****

**分析报告**

|  |  |
| --- | --- |
| **样本名** | **010Editor.exe** |
| **班级** | **软安41期** |
| **作者** | **张海龙** |
| **时间** | **2020年12月1日** |
| **平台** | **Win10** |

15PB信息安全研究院(协议分析报告)

# 目录

[目录 2](#_Toc27874)

[1．样本概况 3](#_Toc5682)

[1.1 应用程序信息 3](#_Toc27657)

[1.2 分析环境及工具 3](#_Toc18139)

[1.3 分析目标 3](#_Toc27732)

[2．具体分析过程 3](#_Toc14671)

[2.1 找到暴力破解关键点 3](#_Toc15517)

[2.2 完美暴力破解 7](#_Toc19623)

[2.3 算法逆向 8](#_Toc1178)

[2.4 注册机 12](#_Toc17084)

[2.5 网络验证 18](#_Toc15317)

[参考文献 20](#_Toc17852)

[致 谢 21](#_Toc22888)

# 1．样本概况

## 1.1 应用程序信息

应用程序名称：010Editor.exe

大小: 46891744 bytes

文件版本:8.0.1

修改时间: 2017年9月29日, 13:57:34

MD5: DB355048EF963913173F64AE0872AEE5

SHA1: 92CABC7485896E40FA3081FDF51425CB6ADBFE81

CRC32: 93694514

010Editor.exe需要注册才能长期使用。

## 1.2 分析环境及工具

系统环境:win10 64位

工具:OD CE DIE MFC注入工具

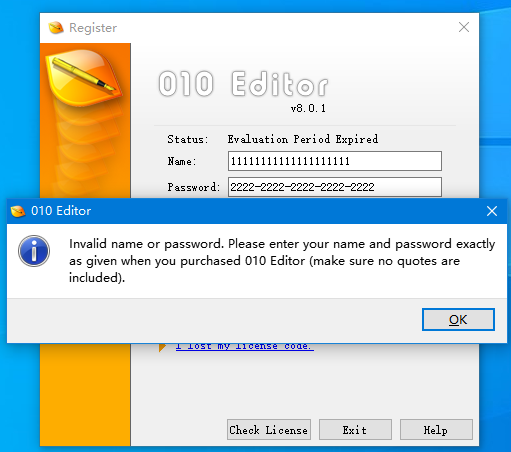
## 1.3 分析目标

写一个010Editor的注册机，能够实现输入用户名就能计算出真正的序列号

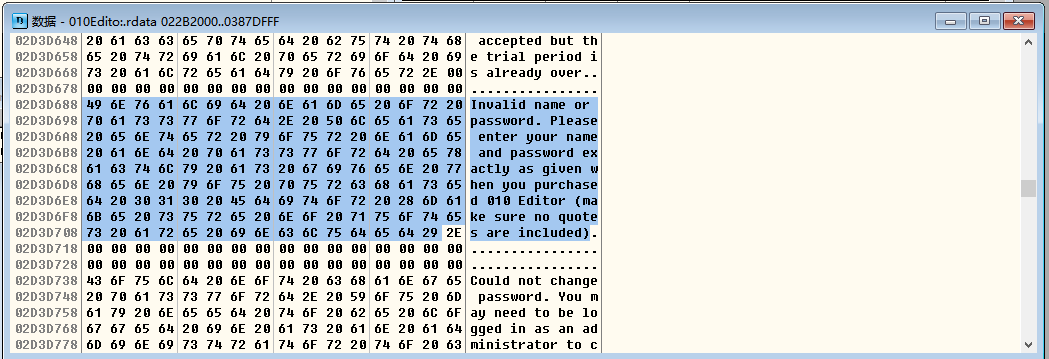
# 2．具体分析过程

## 2.1 找到暴力破解关键点

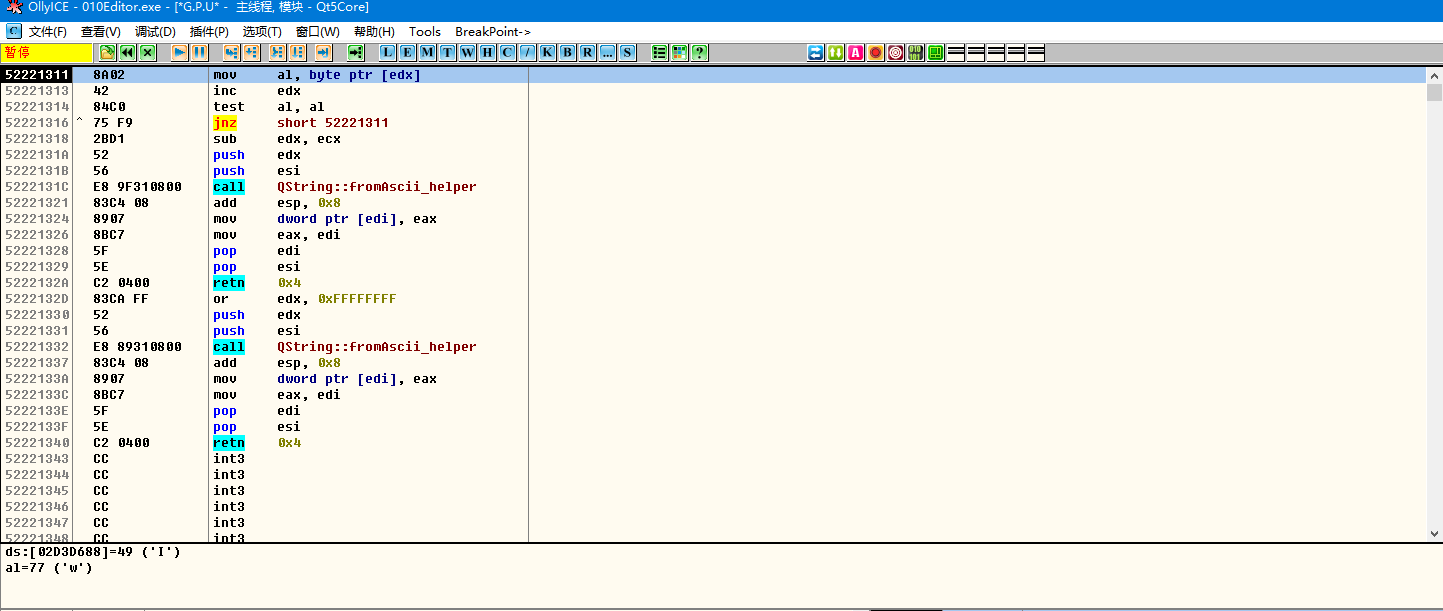
1、我们先填一个假的注册码和账号，点击注册，然后弹出下面的一个对话框，提示以下信息Invalid name or password. Please enter your name and password exactly as given when you purchased 01。尝试使用OD内存窗口中的字符串搜索这个字符串,ASCII和UNICODE都试试。



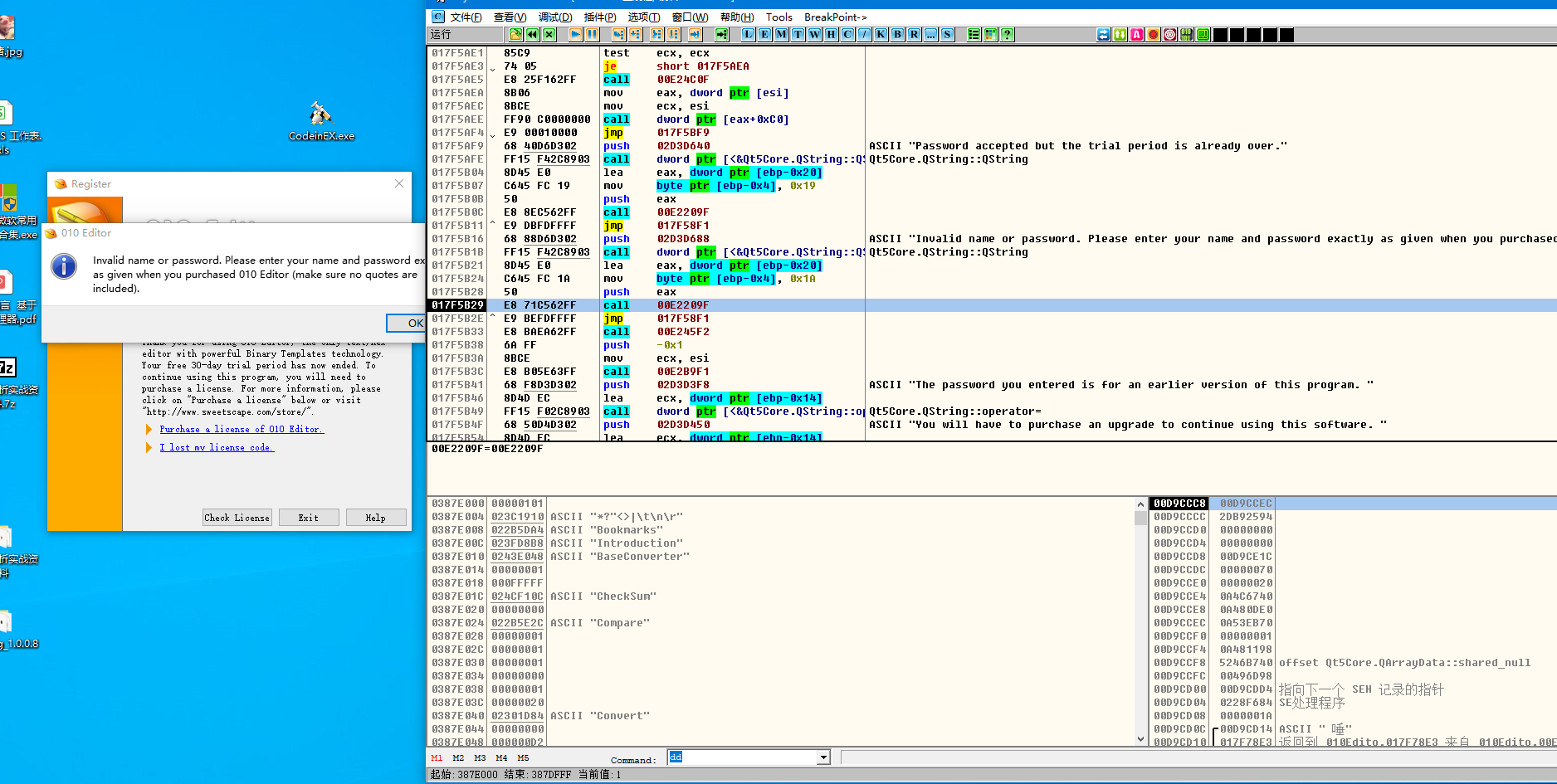
2、OK,成功找到了这个字符串，我们尝试对这个字符串下一个内存访问断点，因为弹出对话框的时候，会访问这个字符串的。



1. 程序断下来了，断在了访问第一个字母的地方，一看地址是52开头的，并且OD标题栏显示断在了Qt5Core模块。

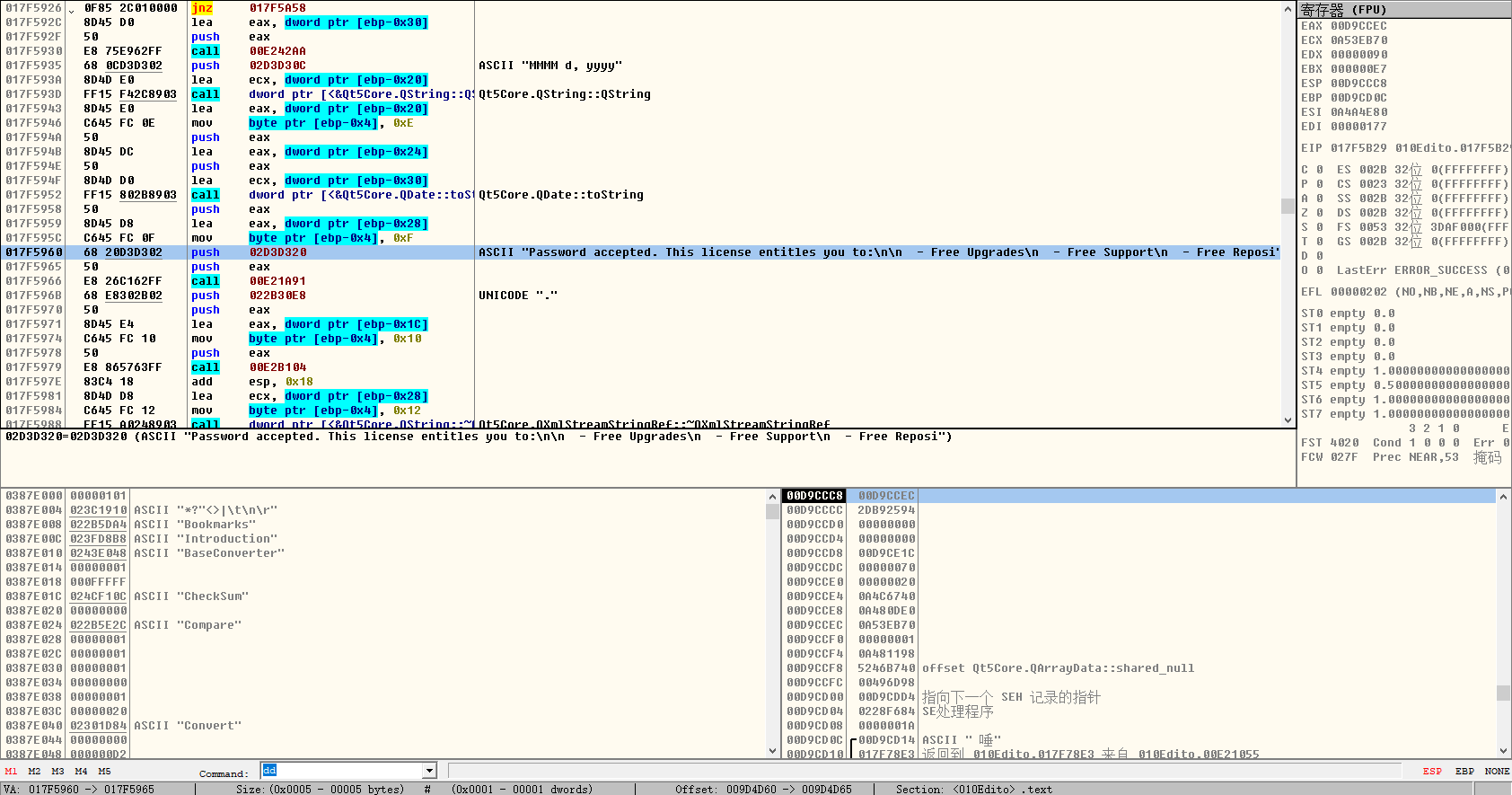


1. 我们一直执行到返回，直到执行到用户代码。并再往下走几步到017F5B29 处。弹出了消息框，断定这个call 00E2209F，就是弹消息框的。上面有相关字符串。大概找到了弹出消息的关键点。

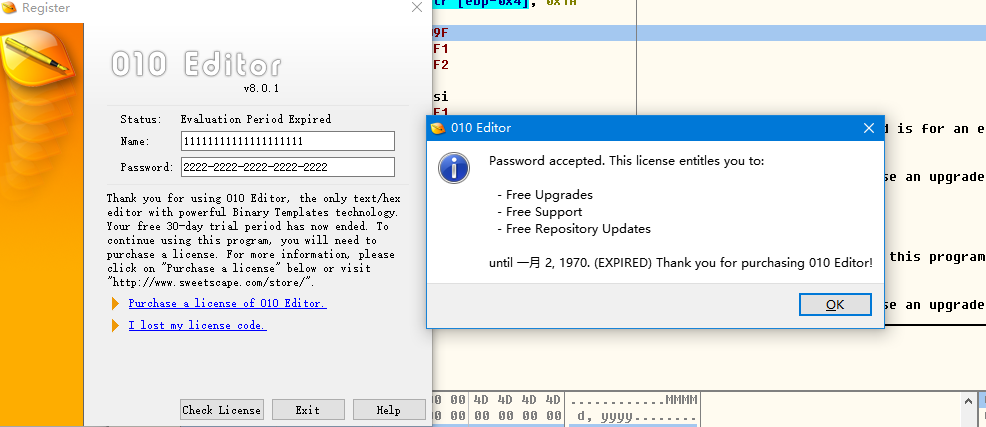


1. 往上翻一翻看看有没有提示注册成功的消息，果然发现了有个提示注册成功的消息，

上面还有个jnz刚好会跳过这个提示成功的代码。



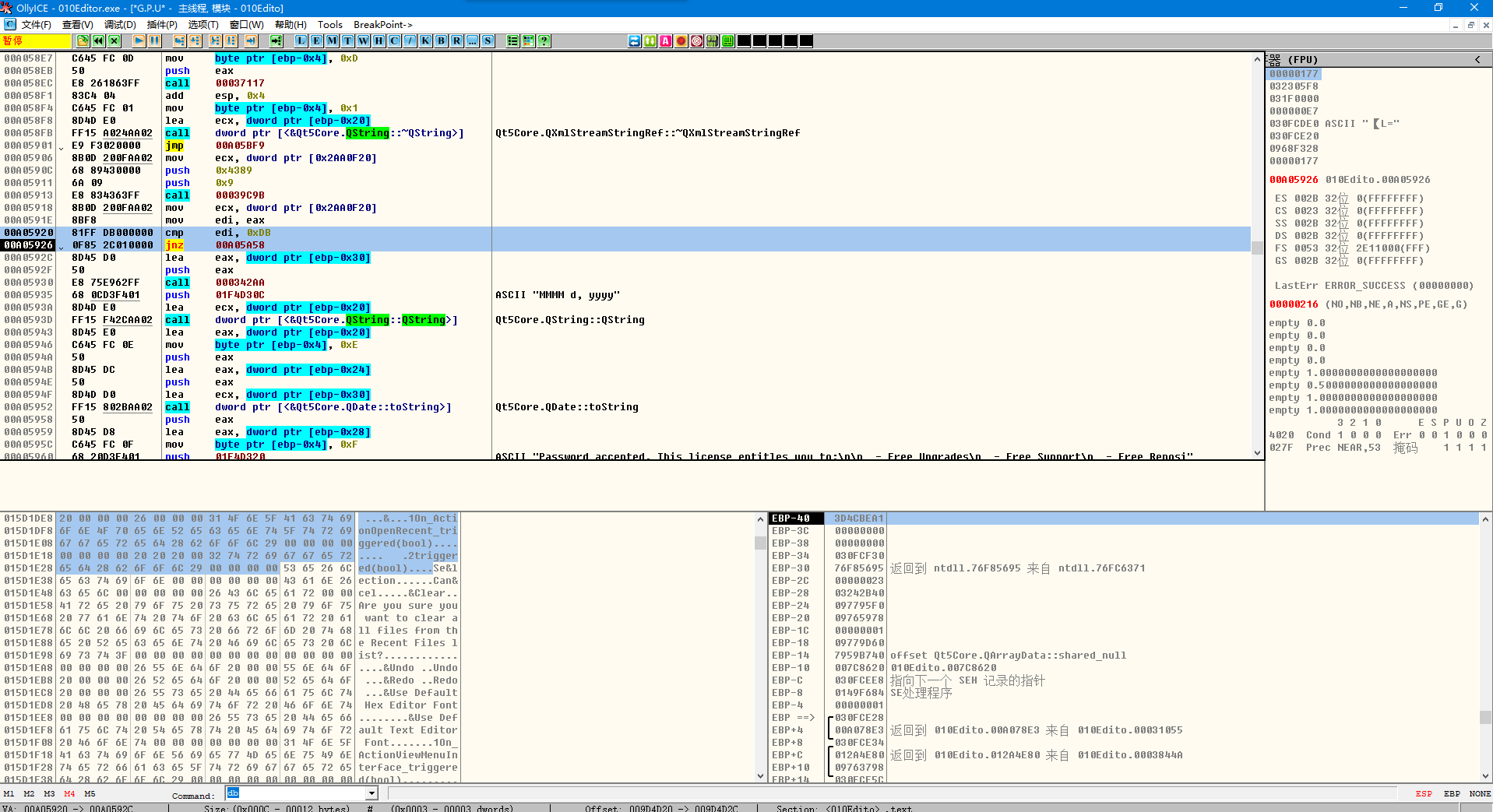
1. 我们直接nop了jnz的代码观察注册的反馈结果。OK，提示注册成功了。



暴力破解成功，找到了关键点。

## **2.2 完美暴力破解**

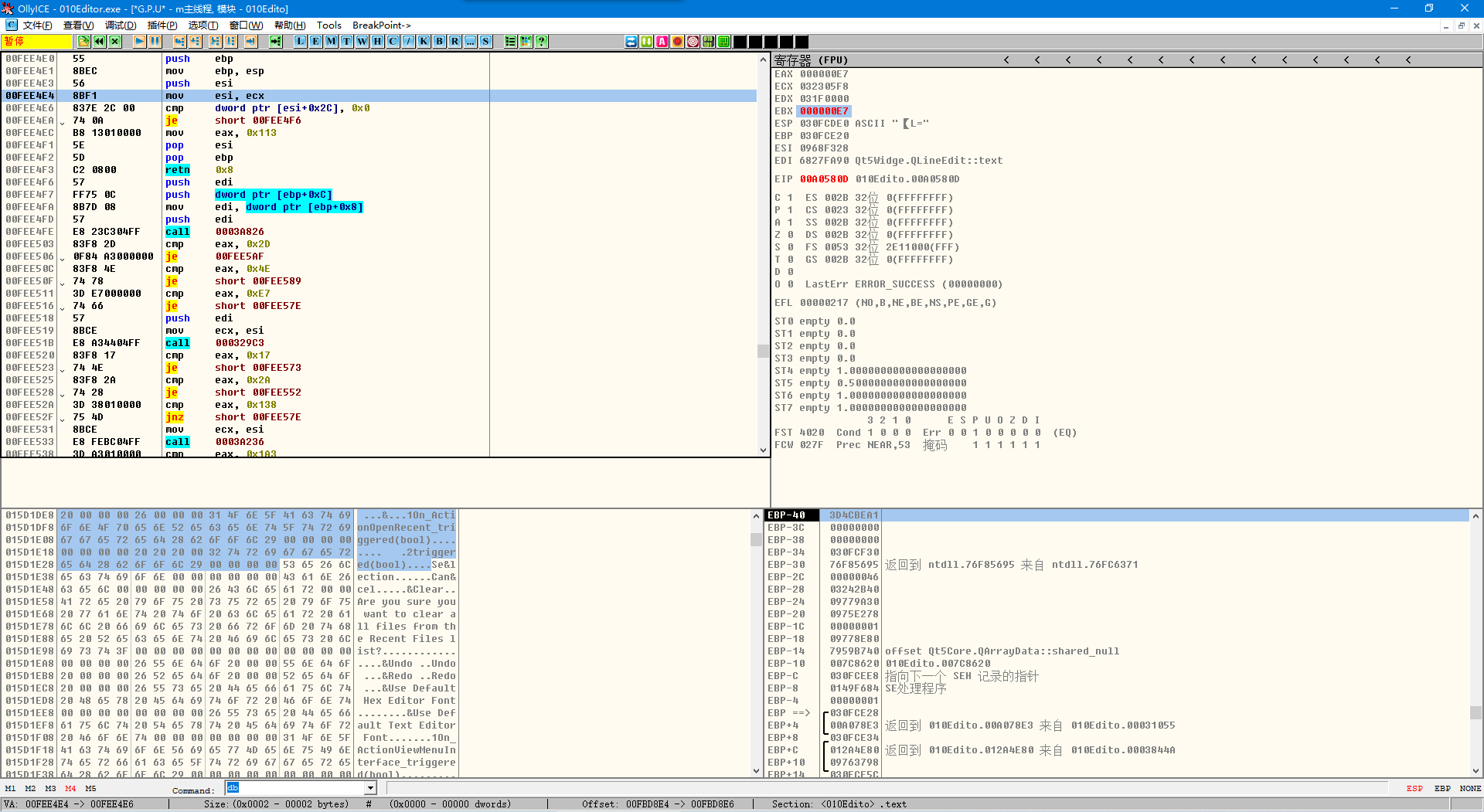
现在还不是很清楚程序的判断流程，先在刚刚找到的关键点那个地址所在函数头部下一个断点，走一遍这个函数，观察执行流程。留意，一下那个关键跳转的上方。F8单歩执行



走到这里发现刚好会跳过我们的正确提示，因为edi和0xDB比较,EDI哪里来的呢？

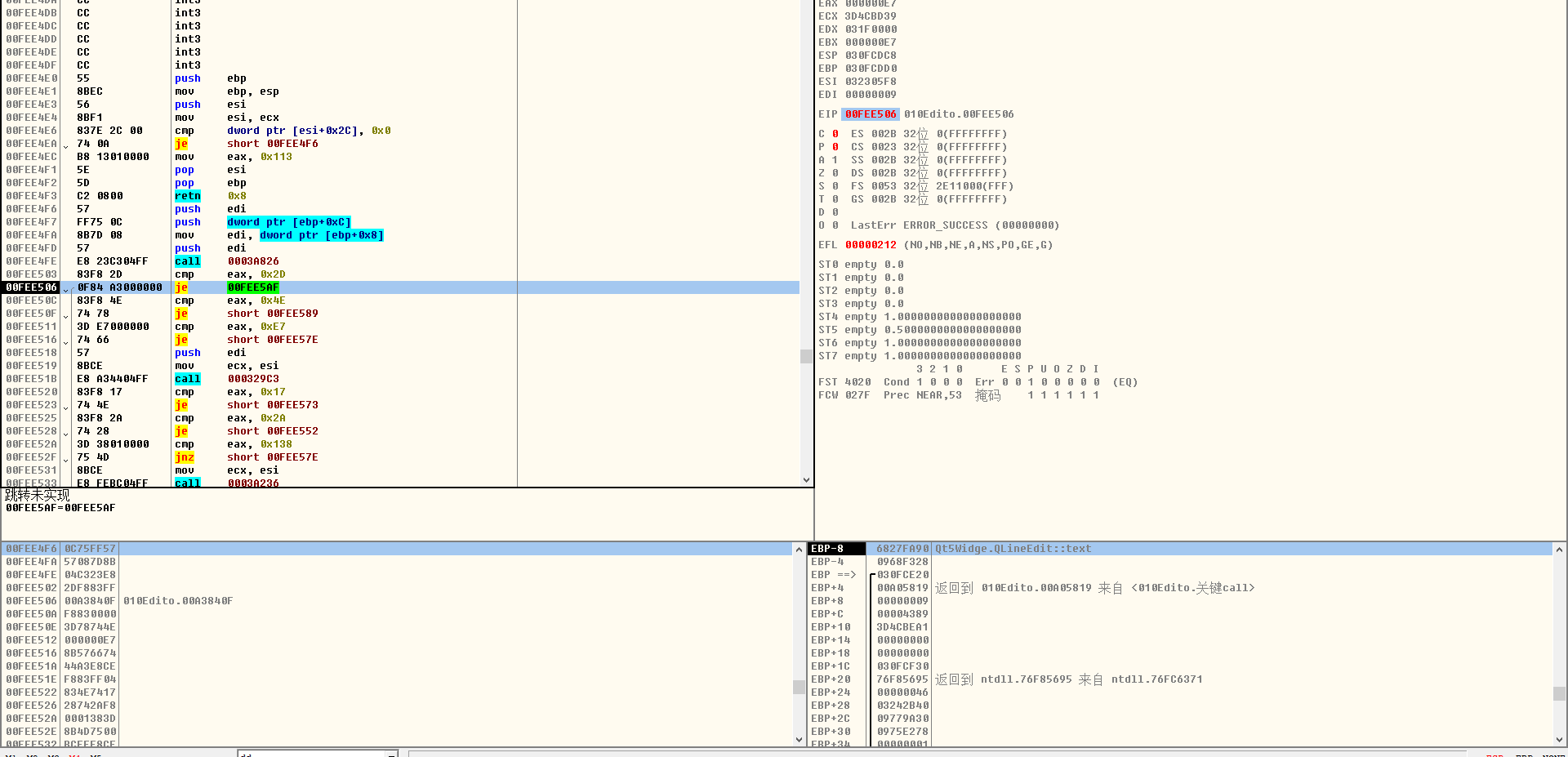
按减号回溯刚才的执行流程。发现edi来源于eax，eax又是上一个call的返回值。那个答案就出来了。只要上一个call返回0xDB，那么就可以注册成功了。

我们来分析一下这个call，进入call发现直接使用了ecx，应该是ecx传参了。



往下拉，看返回值，看到了mov eax,0xDB ret 0x8,是两个栈传参的thiscall

估计这个就是关键的算法了。

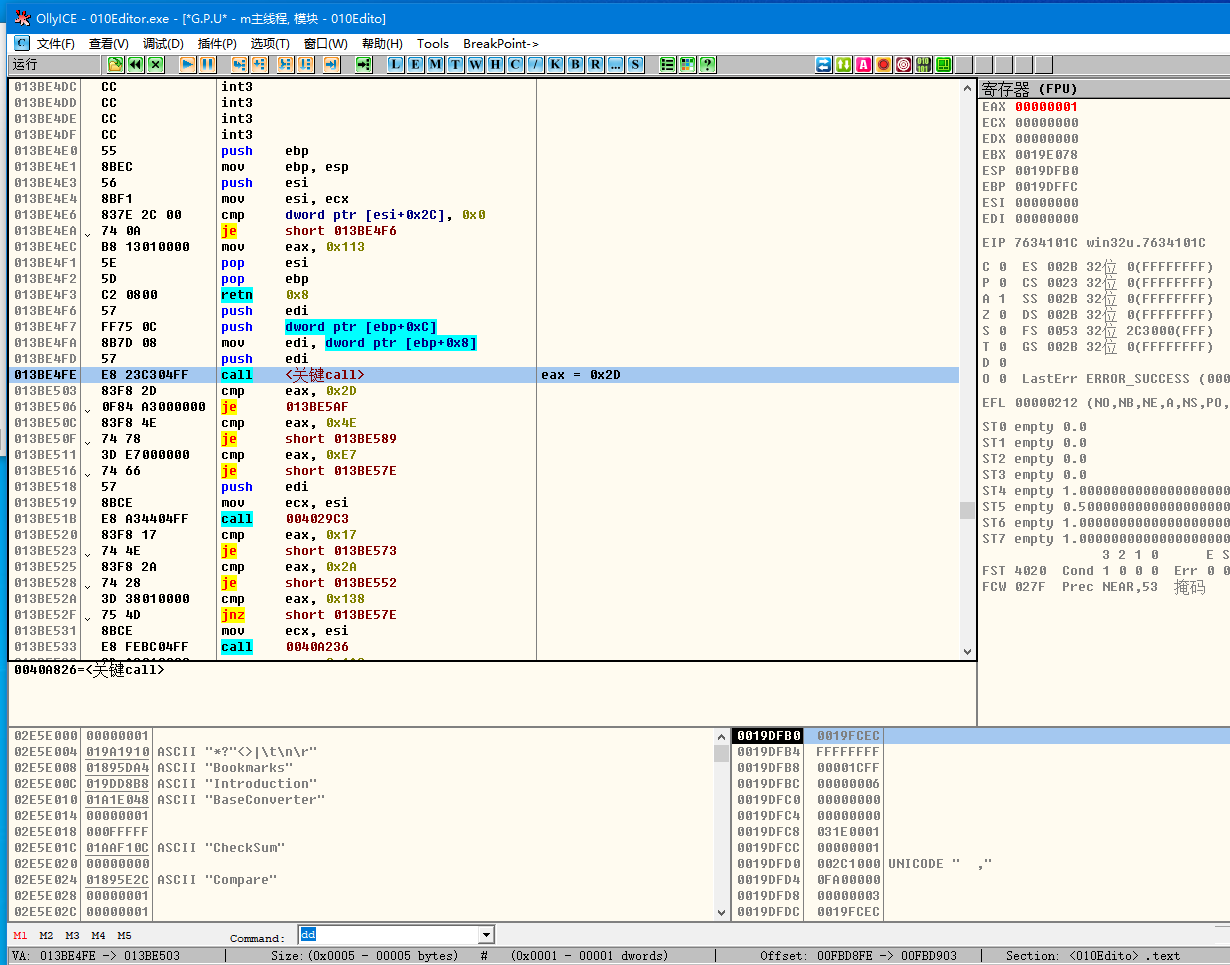


到这里发现je跳转走下去刚好是跳转到返回 eax = 0xDB,所以只要让上面的一个call返回

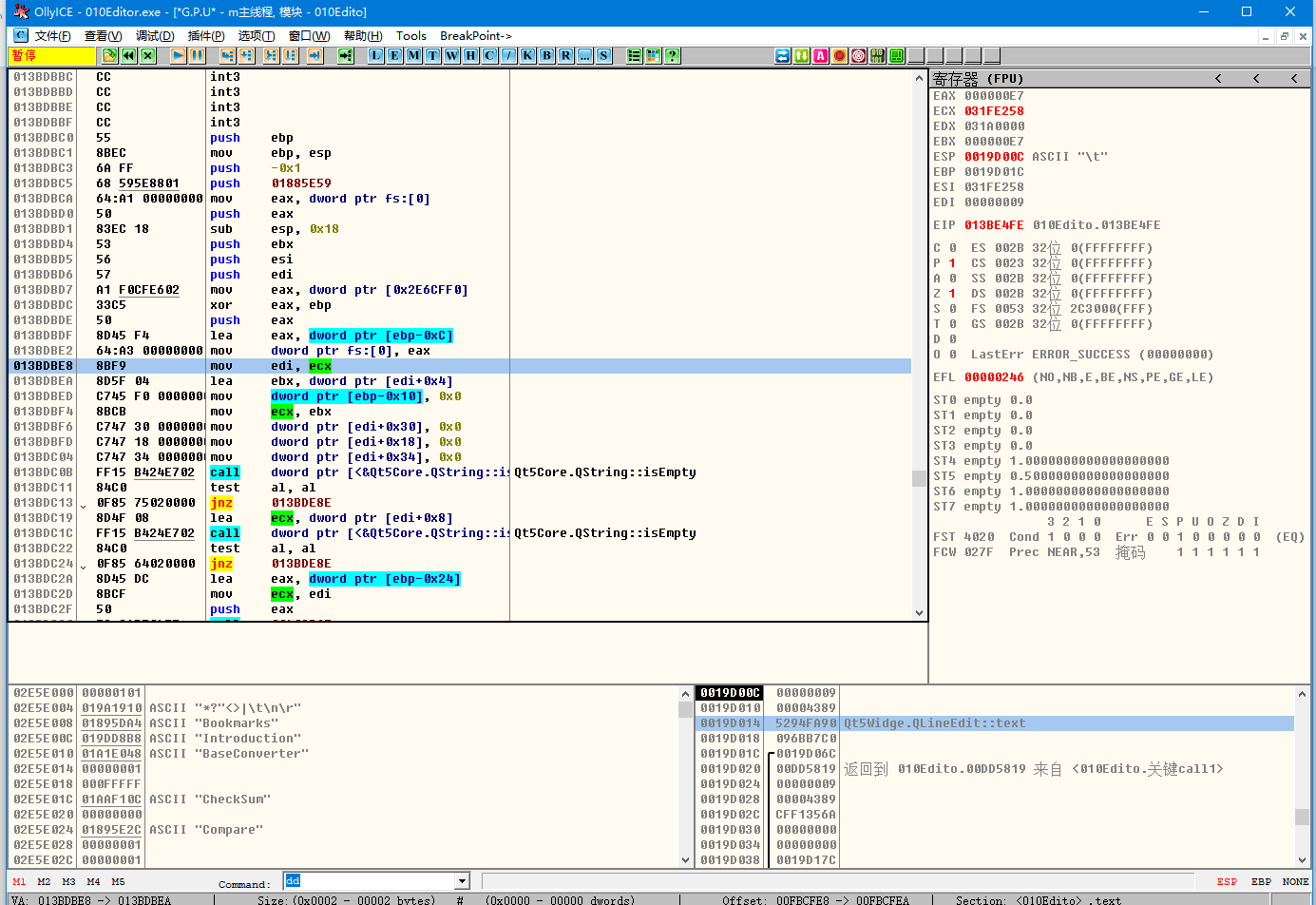
0x2D即可，关掉随机基址后，我们就在上面一个call里面修改代码，mov eax, 0x2D ret 0x8

## 2.3 算法逆向

分析关键call代码。

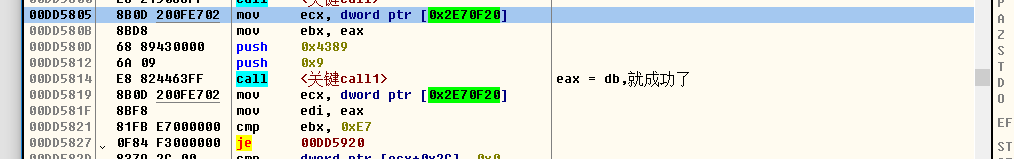


查看里面call的返回值和是否直接操作，ecx的例如lea xxx，ecx mov xxx，ecx

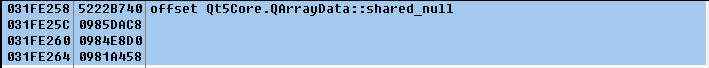


Mov edi,ecx说明是thiscall

Ecx来源于上一层的ecx，来源于最外层的[0x2E70F20]



里面存放到是



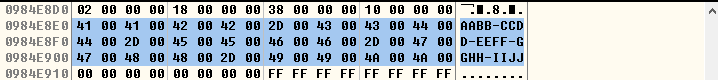
第一个已经给标出来了，QArraryData应该是数组

第二个是个地址：



存放的是我们输入的账户名

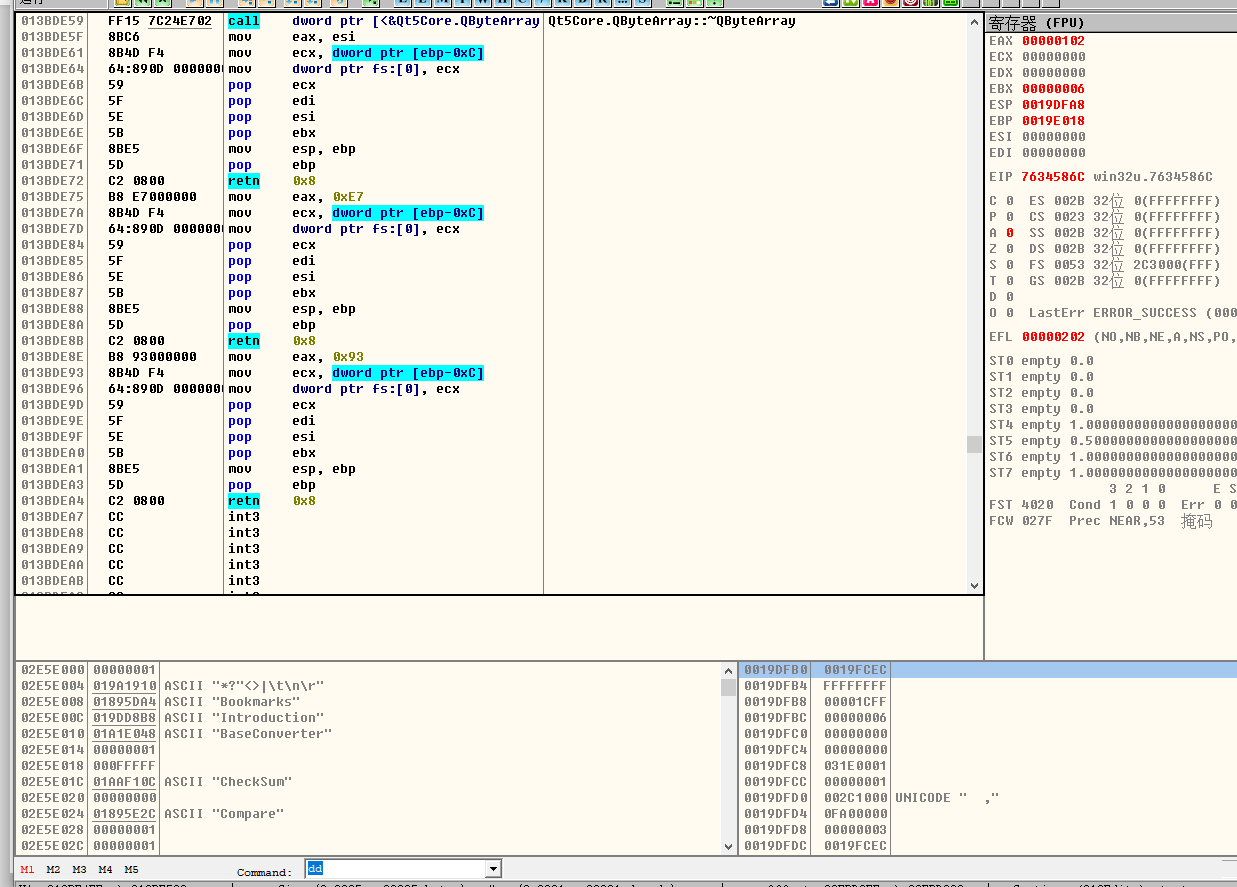
第三个是个地址：



存放的是我们输入的Key

第四个是个地址：什么都没有





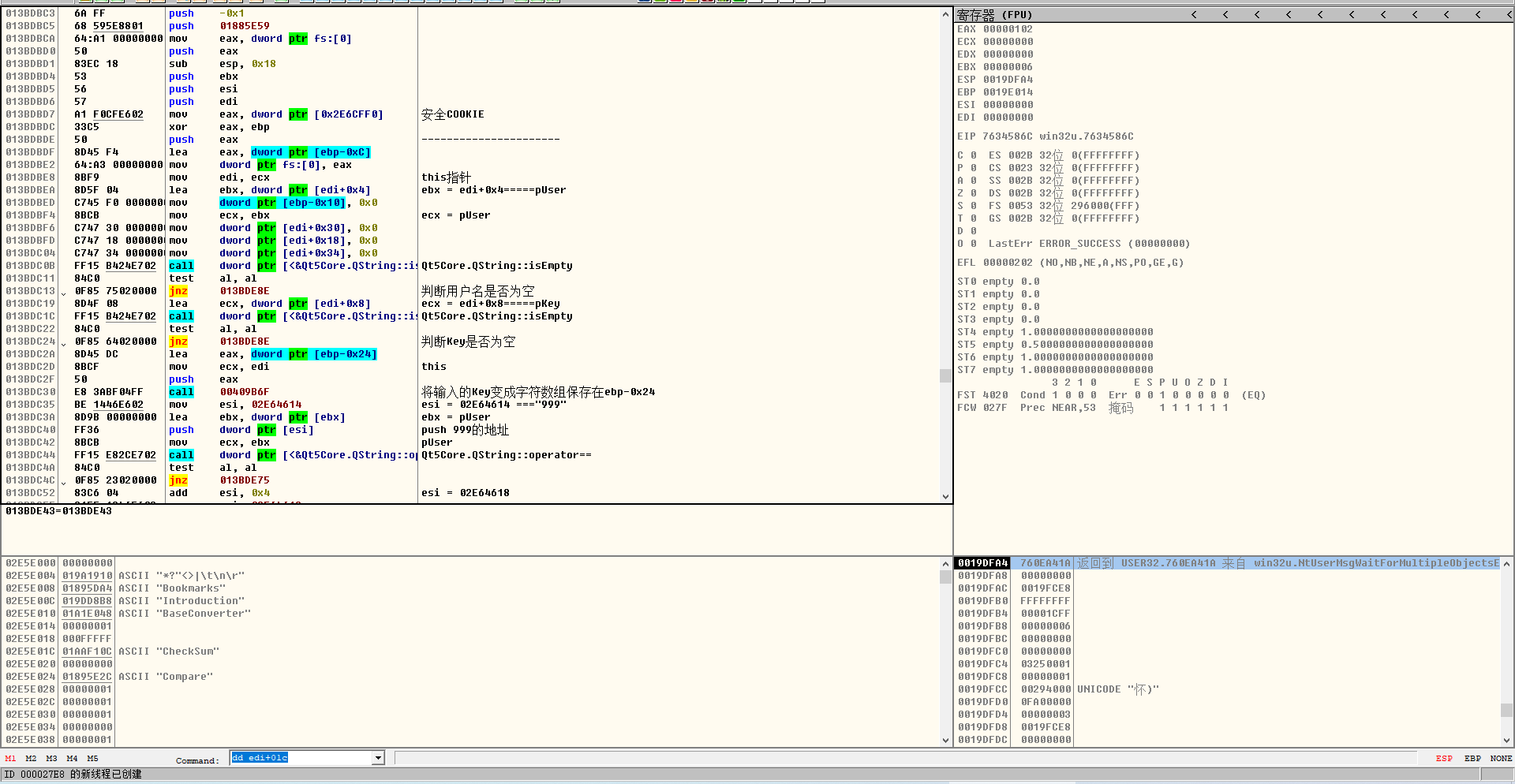
Ret 0x8说明有两个参数，

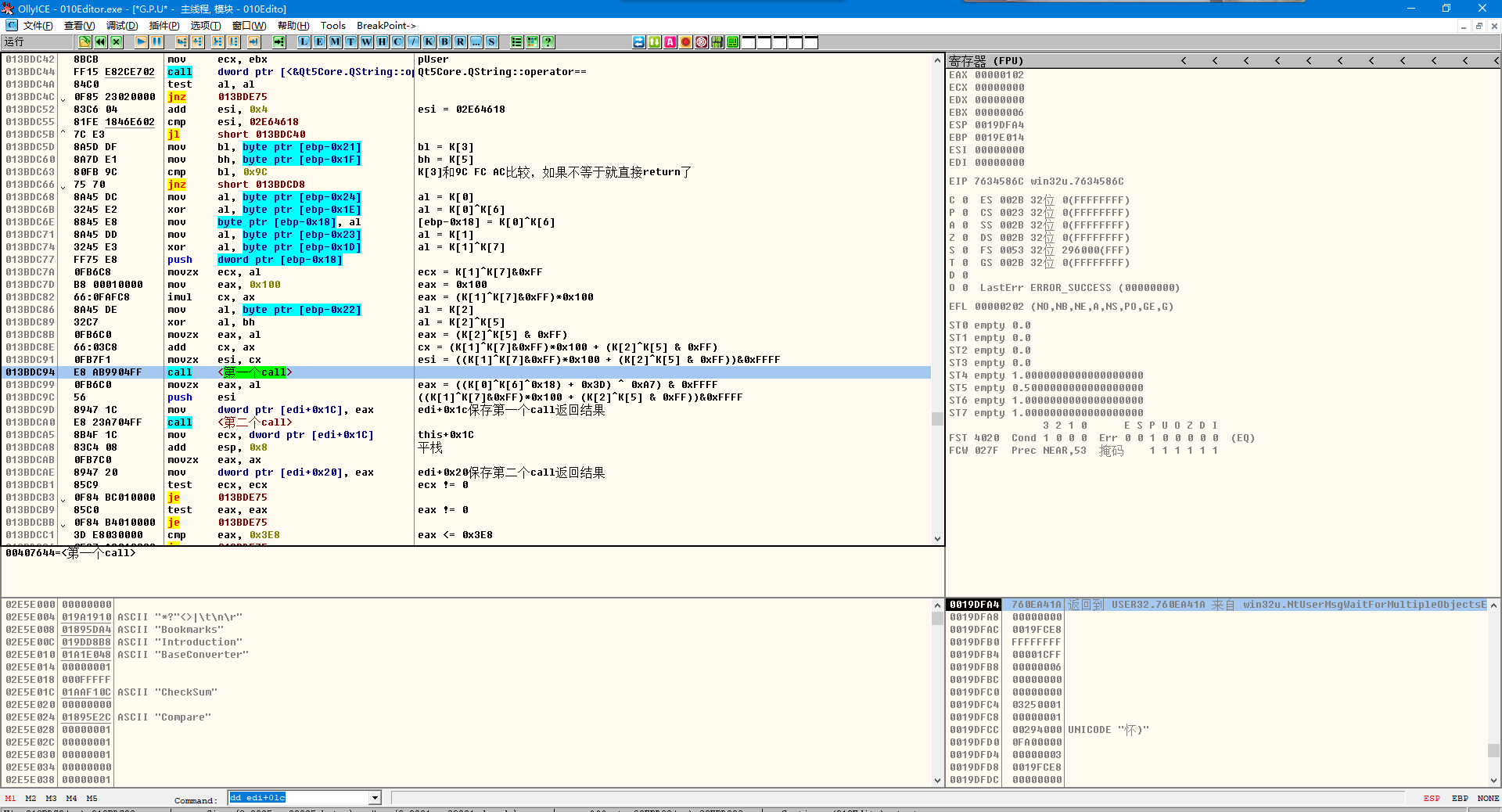
跟踪一下这个call看看做了什么操作。

只要让他返回0x2D就验证成功了。

看看怎么样才能返回0x2D.

进去看一下。

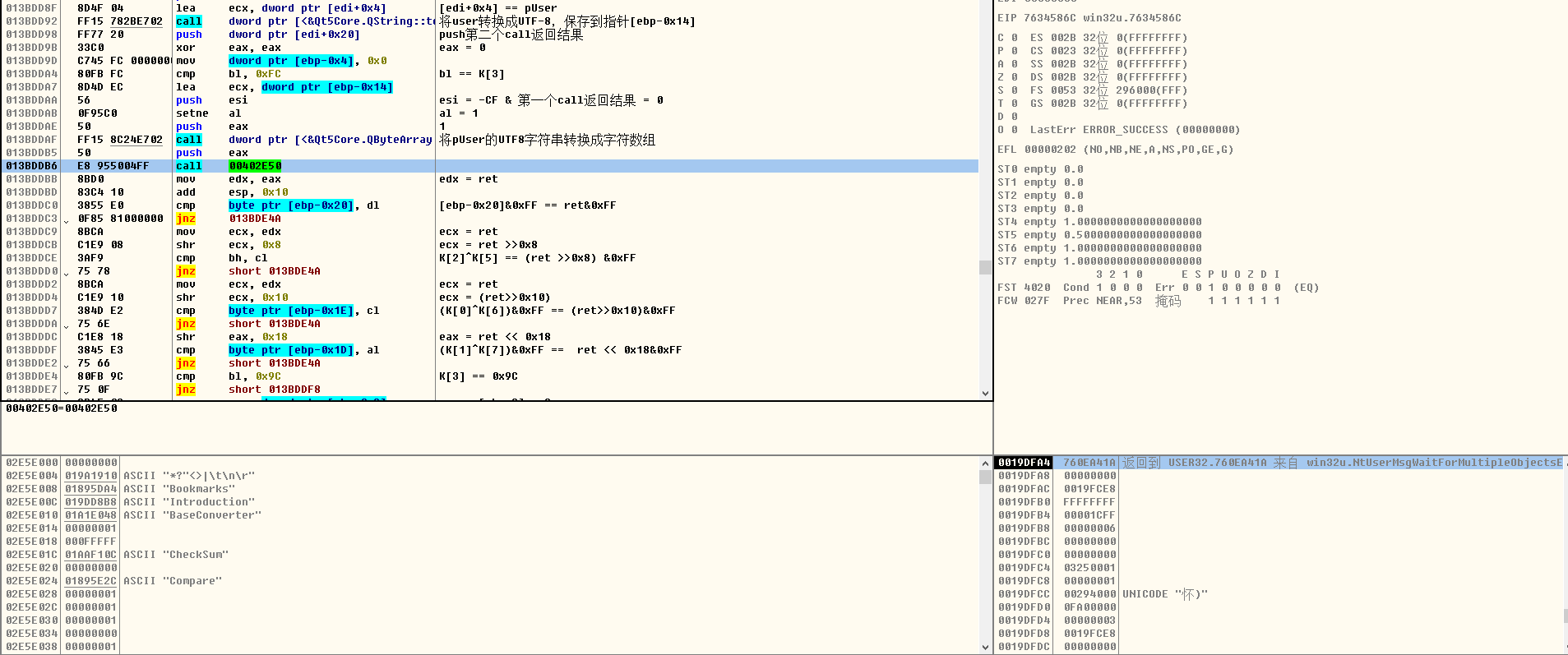




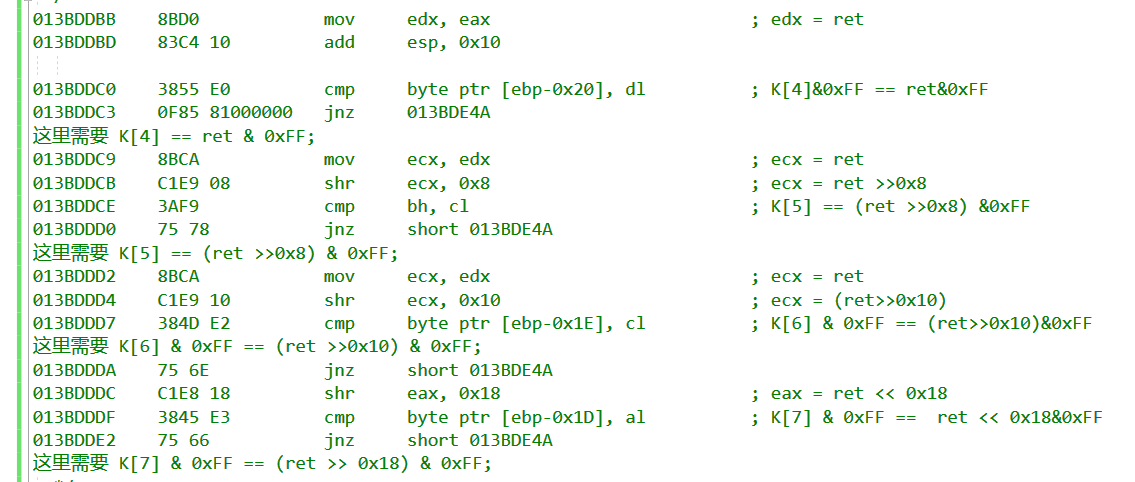
第一个call验证K[0]和K[6]

第二个call 验证K[1]/K[2]/K[5]/K[7]

K[3]只可为9C/FC/AC



这里是给用户名加密，并返回



下面这段代码我给拷出来了，关键代码，是验证加密后的数据进行一下操作，必须以cmp的值相等。

K[4] = dwKey & 0xFF;

K[5] = dwKey >> 0x8 & 0xFF;

K[6] = dwKey >> 0x10 & 0xFF;

K[7] = dwKey >> 0x18 & 0xFF;

## 2.4 注册机

#include <stdio.h>

#include <windows.h>

#include <time.h>

DWORD DeCode[] = {

0x39cb44b8, 0x23754f67, 0x5f017211, 0x3ebb24da, 0x351707c6, 0x63f9774b, 0x17827288, 0x0fe74821, 0x5b5f670f, 0x48315ae8, 0x785b7769, 0x2b7a1547, 0x38d11292, 0x42a11b32, 0x35332244, 0x77437b60,

0x1eab3b10, 0x53810000, 0x1d0212ae, 0x6f0377a8, 0x43c03092, 0x2d3c0a8e, 0x62950cbf, 0x30f06ffa, 0x34f710e0, 0x28f417fb, 0x350d2f95, 0x5a361d5a, 0x15cc060b, 0x0afd13cc, 0x28603bcf, 0x3371066b,

0x30cd14e4, 0x175d3a67, 0x6dd66a13, 0x2d3409f9, 0x581e7b82, 0x76526b99, 0x5c8d5188, 0x2c857971, 0x15f51fc0, 0x68cc0d11, 0x49f55e5c, 0x275e4364, 0x2d1e0dbc, 0x4cee7ce3, 0x32555840, 0x112e2e08,

0x6978065a, 0x72921406, 0x314578e7, 0x175621b7, 0x40771dbf, 0x3fc238d6, 0x4a31128a, 0x2dad036e, 0x41a069d6, 0x25400192, 0x00dd4667, 0x6afc1f4f, 0x571040ce, 0x62fe66df, 0x41db4b3e, 0x3582231f,

0x55f6079a, 0x1ca70644, 0x1b1643d2, 0x3f7228c9, 0x5f141070, 0x3e1474ab, 0x444b256e, 0x537050d9, 0x0f42094b, 0x2fd820e6, 0x778b2e5e, 0x71176d02, 0x7fea7a69, 0x5bb54628, 0x19ba6c71, 0x39763a99,

0x178d54cd, 0x01246e88, 0x3313537e, 0x2b8e2d17, 0x2a3d10be, 0x59d10582, 0x37a163db, 0x30d6489a, 0x6a215c46, 0x0e1c7a76, 0x1fc760e7, 0x79b80c65, 0x27f459b4, 0x799a7326, 0x50ba1782, 0x2a116d5c,

0x63866e1b, 0x3f920e3c, 0x55023490, 0x55b56089, 0x2c391fd1, 0x2f8035c2, 0x64fd2b7a, 0x4ce8759a, 0x518504f0, 0x799501a8, 0x3f5b2cad, 0x38e60160, 0x637641d8, 0x33352a42, 0x51a22c19, 0x085c5851,

0x032917ab, 0x2b770ac7, 0x30ac77b3, 0x2bec1907, 0x035202d0, 0x0fa933d3, 0x61255df3, 0x22ad06bf, 0x58b86971, 0x5fca0de5, 0x700d6456, 0x56a973db, 0x5ab759fd, 0x330e0be2, 0x5b3c0ddd, 0x495d3c60,

0x53bd59a6, 0x4c5e6d91, 0x49d9318d, 0x103d5079, 0x61ce42e3, 0x7ed5121d, 0x14e160ed, 0x212d4ef2, 0x270133f0, 0x62435a96, 0x1fa75e8b, 0x6f092fbe, 0x4a000d49, 0x57ae1c70, 0x004e2477, 0x561e7e72,

0x468c0033, 0x5dcc2402, 0x78507ac6, 0x58af24c7, 0x0df62d34, 0x358a4708, 0x3cfb1e11, 0x2b71451c, 0x77a75295, 0x56890721, 0x0fef75f3, 0x120f24f1, 0x01990ae7, 0x339c4452, 0x27a15b8e, 0x0ba7276d,

0x60dc1b7b, 0x4f4b7f82, 0x67db7007, 0x4f4a57d9, 0x621252e8, 0x20532cfc, 0x6a390306, 0x18800423, 0x19f3778a, 0x462316f0, 0x56ae0937, 0x43c2675c, 0x65ca45fd, 0x0d604ff2, 0x0bfd22cb, 0x3afe643b,

0x3bf67fa6, 0x44623579, 0x184031f8, 0x32174f97, 0x4c6a092a, 0x5fb50261, 0x01650174, 0x33634af1, 0x712d18f4, 0x6e997169, 0x5dab7afe, 0x7c2b2ee8, 0x6edb75b4, 0x5f836fb6, 0x3c2a6dd6, 0x292d05c2,

0x052244db, 0x149a5f4f, 0x5d486540, 0x331d15ea, 0x4f456920, 0x483a699f, 0x3b450f05, 0x3b207c6c, 0x749d70fe, 0x417461f6, 0x62b031f1, 0x2750577b, 0x29131533, 0x588c3808, 0x1aef3456, 0x0f3c00ec,

0x7da74742, 0x4b797a6c, 0x5ebb3287, 0x786558b8, 0x00ed4ff2, 0x6269691e, 0x24a2255f, 0x62c11f7e, 0x2f8a7dcd, 0x643b17fe, 0x778318b8, 0x253b60fe, 0x34bb63a3, 0x5b03214f, 0x5f1571f4, 0x1a316e9f,

0x7acf2704, 0x28896838, 0x18614677, 0x1bf569eb, 0x0ba85ec9, 0x6aca6b46, 0x1e43422a, 0x514d5f0e, 0x413e018c, 0x307626e9, 0x01ed1dfa, 0x49f46f5a, 0x461b642b, 0x7d7007f2, 0x13652657, 0x6b160bc5,

0x65e04849, 0x1f526e1c, 0x5a0251b6, 0x2bd73f69, 0x2dbf7acd, 0x51e63e80, 0x5cf2670f, 0x21cd0a03, 0x5cff0261, 0x33ae061e, 0x3bb6345f, 0x5d814a75, 0x257b5df4, 0x0a5c2c5b, 0x16a45527, 0x16f23945

};

int \_\_cdecl EncodeUserName(const char\* UserName, int a2, char res1, char res2)

{

const char\* TmpUserName; // edx

signed int len; // esi

signed int index; // edi

unsigned \_\_int8 local5; // bl

int UpperChar; // eax

int v9; // ecx

int ret; // ecx

int result; // eax

unsigned \_\_int8 local4; // [esp+8h] [ebp-10h]

unsigned \_\_int8 local3; // [esp+Ch] [ebp-Ch]

unsigned \_\_int8 local2; // [esp+10h] [ebp-8h]

int local1; // [esp+14h] [ebp-4h]

TmpUserName = UserName;

local1 = 0;

len = strlen(UserName);

index = 0;

if (len <= 0)

return 0;

local4 = 0;

local3 = 0;

local5 = 0xF \* res2;

local2 = 0x11 \* res1;

do

{

UpperChar = toupper((unsigned \_\_int8)TmpUserName[index]);

v9 = local1 + DeCode[UpperChar];

if (a2)

ret = DeCode[local3]

+ DeCode[local5]

+ DeCode[local2]

+ DeCode[(unsigned \_\_int8)(UpperChar + 0x2F)] \* (DeCode[(unsigned \_\_int8)(UpperChar + 0xD)] ^ v9);

else

ret = DeCode[local4]

+ DeCode[local5]

+ DeCode[local2]

+ DeCode[(unsigned \_\_int8)(UpperChar + 0x17)] \* (DeCode[(unsigned \_\_int8)(UpperChar + 0x3F)] ^ v9);

result = ret;

local1 = ret;

local3 += 19;

++index;

local2 += 9;

local5 += 13;

local4 += 7;

TmpUserName = UserName;

} while (index < len);

return result;

}

int main()

{

// 1 初始化

srand((unsigned int)time(NULL));// 设置随机数种子

int dwRet = rand() % 0x3E8; //（v1中JA的条件为不可大于0x3e8

byte K[10] = { 0x11,0x22,0x33,0x9C,0x55,0x66,0x77,0x88,0x99,0xAA };// 密码字符串转为16进制字节数组、K[3]只可为9C/FC/AC，以9c为例

// 2 通过用户名 生成对应的key数组

char szName[100] = { 0 };

printf("please input name: ");

scanf\_s("%s", szName, 50);

DWORD dwKey = EncodeUserName(szName, 1, 0, dwRet);// 生成与name相对应的key

// 3 需满足的条件

/\*

013BDDBB 8BD0 mov edx, eax ; edx = ret

013BDDBD 83C4 10 add esp, 0x10

013BDDC0 3855 E0 cmp byte ptr [ebp-0x20], dl ; K[4]&0xFF == ret&0xFF

013BDDC3 0F85 81000000 jnz 013BDE4A

这里需要 K[4] == ret & 0xFF;

013BDDC9 8BCA mov ecx, edx ; ecx = ret

013BDDCB C1E9 08 shr ecx, 0x8 ; ecx = ret >>0x8

013BDDCE 3AF9 cmp bh, cl ; K[5] == (ret >>0x8) &0xFF

013BDDD0 75 78 jnz short 013BDE4A

这里需要 K[5] == (ret >>0x8) & 0xFF;

013BDDD2 8BCA mov ecx, edx ; ecx = ret

013BDDD4 C1E9 10 shr ecx, 0x10 ; ecx = (ret>>0x10)

013BDDD7 384D E2 cmp byte ptr [ebp-0x1E], cl ; K[6] & 0xFF == (ret>>0x10)&0xFF

这里需要 K[6] & 0xFF == (ret >>0x10) & 0xFF;

013BDDDA 75 6E jnz short 013BDE4A

013BDDDC C1E8 18 shr eax, 0x18 ; eax = ret << 0x18

013BDDDF 3845 E3 cmp byte ptr [ebp-0x1D], al ; K[7] & 0xFF == ret << 0x18&0xFF

013BDDE2 75 66 jnz short 013BDE4A

这里需要 K[7] & 0xFF == (ret >> 0x18) & 0xFF;

\*/

K[4] = dwKey & 0xFF;

K[5] = dwKey >> 0x8 & 0xFF;

K[6] = dwKey >> 0x10 & 0xFF;

K[7] = dwKey >> 0x18 & 0xFF;

// 4 穷举-K[0],k[6]已经算出

/\*

\* 013BDC94 E8 AB9904FF call 00407644 这个call处理了

AL= (k[0]^k[6]^0x18 + 0x3D) ^ 0xA7

\*/

while (true)

{

// 随机生成k0（小于0xFE

K[0] = rand() % 0xFF;

// 使用k0、k6构造 al

byte al = (K[0] ^ K[6] ^ 0x18 + 0x3D) ^ 0xA7;

// 若满足第一个JE前的条件，则保存

if (al >= 9)

{

break;

}

}

// 5 穷举剩余的字节 - 1、2

/\*

第二个call中：处理 K[1]、K[7]、k[2]、k[5]

esi = (((K[1] ^ K[7]) ^ 0xFF) \* 0x100 + k[2] ^ k[5] & 0xFF) & 0xFFFF

eax =(((eax^0x7892)+0x4d30)^0x3421) &0xFFFF

edx = 余数；判断是否为0: 为0返回eax=商、不为0返回eax=0

\*/

while (true)

{

// 随机生成1、7、2、5,范围在0-FE

K[1] = rand() % 0xFF;

K[2] = rand() % 0xFF;

// 使用k1、7、2、5构造esi，即第二个call传入的参数，在call内部先提取到eax中

DWORD esi = (((K[1] ^ K[7]) & 0xFF) \* 0x100 + K[2] ^ K[5] & 0xFF) & 0xFFFF;

// 通过传入的参数构造eax，即后面除法中的被除数

DWORD eax = (((esi ^ 0x7892) + 0x4d30) ^ 0x3421) & 0xFFFF;

// 余数为0返回商，也就满足返回值不等于0，同时指定 商即返回值<= 0x3EB（同时满足后面两个条件，则保存

if (eax % 0xB == 0 && eax / 0xB == dwRet)//v1中 <= 0x3EB，v2中 == dwRet = rand() % 0x3E8，同义

{

break;

}

}

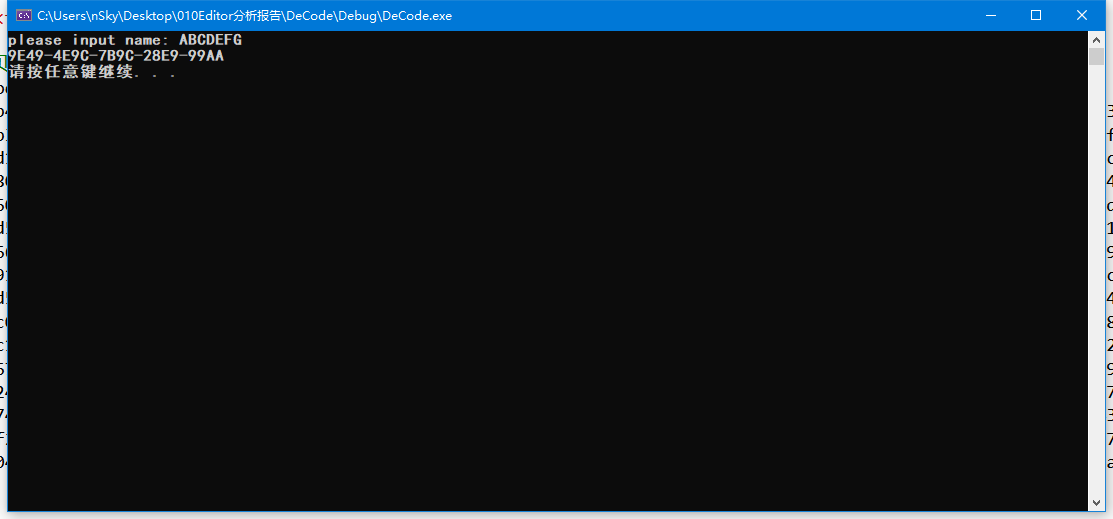
// 6 输出符合条件的序列

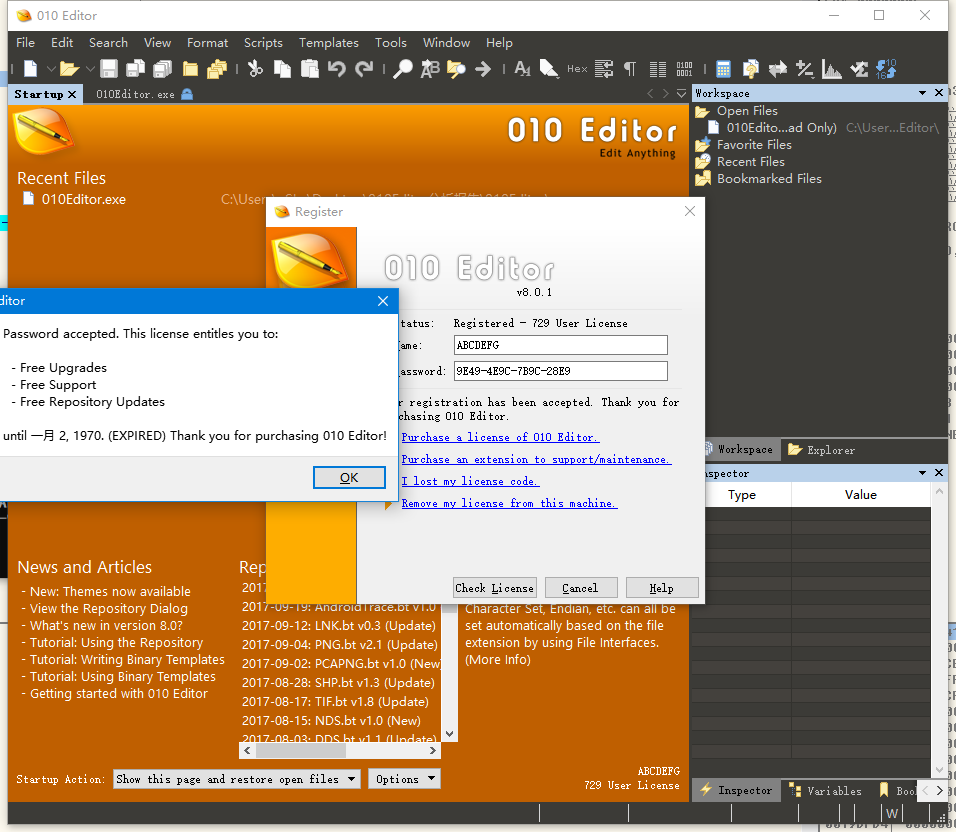
printf("%02X%02X-%02X%02X-%02X%02X-%02X%02X-%02X%02X\n", K[0], K[1], K[2], K[3], K[4], K[5], K[6], K[7], K[8], K[9]);

system("pause");

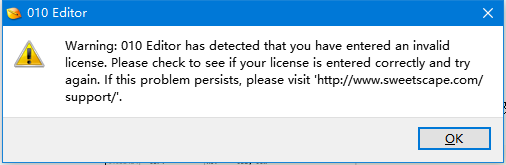
return 0;

}



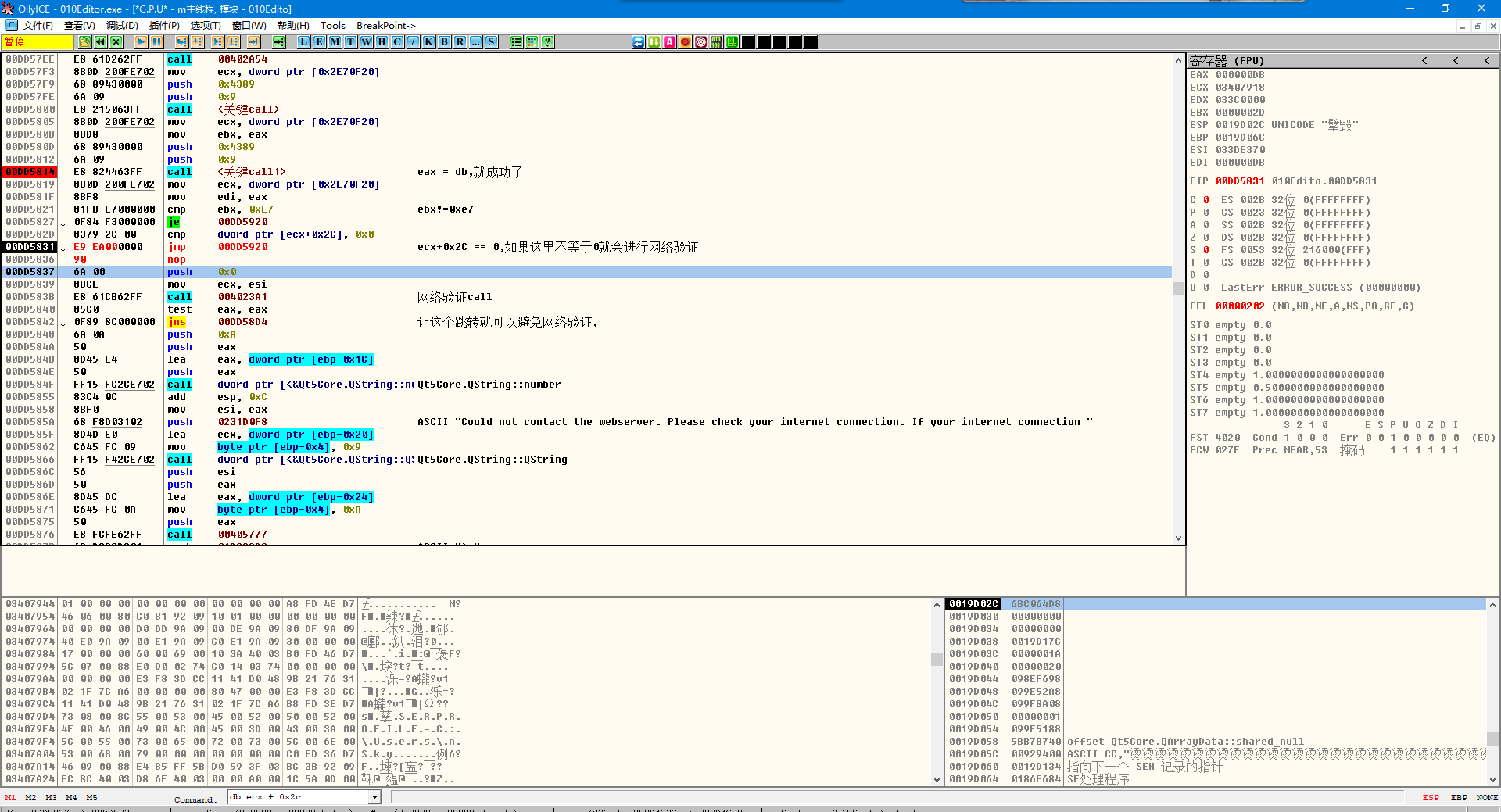


## **2.5 网络验证**

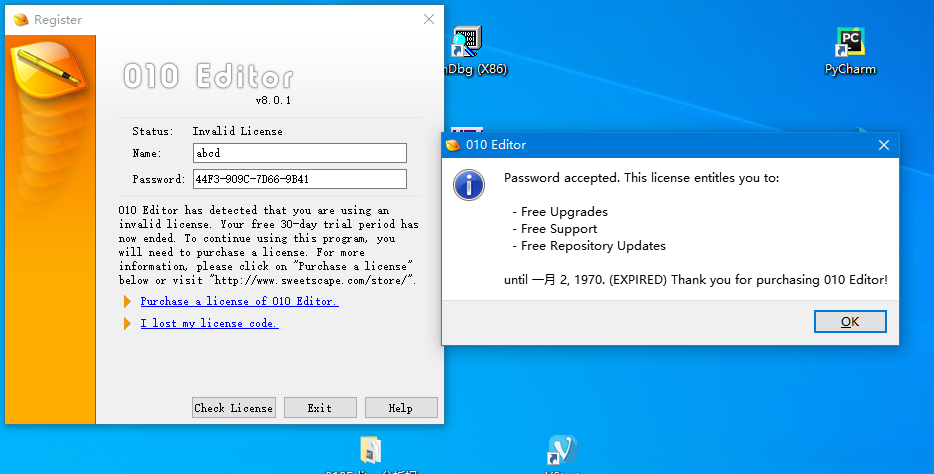


010Editor还进行了网络验证。推测应该是联网的时候把我们序列号发送到服务器进行验证，如果服务器没有我们的序列号就会报错。





这里出来还会进行网络验证，我们不能让他验证，直接jmp。



参考文献

[1] 15PB视频课程-逆向工程实战之010Editor

致 谢

感谢15PB信息安全研究院。