

西普实验吧部分逆向题writeup（二）

原创

[caterpillarous](#) 于 2015-12-24 13:07:30 发布 4709 收藏 2

分类专栏: [逆向之路](#) 文章标签: [CTF 二进制 逆向 汇编](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/caterpillarous/article/details/50393804>

版权



[逆向之路](#) 专栏收录该内容

5 篇文章 0 订阅

订阅专栏

本博客已经弃用, 我的新博客地址: <http://jujuba.me/>

1. 考验你能力的时候了

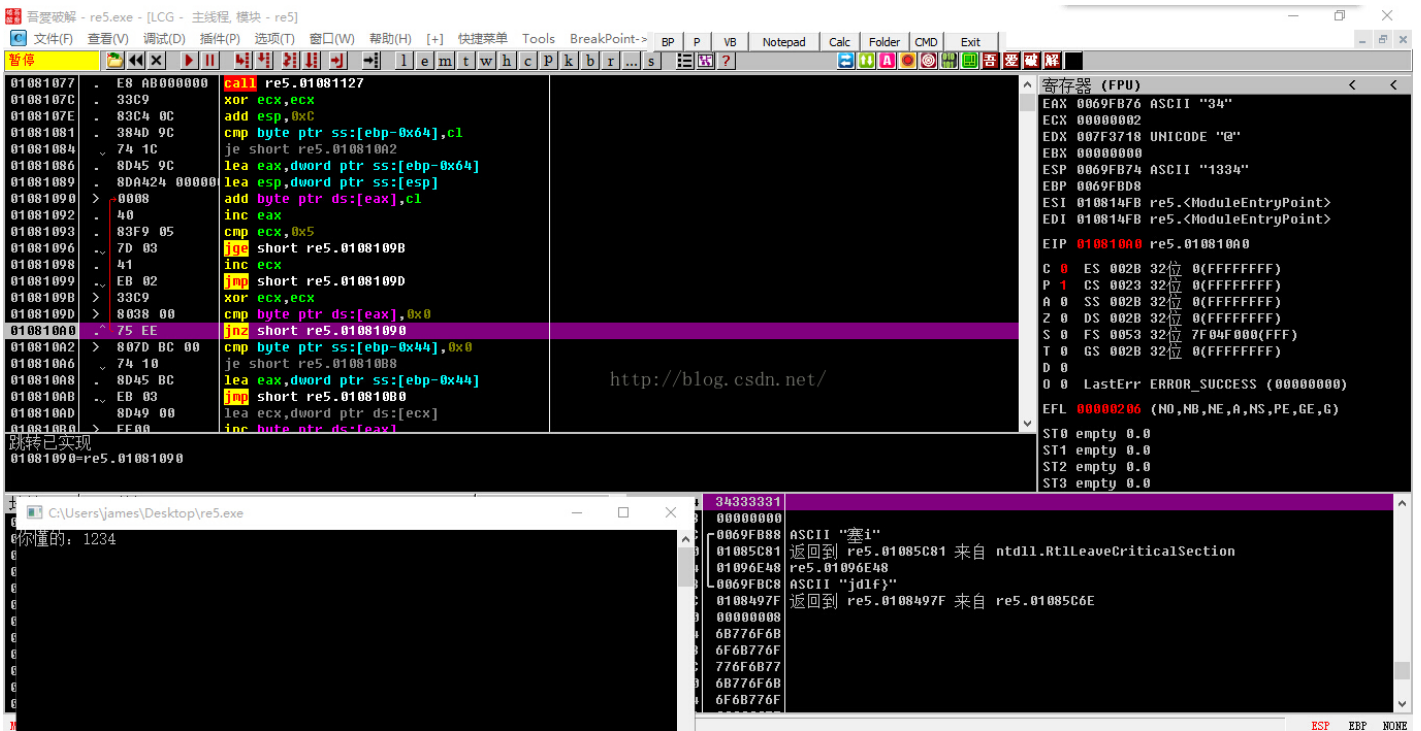
老规矩 PEID查壳 显示什么也没找到 先别管 拖到OD里搜索字符串看看

```
movq mm0, qword ptr ds: [0x1093E34]
movzx eax, word ptr ds: [0x1093E4C]
mov eax, dword ptr ds: [0x1093E60]
movd dword ptr ds: [0x1093E44], mm0
movzx eax, word ptr ds: [0x1093E64]
db 0F
db 68
push re5_01093E74
push re5_01093E78
push re5_01093E98
push re5_01093EB0
push re5_01090198
mov [local.4], re5_010901A0
mov ecx, re5_010901A4
call re5_01085304
push re5_010901D4
push re5_010901D8
push re5_010901E0
push re5_010901E0
movsx eax, byte ptr ds: [eax+0x10901E0]
push re5_01090300
push re5_01090318
push re5_01090CC8
push re5_01090CFC
push re5_01090D2C
push re5_01090D34
push re5_01090D40
push re5_01090D8C
push re5_01090DA8
push re5_01090DB4
push re5_01090DBC
push re5_01090DC8
push re5_01090DD4
push re5_01090DF0
push re5_01090E00
D\VF\XK{I_g45tI(oNs|Hbjdlf}
}
owkow
s|Hbjdlf}
w
kwwkwwkwwkwwkwwkwwkww
你懂的:
Ws
flag不对哟, 再试试呀, 加油! \n
这就是我要的flag!!! \n
pause
COMSPEC
/c
cmd.exe
(initial CPU selection)
.\
PATH
http://blog.csdn.net/
.\
m
CorExitProcess
R
<
.\n
M
k
FlsAlloc
FlsFree
FlsGetValue
FlsSetValue
InitializeCriticalSectionEx
CreateEventExW
CreateSemaphoreExW
```

可以发现最上面的字符串, 其实挺像KEY的, 输入试试, 发现这个并不是KEY。

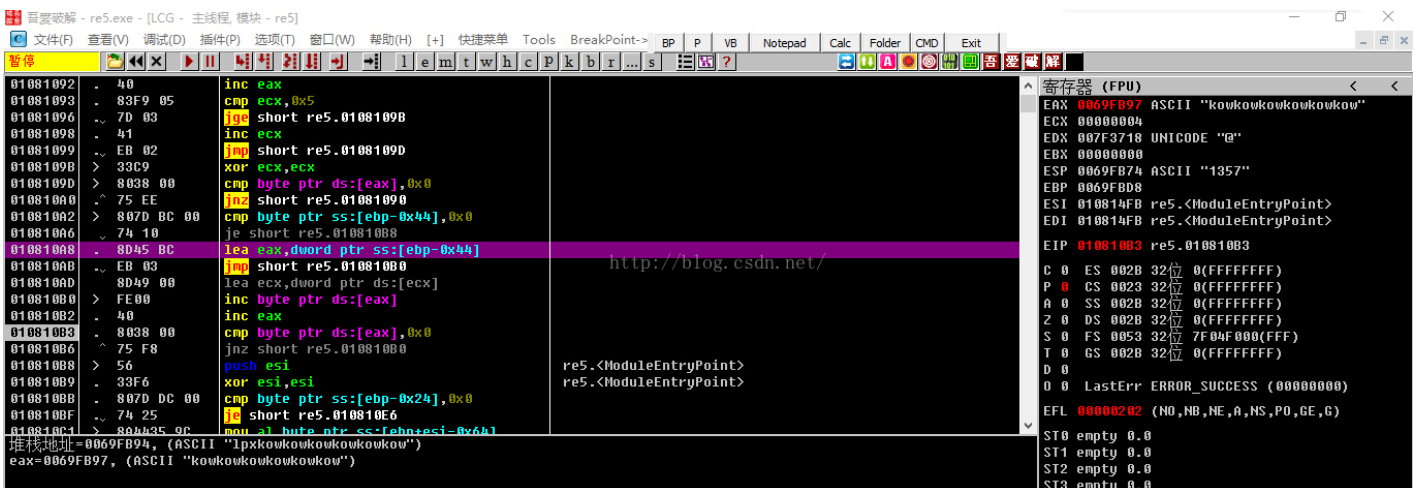
进入字符串“这就是我要的flag!!!”上方下个断点, 先让程序跑一下, 然后断下来开始单步调试

单步了几下发现一个字符串的处理, 我输入的是“1234”

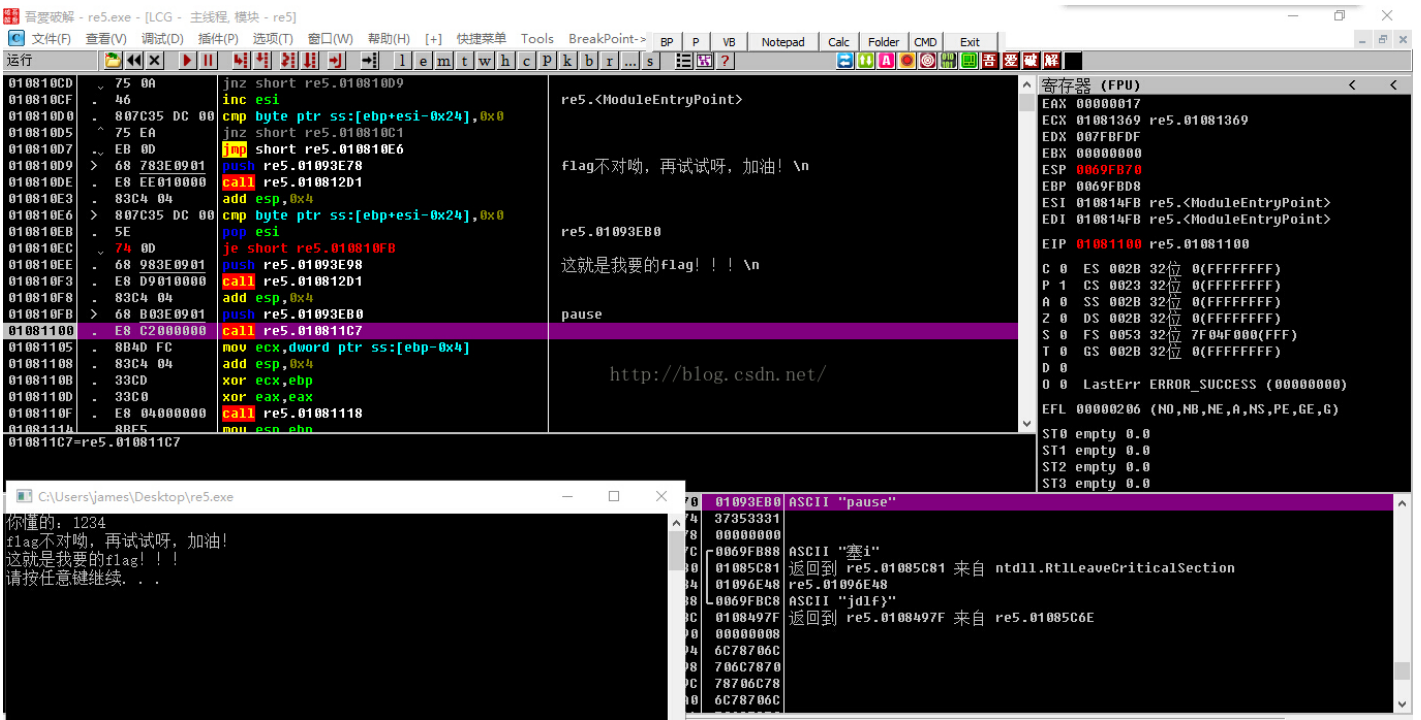


注意左上角的EAX寄存器，我运行了几下发现程序在逐个处理我输入的字符串，这个循环我光看汇编看不懂，等会到IDA里分析去

再向下单步

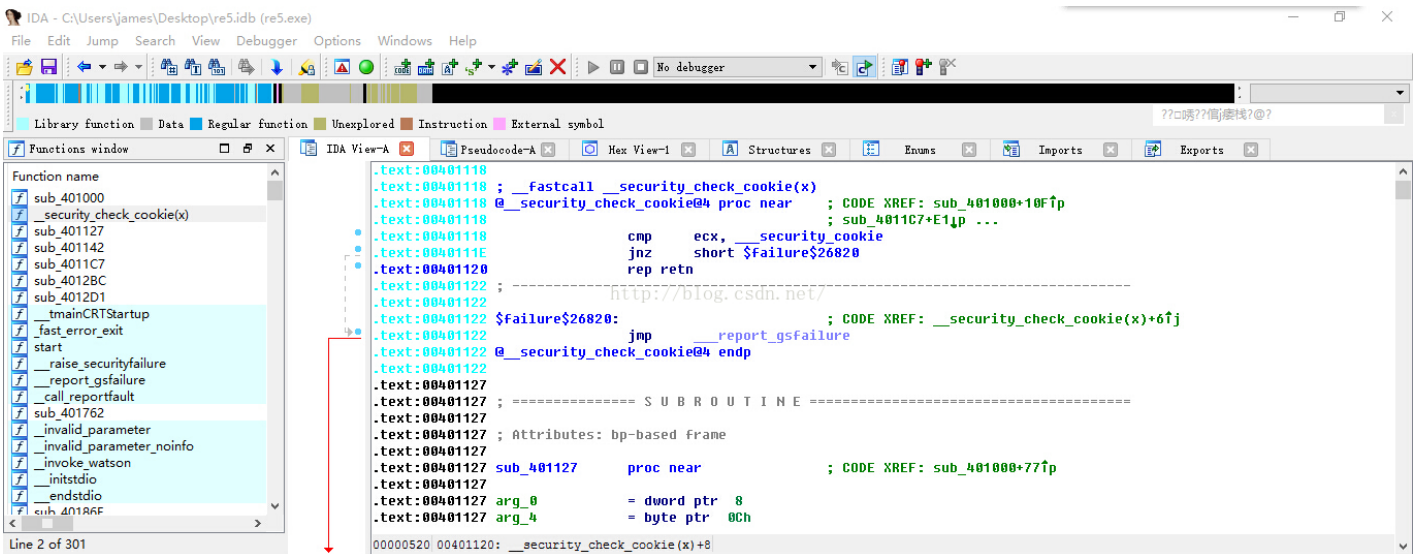


又来到了一个字符串，这个字符串之前见过，7个kow，单步发现程序把kow每个字符+1，变成了7个lpx



程序运行结束，顺手爆破了一下，并没有什么卵用

接下来用IDA分析



注意进去之后会看到这个_security_check_cookie(x)函数.....我以前不懂，还以为这肯定就是存key的地方，后来查了查才知道这是为了保护堆栈平衡设立的，与key无关。

我们F5反汇编一下就可以看到程序大致怎么处理我们的字符串的：

```
int __usercall sub_401000@<eax>(char a1@<sil>)
{
    signed int v1; // ecx@1
    char *v2; // eax@2
    __int128 *v3; // eax@8
    int v4; // esi@10
    char v5; // al@11
    char v7; // [sp-4h] [bp-68h]@10
    char ourinput[32]; // [sp+0h] [bp-64h]@1
    __int128 v9; // [sp+20h] [bp-44h]@1
    int v10; // [sp+30h] [bp-34h]@1
    __int16 v11; // [sp+34h] [bp-30h]@1
```

```

__int64 v12; // [sp+36h] [bp-2Eh]@1
__int128 v13; // [sp+40h] [bp-24h]@1
__int64 v14; // [sp+50h] [bp-14h]@1
__int16 v15; // [sp+58h] [bp-Ch]@1
int v16; // [sp+5Ah] [bp-Ah]@1

v15 = 125;
__mm_storeu_si128((__m128i *)&v13, __mm_loadu_si128((const __m128i *)&xmmword_413E34));
v10 = 1869313903;
__mm_storel_epi64((__m128i *)&v14, __mm_loadl_epi64((const __m128i *)&qword_413E44));
v16 = 0;
__mm_storeu_si128((__m128i *)&v9, __mm_loadu_si128((const __m128i *)&xmmword_413E50));
v11 = 119;
__mm_storel_epi64((__m128i *)&v12, 0i64);
sub_4012D1("你懂的: ", ourinput[0]);
sub_401127("%s", ourinput);
v1 = 0;
if ( ourinput[0] )
{
    v2 = ourinput;
    do
    {
        *v2++ += v1;
        if ( v1 >= 5 )
            v1 = 0;
        else
            ++v1;
    }
    while ( *v2 );
}
if ( (_BYTE)v9 )
{
    v3 = &v9;
    do
    {
        ++*(_BYTE *)v3;
        v3 = (__int128 *)((char *)v3 + 1);
    }
    while ( *(_BYTE *)v3 );
}
v7 = a1;
v4 = 0;
if ( (_BYTE)v13 )
{
    while ( 1 )
    {
        v5 = ourinput[v4];
        if ( !v5 || *((_BYTE *)&v13 + v4) != v5 )
            break;
        ++v4;
        if ( !*((_BYTE *)&v13 + v4) )
            goto LABEL_16;
    }
    sub_4012D1("flag不对哟, 再试试呀, 加油! \n", v7);
}
LABEL_16:
if ( !*((_BYTE *)&v13 + v4) )
    sub_4012D1("这就是我要的flag!!! \n", ourinput[0]);
sub_4011C7("pause");

```

```
return 0;
}
```

我把我们输入的字符串叫做ourinput，可以看到处理字符串的函数有三个，两个加密，一个验算答案，两个字符串的处理我们刚才都在OD里看过了，我们来看看具体是怎么处理的

第一个：

```
if ( ourinput[0] )
{
    v2 = ourinput;
    do
    {
        *v2++ += v1;
        if ( v1 >= 5 )
            v1 = 0;
        else
            ++v1;
    }
    while ( *v2 );
}
```

可以看到这个是把我们输入的字符串分成6个一组，每组每个字节加上该字节的索引值，这是处理我们输入的字符串的过程

第二个：

```
if ( (_BYTE)v9 )
{
    v3 = &v9;
    do
    {
        ++*(_BYTE *)v3;
        v3 = (__int128 *)((char *)v3 + 1);
    }
    while ( *(_BYTE *)v3 );
}
```

这个就是每个字节加上1，很明显这就是kow变成lpx的过程

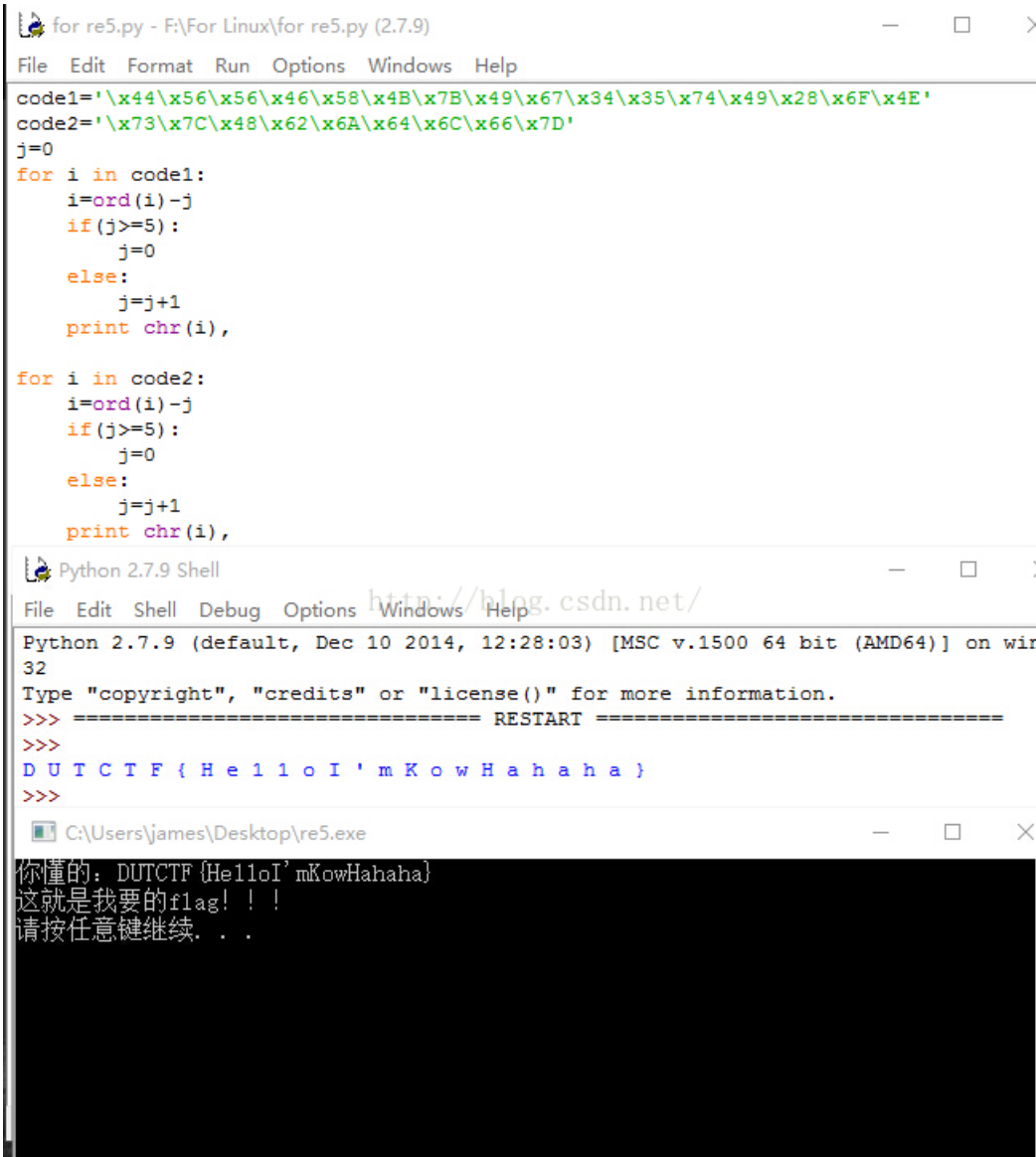
第三个：

```
if ( (_BYTE)v13 )
{
    while ( 1 )
    {
        v5 = ourinput[v4];
        if ( !v5 || *((_BYTE *)&v13 + v4) != v5 )
            break;
        ++v4;
        if ( !*((_BYTE *)&v13 + v4) )
            goto LABEL_16;
    }
    sub_4012D1("flag不对哟，再试试呀，加油！\n", v7);
}
```

可以看到这个是一个验算的函数，而且，这里面根本就没提到第二个函数中所用的字符串，其实那个字符串只是个障眼法

真正的对比字符串是v13，我们可以想办法找到v13的值，然后同样分成6个一组，减去各自的索引值即可。

还原结果如下：



2.bin100(ebCTF 2013)

查壳——C++写成——拖入OD

先打开程序玩一下，发现是个掷骰子游戏，要掷出特定的数字组合3-1-3-1-7。

很明显，永远也掷不出来的嘛。

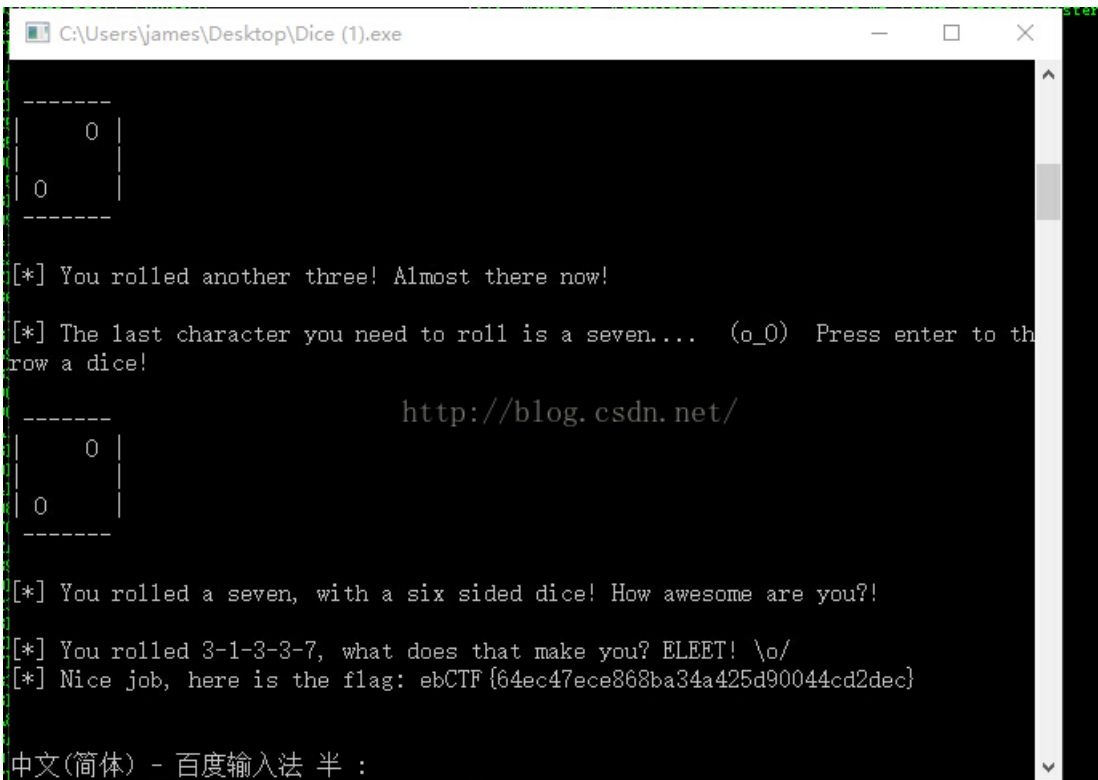
搜一下字符串

这也太有规律了，让我有种想要爆破的冲动，然后嘛，我就爆破了，于是就成功了

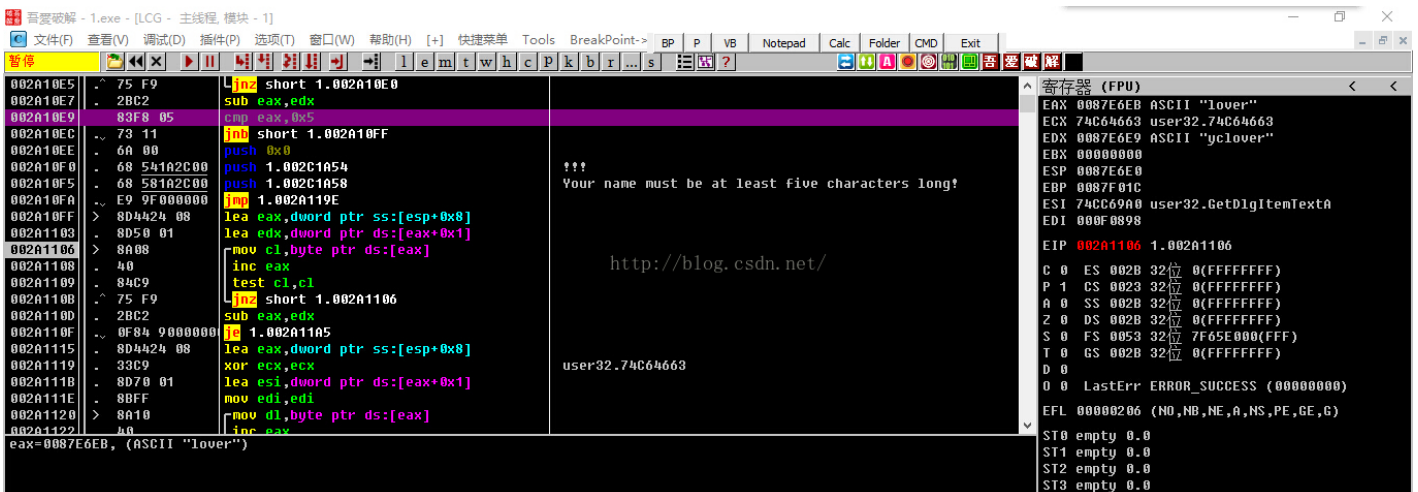
只要把成功的语句：“You rolled a xx!!”上面的跳转改成相反的即可

一共要改5处，改完即可看到key，不过我不知道怎么把key直接复制下来.....反正我是手打的，大神路过还请指教

然后嘛 提供个方法，可以在每个成功的语句的跳转前面那一句断下来，然后开始跑程序，此时直接可以看到你掷骰子的结果，假如不对直接把下面那个跳转改了，然后取消断点，断下一个成功的语句，就是这种作弊的感觉.....key就到手了



3.你知道注册码吗?



拖到OD里一看，有个查字符串长度的函数，就断在这里吧。

程序跑到这里暂停，取消断点，开始单步调试。

这里说一个心得：就是OD里会用小箭头形象地显示跳转，一般在目标字符串附近看到向上的跳转，此时就要留意了，因为它很有可能就是一个for循环，用来逐字符处理我们输入的字符串，从而跟真正的key做比对，在本题中就是如此。

这里我试着分析还是失败了.....所以我还是用IDA吧

```
GetDlgItemTextA(a1, 1001, &String, 1024);
GetDlgItemTextA(a1, 1002, &v7, 1024);
if ( strlen(&String) < 5 )
{
    v4 = "!!!";
    v3 = "Your name must be at least five characters long!";
    return MessageBoxA(a1, v3, v4, 0);
}
result = strlen(&String);
if ( result )
{
    for ( i = 0; i < (signed int)strlen(&String); ++i )
    {
        if ( *(&v7 + i) != i + *(&String + i) - strlen(&String) )
            break;
    }
    result = strlen(&String);
    if ( i == result )
    {
        v4 = "good job";
        v3 = "Yeah, you did it!";
        return MessageBoxA(a1, v3, v4, 0);
    }
}
```

可以看到程序是怎么处理我们的字符串的，字符串每个字符加上它们的索引值并减去输入字符串的长度，那么就是8
结果如下：

