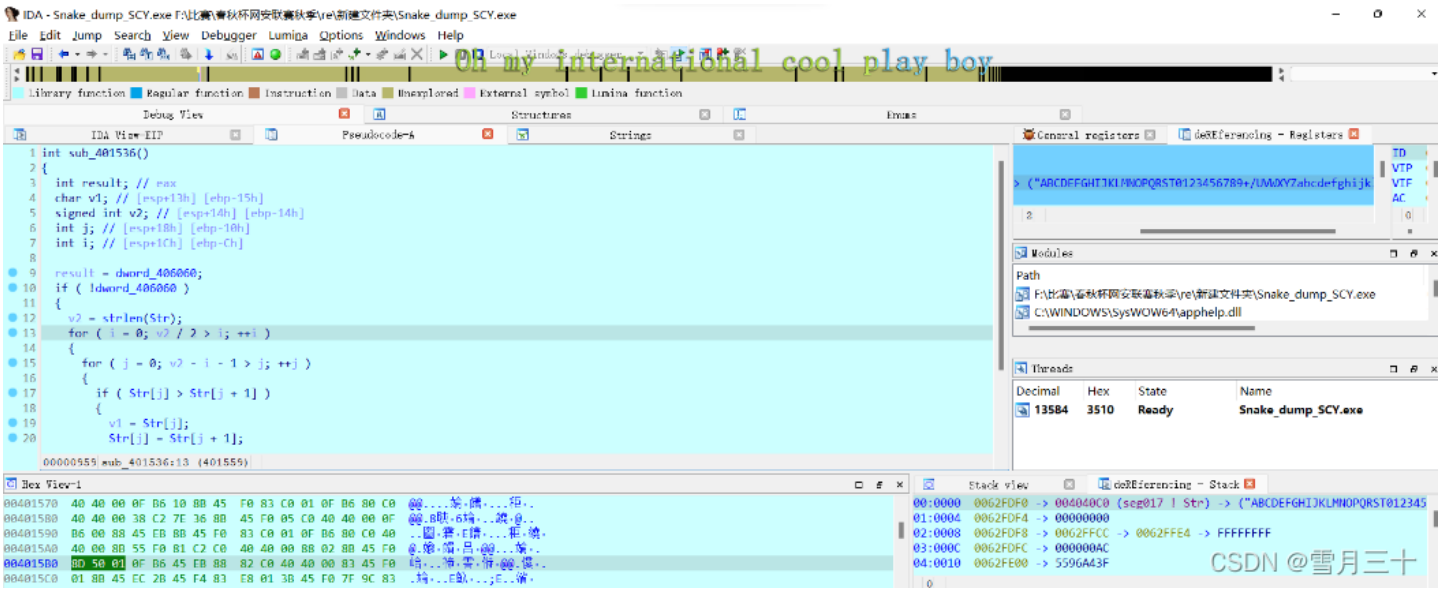



这个异或函数中间写的有点多余 其实我们应该可以猜测是因为函数的返回值问题
我们将鼠标指针移动到函数名上按下v可以让ida插件自动取猜测这个函数的真实类型



这样看起来就舒服很多了

分析完主函数刚才我们还看到了base表我们同样的方法去查看一下




这样可以直接看到最后的base表

接下来就是脚本时间了

# 算法逆向

## base64换表

```
import base64
import string
string = "7G5d5bAy+TMdLWlu5CdkMTlcJnwkNUgb2AQL3CcmPpVf6DAp72scOSlb"
tableBase64 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
tableNew = "ABCDEFGHIJKLMNOPQRST0123456789+/UVWXYZabcdefghijklmnopqrstuvwxyz"
'''
maketrans()：用于创建字符映射的转换表，对于接受两个参数的最简单的调用方式，第一个参数是字符串，表示需要转换的字符，第二个参数也是字符串表示转换的目标；
translate()：法根据参数table给出的表(包含 256 个字符)转换字符串的字符，要过滤掉的字符放到 del 参数中；
decode()：以encoding指定的编码格式解码字符串。
'''
'1.换表'
maketrans = str.maketrans(tableNew, tableBase64)
'2.使用新表转换字符串'
translate = string.translate(maketrans)
print(translate)
'2.Base64解码'
flag = base64.b64decode(translate)
'''
三合一操作：
flag = base64.b64decode(string.translate(str.maketrans(tableNew, tableBase64)))
'''
#flag = base64.b64decode(string.translate(str.maketrans(tableNew, tableBase64)))
flag = list(flag)
print(flag)
#bGZpZnA+eTMpLix6ZCpwMTxoJz8wNgsnWAQLXCoyP1hraDA1bW4oOSxn
#[108, 102, 105, 102, 112, 62, 121, 51, 41, 46, 44, 122, 100, 42, 112, 49, 60, 104, 39, 63, 48, 54, 11, 39, 88, 4, 11, 92, 42, 50, 63, 88, 107, 104, 48, 53, 109, 110, 40, 57, 44, 103]
```

## 异或

```
#include <stdio.h>
int main()
{
 int i, j, a1[43] = {108, 102, 105, 102, 112, 62, 121, 51, 41, 46, 44, 122, 100, 42, 112, 49, 60, 104, 39, 63, 4
8, 54, 11, 39, 88, 4, 11, 92, 42, 50, 63, 88, 107, 104, 48, 53, 109, 110, 40, 57, 44, 103};
 for (i =1; i <= 10; i++)
 {
  for (j = 0; j < 42; j++)
  {
   if (42 % i) //1 2 3 6 7
    a1[j] ^= i + j; // a1[j] = a1[j] ^ (i + j)
   else
   {
    a1[j] ^= (j % i) + j; // a1[j] = a1[j] ^ ((j % i) + j)
    //printf("j=%d,%d\n",j,a1[j]);
   }
  }
 }
 for (i = 0; i < 42; i++)
  printf("%c,", a1[i]);
}
//f,l,a,g,{,5,e,2,2,0,0,b,c,-,f,2,1,a,-,5,4,2,1,-,a,9,0,b,-,5,7,d,e,c,1,9,f,e,1,9,6,
```