XCTF-csaw2013reversing2

原创

cr4ke3 于 2019-12-22 21:03:57 发布 458 收藏 1
分类专栏： XCTF

本文链接：https://blog.csdn.net/weixin_43784056/article/details/103655968

三种做法

1、ida静态分析修改指令

main函数反编译的代码

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  int v3; // ecx
  CHAR *lpMem; // [esp+8h] [ebp-Ch]
  HANDLE hHeap; // [esp+10h] [ebp-4h]

  hHeap = HeapCreate(0x40000u, 0, 0);
  lpMem = (CHAR *)HeapAlloc(hHeap, 8u, MaxCount + 1);
  memcpy_s(lpMem, MaxCount, &unk_409B10, MaxCount);
  if ( sub_40102A() || IsDebuggerPresent() )    // IsDebuggerPresent() 为检测动态调试的函数
  {
    __debugbreak();                             // 中断函数，汇编语言为int 3
    sub_401000(v3 + 4, lpMem);                  // 生成flag的函数
    ExitProcess(0xFFFFFFFF);                    // 退出程序
  }
  MessageBoxA(0, lpMem + 1, "Flag", 2u);        // 弹框输出flag
  HeapFree(hHeap, 0, lpMem);
  HeapDestroy(hHeap);
  ExitProcess(0);
}
```

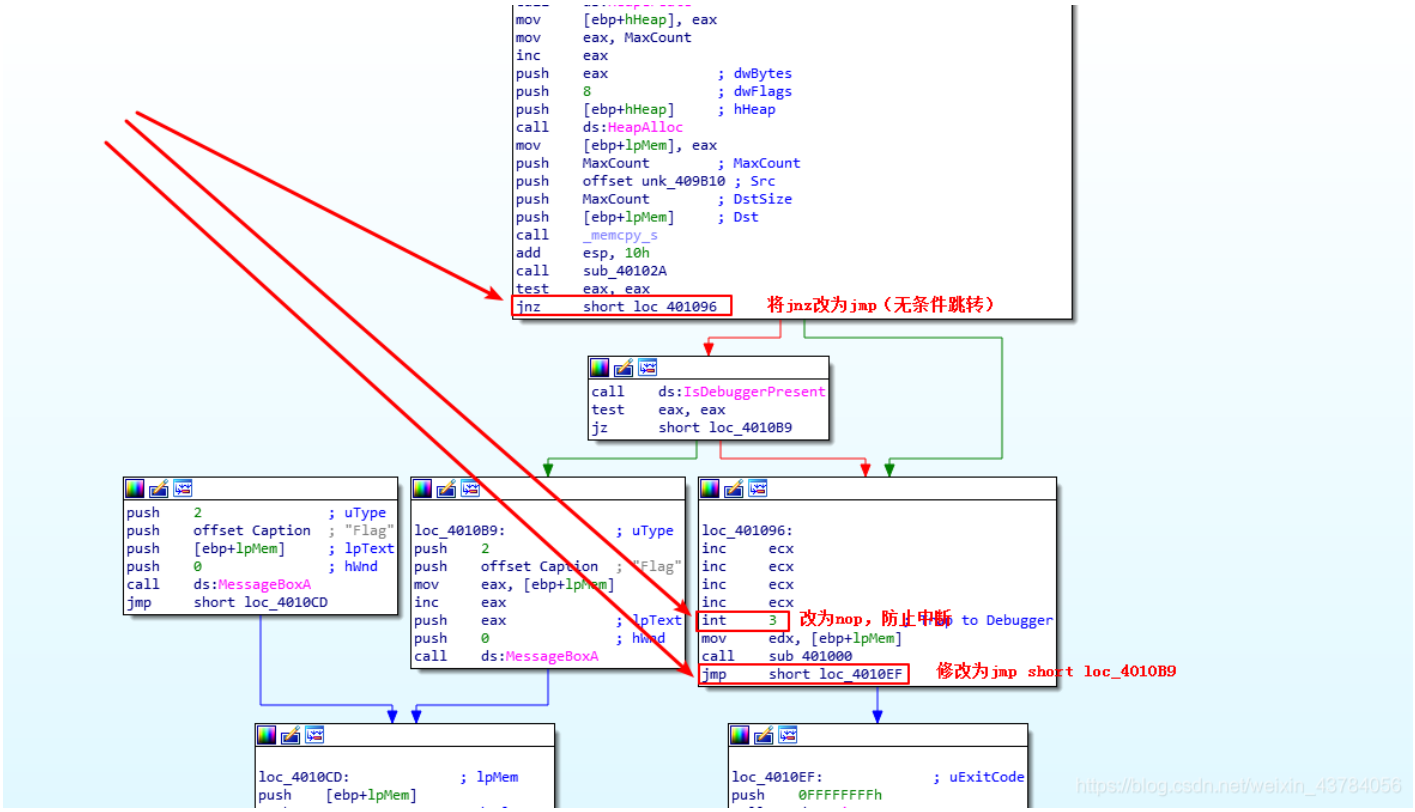

由于运行之后的是乱码，所以可以猜测生成flag的函数没有执行，所以需要跳到生成flag的函数执行，但是前面的中断函数不能执行，需要nop掉，并且后面退出程序的函数不能执行，需要跳到弹框函数继续执行。（修改的路径和文件名不要有中文，我用ida修改的时候踩了坑，大家可以试一试）

ida修改代码的方法：

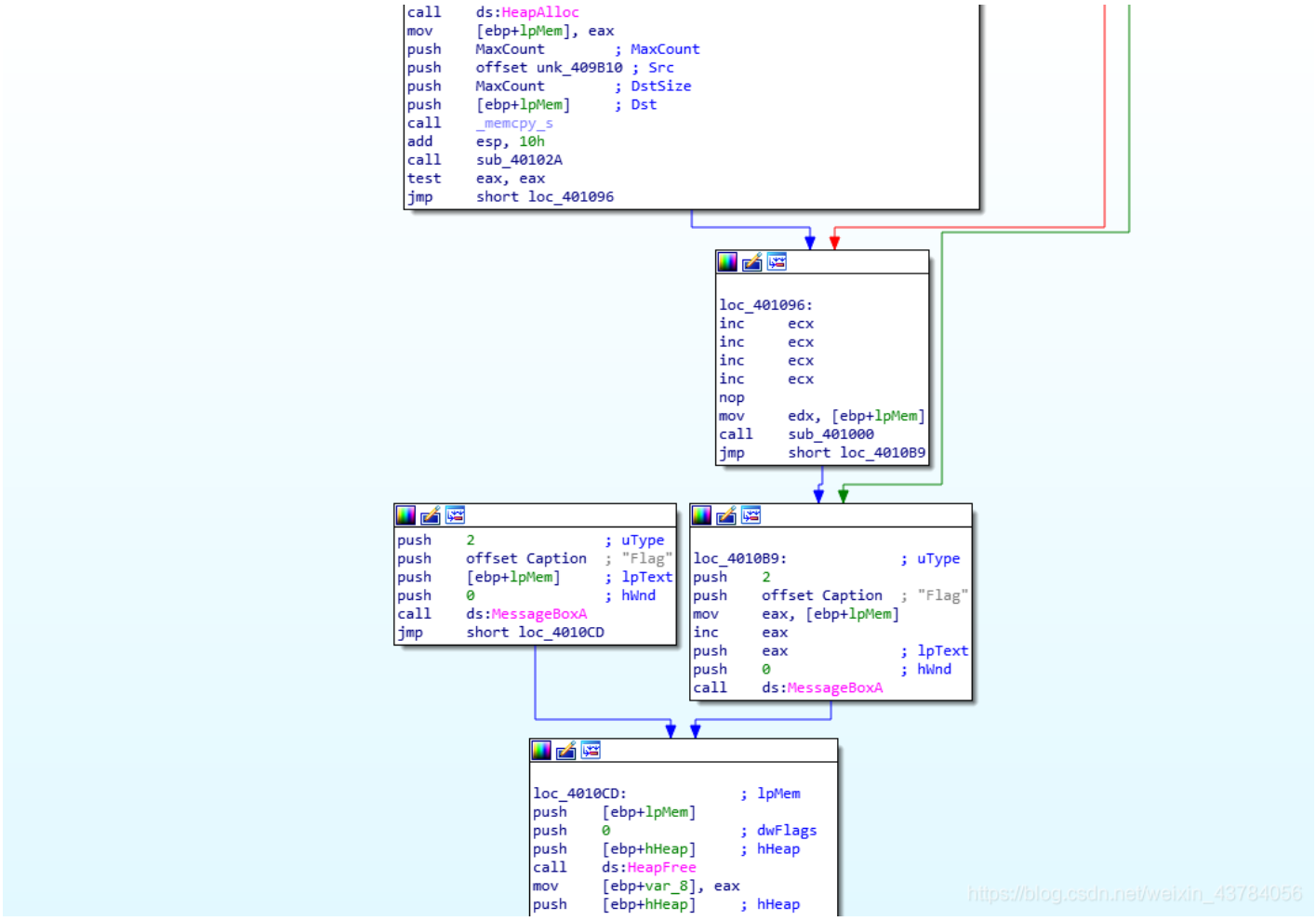
1、鼠标停留在要修改的汇编代码上，然后点击Edit > Patch program > Assemble（中文：编辑 > 修补程序 > 汇编）

2、修改完成后：Edit > Patch program > Apply pathes to input file > OK（中文：编辑 > 修补程序 > 修补程序应用到输入文件 > 确定）

修改之前的汇编代码



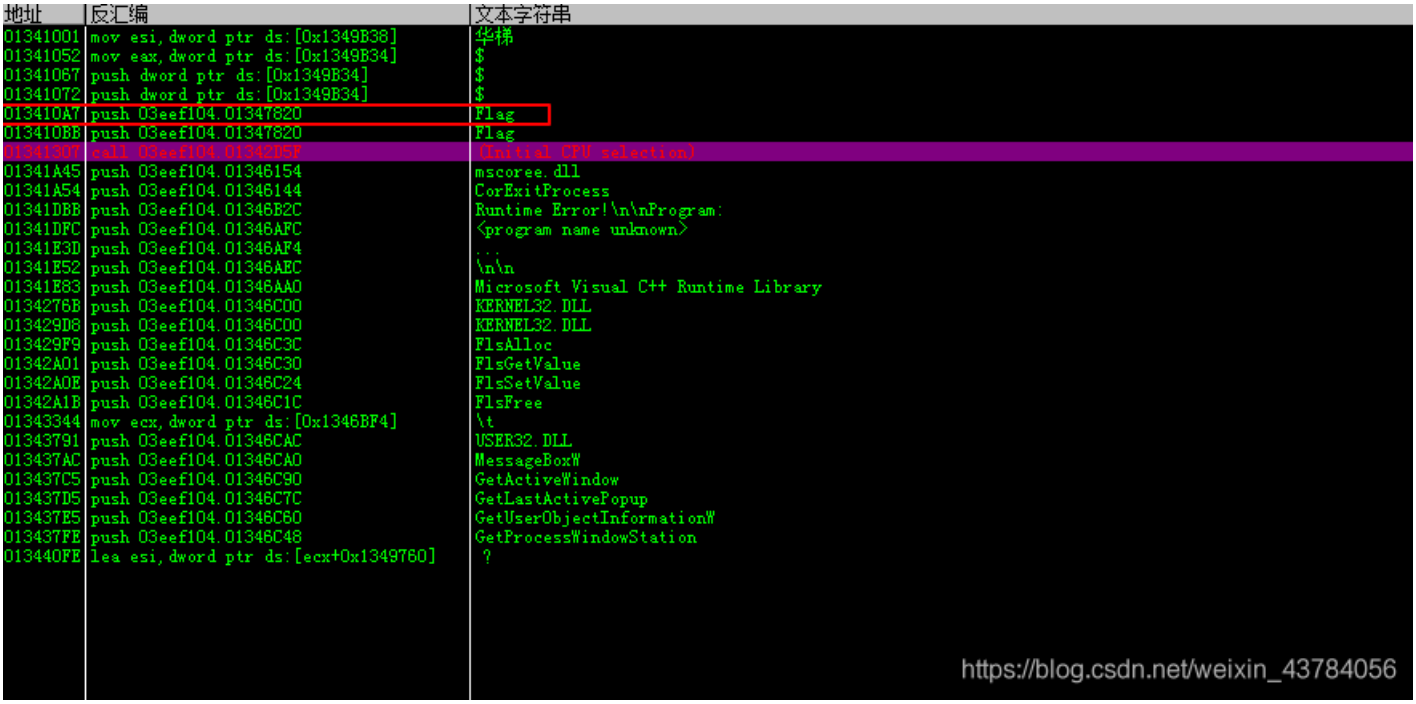
修改之后的汇编代码



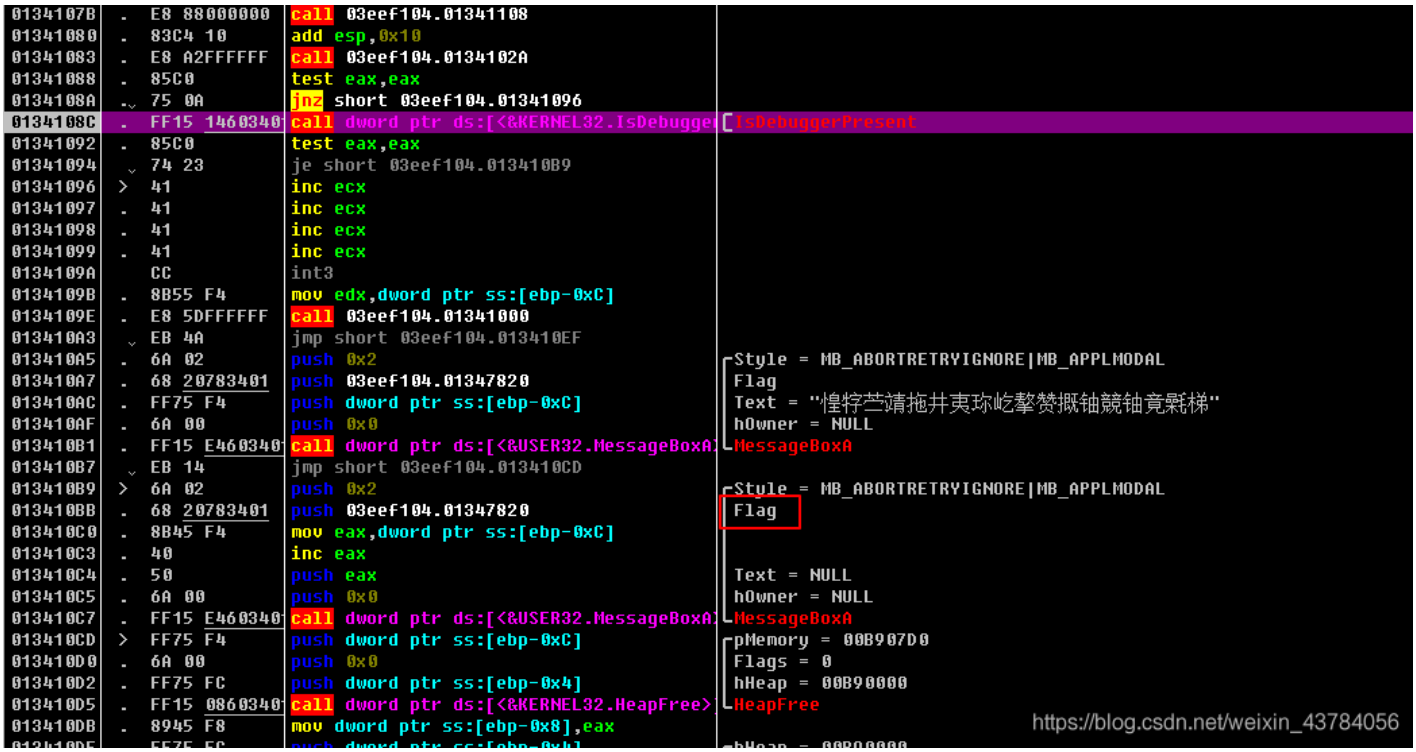
修改完成之后，直接运行文件，得到flag

2、ollydbg动态调试，nop大法

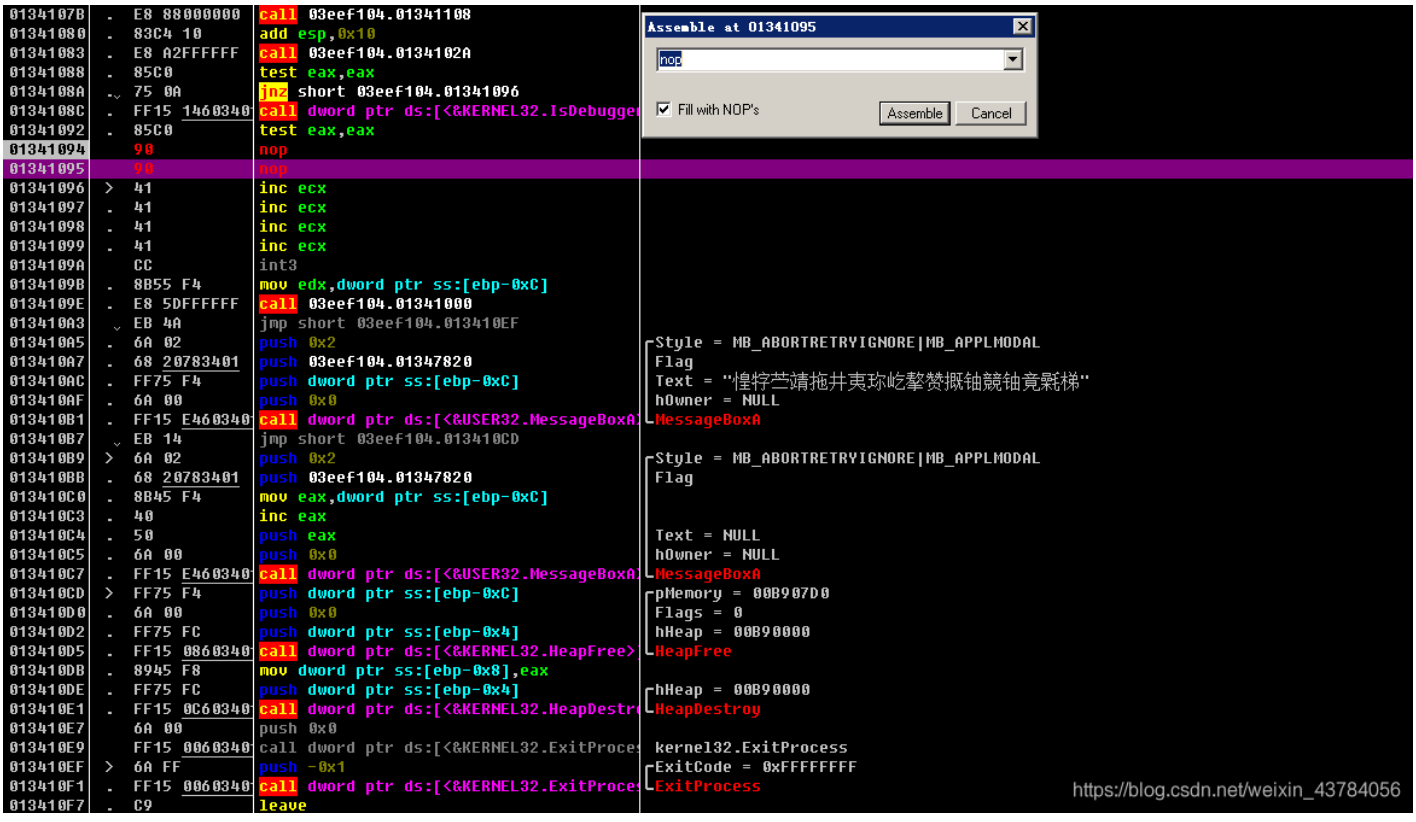
将文件导入ollydbg后，直接右键 > 中文搜索引擎 > 智能搜索，找到Flag



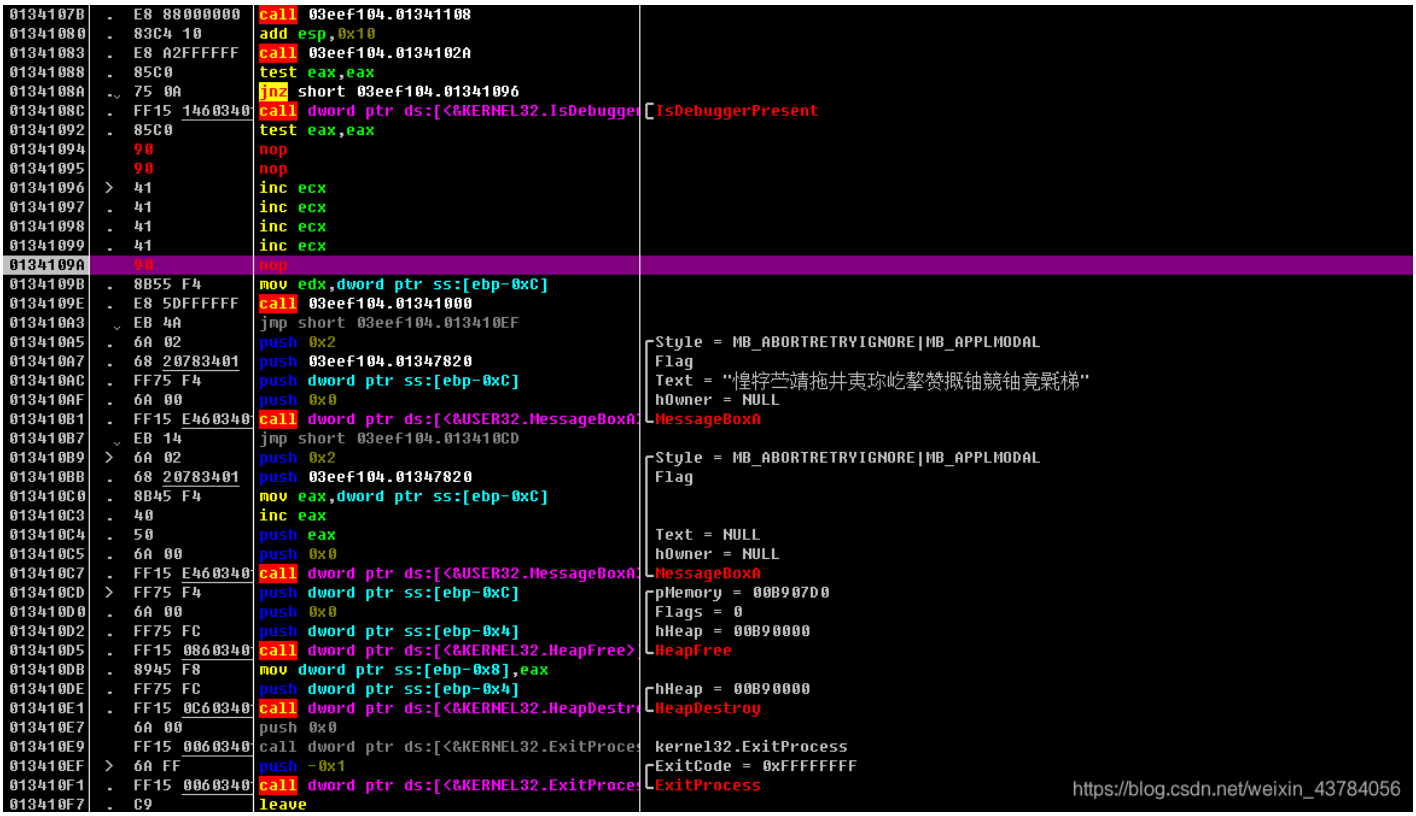
双击之后向上找到IsDebuggerPresent函数，点击这句汇编，F4让程序运行到此处



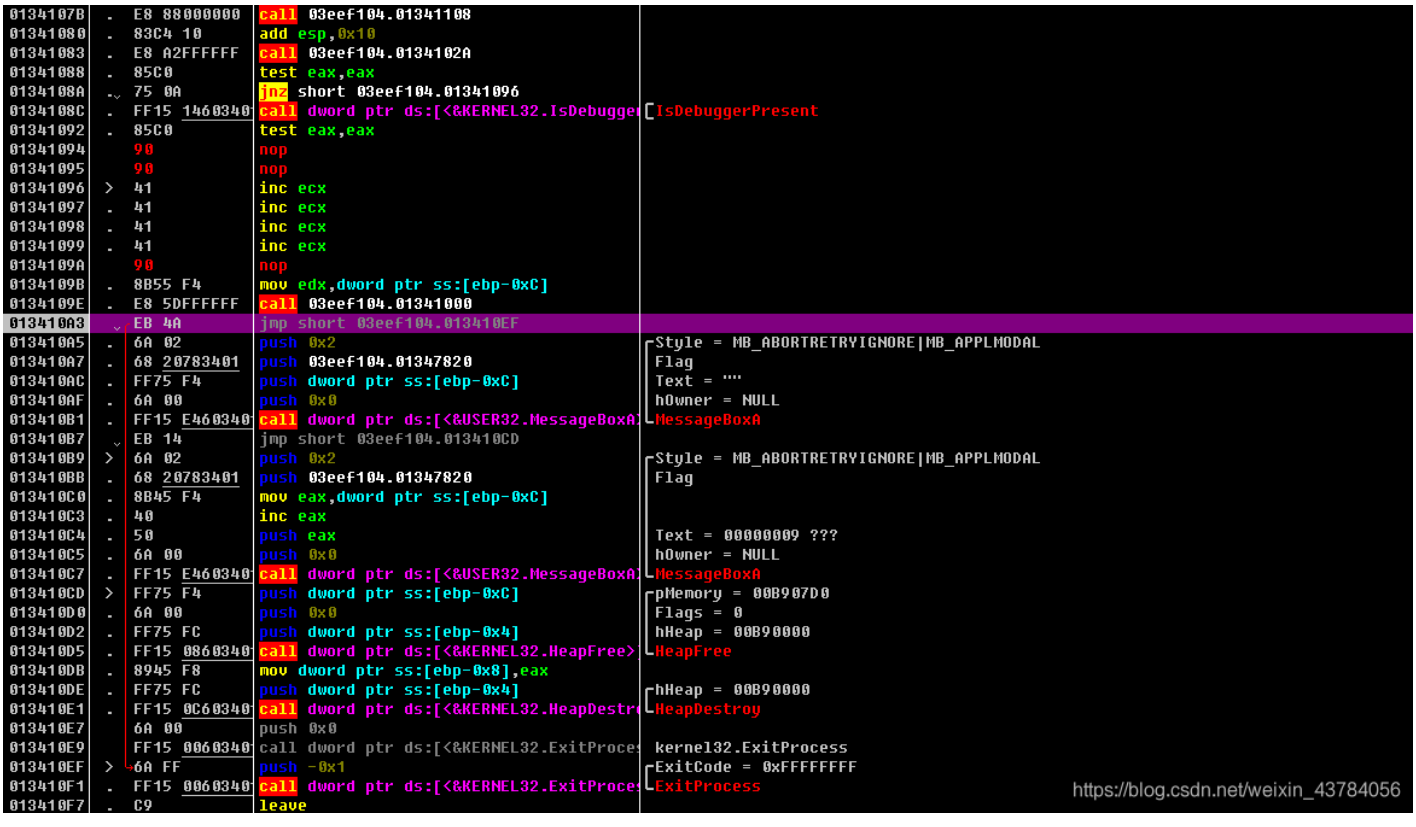
F8两次，发现一个跳转，根据之前ida的分析，这应该就是那个if语句的判断，跳过的中间部分就是生成flag的函数，所以我们把这个跳转nop掉



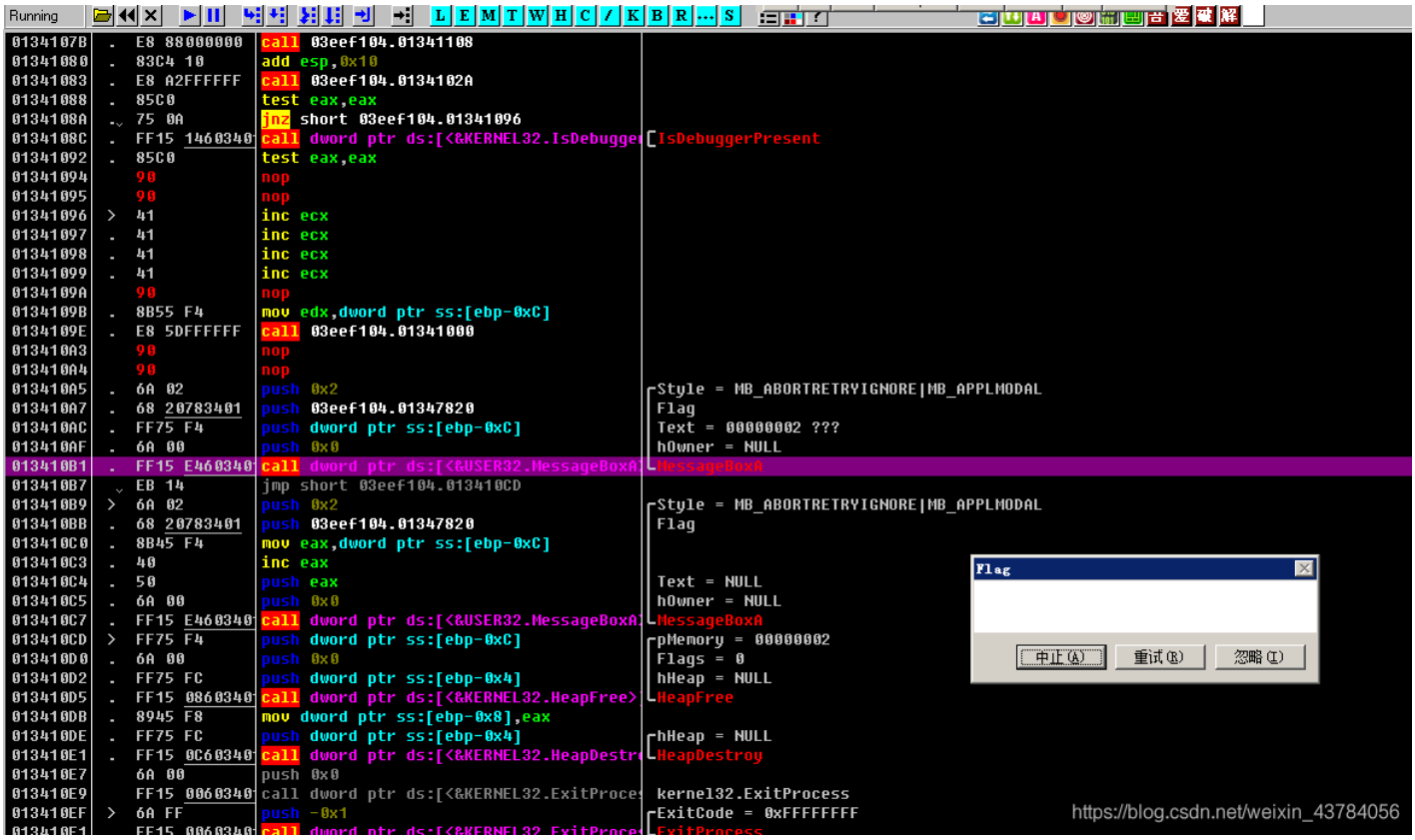
继续F8执行，执行到int 3，这是中断语句，所以也nop掉



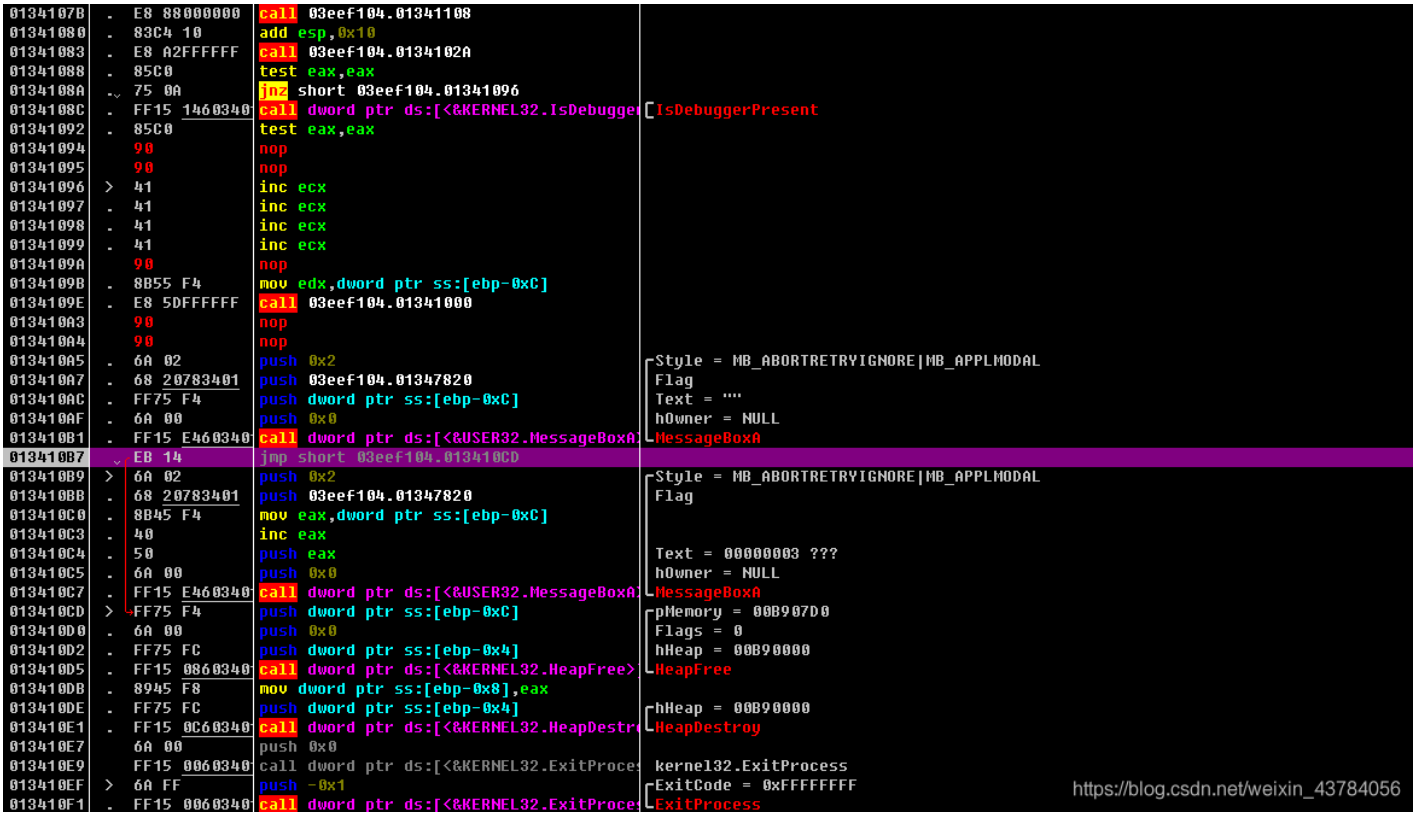
F8执行完生成flag的函数后，后面有一个大跳转，跳到退出程序的函数



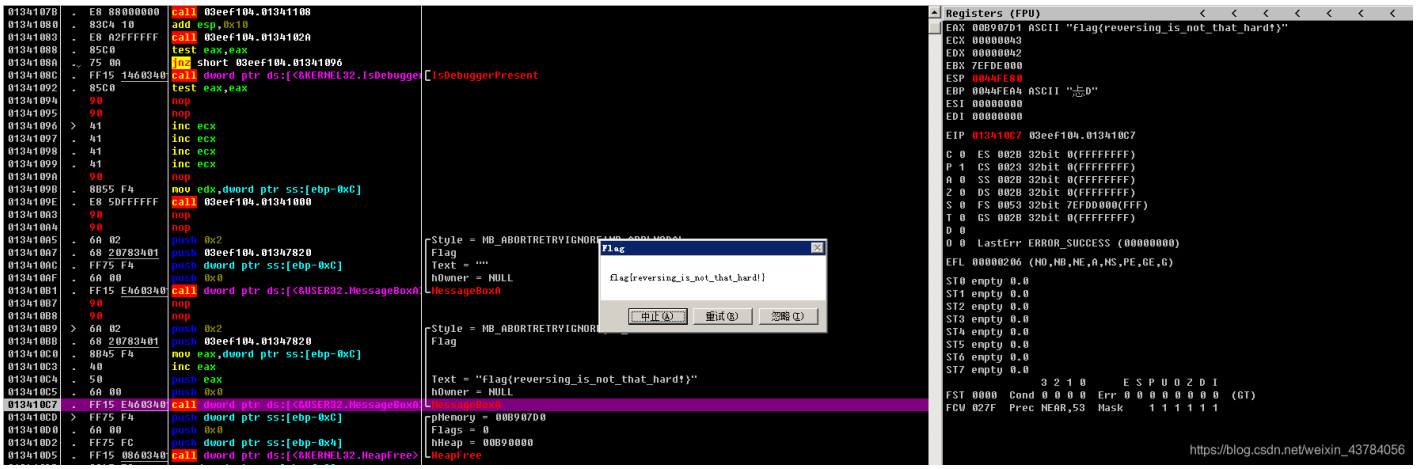
所以我们把这个跳转也给nop掉，继续F8，执行完一个MessageBoxA（弹框）函数后，发现程序此时处于Running状态，弹出一个什么也没有的框，其实这是另外一个弹框函数，真正输出flag的弹框函数是后面那个，在我们之前那个ida的修改之后的汇编图也可以发现，确实是有一个没有被调用的弹框函数，所以我们之前可以那个nop掉的跳转改为跳转到下面那个弹框函数，但既然说了是nop大法，就nop到底



点击中止之后，发现又要执行一个跳转，跳过了我们真正的弹框函数



将这个跳转nop掉，接着F8，就可以看到flag了



3、 分析代码写脚本

main函数代码

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  int v3; // ecx
  CHAR *lpMem; // [esp+8h] [ebp-Ch]
  HANDLE hHeap; // [esp+10h] [ebp-4h]

  hHeap = HeapCreate(0x40000u, 0, 0);
  lpMem = (CHAR *)HeapAlloc(hHeap, 8u, MaxCount + 1);
  memcpy_s(lpMem, MaxCount, &unk_409B10, MaxCount);
  if ( sub_40102A() || IsDebuggerPresent() )
  {
    __debugbreak();
    sub_401000(v3 + 4, lpMem);
    ExitProcess(0xFFFFFFFF);
  }
  MessageBoxA(0, lpMem + 1, "Flag", 2u);
  HeapFree(hHeap, 0, lpMem);
  HeapDestroy(hHeap);
  ExitProcess(0);
}
```

关键函数sub_401000的两个参数，v3后面没有用到，向上找lpMem的赋值语句，memcpy_s，将unk_409B10地址的值给了它，双击查看

```
.data:00409B0C                db    0
.data:00409B0D                db    0
.data:00409B0E                db    0
.data:00409B0F                db    0
.data:00409B10 unk_409B10     db 0BBh                 ; DATA XREF: _main+33↑o
.data:00409B11                db 0CCh
.data:00409B12                db 0A0h
.data:00409B13                db 0BCh
.data:00409B14                db 0DCh
.data:00409B15                db 0D1h
.data:00409B16                db 0BEh
.data:00409B17                db 0B8h
.data:00409B18                db 0CDh
.data:00409B19                db 0CFh
.data:00409B1A                db 0BEh
.data:00409B1B                db 0AEh
.data:00409B1C                db 0D2h
.data:00409B1D                db 0C4h
.data:00409B1E                db 0ABh
.data:00409B1F                db  82h
.data:00409B20                db 0D2h
.data:00409B21                db 0D9h
.data:00409B22                db  93h
.data:00409B23                db 0B3h
.data:00409B24                db 0D4h
.data:00409B25                db 0DEh
.data:00409B26                db  93h
.data:00409B27                db 0A9h
.data:00409B28                db 0D3h
.data:00409B29                db 0CBh
.data:00409B2A                db 0B8h
.data:00409B2B                db  82h
.data:00409B2C                db 0D3h
.data:00409B2D                db 0CBh
.data:00409B2E                db 0BEh
.data:00409B2F                db 0B9h
.data:00409B30                db  9Ah
.data:00409B31                db 0D7h
.data:00409B32                db 0CCh
.data:00409B33                db 0DDh
.data:00409B34 ; rsize_t MaxCount
.data:00409B34 MaxCount       dd 24h                  ; DATA XREF: _main+18↑r
```


进入sub_401000函数内部，代码

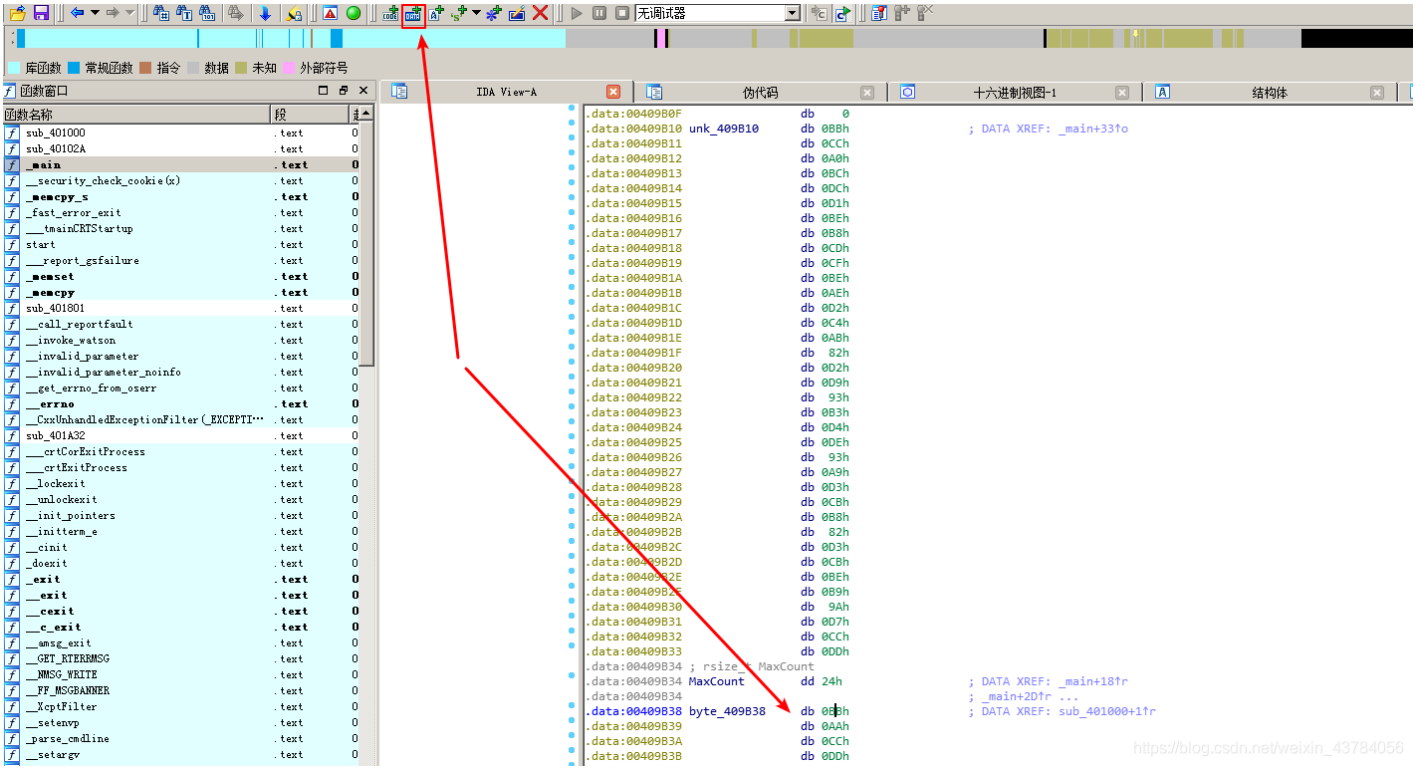
```
unsigned int __fastcall sub_401000(int a1, int a2)
{
  int v2; // esi
  unsigned int v3; // eax
  unsigned int v4; // ecx
  unsigned int result; // eax

  v2 = dword_409B38;
  v3 = a2 + 1 + strlen((const char *)(a2 + 1)) + 1;
  v4 = 0;
  result = ((v3 - (a2 + 2)) >> 2) + 1;
  if ( result )
  {
    do
      *(_DWORD *)(a2 + 4 * v4++) ^= v2;
    while ( v4 < result );
  }
  return result;
}
```

a2也就是lpMem，发现后面的异或语句有v2，向上找v2的赋值语句，找到v2 = dword_409B38，双击dword_409B38，找到内容

```
.data:00409B34 MaxCount         dd 24h                  ; DATA XREF: _main+18↑r
.data:00409B34                                          ; _main+2D↑r ...
.data:00409B38 dword_409B38     dd 0DDCCAABBh           ; DATA XREF: sub_401000+1↑r
.data:00409B3C                  align 10h
.data:00409B40 ; char *dword_409B40
.data:00409B40 dword_409B40     dd 0                    ; DATA XREF: ___tmainCRTStartup+B5↑w
.data:00409B40                                          ; __setenvp+F↑r ...
```

这里是四个字节显示的，又由于小端存储，所以顺序是颠倒的，我们可以将其转换成一个字节查看



然后根据源码写脚本，写的有点不太明白，记录一下

代码参照文章：https://www.cnblogs.com/DirWang/p/11420740.html

```
x=[0xbb,0xaa,0xcc,0xdd]
y=[0xBB,0xCC,0xA0,0xBC,0xDC,0xD1,0xBE,0xB8,0xCD,0xCF,0xBE,0xAE,0xD2,0xC4,0xAB,0x82,0xD2,0xD9,0x93,0xB3,0xD4
i=0
z=[]
while i<len(y):
    t=chr(y[i]^x[i%4])
    z.append(t)
    i+=1
print(z)
print(''.join(z))
```

['\x00', 'f', 'l', 'a', 'g', '{', 'r', 'e', 'v', 'e', 'r', 's', 'i', 'n', 'g', '_', 'i', 's', '_', 'n', 'o', 't', '_', 't', 'h', 'a', 't', '_', 'h', 'a', 'r', 'd', '!', '}', '\x00', '\x00']
flag{reversing_is_not_that_hard!}

这里就可以知道为什么调用第一个弹窗会输出空白，因为第一个弹窗函数，是直接从第一个字符输出的，但是第一个字符解码后为'\0',直接截断，所以会输出空白，第二个弹窗是从lpMem+1开始输出的