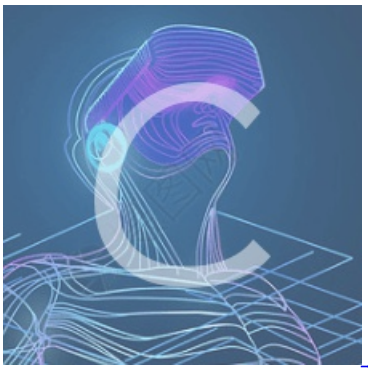
BUUCTF_Youngter-drive

原创

ZYen12138 于 2020-10-30 22:59:26 发布 322 收藏 1

分类专栏： # BUUCTF CTF

本文链接：https://blog.csdn.net/weixin_46009088/article/details/109362576



BUUCTF 同时被 2 个专栏收录

17 篇文章 2 订阅

订阅专栏

CTF

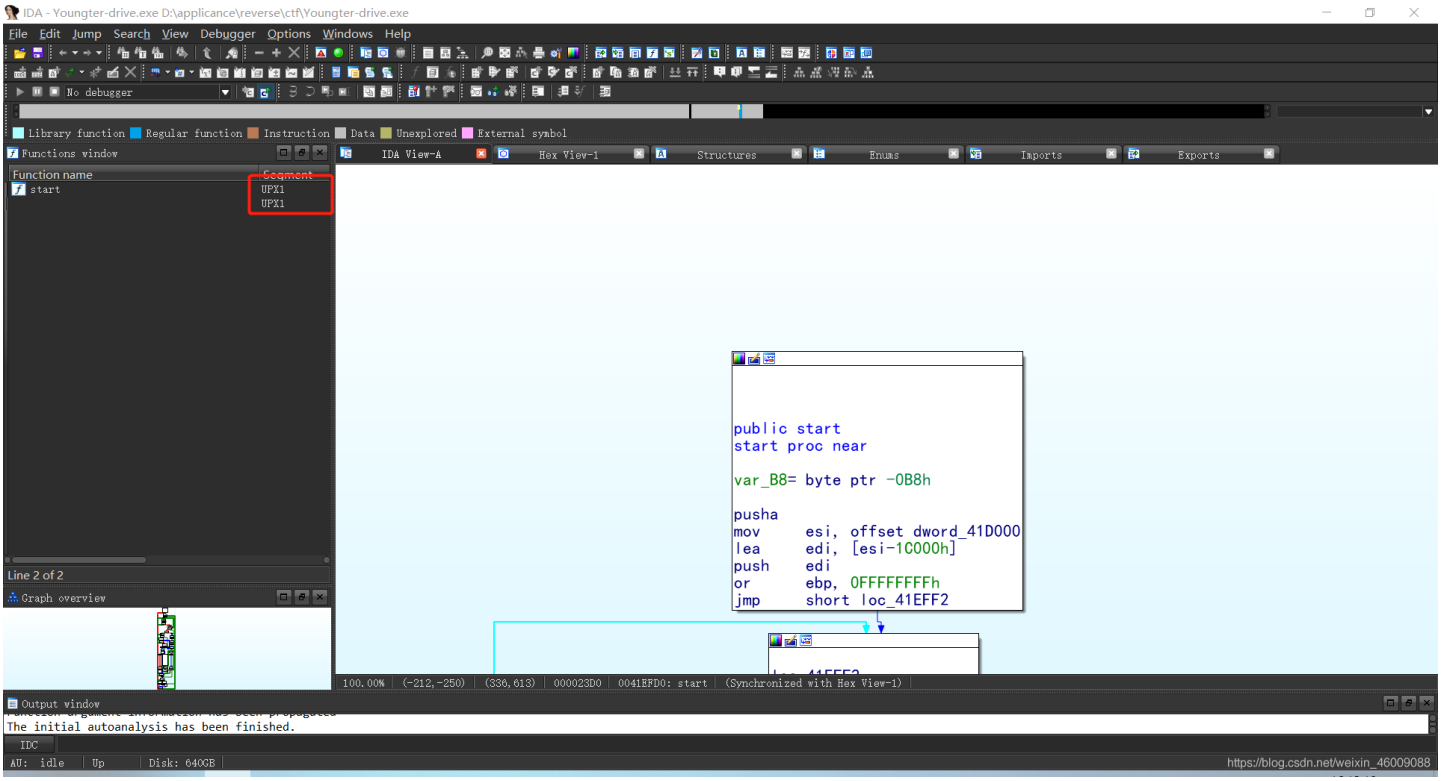
17 篇文章 0 订阅

订阅专栏

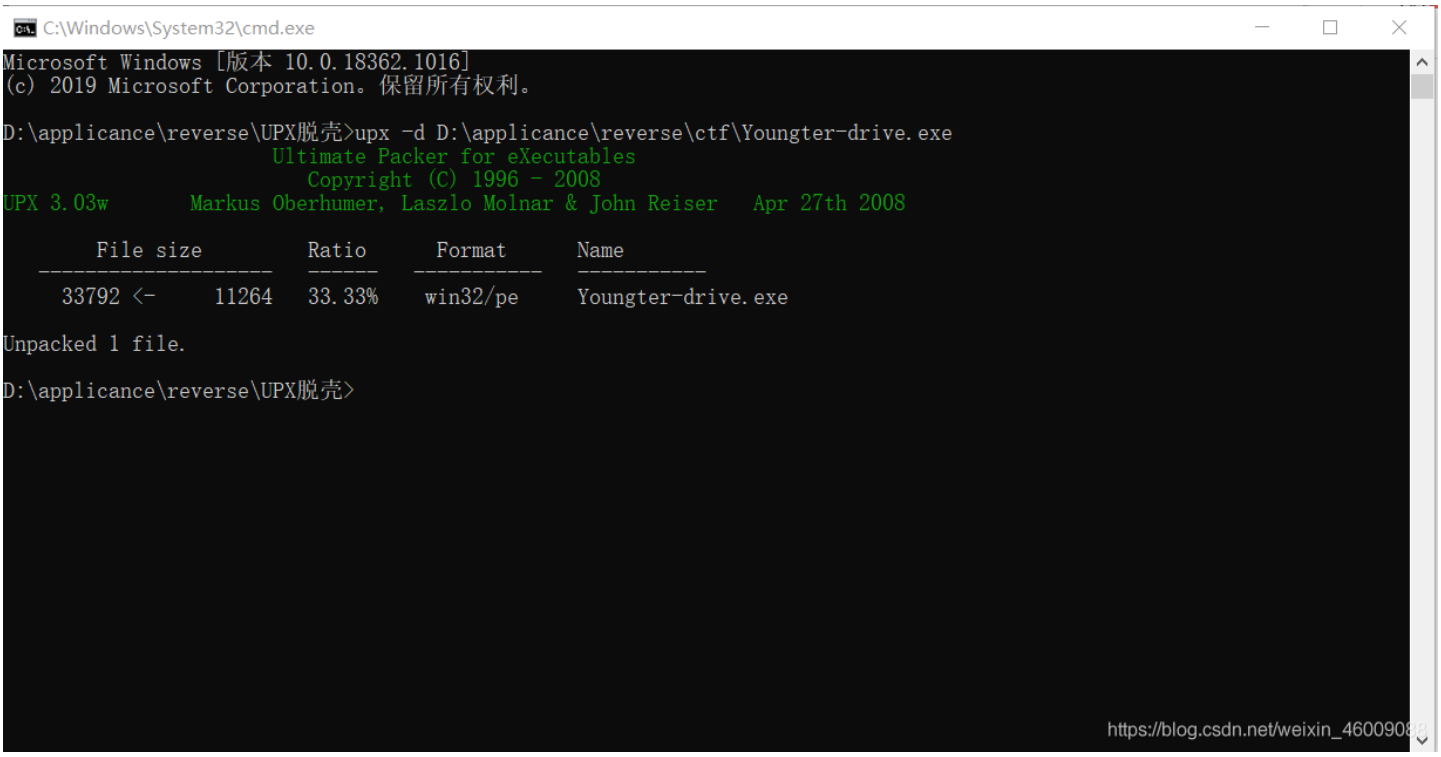
BUUCTF_Youngter-drive

学到一些多线程和IDA平衡堆栈的知识！同样也是尽量写的详细！

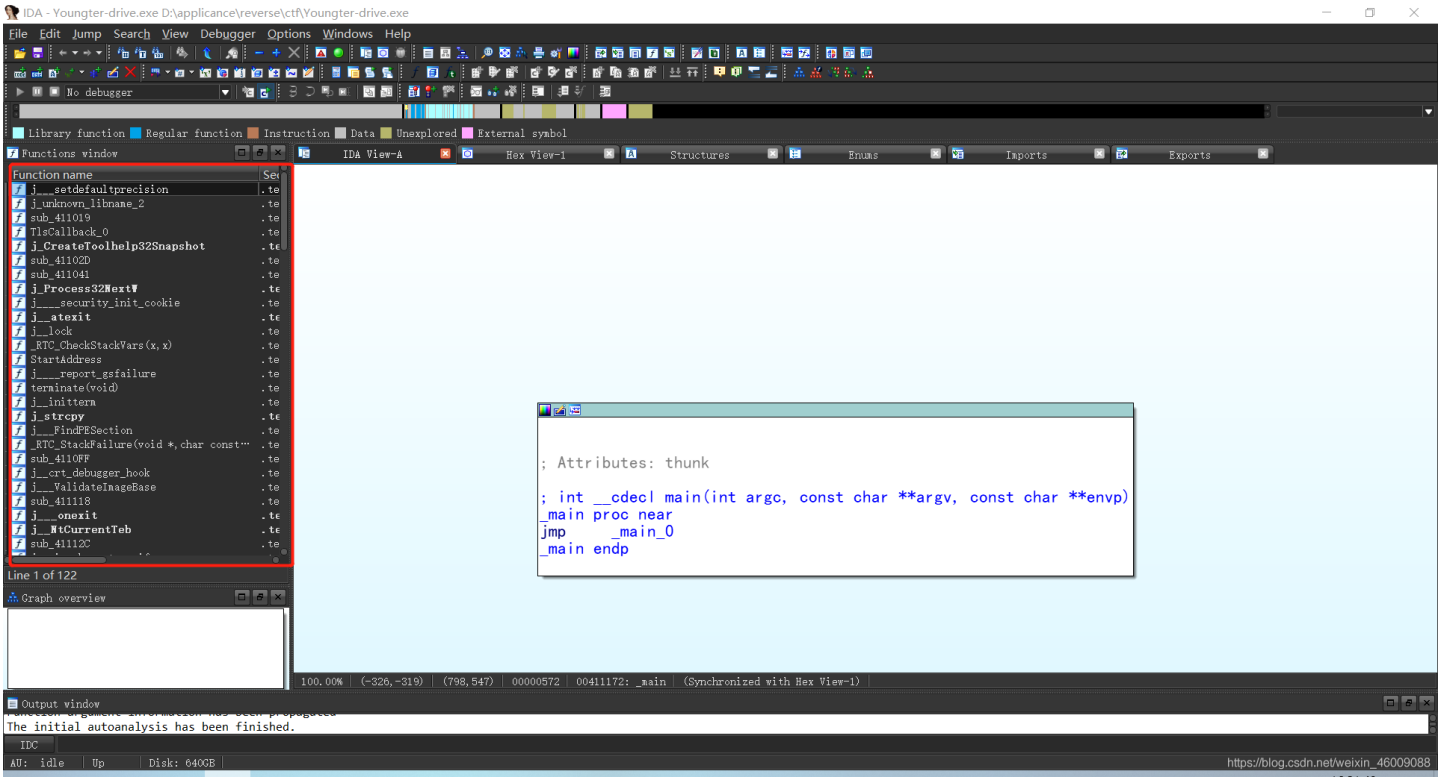
解压后拖进IDA，发现UPX的壳，如图：



因为是exe文件，所以在windows上就可以用upx -d 把它给脱壳了，如图：



接下来就可以用IDA载入了，如图：



找到main_0函数就可以用F5大法了

```
1  int main_0()
2  {
3    HANDLE v1; // [esp+D0h] [ebp-14h]
4    HANDLE hObject; // [esp+DCh] [ebp-8h]
5
6    sub_4110FF();
7    ::hObject = CreateMutexW(0, 0, 0);
8    j_strcpy(Dest, Source);
9    hObject = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress, 0, 0, 0);// 创建线程
10   v1 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)sub_41119F, 0, 0, 0);// 创建线程
11   CloseHandle(hObject);                         // 关闭线程
12   CloseHandle(v1);                              // 关闭线程
13   while ( dword_418008 != -1 )
14     ;
15   sub_411190();                                 // 判断加密后的结果是否正确
16   CloseHandle(::hObject);
17   return 0;
18 }
```


一个函数一个函数来看，先看到sub_4110FF，点进去看看：

```c
int sub_411BD0()
{
  printf(
    "1111111111111111111111111111111111111111111111111111111111111111111111111111111\n"
    "*******************************************************************************\n"
    "**************             ****************************************************\n"
    "**************   ********   *********************                 *************\n"
    "**************   *********  *********************   ***************************\n"
    "**************   *********  *********************   ***************************\n"
    "**************   *********  *********************   ***************************\n"
    "**************   *******   **********************   ***************************\n"
    "**************   ****   *************************   ***************************\n"
    "**************   *    ***************************                **************\n"
    "**************   ***    *************************   ***************************\n"
    "**************   ******   ***********************   ***************************\n"
    "**************   ********   *********************   ***************************\n"
    "**************   **********   *******************   ***************************\n"
    "**************   ***********    *****************                 *************\n"
    "*******************************************************************************\n"
    "1111111111111111111111111111111111111111111111111111111111111111111111111111111\n");
  printf("input flag:\n");
  return scanf("%36s", Source);
}
```


source应该是我们的flag,其他没啥东西，往下看，CreateThread创建了两个线程，MSDN文档1如下：

CreateThread函数(处理器线程.h)

12/05/2018 • 5分钟阅读

创建要在调用进程的虚拟地址空间内执行的线程。

若要创建在另一个进程的虚拟地址空间中运行的线程，请使用CreateRemoteThread功能。

句法

C++ Copy

```cpp
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES   lpThreadAttributes,
  SIZE_T                  dwStackSize,
  LPTHREAD_START_ROUTINE  lpStartAddress,
  __drv_aliasesMem LPVOID lpParameter,
  DWORD                   dwCreationFlags,
  LPDWORD                 lpThreadId
);
```

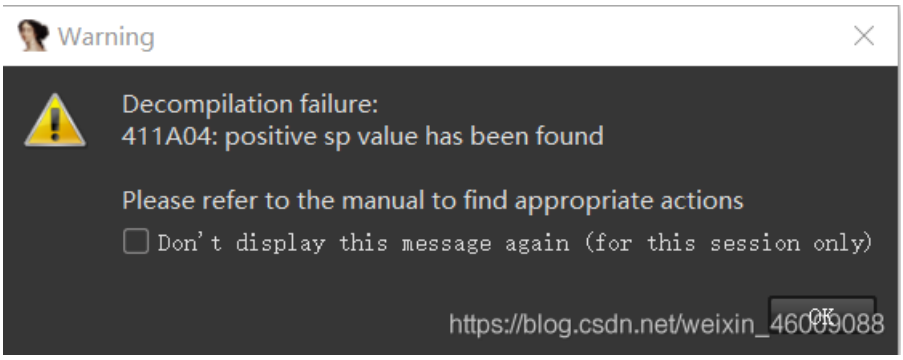

点进StartAddress查看线程，如图：

```
void __stdcall StartAddress_0(int a1)
{
  while ( 1 )
  {
    WaitForSingleObject(hObject, 0xFFFFFFFF);
    if ( dword_418008 > -1 )
    {
      sub_41112C((int)&Source, dword_418008);
      --dword_418008;
      Sleep(0x64u);
    }
    ReleaseMutex(hObject);
  }
}
```


发现一个函数，点进去看，如图：

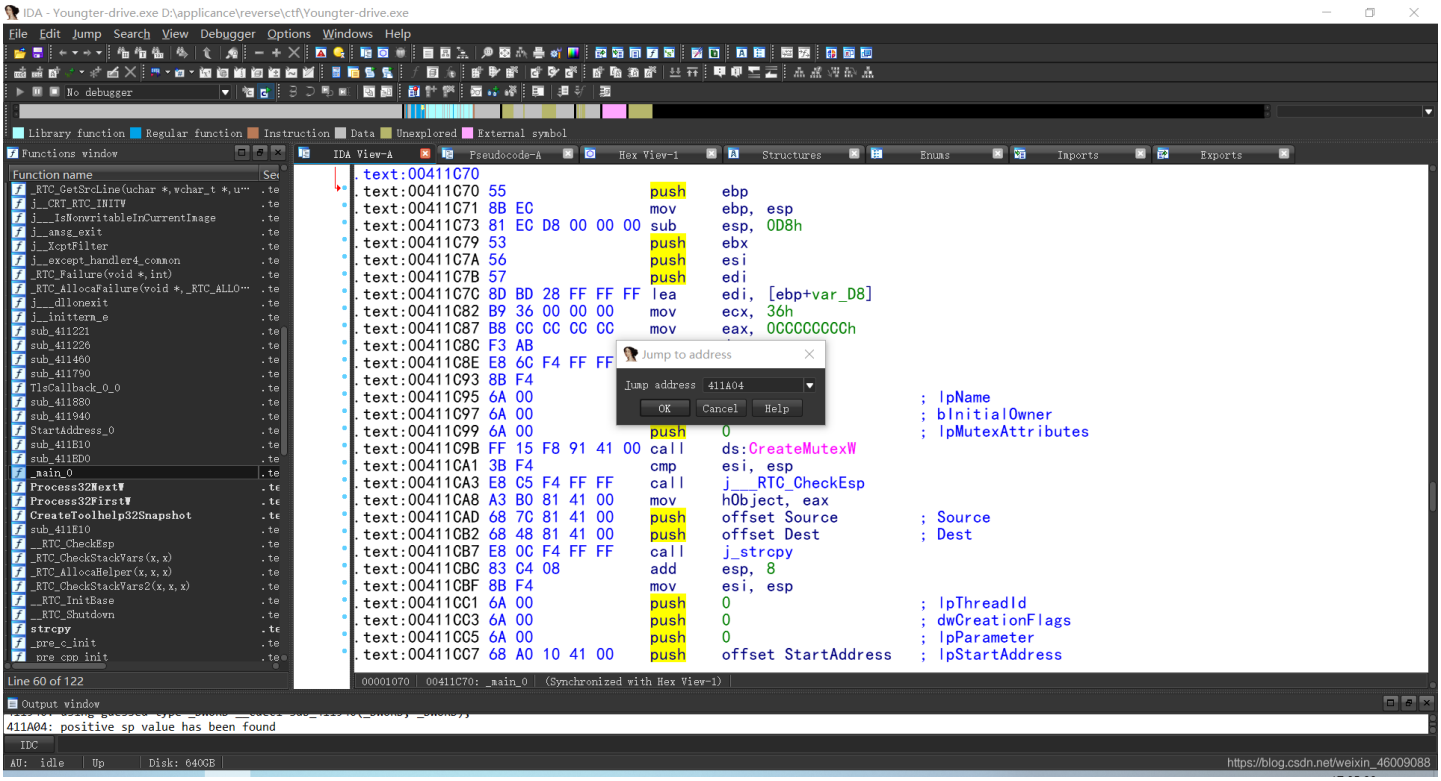
```
sub_41112C((int)&Source, dword_418008);
```

但是弹出了下面的错误：



翻译一下：

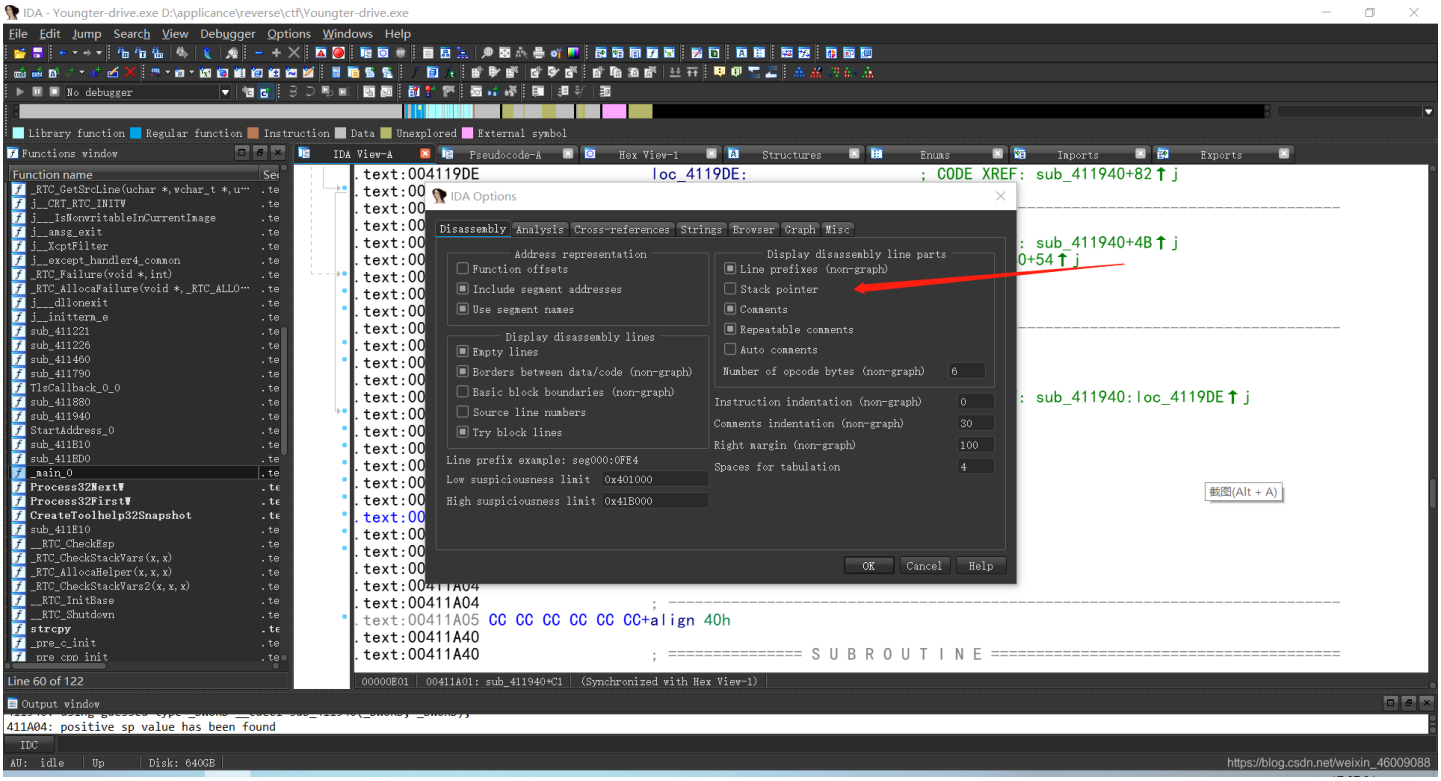
总的来说就是：411A04这个地址的堆栈不平衡，再说的通俗一点，就是IDA发现了一个SP的实际的偏移值是负数，需要我们手动去矫正它，我们按G跳转到411A04修改堆栈的值：



问题便出在这，如图：

```
.text:004119F1                  loc_4119F1:                             ; CODE XREF: sub_411940:loc_4119DE↑j
.text:004119F1 5F               pop     edi
.text:004119F2 5E               pop     esi
.text:004119F3 5B               pop     ebx
.text:004119F4 81 C4 CC 00 00 00 add     esp, 0CCh
.text:004119FA 3B EC            cmp     ebp, esp
.text:004119FC E8 6C F7 FF FF   call    j___RTC_CheckEsp
.text:00411A01 8B E5            mov     esp, ebp
.text:00411A03 5D               pop     ebp
.text:00411A04 C3               retn
.text:00411A04                  sub_411940 endp ; sp-analysis failed
```

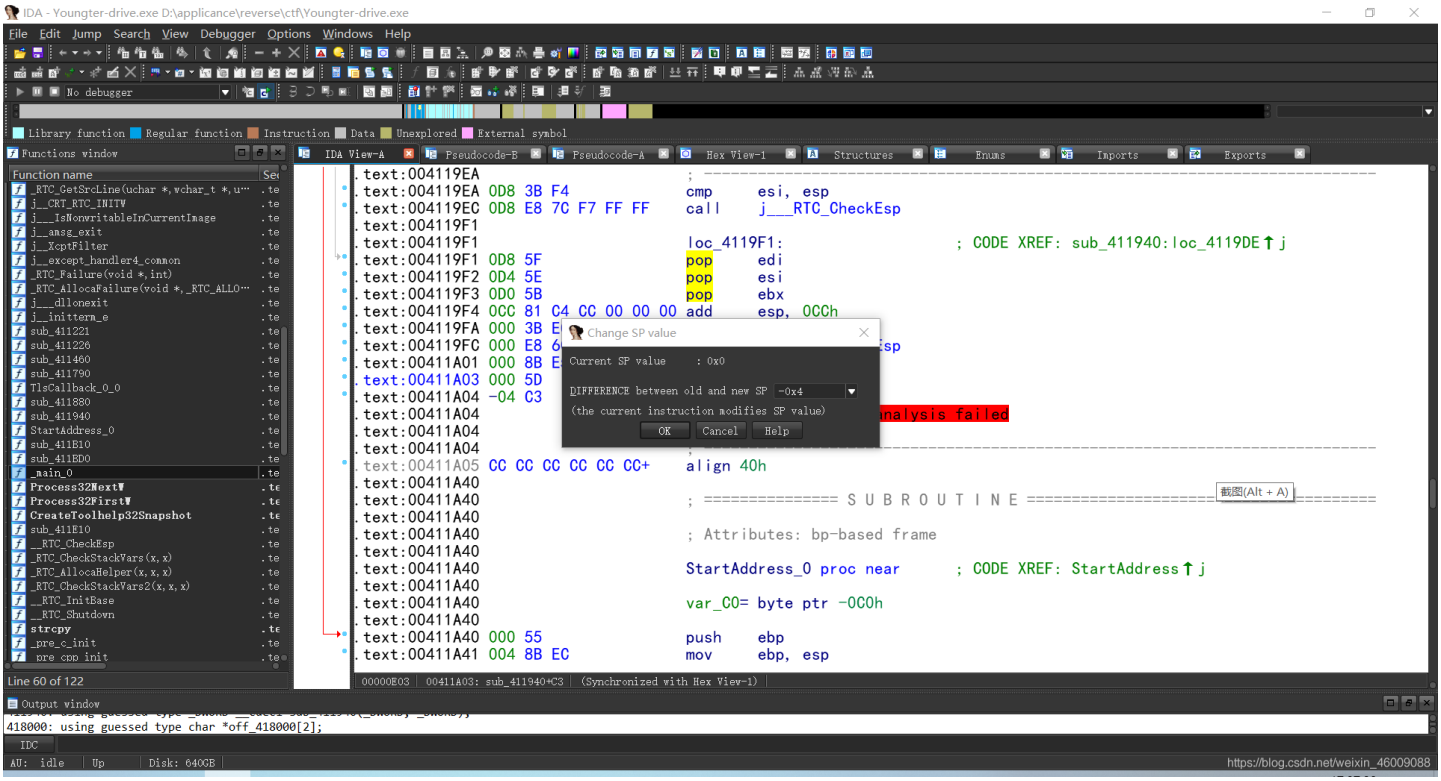

点击Option -> Genereal -> Disassembly，把箭头所指向的选项勾上就可以看到SP的实际的偏移值 (左边绿色那一排)



下图一个很明显的负数，我们只要将上条指令的SP进行修正就可以了

```
.text:004119F1                          loc_4119F1:                             ; CODE XREF: sub_411940:loc_4119DE↑j
.text:004119F1 0D8 5F                   pop     edi
.text:004119F2 0D4 5E                   pop     esi
.text:004119F3 0D0 5B                   pop     ebx
.text:004119F4 0CC 81 C4 CC 00 00 00    add     esp, 0CCh
.text:004119FA 000 3B EC                cmp     ebp, esp
.text:004119FC 000 E8 6C F7 FF FF       call    j___RTC_CheckEsp
.text:00411A01 000 8B E5                mov     esp, ebp
.text:00411A03 000 5D                   pop     ebp
.text:00411A04 -04 C3                   retn
.text:00411A04                          sub_411940 endp ; sp-analysis failed
```


按ALT+K，在SP值的前面加给负号修正它的负值



我们发现他的SP值变成了正值

```
text:004119F1                           loc_4119F1:                     ; CODE XREF: sub_411940:loc_4119DE↑j
text:004119F1 0D8 5F                    pop     edi
text:004119F2 0D4 5E                    pop     esi
text:004119F3 0D0 5B                    pop     ebx
text:004119F4 0CC 81 C4 CC 00 00 00     add     esp, 0CCh
text:004119FA 000 3B EC                 cmp     ebp, esp
text:004119FC 000 E8 6C F7 FF FF        call    j___RTC_CheckEsp
text:00411A01 000 8B E5                 mov     esp, ebp
text:00411A03 000 5D                    pop     ebp
text:00411A04 004 C3                    retn
text:00411A04                           sub_411940 endp ; sp-analysis failed
```


现在就可以进行F5大法了，如图（注释尽力了T-T）：

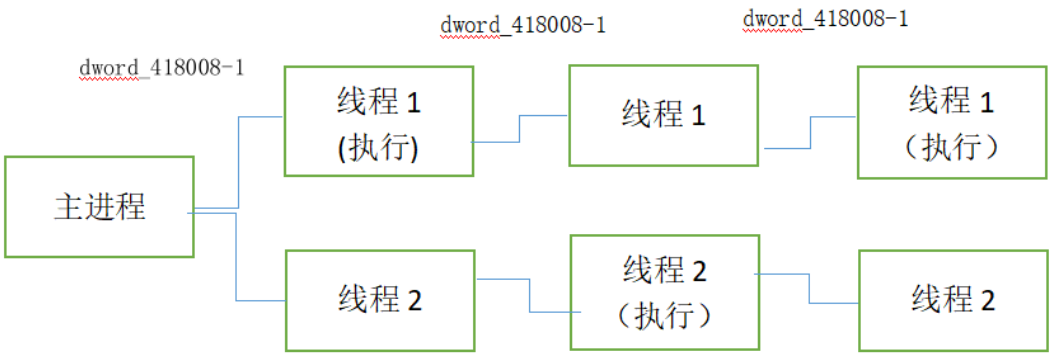
```
// a1指向了我们的flag,接下来将进行一些变换得到（两个线程）TOiZiZtOrYaToUwPnToBsOaOapsyS
// dword_418008 == 29，每运行一次将dword_418008减1，也就是我们的a2，所以它是从后面向前面遍历
// 最后用sub_411190对字符串校验
// 所以我们通过对算法的逆向将得到我们的a1,也就我们要的flag
char *__cdecl sub_411940(int a1, int a2)
{
  char *result; // eax
  char v3; // [esp+D3h] [ebp-5h]

  v3 = *(_BYTE *)(a2 + a1);                     // a1是flag的地址，而dword_418008是1D（29）也就是v3指向了字符串的末尾，有点像a1[a2]
  if ( (v3 < 97 || v3 > 122) && (v3 < 65 || v3 > 90) )// 如果它不是字母将退出
    exit(0);
  if ( v3 < 97 || v3 > 122 )                    // 如果是大写字母
  {
    result = off_418000[0];                     // off_418000 == QWERTYUIOPASDFGHJKLZXCUBNMqwertyuiopasdfghjklzxcvbnm
    *(_BYTE *)(a2 + a1) = off_418000[0][*(char *)(a2 + a1) - 38];// source - 38 , 也就是？ - 38 = source[29]
  }
  else
  {
    result = off_418000[0];
    *(_BYTE *)(a2 + a1) = off_418000[0][*(char *)(a2 + a1) - 96];// 同理~
  }
  return result;
}
```


现在看下一个线程sub_41119F

```
void __stdcall sub_411B10(int a1)
{
  while ( 1 )
  {
    WaitForSingleObject(hObject, 0xFFFFFFFF);
    if ( dword_418008 > -1 )
    {
      Sleep(0x64u);
      --dword_418008;
    }
    ReleaseMutex(hObject);
  }
}
```


并没有做什么，只是占着线程的计算器，每次运行dword_418008减1，现在理一下思路，线程1改变了字符串，而线程2没有改变，也就是说奇数改变偶数不改变，直到dword_418008减为0 ，线程结束，主进程也跟着结束.



打开sub_411880看看，将上面两个进程得到的字符串进行了校验.

```
int sub_411880()
{
  int i; // [esp+D0h] [ebp-8h]

  for ( i = 0; i < 29; ++i )
  {
    if ( Source[i] != off_418004[i] )
      exit(0);
  }
  return printf("\nflag{%s}\n\n", Dest);
}
```

现在可以开始写python脚本：

```
flagpart = 'TOiZiZtOrYaToUwPnToBsOaOapsyS'
flagrange = 'QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm'
flag = ''
for i in range(len(flagpart)):
    if i % 2 == 0:
        flag += flagpart[i]
    else:
        if flagpart[i].isupper():
            flag += chr(flagrange.find(flagpart[i]) + 96)
        else:
            flag += chr(flagrange.find(flagpart[i]) + 38)
```

运行得到结果：

```
ThisisthreadofwindowshahaIsES
```

至此，flag已经逆向得到了.
总结：

CreateThreadMSDN文档：https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createthread ↩