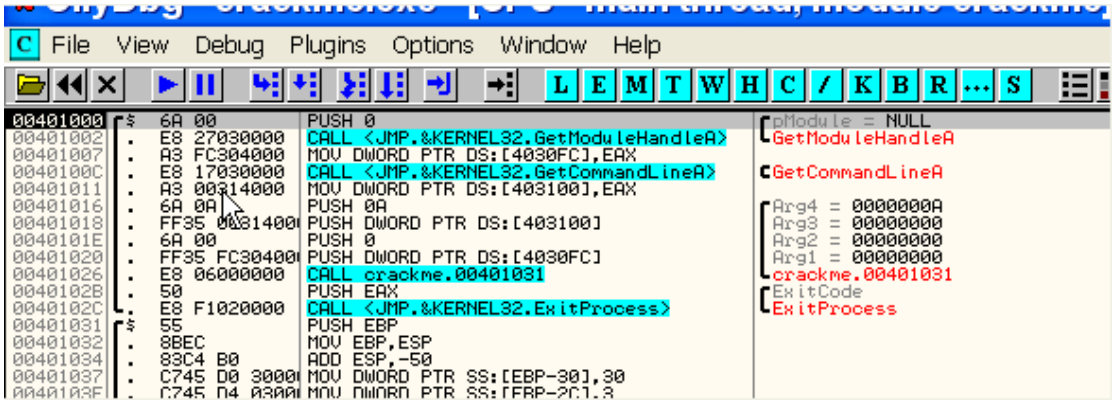
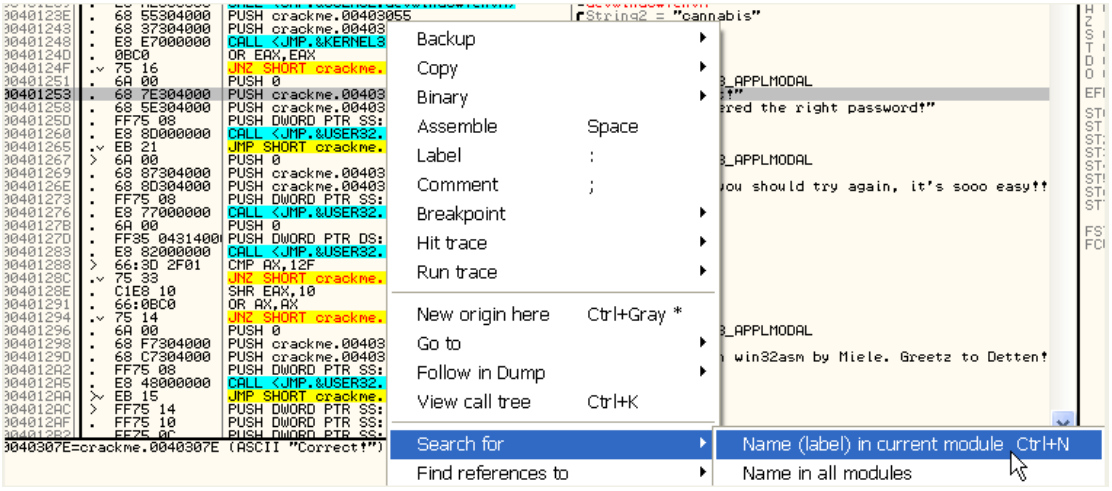
# 第十四章-硬编码序列号寻踪-Part2

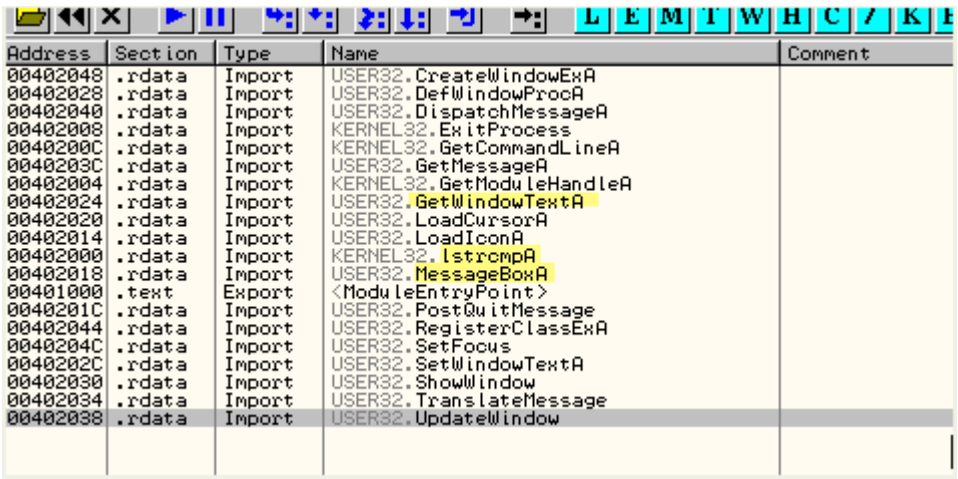
首先我来解答一下上一章留下的那个 CrackMe。




用 OD 加载名为 mielecrackme1 的 CrackMe。断在了入口点处,我们单击鼠标右键选择-Search for-Name(label)in current module 看看该 CrackMe 使用了哪些 API 函数。




下面是找到的 API 函数列表:

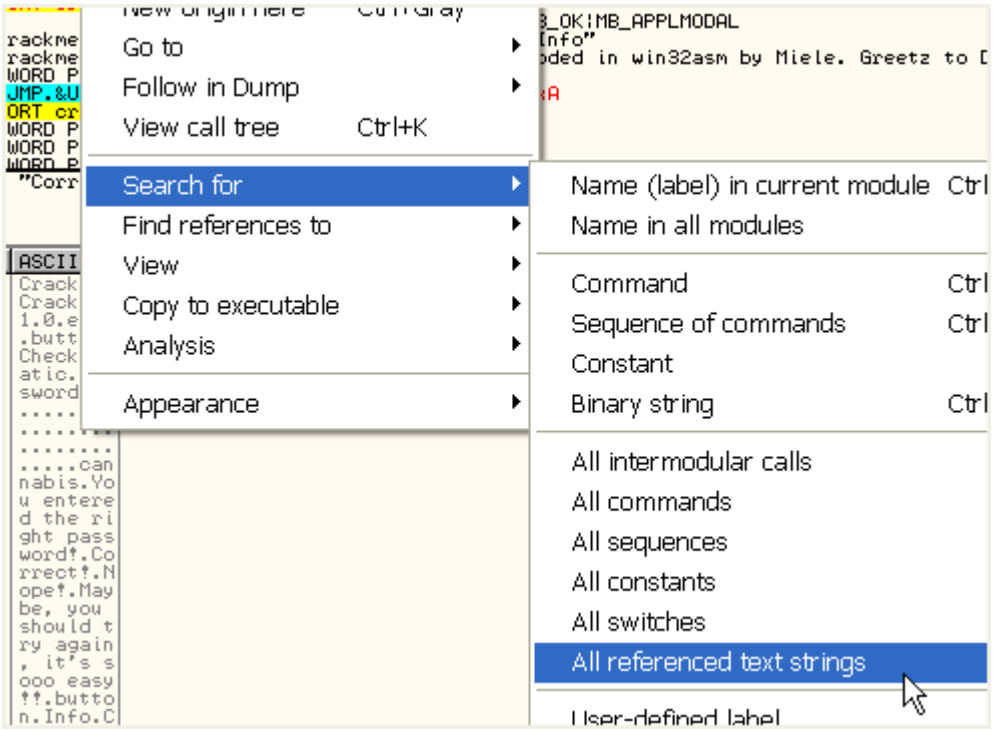


| Address | Section | Type | Name | Comment |
|---|---|---|---|---|
| 00402048 | .rdata | Import | USER32.CreateWindowExA | |
| 00402028 | .rdata | Import | USER32.DefWindowProcA | |
| 00402040 | .rdata | Import | USER32.DispatchMessageA | |
| 00402008 | .rdata | Import | KERNEL32.ExitProcess | |
| 0040200C | .rdata | Import | KERNEL32.GetCommandLineA | |
| 0040203C | .rdata | Import | USER32.GetMessageA | |
| 00402004 | .rdata | Import | KERNEL32.GetModuleHandleA | |
| 00402024 | .rdata | Import | USER32.GetWindowTextA | |
| 00402020 | .rdata | Import | USER32.LoadCursorA | |
| 00402014 | .rdata | Import | USER32.LoadIconA | |
| 00402000 | .rdata | Import | KERNEL32.lstrcmpA | |
| 00402018 | .rdata | Import | USER32.MessageBoxA | |
| 00401000 | .text | Export | <ModuleEntryPoint> | |
| 0040201C | .rdata | Import | USER32.PostQuitMessage | |
| 00402044 | .rdata | Import | USER32.RegisterClassExA | |
| 0040204C | .rdata | Import | USER32.SetFocus | |
| 0040202C | .rdata | Import | USER32.SetWindowTextA | |
| 00402030 | .rdata | Import | USER32.ShowWindow | |
| 00402034 | .rdata | Import | USER32.TranslateMessage | |
| 00402038 | .rdata | Import | USER32.UpdateWindow | |

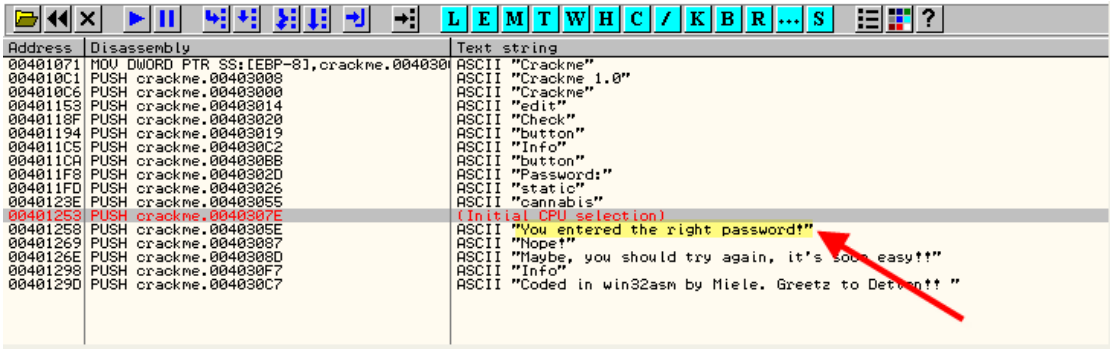
其中有几个 API 函数比较重要,GetWindowTextA:获取用户输入的序列号。lstrcmpA,上一章最后提到的,用于字符串的比较。MessageBoxA:用于显示一条消息,提示是正确或者错误的序列号。

我们可

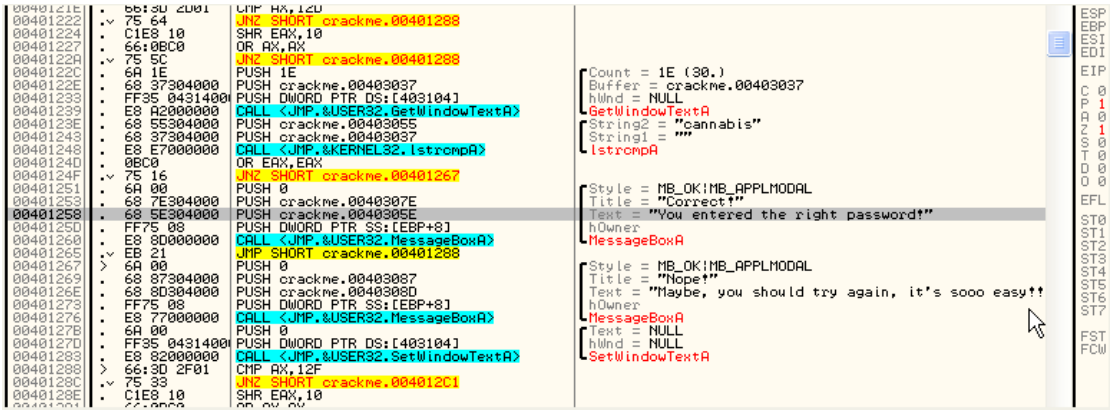
以给这几个 API 设置断点,当我们输入错误的序列号的时候,就会断下来。但这里我们使用更加简单,快速的做法,我们来看看程序中使用的字符串。




我们通过在反汇编窗口中单击鼠标右键选择-Search for-All referenced text strings 打开字符串列表窗口。



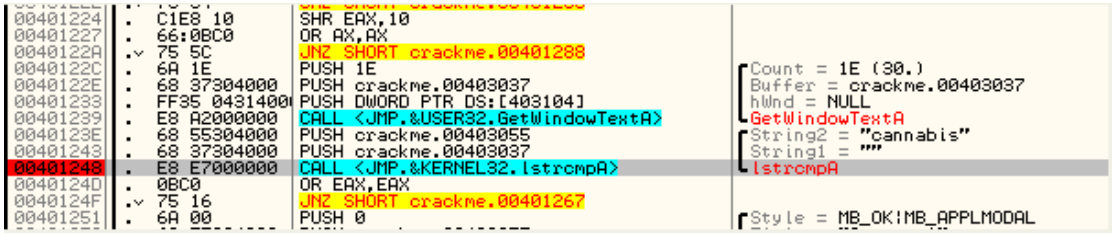

我们可以看到上面列表中有提示成功以及失败的字符串。如果我们在该字符串上面双击鼠标左键,就可以来到 MessageBoxA 调用代码附近。现在我们在”You entered the right password!”字符串上面双击鼠标左键。



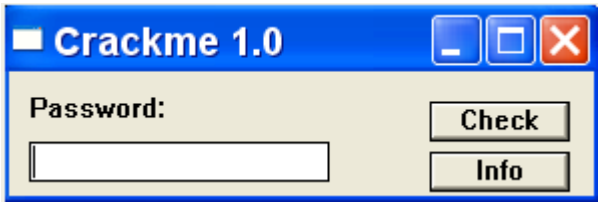

来到了比较关键的地方。

首先是 GetWindowTextA 获取用户输入的序列号,然后 lstrcmpA 将输入序列号与正确的序列号进行比较,相同的话就 MessageBoxA 显示”You entered the right password”的提示框,如果不相同就 MessageBoxA 提示”Maybe, you should try again,it’s sooo easy!!”的提示框。

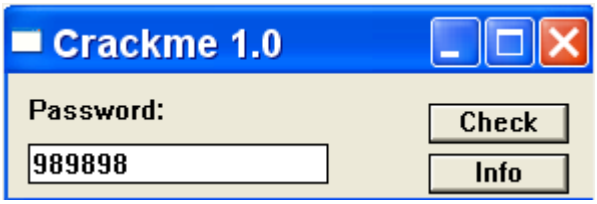
所以我们给 lstrcmpA 函数设置一个断点,看看是如何进行比较的。



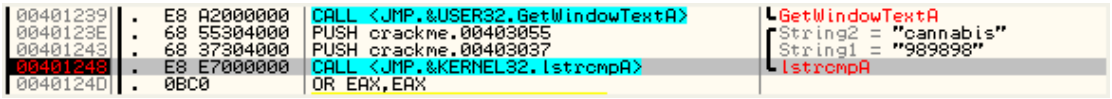

按 F9 键运行起来。




我们在弹出的窗口中随意输入一个序列号,例如:这里我们输入 989898。

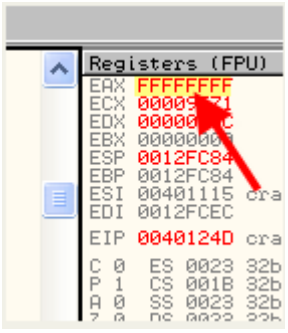



单击 Check 按钮,就会断在我们刚刚设置的断点处。

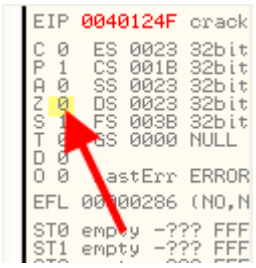



我们可以看到 OD 中显示的参数,进行比较的两个字符串,分别是”989898”和”cannabis”。

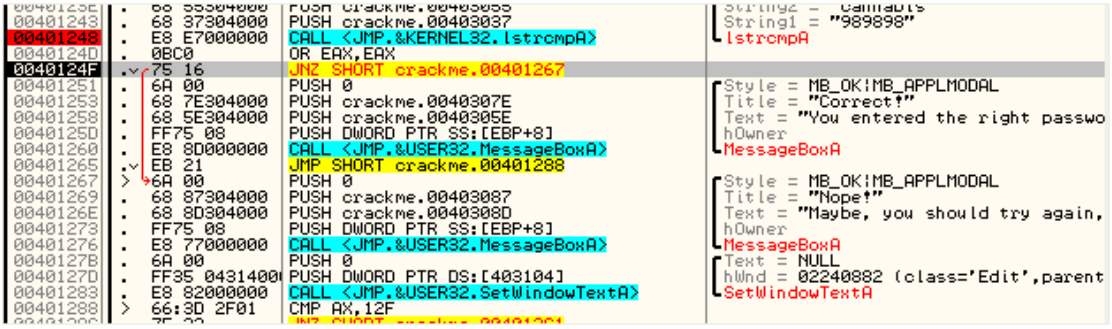
我们按 F8 键单步步过这个 API 调用。



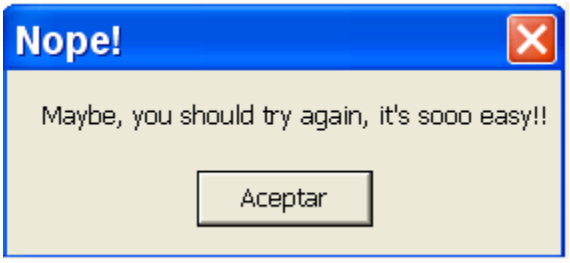

EAX 中存放的返回值为 FFFFFFFF,意味着比较的两个字符串不相同。



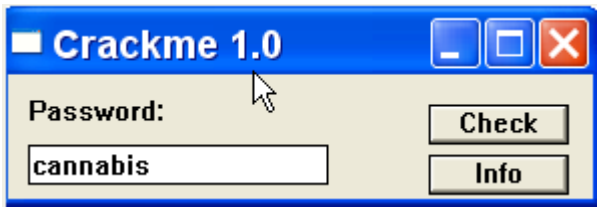

因为比较的结果不为零,所以零标志位 Z 不置位。JNZ 跳转就会实现。(记住:JNZ 当零标志位 Z 置 0 跳转,置 1 不跳转)



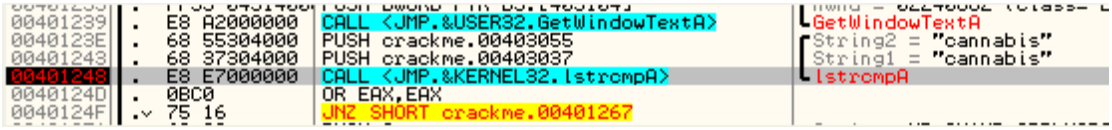

跳转实现以后就弹出一个错误消息框。所以,与我们输入序列号进行比较的”cannabis”就是正确的序列号。我们继续运行程序。



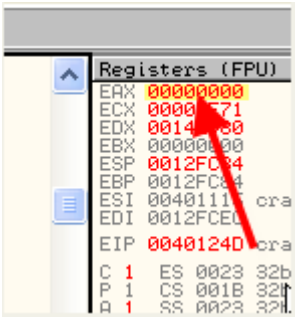

单击 OK,我们回到主窗口,输入正确的序列号”cannabis”。



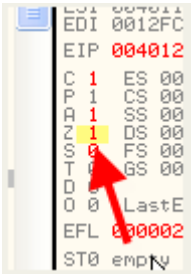

单击 Check 按钮,依然断在了我们设置的断点处。



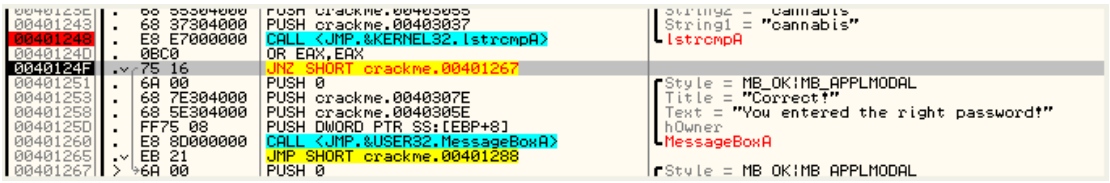

可以看到待比较的两个字符串是相同的,按 F8 键单步步过这个 API 函数。




因为两个字符串是相同的,所以 EAX 中保存的返回值为 0,并且零标志位 Z 置 1。




现在 JNZ 跳转将不会实现。

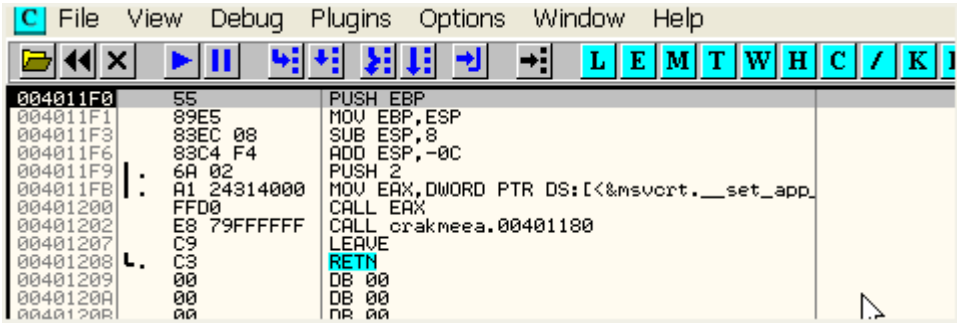



我们直接运行程序,将会弹出一个显示序列号正确的消息框。

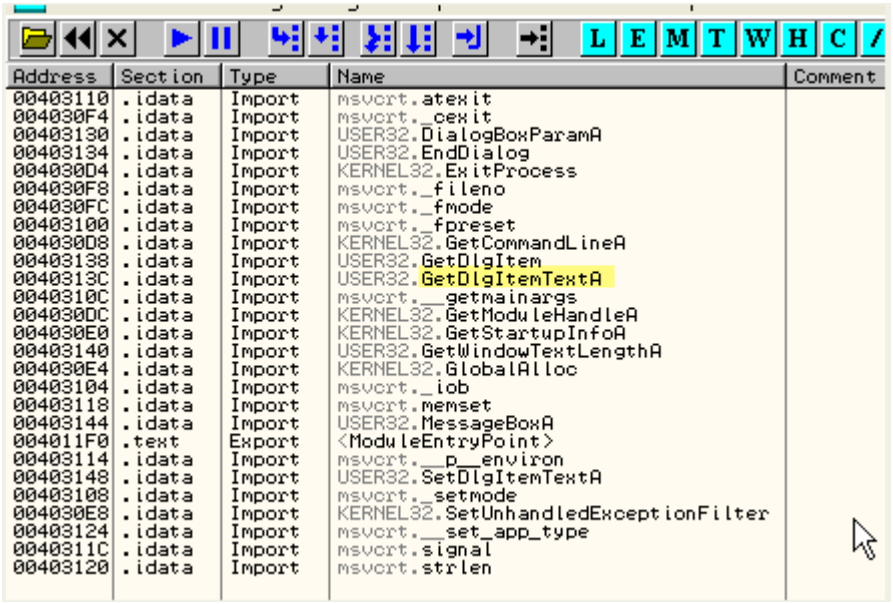

这就是第 13 章遗留下的 CrackMe。正确的序列号是”cannabis”。

我们再来看一个更加复杂的硬编码的 CrackMe(比之前的两个复杂)。

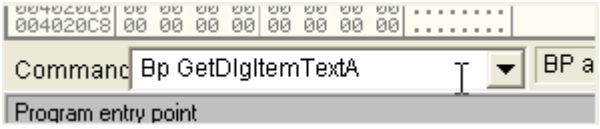
这个 CrackMe 并不是序列号直接进行比较。用 OD 加载这个名为”crakmeeasy”的 CrackMe。



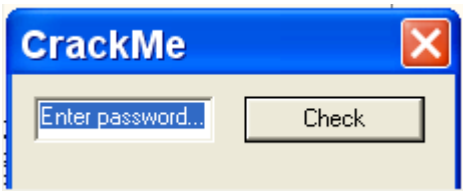

我们像之前一样查看 API 列表,该列表中有 GetDlgItemTextA,我们给这个函数设置一个断点。



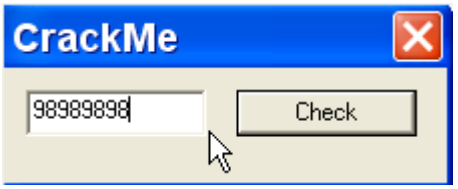

我们在命令栏窗口中输入 bp GetDlgItemTextA。



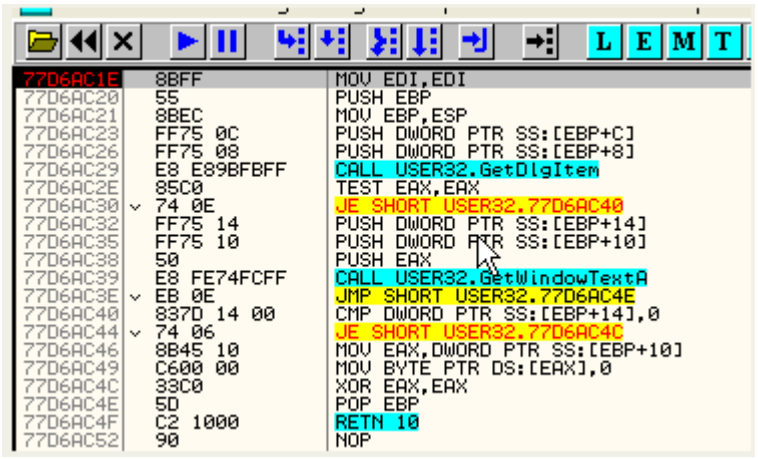

我们现在按 F9 键运行程序并输入序列号。



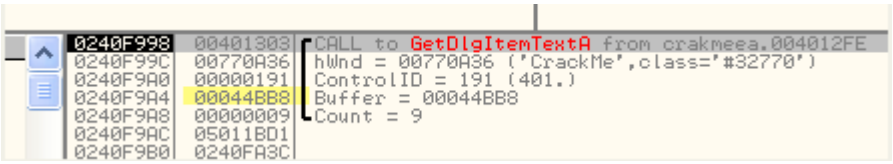

随便输入一个错误的序列号



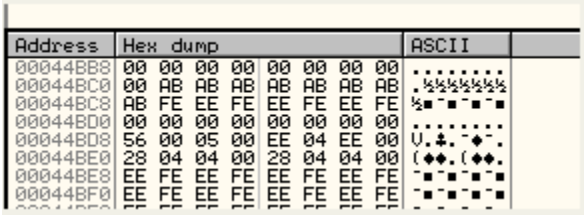

单击 Check 按钮,断在我们刚刚设置的断点处。



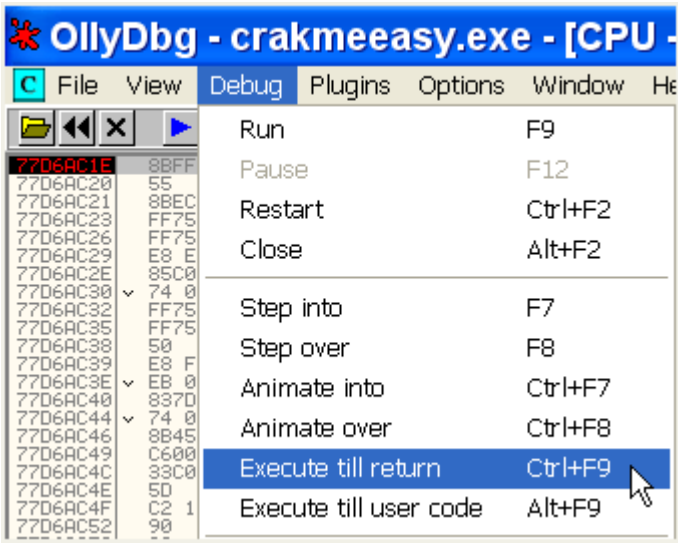

我们来看看堆栈中参数。



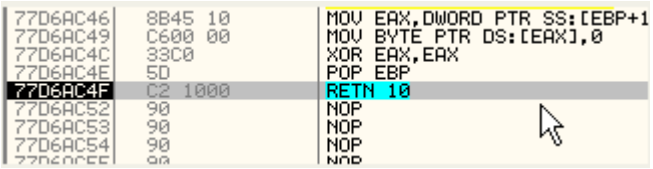

这里我们可以注意到 Buffer 参数:用于保存用户输入的序列号,我们在这个参数上单击鼠标右键选择-Follow in Dump。



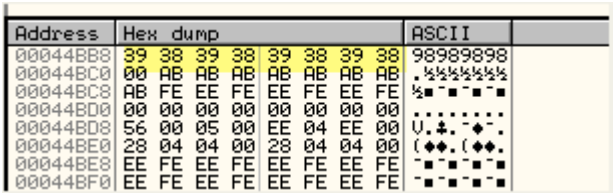

这里缓冲区是空的,因为该函数还没有执行,我们选择主菜单项-Debug-Execute till return。



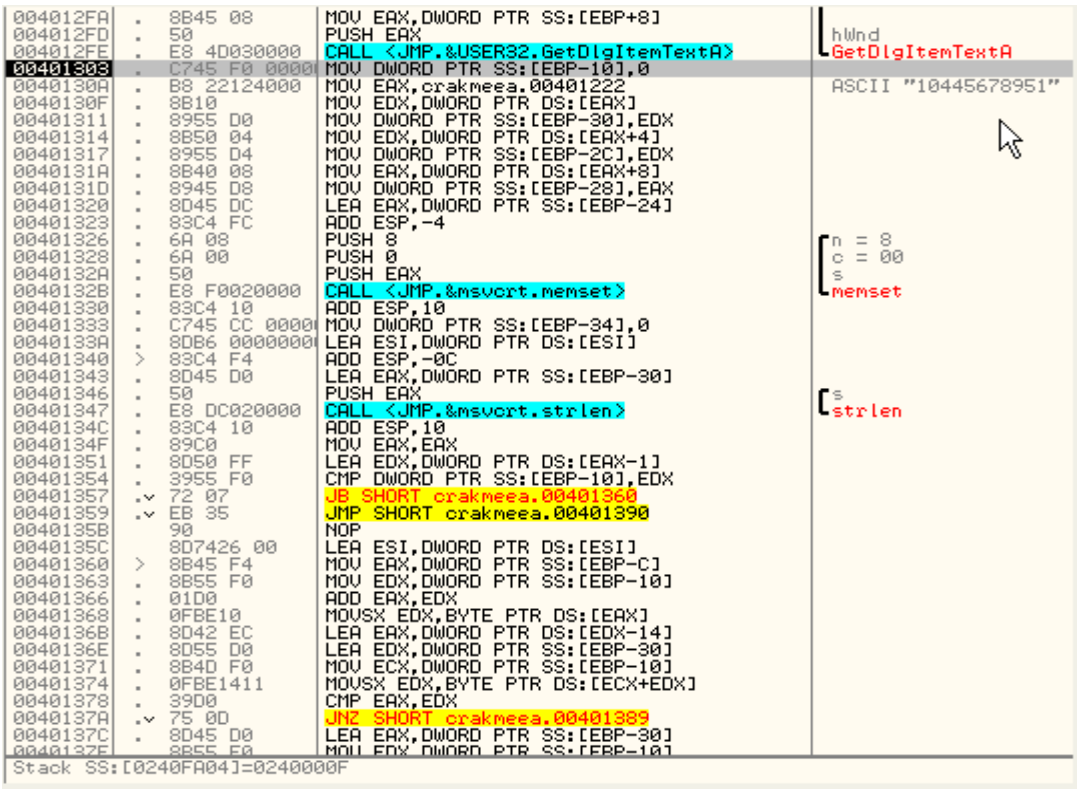

执行到函数返回。



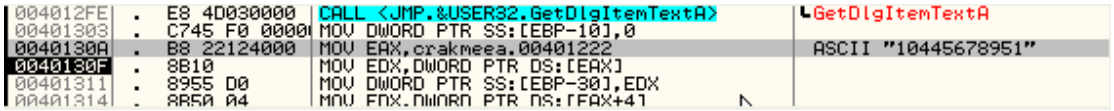

我们按下 F7 键,返回到主程序代码中。



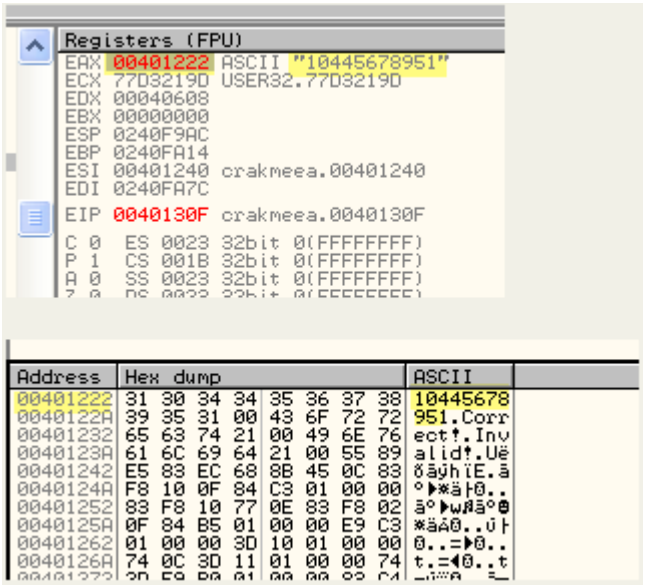

我们可以看到,缓冲区里面保存了错误的序列号。




这里我们看到了一长串数字字符串。有些人可能会问这是正确的序列号吗?呵呵,我也不知道。




这里 EAX 保存了 401222 这个常量地址,该地址指向一个固定的字符串。

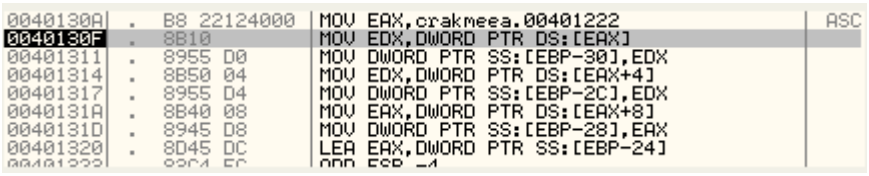



单步一行,EAX 就等于 401222 了。

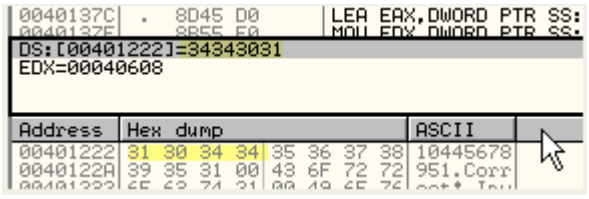
MOV EDX,DWORD PTR DS:[EAX]

等价于:

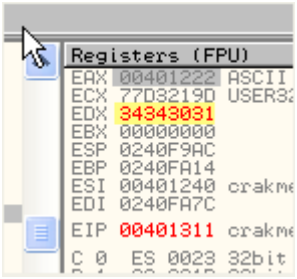
MOV EDX,DWORD PTR DS:[401222]



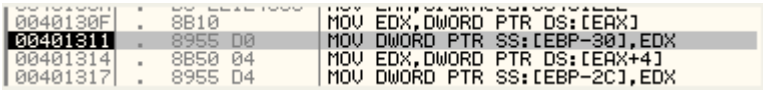

该指令将 401222 地址处内容保存到 EDX 中,OD 中的提示窗口中提示为 10445678951 字符串的首 4 个字节。



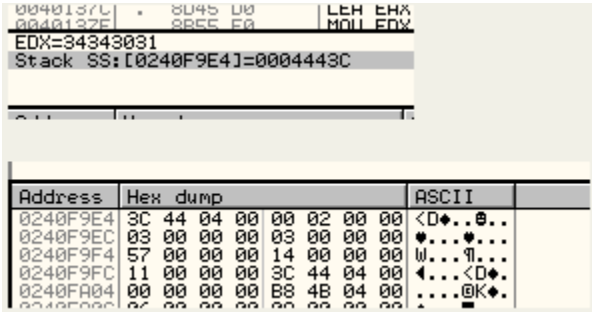

我们按 F7 键将 401222 处内容保存 EDX 中(寄存器中保存的数值和内存中的存放顺序是相反的)。



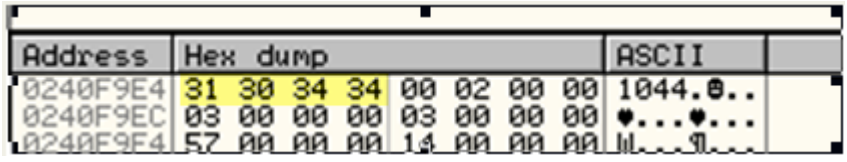

下一条指令将 EDX 的内容保存到[EBP-30]堆栈空间中。



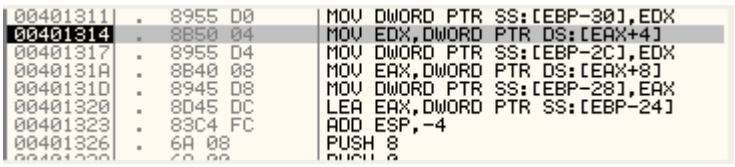

OD 的提示窗口显示,[EBP-30]在我的机器上是 240f9e4,我们在数据窗口中转到该地址。



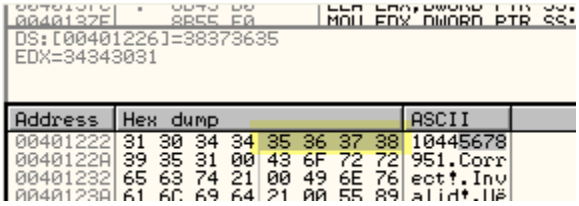

我们按 F7 键将 EDX 的内容保存到[EBP-30]内存单元中。



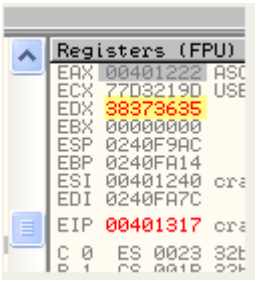

于是



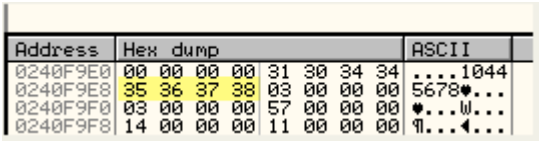

将常量数字字符串的接下来 4 个字节保存到 EDX 中。



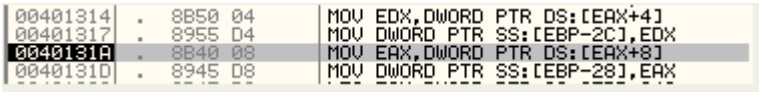

OD 的显示窗口显示,[EAX+4]的对应的地址为 401226,按 F7 键,接下来的 4 个字节被保存到 EDX 中。



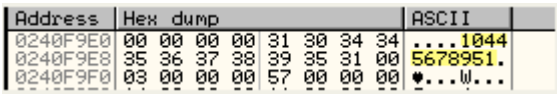

然后和之前的赋值一样。




实际上就是将 4 个字节的内容从一块内存区域拷贝到另一块内存区域。

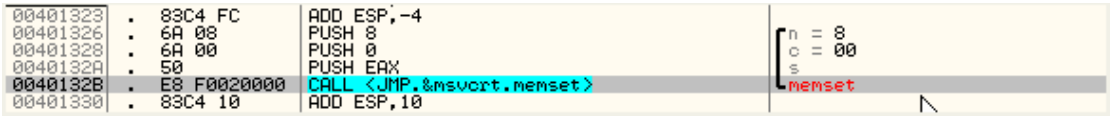



现在是最后的 4 个字节的拷贝。




拷贝完毕。

接下来是调用 memset,我们在 OD 中来看看其参数。

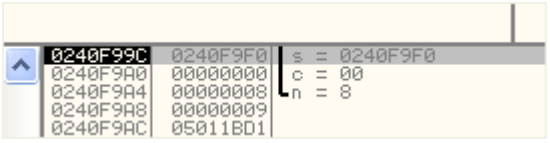



这里有(n,c,s)3 个值。

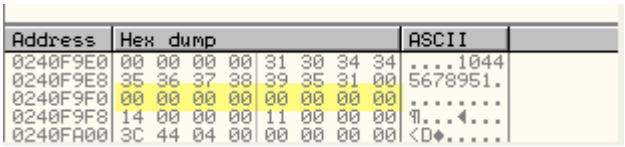
s:待填充的内存单元的起始地址

n:需要填充的字节数

c:待填充的值




我们在堆栈中来看看上述参数:以 240f9f0 为起始地址长度为 8 的内存单元填充零。



按 F8 键单步,我们可以看到一 240f9f0 为起始地址的 8 个字节的内存区域被填充零了。

接下来调用的是 lstrlen 函数,用于计算字符串的长度。

```
00401343  .  8D45 D0      LEA EAX,DWORD PTR SS:[EBP-30]
00401346  .  50           PUSH EAX                              s
00401347  .  E8 DC020000  CALL <JMP.&msvcrt.strlen>             strlen
0040134C  .  83C4 10      ADD ESP,10
```

堆栈中的内容如下:

```
0240F99C  0240F9E4  s = "10445678951"
0240F9A0  00000000
```

是计算起始起始地址为 240f0e4 的字符串的长度(我们知道这个字符串地址)。

按 F8 键执行 lstrlen,EAX 中保存字符串的长度。

```
Registers (FPU)
EAX 0000000B
ECX 0240F9E4 ASCII "10445678951"
EDX 7F303438
EBX 00000000
ESP 0240F99C
EBP 0240FA14
ESI 00401240 crakmeea.00401240
EDI 0240FA7C
EIP 0040134C crakmeea.0040134C
```

我们可以看到长度为 0B,即十进制的 11,就是常量数字字符串的长度。

```
0040134F  .  89C0         MOV EAX,EAX
00401351  .  8D50 FF      LEA EDX,DWORD PTR DS:[EAX-1]
00401354  .  3955 F0      CMP DWORD PTR SS:[EBP-10],EDX
00401357  .v 72 07        JB SHORT crakmeea.00401360
00401359  .v EB 35        JMP SHORT crakmeea.00401390
```

这里将 EAX 的值减去 1 保存到 EDX 中,EDX 就等于 0A 了。

接下来将 EDX(值为 0A)与[EBP - 10](值为 0)进行比较。

```
00401351  .  8D50 FF      LEA EDX,DWORD PTR DS:[EAX-1]
00401354  .  3955 F0      CMP DWORD PTR SS:[EBP-10],EDX
00401357  .v 72 07        JB SHORT crakmeea.00401360
00401359  .v EB 35        JMP SHORT crakmeea.00401390
0040135B     90           NOP
0040135C     8D7426 00    LEA ESI,DWORD PTR DS:[ESI]
00401360  >  8B45 F4      MOV EAX,DWORD PTR SS:[EBP-C]
00401363  .  8B55 F0      MOV EDX,DWORD PTR SS:[EBP-10]
```

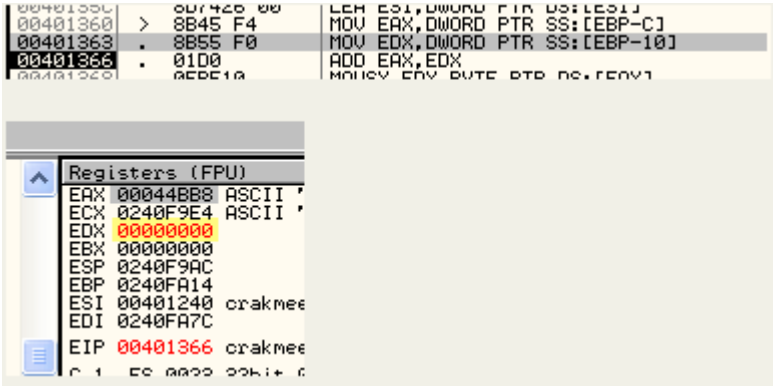
如果 0 小于 0A,就会跳转到 401360。

```
0040135C     8D7426 00    LEA ESI,DWORD PTR DS:[ESI]
00401360  >  8B45 F4      MOV EAX,DWORD PTR SS:[EBP-C]
00401363  .  8B55 F0      MOV EDX,DWORD PTR SS:[EBP-10]
00401366  .  01D0         ADD EAX,EDX
```

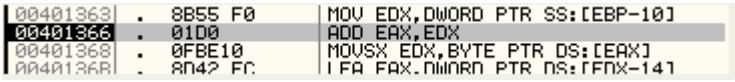
这一行,将我们输入的错误序列号保存到 EAX 中,我们按 F7 键执行,可以看到 EAX 中保存了错误的序列号”98989898”。

```
Registers (FPU)
EAX 00044BB8 ASCII "98989898"
ECX 0240F9E4 ASCII "10445678951"
EDX 0000000A
EBX 00000000
ESP 0240F9AC
EBP 0240FA14
ESI 00401240 crakmeea.00401240
EDI 0240FA7C
EIP 00401363 crakmeea.00401363
C 1  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
```

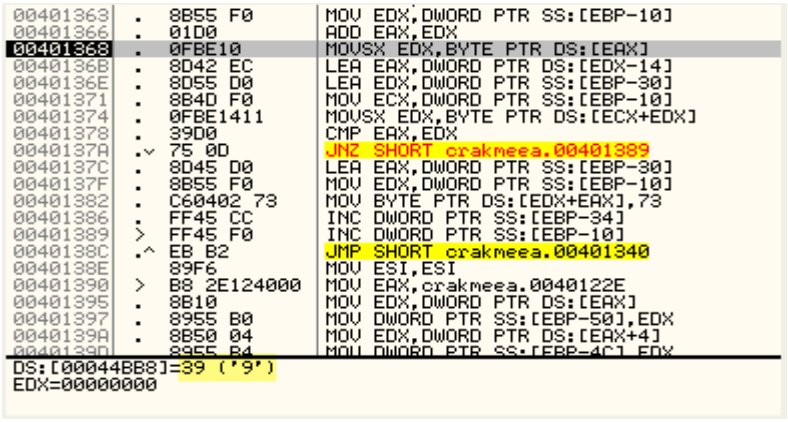
下一条指令,EDX 清零。




接下来一行

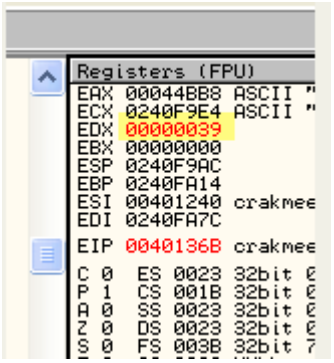



EAX 保存了我们输入的错误序列号的首地址,EDX 的初始值为零,现在创建一个循环,EAX + EDX,然后 EDX 依次递增来获取我们输入序列号的每一个字节。

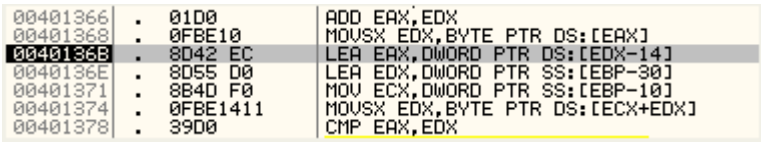



我们知道 MOVSX 指令将指定字节保存到 EDX 中,如果该字节是正数,高位补零,如果该字节为负数高位补 1.

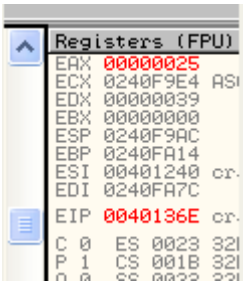
这里,将错误序列号的第一个字节保存 EDX 中,我们单步执行,可以看到 EDX 值变成了 39。



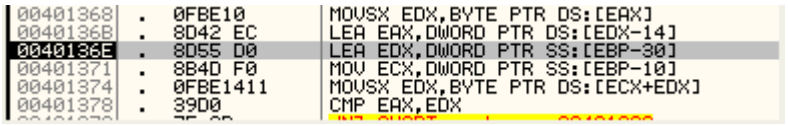

下一条指令是 LEA.



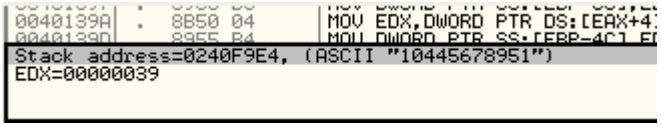

EDX 的值为 39,减去 14,然后将结果保存到 EAX 中。



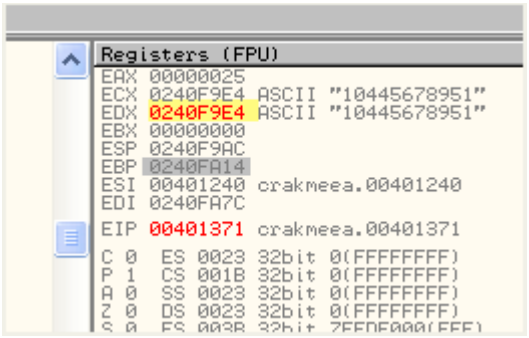

EAX 中保存的结果为 25。




下一条指令是将 EBP-30 的值(我的机器为 240f9e4)保存到 EDX 中。



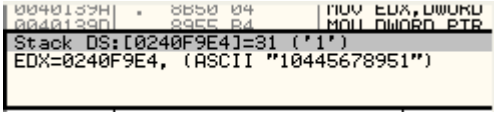

按 F7 键。



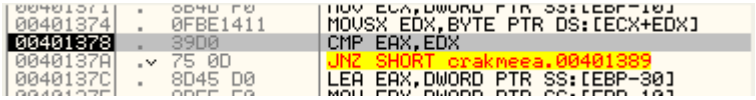

EDX 就保存了常量数字字符串的首地址。


我们可以看到 ECX 被作为一个循环变量初始化为零,

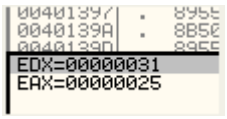

这里我们看到ECX已经被赋值为了零,然后EDX + ECX就指向了常量数字字符串的第一个字节,我们来看看OD解释窗口的信息。




31 对应的 ASCII 码为'1',为常量数字字符串的第一个字符。




这里就进行比较了



EAX 保存了我们输入的错误字符串的第一个字符 39 减去 14 的结果,这里为 25,EDX 保存了常量数字字符串的第一个字节 31。

因此,我们可以看到的

CMP EAX, EDX

实际上是

CMP 错误的序列号的第一个字节-14 ,常量数字字符串的第一个字节

CMP 25,31

由于这两个操作数的差值不为零,所以零标志位 Z 不会置 1,JNZ 跳转就会实现。

我们输入的是一个错误的序列号,如果输入的是一个正确的序列号的话,那么比较的结果就是正确的。

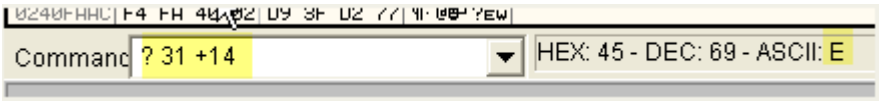
CMP (正确序列号的第一个字节值-14),31

判断两者相等的条件为:

正确序列号的第一个字节值-14 = 常量数字字符串的第一个字节

所以

正确的序列号的第一个字节值 = 常量数字字符串第一个字节值 + 14




正确的序列号的第一个字节值 = 31 + 14

正确的序列号的第一个字节值 = 45,该 ASCII 码对应的字符为"E"。

所以序列号的第一个字母为 E。

以上字符校验过程重复进行,直到所有字符都校验完毕。

第一个字节值 = 常量数字字符串第一个字节值+ 14

第二个字节值 = 常量数字字符串第二个字节值+14

第三个字节值 = 常量数字字符串第三个字节值+14

依次类推。

我们用这个公式来计算序列号(正确序列号字节值 = 常量数字字符串对应字节值 + 14)的每个字符。

```
31 30 34 34 35 36 37 38  10445678
39 35 31 00 00 00 00 00  951.....
```

31 + 14 = 45 对应字符 **E**

30 + 14 = 44 对应字符 **D**

34 + 14 = 48 对应字符 **H**

34 + 14 = 48 对应字符 **H**

35 + 14 = 49 对应字符 **I**

36 + 14 = 4A 对应字符 **J**

37 + 14 = 4B 对应字符 **K**

38 + 14 = 4C 对应字符 **L**

39 + 14 = 4D 对应字符 **M**

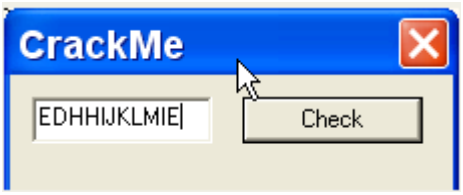
35 + 14 = 49 对应字符 **I**

31 + 14 = 45 对应字符 **E**

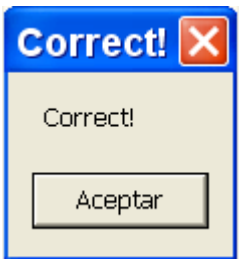
因此,正确的序列号为:

EDHHIJKLMIE

我们删除之前设置过的断点, 然后在主窗口中输入正确的序列号。




单击 check 按钮。




这个 CrackMe 就完成了。

这个 CrackMe 相比之前的要复杂一点,但是我觉得稍加练习也是可以很容易的解决它的。

好了,这里有一个名为 Splish 的 CrackMe,大家尝试一下找到它的序列号。