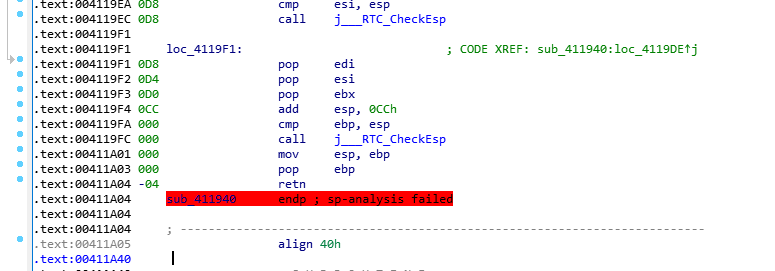
# 题目分析

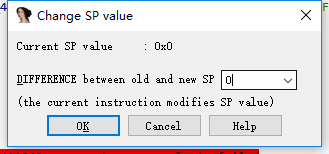
查壳发现是加了upx壳。

使用upx –d xxxxx 命令脱壳后拖进ida进行分析。

分析时发现这里因为栈指针不平衡不能进行反编译。

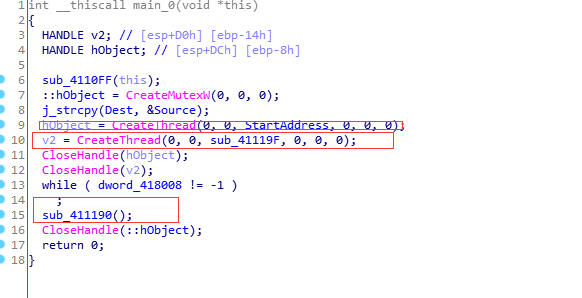


我们在0x00411A03一行右击选择change stack pointer改0x4为0 。



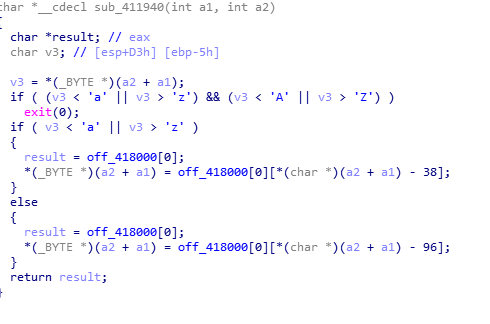
然即可正常反编译。

然后看主程序逻辑。



程序逻辑很简单，先开启了两个线程，然后sub\_411190函数是最后对比数据的地方。

StartAddress里面的逻辑也很简单，对大小写字母进行不同处理。



但是这时候并不能直接开始写脚本得flag，因为有第二个线程这个骚操作。

事情似乎变得有趣了，CreateThread API 会创建新线程，这道题涉及到多线程。CreateMutex 创建一个互斥体，用于防止多线程中出现资源争用，即多个线程同时读写同一个资源的情况，所创建的互斥体的句柄会存到全局变量 hObject 中（注意前面的两个冒号表示是全局变量，而不是这个函数里同名的局部变量）。这里创建了两个线程，入口点分别位于函数 StartAddress 和 sub\_41119F，且这两个函数都没有传入参数。

查 MSDN 知，可以用 WaitForSingleObject 等待互斥体的使用权（ownership）空闲出来，并获取使用权，然后再访问和其他线程共享的资源，访问完后，用 ReleaseMutex 释放使用权，给其他线程使用的机会4。通过比较两线程的函数，很容易知道所共享的资源就是全局变量 dword\_418008，它的初值是 29。而这两个线程一前一后创建，理论上是 StartAddress 先获得使用权，后来的 sub\_41119F 进入等待状态，前者执行一次循环后释放使用权，与此同时后者等待结束、获得使用权，进入循环，循环完后释放使用权，前者又获得使用权，如此循环往复。也就是说，两个线程的操作是交替进行的。但是由于操作系统对线程的调度取决于当时的环境，实际情况可能不同。

简单地说就是，线程hObject是减个下标dword\_418008然后进行加密，线程v2就是减个下标dword\_418008。两个线程轮流进行，而dword\_418008初始值为0x1d即29，这就相当于如果dword\_418008是奇数，进行加密，如果是偶数，密文和明文相同。

然后写出脚本即可。

# 脚本

str1 = "QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm"  
str2 = "TOiZiZtOrYaToUwPnToBsOaOapsyS"  
flag = ""  
for i in range(len(str2)):  
 if i%2 == 0:  
 flag += str2[i]  
 else:  
 if((ord(str2[i]) >= ord('a')) & (ord(str2[i]) <= ord('z'))):  
 flag += chr(str1.find(str2[i]) + 38)  
 else:  
 flag += chr(str1.find(str2[i]) + 96)  
print flag

得到ThisisthreadofwindowshahaIsES

之后还有个坑，下标从29开始的话意味着输入长度有30，而在函数sub\_411880进行check时只比较前29个字符，相等即正确。那么我们只需还原明文的前29位，加上一个任意字母即可。