【软件名称】：CyberBlade.2.exe

【软件大小】：61.0 KB

【下载地址】：https://github.com/TonyChen56/160-Crackme

【加壳方式】：未加壳

【保护方式】：Name/Serial

【编译语言】：VB P-Code

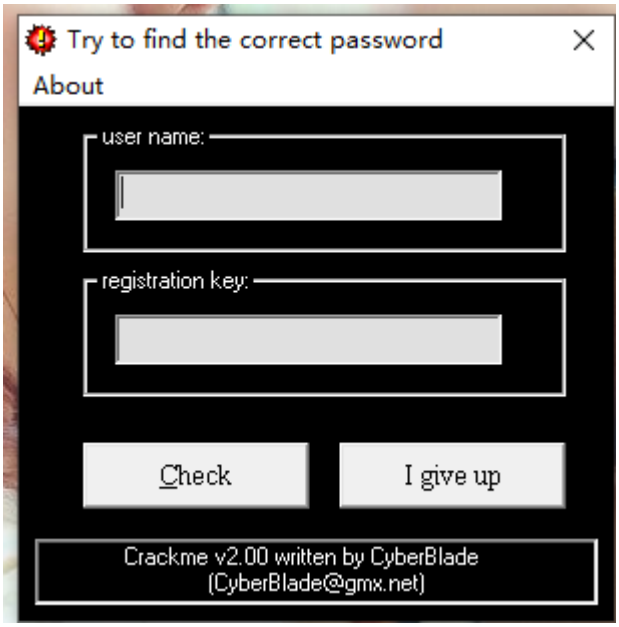
【调试环境】：W10 x64

【使用工具】 OD，VBExplorer，VB Decompiler

【破解日期】：2019-5-1

【破解目的】：学习分析P-Code类型的程序，理解P-Code虚拟机的解释过程

【目标程序】：



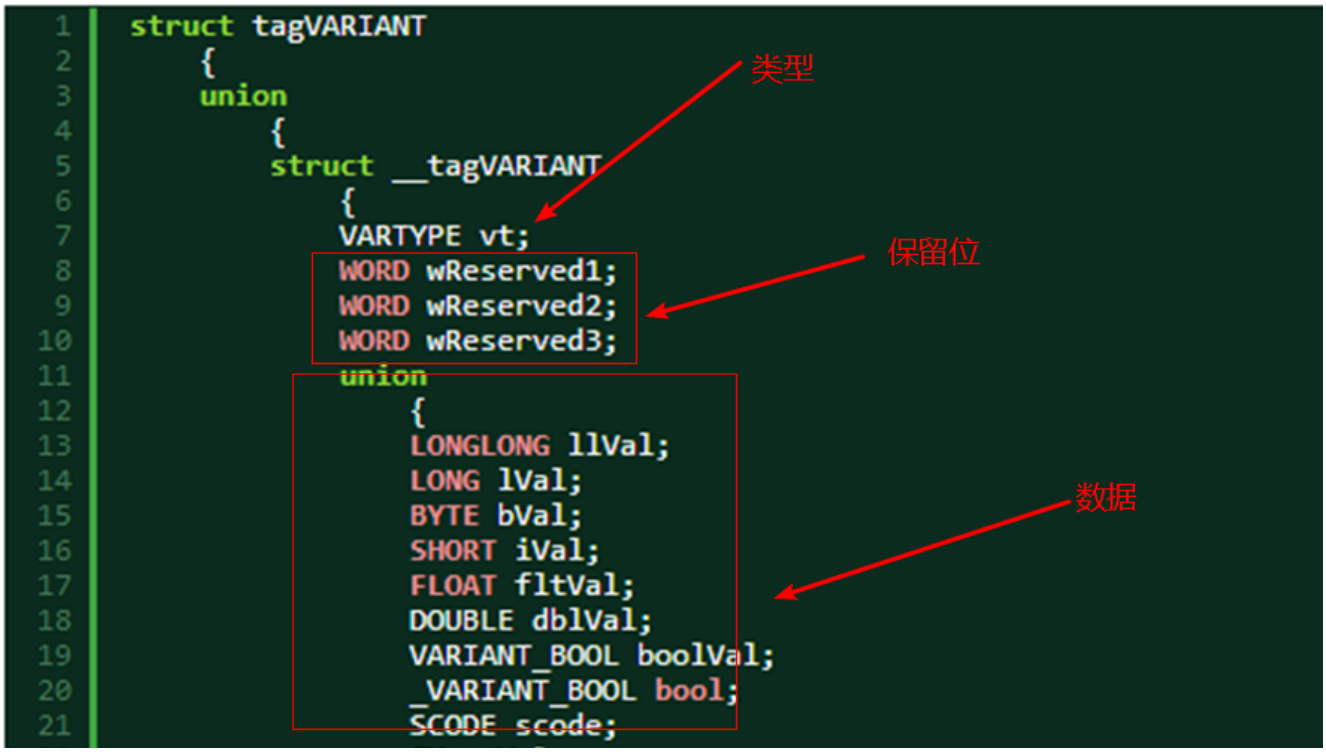
这一次的目标程序的这个Crackme，是160个Crackme里面的第38个。运行时需要 Visual Basic 5.0 运行库支持。这个Crackme的分析，用到了三个工具，每个工具都有各自的用途：

- OD：用于跟踪P-Code伪指令的具体细节及在静态分析过程中无法查看的数据
- VBExplorer 用于查看P-Code伪指令及注释
- VB Decompiler 用于静态查看反汇编的伪代码，减少OD跟踪伪指令的工作量

VB的变量类型

想要分析这个Crackme，首先需要了解VB变量类型在内存中的存储方式。

VB中的variant类型属于一种结构体,该结构体的前两个字节表示变量的类型,后面有3个WORD是保留的,接下来才是其真正的值，如下图：

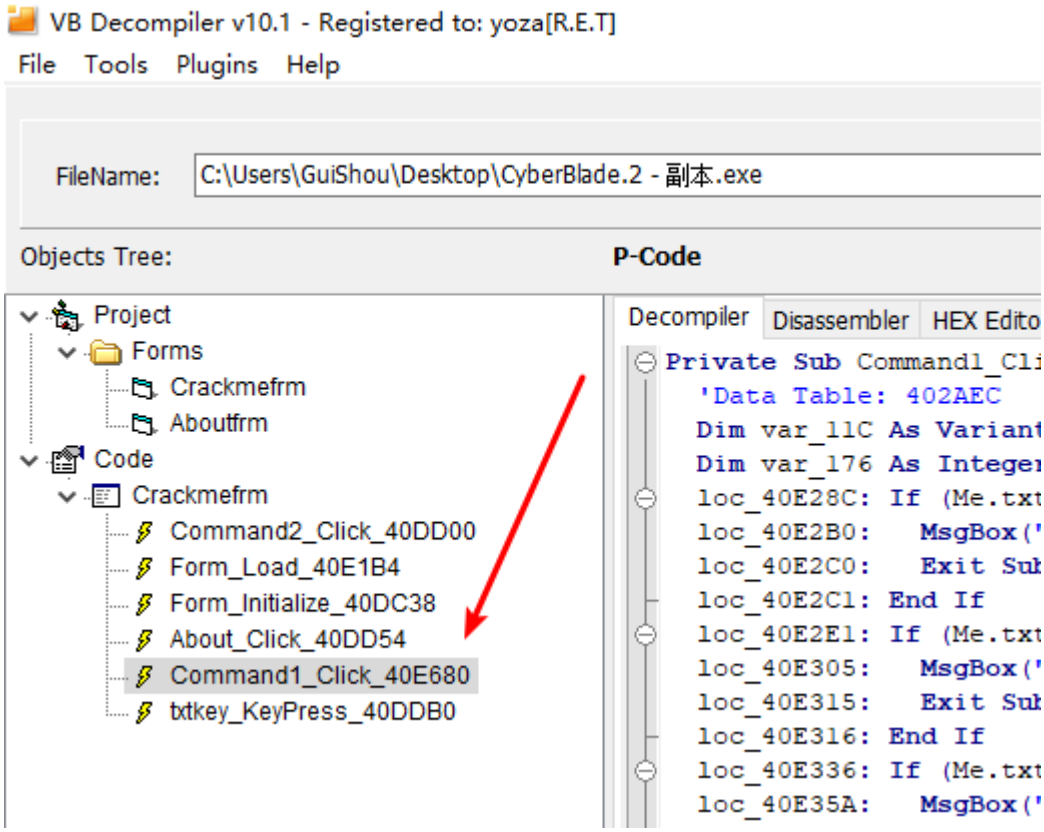


也就是说VB变量中真正的数据是存储在首地址+8的位置处，下图显示了VB的所有的变量类型及含义

Data Type	VB Constant	Value
Empty (*uninitialised*)	vbEmpty	0
Null (*no valid data*)	vbNull	1
Integer	vbInteger	2
Long integer	vbLong	3
Single-precision floating-point number	vbSingle	4
Double-precision floating-point number	vbDouble	5
Currency value	vbCurrency	6
Date value	vbDate	7
String	vbString	8
Object	vbObject	9
Error value	vbError	10
Boolean value	vbBoolean	11
Variant (*used only with arrays of variants*)	vbVariant	12
Byte value	vbByte	17
User-Defined Types (*UDT*)	vbUserDefinedType	36
Array	vbArray	8192

实战分析

首先用 VB Decompiler反编译目标程序，找到Check按钮的点击事件，分析整个点击事件的校验过程



这个程序的校验过程分为五个部分，下面讲解每一个部分的校验过程

第一部分 基础校验

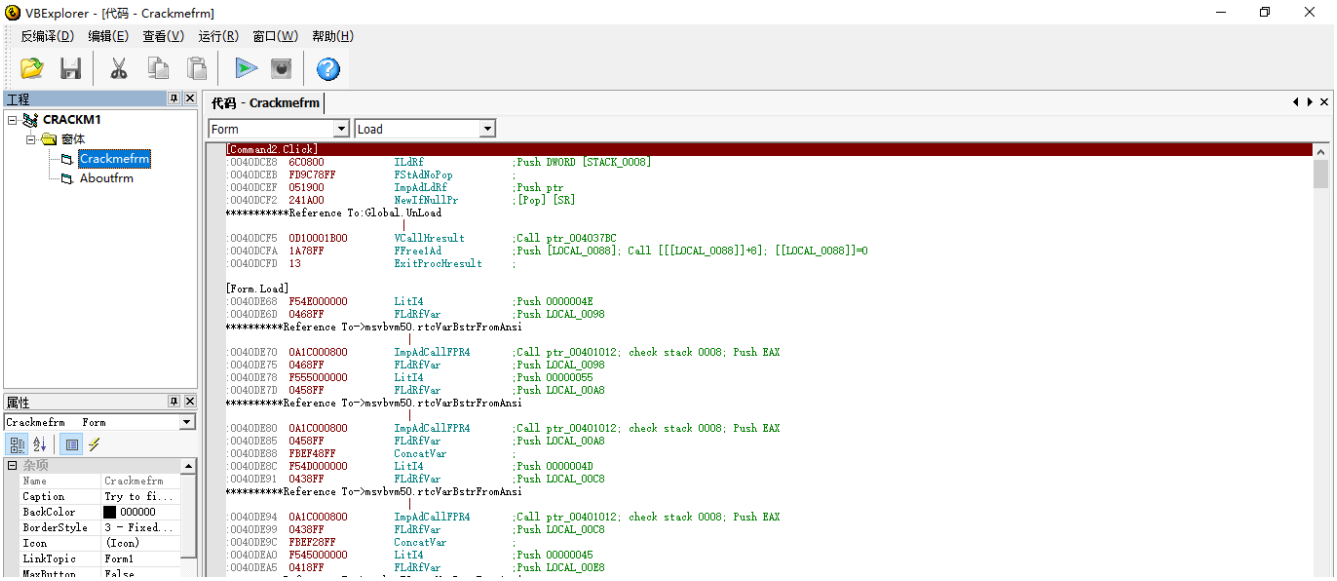
```
Decompiler  Disassembler  HEX Editor
Private Sub Command1_Click() '40E680
  'Data Table: 402AEC
  Dim var_11C As Variant
  Dim var_176 As Integer
  loc_40E28C: If (Me.txtname.Text = vbNullString) Then
  loc_40E2B0:   MsgBox("You have to enter you name first.", &H40, "Error", var_FC, var_11C)
  loc_40E2C0:   Exit Sub
  loc_40E2C1: End If
  loc_40E2E1: If (Me.txtkey.Text = vbNullString) Then
  loc_40E305:   MsgBox("You have to enter a key first.", &H40, "Error", var_FC, var_11C)
  loc_40E315:   Exit Sub
  loc_40E316: End If
  loc_40E336: If (Me.txtkey.Text = vbNullString) Then
  loc_40E35A:   MsgBox("You have to enter at least 5 chars.", &H40, "Error", var_FC, var_11C)
  loc_40E36A:   Exit Sub
  loc_40E36B: End If
```

首先根据静态分析的结果可以看到，该程序首先会校验用户名和序列号是否为空，然后判断序列号长度是否小于5个字节，否则提示错误。即使看不懂反汇编后的VB代码，也可以通过字符串知道整个过程。

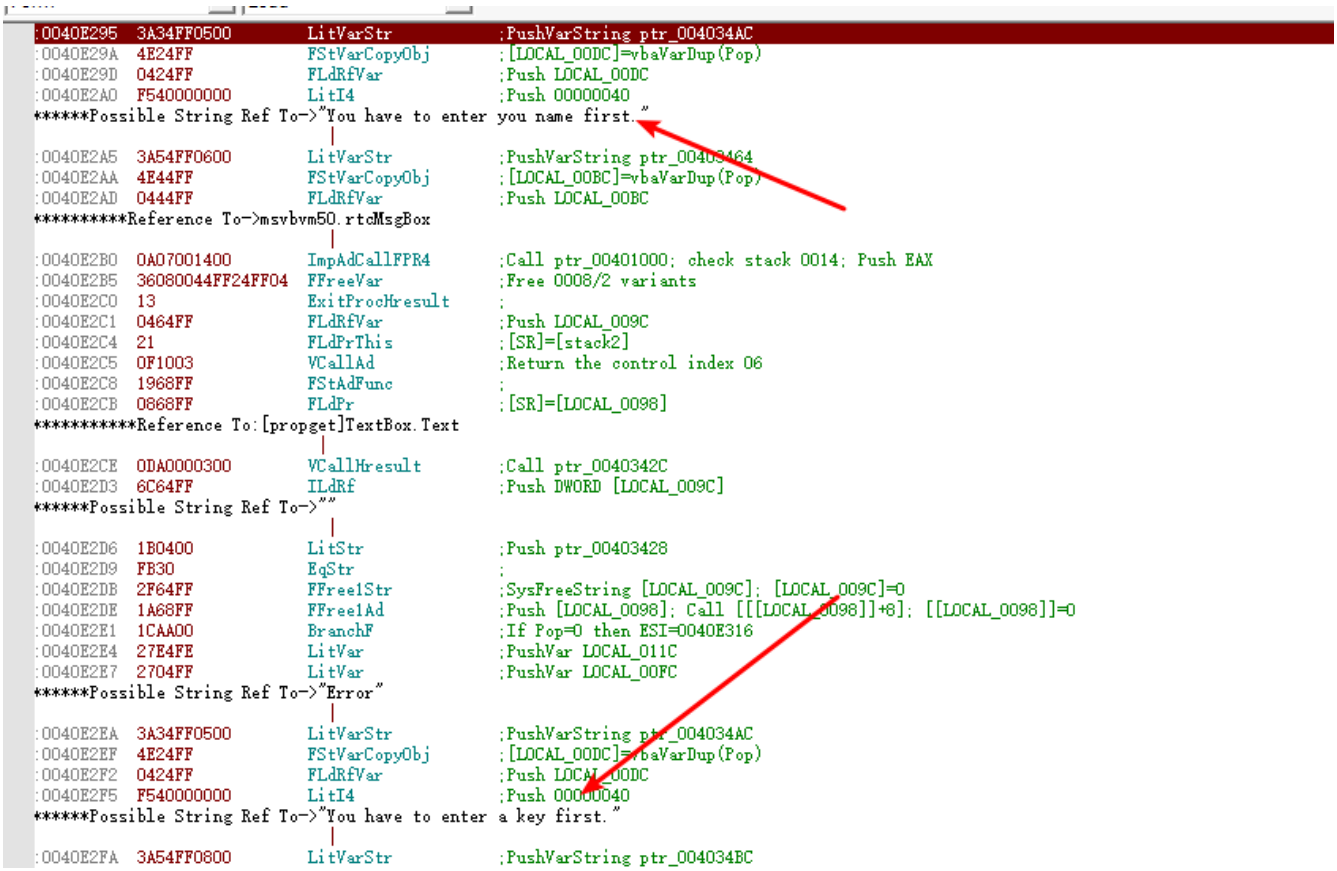
第二部分 根据用户名计算结果

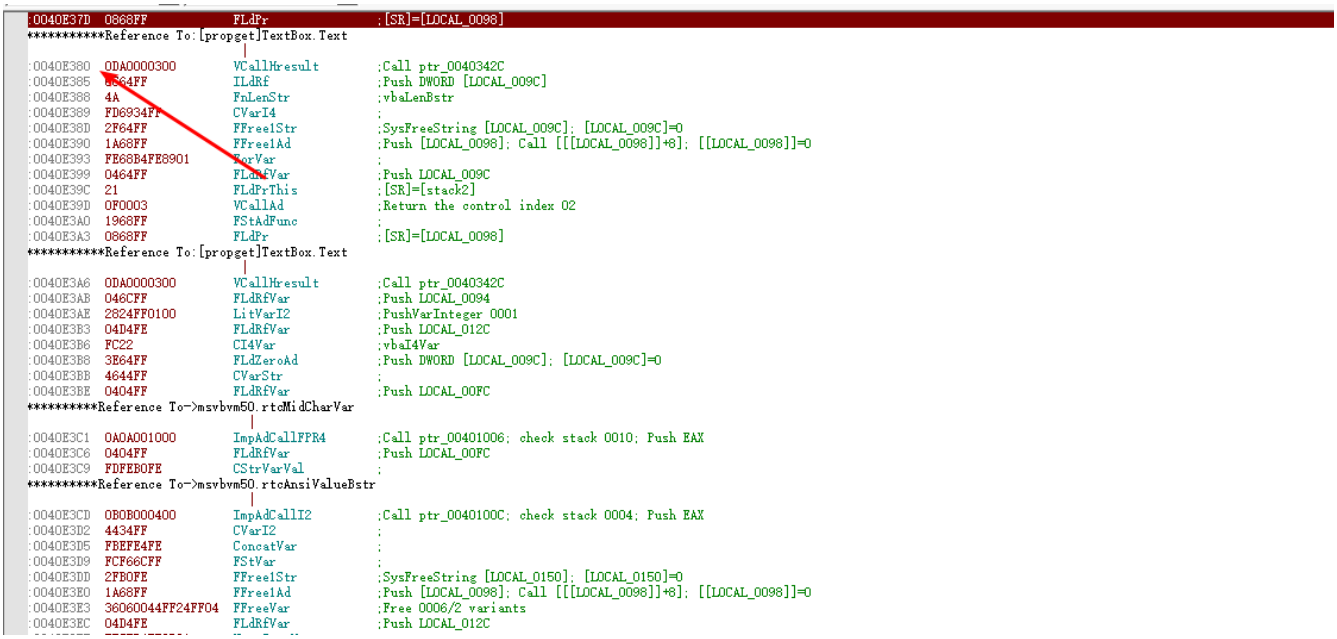
```
loc_40E393: For var_14C = 1 To CVar(Len(Me.txtname.Text)): var_12C = var_14C 'Variant
loc_40E3C1:   var_FC = Mid(CVar(Me.txtname.Text), CLng(var_12C), 1)
loc_40E3D5:   var_11C = var_94 & CVar(Asc(CStr(var_FC)))
loc_40E3D9:   var_94 = var_11C 'Variant
loc_40E3EF: Next var_14C 'Variant
loc_40E3F5: ' Referenced from: 40E422
```

第二部分的校验过程看的就不那么清晰了，需要利用OD动态跟踪每一个伪指令的具体操作流程。



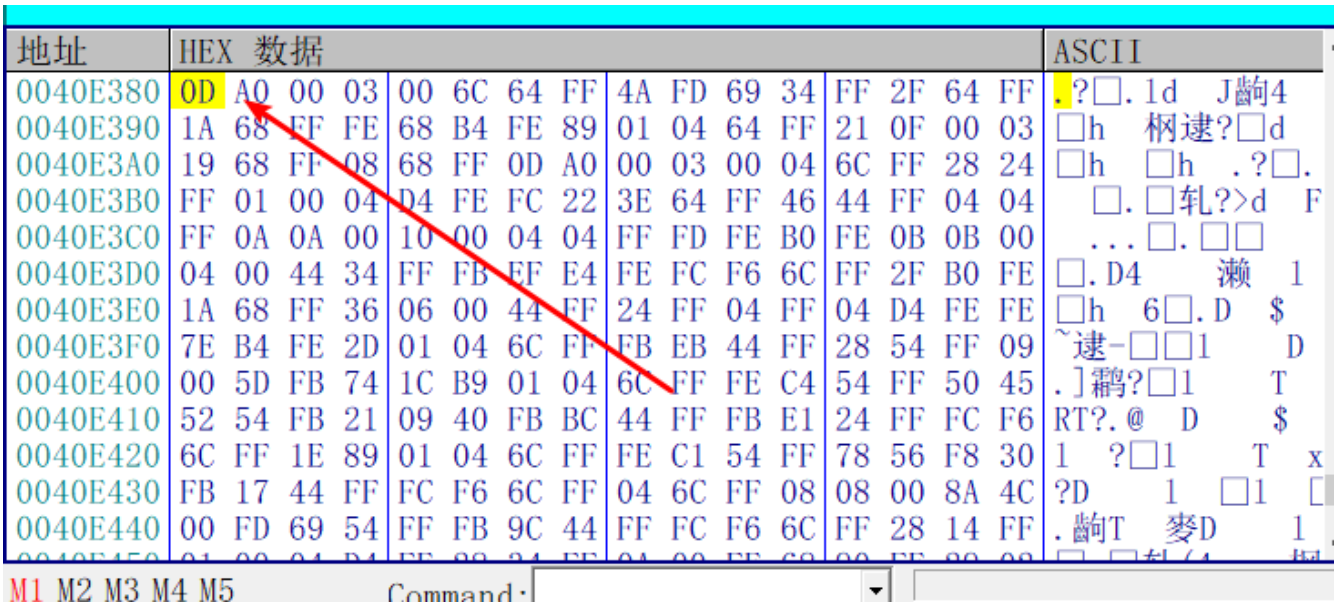
用VBExplorer对目标程序进行反编译，一直往下拉，根据字符串直接忽略第一部分的基础校验，来到0040E380的位置



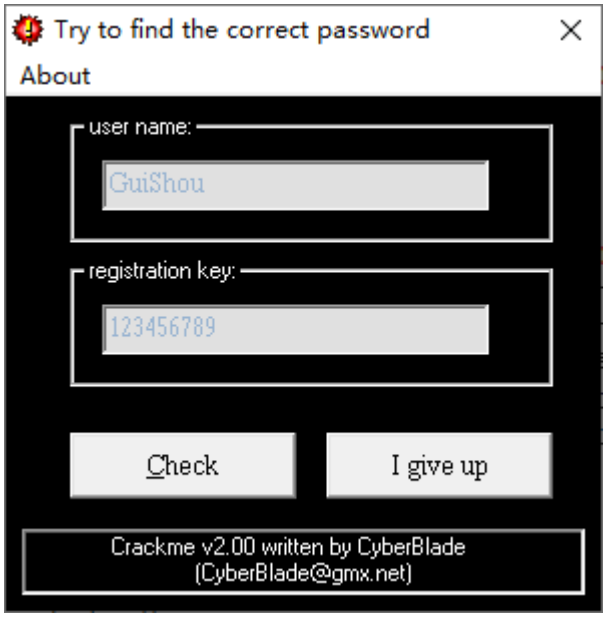


我们可以看到第一个被执行的伪指令是0040E380处的0D，接下来将程序载入OD，数据窗口跟随->0040E380

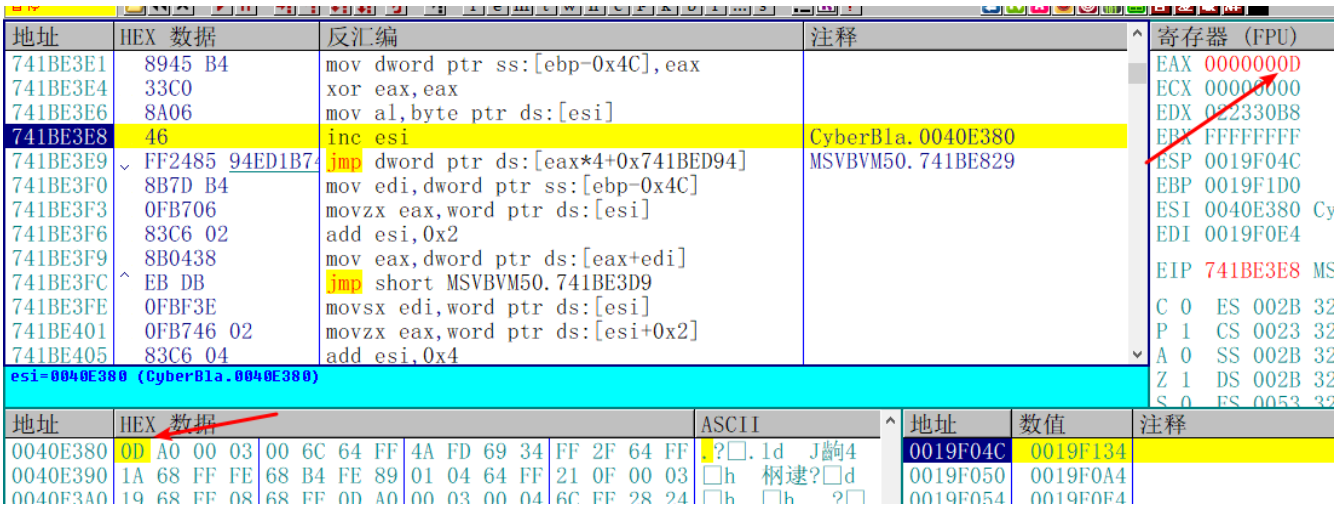
然后给第一个字节0D下内存访问断点，F9运行



然后随便输入一个用户名和序列号，点击Check

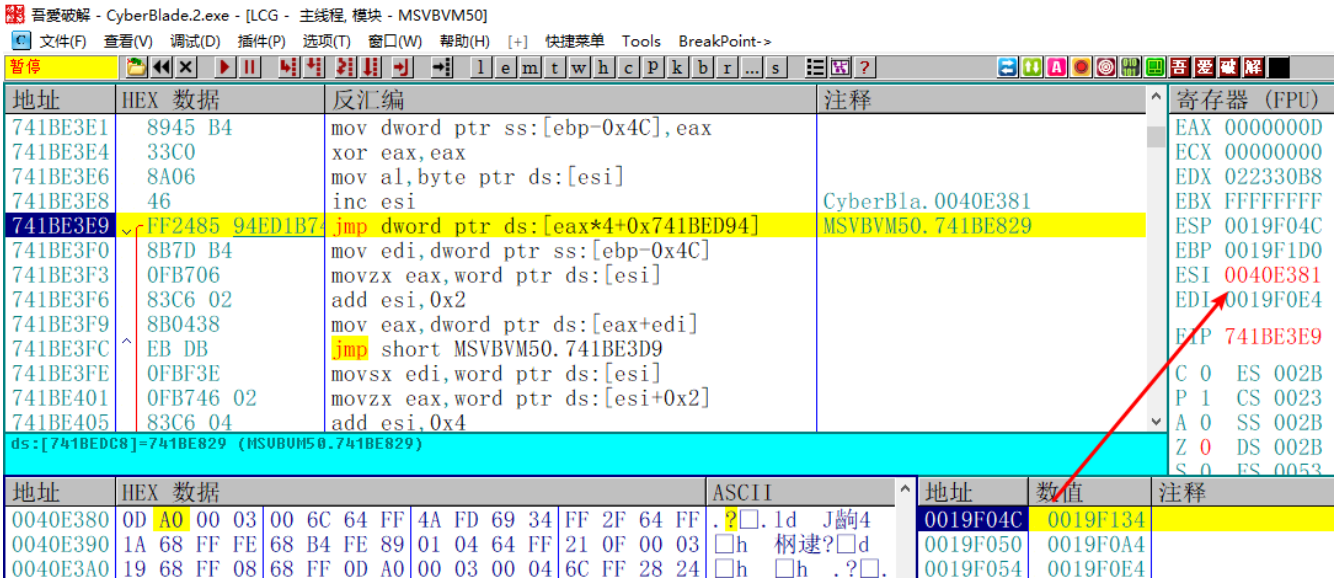


程序首先会读取一个字节的操作码到AL，

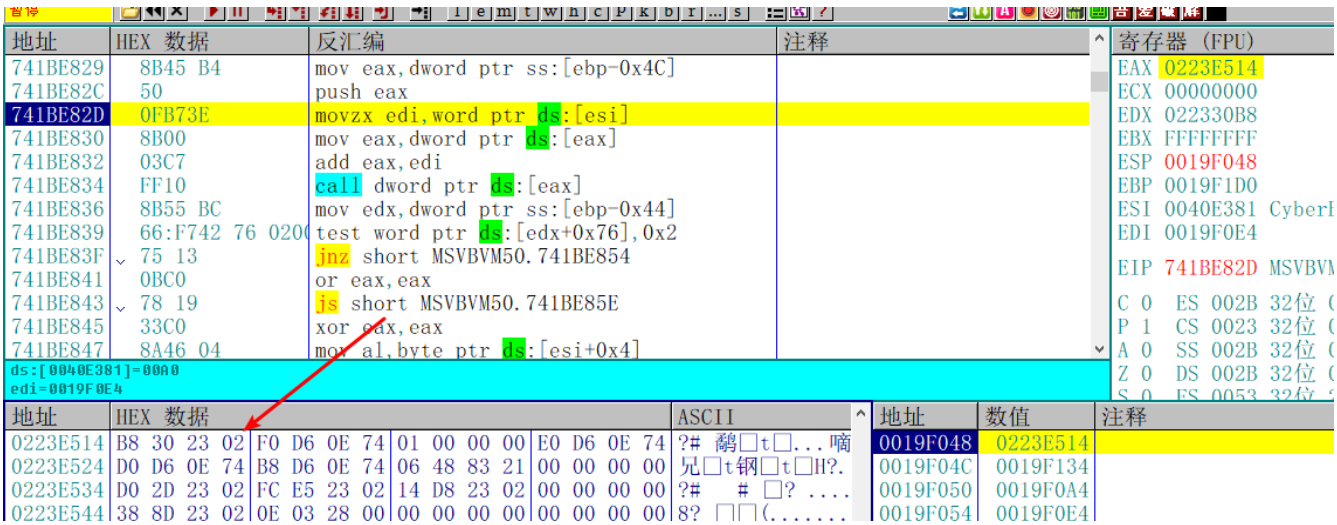



```
:0040E380  0DA0000300        VCallHresult       ;Call ptr_00403420
:0040E385  6C64FF            ILdRf              ;Push DWORD [LOCAL_009C]
:0040E388  4A                FnLenStr           ;vbaLenBstr
:0040E389  FD6934FF          CVarI4             ;
:0040E38D  2F64FF            FFreeStr           ;SysFreeString [LOCAL_009C]; [LOCAL_009C]=0
:0040E390  1A68FF            FFreeAd            ;Push [LOCAL_0098]; Call [[[LOCAL_0098]]+8]; [[LOCAL_
:0040E393  FE68B4FE8901      ForVar             ;
:0040E399  0464FF            FLdRfVar           ;Push LOCAL_009C
:0040E39C  21                FLdPrThis          ;[SR]=[stack2]
:0040E39D  0F0003            VCallAd            ;Return the control index 02
:0040E3A0  1968FF            FStAdFunc          ;
:0040E3A3  0868FF            FLdPr              ;[SR]=[LOCAL_0098]
**********Reference To:[propget]TextBox.Text
:0040E3A6  0DA0000300        VCallHresult       ;Call ptr_00403420
```

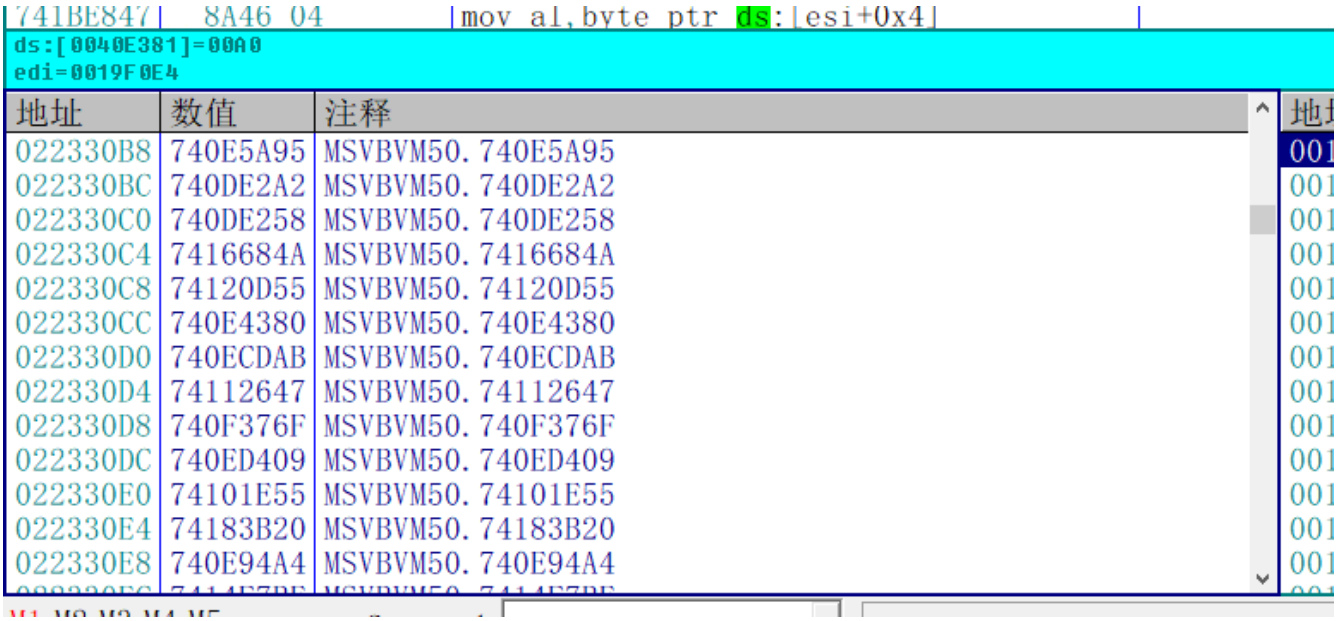
来看下VB Explorer中显示的操作码，后面的注释提示这是在调用一个函数，0D后面的是操作码的参数，接着esi自增1，指向操作码的参数



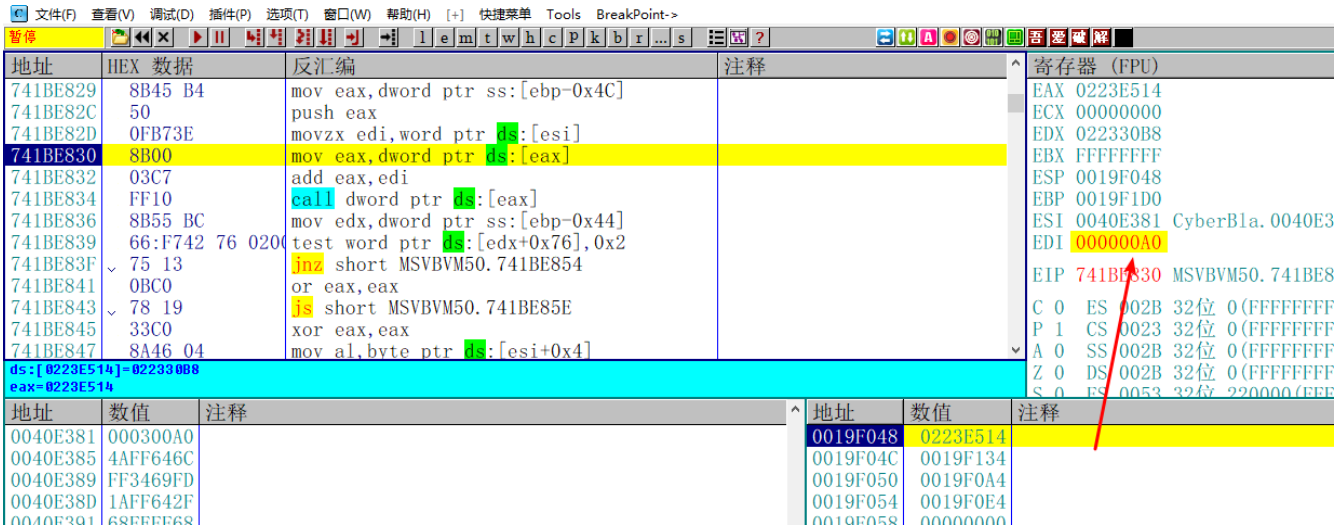
接着通过一个jmp跳转去执行操作码，0x741BED94是地址跳转表的首地址，eax保存下一条指令的操作码，由于每一个跳转地址是一个DWORD，所以用eax乘以4的值加上跳转表的基地址来索引下一条伪指令的解释单元，我们跟随这个jmp

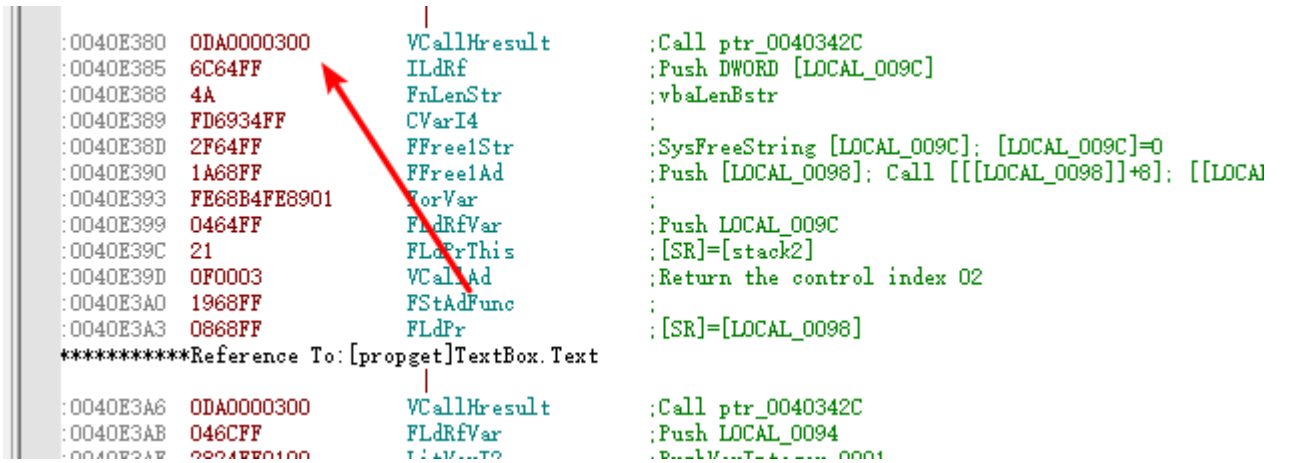


首先把[ebp-0x4C]赋值给eax，然后将eax压栈。我们需要知道eax的含义。数据窗口跟随之后，发现是一个指针，再次选中前四个字节，数据窗口跟随DWORD，然后将数据显示方式切换为长型->地址

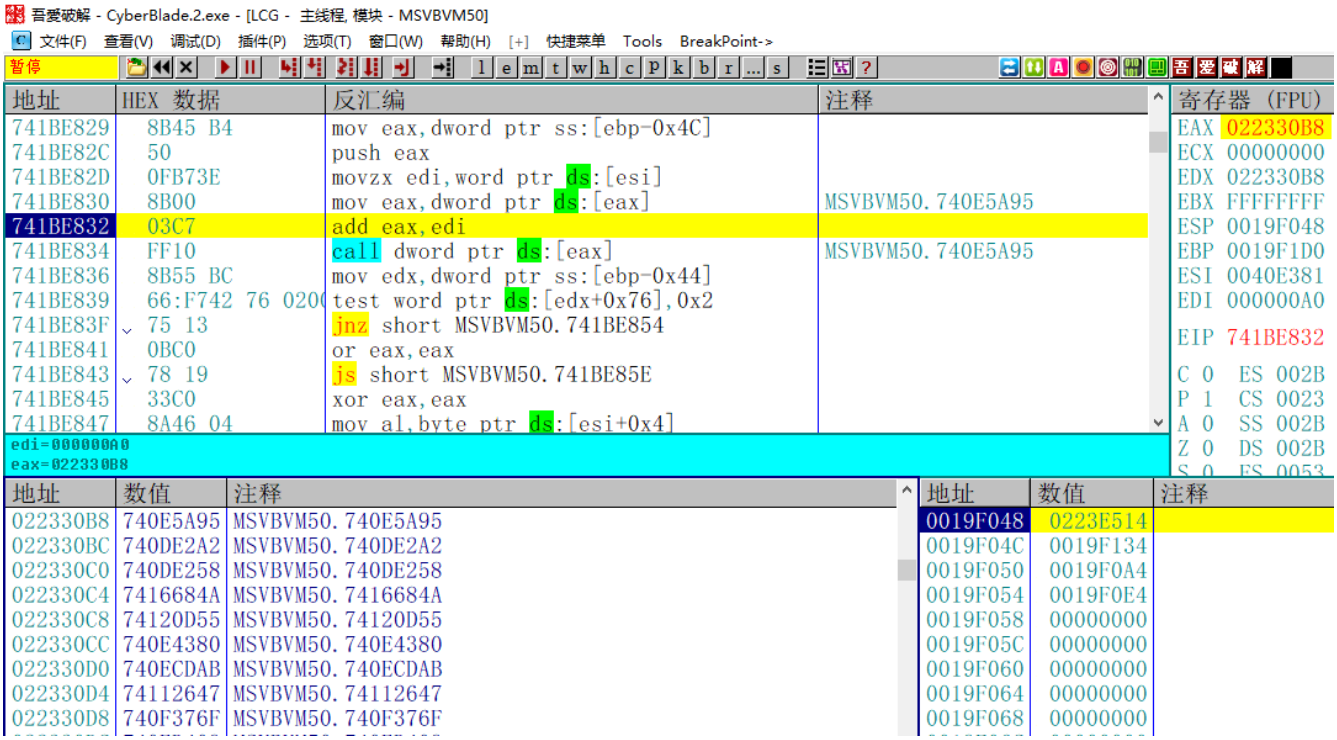


这其实是一个函数的跳转表，再接着把esi的内容A0赋值给edi，而esi始终执行的是操作码

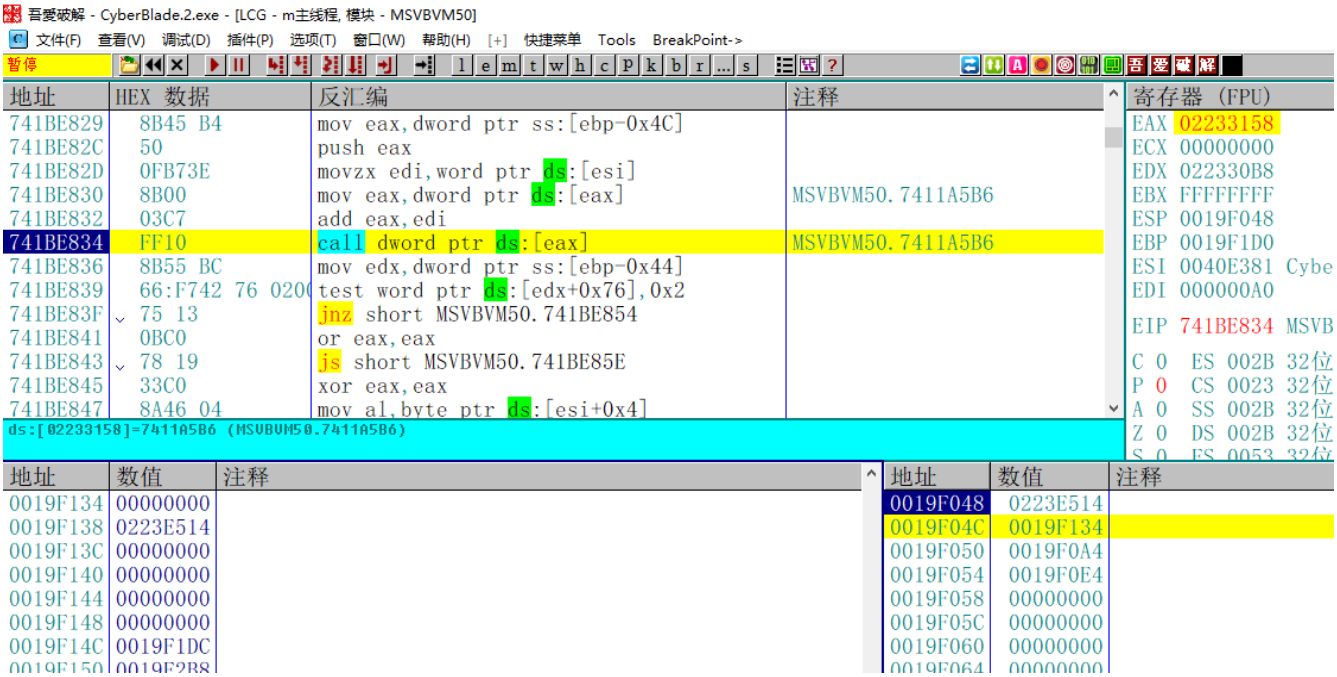




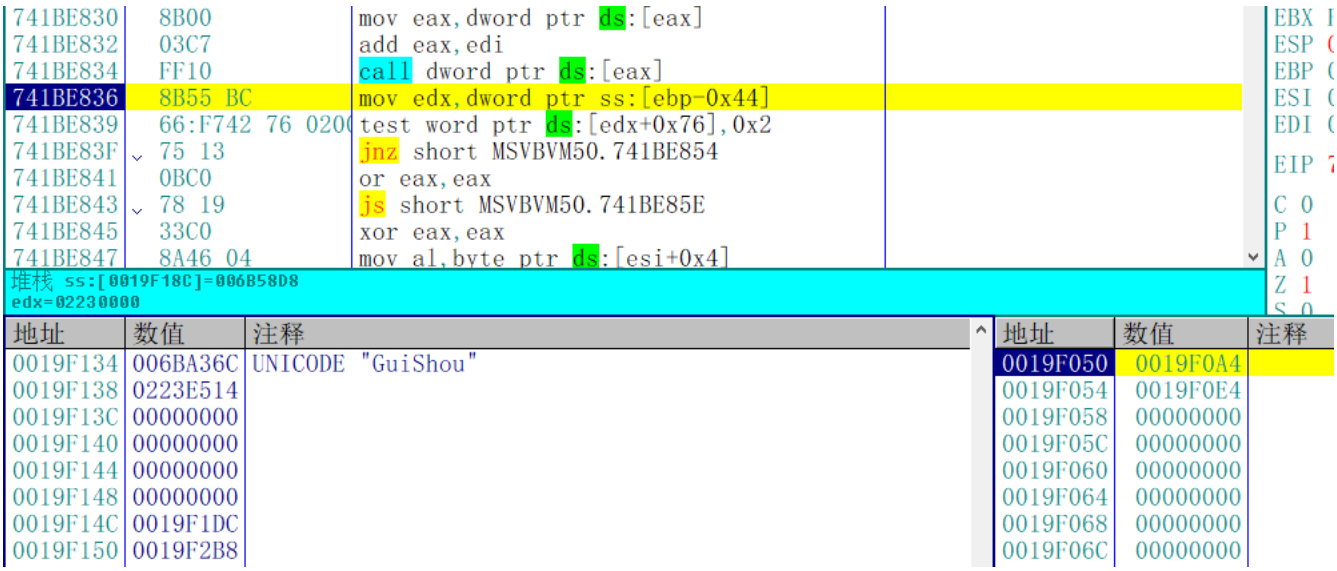
可以看到这一步实际上是在取操作码的参数A0了



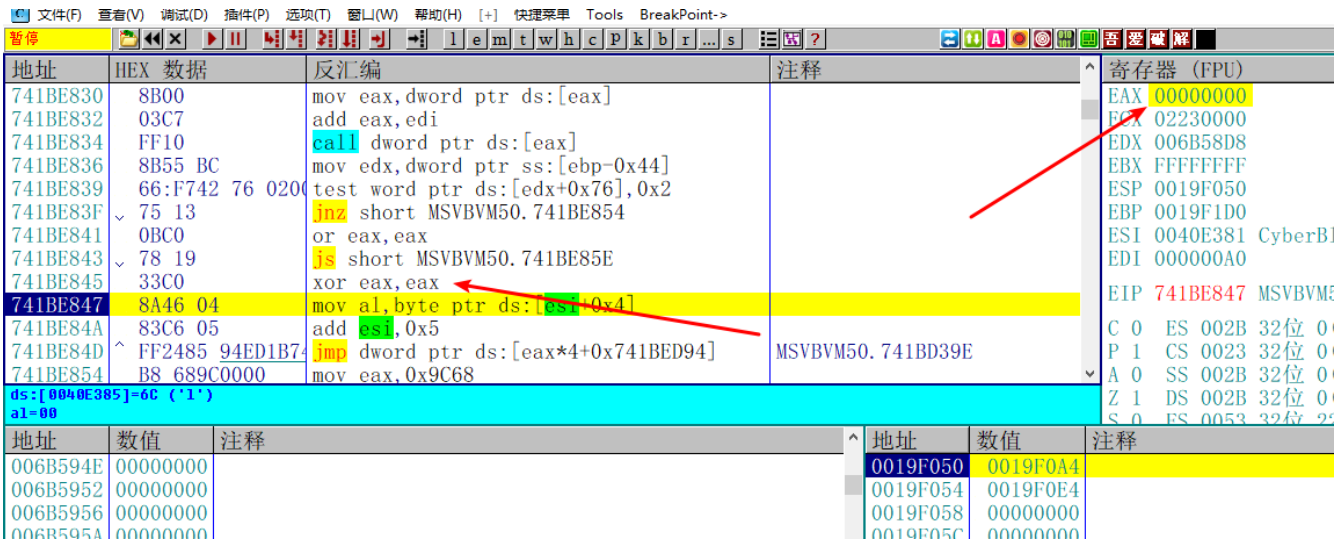
接着取出eax的内容，然后将eax加上edi，eax实际是跳转表的首地址，那么参数A0，就是跳转表的偏移



接着call eax，我们不需要跟进这个函数，只需要关注栈中的第二个参数0x19F134即可。直接步过这个函数

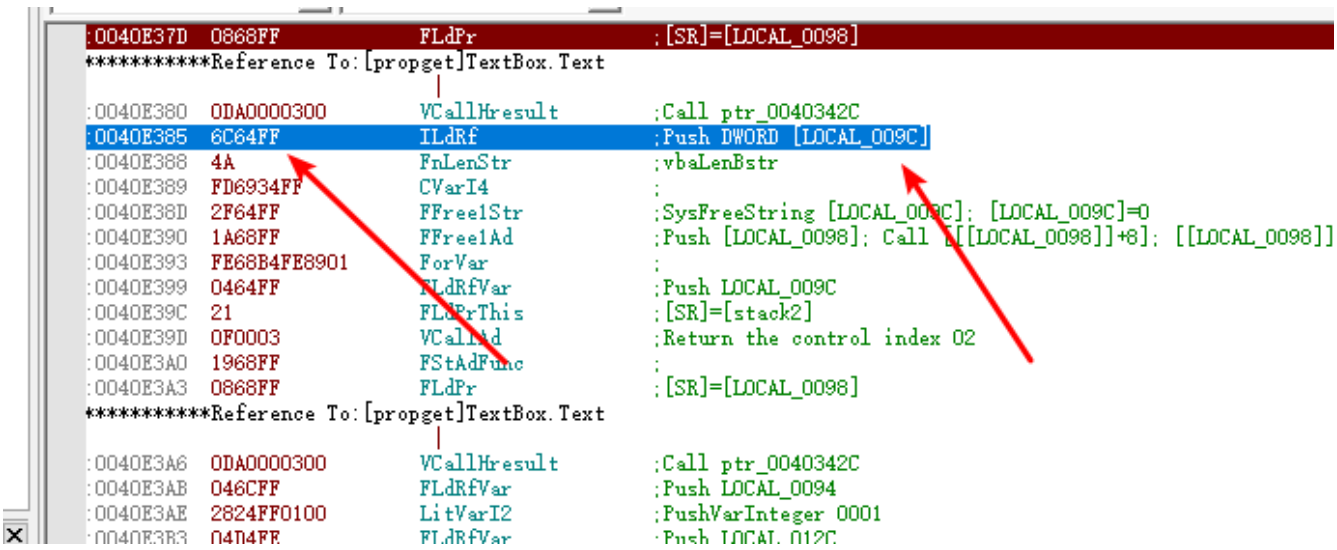


可以看到栈中的参数显示出了我们刚才输入的用户名，再接着单步到xor eax,eax的地址

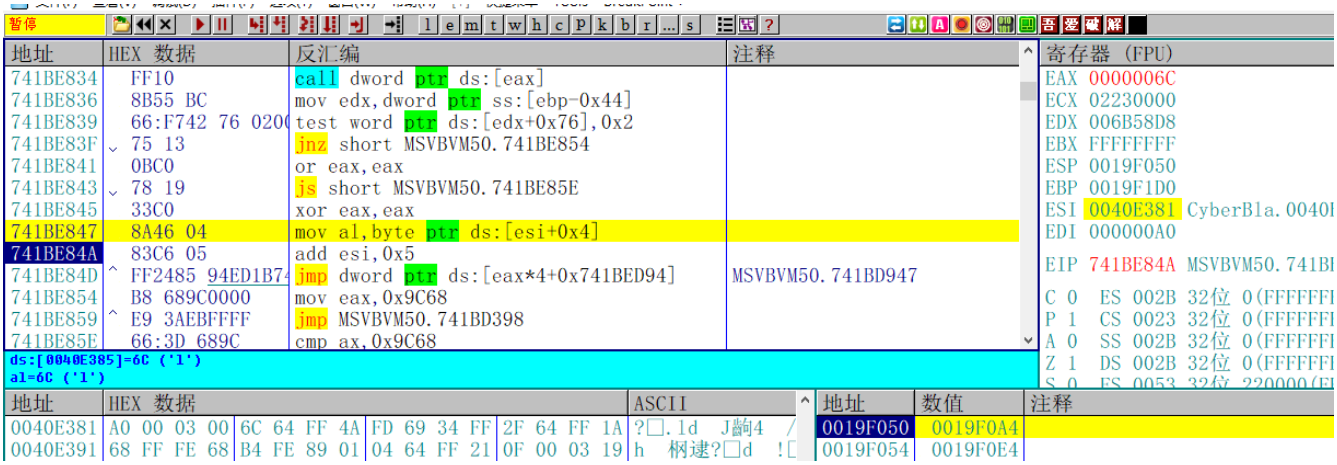


这里把eax的值清零了。也就是说第一条伪指令已经执行完成了。

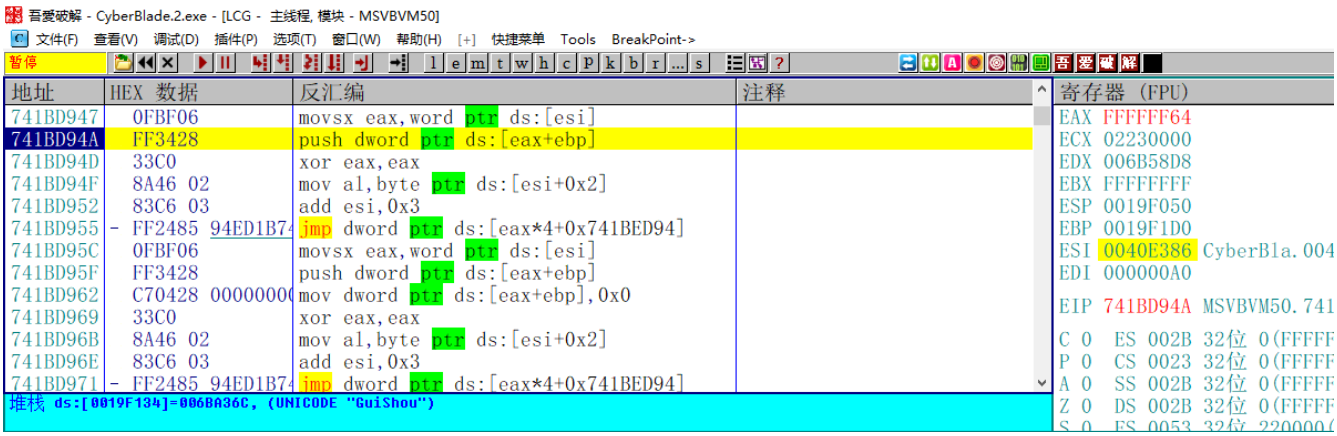
这个就是P-Code虚拟机的解释过程，其中esi始终指向要解释的伪指令，eax保存的是将要解释的伪指令。



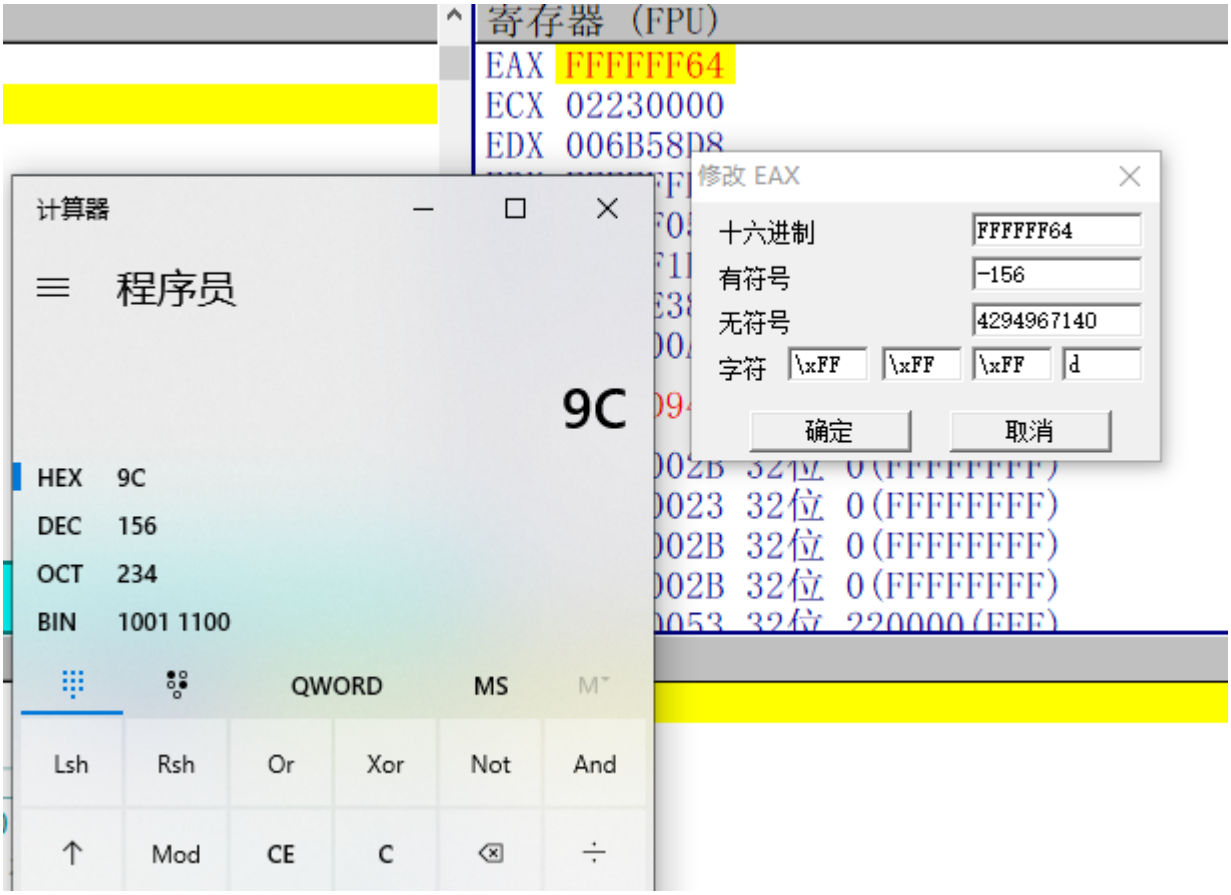
接着看下一条伪指令，6C是操作码，64FF是参数。后面的注释告诉我们这条伪指令是在将某个DWORD值入栈。继续跟踪



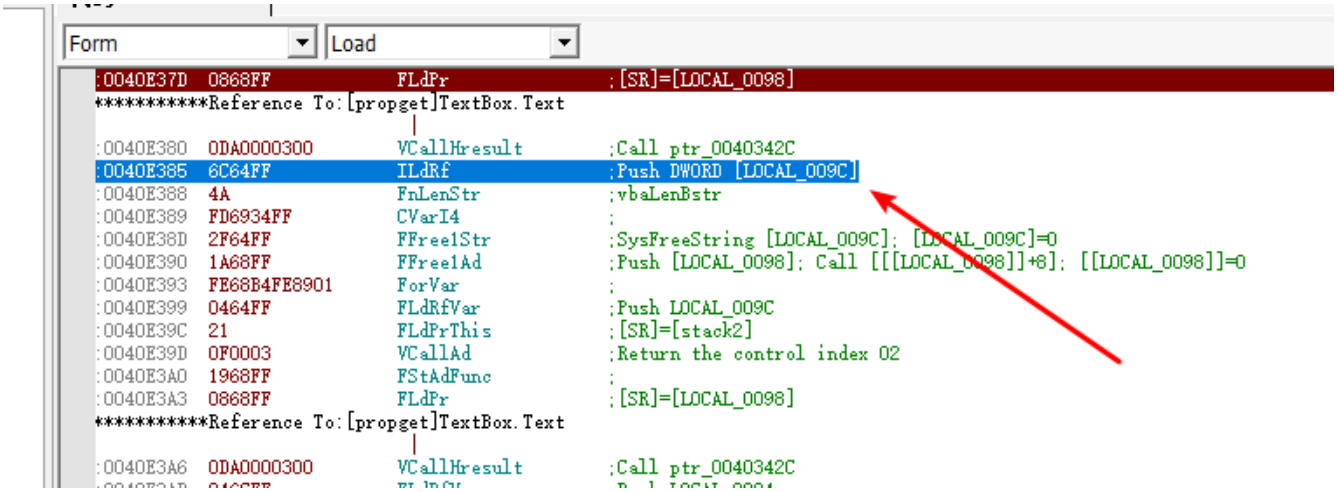
首先取出操作码6C，然后esi+5执向伪指令参数，接着跳转执行这条伪指令，直接跟进jmp

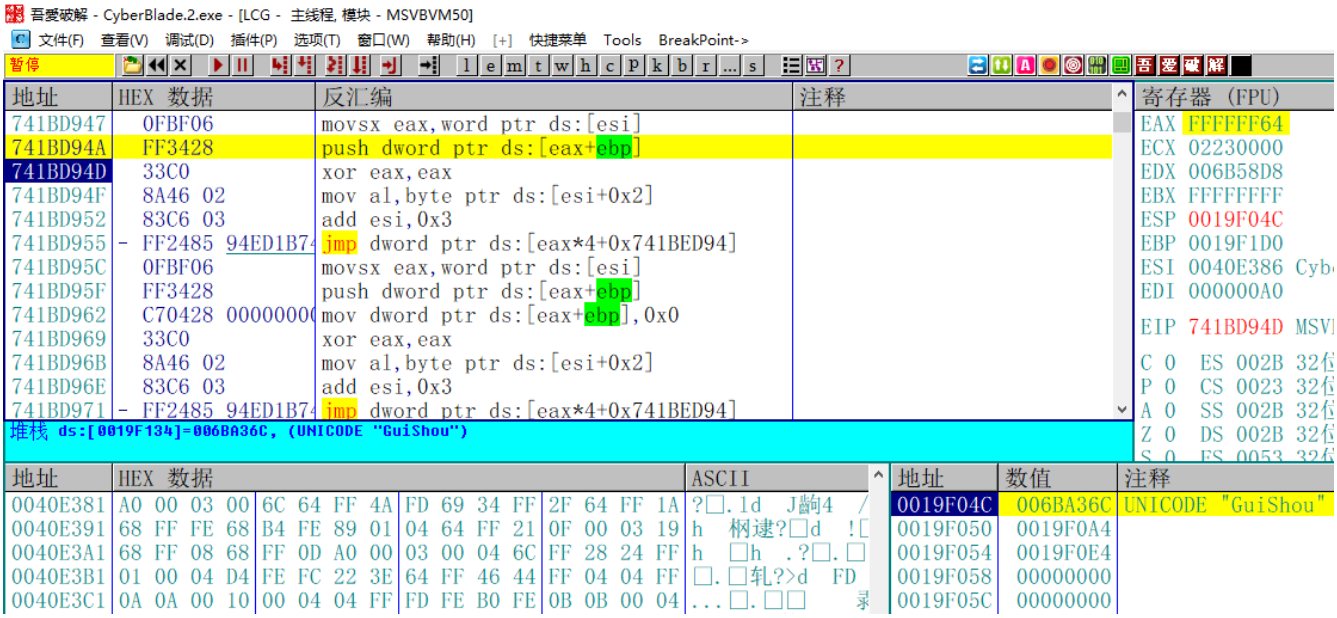


首先取出参数FF64放到eax中，FF64是一个负数，



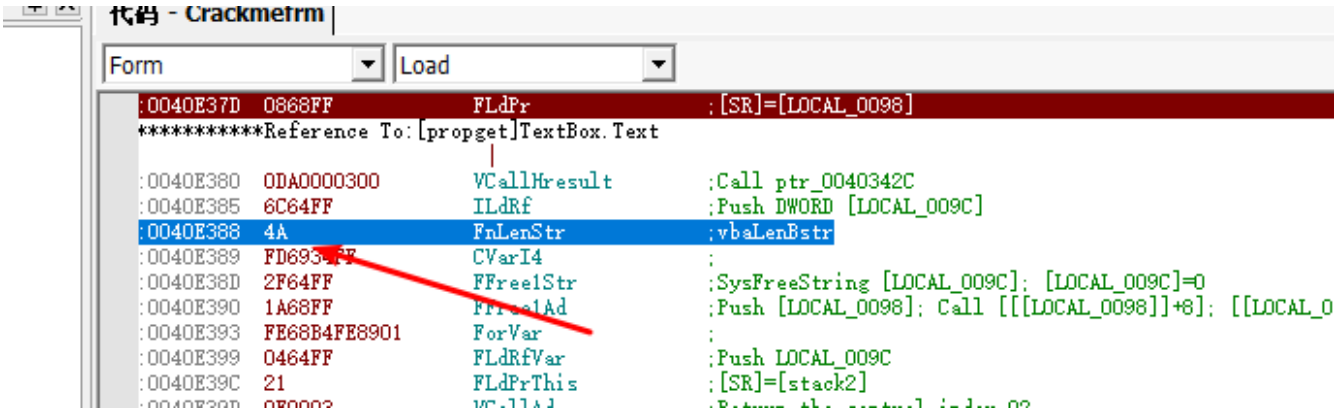
也就是十进制的9C，这也是为什么VBExplorer里会显示LOCAL_009C的原因，这个9C代表[ebp-9C]，是个局部变量



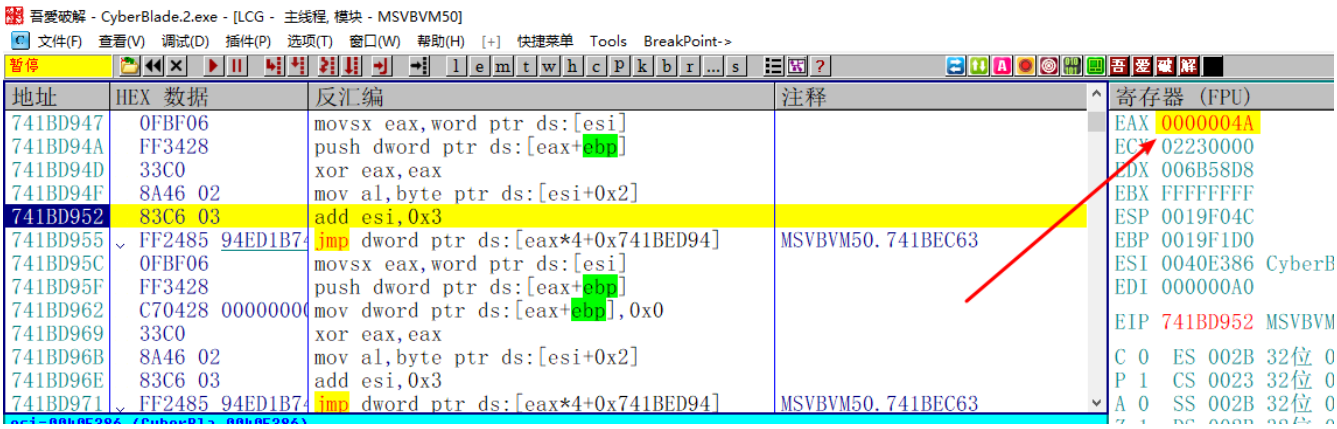


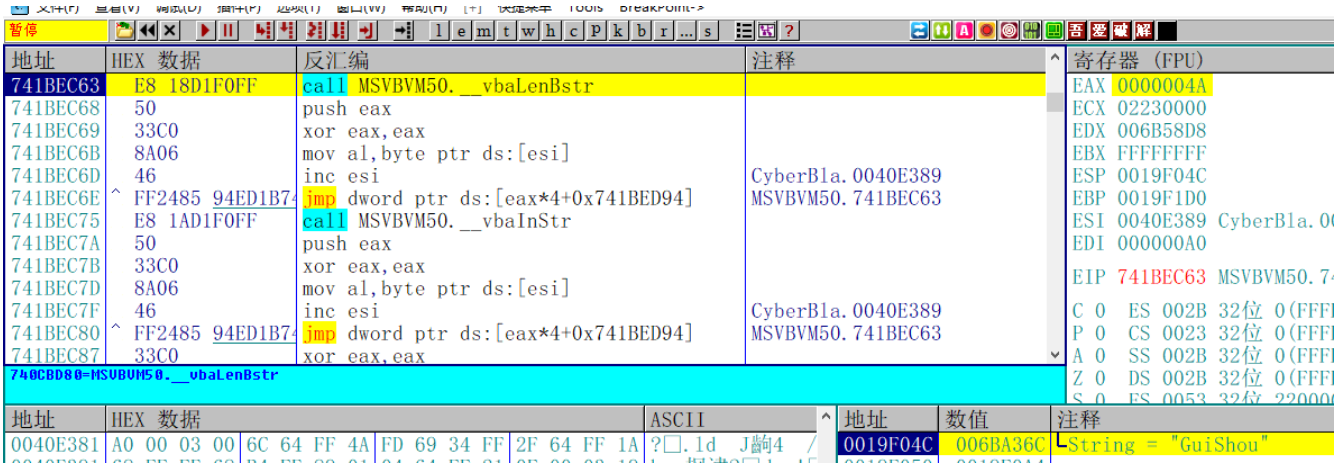
接着将[eax+ebp]入栈，压栈的是刚刚输入的用户名[eax]的值是-98，这里其实是将[ebp-98]局部变量压入栈，然后eax清零，表示这条伪指令结束。

继续看下一条，伪代码解释的求长度

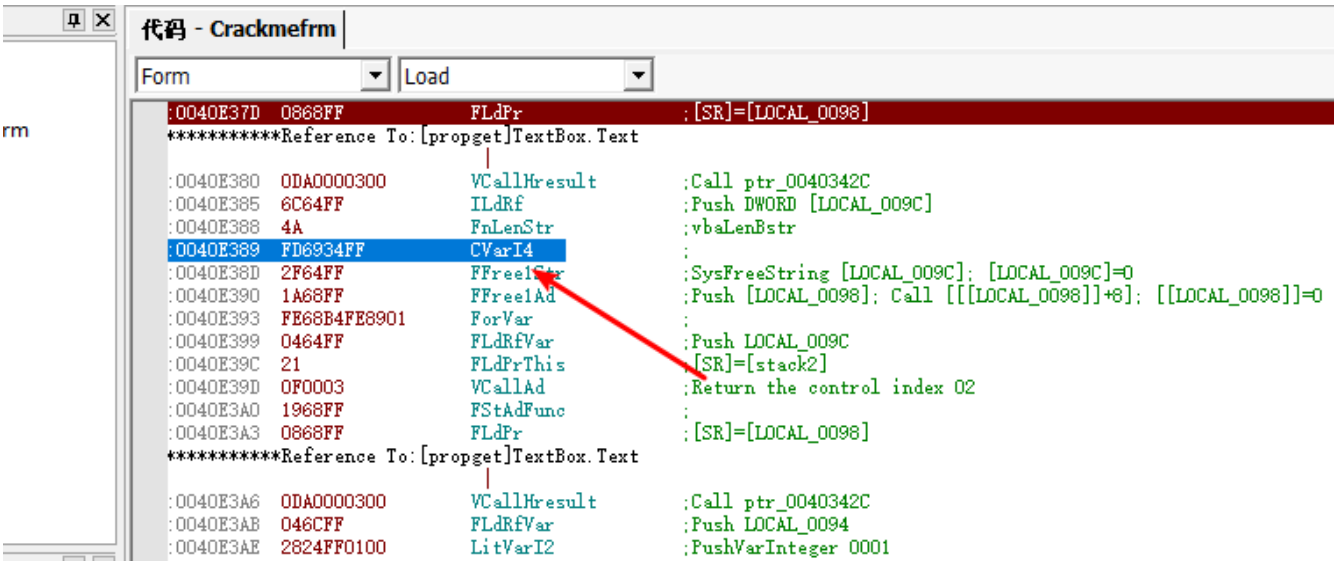


首先取出伪指令，然后直接跟进jmp

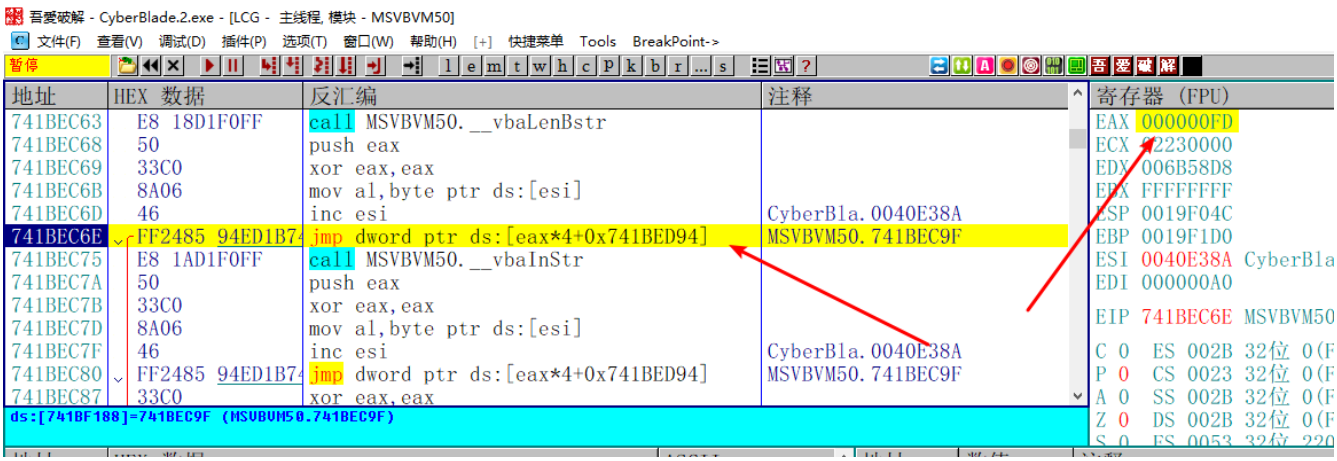




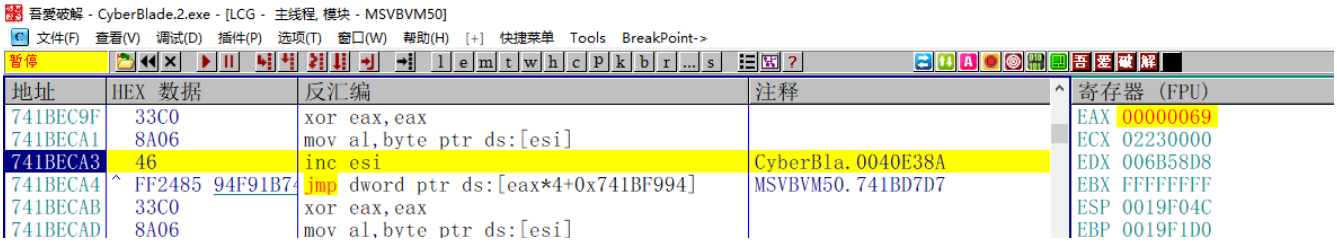
这里调用vbaLenBster，参数是之前输入的用户名，接着将用户名长度入栈后，eax清零，接着看下一条FD6934EF



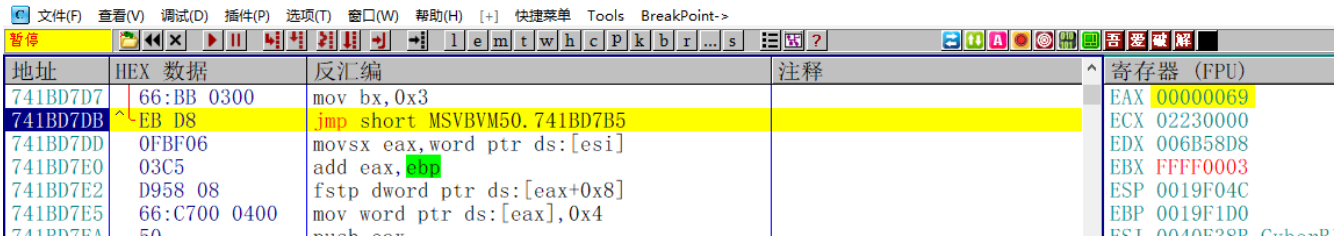
这条伪指令后面并没有给出解释，但是没有关系，我们可以根据伪指令的执行过程猜测指令含义，这里其实是一个双操作码的指令，



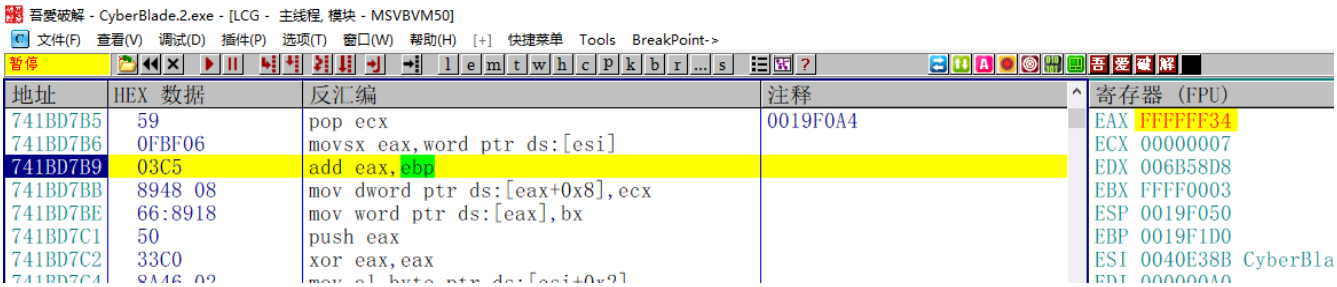
首先取出操作码FD，直接跟进jmp，什么都没有做，直接将eax清零。之后再次取出操作码69，继续跟进jmp



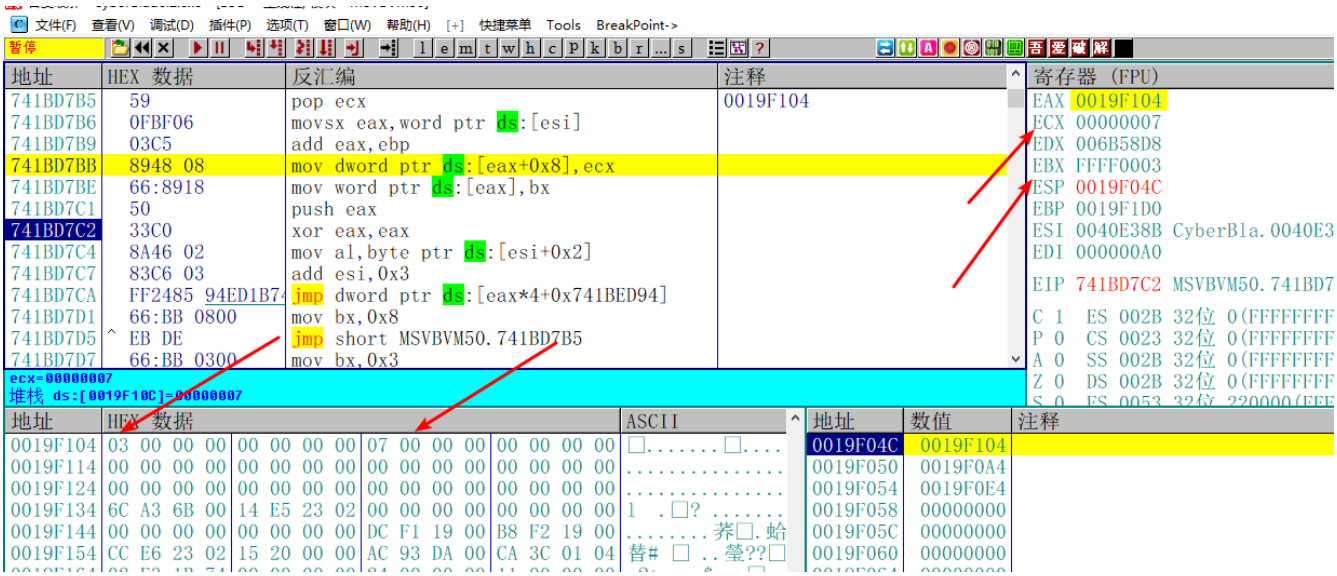
这里将bx赋值为0x3，然后跳转，继续跟进



接着取出参数FF34，FF34也是个负数，然后再将FF34加上ebp，代表这是一个局部变量

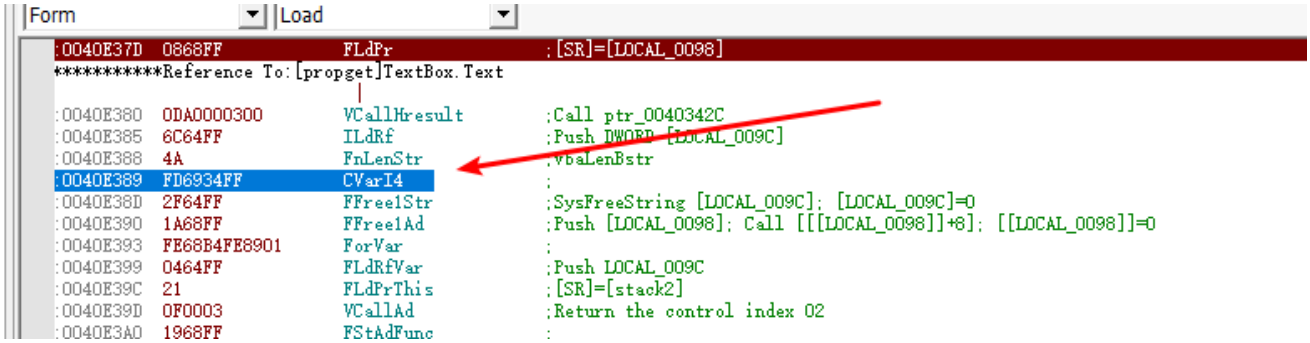


接下来将ecx赋值给[eax+0x8]的地址处，我们数据窗口跟随eax，然后将bx赋值给eax，赋值完成后eax值如下：

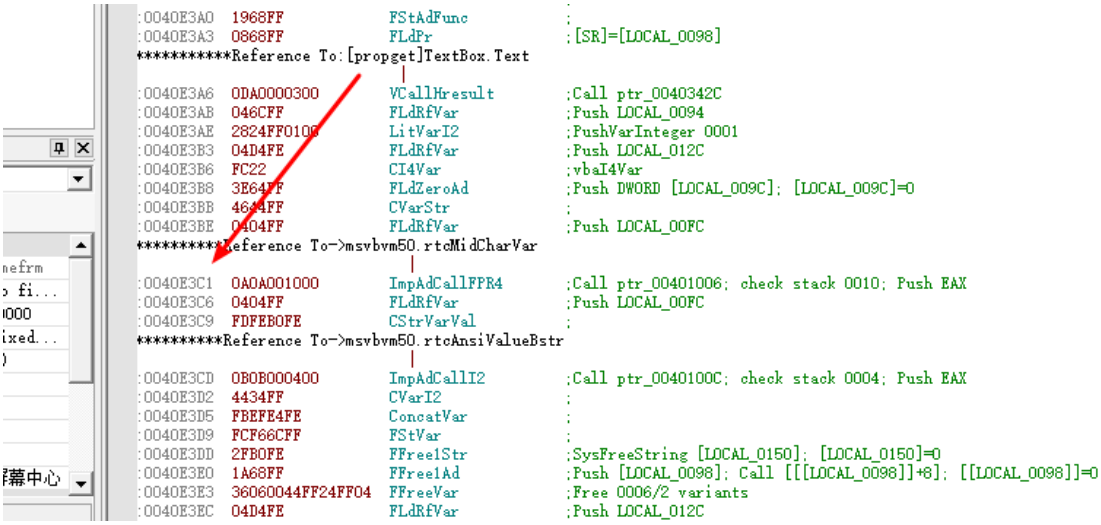


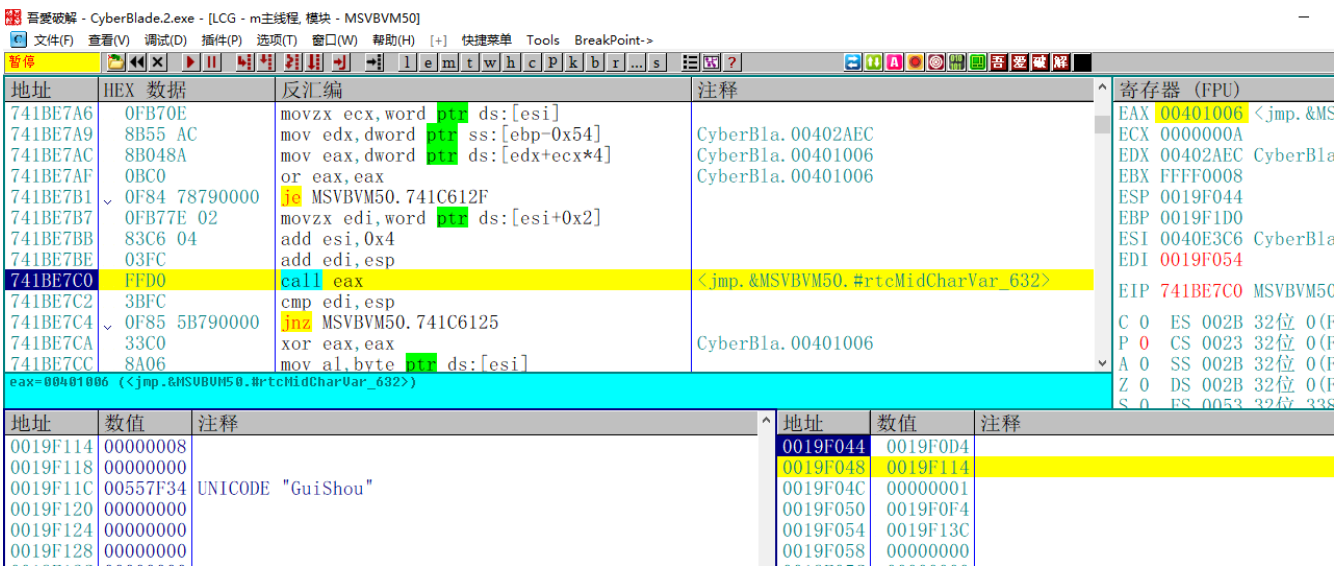
还记得VB的变量类型吗？前两个字节是变量类型，03代表是Long，中间是6个字节的保留位，首地址+8的位置才是真正的数值。

这个07是之前通过vbaLenBstr获取到的用户名长度，在这里转成了变量，并将变量首地址压栈。

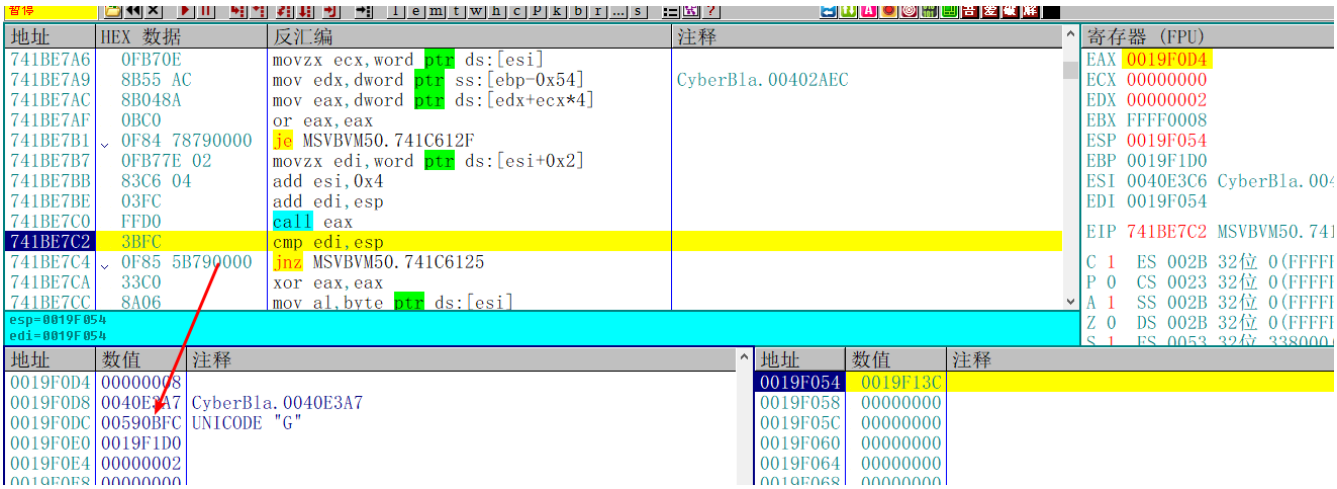


现在我们就能通过实际的跟踪结果来得出这条指令的含义了。就是将int值转成变量类型。由于整个跟踪过程实在是复杂，我这里只贴出算法的关键部分

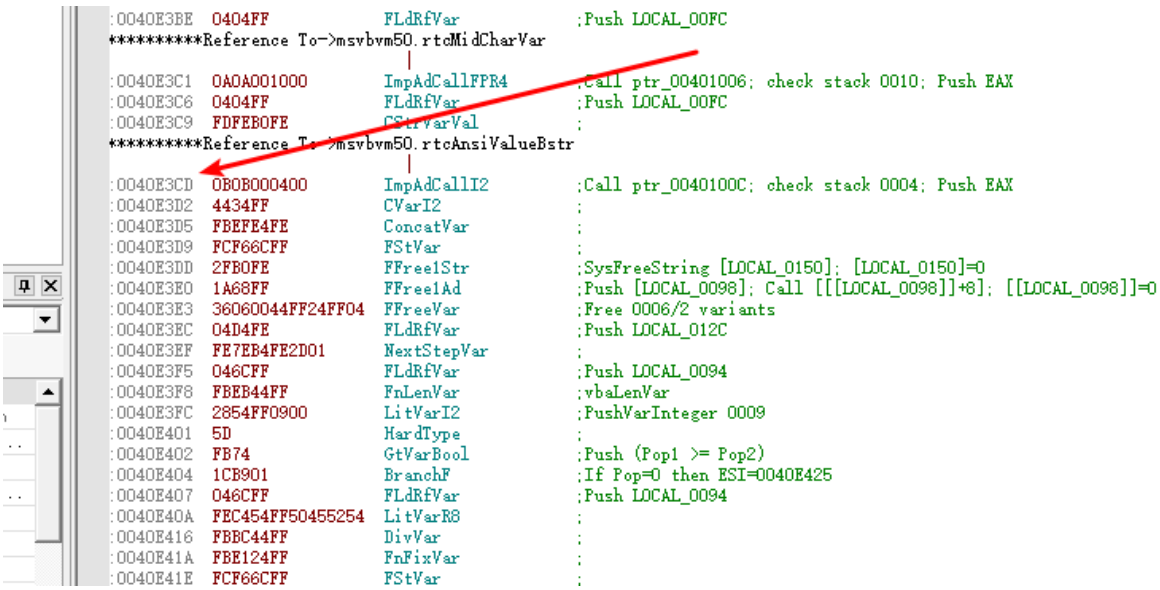


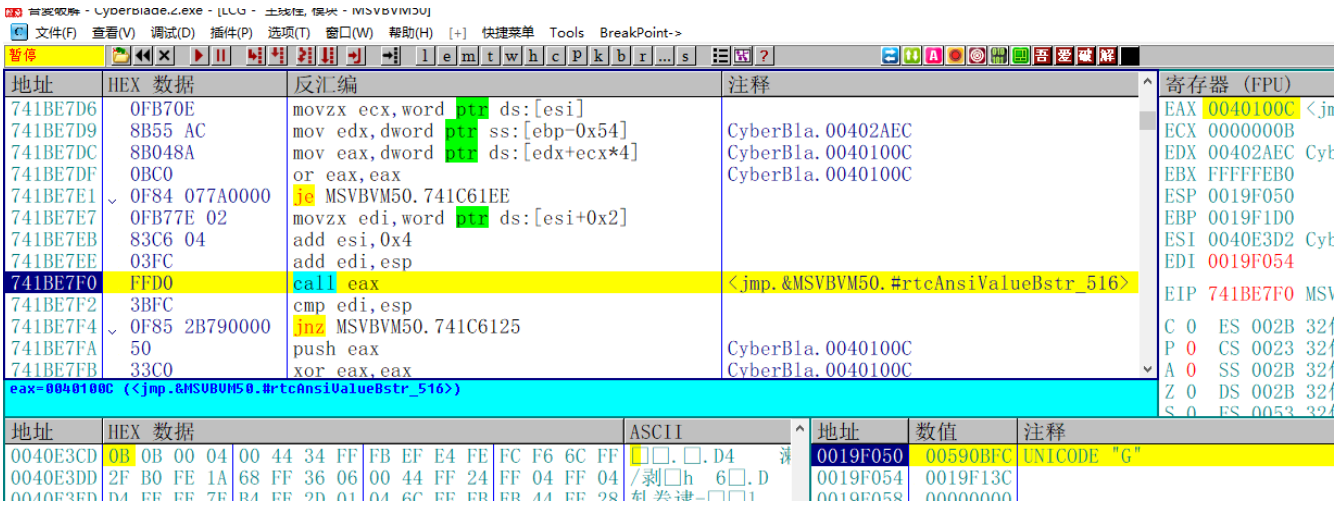


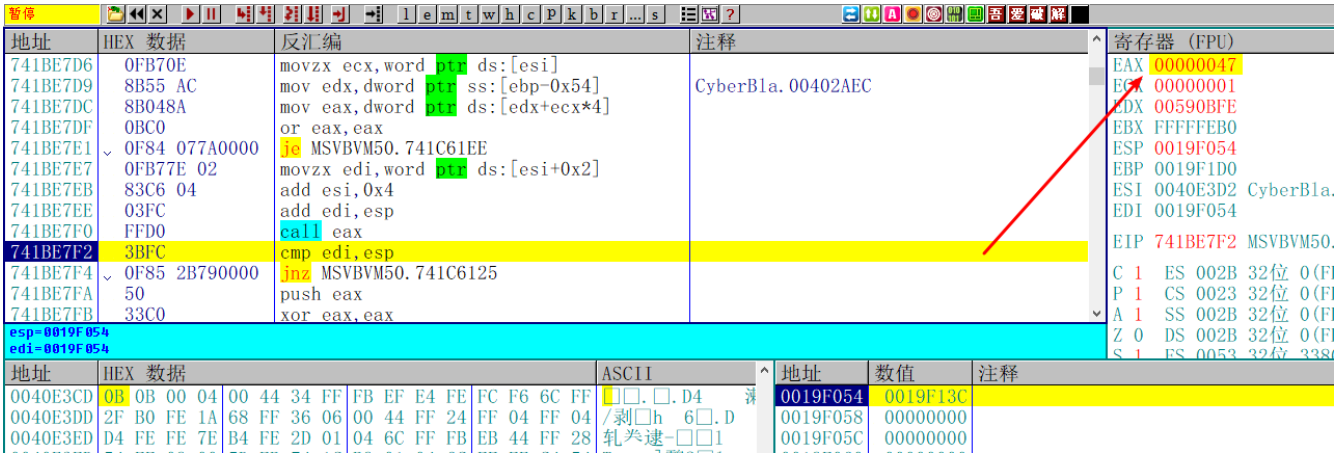
在40E3C1处截取用户名的第一个字符串



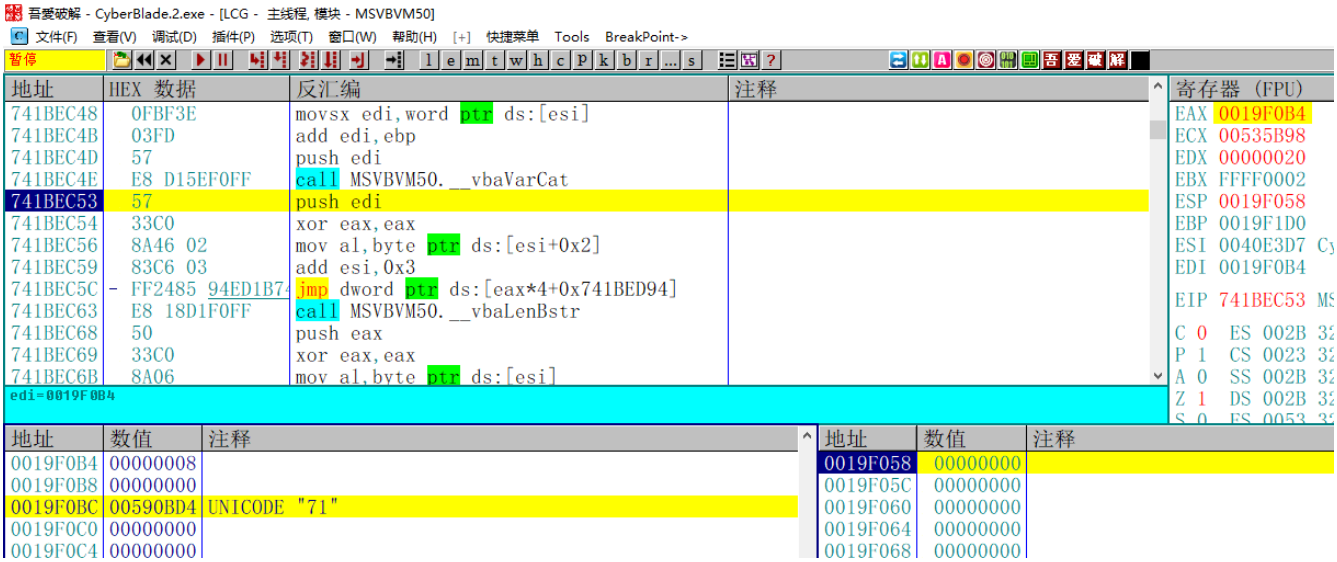
接着在40E3CD处将截取的用户名每一位转成ASCII值



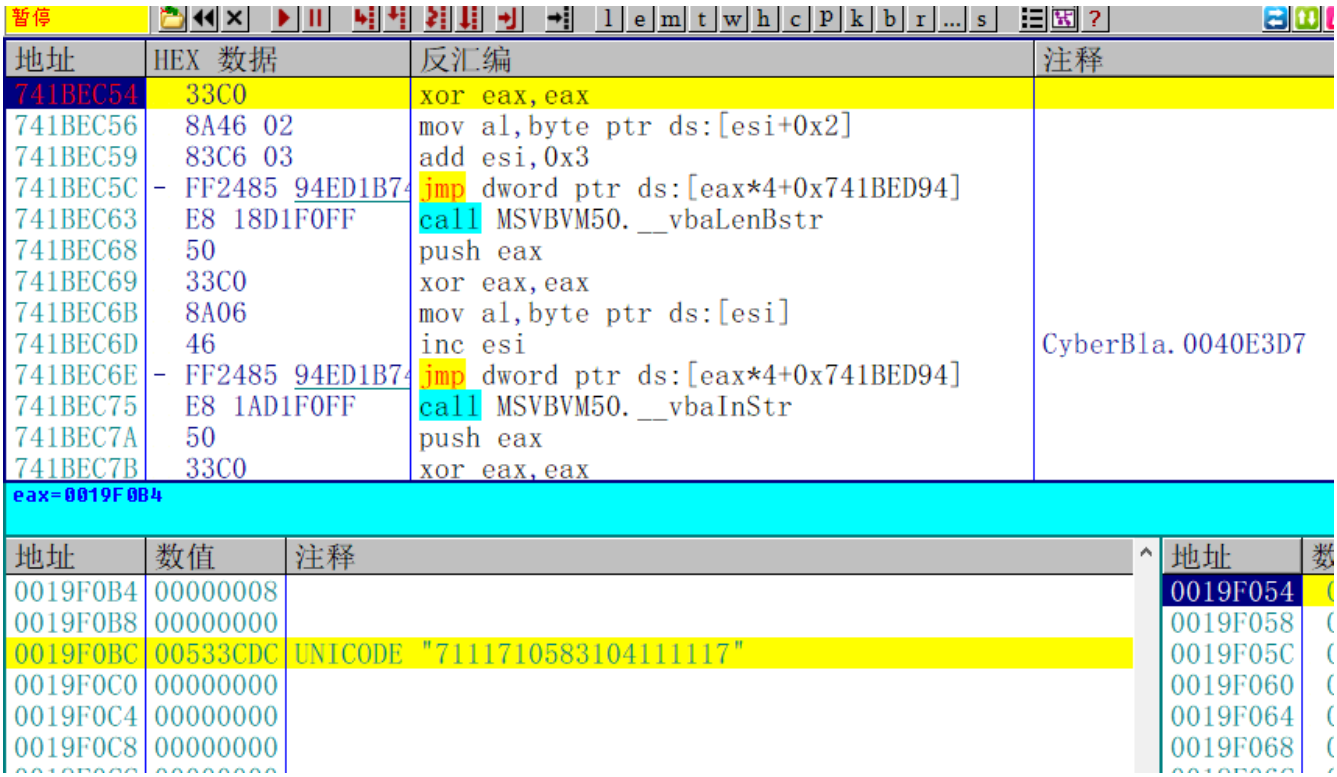




接着将用户名每一位的ASCII值转为十进制后进行字符串拼接



整个过程循环，循环次数为用户名的长度，可以直接在这个地方下断点看到最后的结果



即拼接用户名的ASCII十进制字符串，第二部分的算法就完成

```
loc_40E393: For var_14C = 1 To CVar(Len(Me.txtname.Text)): var_12C = var_14C 'Variant
loc_40E3C1:   var_FC = Mid(CVar(Me.txtname.Text), CLng(var_12C), 1)
loc_40E3D5:   var_11C = var_94 & CVar(Asc(CStr(var_FC)))
loc_40E3D9:   var_94 = var_11C 'Variant
loc_40E3EF: Next var_14C 'Variant
```

在VB伪代码中,var_94就是最后拼接的结果

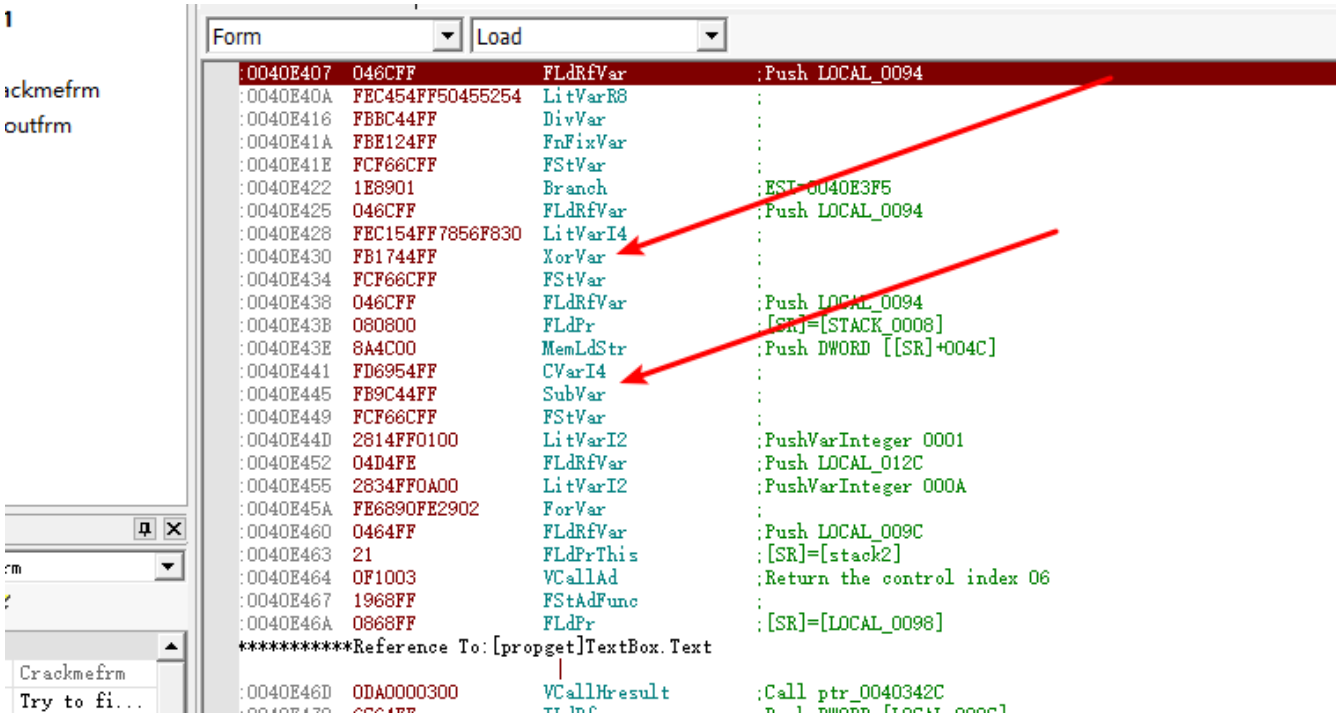
第三部分 除以圆周率

接下来是第三部分，直接来看VB Decompiler中的伪代码

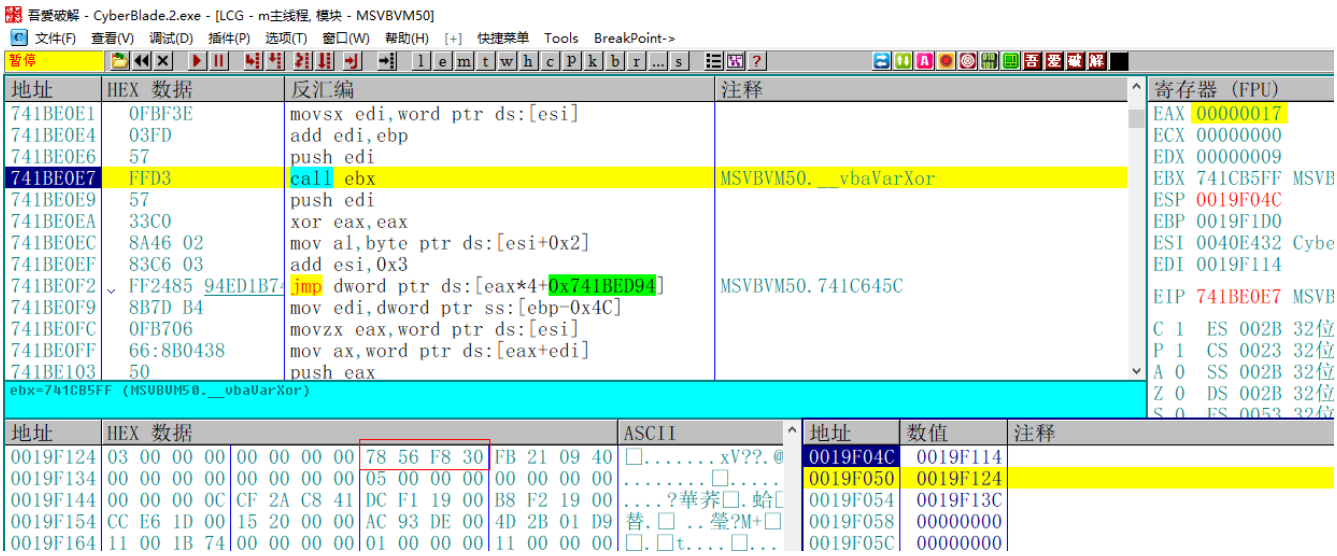
```
loc_40E3F5: ' Referenced from: 40E422
loc_40E404: If (Len(var_94) > 9) Then
loc_40E41E:   var_94 = Fix((var_94 / 3.141592654)) 'Variant
loc_40E422:   GoTo loc_40E3F5
loc_40E425: End If
loc_40E449: var_94 = (var_94 Xor &H30F85678 - CVar(global_76)) 'Variant
```

这一部分的逻辑也很清晰，如果用户名拼接的字符串长度大于9的话，就将这个结果转为浮点数除以圆周率，一直除到结果的长度小于9。这个部分我也用OD详细跟过每一条伪指令，确实和静态反汇编的逻辑是一样的

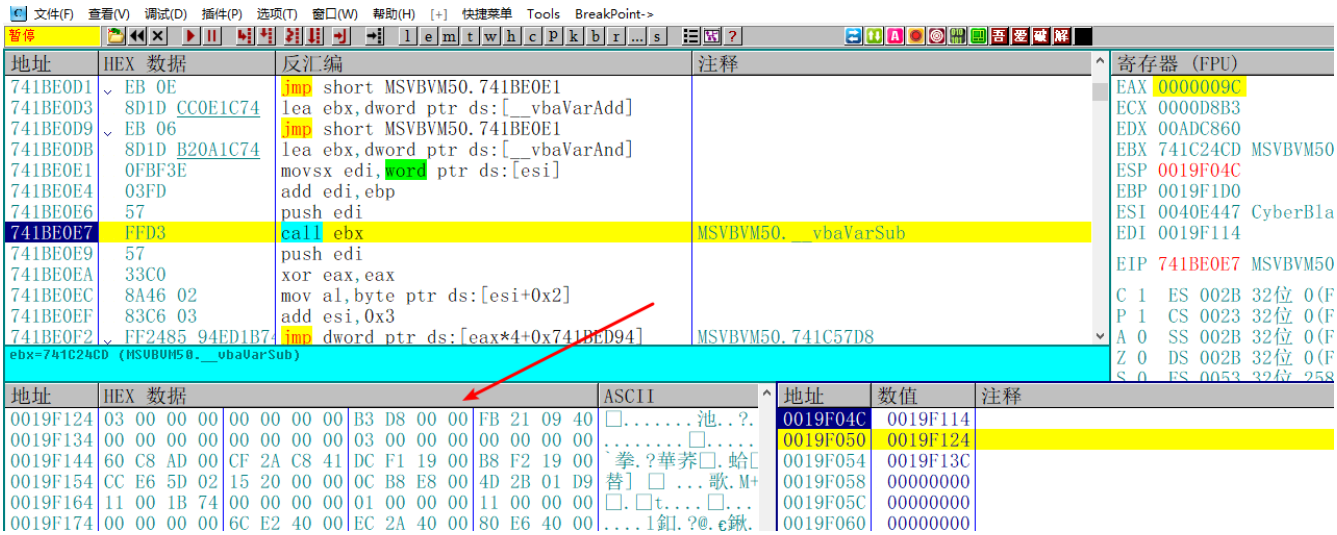
但是在loc_40E449的位置，将var_94和一个值进行了异或，并且还减去了另外一个值。这两个数值我们无从得知，只能跟踪OD



根据伪指令的助记符XorVar和SubVar快速定位到这两个地址，下内存访问断点，很快就能找到这两个值

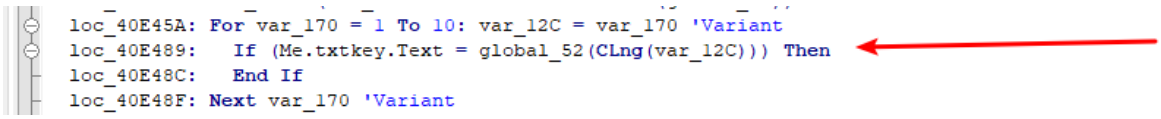


可以看到这里实际上是将0x30F85678和用户名的结果进行异或



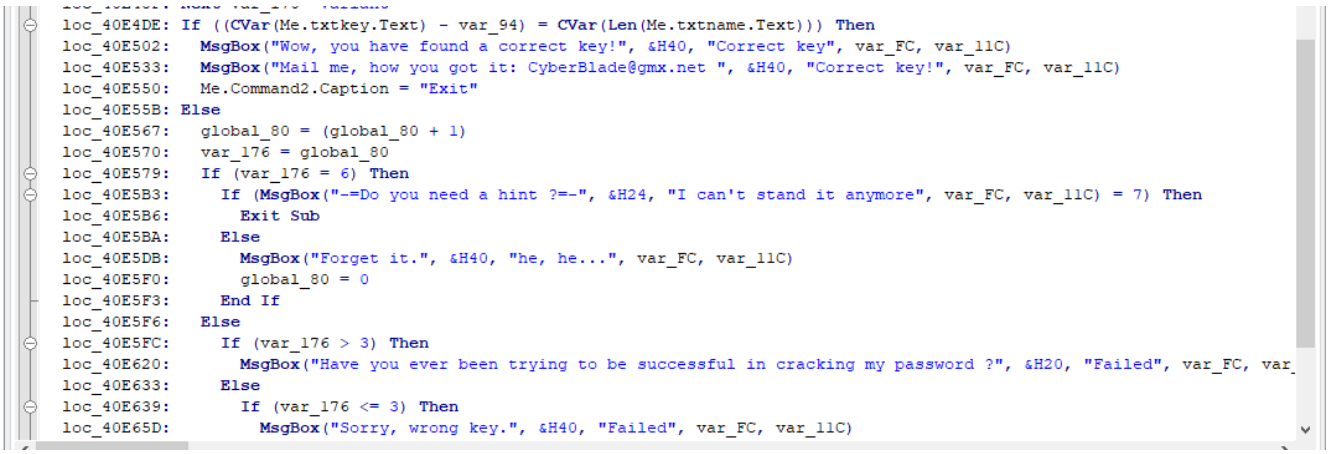
然后减去0xD8B3，找到了这两个数，第三部分也就结束了

第四部分 干扰代码



这个是最有意思的，你会发现代码初始化了10次循环，循环将一个变量和Key值进行比较，但问题在于Then分支没有任何代码。这也就是说不管这个循环中的比较成立与否 都对我们没有任何影响

第五部分 关键比较



最后一部分，比较序列号减去var_94是否等于用户名长度，也就是说用户名计算的结果再加上用户名的长度就是真正的序列号。最后对这个程序的校验过程做一个总结

总结:

1. 检测用户名和序列号是否为空，序列号长度是否小于5
2. 将用户名各个字符的ASCII码的十进制转换成字符串连接起来得到一个数result
3. 判断result长度是否大于9 如果大于则将result转为浮点数之后除以圆周率 直到resultl的长度小于或等于9

4. 将result和0x30F85678进行异或并减去0xD8B3，再加上用户名长度就是正确的序列号

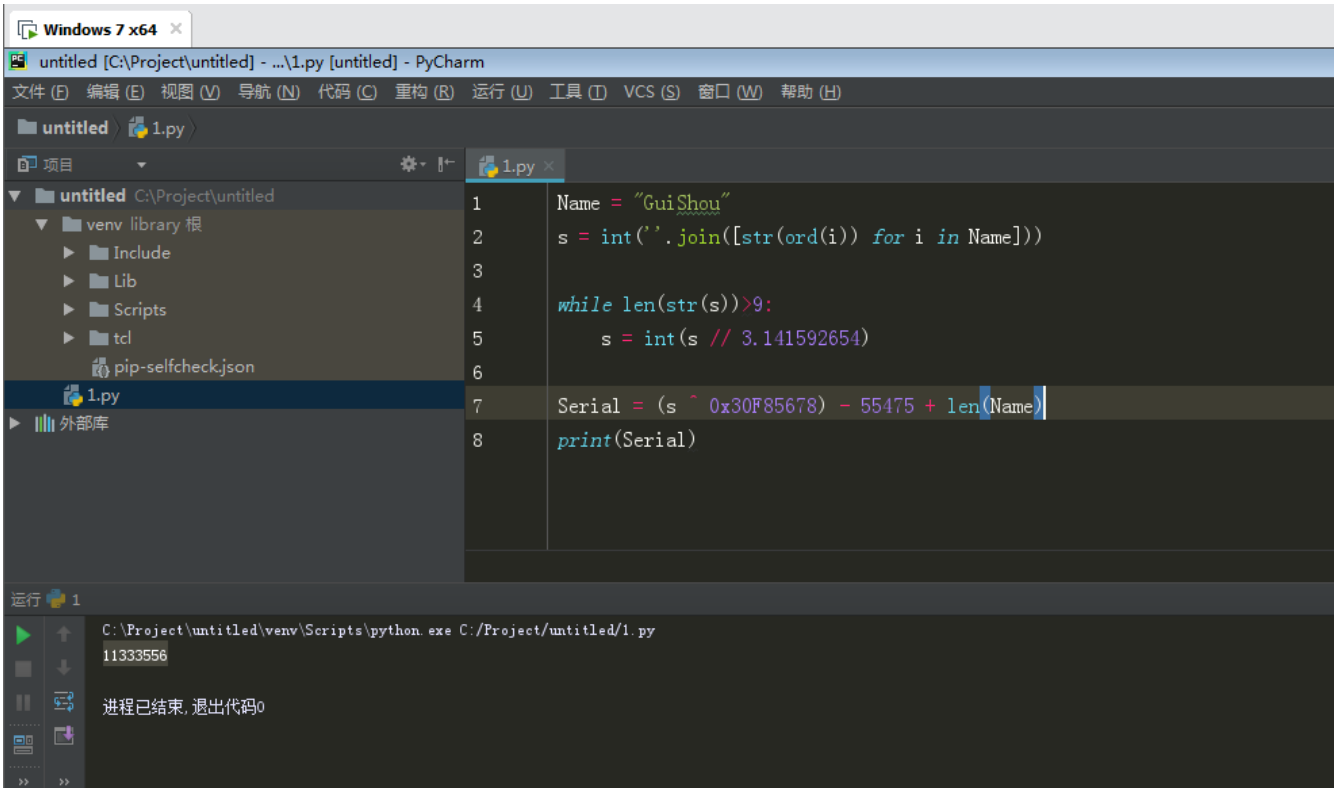
写出注册机

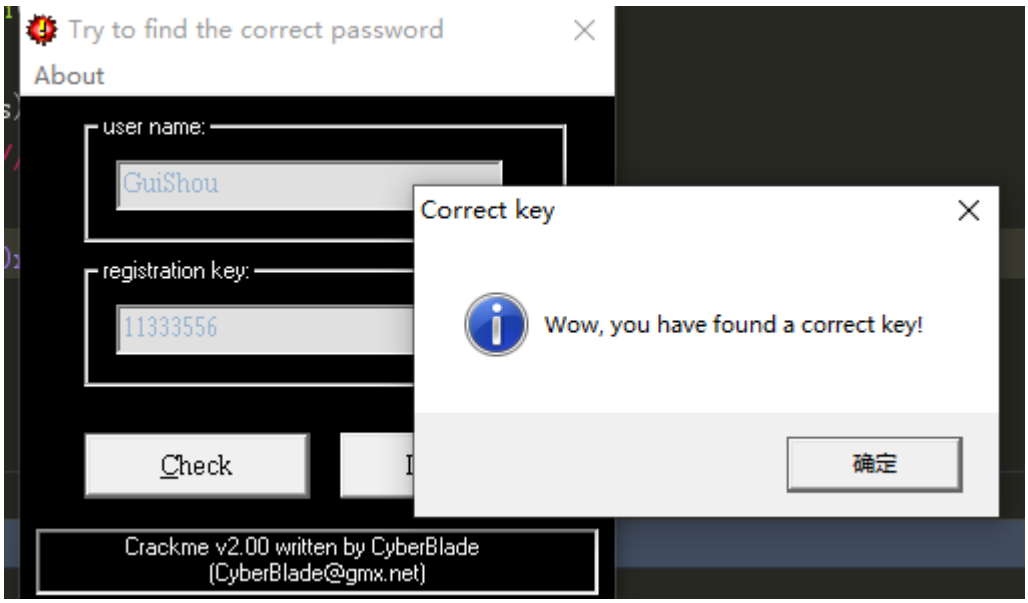
接着我们根据已经分析的算法写出这个程序的注册机，这个注册机用C++写还是太费劲了 直接用python快

```
Name = "GuiShou"
s = int(''.join([str(ord(i)) for i in Name]))

while len(str(s))>9:
    s = int(s // 3.141592654)

Serial = (s ^ 0x30F85678) - 55475 + len(Name)
print(Serial)
```





总结

P-Code的程序并非如传言一样不可战胜，只要你有足够的耐心，配合VB Decompiler+VBExplorer+OD的黄金组合，剩下的就是纯体力活了。

P-Code类的程序用OD跟踪伪指令虽然能看到每一处实现细节，但是毕竟还是太费力了。这个时候如果能总结出一套相对比较完整的P-Code的伪指令及每个参数的具体含义再配合WKTVBDebugger，调试P-Code就显得游刃有余了。如果有大佬总结出来了还请发我一份 哈哈。

附上分析过程

最后附上分析过程和相关文件

```
:0040E380  0DA0000300          VCallHresult        ;Call ptr_0040342C
获取输入的用户名
:0040E385  6C64FF              ILdRf               ;Push DWORD [LOCAL_009C]
将用户名压入堆栈
:0040E388  4A                  FnLenStr            ;vbaLenBstr
求用户名长度
:0040E389  FD6934FF            CVarI4              ;
将用户名长度转为变量
:0040E38D  2F64FF              FFree1Str           ;SysFreeString
[LOCAL_009C]; [LOCAL_009C]=0      释放用户名内存
:0040E390  1A68FF              FFree1Ad            ;Push [LOCAL_0098]; Call
[[[LOCAL_0098]]+8]; [[LOCAL_0098]]=0    释放局部变量
:0040E393  FE68B4FE8901        ForVar              ;
 初始化循环次数  开始循环
:0040E399  0464FF              FLdRfVar            ;Push LOCAL_009C
 将局部变量0压入堆栈
```

```
:0040E39C  21                  FLdPrThis          ;[SR]=[stack2]
:0040E39D  0F0003              VCallAd            ;Return the control index
02
:0040E3A0  1968FF              FStAdFunc          ;
:0040E3A3  0868FF              FLdPr              ;[SR]=[LOCAL_0098]
***********Reference To:[propget]TextBox.Text
                               |
:0040E3A6  0DA0000300          VCallHresult       ;Call ptr_0040342C
 获取输入的用户名
:0040E3AB  046CFF              FLdRfVar           ;Push LOCAL_0094
 将局部变量0压入堆栈
:0040E3AE  2824FF0100          LitVarI2           ;PushVarInteger 0001
 将局部变量1压入堆栈
:0040E3B3  04D4FE              FLdRfVar           ;Push LOCAL_012C
 将局部变量0压入堆栈
:0040E3B6  FC22                CI4Var             ;vbaI4Var
将变量1转为数字
:0040E3B8  3E64FF              FLdZeroAd          ;Push DWORD [LOCAL_009C];
[LOCAL_009C]=0     将用户名压入堆栈
:0040E3BB  4644FF              CVarStr            ;
将用户名字符串转为变量
:0040E3BE  0404FF              FLdRfVar           ;Push LOCAL_00FC
将局部变量0压入堆栈
**********Reference To->msvbvm50.rtcMidCharVar
                               |
:0040E3C1  0A0A001000          ImpAdCallFPR4      ;Call ptr_00401006; check
stack 0010; Push EAX    截取用户名的第一个字符
:0040E3C6  0404FF              FLdRfVar           ;Push LOCAL_00FC
将用户名的第一个字符压入堆栈
:0040E3C9  FDFEB0FE            CStrVarVal         ;
将用户名的第一个字符从变量转为字符串
**********Reference To->msvbvm50.rtcAnsiValueBstr
                               |
:0040E3CD  0B0B000400          ImpAdCallI2        ;Call ptr_0040100C; check
stack 0004; Push EAX  将用户名的第一个字符转为ASCII值
:0040E3D2  4434FF              CVarI2             ;                    将用户名
的第一个字符的ASCII值转为变量
:0040E3D5  FBEFE4FE            ConcatVar          ;                    将ASCII
值的十进制进行字符串拼接
:0040E3D9  FCF66CFF            FStVar             ;                    将拼接的
字符串转为变量
:0040E3DD  2FB0FE              FFree1Str          ;SysFreeString
[LOCAL_0150]; [LOCAL_0150]=0    释放字符串
:0040E3E0  1A68FF              FFree1Ad           ;Push [LOCAL_0098]; Call
[[[LOCAL_0098]]+8]; [[LOCAL_0098]]=0
```

```
:0040E3E3   36060044FF24FF04      FFreeVar            ;Free 0006/2 variants
释放变量
:0040E3EC   04D4FE                FLdRfVar            ;Push LOCAL_012C
:0040E3EF   FE7EB4FE2D01          NextStepVar         ;
开始下一轮循环
:0040E3F5   046CFF                FLdRfVar            ;Push LOCAL_0094
    将用户名拼接的字符串压入堆栈--7111710583104111117(0x13)
:0040E3F8   FBEB44FF              FnLenVar            ;vbaLenVar
求用户名拼接的字符串长度
:0040E3FC   2854FF0900            LitVarI2            ;PushVarInteger 0009
将整形变量9压入堆栈
:0040E401   5D                    HardType            ;
修改变量9的类型
:0040E402   FB74                  GtVarBool           ;Push (Pop1 >= Pop2)
比较长度是否大于9
:0040E404   1CB901                BranchF             ;If Pop=0 then ESI=0040E425
如果不大于9则跳转到0040E425
:0040E407   046CFF                FLdRfVar            ;Push LOCAL_0094
将用户名拼接的字符串压入堆栈
:0040E40A   FEC454FF50455254      LitVarR8            ;
将参数一(圆周率)的类型修改为浮点数
:0040E416   FBBC44FF              DivVar              ;
    将用户名拼接的字符串除以圆周率
:0040E41A   FBE124FF              FnFixVar            ;
相当于字符串拷贝
:0040E41E   FCF66CFF              FStVar              ;
将用户名拼接的字符串转为浮点数
:0040E422   1E8901                Branch              ;ESI=0040E3F5
如果长度大于9则跳转至0040E3F5
:0040E425   046CFF                FLdRfVar            ;Push LOCAL_0094
将浮点数结果压入堆栈
:0040E428   FEC154FF7856F830      LitVarI4            ;
将浮点数转为整形
:0040E430   FB1744FF              XorVar              ;
将结果和30F85678进行异或
:0040E434   FCF66CFF              FStVar              ;
保存结果
:0040E438   046CFF                FLdRfVar            ;Push LOCAL_0094
将异或后的结果压栈
:0040E43B   080800                FLdPr               ;[SR]=[STACK_0008]
:0040E43E   8A4C00                MemLdStr            ;Push DWORD [[SR]+004C]
将0xDBD3压栈
:0040E441   FD6954FF              CVarI4              ;
将0xDBD3转为变量
:0040E445   FB9C44FF              SubVar              ;
用异或后的结果减去0xDBD3
```

```
:0040E449  FCF66CFF            FStVar              ;
保存结果
```

需要分析记录和相关文件可以到我的Github下载：https://github.com/TonyChen56/160-Crackme