

# 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](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: [https://blog.csdn.net/weixin\\_49764009/article/details/121660322](https://blog.csdn.net/weixin_49764009/article/details/121660322)

版权

## 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

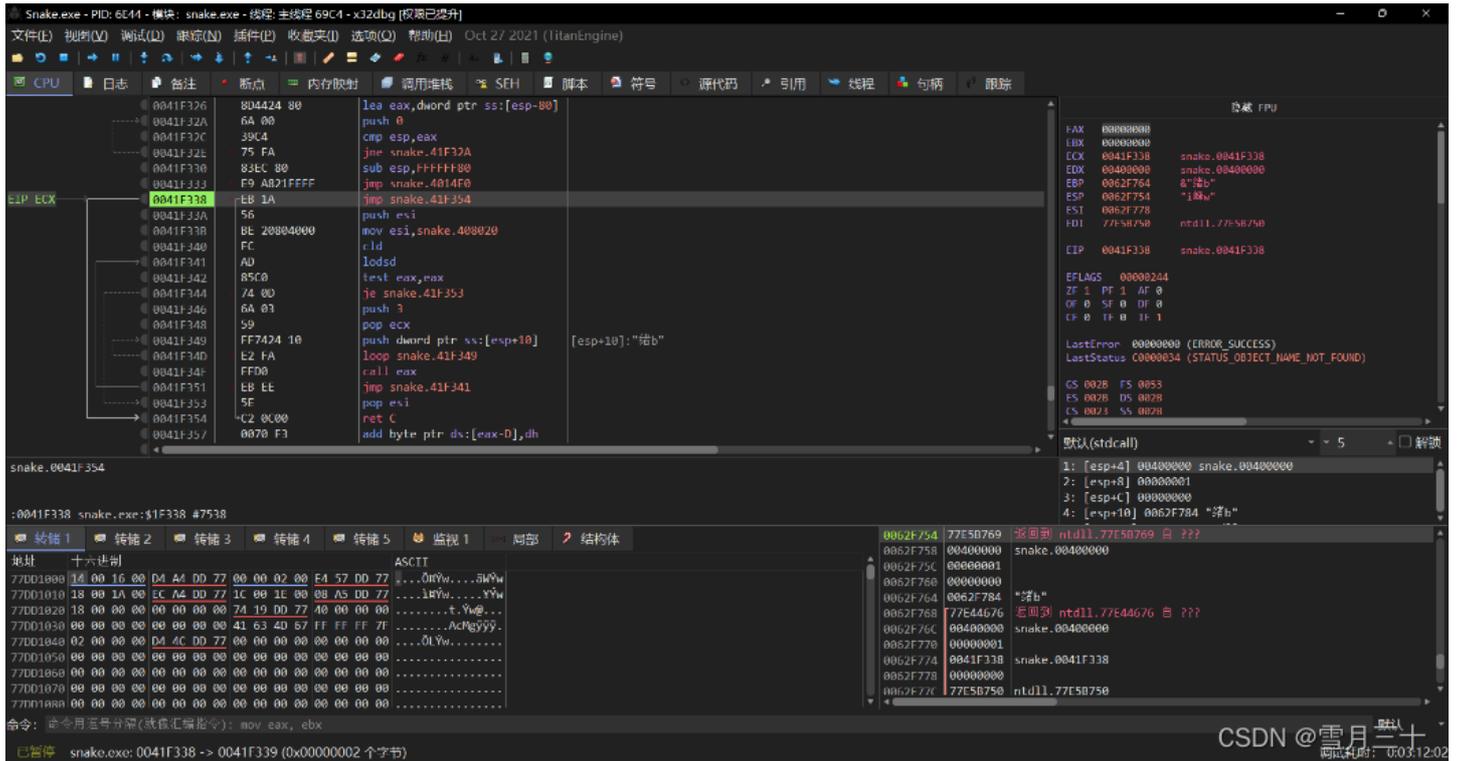


32位 UPX(-)[modified]壳 并不是upx壳

## 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

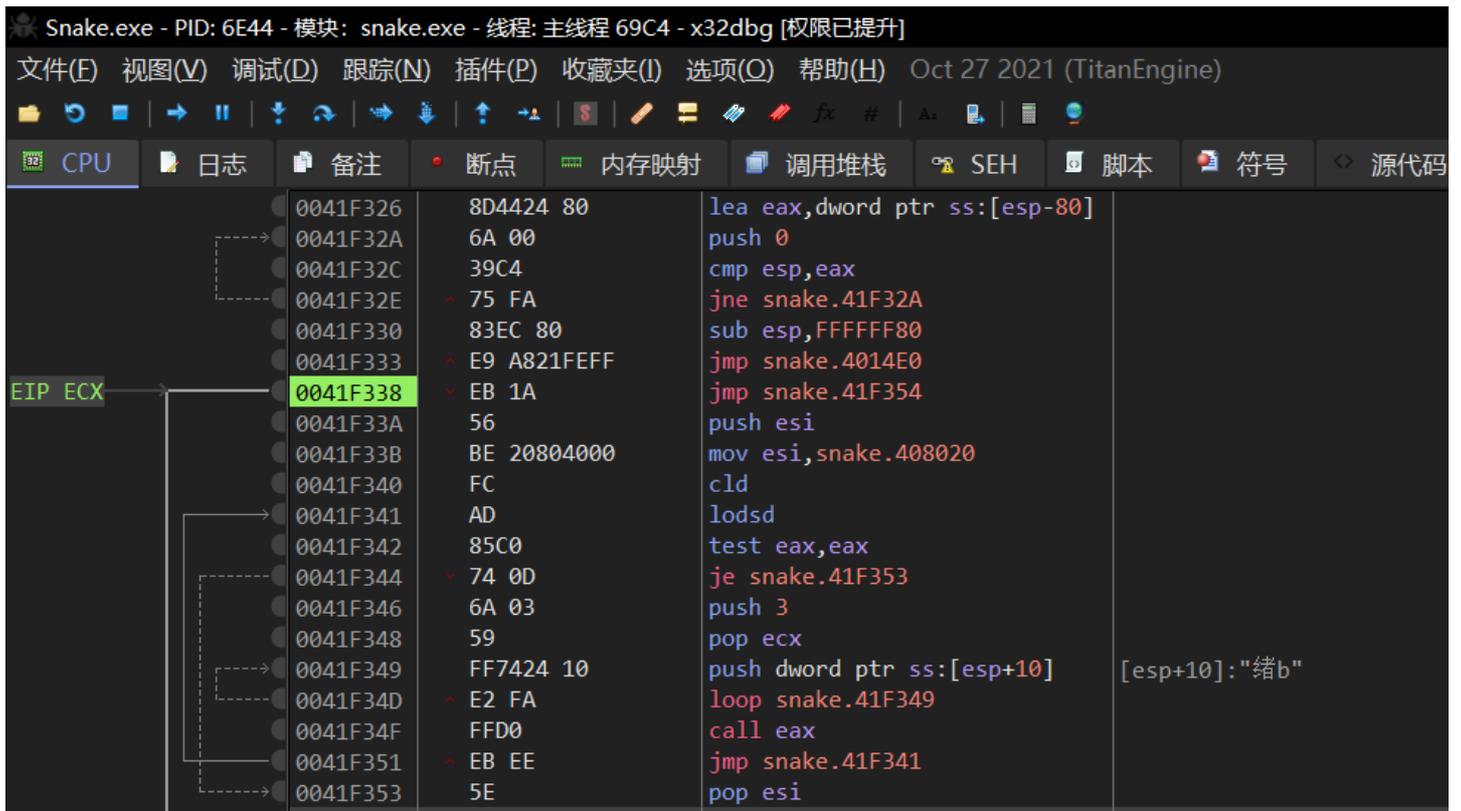
所以这些代码还不是程序真正的代码我们需要手动来脱一下这个upx壳

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：



0041F354	C2 0C00	ret C
0041F357	0070 F3	add byte ptr ds:[eax-D],dh

CSDN @雪月三十

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

0041F330	B3EC 80	sub esp,FFFFFF80
0041F333	E9 A821FEFF	jmp snake.4014E0
0041F338	EB 1A	jmp snake.41F354
0041F33A	56	push esi
0041F33B	BE 20804000	mov esi,snake.408020
0041F340	FC	cld
0041F341	AD	lodsd
0041F342	85C0	test eax,eax
0041F344	74 0D	je snake.41F353
0041F346	6A 03	push 3
0041F348	59	pop ecx
0041F349	FF7424 10	push dword ptr ss:[esp+10] [esp+10]: "结b"
0041F34D	E2 FA	loop snake.41F349
0041F34F	FFD0	call eax
0041F351	EB EE	jmp snake.41F341
0041F353	5E	pop esi
0041F354	C2 0C00	ret C
0041F357	0070 F3	add byte ptr ds:[eax-D],dh

CSDN @雪月三十

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

Snake.exe - PID: 2E88 - 模块: ntdll.dll - 线程: 主线程 4004 - x32dbg [权限已提升]

文件(F) 视图(V) 调试(D) 跟踪(N) 插件(P) 收藏夹(I) 选项(O) 帮助(H) Oct 27 2021 (TitanE)

CPU 日志 备注 断点 内存映射 调用堆栈 SEH 脚本

EIP	778EB769	33C0	xor eax,eax
	778EB76B	40	inc eax
	778EB76C	5D	pop ebp
	778EB76D	C2 0C00	ret C
	778EB770	CC	int3
	778EB771	CC	int3
	778EB772	CC	int3
	778EB773	CC	int3
	778EB774	CC	int3
	778EB775	CC	int3
	778EB776	8BFF	mov edi,edi
	778EB778	55	push ebp
	778EB779	8BEC	mov ebp,esp
	778EB77B	83EC 14	sub esp,14
	778EB77E	A1 60D39877	mov eax,dword ptr ds:[7798D360]
	778EB783	33C5	xor eax,ebp
	778EB785	8945 FC	mov dword ptr ss:[ebp-4],eax
	778EB788	56	push esi
	778EB789	33F6	xor esi,esi
	778EB78B	C745 F0 544C5348	mov dword ptr ss:[ebp-10],485348
	778EB792	C745 F4 65617000	mov dword ptr ss:[ebp-C],70617000
	778EB799	3935 C4809877	cmp dword ptr ds:[779880C4],esi

eax=0

CSDN @雪月三十

Snake.exe - PID: 2E88 - 模块: ntdll.dll - 线程: 主线程 4004 - x32dbg [权限已提升]

文件(F) 视图(V) 调试(D) 跟踪(N) 插件(P) 收藏夹(I) 选项(O) 帮助(H) Oct 27 2021 (TitanE)

CPU 日志 备注 断点 内存映射 调用堆栈 SEH 脚本

EIP	778EB769	33C0	xor eax,eax
	778EB76B	40	inc eax
	778EB76C	5D	pop ebp
	778EB76D	C2 0C00	ret C
	778EB770	CC	int3
	778EB771	CC	int3
	778EB772	CC	int3
	778EB773	CC	int3
	778EB774	CC	int3
	778EB775	CC	int3
	778EB776	8BFF	mov edi,edi
	778EB778	55	push ebp
	778EB779	8BEC	mov ebp,esp
	778EB77B	83EC 14	sub esp,14
	778EB77E	A1 60D39877	mov eax,dword ptr ds:[7798D360]
	778EB783	33C5	xor eax,ebp
	778EB785	8945 FC	mov dword ptr ss:[ebp-4],eax
	778EB788	56	push esi
	778EB789	33F6	xor esi,esi
	778EB78B	C745 F0 544C5348	mov dword ptr ss:[ebp-10],485348
	778EB792	C745 F4 65617000	mov dword ptr ss:[ebp-C],70617000
	778EB799	3935 C4809877	cmp dword ptr ds:[779880C4],esi

eax=0

CSDN @雪月三十

	778EB769	33C0	xor eax,eax	
	778EB76B	40	inc eax	
	778EB76C	5D	pop ebp	
EIP	778EB76D	C2 0C00	ret C	
	778EB770	CC	int3	
	778EB771	CC	int3	
	778EB772	CC	int3	
	778EB773	CC	int3	
	778EB774	CC	int3	
	778EB775	CC	int3	

CSDN @雪月三十

Snake.exe - PID: 2E88 - 模块: snake.exe - 线程: 主线程 4004 - x32dbg [权限已提升]

文件(F) 视图(V) 调试(D) 跟踪(N) 插件(P) 收藏夹(I) 选项(O) 帮助(H) Oct 27 2021 (TitanEngine)

CPU 日志 备注 断点 内存映射 调用堆栈 SEH 脚本 符号

EIP	ECX	EDX	ESI	EDI	0041F190	60	pushad	EntryPoint
					0041F191	BE 15804100	mov esi,snake.418015	esi:EntryPoint
					0041F196	8DBE EB8FFEFF	lea edi,dword ptr ds:[esi-1701]	edi:EntryPoint
					0041F19C	57	push edi	edi:EntryPoint
					0041F19D	83CD FF	or ebp,FFFFFFFF	
					0041F1A0	EB 10	jmp snake.41F1B2	
					0041F1A2	90	nop	
					0041F1A3	90	nop	
					0041F1A4	90	nop	

CSDN @雪月三十

Snake.exe - PID: 2E88 - 模块: snake.exe - 线程: 主线程 4004 - x32dbg [权限已提升]

文件(F) 视图(V) 调试(D) 跟踪(N) 插件(P) 收藏夹(I) 选项(O) 帮助(H) Oct 27 2021 (TitanEngine)

CPU 日志 备注 断点 内存映射 调用堆栈 SEH 脚本 符号 源代码

	0041F308	53	push ebx	
	0041F309	57	push edi	edi:EntryPoint
	0041F30A	FFD5	call ebp	
	0041F30C	58	pop eax	
	0041F30D	8D9E 00F0FFFF	lea ebx,dword ptr ds:[esi-1000]	
	0041F313	8DBB 39F30100	lea edi,dword ptr ds:[ebx+1F335]	edi:EntryPoint
	0041F319	57	push edi	edi:EntryPoint
	0041F31A	31C0	xor eax,eax	
	0041F31C	AA	stosb	
	0041F31D	59	pop ecx	ecx:EntryPoint
	0041F31E	49	dec ecx	ecx:EntryPoint
	0041F31F	50	push eax	
	0041F320	6A 01	push 1	
	0041F322	53	push ebx	
	0041F323	FFD1	call ecx	ecx:EntryPoint
	0041F325	61	popad	
	0041F326	8D4424 80	lea eax,dword ptr ss:[esp-80]	
	0041F32A	6A 00	push 0	
	0041F32C	39C4	cmp esp,eax	
	0041F32E	75 FA	jne snake.41F32A	
	0041F330	83EC 80	sub esp,FFFFFF80	
	0041F333	E9 A821FEFF	jmp snake.4014E0	

CSDN @雪月三十

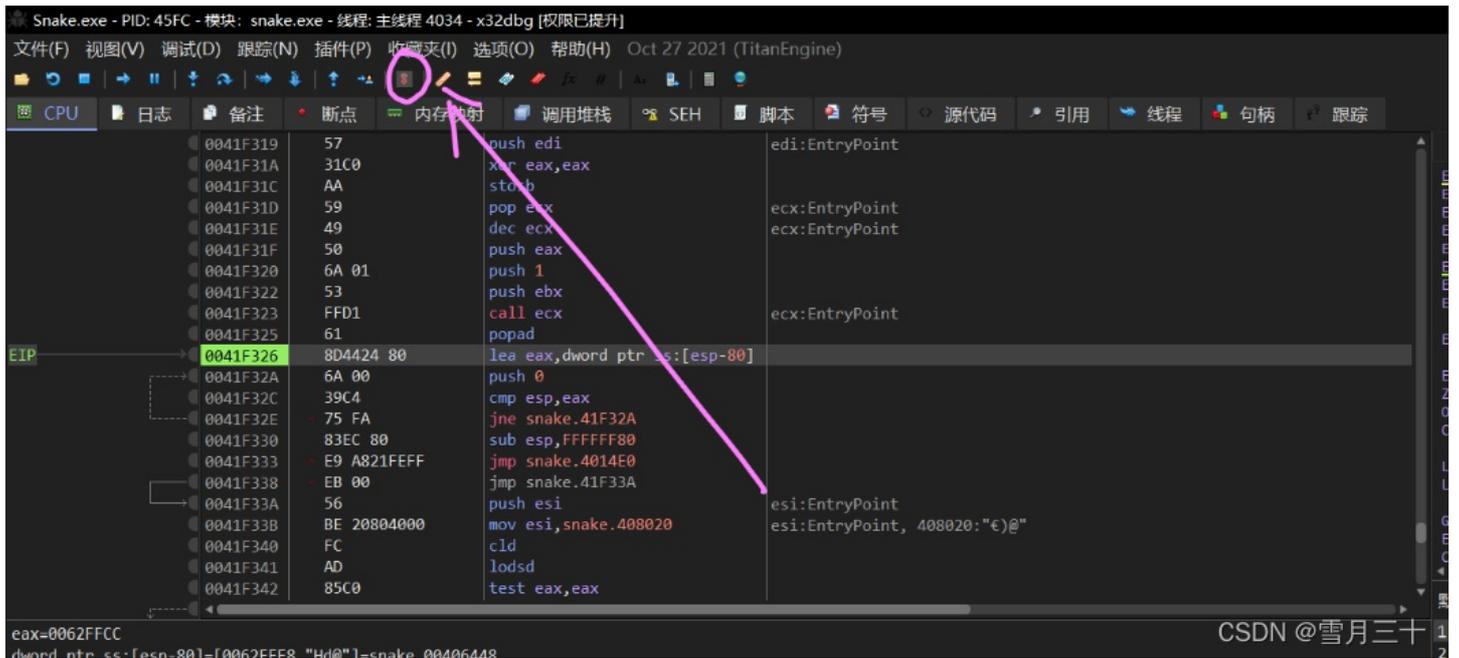
重复两三次就可以看到popad指令也就是结束了壳的程序的、

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

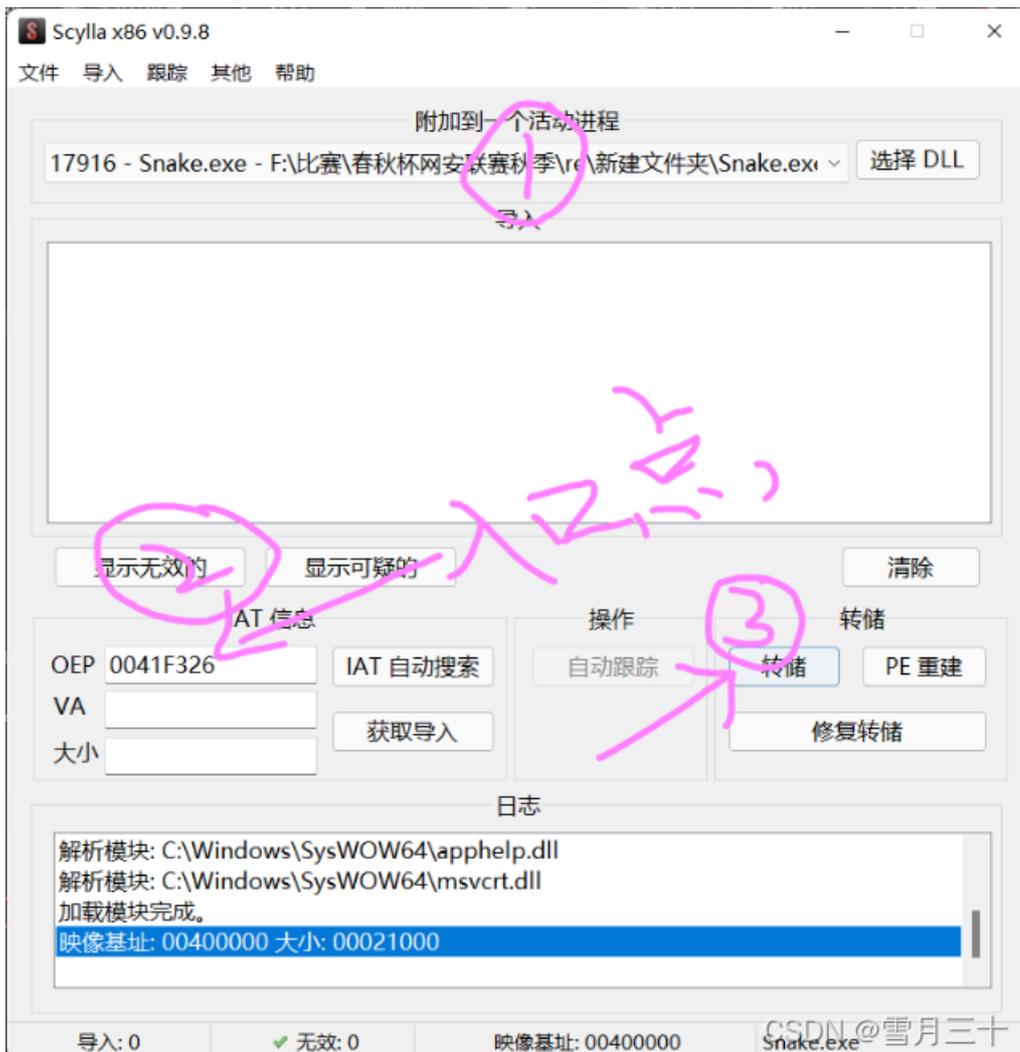
开始来dump我们的程序

点击我们dbg自带的插件

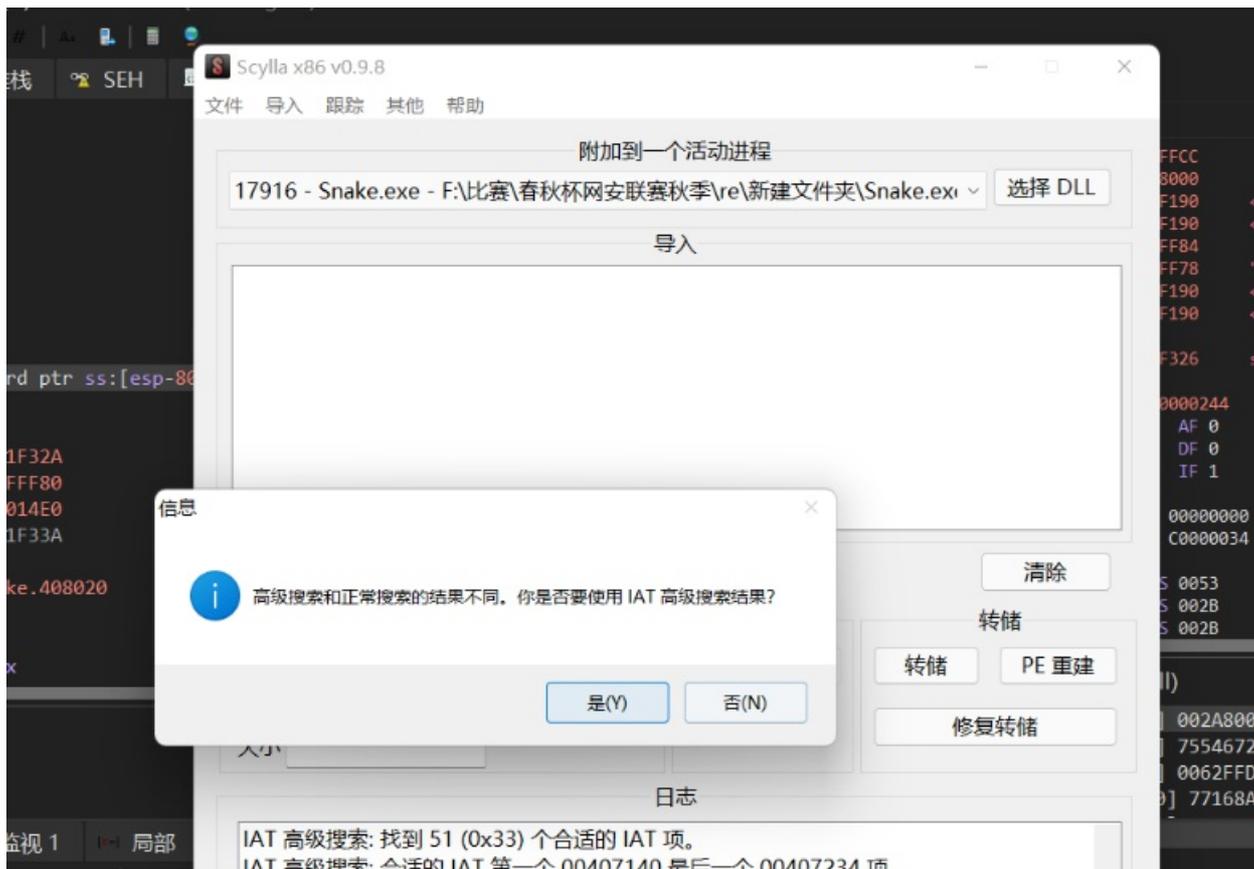


可以看到我们插件已经自动帮助我们调好了信息

- 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

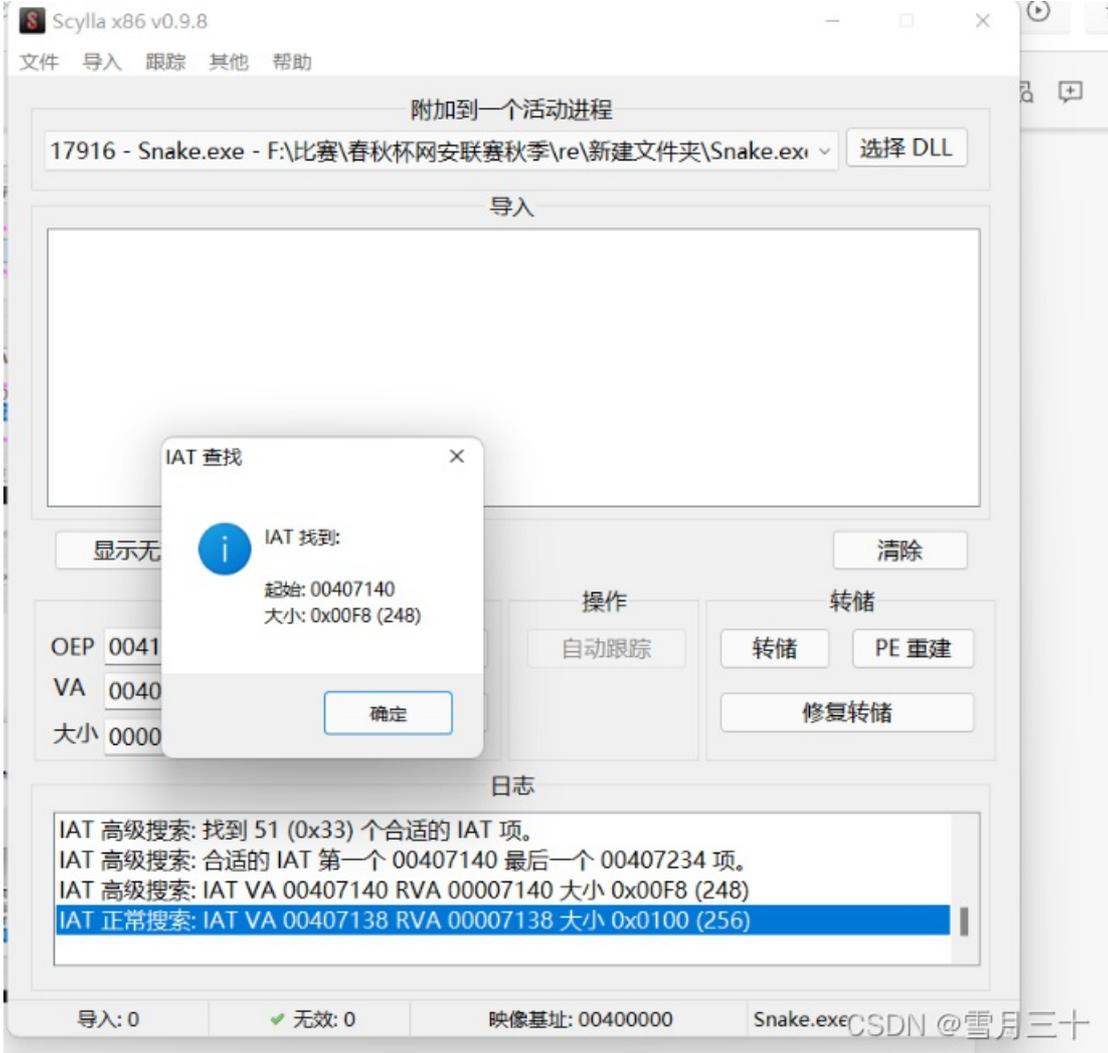


保存文件后再单击IAT自动搜索，修复IAT



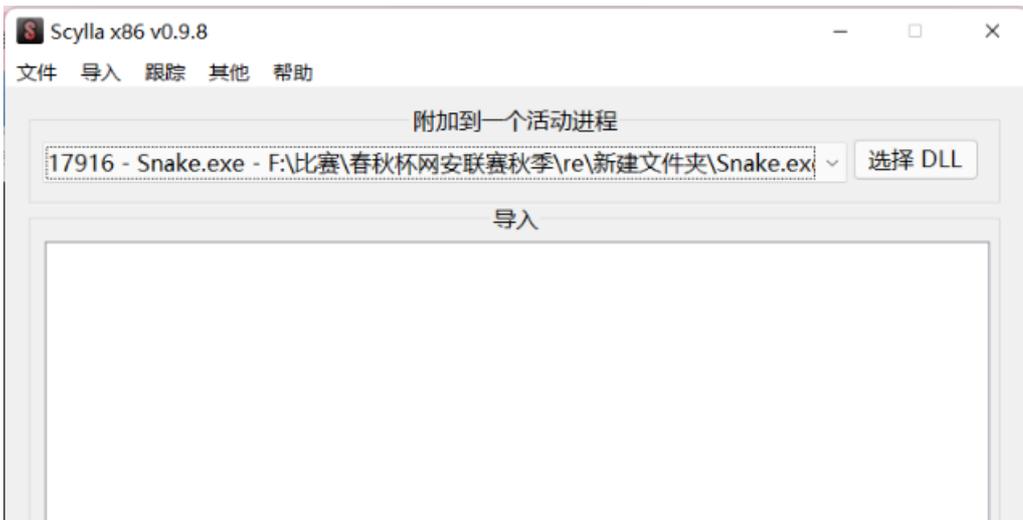


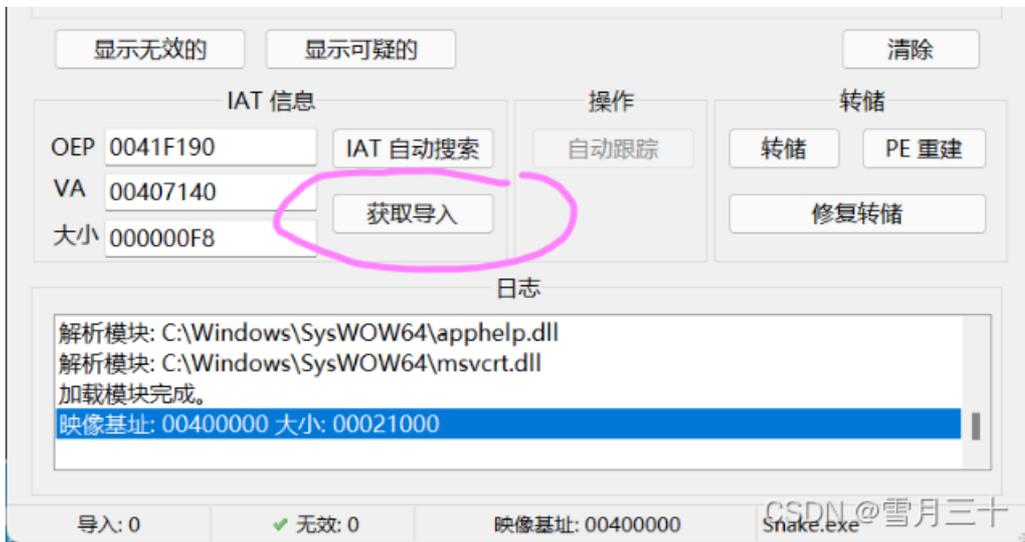
确定



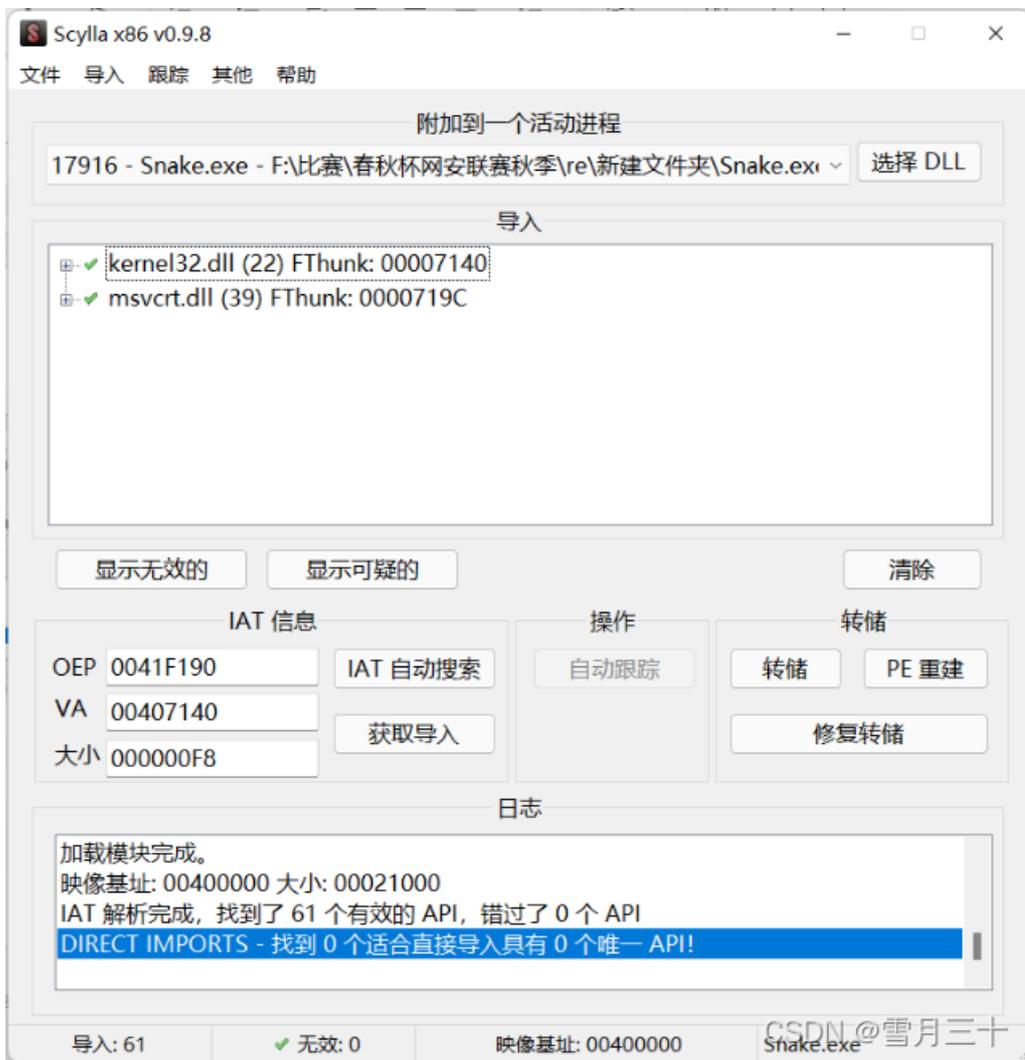
可以看到很幸运找到了IAT的地址

点击获取导入



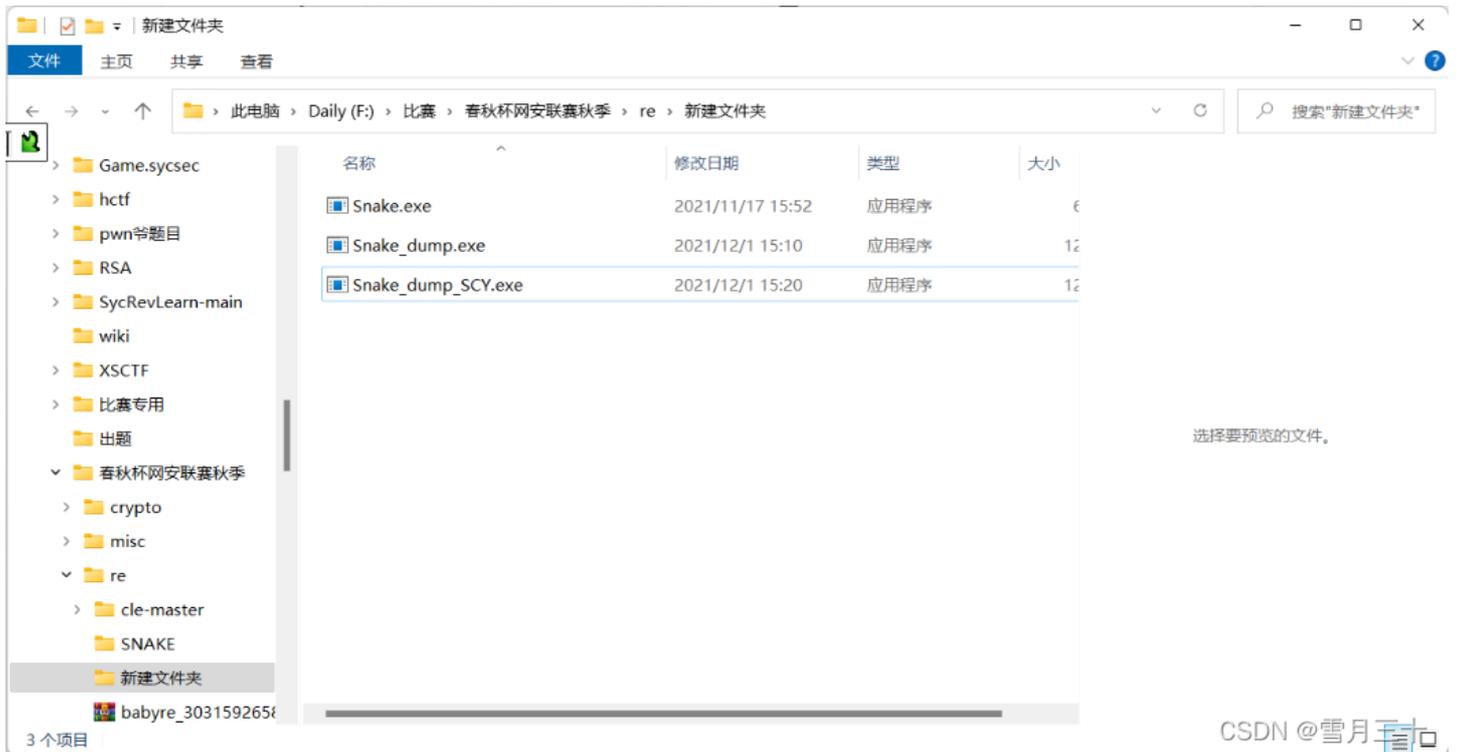


可以看到函数没有异常

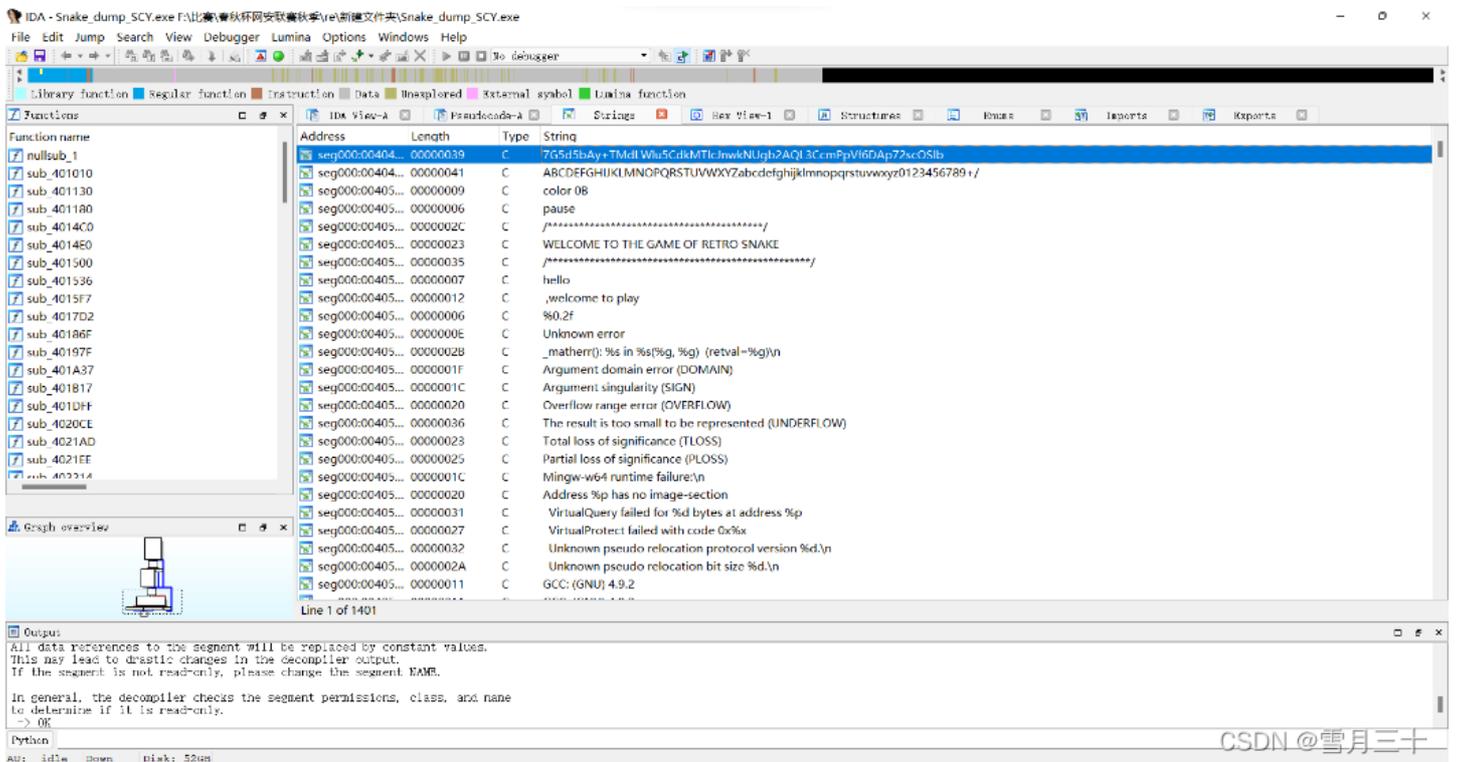


选择我们之前刚存储的文件点击打开

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

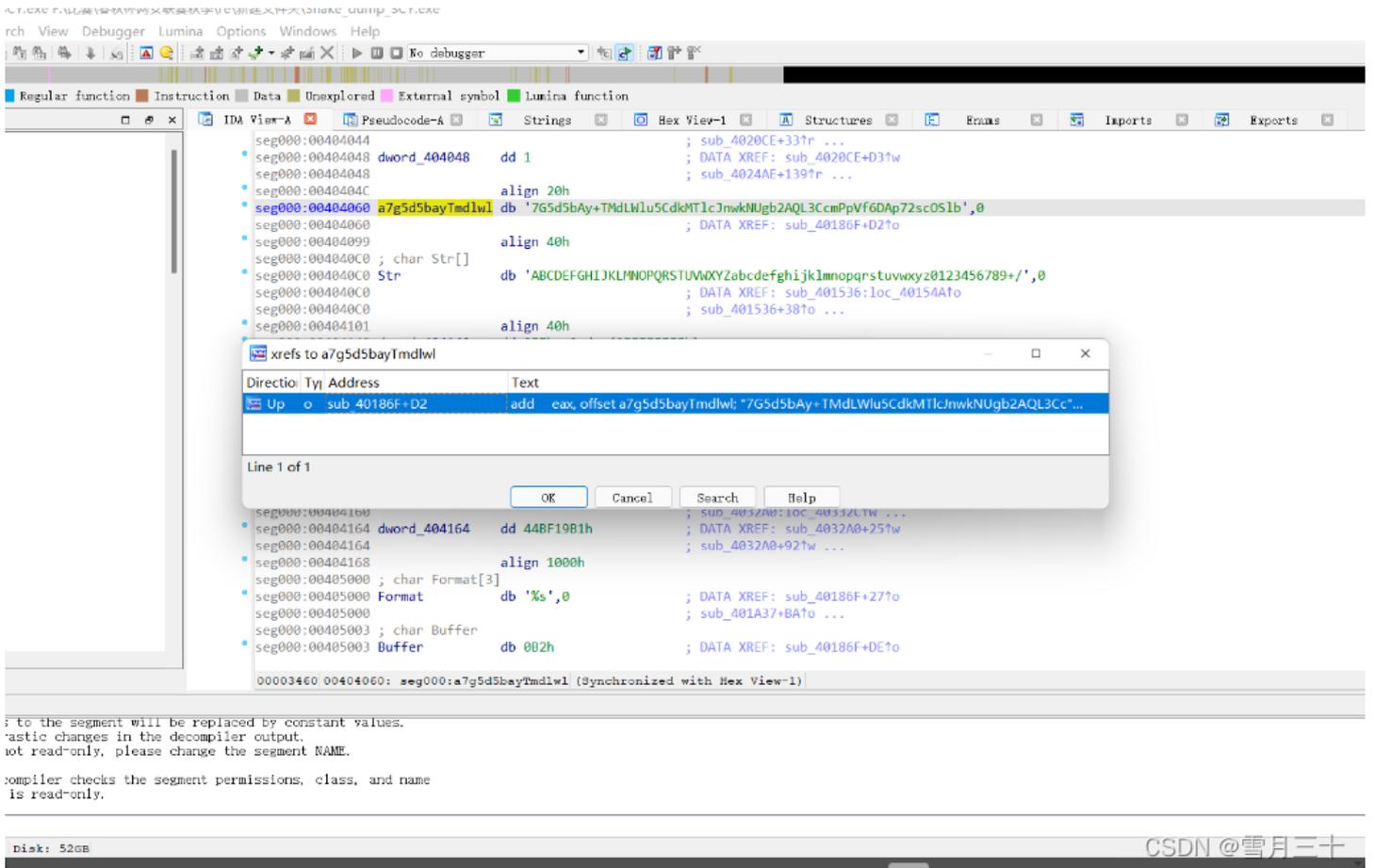


## 静态分析



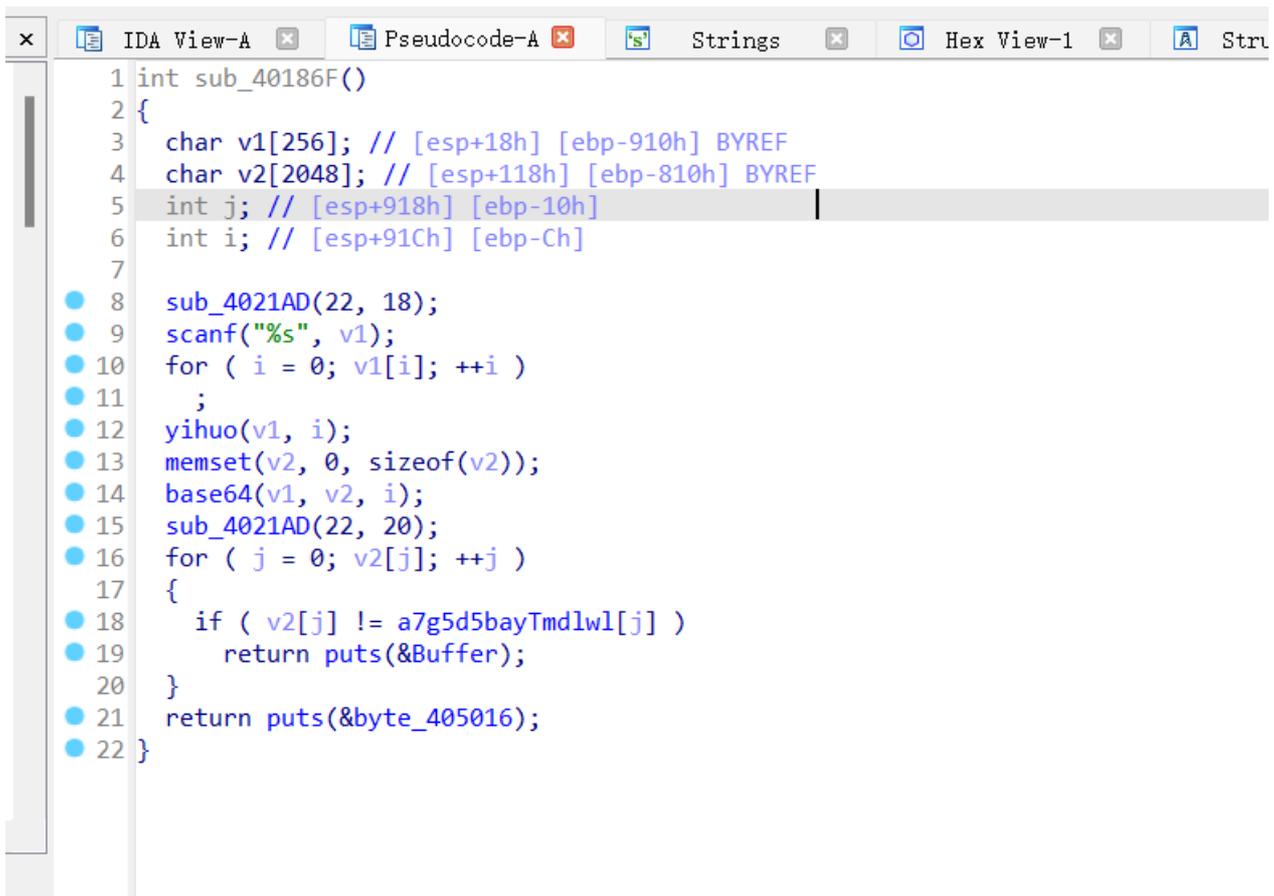
搜索字符串 可以看到第一行有可疑的字符串出现  
第二行是明显的base表可以猜测是有base解密的

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



其实主要就是就是一个异或的加密其他的都是很简单的



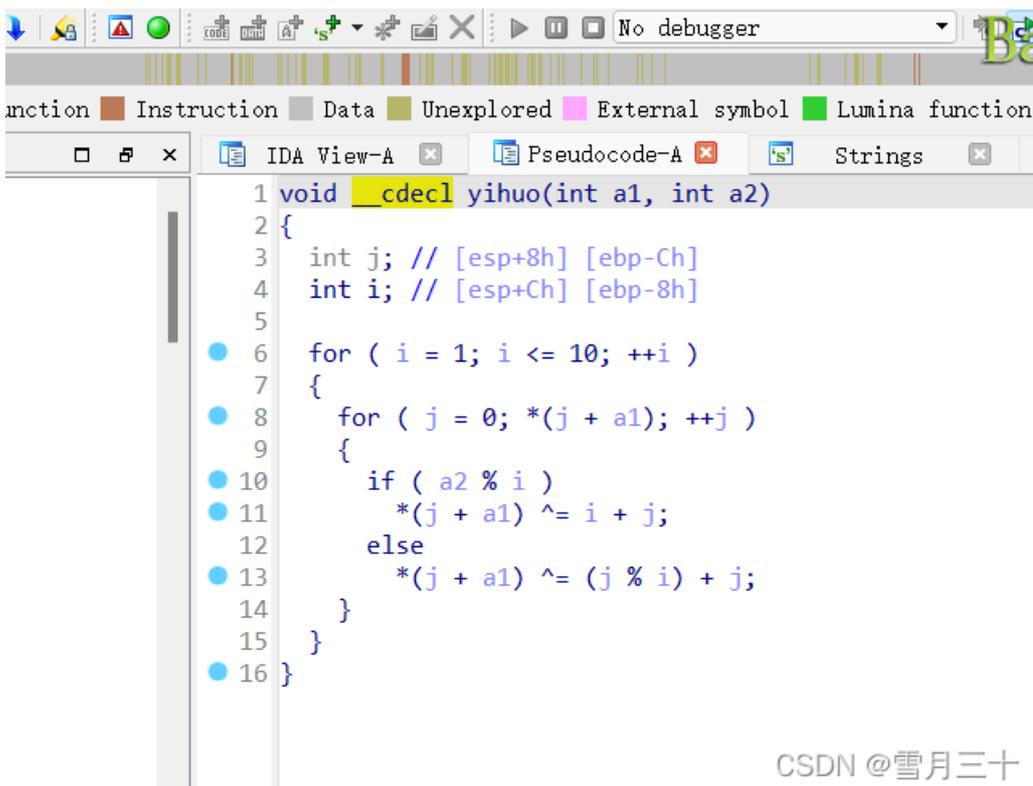
```

Instruction  Data  Unexplored  External symbol  Lumina function
IDA View-A  Pseudocode-A  Strings
1 int __cdecl yihuo(int a1, int a2)
2 {
3     int result; // eax
4     int j; // [esp+8h] [ebp-Ch]
5     int i; // [esp+Ch] [ebp-8h]
6
7     for ( i = 1; i <= 10; ++i )
8     {
9         for ( j = 0; ; ++j )
10        {
11            result = *(j + a1);
12            if ( !result )
13                break;
14            if ( a2 % i )
15                *(j + a1) ^= i + j;
16            else
17                *(j + a1) ^= (j % i) + j;
18        }
19    }
20    return result;
21 }

```

CSDN @雪月三十

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



```

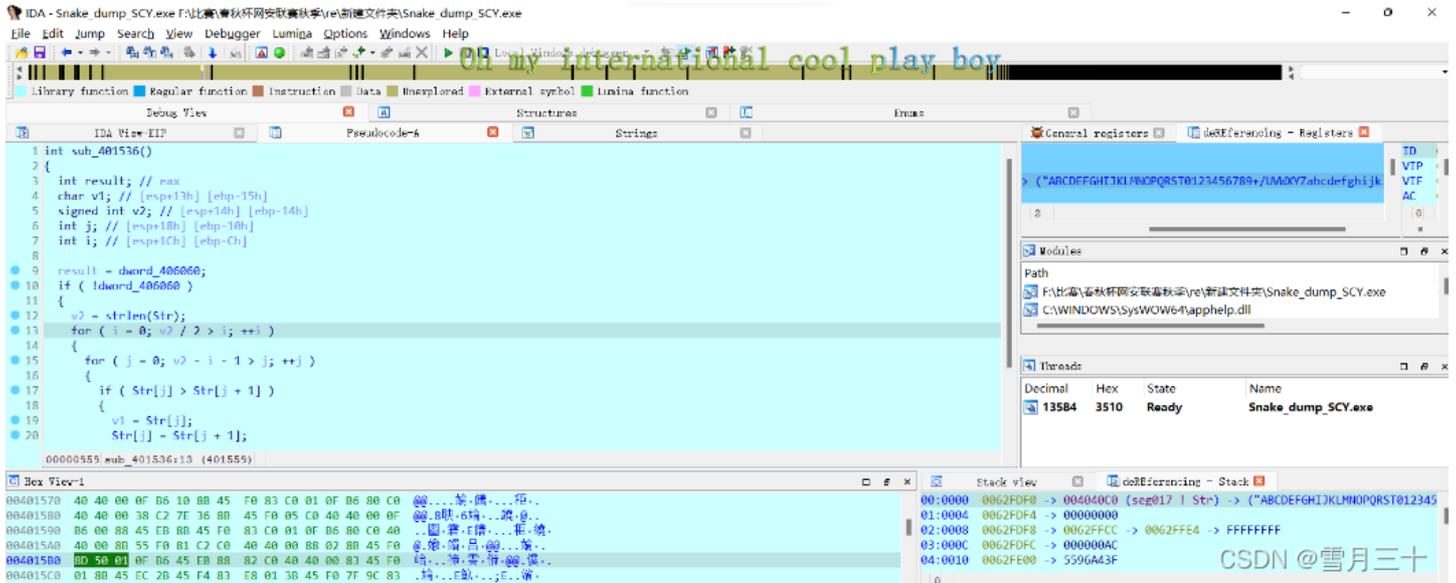
function  Instruction  Data  Unexplored  External symbol  Lumina function
IDA View-A  Pseudocode-A  Strings
1 void __cdecl yihuo(int a1, int a2)
2 {
3     int j; // [esp+8h] [ebp-Ch]
4     int i; // [esp+Ch] [ebp-8h]
5
6     for ( i = 1; i <= 10; ++i )
7     {
8         for ( j = 0; *(j + a1); ++j )
9         {
10            if ( a2 % i )
11                *(j + a1) ^= i + j;
12            else
13                *(j + a1) ^= (j % i) + j;
14        }
15    }
16 }

```

CSDN @雪月三十

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+TmdLWlu5CdkMT1cJnwkNUgb2AQL3CcmPpVf6DAp72sc0S1b"
tableBase64 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
tableNew = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+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+eTmPlx6ZCpwMTxoJz8wNgsnWAQLXCoyP1hraDA1bw4o0Sxn
#[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,
```