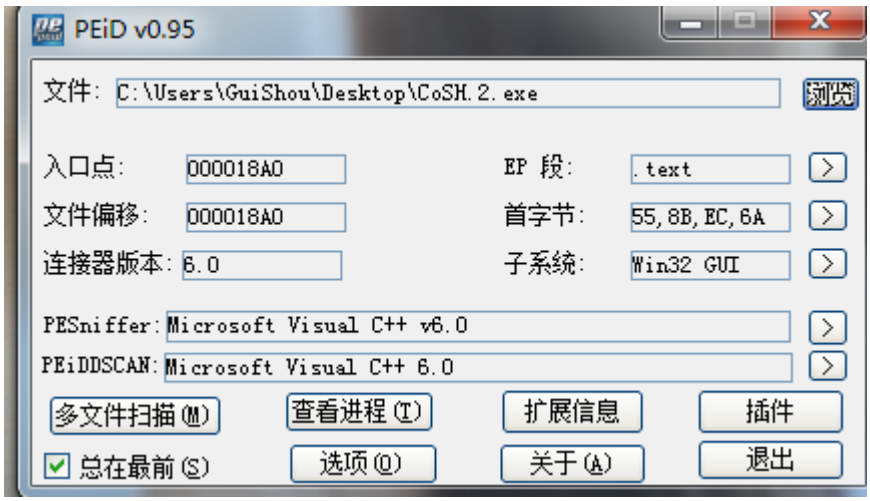
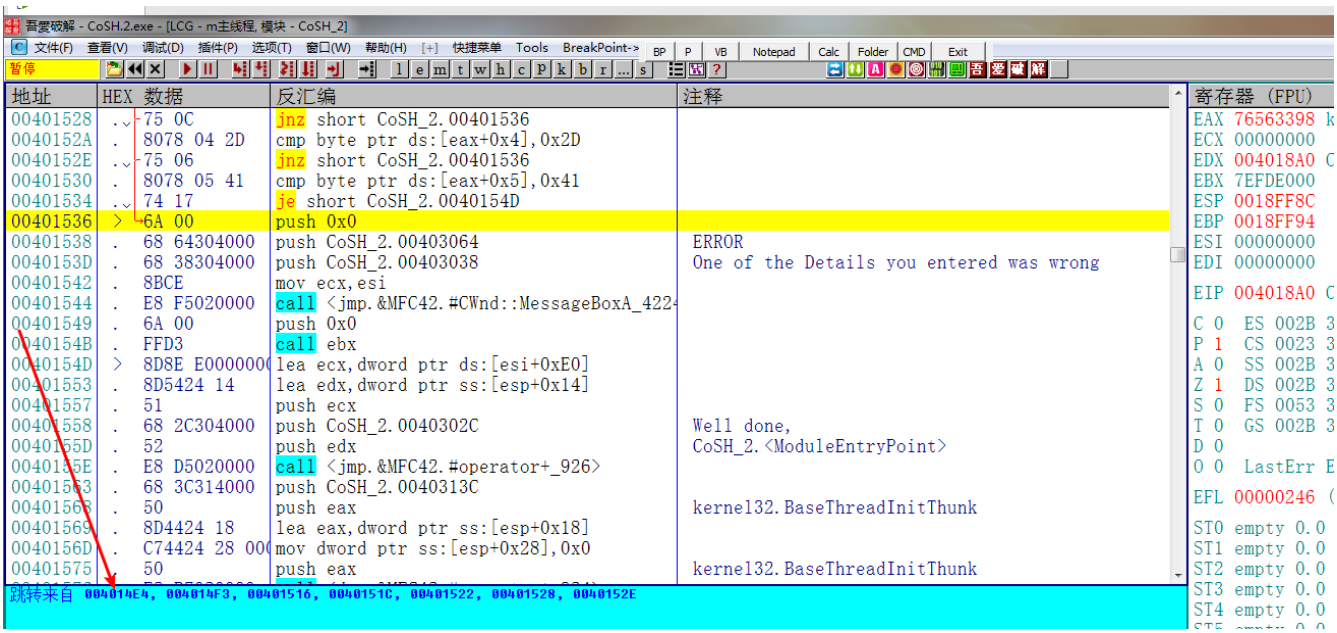
查壳
分析程序
分析算法
对抗花指令
校验过程
校验结果

查壳



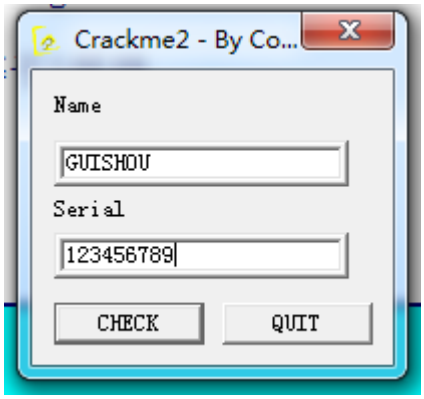
目标程序是用VC6写的，没有加壳，跟027是同一个作者，难度为一颗星

分析程序



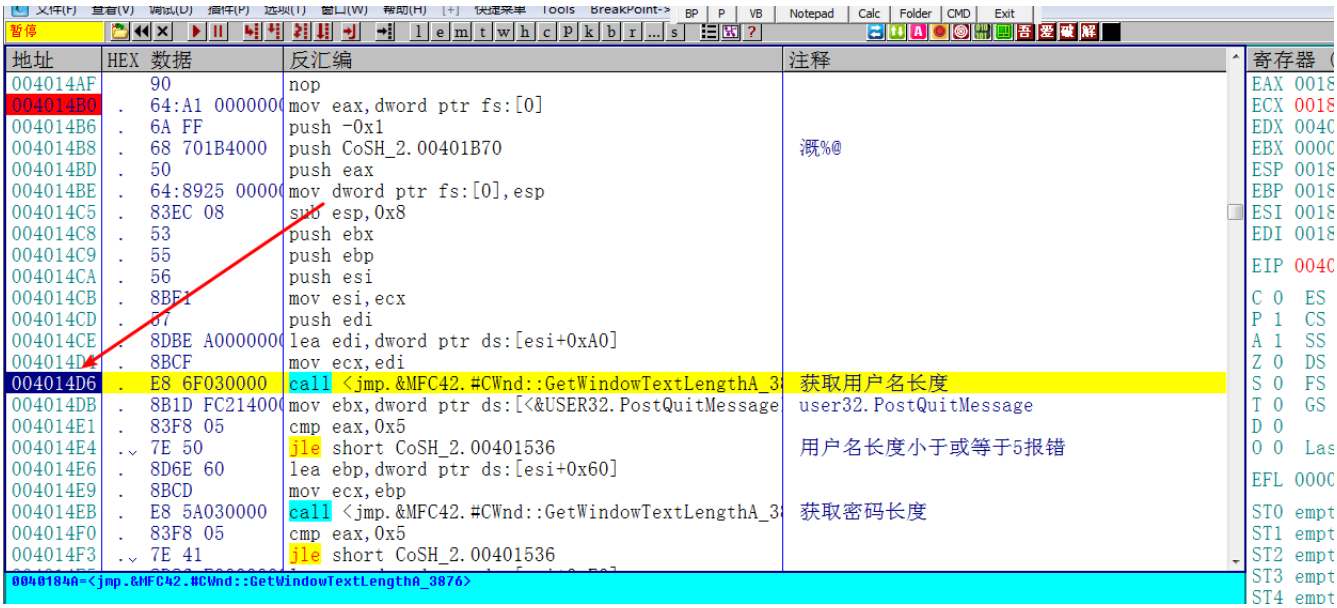
首先根据错误提示可以看到跳转到这个地址的地方有很多，我们直接选择最前面那个地址，开始分析整个算法

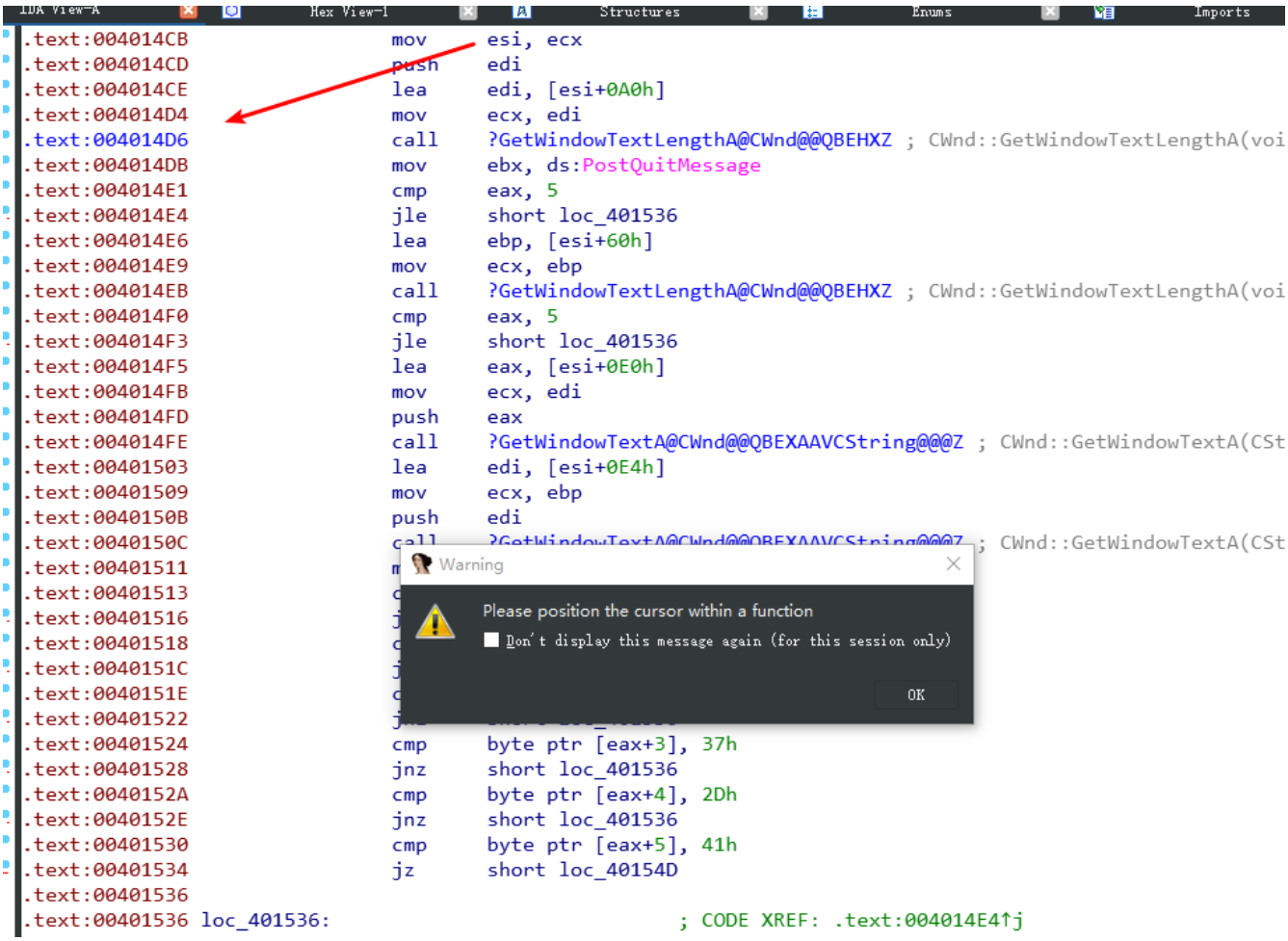
先随便输入一个序列号



另外，这个程序如果有IDA的帮助会分析的更快

分析算法



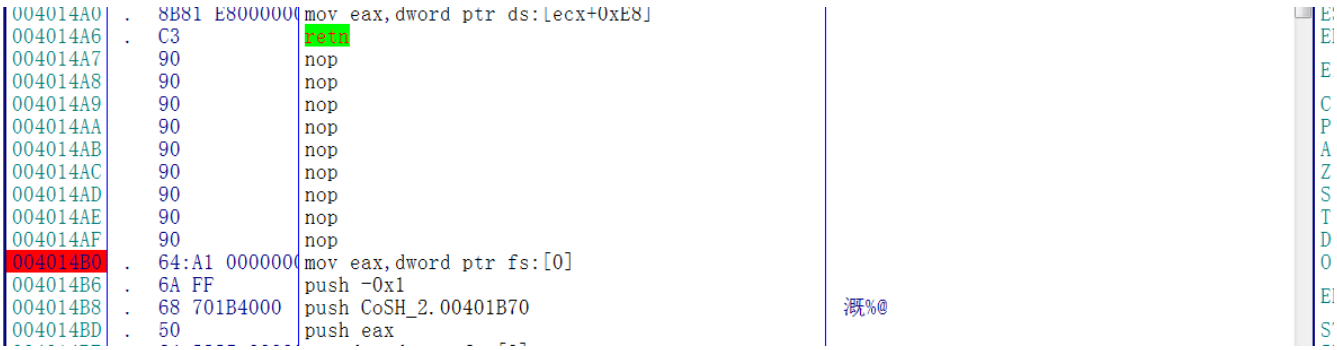


我们找到算法开始处的地址，对应到IDA中，F5查看一下伪C代码，但是你会发现有个提示，让我们把光标选择在一个函数，这表示IDA的反编译失败了!这是为什么呢?

对抗花指令

上一个Crackme作者犯了一个低级错误，写错了一个参数，这一次他玩了一个小把戏，用花指令干扰我们静态分析。现在来破解一个小把戏

OD直接拉到函数开头的部分，你会看到有一堆nop指令



在IDA同样地址处这里却不是nop，而是被解释成了align10，IDA的反汇编器在这里出现了错误，把数据解释成了代码

```
.text:004014A6 ; ---------------------------------------------------------------------------
.text:004014A7                 align 10h
.text:004014B0                 mov     eax, large fs:0
.text:004014B6                 push    0FFFFFFFFh
.text:004014B8                 push    offset loc_401B70
.text:004014BD                 push    eax
.text:004014BE                 mov     large fs:0, esp
.text:004014C5                 sub     esp, 8
.text:004014C8                 push    ebx
.text:004014C9                 push    ebp
.text:004014CA                 push    esi
.text:004014CB                 mov     esi, ecx
.text:004014CD                 push    edi
.text:004014CE                 lea     edi, [esi+0A0h]
.text:004014D4                 mov     ecx, edi
.text:004014D6                 call    ?GetWindowTextLengthA@CWnd@@QBEHXZ ; CWnd::GetWindowTextLengthA(void)
.text:004014DB                 mov     ebx, ds:PostQuitMessage
.text:004014E1                 cmp     eax, 5
.text:004014E4                 jle     short loc_401536
.text:004014E6                 lea     ebp, [esi+60h]
.text:004014E9                 mov     ecx, ebp
.text:004014EB                 call    ?GetWindowTextLengthA@CWnd@@QBEHXZ ; CWnd::GetWindowTextLengthA(void)
.text:004014F0                 cmp     eax, 5
.text:004014F3                 jle     short loc_401536
.text:004014F5                 lea     eax, [esi+0E0h]
.text:004014FB                 mov     ecx, edi
```

那么解决办法就是在这一行按字母D键，将代码强行解释为数据，修改后如下：

```
.text:004014A6                 retn
.text:004014A6 ; ---------------------------------------------------------------------------
.text:004014A7                 db 90h
.text:004014A8                 db  90h
.text:004014A9                 db  90h
.text:004014AA                 db  90h
.text:004014AB                 db  90h
.text:004014AC                 db  90h
.text:004014AD                 db  90h
.text:004014AE                 db  90h
.text:004014AF                 db  90h
.text:004014B0 ; ---------------------------------------------------------------------------
.text:004014B0                 mov     eax, large fs:0
.text:004014B6                 push    0FFFFFFFFh
.text:004014B8                 push    offset loc_401B70
.text:004014BD                 push    eax
.text:004014BE                 mov     large fs:0, esp
.text:004014C5                 sub     esp, 8
.text:004014C8                 push    ebx
.text:004014C9                 push    ebp
.text:004014CA                 push    esi
.text:004014CB                 mov     esi, ecx
.text:004014CD                 push    edi
.text:004014CE                 lea     edi, [esi+0A0h]
.text:004014D4                 mov     ecx, edi
```

然后选中整个函数，按P键组合成一个函数

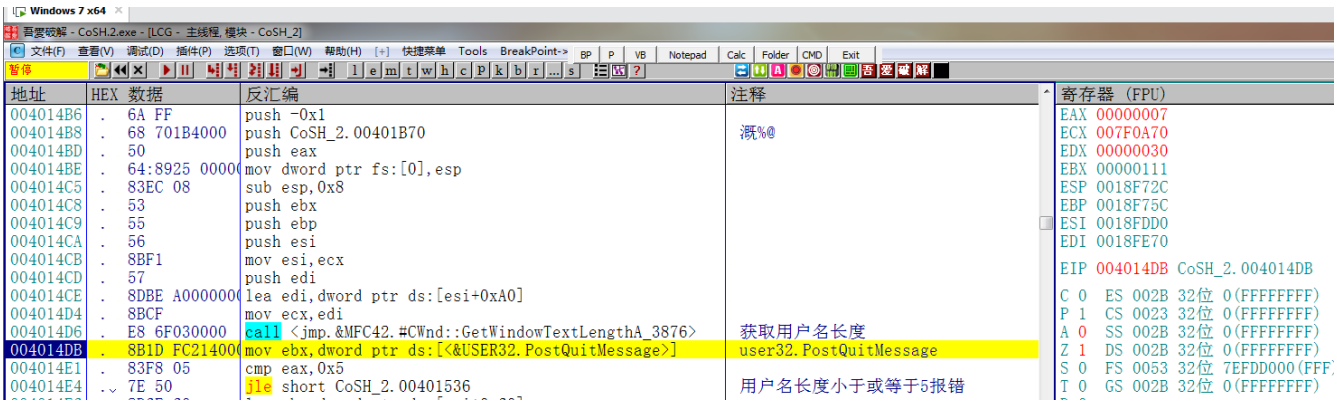
```
1void __thiscall sub_4014B0(CWnd *this)
2{
3  CWnd *v1; // esi
4  CWnd *v2; // edi
5  _BYTE *v3; // eax
6  int v4; // eax
7  const char *v5; // ST00_4
8  char v6; // [esp+10h] [ebp-14h]
9  char v7; // [esp+14h] [ebp-10h]
10  int v8; // [esp+20h] [ebp-4h]
11
12  v1 = this;
13  v2 = (CWnd *)((char *)this + 160);
14  if ( CWnd::GetWindowTextLengthA((CWnd *)((char *)this + 160)) <= 5// 用户名长度必须大于5
15    || CWnd::GetWindowTextLengthA((CWnd *)((char *)v1 + 96)) <= 5// 序列号长度必须大于5
16    || (CWnd::GetWindowTextA(
17          v2,
18          (CWnd *)((char *)v1 + 224)),          // 获取用户名
19        CWnd::GetWindowTextA(
20          (CWnd *)((char *)v1 + 96),
21          (CWnd *)((char *)v1 + 228)),          // 获取序列号
22        v3 = (_BYTE *)*((_DWORD *)v1 + 57),
23        **((_BYTE **)v1 + 57) != 54)
24    || v3[1] != 50
25    || v3[2] != 56
26    || v3[3] != 55
27    || v3[4] != 45
28    || v3[5] != 65 )
29  {
30    CWnd::MessageBoxA(v1, aOneOfTheDetail, aError, 0);
31    PostQuitMessage(0);
32  }
33  v4 = operator+(&v7, aWellDone, (char *)v1 + 224);
34  v8 = 0;
35  v5 = *(const char **)operator+(&v6, v4, &unk_40313C);
36  LOBYTE(v8) = 1;
37  CWnd::MessageBoxA(v1, v5, aYouDidIt, 0);
38  LOBYTE(v8) = 0;
39  CString::~CString((CString *)&v6);
40  v8 = -1;
41  CString::~CString((CString *)&v7);
42  PostQuitMessage(1);
43}
```

这个时候你再按F5，就能显示整个函数的伪代码了

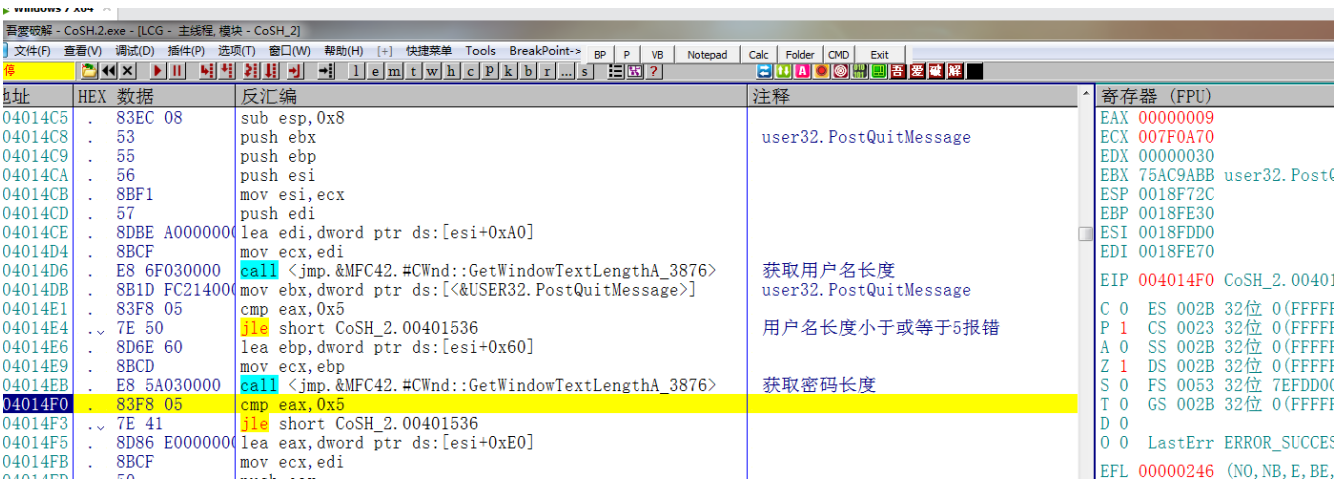
校验过程

这个程序的校验过程相对来说比较简单，过程如下

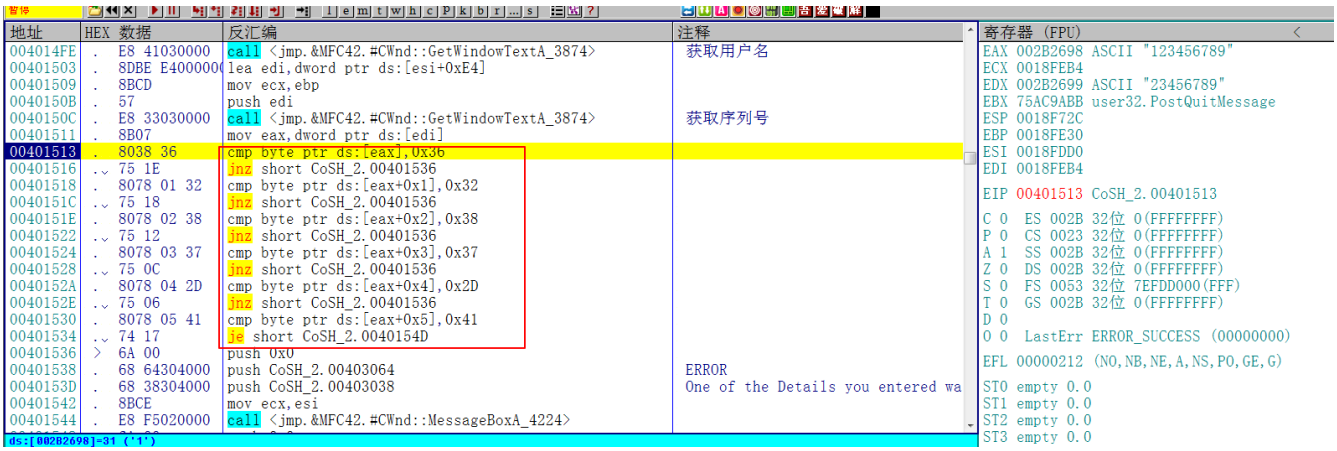
1. 获取用户名长度，比较是否小于等于5



2. 获取密码长度 比较是否小于等于5



3. 获取序列号 开始校验序列号



可以看到校验的部分就是6个cmp和jcc指令了，直接看IDA

```
10  int v8; // [esp+20h] [ebp-4h]
11
12  v1 = this;
13  v2 = (CWnd *)((char *)this + 160);
14  if ( CWnd::GetWindowTextLengthA((CWnd *)((char *)this + 160)) <= 5// 用户名长度必须大于5
15    || CWnd::GetWindowTextLengthA((CWnd *)((char *)v1 + 96)) <= 5// 序列号长度必须大于5
16    || (CWnd::GetWindowTextA(
17          v2,
18          (CWnd *)((char *)v1 + 224)),          // 获取用户名
19        CWnd::GetWindowTextA(
20          (CWnd *)((char *)v1 + 96),
21          (CWnd *)((char *)v1 + 228)),          // 获取序列号
22        v3 = (_BYTE *)*((_DWORD *)v1 + 57),
23        **((_BYTE **)v1 + 57) != 54)
24    || v3[1] != 50
25    || v3[2] != 56
26    || v3[3] != 55
27    || v3[4] != 45
28    || v3[5] != 65 )
29  {
30    CWnd::MessageBoxA(v1, aOneOfTheDetail, aError, 0);
31    PostQuitMessage(0);
32  }
```

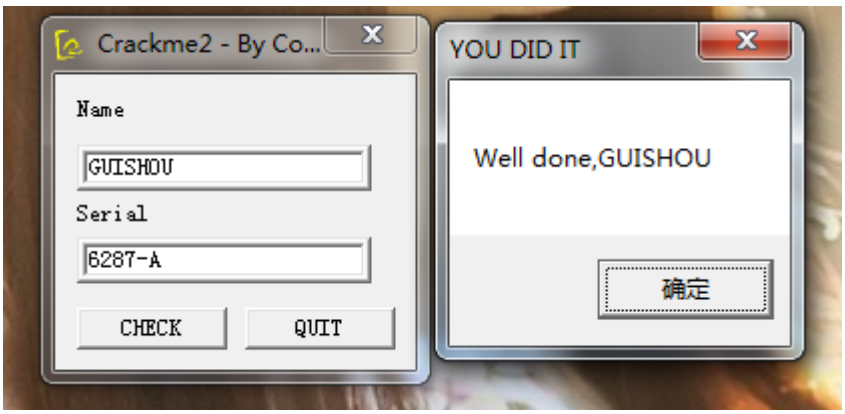
这里显示的是一堆十进制的数字，这个对我们来说没有意义，按R键，可以把数字转为字符串，转换后如下

```
 9  char v7; // [esp+14h] [ebp-10h]
10  int v8; // [esp+20h] [ebp-4h]
11
12  v1 = this;
13  v2 = (CWnd *)((char *)this + 160);
14  if ( CWnd::GetWindowTextLengthA((CWnd *)((char *)this + 160)) <= 5// 用户名长度必须大于5
15    || CWnd::GetWindowTextLengthA((CWnd *)((char *)v1 + 96)) <= 5// 序列号长度必须大于5
16    || (CWnd::GetWindowTextA(
17          v2,
18          (CWnd *)((char *)v1 + 224)),          // 获取用户名
19        CWnd::GetWindowTextA(
20          (CWnd *)((char *)v1 + 96),
21          (CWnd *)((char *)v1 + 228)),          // 获取序列号
22        v3 = (_BYTE *)*((_DWORD *)v1 + 57),
23        **((_BYTE **)v1 + 57) != '6')
24    || v3[1] != '2'
25    || v3[2] != '8'
26    || v3[3] != '7'
27    || v3[4] != '-'
28    || v3[5] != 'A' )
29  {
30    CWnd::MessageBoxA(v1, aOneOfTheDetail, aError, 0);
31    PostQuitMessage(0);
32  }
33  v4 = operator+(&v7, aWellDone, (char *)v1 + 224);
```

可以很直观的看到序列号就是6287-A

校验结果

随便输入一个用户名，但是长度必须大于5，然后输入刚才的序号，提示正确 破解完成



最后，需要相关文件可以到我的Github下载:

https://github.com/TonyChen56/160-Crackme