目录

[Reverse 2](#_Toc10601)

[一. IDA 2](#_Toc30042)

[二. C&C++语言语法常见函数 10](#_Toc28176)

[三. 脱壳 12](#_Toc2367)

[1. 万能脱壳助手 12](#_Toc5867)

[2. Kali脱壳 12](#_Toc17067)

[3. Themida/Winlicense(3.XX)壳 13](#_Toc10708)

[4. UPX脱壳 13](#_Toc15117)

[四. APK 13](#_Toc24946)

[五. 常见函数 14](#_Toc5512)

[六. Dnspy 22](#_Toc9002)

[七. Perl 23](#_Toc23945)

[八. X64dbg 23](#_Toc8651)

[九. 花指令 23](#_Toc25431)

[1.JZ/JNZ花指令 23](#_Toc31339)

[十. 控制流平坦化 24](#_Toc19953)

[十一. PyInstaller Extractor 24](#_Toc17739)

[十二. WinRAR 25](#_Toc5352)

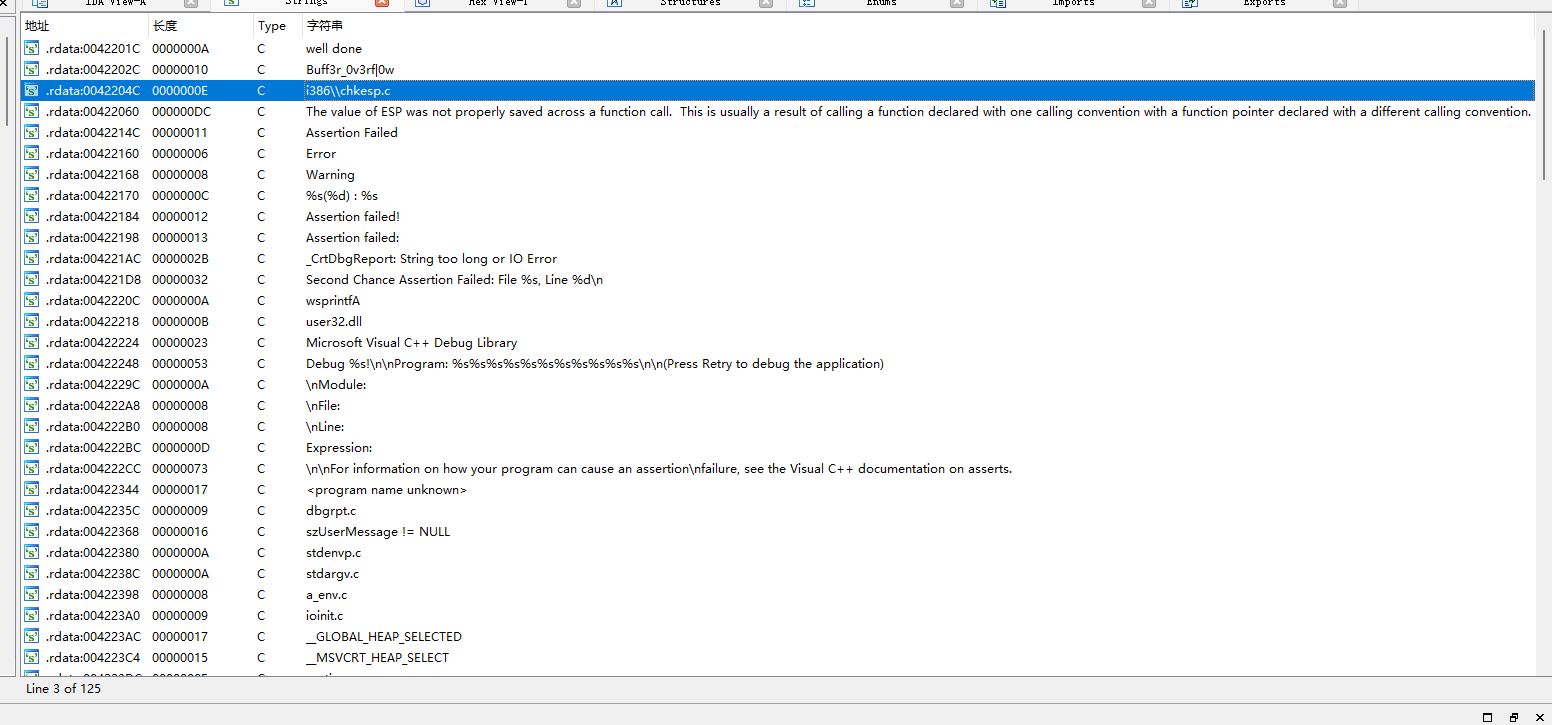
[十三. ARM 26](#_Toc8895)

[十四. 杂七杂八的打法 26](#_Toc27173)

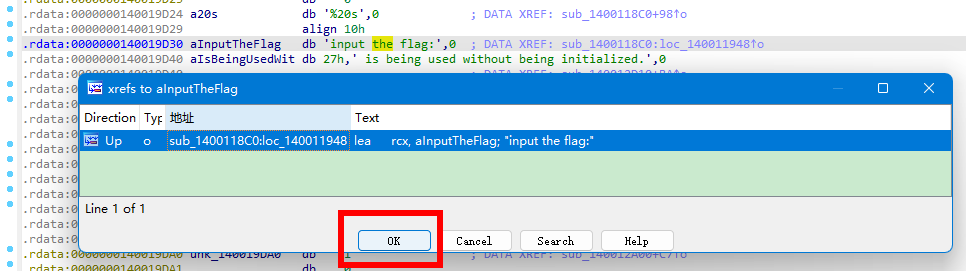
# Reverse

## IDA

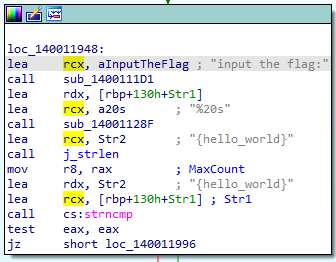
SHIFT+F12查看所有IDA程序中字符串，双击字符串进入内部查看



Ctrl+X查看当前引用位置

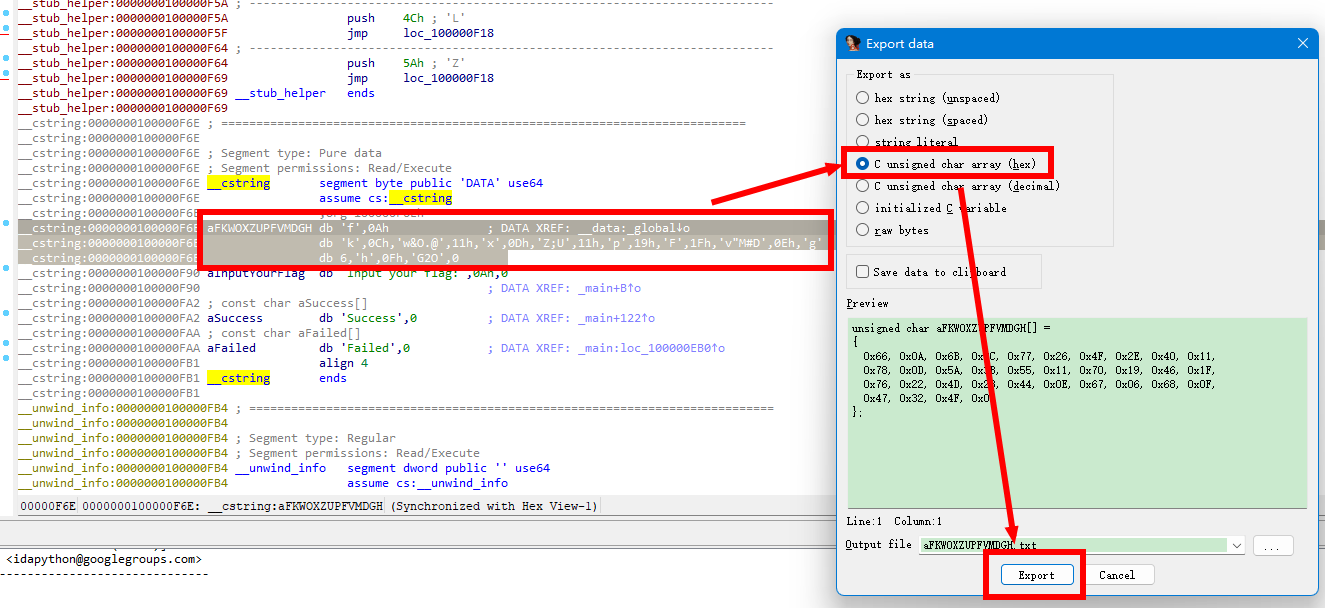


点击OK查看

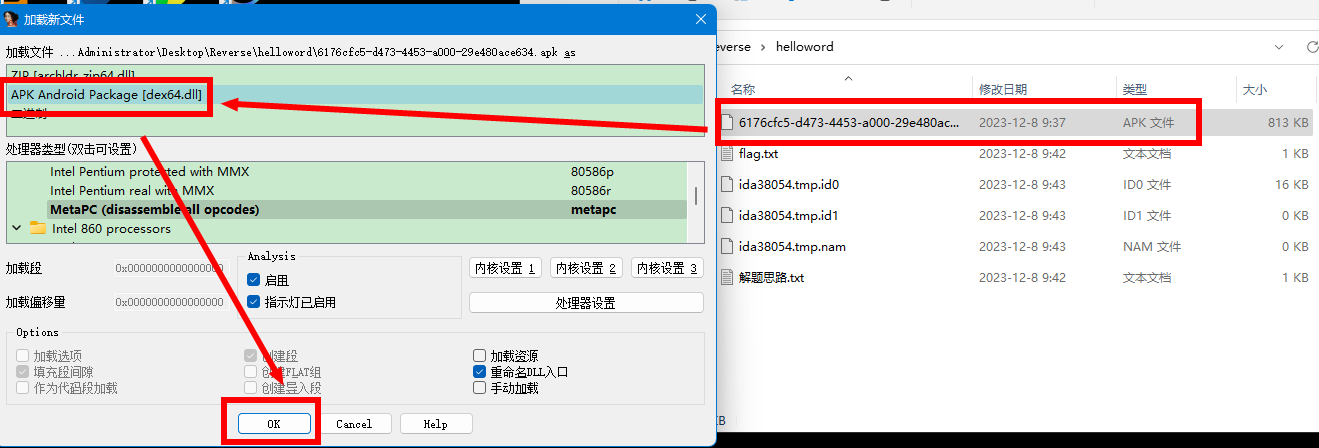


Esc回退

选中数据Shift+E可以将数据导出为任意格式

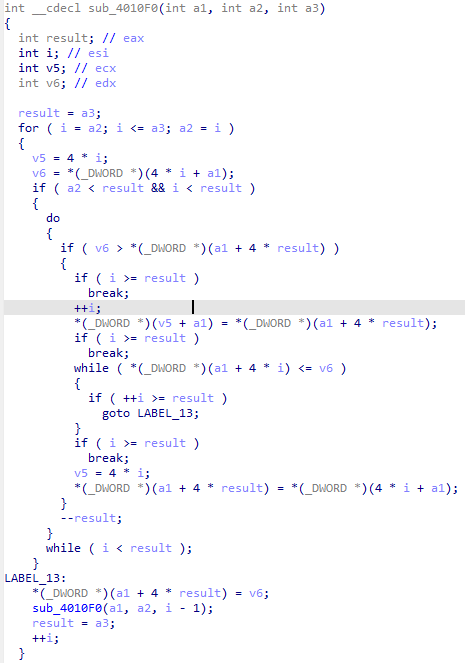


IDA查看APK文件



IDA字符串格式转换：右键字符串选择格式即可

伪代码转换为C语言代码，按F5同时双击



在数字变量按R键即可显示数值



使用Angr脚本模板解答exe文件得到结果，注意修改文件名、避免走到的分支地址，需要找到的结果地址



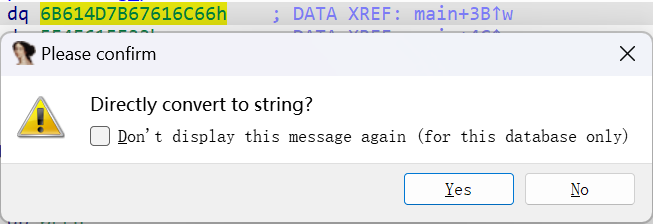
| **ida快捷键** | **功能** |
| --- | --- |
| F7 | 单步步进 |
| F8 | 单步步过 |
| F9 | 继续运行程序 |
| F4 | 运行到光标所在行 |
| Ctrl + F7 | 直到该函数返回时才停止 |
| Ctrl + F2 | 终止一个正在运行的进程 |
| F2 | 设置断点 |

调试快捷键

打断点之后，左键点击两次变量查看其值，选中数据范围按Shift+K可以得到数组格式数据并复制

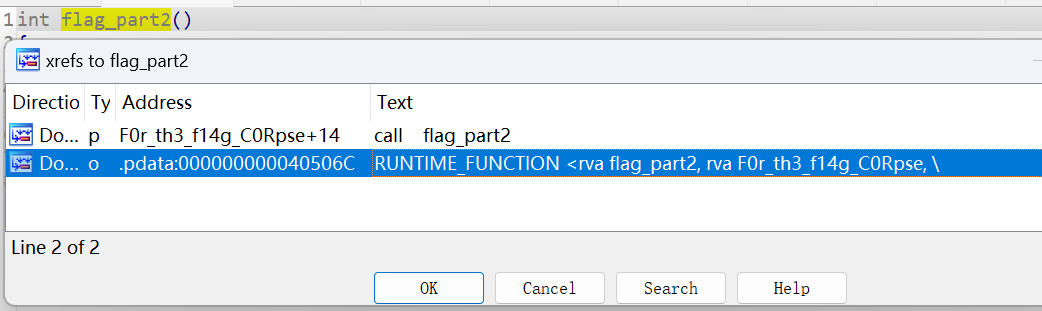


在汇编处按A键->Yes可以将16进制乱码改为正确字符串

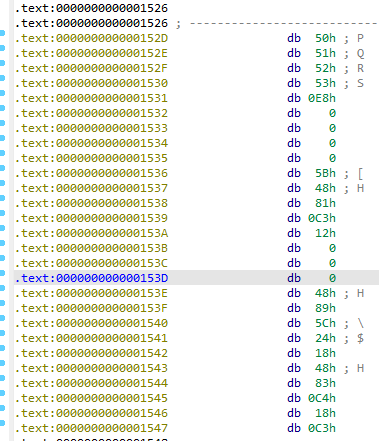




在方法上按X可以转移到引用当前方法的地方



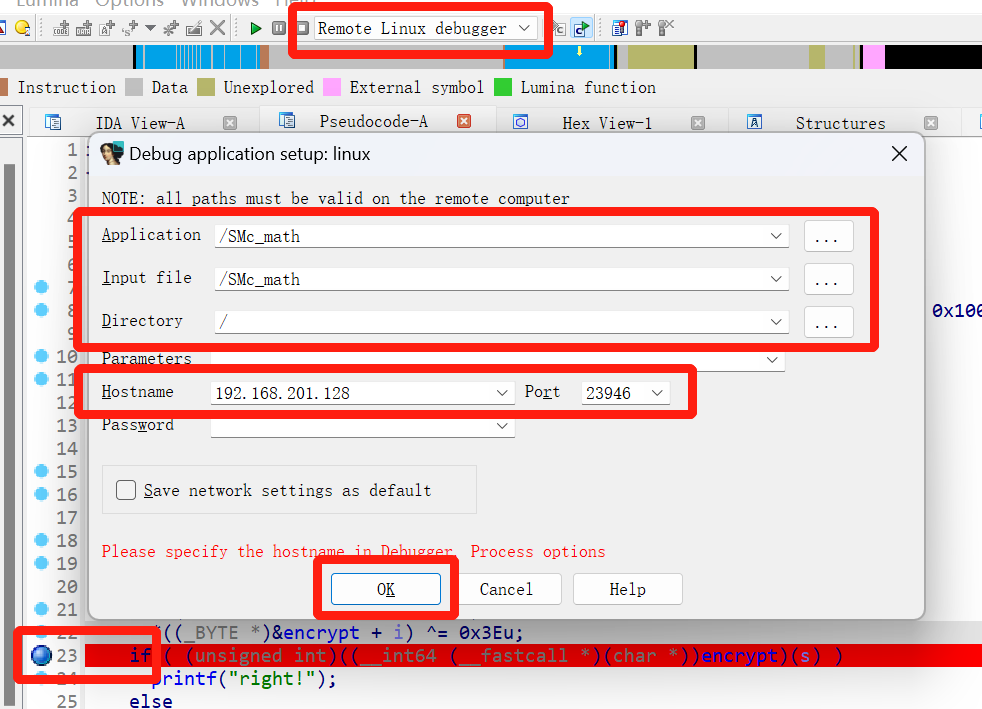
脚本批量去除花指令：依次选中sp-analysis failed花指令，一直按U展开直到红色部分消失，取公共的部分就是花指令全部内容



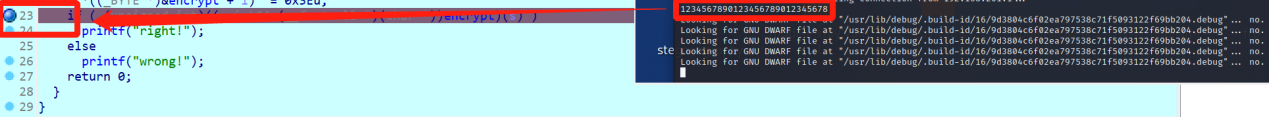
调试linux文件：将文件在IDA打开，并拷贝到kali桌面中，kali桌面打开cmd输入：./linux\_server64



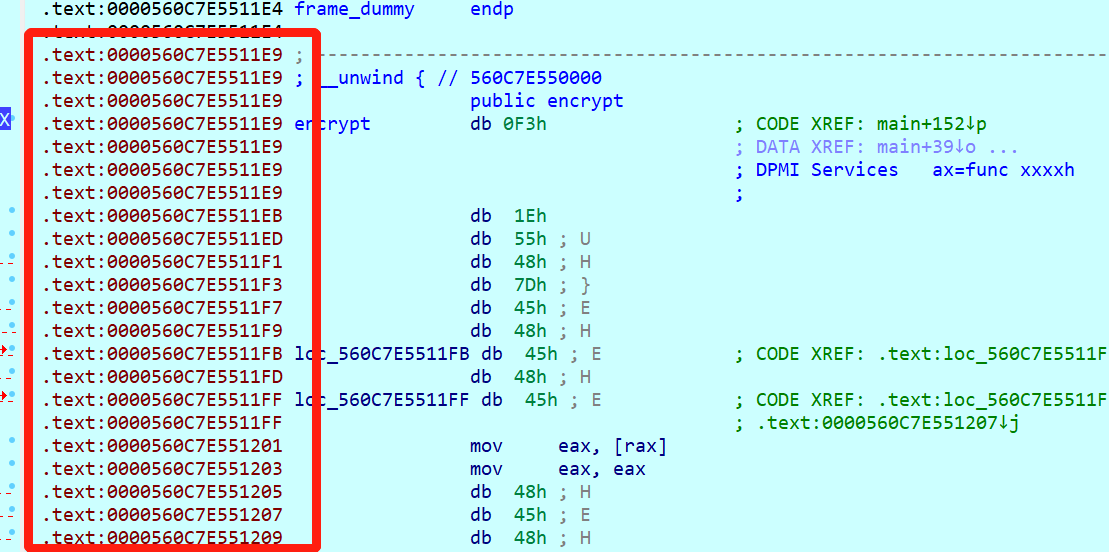
IDA打好断点->选择Remote Linux debugger->点击运行->输入Application为/文件名->Input file为/文件名->Directory为/->Hostname为kali的IP地址->点击OK



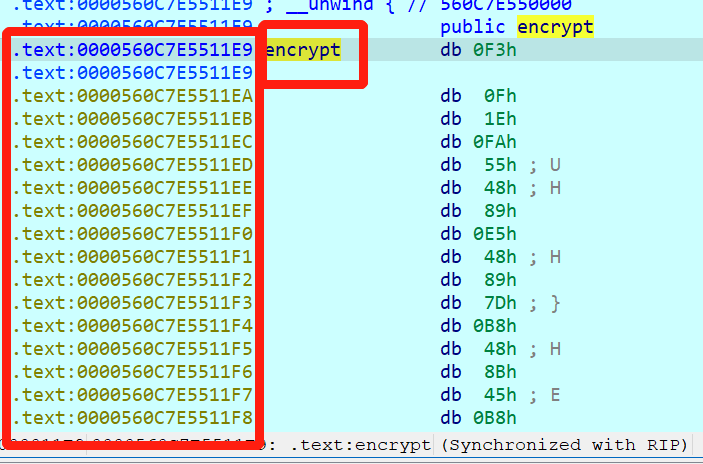
./linux\_server64接收到消息，在下面输入符合进入断点条件的值触发



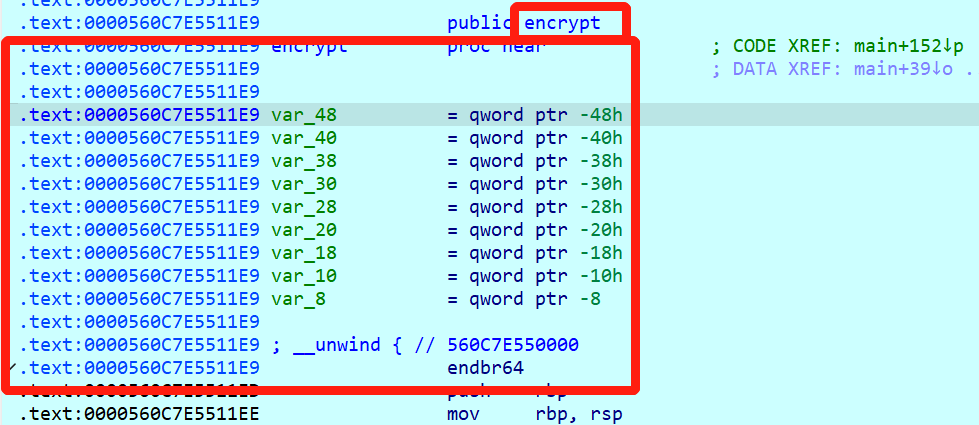
之后如果存在红色汇编区域



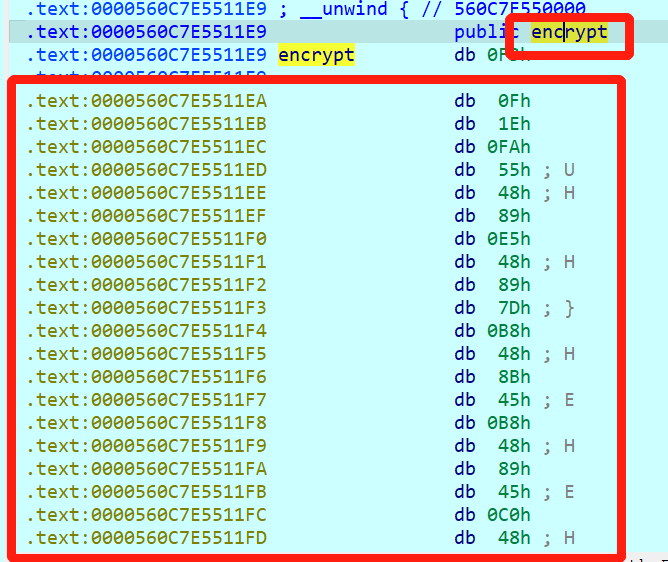
函数恢复先将红色头部函数名开始按U将红色部分全部展开



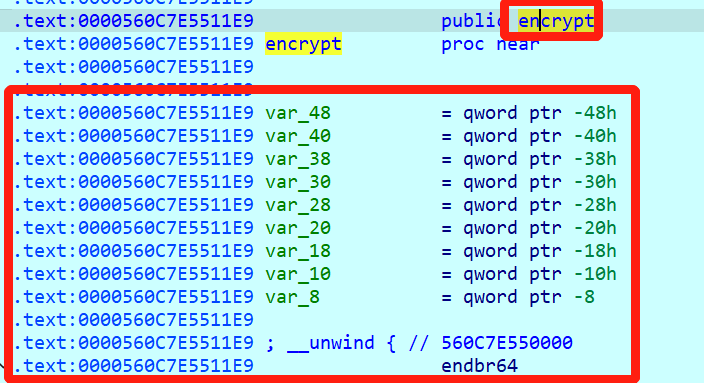
之后在头部函数按P识别



之后在头部函数再按U展开



最后再头部函数按P识别为函数(注意其实时重复上面的U+P过程，只不过第一次F5看会是空函数，第二次才正常)



在头部函数按F5查看得到逻辑



右键非负16进制数据可以点击任意格式转化(10进制&2进制)，同时10进制的值右键可以统一转换成负数

## C&C++语言语法常见函数

atoi(x)：x为string，将字符串转为整数，注意”123xyz”会变成123，”-12.34”会变成-12，”123w45”会变成123，” 123w456”会变成123，”123 456”会变成123

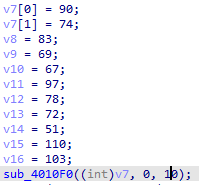
strncmp(a, b, 0x21uLL)代表：将字符串a和字符串b的前0x21个字符进行比较

strcat(a, b)代表：得到字符串为a+b

int占四个字节，所以需要\*4。如果是char类型或者系统作为下标，就不需要，例如a1 + result \* 4的值就是a1[result]，而v5+a1的值就是a1[v5]



IDA伪代码中，例如v开头接数字的，表示为一个字符串，如果函数调用(int)v7，则代表从7开始后的字符串的int数组，例如下面的v实际上是：ZJSECaNH3ng



函数地址重置并覆盖：例如：off\_201028 = sub\_6EA;即将off\_201028地址的函数赋值为函数sub\_6EA

arr中按照数值数目得到键(数值)值(数值个数)对：sumerizeArr = Counter(arr)

UnDecorateSymbolName(a, b, c, d)函数中根据b值(假设是：private: char \* \_\_thiscall R0Pxx::My\_Aut0\_PWN(unsigned char \*))确定a的值确定方式：

无论 \_\_cdecl，\_\_fastcall还是\_\_stdcall调用方式，函数修饰都是以一个“?”開始，后面紧跟函数的名字。 再后面是參数表的開始标识和 依照參数类型代号拼出的參数表。即：a = ?My\_Aut0\_PWN

对于C++的类成员函数（其调用方式是thiscall）。 函数的名字修饰与非成员的C++函数稍有不同，首先就是在函数名字和參数表之间插入以“@”字 符引导的类名。即：a = ?My\_Aut0\_PWN@R0Pxx

其次是參数表的開始标识不同，公有（public）成员函数的标识是“@@QAE”,保护（protected）成员函数的标识是 “@@IAE”,私有（private）成员函数的标识是“@@AAE”，假设函数声明使用了constkeyword，则对应的标识应分别为 “@@QBE”，“@@IBE”和“@@ABE”。即：a = ?My\_Aut0\_PWN@R0Pxx@@AAE

參数表的拼写代号例如以下所看到的：

X--void

D--char

E--unsigned char

F--short

H--int

I--unsigned int

J--long

K--unsigned long（DWORD）

M--float

N--double

\_N--bool

U--struct

....

指针的方式有些特别。用PA表示指针，用PB表示const类型的指针。即如果函数返回值类型为 char \* ：a = ?My\_Aut0\_PWN@R0Pxx@@AAE

如果形参类型为unsigned char \* 对应PAE。即v2 = ?My\_Aut0\_PWN@R0Pxx@@AAEPADPAE

參数表后以“@Z”标识整个名字的结束。假设该函数无參数，则 以“Z”标识结束。最终v2 = ?My\_Aut0\_PWN@R0Pxx@@AAEPADPAE@Z

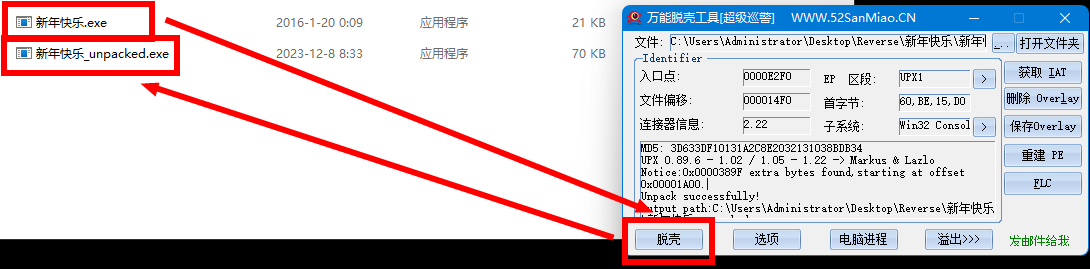
在正向遍历爆破时，如果涉及到移位，可以在向左移位时余上遍历最大值保证数据不会溢出

Strcpy如果存在将多位字符串赋值给少位字符串的情况，为栈溢出，可以通过追加指定函数位置的逆序，实现调用指定函数，例如sub\_40233D，就要在字符串后面加上chr（0x3D）chr（0x23）chr（0x40）

## 脱壳

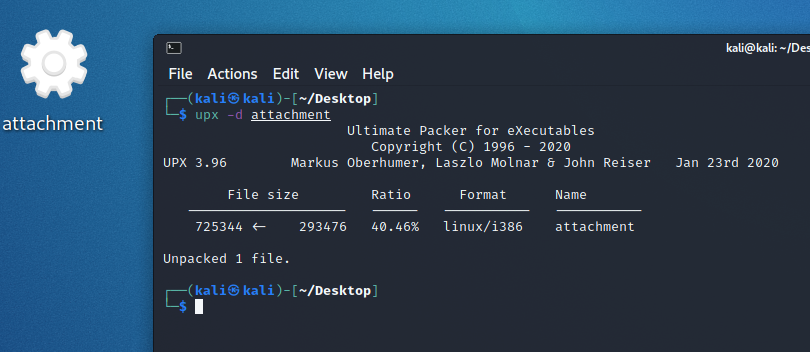
### 万能脱壳助手

将文件拖入万能脱壳助手后点击脱壳得到脱壳后的文件



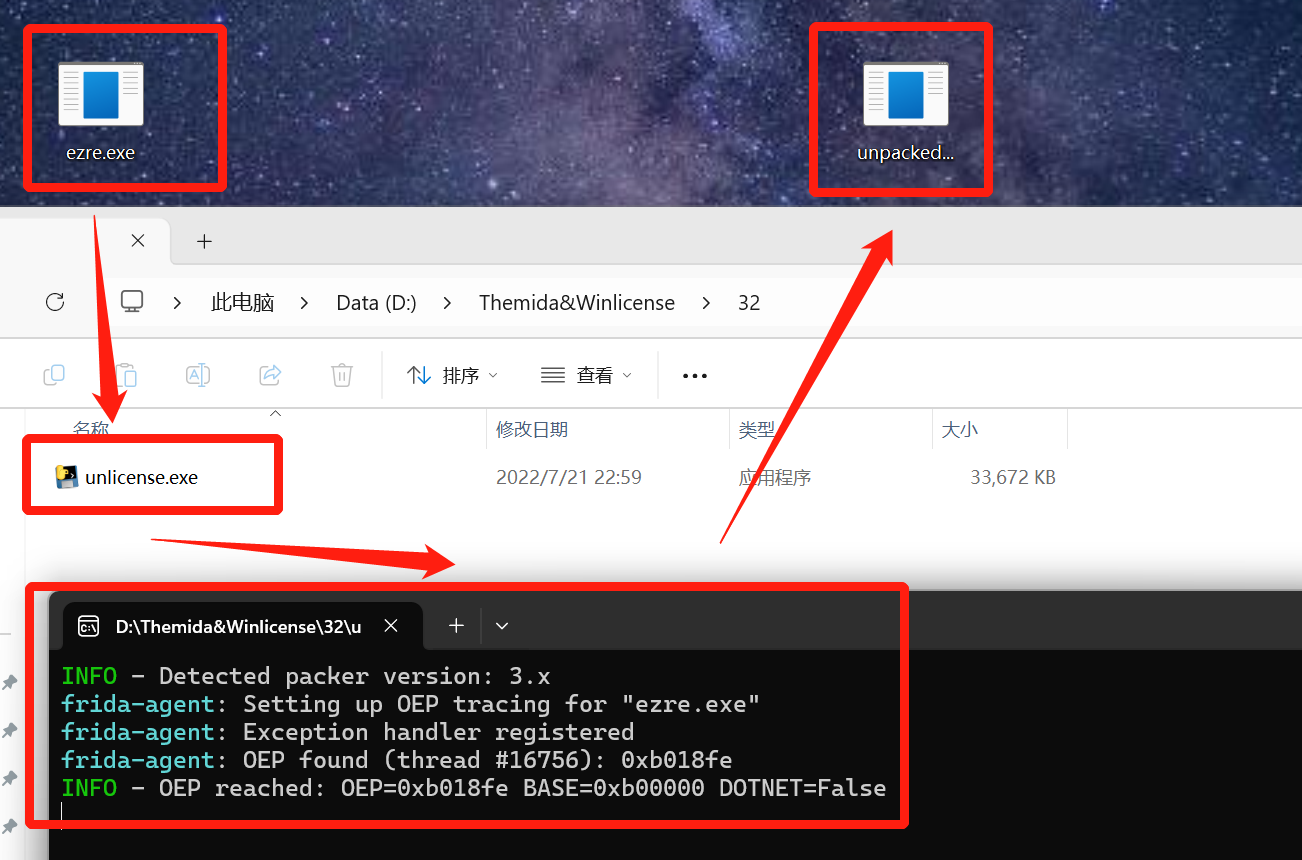
### **Kali脱壳**

在文件所在文件夹打开cmd输入：upx -d 文件名



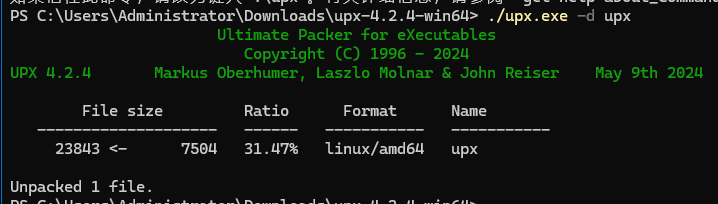
### **Themida/Winlicense(3.XX)壳**

将文件拖到Themida&Winlicense文件夹中的对应位数的文件中，自动脱壳得到unpacked开头的脱壳文件



### **UPX脱壳**

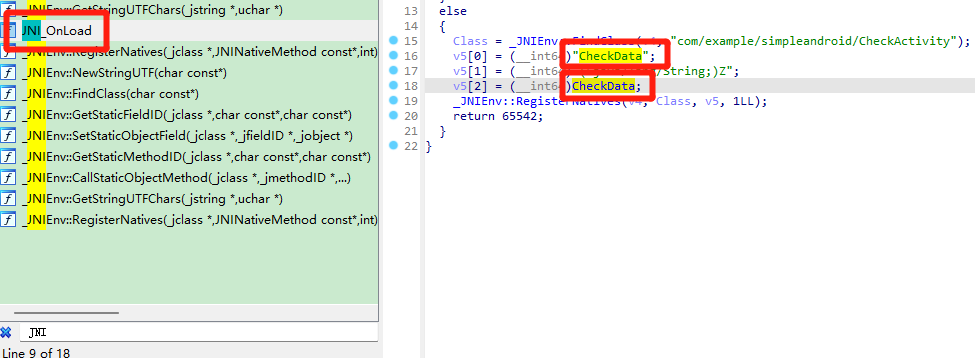
在upx.exe文件夹中打开cmd，输入./upx.exe -d 加壳文件名，进行脱壳



## APK

jadx-gui打开apk文件，之后保存到文件夹下，查看MainActivity主函数信息，内部如果存在libcore.so文件就用IDA打开查看

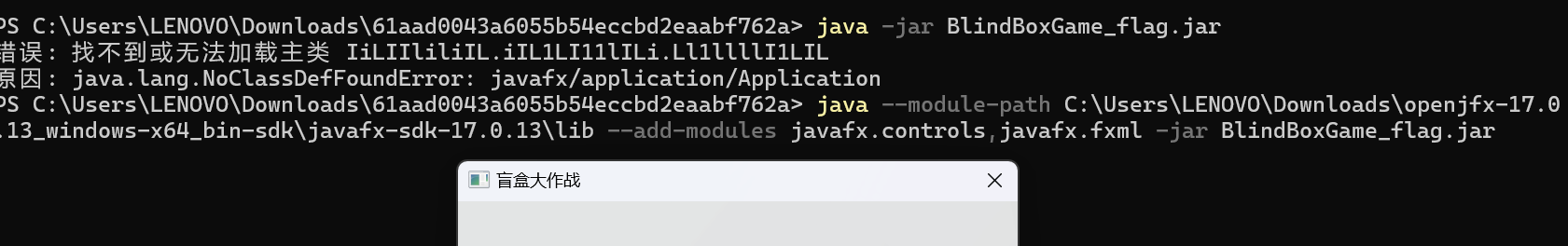
Jadx编译的文件中的native 层方法如果在IDA中搜索不到，考虑不是静态注册，而是动态注册。搜索 JNI\_OnLoad，找到native 层方法函数，点进去即可查看



反混淆：点击jadx的反混淆按钮

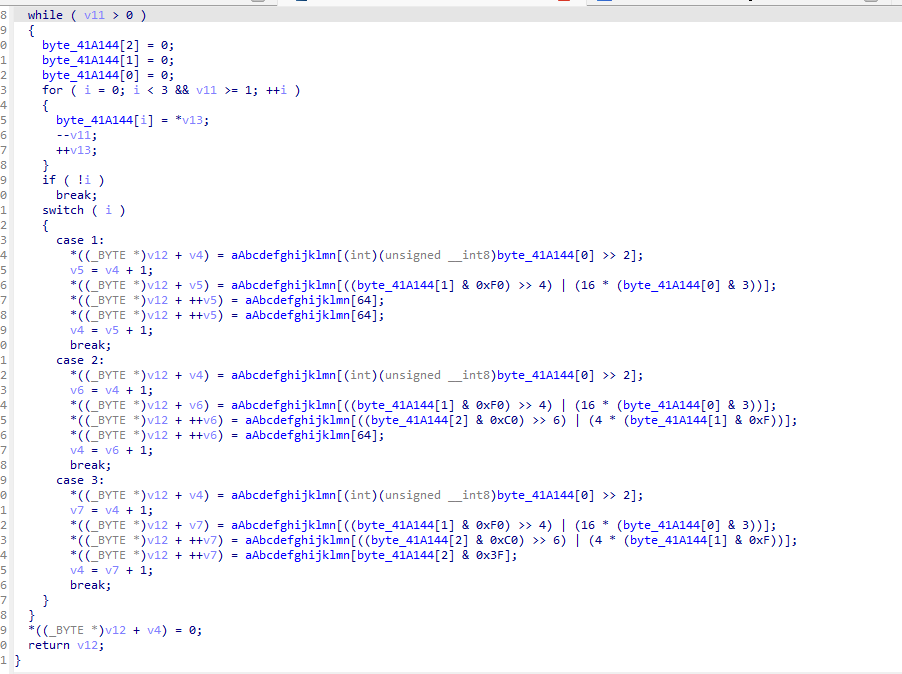


jadx同时也可以打开jar文件反编译，运行jar文件如果缺少类包，可以下载缺少的类包后在jar文件夹cmd输入：java --module-path 类包路径 --add-modules 类包名 -jar jar文件名

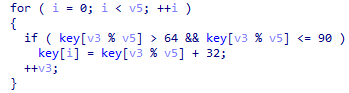


## 常见函数

BASE64方法代码

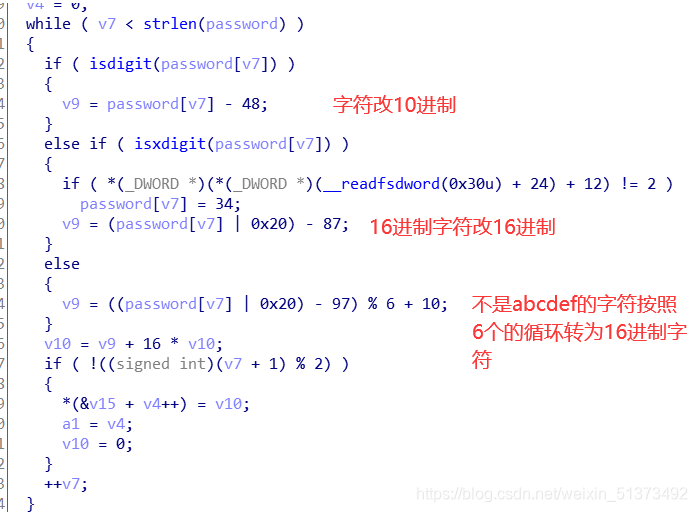


大小写转换方法代码

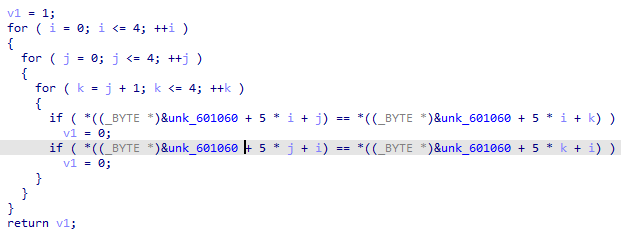


存在两个字符串并且输出存在\*left和\*right可以考虑二叉树两种遍历方式去复原另一种遍历方式

进制转换方法代码



数独算法，外部是矩阵，内部同行同列不允许有相同的值。



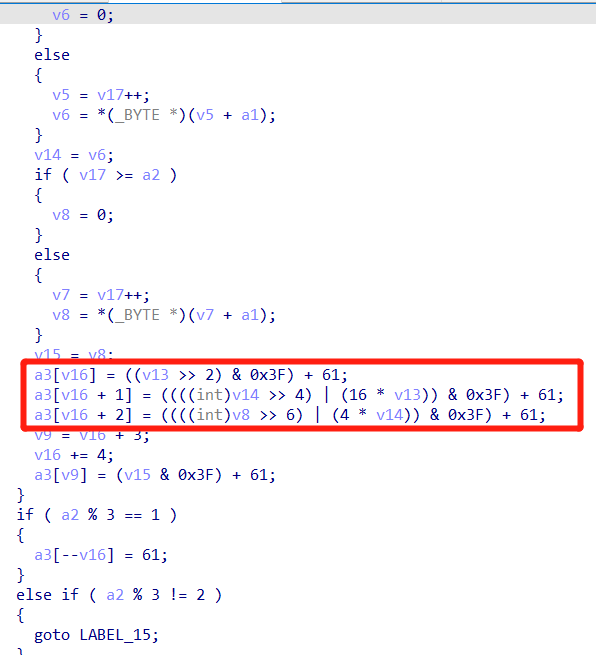
二叉树中序遍历算法



各种base64解码打法



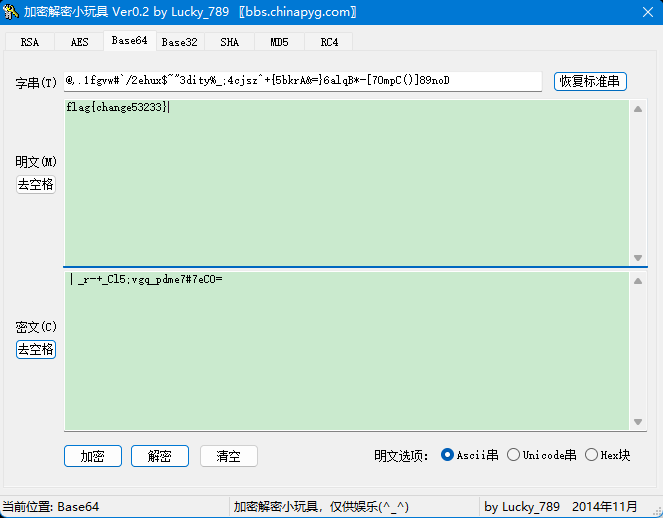
三个字符一组的base64



解码脚本

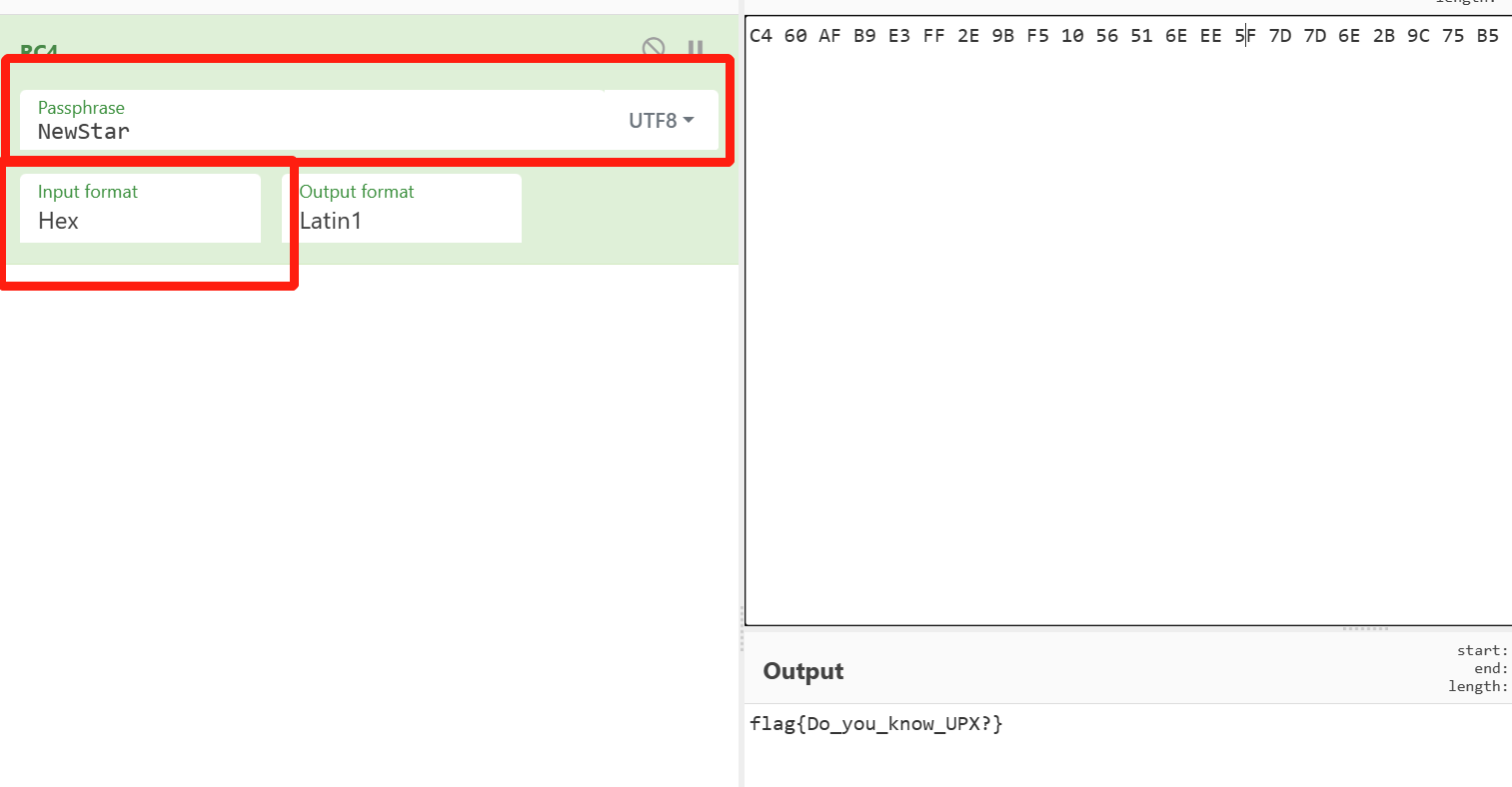


使用加密解密小玩具实现base64码表字串替换解密，注意存在/的时候可能需要删除(作为转义字符)

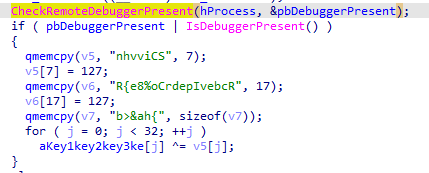


RC4解密：

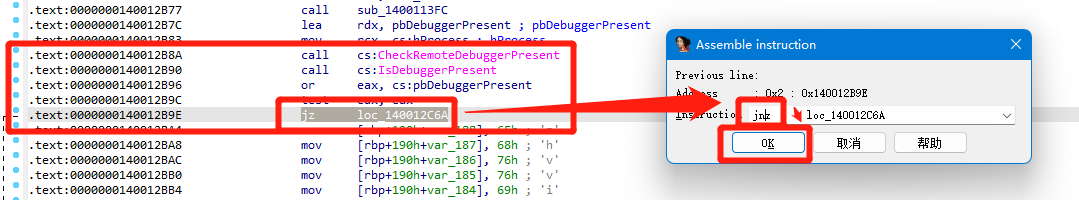
cyberchef使用RC4，输入key值作为Passphrase，设置对应的Input format解密



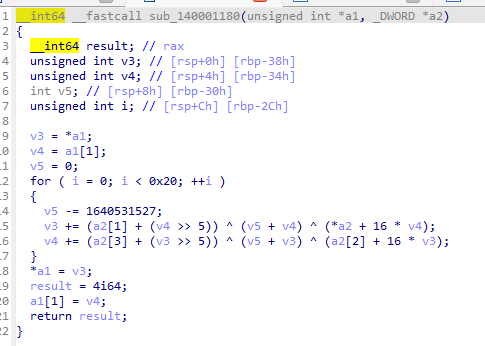
反调试算法：函数名为CheckRemoteDebuggerPresent即反调试函数



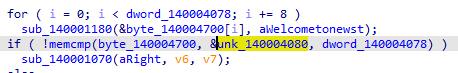
找到对应的汇编位置将下面的jz改为jnz



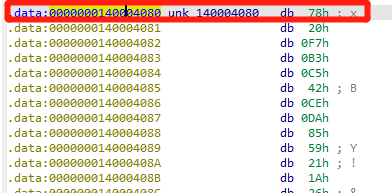
TEA算法：



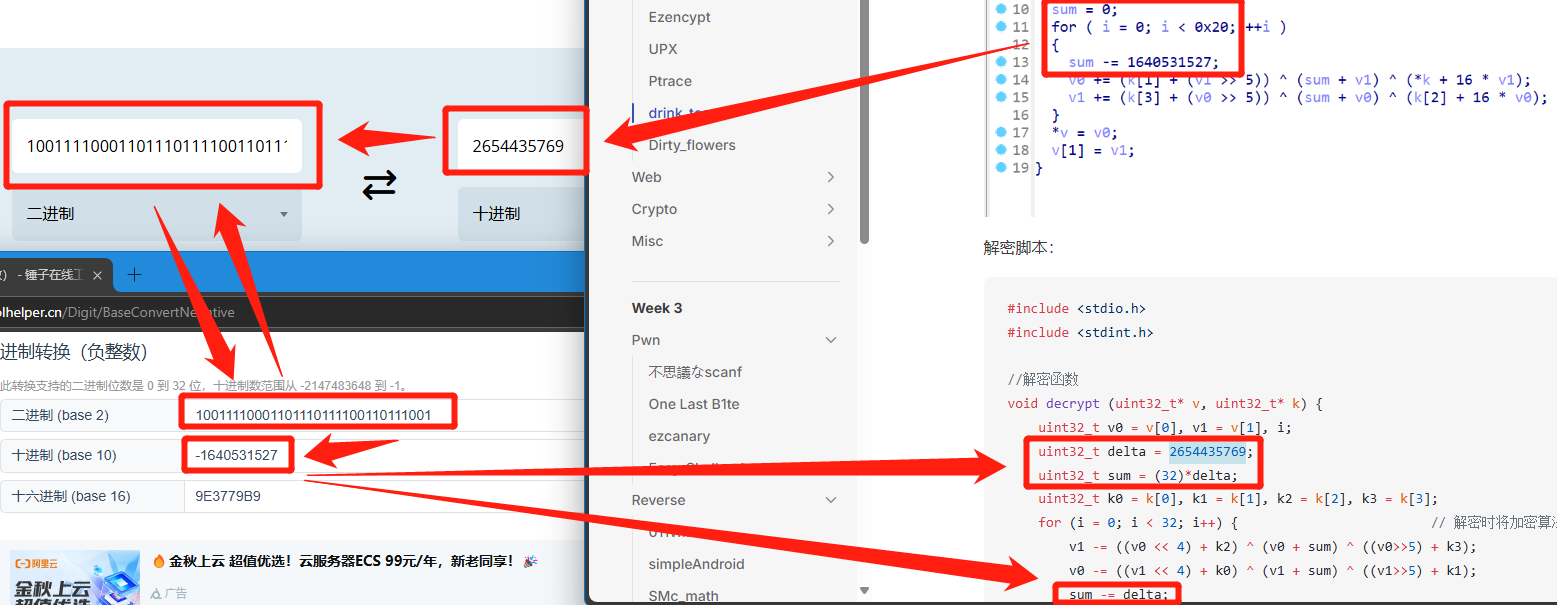
其中a2为key值，tea算法密文长度为64位(相当于16个16进制数，例如上面a2为4个16进制数进行一次加密，那么一共进行4次tea加密)



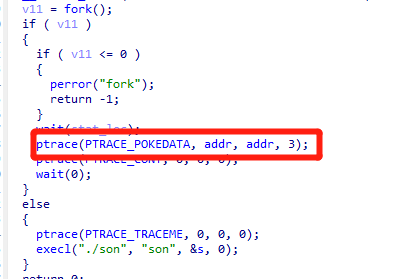
找到判断字符串相等的地方进去得到地址对应的数据跑脚本



如果循环加密是出现负数，解密是需要用他的补码(负数二进制对应的正数)进行解密



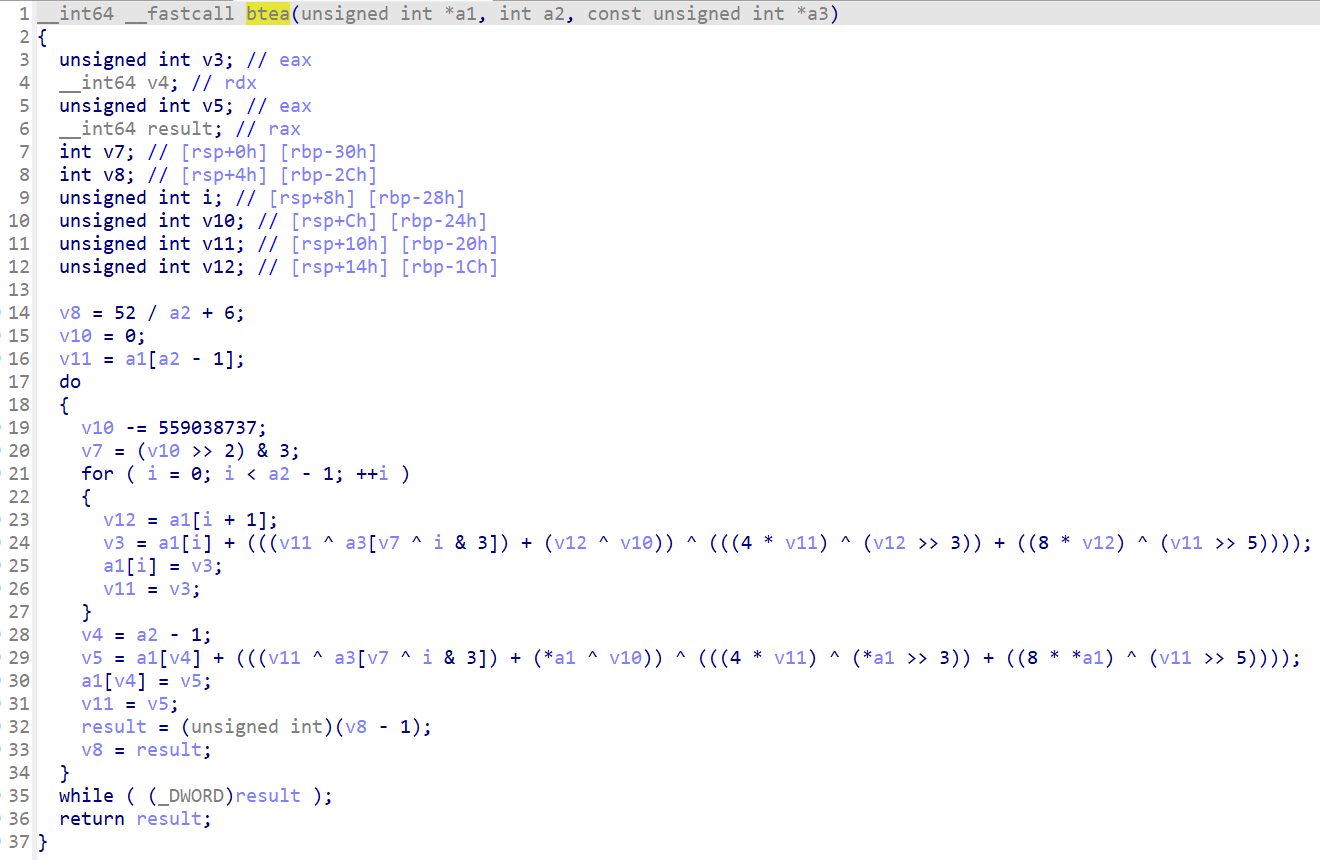
多进程：在父进程中ptrace函数可以修改子进程中的地址对应的值(比如下面就是将子进程中addr地址对应的值改为3，原子进程addr地址对应的值作废)



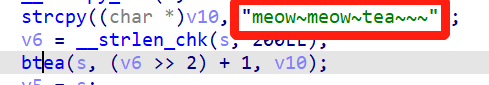
构造方法先运行，之后调native的方法，如果没有方法内容就去so文件中用IDA打开找



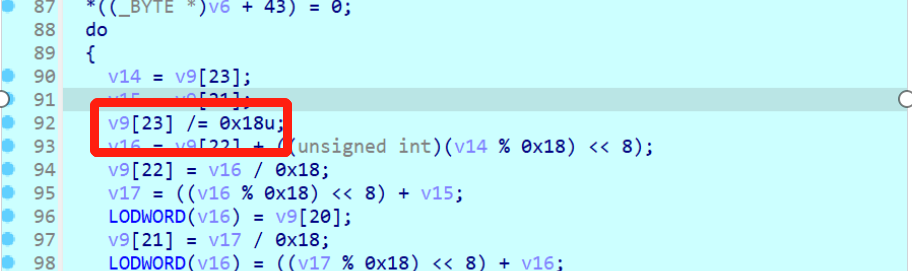
xxtea加密



Key值为固定字符串

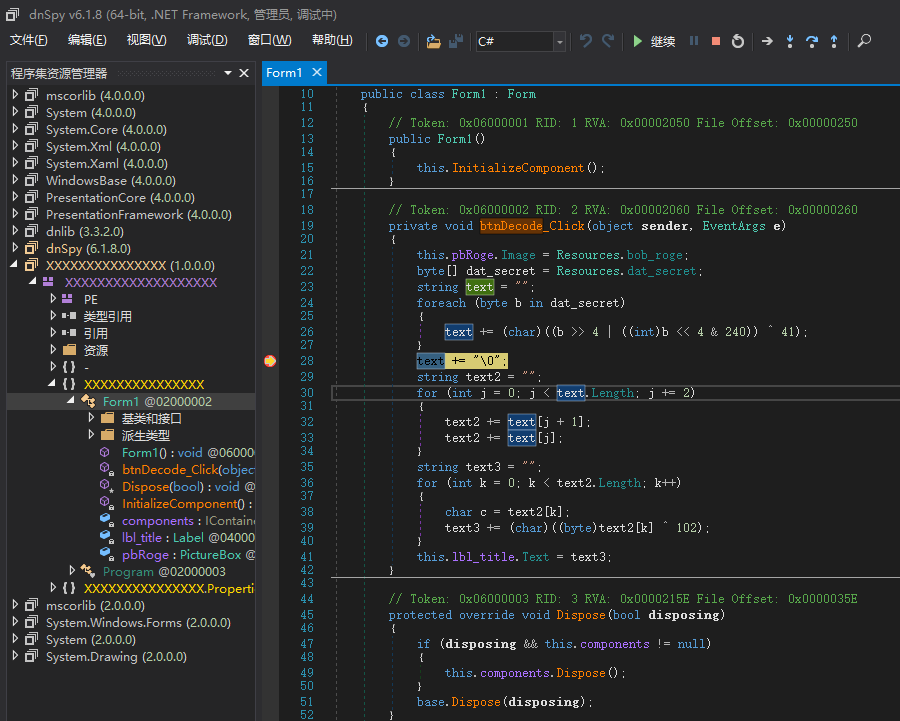


base24加密：存在类似v9[23] /= 0x18u(除以24)，解密脚本在刷题文件夹，码表长度不够就重复多次



## Dnspy

反编译.net项目，在程序里可以打断点后点击运行按钮并触发断点操作

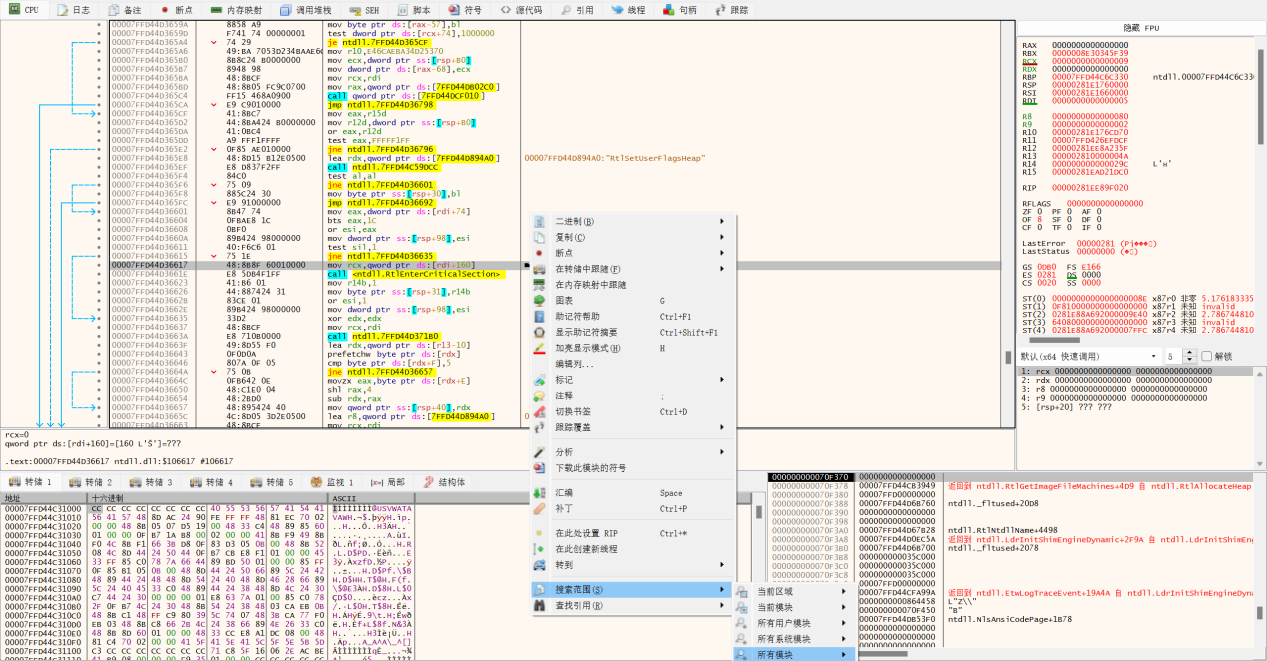


## Perl

搜索script相关信息

## X64dbg

拖入程序->右键->搜素范围->所有模块->字符串引用



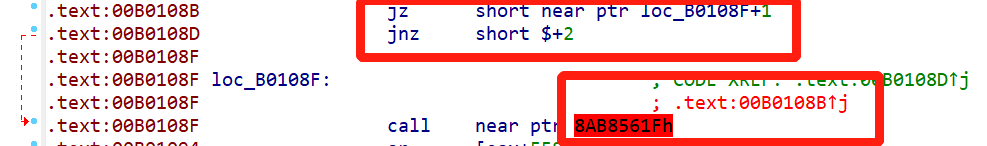
跟IDA一样输入字符串搜索后双击需要看的行数据，之后下断点，一直按F9步步运行



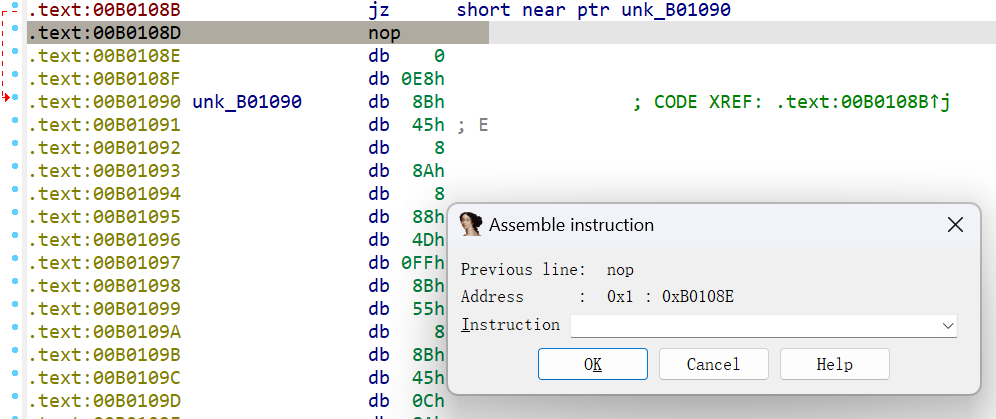
## 花指令

### **1.JZ/JNZ花指令**

JZ/JNZ花指令如下



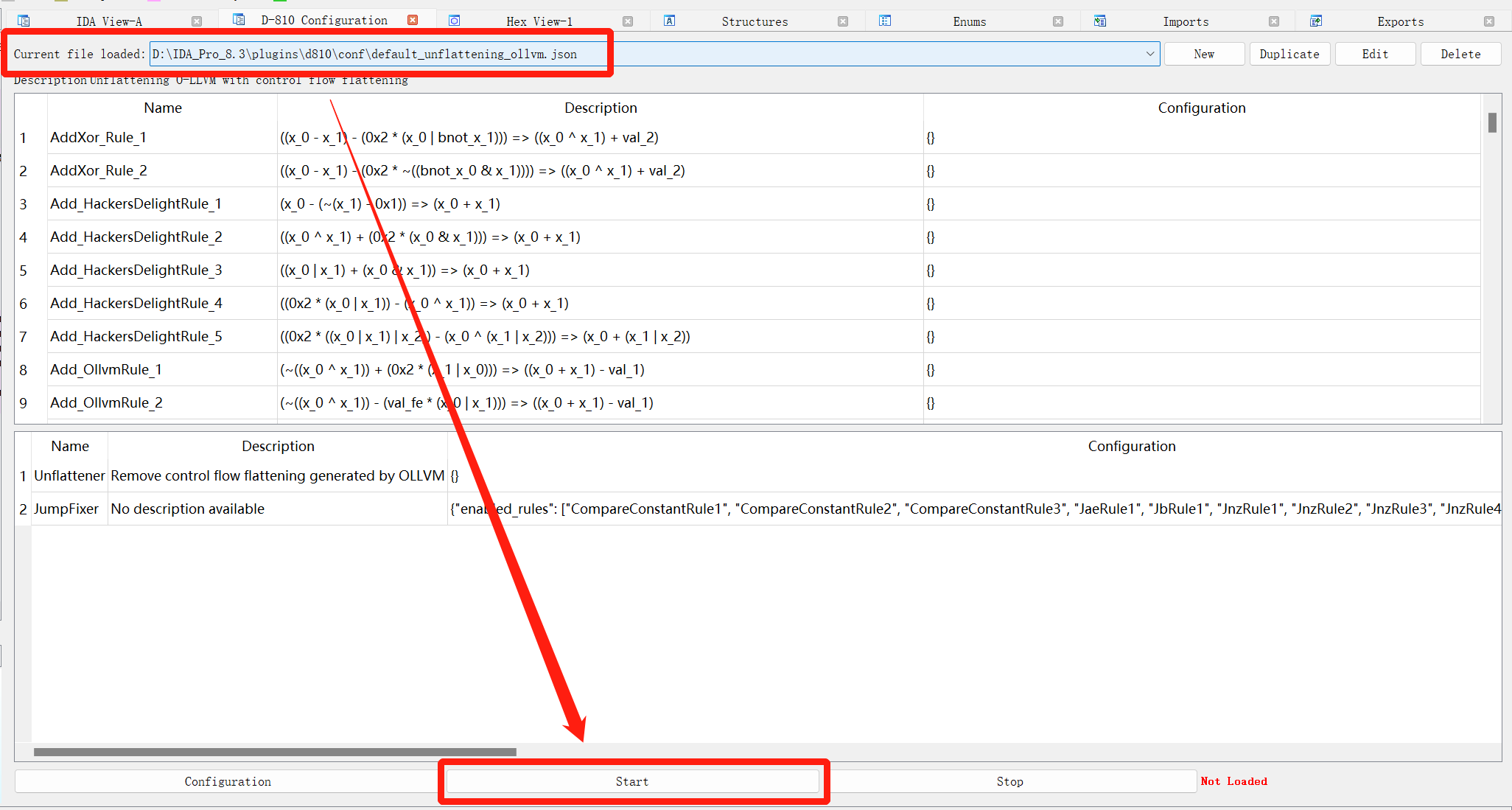
按U展开JNZ，选中75h->点击Edit->选择Patch program->assemble，改为nop



之后在左侧爆红头部按P识别为函数

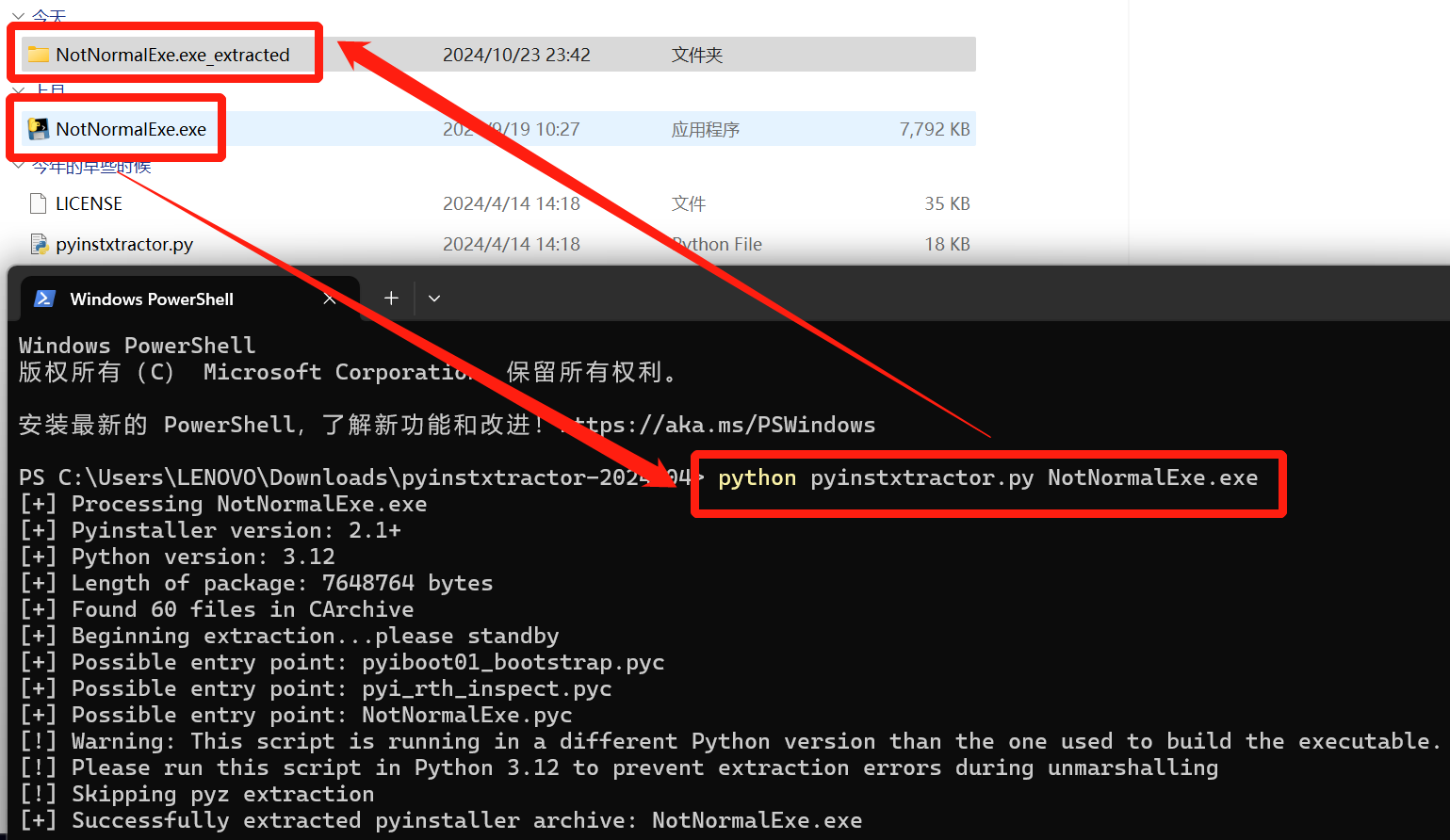
## 控制流平坦化

使用D810去除控制流平坦化：在IDA主界面按Ctrl+Shift+D，之后选择对应的Current file loaded，点击Start

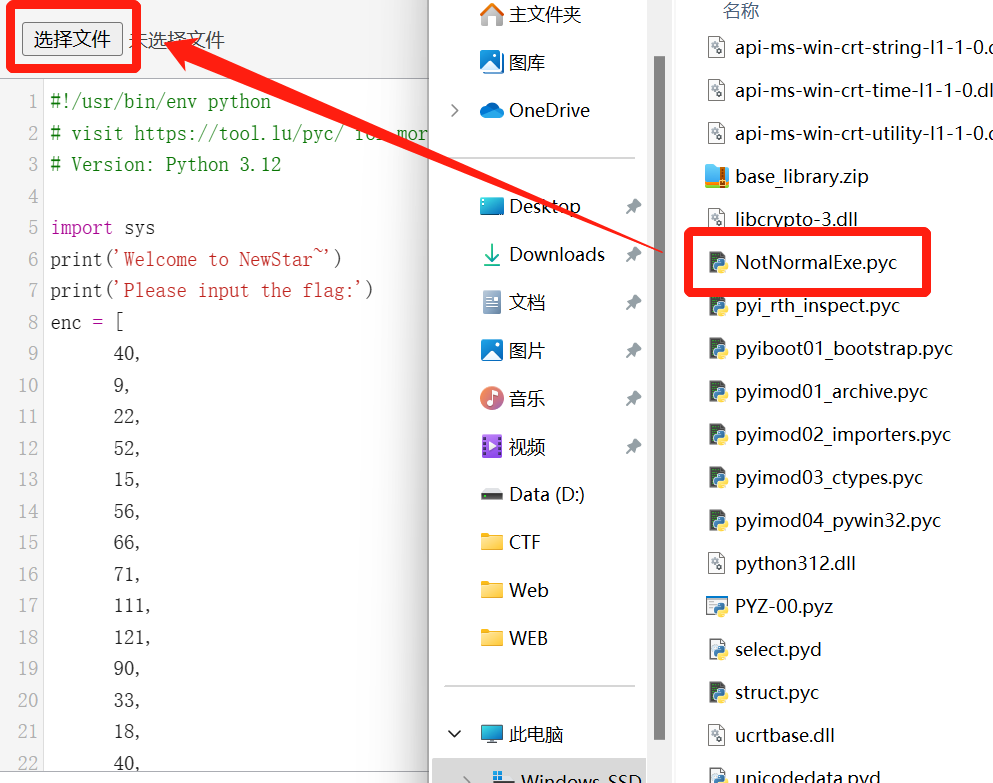


## PyInstaller Extractor

如果IDA打开文件中看到python相关特征，在pyinstxtractor.py文件夹打开cmd，输入：python pyinstxtractor.py 文件名，得到解析文件夹



其中的pyc文件可以使用网站：https://tool.lu/pyc/，选择文件后自动解析出代码



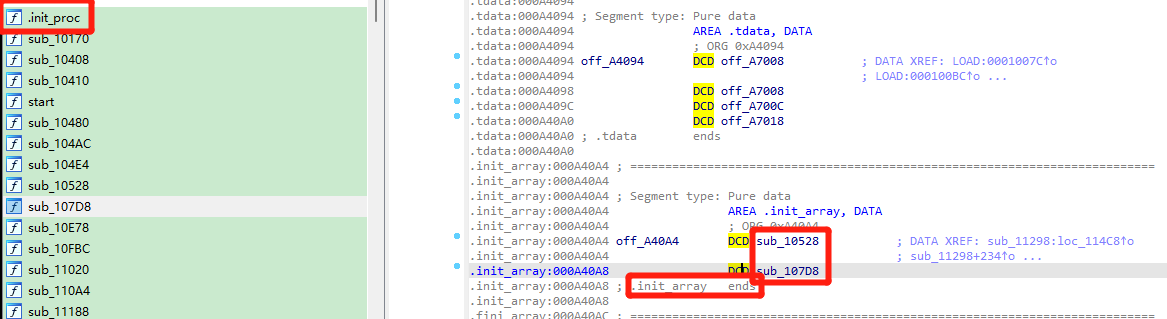
## WinRAR

[CVE-2023-38831(WinRAR漏洞：获取点击的文件 → 释放到临时目录 → 执行)](https://mp.weixin.qq.com/s?__biz=MjM5NTc2MDYxMw==&mid=2458544969&idx=1&sn=473822d99738dc8c20cf0c7df866adea" \t "https://ns.openctf.net/wp/2024/week4/reverse/_blank)：在函数0x140089948中存在漏洞，溯源链为：sub\_1400EF508->sub\_140009290->sub\_14000A650->sub\_1400D6070->0x1400D6478->0x1400D3474->0x1400CEBF4->0x140077054->0x140089948



## ARM

ARM程序中注意如果存在init\_array段，其中的函数会在主函数前执行，覆盖数据



## 杂七杂八的打法

数据可以逆序尝试