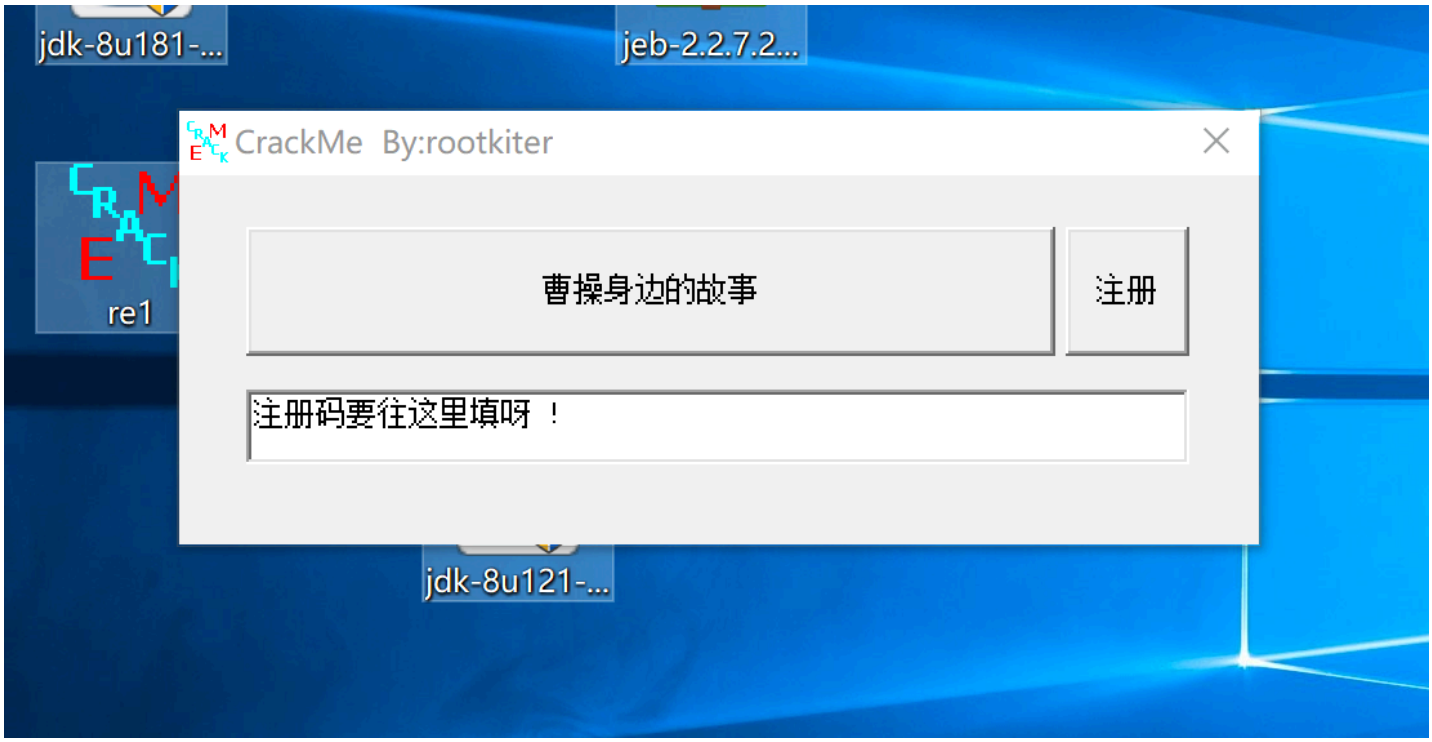
re1

首先 这是一个 32位 pe可执行文件 用mfc写的

我们运行一下：



现在我们要输入一个注册码 我们要推测出正确的注册码。

因为是mfc，招 main的意义并不大 ， 先找 messsageboxa 。

不断通过查找message的调用函数 在回溯到关键函数，查找调用函数的 快捷键是 `x` 。

找到一个地方

```
.text:004015D1 ; ---------------------------------------------------------------------------
.text:004015D2                 align 10h
.text:004015E0                 push    ecx
.text:004015E1                 push    esi
.text:004015E2                 mov     esi, ecx
.text:004015E4                 push    1
.text:004015E6                 call    ?UpdateData@CWnd@@QAEHH@Z ; CWnd::UpdateData(int)
.text:004015EB                 mov     ecx, [esi+294h]
.text:004015F1                 lea     eax, [esi+294h]
.text:004015F7                 cmp     dword ptr [ecx-8], 21h
.text:004015FB                 jnz     short loc_40161F
.text:004015FD                 push    ecx
.text:004015FE                 mov     ecx, esp
.text:00401600                 mov     [esp+8], esp
.text:00401604                 push    eax
.text:00401605                 call    ??0CString@@QAE@ABV0@@Z ; CString::CString(CString const &)
```

p

不是一个函数 但是结构想一个函数， 我们试着在 push的地方 按 `p` 然后f5试一下 。发现可以反编译

```
1  int __thiscall sub_4015E0(CWnd *this)
2  {
3    CWnd *v1; // esi
4    int result; // eax
5    int v3; // [esp-4h] [ebp-Ch]
6    int *v4; // [esp+4h] [ebp-4h]
7
8    v1 = this;
9    CWnd::UpdateData(this, 1);
10   if ( *(_DWORD *)(*((_DWORD *)v1 + 165) - 8) == 33
11     && (v3 = *((_DWORD *)v1 + 165),
12         v4 = &v3,
13         CString::CString((CString *)&v3, (CWnd *)((char *)v1 + 660)),
14         (unsigned __int8)sub_401630(v3)) )
15   {
16     result = sub_4016E0(v1);
17   }
18   else
19   {
20     result = sub_401720(v1);
```

000015E0 sub_4015E0:19 (4015E0)

我们现在分析了一下 if语句：

```
if ( *(_DWORD *)(*((_DWORD *)v1 + 165) - 8) == 33
    && (v3 = *((_DWORD *)v1 + 165),
        v4 = &v3,
        CString::CString((CString *)&v3, (CWnd *)((char *)v1 + 660)),
        (unsigned __int8)sub_401630(v3)) )
```

可以看到 有一个 ==33 看不懂这是什么。

发现循环要进行33次 我们猜测flag的长度会不会就是33

我们动态调试一下，这样更快速并且直观：

```
.text:004015E4                 push    1               ; int
.text:004015E6                 call    ?UpdateData@CWnd@@QAEHH@Z ; CWnd::UpdateData(
.text:004015EB ; 9:   if ( *(_DWORD *)(*((_DWORD *)v1 + 165) - 8) == 33
.text:004015EB ; 10:     && (v3 = *((_DWORD *)v1 + 165),
.text:004015EB ; 11:         v4 = &v3,
.text:004015EB ; 12:         CString::CString((CString *)&v3, (CWnd *)((char *)v1 + 
.text:004015EB ; 13:         sub_401630(v1, v3)) )
.text:004015EB                 mov     ecx, [esi+294h]
.text:004015F1                 lea     eax, [esi+294h]
.text:004015F7                 cmp     dword ptr [ecx-8], 21h
.text:004015FB                 jnz     short loc_40161F
.text:004015FD                 push    ecx
.text:004015FE                 mov     ecx, esp        ; this
.text:00401600                 mov     [esp+0Ch+var_4], esp
.text:00401604                 push    eax             ; struct CString *
.text:00401605                 call    ??0CString@@QAE@ABV0@@Z ; CString::CString(CS
.text:0040160A                 mov     ecx, esi
.text:0040160C                 call    sub_401630
.text:00401611                 test    al, al

000015F7 004015F7: sub_4015E0+17 (Synchronized with Hex View-1)
```

如上图所示 下一个断点：

debugger一下：

在弹出的输入框里 输入12345

查看[ecx-8] 刚好就是 5，21h 就是33，可以 推测 输入的字符串长度 就是33.

2

现在先看一下 `sub_401630`

```
  v5 = 0;
  do
  {
    srand(v4);
    v4 = rand() % 10;
    if ( *(_BYTE *)(v2 + a2) != v3[v5 + 96 + v4] )
      v7 = 0;
    v5 += 10;
    ++v2;
  }
  while ( v5 < 330 );
  CString::~CString((CString *)&a2);
  return v7;
```

里面虽然有一个随机 但是 种子是固定的 所以v4的值 其实也是固定的。

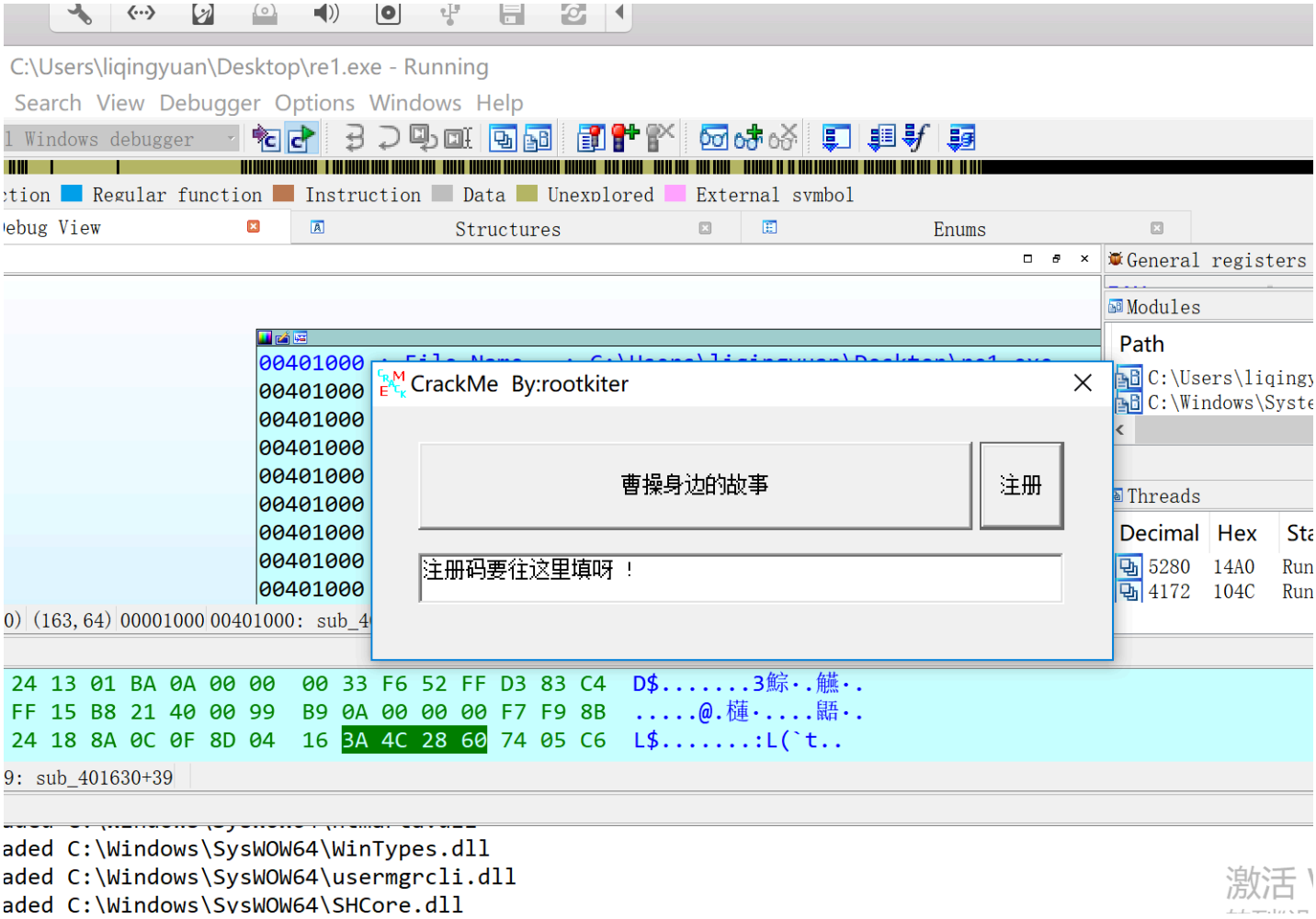
可以看到 是将a2中的每一个字符与v3中的字符进行比较。我们可以 debug一下，

鼠标 放在 if语句同一行，然后 右键 选中 `copy to assenbly` 可以看到对应的汇编代码

```
.text:0040165F ; 17:     if ( *(_BYTE *)(v2 + a2) != v3[v5 + 96 + v4] )
.text:0040165F                 mov     ecx, [esp+14h+arg_0]
.text:00401663                 mov     cl, [edi+ecx]
.text:00401666                 lea     eax, [esi+edx]
.text:00401669                 cmp     cl, [eax+ebp+60h]
.text:0040166D                 jz      short loc_401674
.text:0040166F ; 18:       v7 = 0;
```

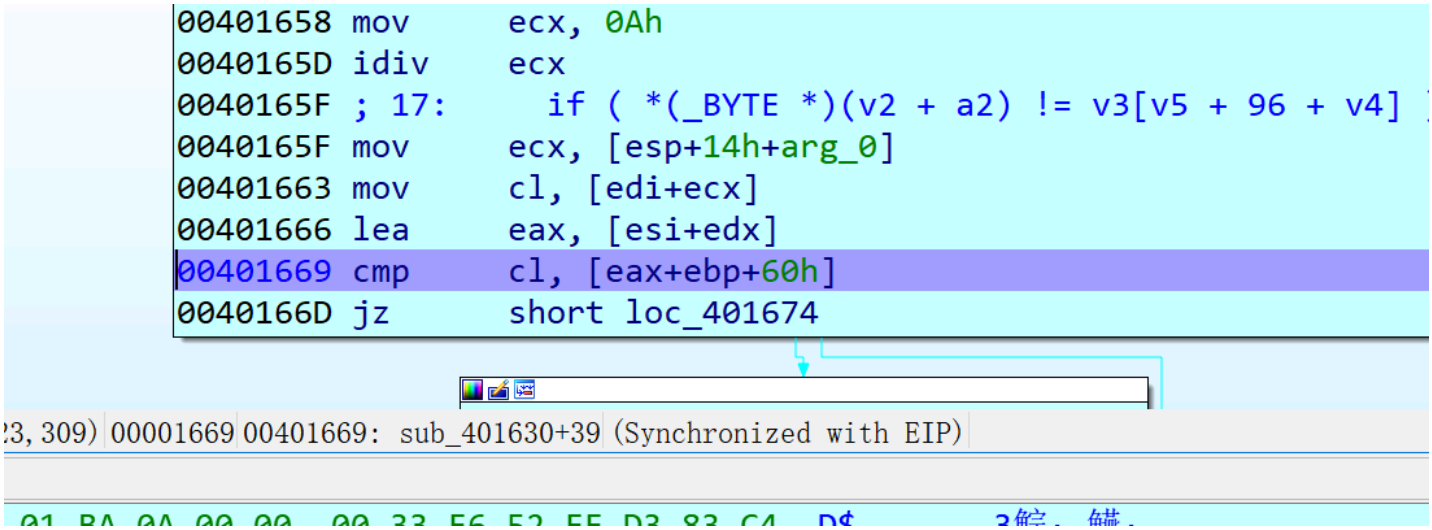
我们在 cmp的地方下一个断点：

运行：



输入33个长度的字符串

运行之后 运行至我们下的断点处 查看 这个cmp中操作数 的数据：



f9 运行 每次 运行到断点处 就停下来，循环4次可以看到 flag ， 所以这应该就是flag了。

我们运行 33次 就可以得到flag了。

这道题 还可以通过patch 的方式 来得到flag。在这里记一下，就是把下断点的 汇编cmp 改成别的 。弄懂的话再更新。