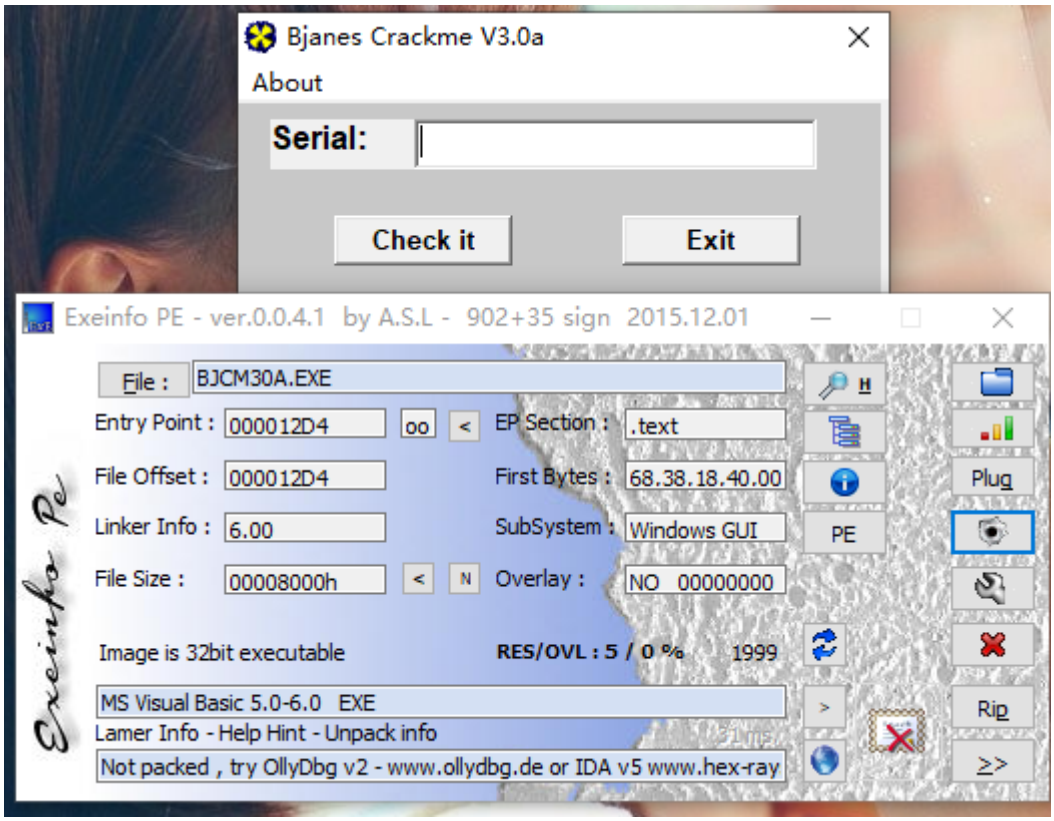
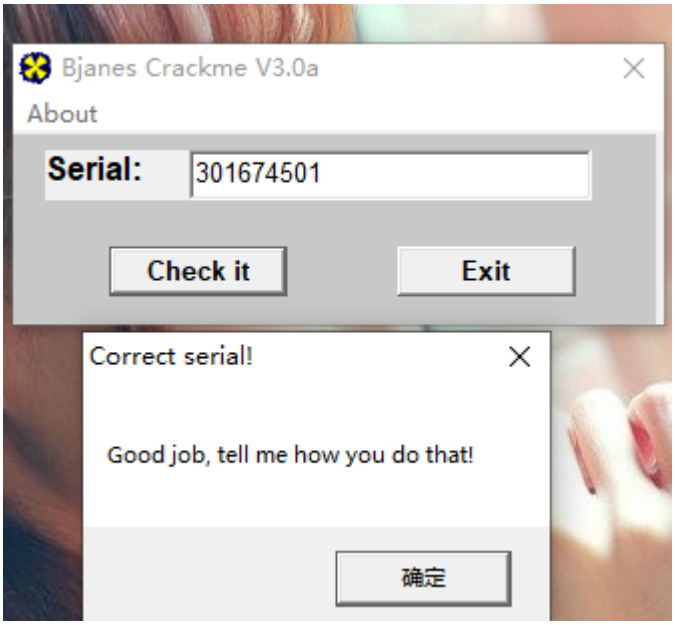
查壳



一样的014和017?

这个Crackme和014和016是同一个作者，不会又是重复的吧？输入014的序列号试试



好吧，虽然是能蒙对，但其实里面的算法是完全不一样的，这到底是为什么？这个问题留到最后面

直接根据字符串的错误提示，来到函数头的位置，也就是按钮事件开头，来分析整个算法

破解反调试

这个Crackme跟其他的不一样的地方是有一个时间戳检查，

```
00404314  .  89BD 54FEFFFF  mov dword ptr ss:[ebp-0x1AC],edi
0040431A  .  89BD 44FEFFFF  mov dword ptr ss:[ebp-0x1BC],edi
00404320  .  FF15 94104000  call dword ptr ds:[<&MSVBVM60.#rtcGetTimer_5  第一次获取时间戳
00404326  .  FF15 D0104000  call dword ptr ds:[<&MSVBVM60.__vbaFpI4>]    msvbvm60.__vbaFpI4
0040432C  .  8945 A4        mov dword ptr ss:[ebp-0x5C],eax              保存结果
0040432F  .  8D95 08FFFFFF  lea edx,dword ptr ss:[ebp-0xF8]
00404335  .  8D85 F8FEFFFF  lea eax,dword ptr ss:[ebp-0x108]
0040433B  .  52             push edx                                     Step8 = 00000006
```

首先，获取当前事件，保存结果

```
地址      HEX 数据        反汇编                                          注释
00404430  .  8D85 94FEFFFF  lea eax,dword ptr ss:[ebp-0x16C]
00404436  .  52             push edx                                     TMPend8 = 0019F208
00404437  .  8D4D A8        lea ecx,dword ptr ss:[ebp-0x58]
0040443A  .  50             push eax                                     TMPstep8 = 0019F1F
0040443B  .  51             push ecx                                     Counter8 = 0000003
0040443C  .  FF15 E8104000  call dword ptr ds:[<&MSVBVM60.__vbaVarForNext>]  __vbaVarForNext
00404442  .^ EB AF          jmp short BJCM30A.004043F3
00404444  >  8D95 A4FEFFFF  lea edx,dword ptr ss:[ebp-0x15C]
0040444A  .  8D85 B4FEFFFF  lea eax,dword ptr ss:[ebp-0x14C]
00404450  .  52             push edx                                     TMPend8 = 0019F208
00404451  .  8D4D 80        lea ecx,dword ptr ss:[ebp-0x80]
00404454  .  50             push eax                                     TMPstep8 = 0019F1F
00404455  .  51             push ecx                                     Counter8 = 0000003
00404456  .  FF15 E8104000  call dword ptr ds:[<&MSVBVM60.__vbaVarForNext>]  __vbaVarForNext
0040445C  .^ E9 30FFFFFF    jmp BJCM30A.00404391
00404391=BJCM30A.00404391
```

接着是一个两层的嵌套循环，具体什么作用不知道，大概是为了拖延调试者的时间，

```
0040445C  .^ E9 30FFFFFF    jmp BJCM30A.00404391
00404461  >  FF15 9410400(  call dword ptr ds:[<&MSVBVM60.#rtcGetTimer_535>]   第二次获取时间戳
00404467  .  FF15 D010400(  call dword ptr ds:[<&MSVBVM60.__vbaFpI4>]         msvbvm60.__vbaFpI4
0040446D  .  2B45 A4        sub eax,dword ptr ss:[ebp-0x5C]                   计算两次时间的差值
00404470  .v 0F80 340C000(  jo BJCM30A.004050AA
00404476  .  83F8 05        cmp eax,0x5                                       比较是否小于等于5
00404479  .v 0F8E AD000000  jle BJCM30A.0040452C
0040447F  .  8B1D CC104000  mov ebx,dword ptr ds:[<&MSVBVM60.__vbaVarDup>]    msvbvm60.__vbaVarDup
00404485  .  B9 04000280    mov ecx,0x80020004
```

两次循环过后，再次获取时间戳，计算两次获取的时间的差值，比较是否小于等于5，如果小于继续往下走，否则报错

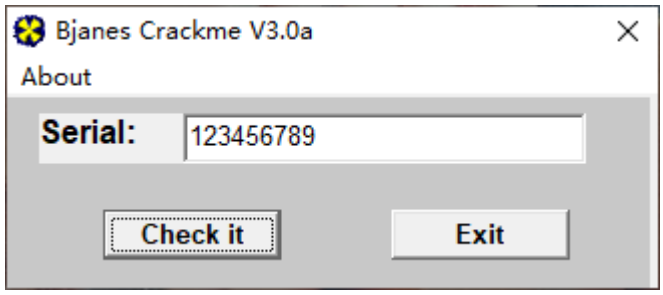
Decompiler | Disassembler | HEX Editor

```
Private Sub Command1_Click() '404230
  Dim var_A8 As TextBox
  loc_0040432C: var_5C = CLng(Timer)        第一次获取时间戳
  loc_0040438E: For var_80 = 1 To 1000 Step 1
  loc_00404393: If var_14C = 0 Then GoTo loc_00404461      两层循环
  loc_004043ED: For var_58 = 1 To 250 Step 1
  loc_004043F5: If var_16C = 0 Then GoTo loc_00404444
  loc_00404409: If (var_00402A34 <> var_00402A34) <> 0 Then GoTo loc_0040442A
  loc_00404424: var_24 = CInt(1)
  loc_0040442A: 'Referenced from: 00404409
  loc_0040443C: Next var_58
  loc_00404442: GoTo loc_004043F3
  loc_00404444: 'Referenced from: 004043F5
  loc_00404456: Next var_80
  loc_0040445C: GoTo loc_00404391
  loc_00404461: 'Referenced from: 00404393
  loc_0040446D: CLng(Timer) = CLng(Timer) - var_5C
  loc_00404479: If CLng(Timer) <= 5 Then GoTo loc_0040452C      计算差值 比较是否小于等于5
  loc_004044B8: var_100 = "Cheater!!!   CHEATER!!!   Cheater!!!   CHEATER!!!"
  loc_004044D6: var_F0 = "  You have SmartCheck loaded!...Close it and try again!!!"
  loc_004044E6: var_B8 = "  You have SmartCheck loaded!...Close it and try again!!!"
  loc_00404505: MsgBox(var_B8, 0, "Cheater!!!   CHEATER!!!   Cheater!!!   CHEATER!!!", 10, 10)
```

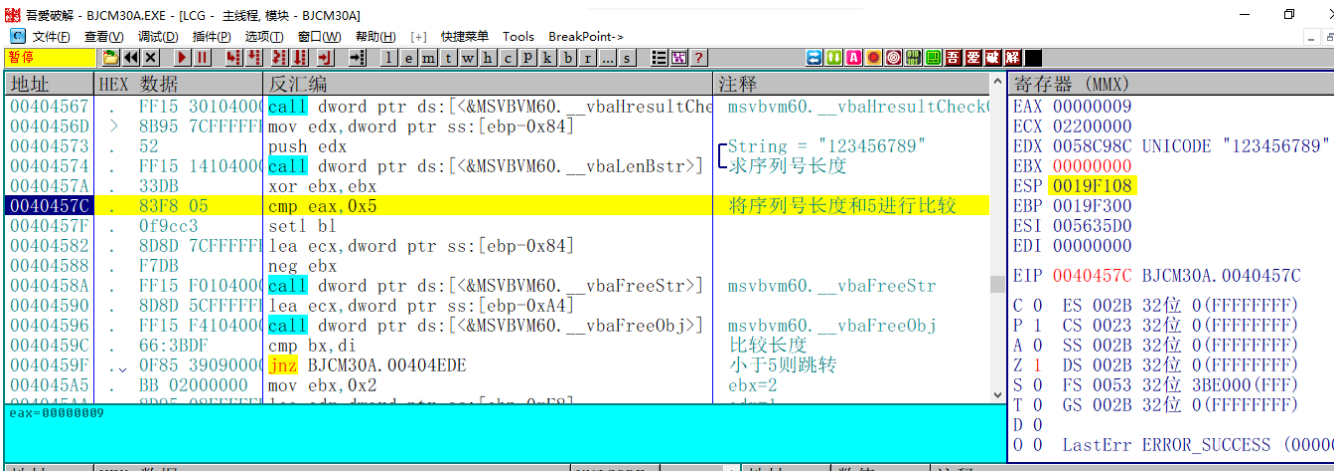
从VB Decomplier反汇编出的N-Code也可以直观的看出结果

知道反调试原理，破解其实就很简单了，只要在跳转的地址处下断点F9运行就可以了，或者修改ZF标志位

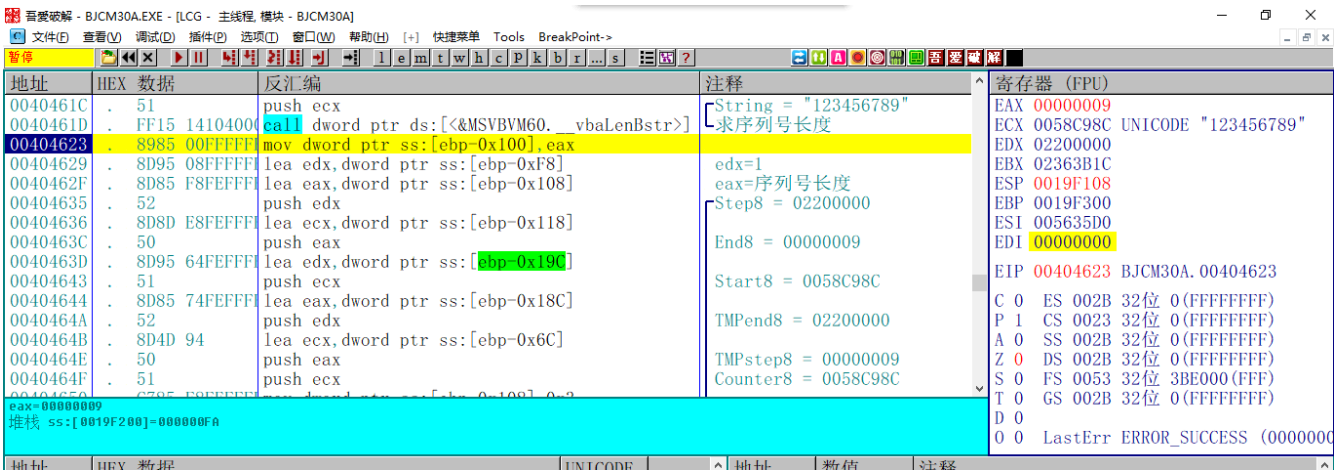
分析算法



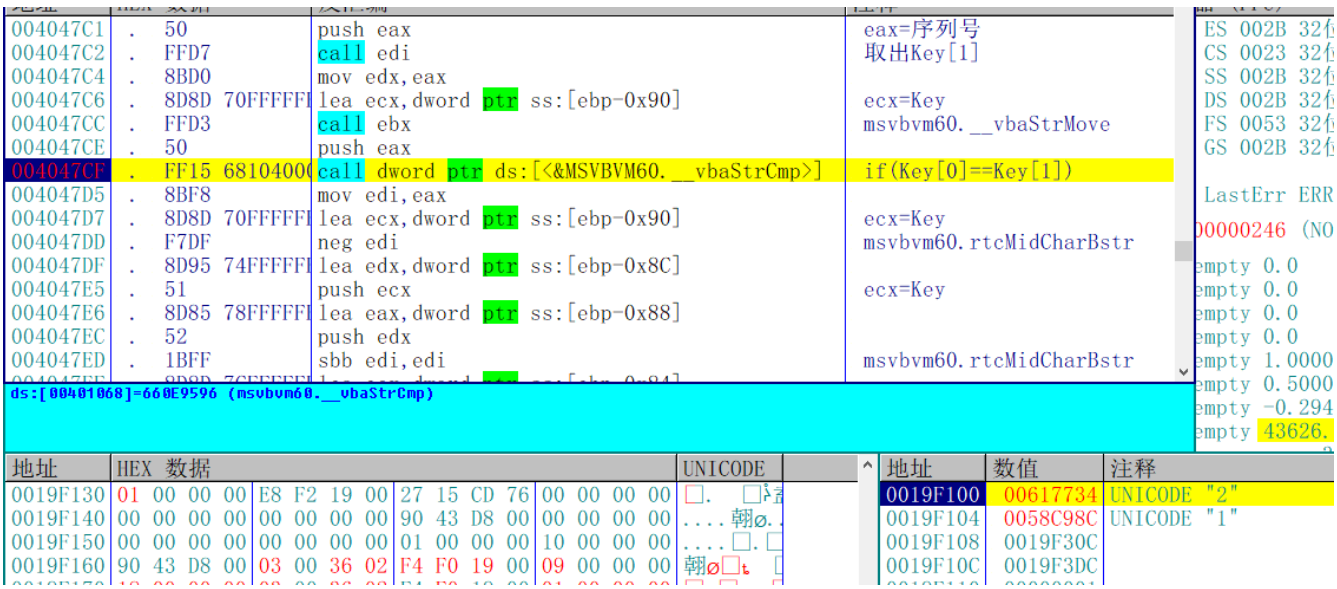
先随便输入一个序列号，开始分析算法



首先求序列号长度，将序列号长度和5进行比较，小于则跳转，即长度必须大于5



接着 再次求序列号长度，将长度作为循环的次数



循环比较序列号两两之间是否相等

```
地址      HEX 数据     反汇编                                    注释
0040482C  .   50       push eax
0040482D  .   6A 03    push 0x3
0040482F  .   FF15 1810400( call dword ptr ds:[<&MSVBVM60.__vbaFreeVarLis  msvbvm60.__vbaFreeVarList
00404835  .   83C4 30  add esp,0x30
00404838  .   66:85FF  test di,di                                 比较序列号的前两位是否相等
0040483B  .v  74 37    je short BJCM30A.00404874                  不相等则跳转
0040483D  .   8D4D B8  lea ecx,dword ptr ss:[ebp-0x48]            相等则记录循环次数->[ebp-0x48]
00404840  .   8D95 08FFFFFI lea edx,dword ptr ss:[ebp-0xF8]
00404846  .   51       push ecx                                   var18 = 00190002
00404847  .   8D85 48FFFFFI lea eax,dword ptr ss:[ebp-0xB8]
0040484D  .   52       push edx                                   var28 = 0019F238
0040484E  .   50       push eax                                   saveto8 = 00000003
0040484F  .   C785 10FFFFFI mov dword ptr ss:[ebp-0xF0],0x1
00404859  .   C785 08FFFFFI mov dword ptr ss:[ebp-0xF8],0x2
00404863  .   FF15 C8104000 call dword ptr ds:[<&MSVBVM60.__vbaVarAdd>]  __vbaVarAdd
```

如果相等 则记录循环次数到[ebp-0x48]处，不相等则不记录

```
地址      HEX 数据     反汇编                                    注释
00404874  >   8D8D 64FEFFFI lea ecx,dword ptr ss:[ebp-0x19C]     ecx=序列号长度
0040487A  .   8D95 74FEFFFI lea edx,dword ptr ss:[ebp-0x18C]     edx=1
00404880  .   51       push ecx                                   TMPend8 = 00000001
00404881  .   8D45 94  lea eax,dword ptr ss:[ebp-0x6C]
00404884  .   52       push edx                                   TMPstep8 = 80000000
00404885  .   50       push eax                                   Counter8 = 00000001
00404886  .   FF15 E8104000 call dword ptr ds:[<&MSVBVM60.__vbaVarForNext  __vbaVarForNext
0040488C  .   8985 30FEFFFI mov dword ptr ss:[ebp-0x1D0],eax
00404892  .   33FF     xor edi,edi
00404894  .^  E9 FFFDFFFF jmp BJCM30A.00404698                   跳转到循环开始处
00404899  >   8B0E     mov ecx,dword ptr ds:[esi]                 BJCM30A.00406A74
0040489B      56       push esi
```

然后开始新一轮循环

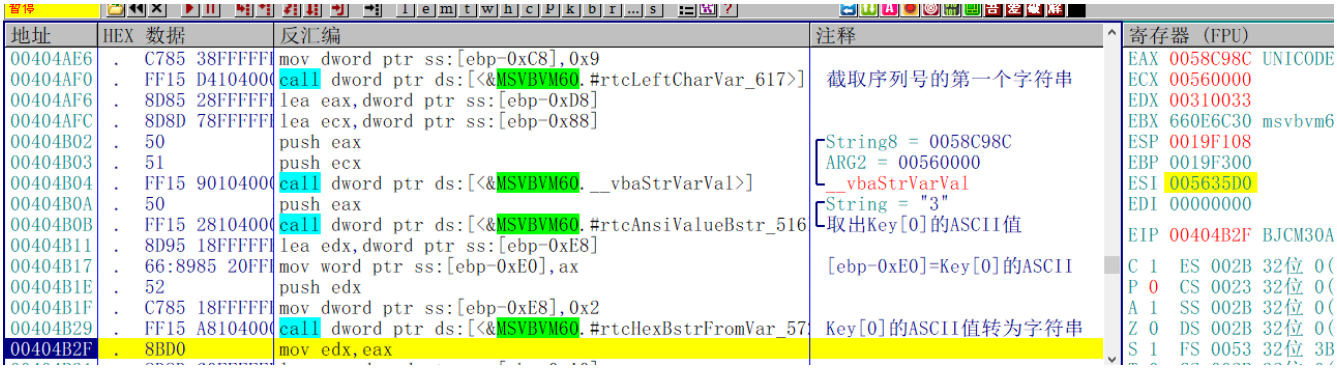
```
地址      HEX 数据     反汇编                                    注释
004048DE  .   FF15 3010400( call dword ptr ds:[<&MSVBVM60.__vbaIlresultChe  msvbvm60.__vbaIlresultCheckObj
004048E4  >   8B95 7CFFFFFI mov edx,dword ptr ss:[ebp-0x84]
004048EA  .   52       push edx                                   String = "123456789"
004048EB  .   FF15 1410400( call dword ptr ds:[<&MSVBVM60.__vbaLenBstr>]  求序列号长度
004048F1  .   83E8 01  sub eax,0x1                                序列号长度-1
004048F4  .   8D8D 08FFFFFI lea ecx,dword ptr ss:[ebp-0xF8]       ecx=序列号长度-1
004048FA  .v  0F80 AA070000 jo BJCM30A.004050AA
00404900  .   8985 10FFFFFI mov dword ptr ss:[ebp-0xF0],eax
00404906  .   8D45 B8  lea eax,dword ptr ss:[ebp-0x48]            [ebp-0x48]=记录的循环次数
00404909  .   50       push eax                                   var18 = 0019F2B8
0040490A  .   51       push ecx                                   var28 = 0019F208
0040490B  .   C785 08FFFFFI mov dword ptr ss:[ebp-0xF8],0x8003
00404915  .   FF15 6C104000 call dword ptr ds:[<&MSVBVM60._vbaVarTstEq>]  比较记录的循环次数和序列号长度-1
0040491B  .   8D8D 7CFFFFFI lea ecx,dword ptr ss:[ebp-0x84]
00404921  .   66:8985 CCFEI mov word ptr ss:[ebp-0x134],ax
```

循环结束之后，将记录的循环次数和序列号长度-1进行比较，比较成功则报错，也就是说序列号的每一位不能为同一个数字

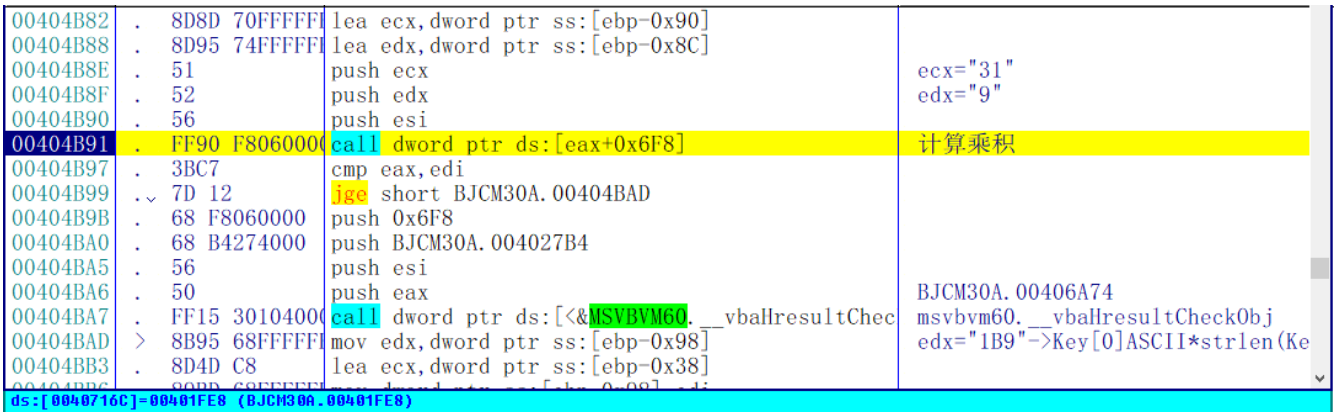
接下来才是真正的序列号算法，这个算法也是很有意思

```
地址      HEX 数据     反汇编                                    注释                          寄存器 (FPU)
00404A7F  .   50       push eax                                  String = "9"                  EAX 0058C7D4 UNICODE "9"
00404A80  .   FF15 1410400( call dword ptr ds:[<&MSVBVM60.__vbaLenBstr>]  求序列号长度          ECX 00560000
00404A86  .   8D8D 48FFFFFI lea ecx,dword ptr ss:[ebp-0xB8]     ecx=1                         EDX 00000002
00404A8C  .   8985 50FFFFFI mov dword ptr ss:[ebp-0xB0],eax      [ebp-0xB0]=序列号长度         EBX 660E6C30 msvbvm60.__v
00404A92  .   51       push ecx                                                                ESP 0019F108
00404A93  .   C785 48FFFFFI mov dword ptr ss:[ebp-0xB8],0x3                                    EBP 0019F300
00404A9D  .   FF15 A8104000 call dword ptr ds:[<&MSVBVM60.#rtcHexBstrFromVar_57  将序列号长度转为字符串  ESI 005635D0
00404AA3  .   8BD0     mov edx,eax                                                             EDI 00000000
00404AA5  .   8D8D 64FFFFFI lea ecx,dword ptr ss:[ebp-0x9C]                                    EIP 00404AA3 BJCM30A.0040
00404AAB  .   FFD3     call ebx                                  msvbvm60.__vbaStrMove
00404AAD  .   8B16     mov edx,dword ptr ds:[esi]                BJCM30A.00406A74            C 1  ES 002B 32位 0(FFFF
00404AAF  .   56       push esi                                                              P 0  CS 0023 32位 0(FFFF
00404AB0  .   FF92 0803000( call dword ptr ds:[edx+0x308]                                    A 1  SS 002B 32位 0(FFFF
00404AB6  .   50       push eax                                                              Z 0  DS 002B 32位 0(FFFF
00404AB7  .   8D85 58FFFFFI lea eax,dword ptr ss:[ebp-0xA8]                                   S 1  FS 0053 32位 3BE000(
                                                                                             T 0  GS 002B 32位 0(FFFF
eax=0058C7D4, (UNICODE "9")                                                                  D 0
edx=00000002
```

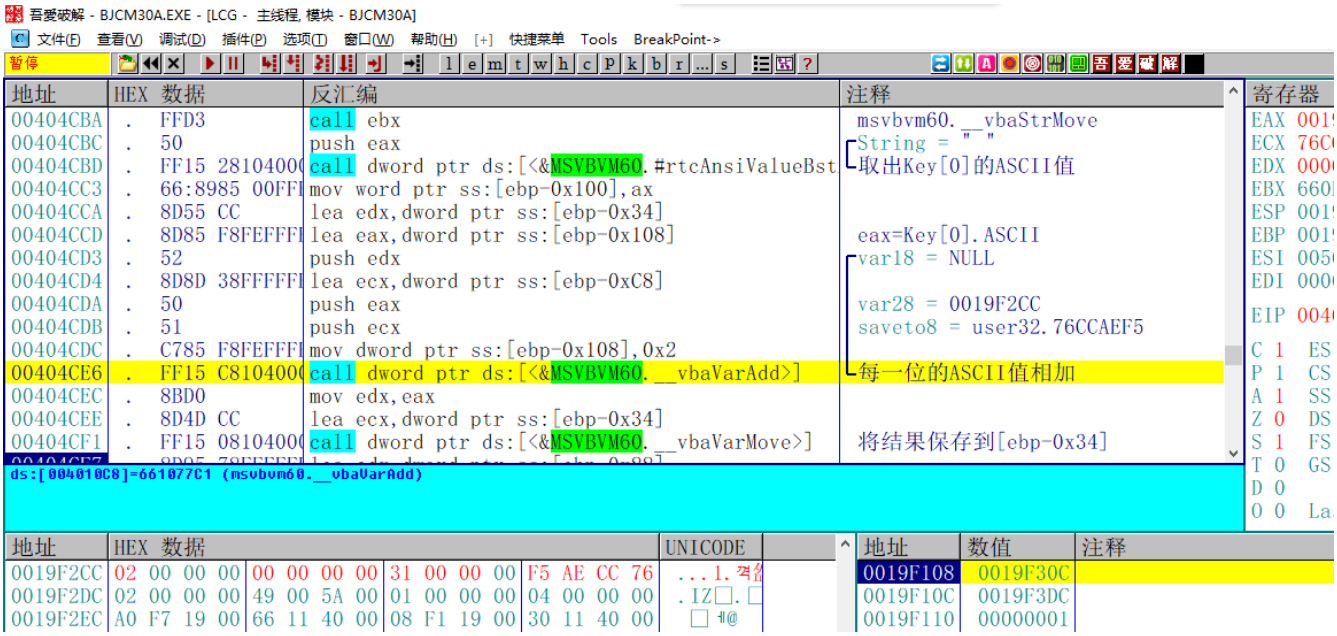
首先计算出序列号的长度，将长度转为字符串



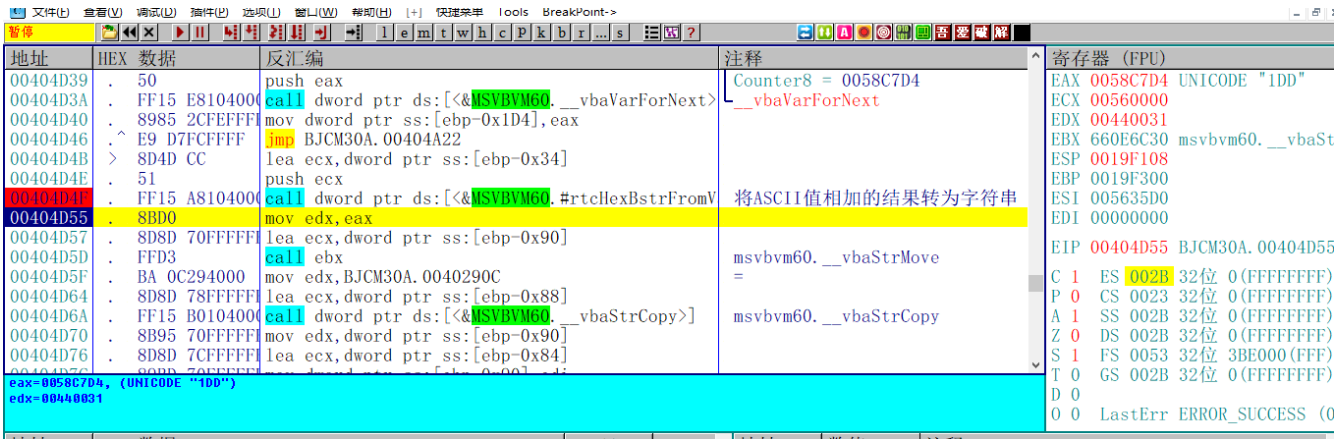
接着，再取出序列号的第一个字符，将ASCII值转成字符串。



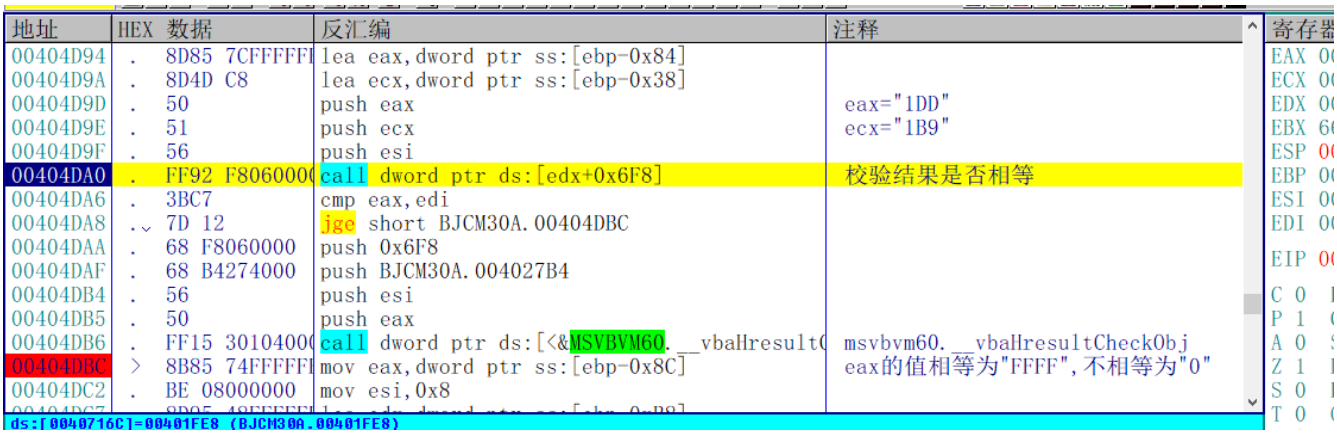
接着计算这两个的乘积，结果为0x1B9



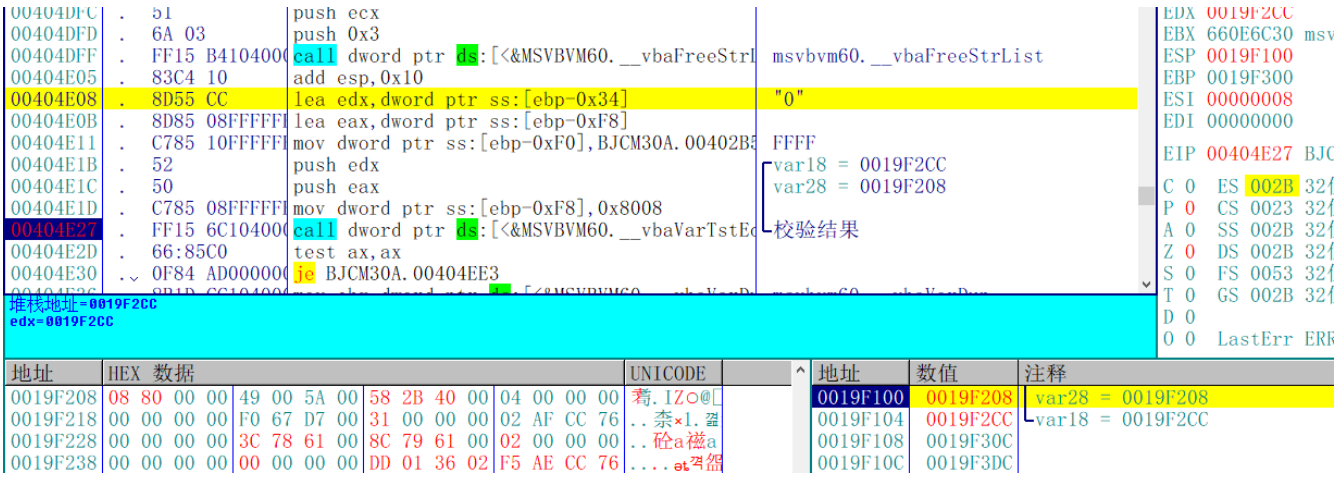
然后将计算序列号每一个为ASCII值相加的结果，将结果保存到[ebp-0x34]



接着将序列号ASCII相加的结果转为字符串->"1DD"



然后比较"1DD"和"1B9"是否相等，eax的值相等为"FFFF",不相等为"0"



接着校验刚才的比较结果，根据结果提示正确或者失败

也就是说这个Crackme的序列号并不是只有唯一解，必须满足以下三个条件

1. 序列号每一位不能为同一个数字
2. 序列号长度必须大于5
3. 序列号第一个字符的ASCII值乘以序列号长度必须等于序列号每一位的ASCII和

014和017区别

那么再来回到一开始的问题，为什么算法不一样但是用014的序列号301674501依然可以通过校验

1. 301674501满足前两个条件 长度大于5 每一位不相同
2. 301674501的ASCII的和为0x1CB
3. 301674501第一位的ASCII值33*9(序列号长度)也是等于0x1CB

所以这个序列号才能通过校验，估计作者是先写的017这个Crackme，然后将017其中的一个解直接作为014的唯一解

写出注册机

接下来根据算法写出注册机，为了减少计算量，我把长度固定为5

```
int CalcKey()
{
    srand(time(NULL));
    byte key[6] = { 0 };

    while (true)
    {
        byte K0 = rand() % 123;
        byte K1 = rand() % 123;
        byte K2 = rand() % 123;
        byte K3 = rand() % 123;
        byte K4 = rand() % 123;

        //限制随机数不出现无意义字符
        if ((K0>=65&&K0>=90)|| (K0 >= 97 && K0 >= 122)||( K0 >= 49 && K0 >=
57))
        {
            if ((K1 >= 65 && K1 >= 90) || (K1 >= 97 && K1 >= 122 )|| (K1 >= 49
&& K1 >= 57))
            {
                if ((K2 >= 65 && K2 >= 90 )|| (K2 >= 97 && K2 >= 122) || (K2 >=
49 && K2 >= 57))
                {
                    if ((K3 >= 65 && K3 >= 90) || (K3 >= 97 && K3 >= 122) ||
(K3 >= 49 && K3 >= 57))
                    {
                        if ((K4 >= 65 && K4 >= 90) ||( K4 >= 97 && K4 >= 122)
|| (K4 >= 49 && K4 >= 57))
                        {
                            //满足限制条件
                            if (K0 + K1 + K2 + K3 + K4 == K0 * 5)
                            {
```

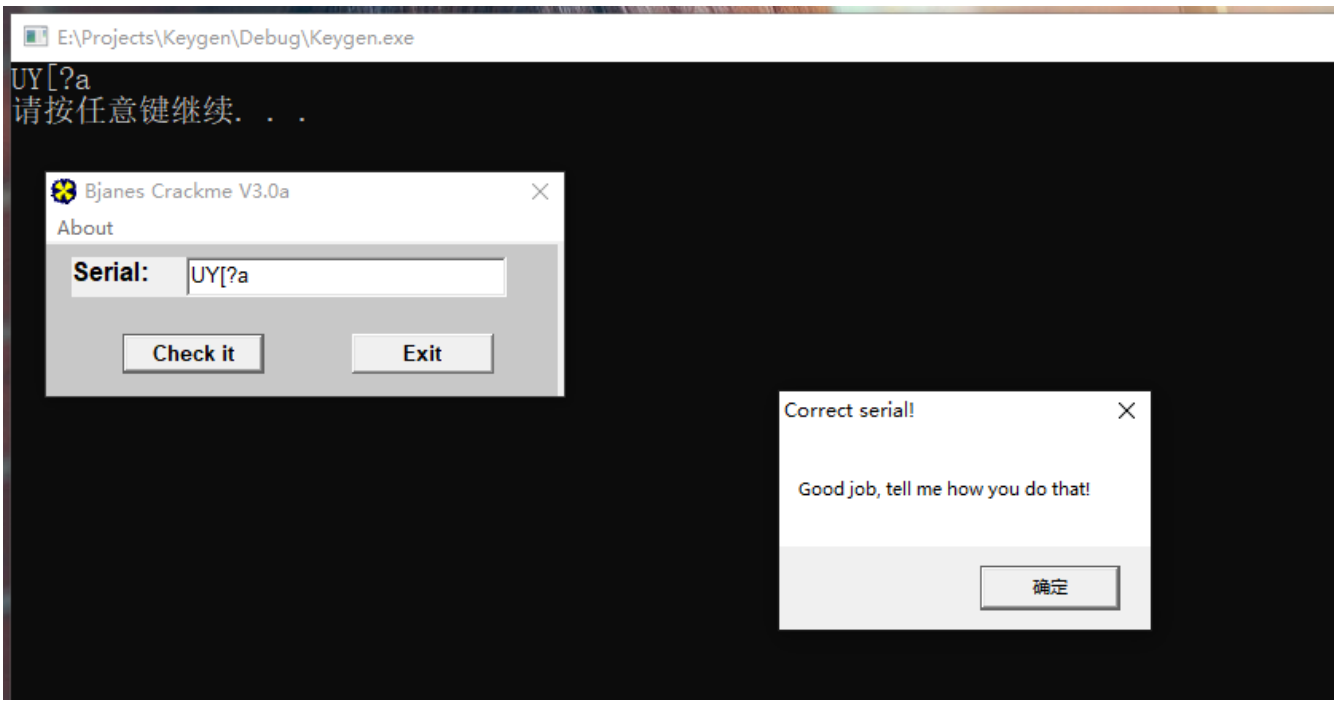
```
                                key[0] = K0;
                                key[1] = K1;
                                key[2] = K2;
                                key[3] = K3;
                                key[4] = K4;
                                key[5] = 0;
                                break;
                            }
                        }
                    }
                }
            }
        }


    }

    printf("%s\n", key);

    return 0;
}
```

输入注册机的结果，提示成功，这个Crackme就完成了



需要相关文件的可以到我的Github下载：https://github.com/TonyChen56/160-Crackme