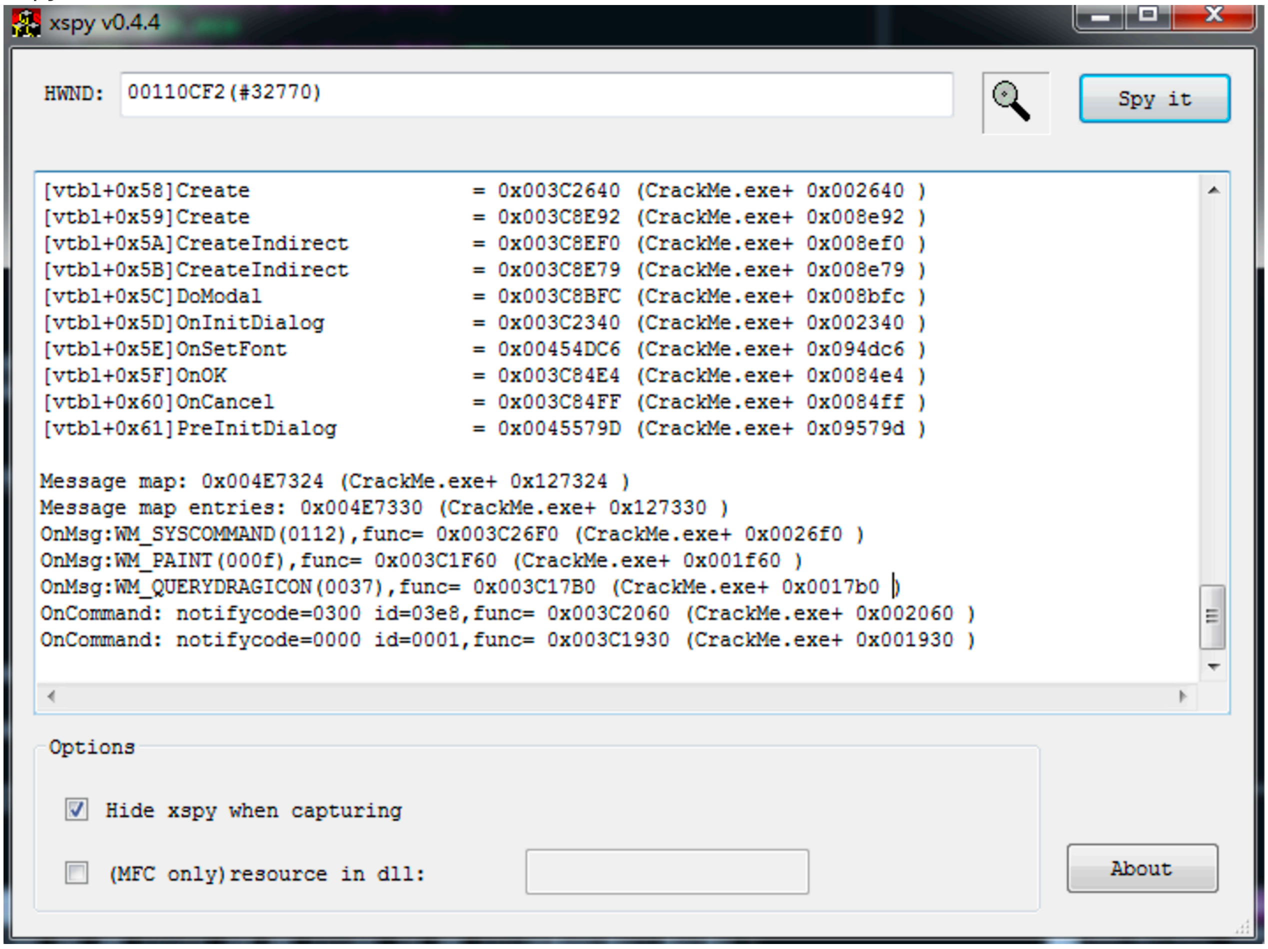
逆向作业

//这cm4有点东西

cm4

xspy看到关于灰色按钮的两个函数地址：+0x2060和+0x1930



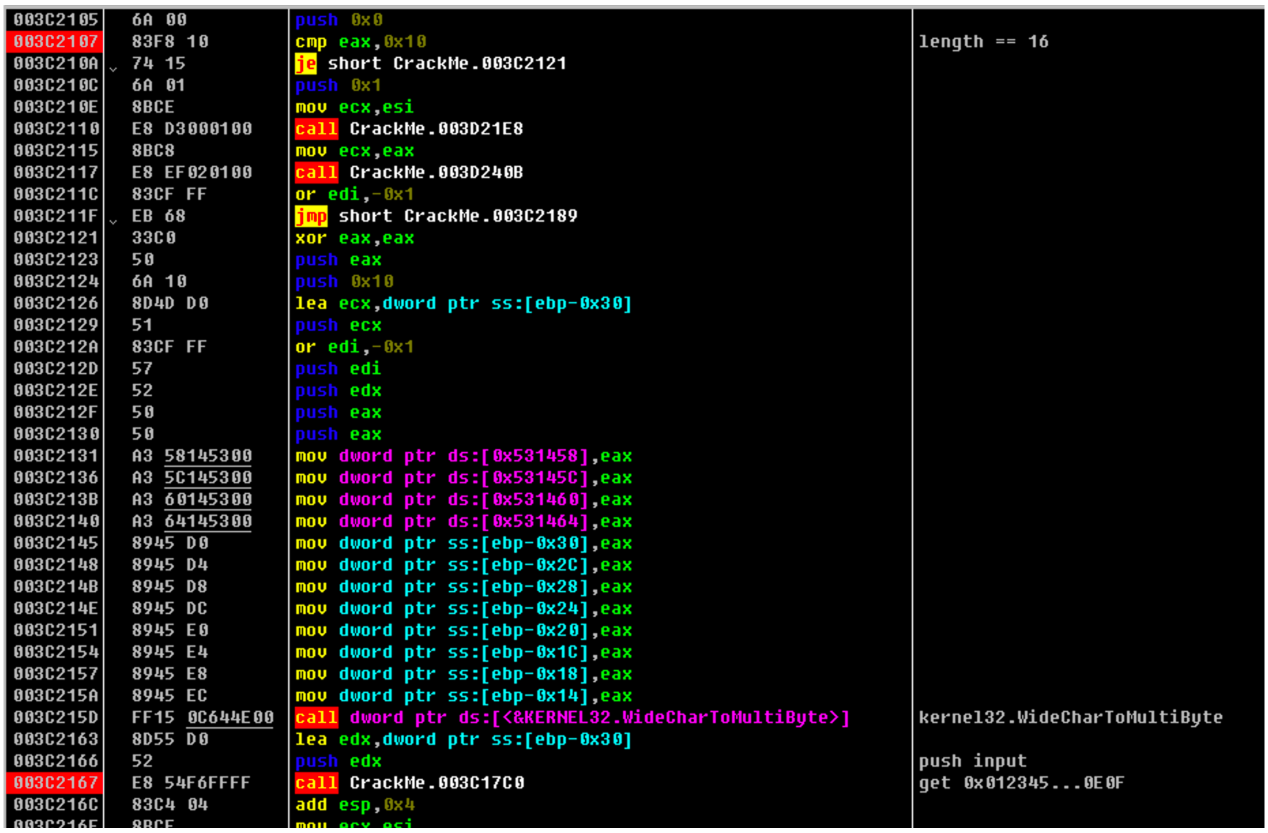
HWND: 00110CF2(#32770)

```
[vtbl+0x58]Create                 = 0x003C2640 (CrackMe.exe+ 0x002640 )
[vtbl+0x59]Create                 = 0x003C8E92 (CrackMe.exe+ 0x008e92 )
[vtbl+0x5A]CreateIndirect         = 0x003C8EF0 (CrackMe.exe+ 0x008ef0 )
[vtbl+0x5B]CreateIndirect         = 0x003C8E79 (CrackMe.exe+ 0x008e79 )
[vtbl+0x5C]DoModal                = 0x003C8BFC (CrackMe.exe+ 0x008bfc )
[vtbl+0x5D]OnInitDialog           = 0x003C2340 (CrackMe.exe+ 0x002340 )
[vtbl+0x5E]OnSetFont              = 0x00454DC6 (CrackMe.exe+ 0x094dc6 )
[vtbl+0x5F]OnOK                   = 0x003C84E4 (CrackMe.exe+ 0x0084e4 )
[vtbl+0x60]OnCancel               = 0x003C84FF (CrackMe.exe+ 0x0084ff )
[vtbl+0x61]PreInitDialog          = 0x0045579D (CrackMe.exe+ 0x09579d )

Message map: 0x004E7324 (CrackMe.exe+ 0x127324 )
Message map entries: 0x004E7330 (CrackMe.exe+ 0x127330 )
OnMsg:WM_SYSCOMMAND(0112),func= 0x003C26F0 (CrackMe.exe+ 0x0026f0 )
OnMsg:WM_PAINT(000f),func= 0x003C1F60 (CrackMe.exe+ 0x001f60 )
OnMsg:WM_QUERYDRAGICON(0037),func= 0x003C17B0 (CrackMe.exe+ 0x0017b0 )
OnCommand: notifycode=0300 id=03e8,func= 0x003C2060 (CrackMe.exe+ 0x002060 )
OnCommand: notifycode=0000 id=0001,func= 0x003C1930 (CrackMe.exe+ 0x001930 )
```

[✓] Hide xspy when capturing

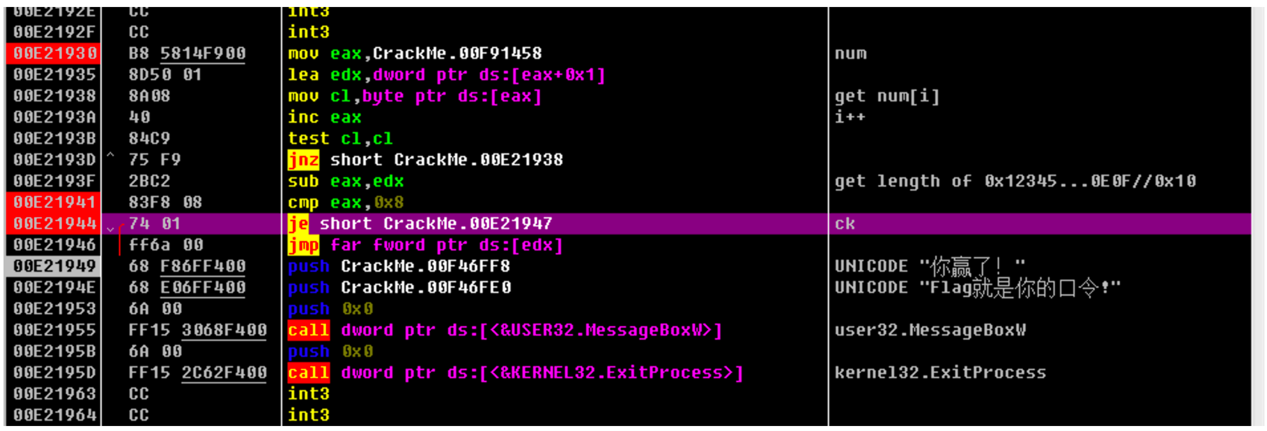
[ ] (MFC only)resource in dll:

先调2060的函数 发现是用来检查输入是否长16，并且转换成十六进制的，检查成功则按钮可按



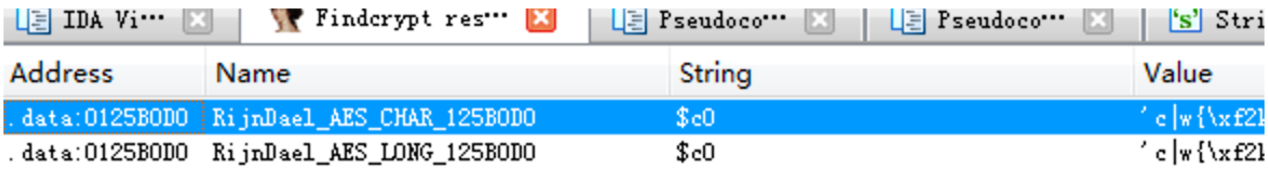
```
003C2105   6A 00          push 0x0
003C2107   83F8 10        cmp eax,0x10                                              length == 16
003C210A   74 15          je short CrackMe.003C2121
003C210C   6A 01          push 0x1
003C210E   8BCE           mov ecx,esi
003C2110   E8 D3000100    call CrackMe.003D21E8
003C2115   8BC8           mov ecx,eax
003C2117   E8 EF020100    call CrackMe.003D240B
003C211C   83CF FF        or edi,-0x1
003C211F   EB 68          jmp short CrackMe.003C2189
003C2121   33C0           xor eax,eax
003C2123   50             push eax
003C2124   6A 10          push 0x10
003C2126   8D4D D0        lea ecx,dword ptr ss:[ebp-0x30]
003C2129   51             push ecx
003C212A   83CF FF        or edi,-0x1
003C212D   57             push edi
003C212E   52             push edx
003C212F   50             push eax
003C2130   50             push eax
003C2131   A3 58145300    mov dword ptr ds:[0x531458],eax
003C2136   A3 5C145300    mov dword ptr ds:[0x53145C],eax
003C213B   A3 60145300    mov dword ptr ds:[0x531460],eax
003C2140   A3 64145300    mov dword ptr ds:[0x531464],eax
003C2145   8945 D0        mov dword ptr ss:[ebp-0x30],eax
003C2148   8945 D4        mov dword ptr ss:[ebp-0x2C],eax
003C214B   8945 D8        mov dword ptr ss:[ebp-0x28],eax
003C214E   8945 DC        mov dword ptr ss:[ebp-0x24],eax
003C2151   8945 E0        mov dword ptr ss:[ebp-0x20],eax
003C2154   8945 E4        mov dword ptr ss:[ebp-0x1C],eax
003C2157   8945 E8        mov dword ptr ss:[ebp-0x18],eax
003C215A   8945 EC        mov dword ptr ss:[ebp-0x14],eax
003C215D   FF15 0C644E00  call dword ptr ds:[<&KERNEL32.WideCharToMultiByte>]       kernel32.WideCharToMultiByte
003C2163   8D55 D0        lea edx,dword ptr ss:[ebp-0x30]
003C2166   52             push edx                                                  push input
003C2167   E8 54F6FFFF    call CrackMe.003C17C0                                     get 0x012345...0E0F
003C216C   83C4 04        add esp,0x4
003C216E   8BCE           mov ecx,esi
```

点击确定后调用+0x1930处的函数。//这里看到有成功的提示，若想je跳转必须让eax==8，也就是输入的第9位是0，但是2060处的函数有个eax!=8才能通过的判断，所以不管怎么输入都不可能在这里满足跳转



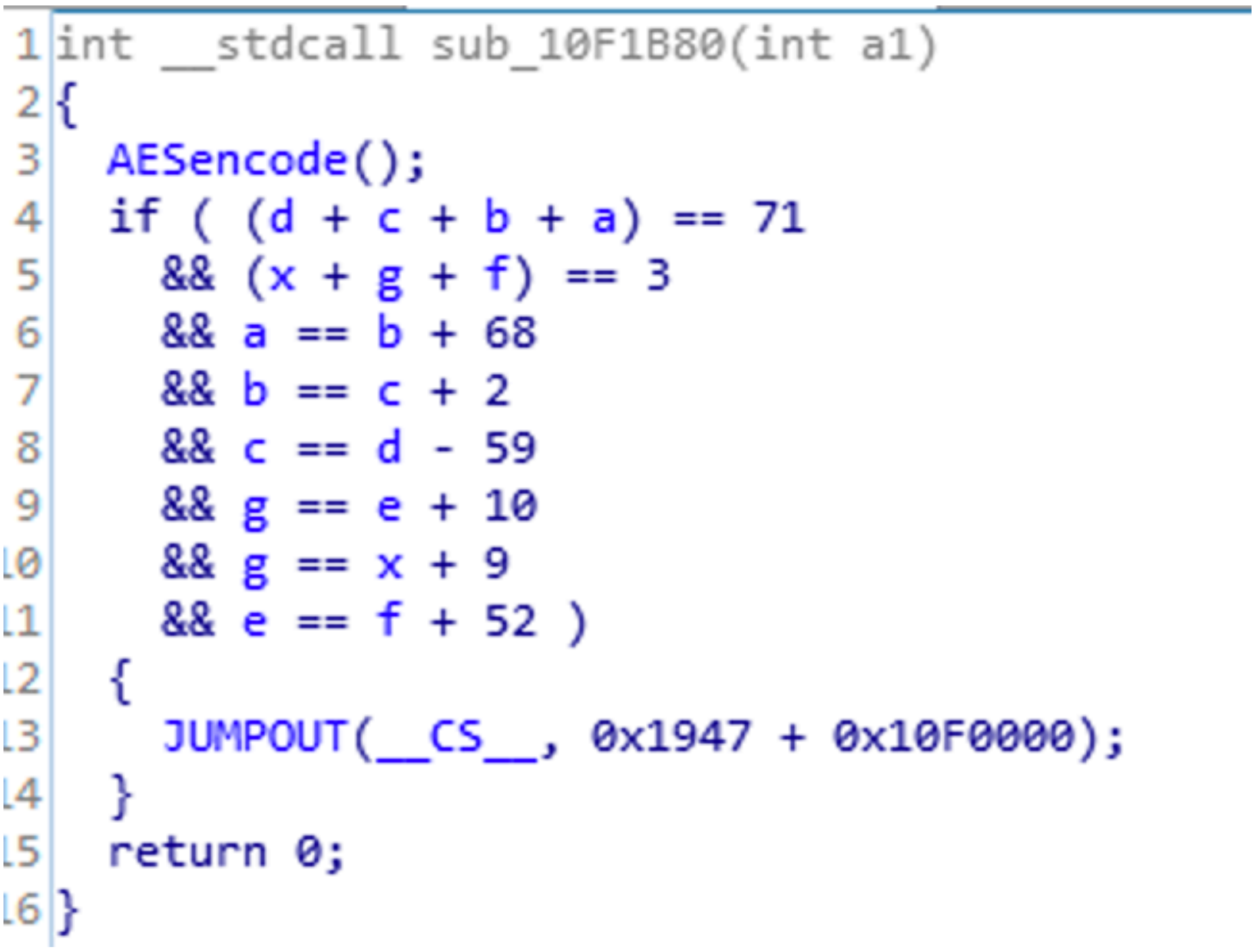
```
00E2192E   CC             int3
00E2192F   CC             int3
00E21930   B8 5814F900    mov eax,CrackMe.00F91458                   num
00E21935   8D50 01        lea edx,dword ptr ds:[eax+0x1]
00E21938   8A08           mov cl,byte ptr ds:[eax]                   get num[i]
00E2193A   40             inc eax                                    i++
00E2193B   84C9           test cl,cl
00E2193D   75 F9          jnz short CrackMe.00E21938
00E2193F   2BC2           sub eax,edx                                get length of 0x12345...0E0F//0x10
00E21941   83F8 08        cmp eax,0x8
00E21944   74 01          je short CrackMe.00E21947                  ck
00E21946   ff6a 00        jmp far fword ptr ds:[edx]
00E21949   68 F86FF400    push CrackMe.00F46FF8                      UNICODE "你赢了！"
00E2194E   68 E06FF400    push CrackMe.00F46FE0                      UNICODE "Flag就是你的口令!"
00E21953   6A 00          push 0x0
00E21955   FF15 3068F400  call dword ptr ds:[<&USER32.MessageBoxW>]  user32.MessageBoxW
00E2195B   6A 00          push 0x0
00E2195D   FF15 2C62F400  call dword ptr ds:[<&KERNEL32.ExitProcess>] kernel32.ExitProcess
00E21963   CC             int3
00E21964   CC             int3
```

丢ida里搜到有AES，可以交叉引用到+0x1970处的函数。发现这个函数只是对几个变量调用了AES的S盒



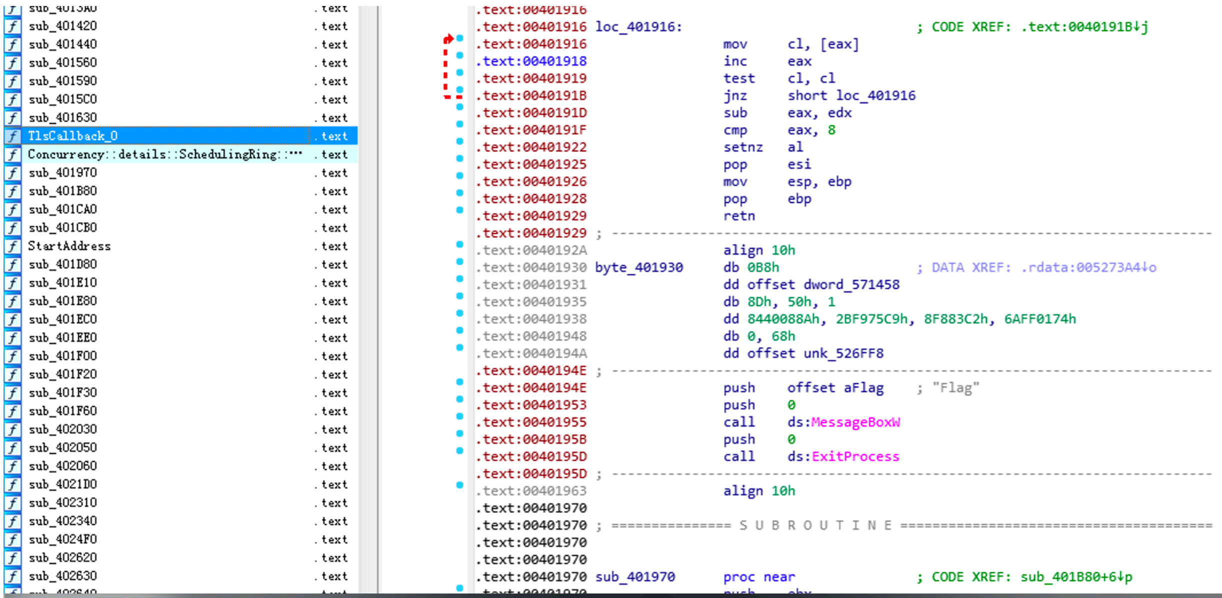
```
Address          Name                       String    Value
.data:0125B0D0   RijnDael_AES_CHAR_125B0D0  $c0       'c|w{\xf2
.data:0125B0D0   RijnDael_AES_LONG_125B0D0  $c0       'c|w{\xf2
```

这个函数被sub_10F1B80调用。显然是个检查函数。



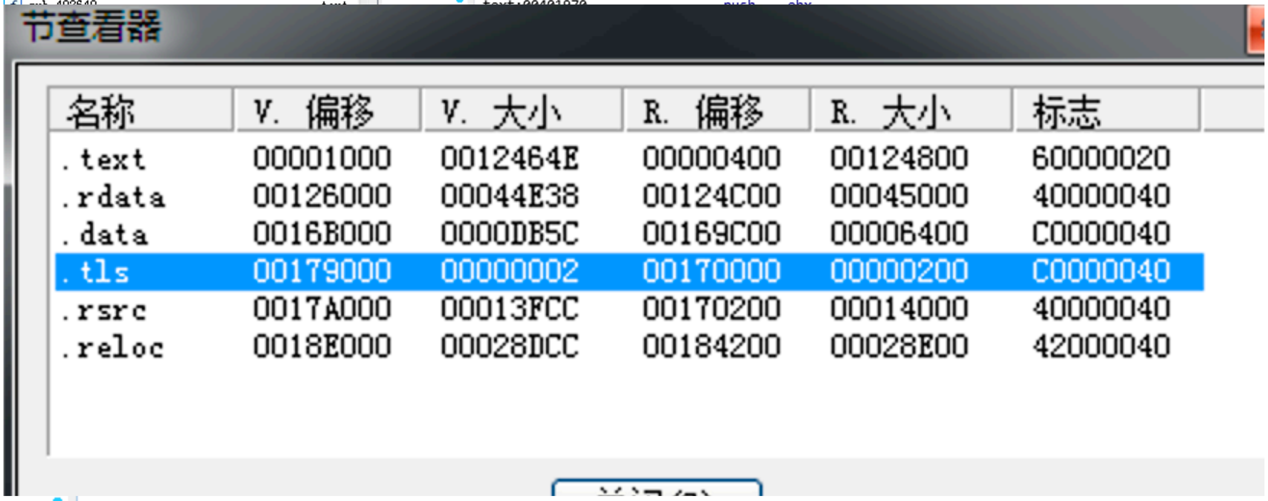
```
int __stdcall sub_10F1B80(int a1)
{
  AESencode();
  if ( (d + c + b + a) == 71
    && (x + g + f) == 3
    && a == b + 68
    && b == c + 2
    && c == d - 59
    && g == e + 10
    && g == x + 9
    && e == f + 52 )
  {
    JUMPOUT(__CS__, 0x1947 + 0x10F0000);
  }
  return 0;
}
```

同时这里看到了tls猜测有tls反调试，用peid证实了。用ida调试执行到了不能反汇编的代码，可以用ida强制分析。猜测是手写的汇编



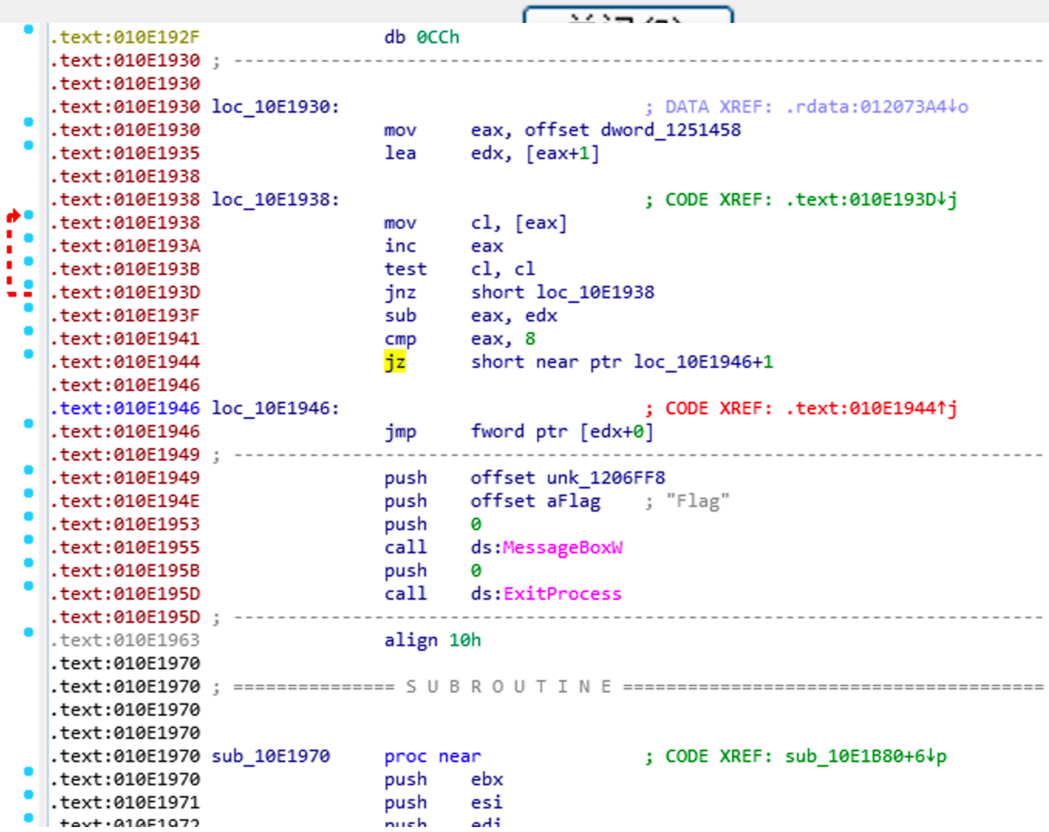
```
sub_4013A0                             .text
sub_401420                             .text
sub_401440                             .text
sub_401560                             .text
sub_401590                             .text
sub_4015C0                             .text
sub_401630                             .text
TlsCallback_0                          .text
Concurrency::details::SchedulingRing::··· .text
sub_401970                             .text
sub_401B80                             .text
sub_401CA0                             .text
sub_401CB0                             .text
StartAddress                           .text
sub_401D80                             .text
sub_401E10                             .text
sub_401E80                             .text
sub_401EC0                             .text
sub_401EE0                             .text
sub_401F00                             .text
sub_401F20                             .text
sub_401F30                             .text
sub_401F60                             .text
sub_402030                             .text
sub_402050                             .text
sub_402060                             .text
sub_4021D0                             .text
sub_402310                             .text
sub_402340                             .text
sub_4024F0                             .text
sub_402620                             .text
sub_402630                             .text
```

```
.text:00401916
.text:00401916 loc_401916:                             ; CODE XREF: .text:0040191B↓j
.text:00401916                 mov     cl, [eax]
.text:00401918                 inc     eax
.text:00401919                 test    cl, cl
.text:0040191B                 jnz     short loc_401916
.text:0040191D                 sub     eax, edx
.text:0040191F                 cmp     eax, 8
.text:00401922                 setnz   al
.text:00401925                 pop     esi
.text:00401926                 mov     esp, ebp
.text:00401928                 pop     ebp
.text:00401929                 retn
.text:00401929 ; ---------------------------------------------------------------------------
.text:0040192A                 align 10h
.text:00401930 byte_401930     db 0B8h                 ; DATA XREF: .rdata:005273A4↓o
.text:00401931                 dd offset dword_571458
.text:00401935                 db 8Dh, 50h, 1
.text:00401938                 dd 8440088Ah, 2BF975C9h, 8F883C2h, 6AFF0174h
.text:00401948                 db 0, 68h
.text:0040194A                 dd offset unk_526FF8
.text:0040194E ; ---------------------------------------------------------------------------
.text:0040194E                 push    offset aFlag    ; "Flag"
.text:00401953                 push    0
.text:00401955                 call    ds:MessageBoxW
.text:0040195B                 push    0
.text:0040195D                 call    ds:ExitProcess
.text:0040195D ; ---------------------------------------------------------------------------
.text:00401963                 align 10h
.text:00401970
.text:00401970 ; =============== S U B R O U T I N E =======================================
.text:00401970
.text:00401970
.text:00401970 sub_401970      proc near               ; CODE XREF: sub_401B80+6↓p
```



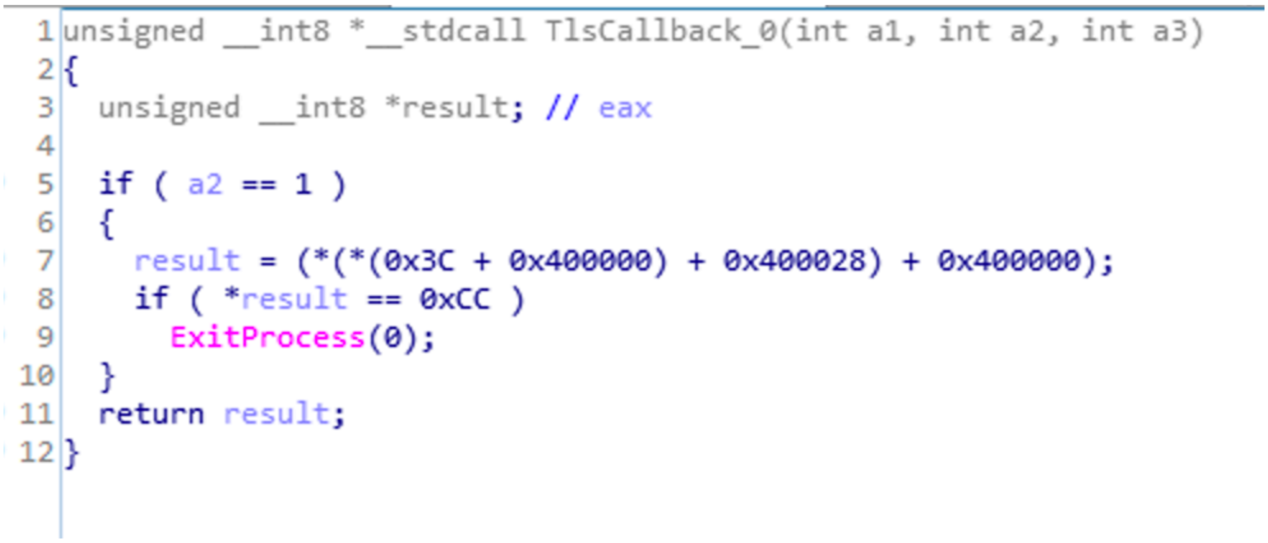
节查看器

名称	V. 偏移	V. 大小	R. 偏移	R. 大小	标志
.text	00001000	0012464E	00000400	00124800	60000020
.rdata	00126000	00044E38	00124C00	00045000	40000040
.data	0016B000	0000DB5C	00169C00	00006400	C0000040
.tls	00179000	00000002	00170000	00000200	C0000040
.rsrc	0017A000	00013FCC	00170200	00014000	40000040
.reloc	0018E000	00028DCC	00184200	00028E00	42000040



```
.text:010E192F                 db 0CCh
.text:010E1930 ; ---------------------------------------------------------------------------
.text:010E1930
.text:010E1930 loc_10E1930:                            ; DATA XREF: .rdata:012073A4↓o
.text:010E1930                 mov     eax, offset dword_1251458
.text:010E1935                 lea     edx, [eax+1]
.text:010E1938
.text:010E1938 loc_10E1938:                            ; CODE XREF: .text:010E193D↓j
.text:010E1938                 mov     cl, [eax]
.text:010E193A                 inc     eax
.text:010E193B                 test    cl, cl
.text:010E193D                 jnz     short loc_10E1938
.text:010E193F                 sub     eax, edx
.text:010E1941                 cmp     eax, 8
.text:010E1944                 jz      short near ptr loc_10E1946+1
.text:010E1946
.text:010E1946 loc_10E1946:                            ; CODE XREF: .text:010E1944↑j
.text:010E1946                 jmp     fword ptr [edx+0]
.text:010E1949 ; ---------------------------------------------------------------------------
.text:010E1949                 push    offset unk_1206FF8
.text:010E194E                 push    offset aFlag    ; "Flag"
.text:010E1953                 push    0
.text:010E1955                 call    ds:MessageBoxW
.text:010E195B                 push    0
.text:010E195D                 call    ds:ExitProcess
.text:010E195D ; ---------------------------------------------------------------------------
.text:010E1963                 align 10h
.text:010E1970
.text:010E1970 ; =============== S U B R O U T I N E =======================================
.text:010E1970
.text:010E1970
.text:010E1970 sub_10E1970     proc near               ; CODE XREF: sub_10E1B80+6↓p
.text:010E1970                 push    ebx
.text:010E1971                 push    esi
```

tls好像只是检查了断点。//但是后来解出flag后，尝试将tls内容直接修改成ret程序不能识别正确flag，所以可以判断tls还包括打乱S盒的函数（但是没找到在哪被改变）



```c
unsigned __int8 *__stdcall TlsCallback_0(int a1, int a2, int a3)
{
  unsigned __int8 *result; // eax

  if ( a2 == 1 )
  {
    result = (*(*(0x3C + 0x400000) + 0x400028) + 0x400000);
    if ( *result == 0xCC )
      ExitProcess(0);
  }
  return result;
}
```

ok 现在目的很明确，先求出这几个方程的解

```python
from z3 import *
a,b,c,d,e,f,g,x = Ints('a b c d e f g x')
solv = Solver()
#这里的变量最大一个字节
solv.add(a<0x100)
solv.add(b<0x100)
solv.add(c<0x100)
solv.add(d<0x100)
solv.add(e<0x100)
solv.add(f<0x100)
solv.add(g<0x100)
solv.add(x<0x100)
solv.add(a>=0)
solv.add(b>=0)
solv.add(c>=0)
solv.add(d>=0)
solv.add(e>=0)
solv.add(f>=0)
solv.add(g>=0)
solv.add(x>=0)
#这里有个两比较需要小于0x100，否则无解
solv.add((a+b+c+d)%0x100==71)
solv.add((x+f+g)%0x100==3)
solv.add((b+68)==a)
solv.add((c+2)==b)
solv.add((d-59)==c)
solv.add((e+10)==g)
solv.add((x+9)==g)
solv.add((f+52)==e)

print(solv.check())

print(solv.model())


```

得到

> [f = 48, b = 115, a = 183, d = 172, g = 110, c = 113, e = 100, x = 101]

本来想把求到的变量丢到AES的S^{-1}盒里，但是发现S盒被改过，所以在od里把表复制了出来用，结果调试能过但实际程序不能过。最后发现程序运行时dump下来的表才是正确的//所以这里的tls不能简单地绕过，程序还有一个进程是用来修改这个S盒的 正确的表：

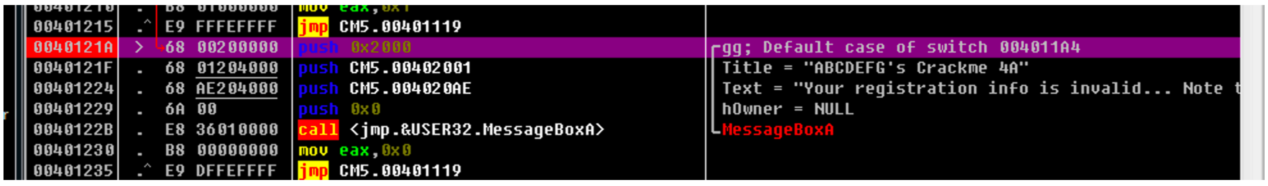
```
[0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x1,0x67,0x2B,0xFE,0xD7,0xAB,0x76,0xCA,0x82,0xC9,0x7D,0xFA,0x59,0x47,0xF0,0xAD,0xD4,0xA2,0xAF,0x9C,0xA4,0x72,0xC0,0xB7,0xFD,0x93,0x26,0x36,0x3F,0xF7,0xCC,0x34,0xA5,0xE5,0xF1,0x71,0xD8,0x31,0x15,0x4,0xC7,0x23,0xC3,0x18,0x96,0x5,0x9A,0x7,0x12,0x80,0xE2,0xEB,0x27,0xB2,0x75,0x9,0x83,0x2C,0x1A,0x1B,0x6E,0x5A,0xA0,0x52,0x3B,0xD6,0xB3,0x29,0xE3,0x2F,0x84,0x53,0xD1,0x0,0xED,0x20,0xFC,0xB1,0x5B,0x6A,0xCB,0xBE,0x39,0x4A,0x4C,0x58,0xCF,0xD0,0xEF,0xAA,0xFB,0x43,0x4D,0x33,0x85,0x45,0xF9,0x2,0x7F,0x50,0x3C,0x9F,0xA8,0x51,0xA3,0x40,0x8F,0x92,0x9D,0x38,0xF5,0xBC,0xB6,0xDA,0x21,0x10,0xFF,0xF3,0xD2,0xCD,0x0C,0x13,0xEC,0x5F,0x97,0x44,0x17,0xC4,0xA7,0x7E,0x3D,0x64,0x5D,0x19,0x73,0x60,0x81,0x4F,0xDC,0x22,0x2A,0x90,0x88,0x46,0xEE,0xB8,0x14,0xDE,0x5E,0x0B,0xDB,0xE0,0x32,0x3A,0x0A,0x49,0x6,0x24,0x5C,0xC2,0xD3,0xAC,0x62,0x91,0x95,0xE4,0x79,0xE7,0xC8,0x37,0x6D,0x8D,0xD5,0x4E,0xA9,0x6C,0x56,0xF4,0xEA,0x65,0x7A,0xAE,0x8,0xBA,0x78,0x25,0x2E,0x1C,0xA6,0xB4,0xC6,0xE8,0xDD,0x74,0x1F,0x4B,0xBD,0x8B,0x8A,0x70,0x3E,0xB5,0x66,0x48,0x3,0xF6,0x0E,0x61,0x35,0x57,0xB9,0x86,0xC1,0x1D,0x9E,0xE1,0xF8,0x98,0x11,0x69,0xD9,0x8E,0x94,0x9B,0x1E,0x87,0xE9,0xCE,0x55,0x28,0xDF,0x8C,0xA1,0x89,0x0D,0xBF,0xE6,0x42,0x68,0x41,0x99,0x2D,0x0F,0xB0,0x54,0xBB,0x16]

```

接下来就好办了

```python
dir=[0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x1,0x67,0x2B,0xFE,0xD7,0xAB,0x76,0xCA,0x82,0xC9,0x7D,0xFA,0x59,0x47,0xF0,0xAD,0xD4,0xA2,0xAF,0x9C,0xA4,0x72,0xC0,0xB7,0xFD,0x93,0x26,0x36,0x3F,0xF7,0xCC,0x34,0xA5,0xE5,0xF1,0x71,0xD8,0x31,0x15,0x4,0xC7,0x23,0xC3,0x18,0x96,0x5,0x9A,0x7,0x12,0x80,0xE2,0xEB,0x27,0xB2,0x75,0x9,0x83,0x2C,0x1A,0x1B,0x6E,0x5A,0xA0,0x52,0x3B,0xD6,0xB3,0x29,0xE3,0x2F,0x84,0x53,0xD1,0x0,0xED,0x20,0xFC,0xB1,0x5B,0x6A,0xCB,0xBE,0x39,0x4A,0x4C,0x58,0xCF,0xD0,0xEF,0xAA,0xFB,0x43,0x4D,0x33,0x85,0x45,0xF9,0x2,0x7F,0x50,0x3C,0x9F,0xA8,0x51,0xA3,0x40,0x8F,0x92,0x9D,0x38,0xF5,0xBC,0xB6,0xDA,0x21,0x10,0xFF,0xF3,0xD2,0xCD,0x0C,0x13,0xEC,0x5F,0x97,0x44,0x17,0xC4,0xA7,0x7E,0x3D,0x64,0x5D,0x19,0x73,0x60,0x81,0x4F,0xDC,0x22,0x2A,0x90,0x88,0x46,0xEE,0xB8,0x14,0xDE,0x5E,0x0B,0xDB,0xE0,0x32,0x3A,0x0A,0x49,0x6,0x24,0x5C,0xC2,0xD3,0xAC,0x62,0x91,0x95,0xE4,0x79,0xE7,0xC8,0x37,0x6D,0x8D,0xD5,0x4E,0xA9,0x6C,0x56,0xF4,0xEA,0x65,0x7A,0xAE,0x8,0xBA,0x78,0x25,0x2E,0x1C,0xA6,0xB4,0xC6,0xE8,0xDD,0x74,0x1F,0x4B,0xBD,0x8B,0x8A,0x70,0x3E,0xB5,0x66,0x48,0x3,0xF6,0x0E,0x61,0x35,0x57,0xB9,0x86,0xC1,0x1D,0x9E,0xE1,0xF8,0x98,0x11,0x69,0xD9,0x8E,0x94,0x9B,0x1E,0x87,0xE9,0xCE,0x55,0x28,0xDF,0x8C,0xA1,0x89,0x0D,0xBF,0xE6,0x42,0x68,0x41,0x99,0x2D,0x0F,0xB0,0x54,0xBB,0x16]

num = [183,115,113,172,100,48,110,101]

for i in range(len(num)):
    for j in range(64*4):
        num[i] = dir.index(num[i])

flag = ''

for i in range(len(num)):
    flag+=hex(num[i])[2:]

print '0'+flag.upper()

```

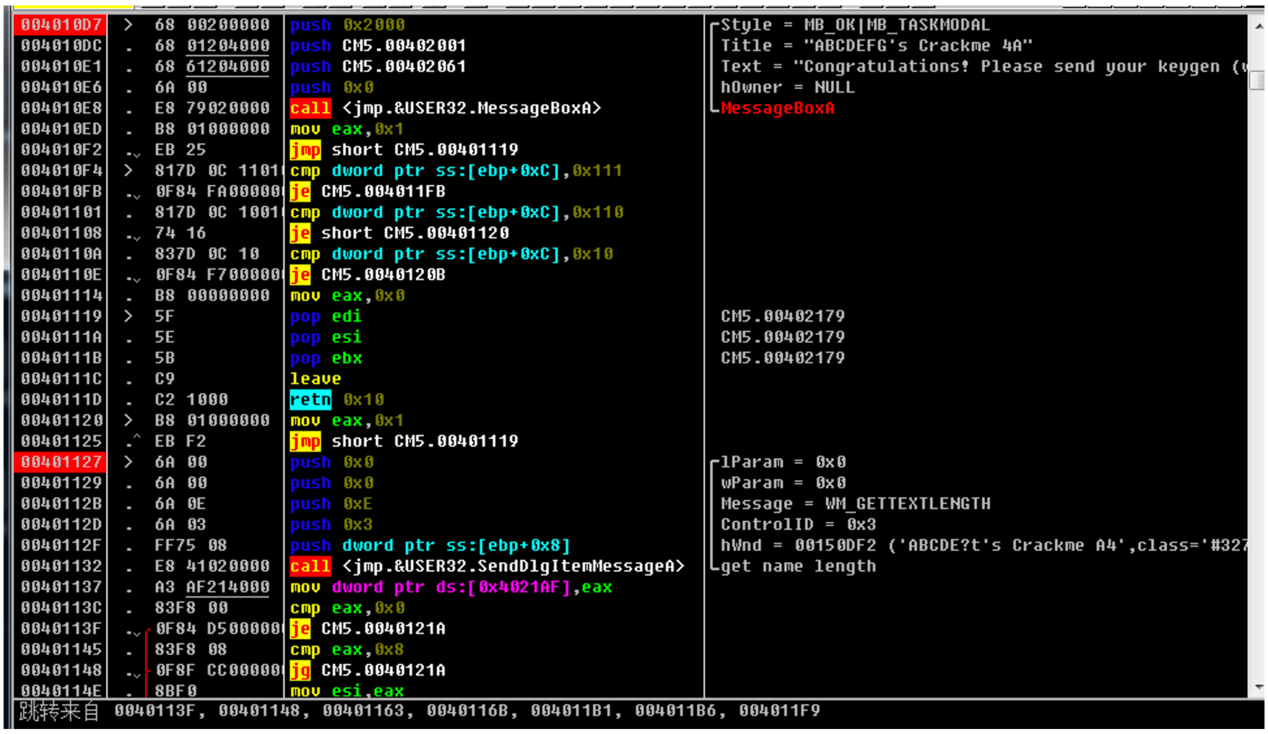
cm5

od里可以搜到字符串，找到弹窗，在弹窗处下断点，可以从栈找到jmp过来的地址//失败的跳转是40121a



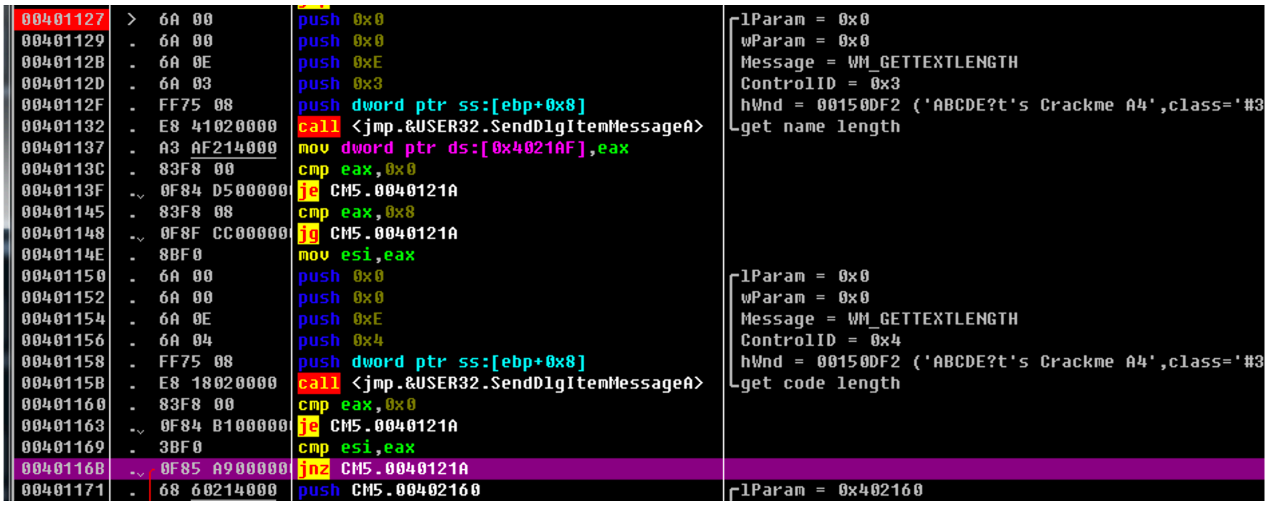
```
00401210  .  B8 01000000  mov eax,0x1
00401215  .^ E9 FFFEFFFF  jmp CM5.00401119
0040121A  >  68 00200000  push 0x2000             ┌gg; Default case of switch 004011A4
0040121F  .  68 01204000  push CM5.00402001       │Title = "ABCDEFG's Crackme 4A"
00401224  .  68 AE204000  push CM5.004020AE       │Text = "Your registration info is invalid... Note t
00401229  .  6A 00        push 0x0                │hOwner = NULL
0040122B  .  E8 36010000  call <jmp.&USER32.MessageBoxA>  └MessageBoxA
00401230  .  B8 00000000  mov eax,0x0
00401235  .^ E9 DFFEFFFF  jmp CM5.00401119
```

找到跳转过来的最远的地方，在函数入口下断点 //往上翻一下可以看到成功的窗口，所以成功的跳转是4010d7



```
004010D7  >  68 00200000   push 0x2000                                     ┌Style = MB_OK|MB_TASKMODAL
004010DC  .  68 01204000   push CM5.00402001                               │Title = "ABCDEFG's Crackme 4A"
004010E1  .  68 61204000   push CM5.00402061                               │Text = "Congratulations! Please send your keygen (
004010E6  .  6A 00         push 0x0                                        │hOwner = NULL
004010E8  .  E8 79020000   call <jmp.&USER32.MessageBoxA>                  └MessageBoxA
004010ED  .  B8 01000000   mov eax,0x1
004010F2  .v EB 25         jmp short CM5.00401119
004010F4  >  817D 0C 1101  cmp dword ptr ss:[ebp+0xC],0x111
004010FB  .v 0F84 FA00000  je CM5.004011FB
00401101  .  817D 0C 1001  cmp dword ptr ss:[ebp+0xC],0x110
00401108  .v 74 16         je short CM5.00401120
0040110A  .  837D 0C 10    cmp dword ptr ss:[ebp+0xC],0x10
0040110E  .v 0F84 F700000  je CM5.0040120B
00401114  .  B8 00000000   mov eax,0x0
00401119  >  5F            pop edi                                          CM5.00402179
0040111A  .  5E            pop esi                                          CM5.00402179
0040111B  .  5B            pop ebx                                          CM5.00402179
0040111C  .  C9            leave
0040111D  .  C2 1000       retn 0x10
00401120  >  B8 01000000   mov eax,0x1
00401125  .^ EB F2         jmp short CM5.00401119
00401127  >  6A 00         push 0x0                                        ┌lParam = 0x0
00401129  .  6A 00         push 0x0                                        │wParam = 0x0
0040112B  .  6A 0E         push 0xE                                        │Message = WM_GETTEXTLENGTH
0040112D  .  6A 03         push 0x3                                        │ControlID = 0x3
0040112F  .  FF75 08       push dword ptr ss:[ebp+0x8]                     │hWnd = 00150DF2 ('ABCDE?t's Crackme A4',class='#327
00401132  .  E8 41020000   call <jmp.&USER32.SendDlgItemMessageA>          └get name length
00401137  .  A3 AF214000   mov dword ptr ds:[0x4021AF],eax
0040113C  .  83F8 00       cmp eax,0x0
0040113F  .v 0F84 D500000  je CM5.0040121A
00401145  .  83F8 08       cmp eax,0x8
00401148  .v 0F8F CC00000  jg CM5.0040121A
0040114E  .  8BF0          mov esi,eax
跳转来自 0040113F, 00401148, 00401163, 0040116B, 004011B1, 004011B6, 004011F9
```

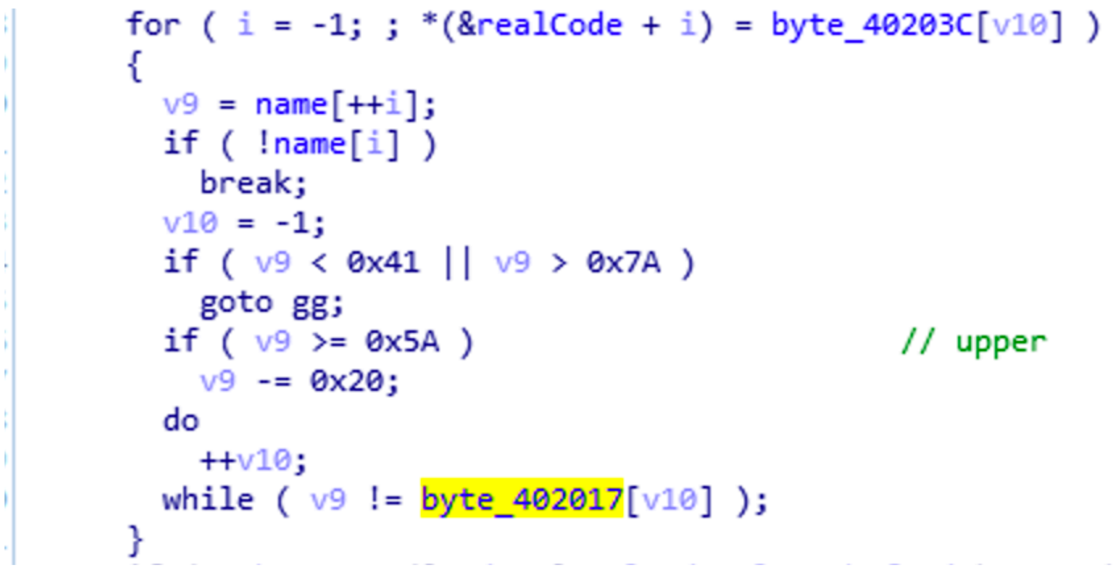
单步调，观察前两个call前的push，和call之后的eax，可以发现401132的call是取得name的长度（name长度不能超过8）， 40115b处的call是取得code的长度，然后两者cmp，必须相同否则跳到40121a



```
00401127  >  6A 00         push 0x0                                        ┌lParam = 0x0
00401129  .  6A 00         push 0x0                                        │wParam = 0x0
0040112B  .  6A 0E         push 0xE                                        │Message = WM_GETTEXTLENGTH
0040112D  .  6A 03         push 0x3                                        │ControlID = 0x3
0040112F  .  FF75 08       push dword ptr ss:[ebp+0x8]                     │hWnd = 00150DF2 ('ABCDE?t's Crackme A4',class='#3
00401132  .  E8 41020000   call <jmp.&USER32.SendDlgItemMessageA>          └get name length
00401137  .  A3 AF214000   mov dword ptr ds:[0x4021AF],eax
0040113C  .  83F8 00       cmp eax,0x0
0040113F  .v 0F84 D500000  je CM5.0040121A
00401145  .  83F8 08       cmp eax,0x8
00401148  .v 0F8F CC00000  jg CM5.0040121A
0040114E  .  8BF0          mov esi,eax
00401150  .  6A 00         push 0x0                                        ┌lParam = 0x0
00401152  .  6A 00         push 0x0                                        │wParam = 0x0
00401154  .  6A 0E         push 0xE                                        │Message = WM_GETTEXTLENGTH
00401156  .  6A 04         push 0x4                                        │ControlID = 0x4
00401158  .  FF75 08       push dword ptr ss:[ebp+0x8]                     │hWnd = 00150DF2 ('ABCDE?t's Crackme A4',class='#3
0040115B  .  E8 18020000   call <jmp.&USER32.SendDlgItemMessageA>          └get code length
00401160  .  83F8 00       cmp eax,0x0
00401163  .v 0F84 B100000  je CM5.0040121A
00401169  .  3BF0          cmp esi,eax
0040116B  .v 0F85 A900000  jnz CM5.0040121A
00401171  .  68 60214000   push CM5.00402160                               ┌lParam = 0x402160
```

接下来两个call分别取得了name和code//可以观察这两个call的第一个push

然后是一个循环，可以发现这个循环是通过name构造一个code（一个从402017到40203c的映射）。然后在sub_401244和输入的code进行对比



```
for ( i = -1; ; *(&realCode + i) = byte_40203C[v10] )
{
  v9 = name[++i];
  if ( !name[i] )
    break;
  v10 = -1;
  if ( v9 < 0x41 || v9 > 0x7A )
    goto gg;
  if ( v9 >= 0x5A )                          // upper
    v9 -= 0x20;
  do
    ++v10;
  while ( v9 != byte_402017[v10] );
}
```

```python
newcode = 'SU7CSJKF09NCSDO9SDF09SDRLVK7809S4NF'

name = 'A1LSK2DJF4HGP3QWO5EIR6UTYZ8MXN7CBV9'

inputName = raw_input('input you name(a~z or A~Z) (no longer than 8): \n')

inputName = inputName.upper()

code = ''

for i in range(len(inputName)):
    code += newcode[name.index(inputName[i])]

print code
```

aiQG_

2019.04