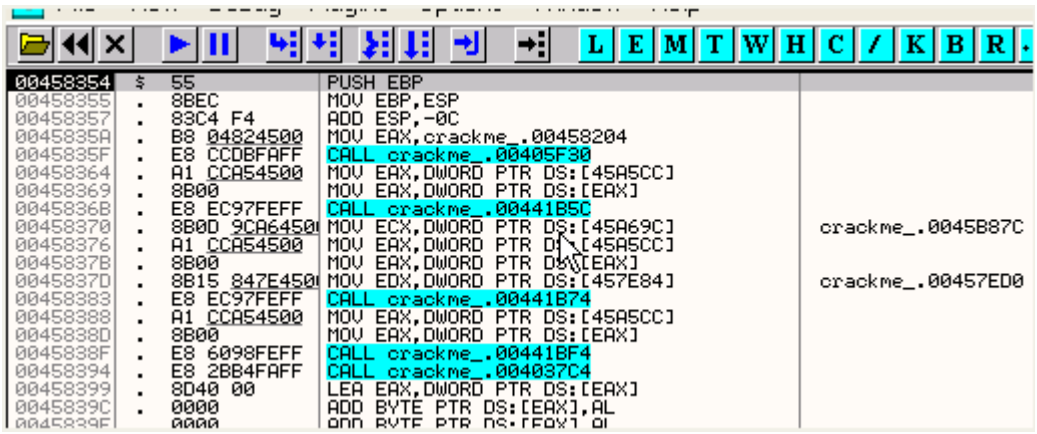
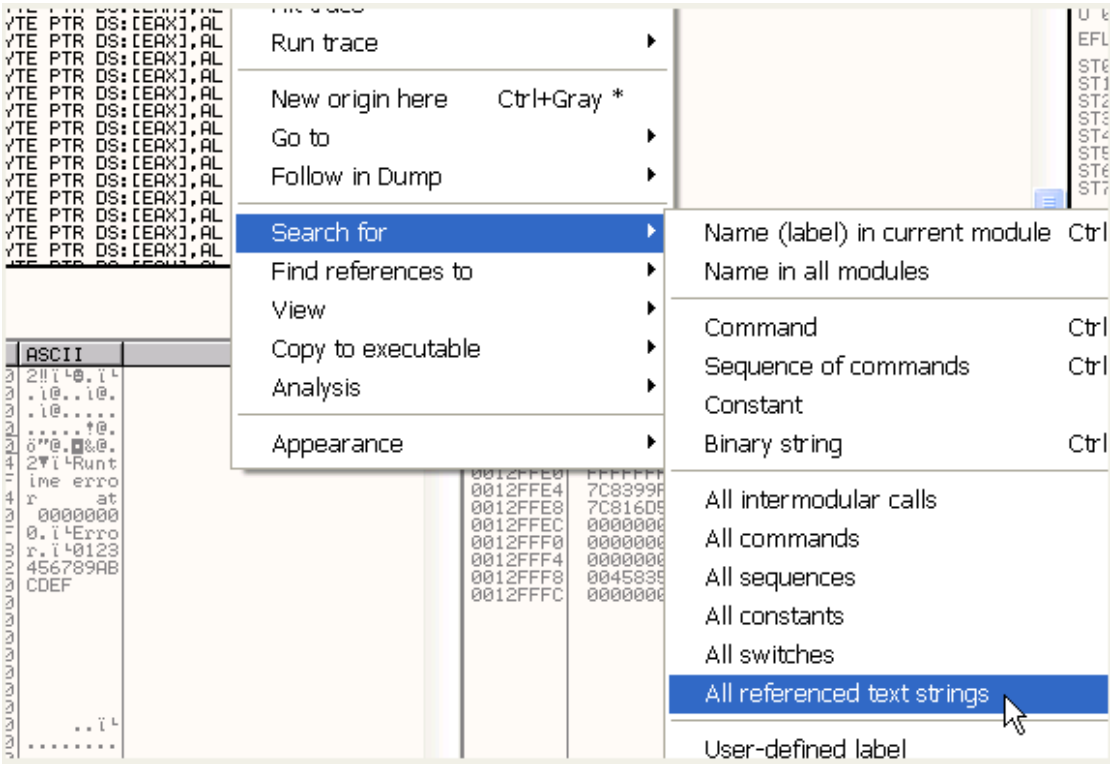
第十八章-序列号生成算法分析-Part3

本章,我们将讨论 Stzwei'em 提供的 CrackMe,名字叫"crackme_4stz"。

用 OD 加载它。



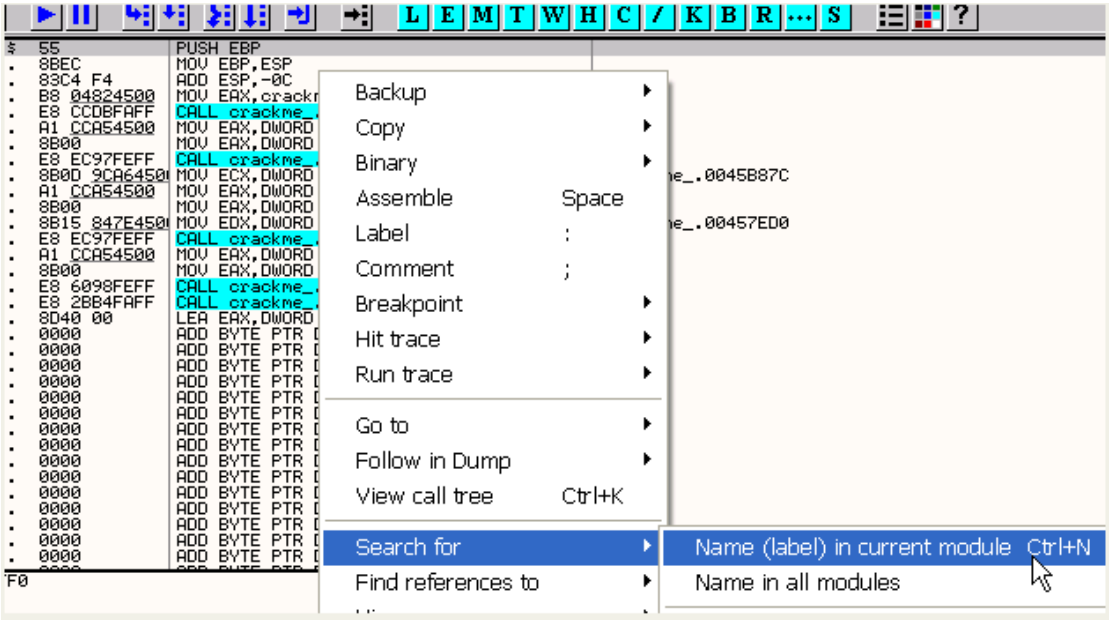
我们断在了入口点处,我们首先看一下 API 函数列表。



程序中使用的字符串列表如下:

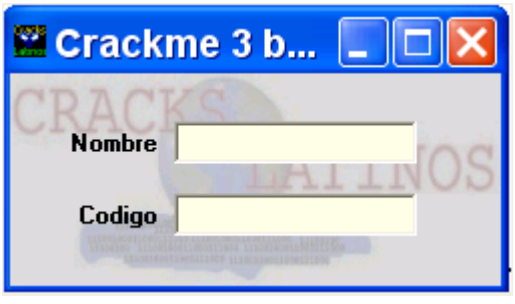
```
Address  Disassembly                                Text string
00401006 ASCII "Boolean"
0040101B ASCII "False"
00401021 ASCII "True"
0040102E ASCII "Char"
00401042 ASCII "Integer"
0040105A ASCII "Byte"
0040106E ASCII "Word"
00401082 ASCII "String"
00401088 DD crackme_.004010D4                     ASCII 07,"TObject"
004010A8 DD crackme_.004010D4                     ASCII 07,"TObject"
004010D5 ASCII "TObject"
004010E2 ASCII "TObject"
004010E9 DD crackme_.004010D4                     ASCII 07,"TObject"
004010F4 ASCII "System"
00401102 ASCII "IUnknown"
00401120 ASCII "System"
00401178 DD crackme_.004011C4                     ASCII 11,"TInterfacedObject"
00401198 DD crackme_.004011C4                     ASCII 11,"TInterfacedObject"
004011C5 ASCII "TInterfacedObjec"
004011D5 ASCII "t"
00402B04 PUSH crackme_.00402B84                   ASCII "SOFTWARE\Borland\Delphi\RTL"
00402B38 PUSH crackme_.00402BA0                   ASCII "FPUMaskValue"
00402B84 ASCII "SOFTWARE\Borland"
00402B94 ASCII "\Delphi\RTL",0
00402BA0 ASCII "FPUMaskValue",0
00403747 MOV EBX,crackme_.0045903C                ASCII "  at 00000000"
0040380A MOV EDX,crackme_.0045902C                ASCII "Runtime error     at 00000000"
0040382B PUSH crackme_.0045904C                   ASCII "Error"
00403830 PUSH crackme_.0045902C                   ASCII "Runtime error     at 00000000"
004038B7 ASCII "Portions Copyrig"
004038C7 ASCII "ht (c) 1983,99 B"
004038D7 ASCII "orland",0
00404C99 PUSH crackme_.00404E08                   ASCII "kernel32.dll"
00404CA9 PUSH crackme_.00404E18                   ASCII "GetLongPathNameA"
00404E08 ASCII "kernel32.dll",0
00404E18 ASCII "GetLongPathNameA"
00404E28 ASCII 0
00404E5C PUSH crackme_.00404FE8                   ASCII "Software\Borland\Locales"
00404E7A PUSH crackme_.00405004                   ASCII "Software\Borland\Delphi\Locales"
00404FE8 ASCII "Software\Borland"
00404FF8 ASCII "\Locales",0
00405004 ASCII "Software\Borland"
00405014 ASCII "\Delphi\Locales",0
004051A4 CMP DWORD PTR DS:[EBX+4],10000           UNICODE "ALLUSERSPROFILE=C:\Documents and Settings\All Users"
00405661 MOV EDX,crackme_.00459064                ASCII 20,"                    "
0040587C ASCII " @",0
00405886 ASCII "#@",0
```

貌似没有什么我们感兴趣的字符串,我们再来看看 API 函数列表。

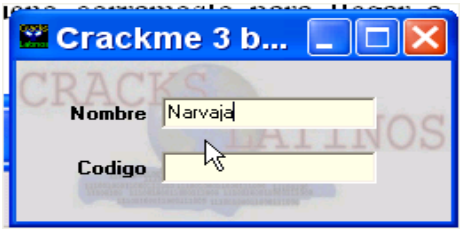


Address	Section	Type	Name	Comment
3045C600	.idata	Import	user32.DestroyCursor	
3045C5FC	.idata	Import	user32.DestroyIcon	
3045C5F8	.idata	Import	user32.DestroyMenu	
3045C5F4	.idata	Import	user32.DestroyWindow	
3045C5F0	.idata	Import	user32.DispatchMessageA	
3045C5EC	.idata	Import	user32.DrawEdge	
3045C5E8	.idata	Import	user32.DrawFrameControl	
3045C5E4	.idata	Import	user32.DrawIcon	
3045C5E0	.idata	Import	user32.DrawIconEx	
3045C5DC	.idata	Import	user32.DrawMenuBar	
3045C5D8	.idata	Import	user32.DrawTextA	
3045C5D4	.idata	Import	user32.EnableMenuItem	
3045C5D0	.idata	Import	user32.EnableScrollBar	
3045C5CC	.idata	Import	user32.EnableWindow	
3045C5C8	.idata	Import	user32.EndPaint	
3045C0FC	.idata	Import	kernel32.EnterCriticalSection	
3045C2C0	.idata	Import	kernel32.EnterCriticalSection	
3045C2BC	.idata	Import	kernel32.EnumCalendarInfoA	
3045C5C4	.idata	Import	user32.EnumThreadWindows	
3045C5C0	.idata	Import	user32.EnumWindows	
3045C5BC	.idata	Import	user32.EqualRect	
3045C3A4	.idata	Import	gdi32.ExcludeClipRect	
3045C164	.idata	Import	kernel32.ExitProcess	
3045C5B8	.idata	Import	user32.FillRect	
3045C160	.idata	Import	kernel32.FindClose	
3045C15C	.idata	Import	kernel32.FindFirstFileA	
3045C2B8	.idata	Import	kernel32.FindResourceA	
3045C5B4	.idata	Import	user32.FindWindowA	
3045C2B4	.idata	Import	kernel32.FormatMessageA	
3045C5B0	.idata	Import	user32.FrameRect	
3045C158	.idata	Import	kernel32.FreeLibrary	
3045C2B0	.idata	Import	kernel32.FreeLibrary	
3045C2AC	.idata	Import	kernel32.FreeResource	
3045C3A0	.idata	Import	gdi32.GdiFlush	
3045C5AC	.idata	Import	user32.GetActiveWindow	
3045C39C	.idata	Import	gdi32.GetBitmapBits	
3045C398	.idata	Import	gdi32.GetBrushOrgEx	
3045C5A8	.idata	Import	user32.GetCapture	
3045C5A4	.idata	Import	user32.GetClassInfoA	
3045C5A0	.idata	Import	user32.GetClientRect	
3045C59C	.idata	Import	user32.GetClipboardData	
3045C394	.idata	Import	gdi32.GetClipBox	
3045C154	.idata	Import	kernel32.GetCommandLineA	
3045C2A8	.idata	Import	kernel32.GetCPInfo	
3045C390	.idata	Import	gdi32.GetCurrentPositionEx	
3045C2A4	.idata	Import	kernel32.GetCurrentProcessId	
3045C0F0	.idata	Import	kernel32.GetCurrentThreadId	
3045C2A0	.idata	Import	kernel32.GetCurrentThreadId	
3045C598	.idata	Import	user32.GetCursor	
3045C594	.idata	Import	user32.GetCursorPos	
3045C590	.idata	Import	user32.GetDC	
3045C58C	.idata	Import	user32.GetDCEx	
3045C38C	.idata	Import	gdi32.GetDCOrgEx	
3045C588	.idata	Import	user32.GetDesktopWindow	
3045C380	.idata	Import	gdi32.GetDeviceCaps	
3045C388	.idata	Import	gdi32.GetDIBColorTable	
3045C384	.idata	Import	gdi32.GetDIBits	
3045C29C	.idata	Import	kernel32.GetDiskFreeSpaceA	
3045C37C	.idata	Import	gdi32.GetEnhMetaFileBits	
3045C378	.idata	Import	gdi32.GetEnhMetaFileHeader	
3045C374	.idata	Import	gdi32.GetEnhMetaFilePaletteEntries	
3045C188	.idata	Import	kernel32.GetFileSize	

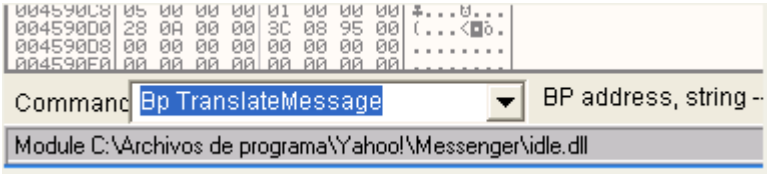
哇,这么多...好吧,我们运行起来,出现了主窗口,可以输入序列号。



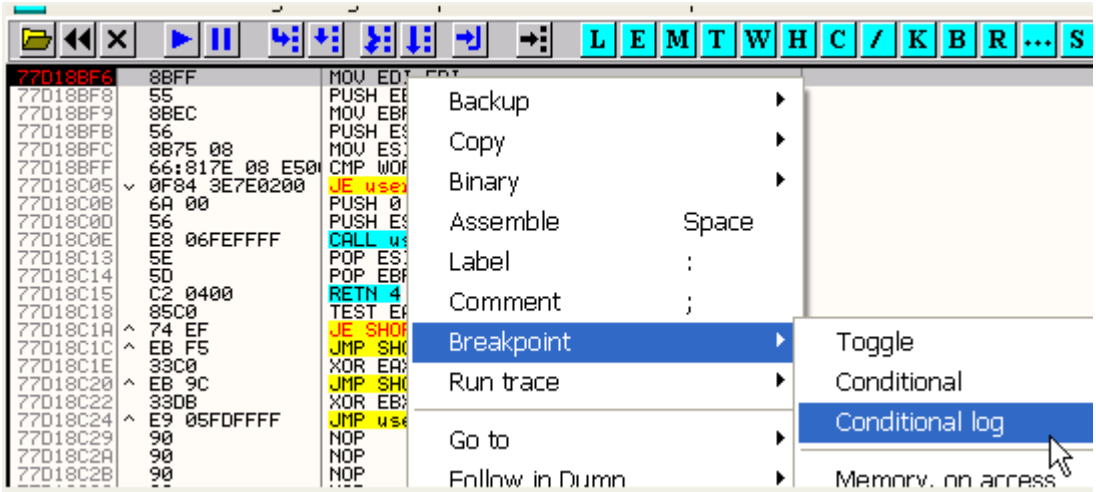
我们可以看到没有注册按钮,我们随便输入一个用户名。



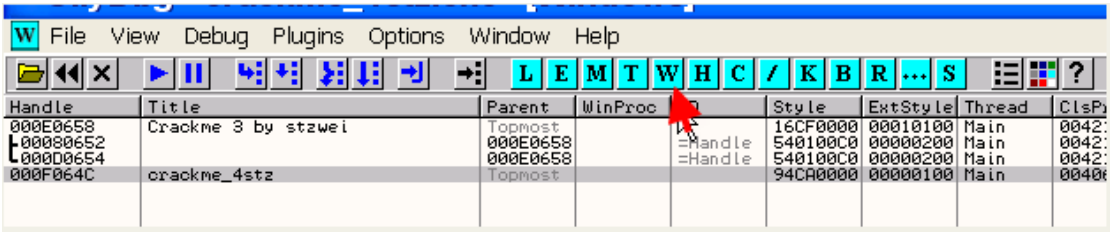
我们需要给 TranslateMessage 这个 API 函数设置条件记录断点,首先给该函数设置一个普通断点。



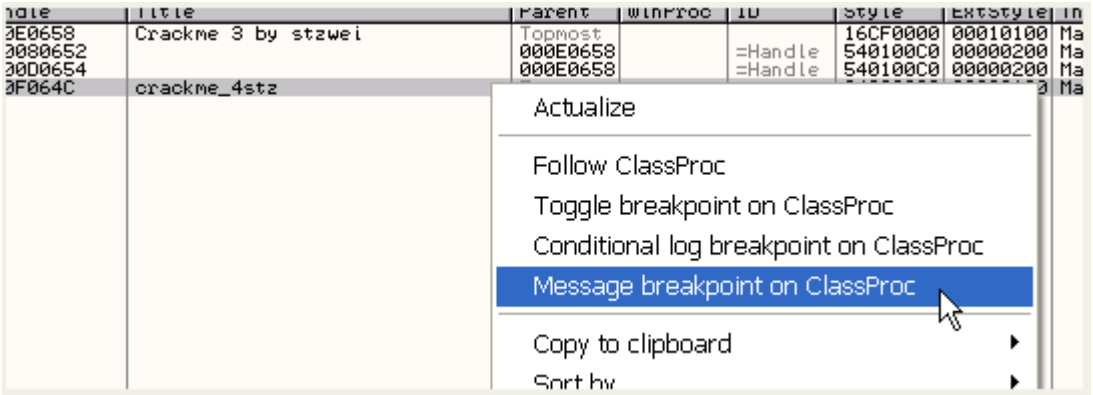
我们回到 CrackMe 的主窗口,继续编辑断点的条件。



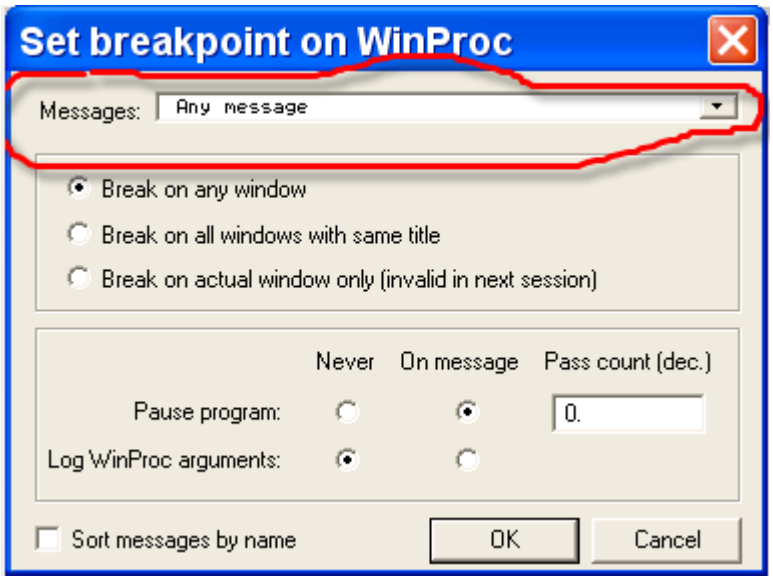
如果你不知道要拦截什么的消息的话,那就先单击工具栏中 W 按钮打开窗口列表吧。



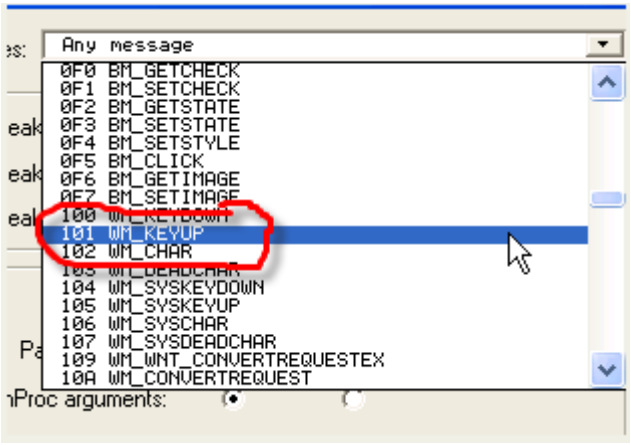
我们找到 CrackMe 的主窗口这行,在上面单击鼠标右键。



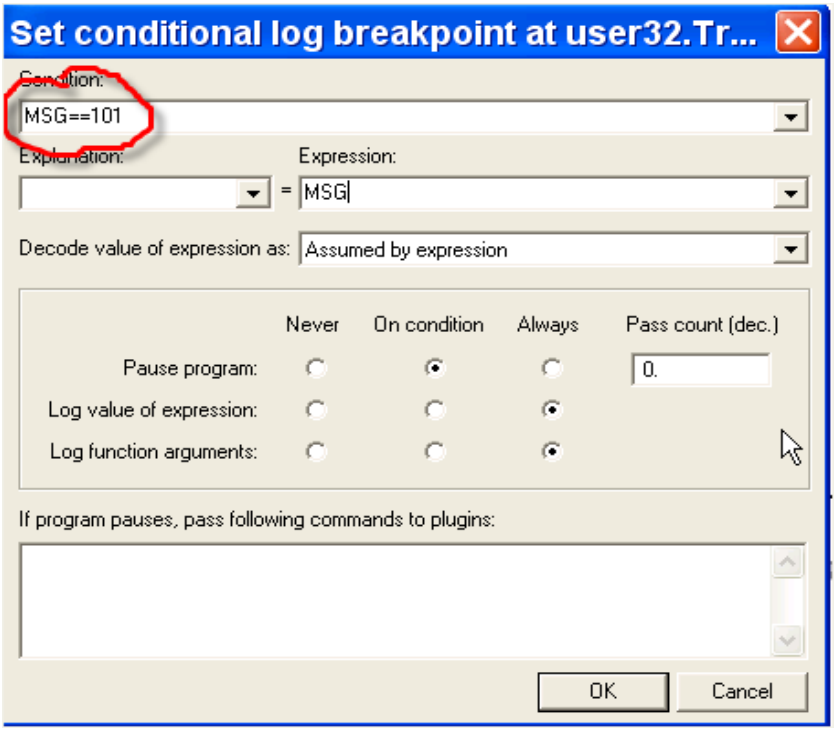
出现了消息断点窗口,我们打开 Messages 消息下拉框。



我们找到 WM_KEYUP 消息。



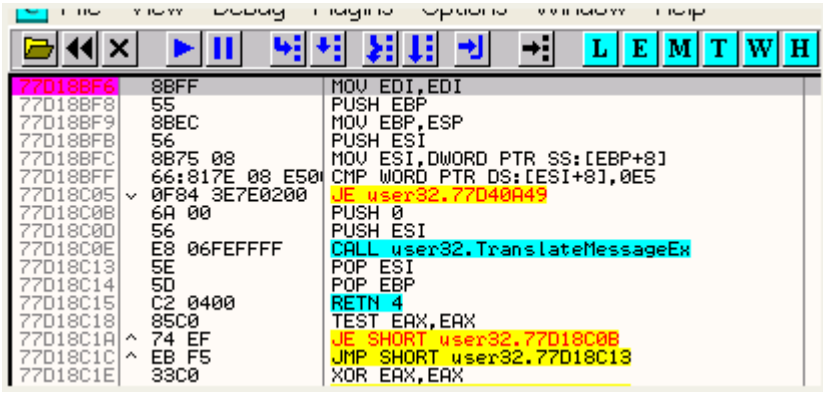
我们可以看到 WM_KEYUP 的值为 101,所以我们关闭该窗口,打开 TranslateMessage 断点所对应的条件记录断点窗口。



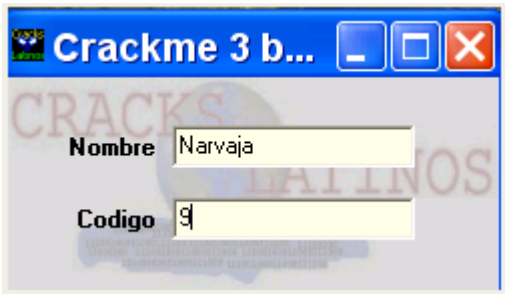
这里,我们设置条件为 MSG == 101(注意是双等号),如果我们没有找到这个值的话,我们写成 MSG== WM_KEYUP 同样可以。

但是我更喜欢写成 MSG==101 这样形式,按照的自己的习惯来就行了。

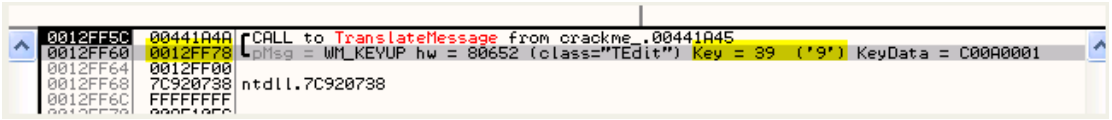
单击 OK 以后 TranslateMessage 函数入口处的断点就会变成粉红色。



我们运行起来,输入错误序列号的第一个字母。

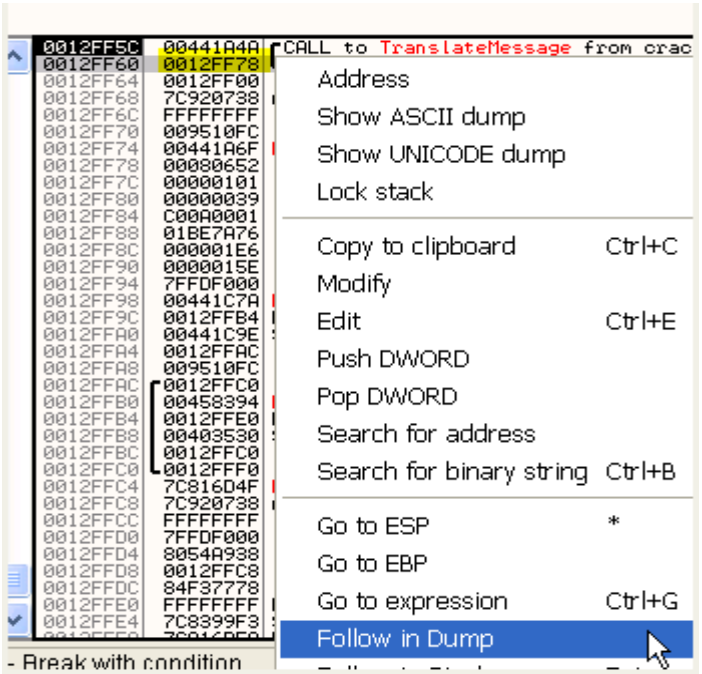


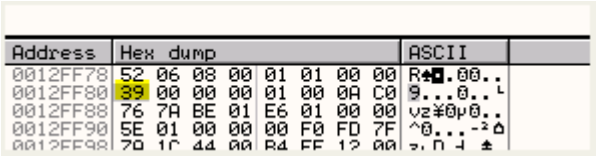
马上就触发刚刚设置的条件断点了。



我们可以看到堆栈中 TranslateMessage 的参数情况,12FF78 指向的结构中保存 key 值等于 39(十六进制),该 ASCII 值正好对应的得我们刚刚输入的字符'9'。

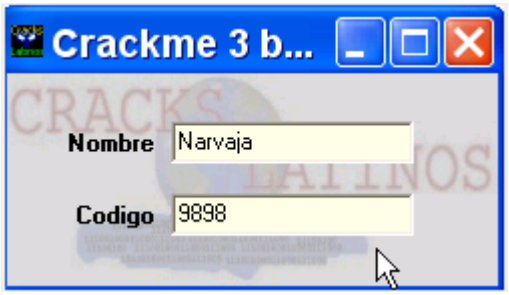
我们在该参数上面单击鼠标右键选择-Follow in Dump 在数据窗口中定位到内存单元。





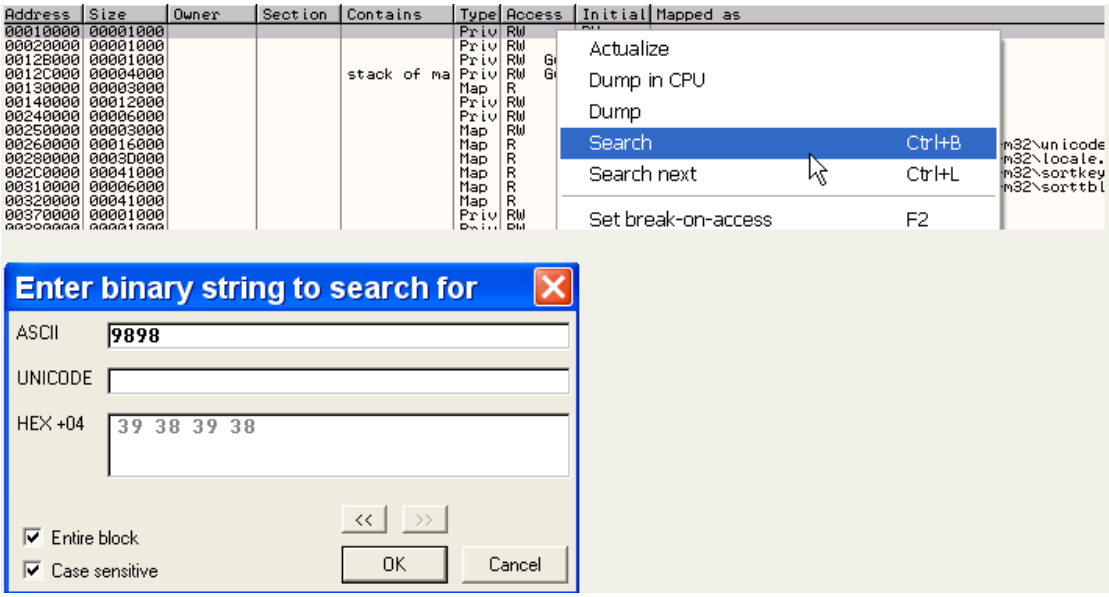
我们可能会想到对该字节设置内存访问断点,让程序在读取该字节与正确的序列号进行比较的时候断下来,但是该 CrackMe 是个 Delphi 的程序,Delphi 程序是经过深度封装的程序,可能在与正确的序列号进行比较之前,12FF80 对应内存单元中序列号已经经过了很多次的内存的中拷贝了,所以直接对该字节设置内存访问断点是不可取的。我们可以采用上一章使用的方法,先在内存中搜索我们输入的序列号,定位到以后再对区域设置内存访问断点可能更方便一些。

运行起来。

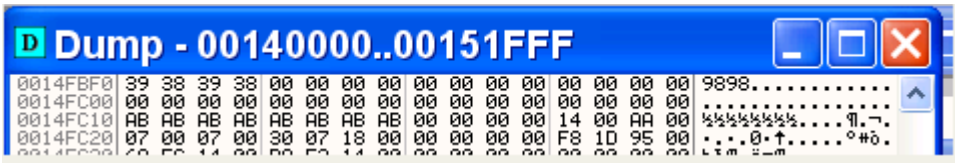


我们输入 9898,然后回到 OD 中,通过单击工具栏中 M 按钮查看所有内存。

单击鼠标右键选择-Search。

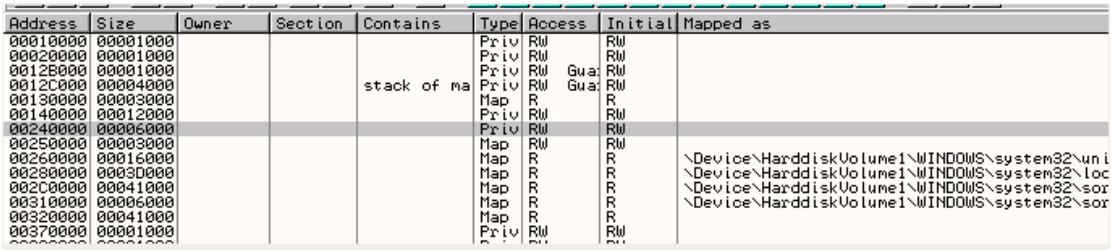


查找 9898。

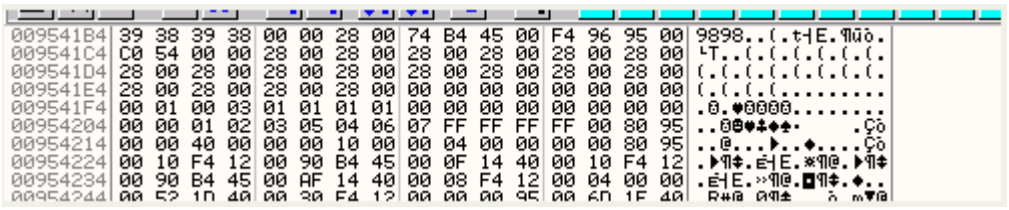


我们继续 CTRL+L 查看看看还有没有其他内存中有该字符串。

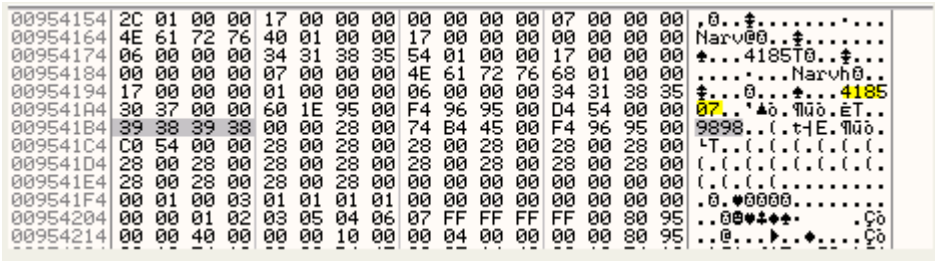
直到 OD 底部显示黄色的”Item not found”的字样,这个区段就查找完毕了,我们继续回到内存列表窗口中。



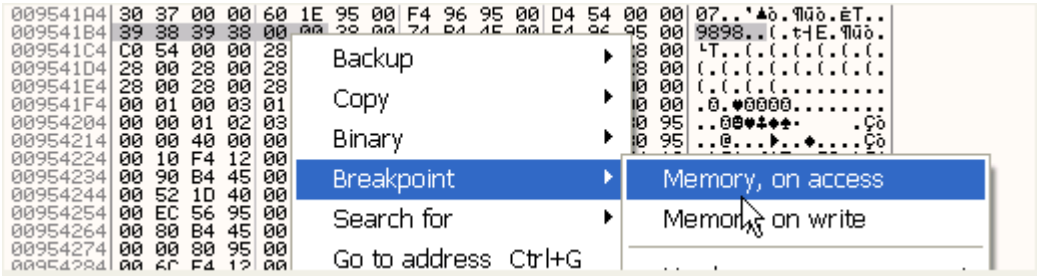
我们再次 CTRL+L。



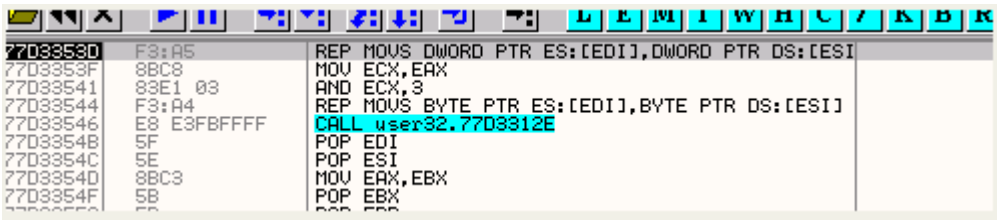
找到了该区段的第一处,我们继续 CTRL+L 看该区段还有没有其他地方有。



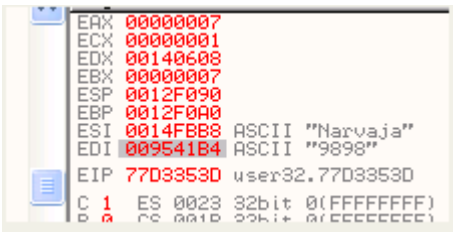
继续查找没有了,我们可以看到上方有一个 418507 可能是正确的序列号,我们对我们输入的错误序列号设置内存访问断点,看看程序是否会取该序列号来进行比较。



运行起来。

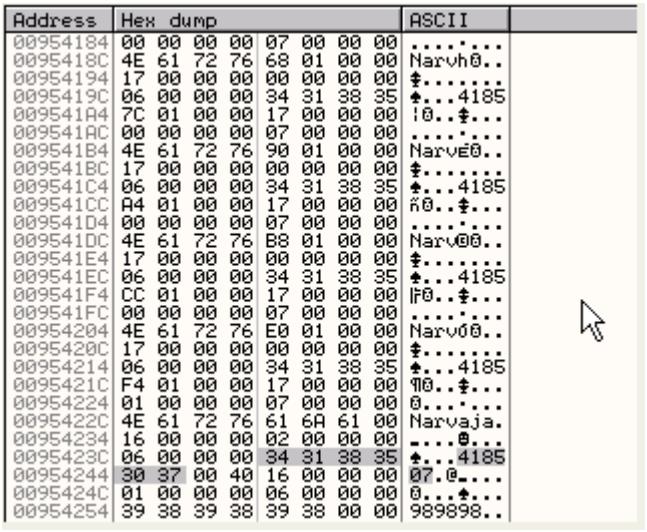


这里,我们可以看到正在拷贝我们输入的用户名。



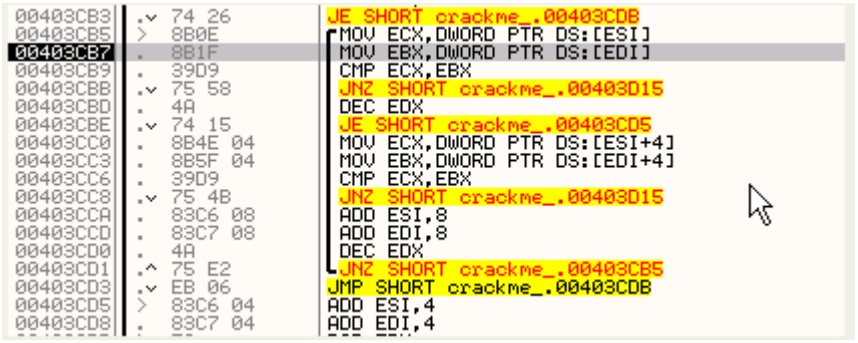
我们可以看到错误序列号的内存单元将被覆盖为我们输入的用户名。那我们可以对下面可能是正确序列号的字符串设置内存访

问断点。



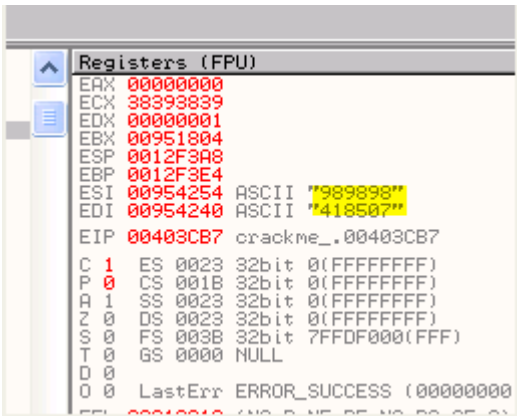
Address	Hex dump	ASCII
00954184	00 00 00 00 07 00 00 00	....•...
0095418C	4E 61 72 76 68 01 00 00	Narvh0..
00954194	17 00 00 00 00 00 00 00	↨.....
0095419C	06 00 00 00 34 31 38 35	♠...4185
009541A4	7C 01 00 00 17 00 00 00	!0..↨...
009541AC	00 00 00 00 07 00 00 00	....•...
009541B4	4E 61 72 76 90 01 00 00	NarvÉ0..
009541BC	17 00 00 00 00 00 00 00	↨.....
009541C4	06 00 00 00 34 31 38 35	♠...4185
009541CC	A4 01 00 00 17 00 00 00	ñ0..↨...
009541D4	00 00 00 00 07 00 00 00	....•...
009541DC	4E 61 72 76 B8 01 00 00	Narv@0..
009541E4	17 00 00 00 00 00 00 00	↨.....
009541EC	06 00 00 00 34 31 38 35	♠...4185
009541F4	CC 01 00 00 17 00 00 00	╠0..↨...
009541FC	00 00 00 00 07 00 00 00	....•...
00954204	4E 61 72 76 E0 01 00 00	Narv∝0..
0095420C	17 00 00 00 00 00 00 00	↨.....
00954214	06 00 00 00 34 31 38 35	♠...4185
0095421C	F4 01 00 00 17 00 00 00	¶0..↨...
00954224	01 00 00 00 07 00 00 00	0...•...
0095422C	4E 61 72 76 61 6A 61 00	Narvaja.
00954234	16 00 00 00 02 00 00 00	▬...☻...
0095423C	06 00 00 00 34 31 38 35	♠...4185
00954244	30 37 00 40 16 00 00 00	07.@▬...
0095424C	01 00 00 00 06 00 00 00	0...♠...
00954254	39 38 39 38 39 38 00 00	989898..

可以看到断下来了,正在进行比较。



```
00403CB3  .v 74 26      JE SHORT crackme_.00403CDB
00403CB5  > 8B0E        MOV ECX,DWORD PTR DS:[ESI]
00403CB7  . 8B1F        MOV EBX,DWORD PTR DS:[EDI]
00403CB9  . 39D9        CMP ECX,EBX
00403CBB  .v 75 58      JNZ SHORT crackme_.00403D15
00403CBD  . 4A          DEC EDX
00403CBE  .v 74 15      JE SHORT crackme_.00403CD5
00403CC0  . 8B4E 04     MOV ECX,DWORD PTR DS:[ESI+4]
00403CC3  . 8B5F 04     MOV EBX,DWORD PTR DS:[EDI+4]
00403CC6  . 39D9        CMP ECX,EBX
00403CC8  .v 75 4B      JNZ SHORT crackme_.00403D15
00403CCA  . 83C6 08     ADD ESI,8
00403CCD  . 83C7 08     ADD EDI,8
00403CD0  . 4A          DEC EDX
00403CD1  .^ 75 E2      JNZ SHORT crackme_.00403CB5
00403CD3  .v EB 06      JMP SHORT crackme_.00403CDB
00403CD5  > 83C6 04     ADD ESI,4
00403CD8  . 83C7 04     ADD EDI,4
```

这里,我们可以看到,ESI,EDI 分别指向了正确和错误的序列号。

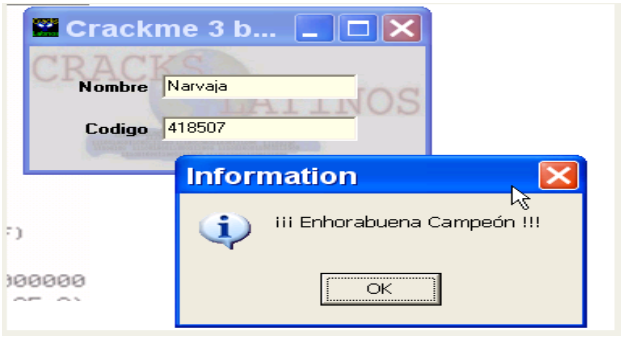


```
Registers (FPU)
EAX 00000000
ECX 38393839
EDX 00000001
EBX 00951804
ESP 0012F3A8
EBP 0012F3E4
ESI 00954254 ASCII "989898"
EDI 00954240 ASCII "418507"

EIP 00403CB7 crackme_.00403CB7

C 1  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000
```

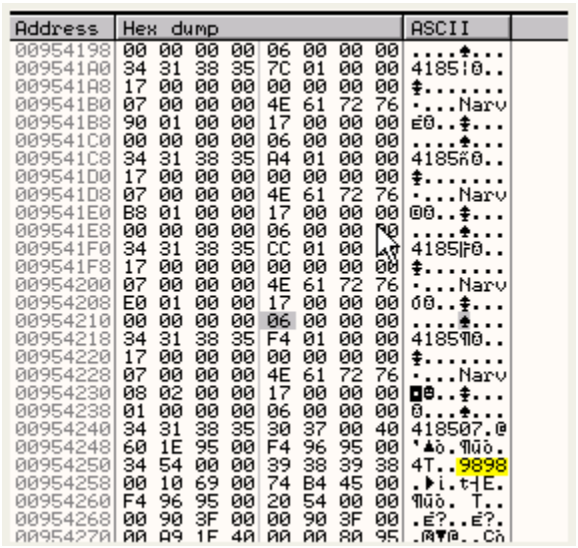
每次比较四个字节,我们验证一下该序列号是否正确。



提示序列号正确。

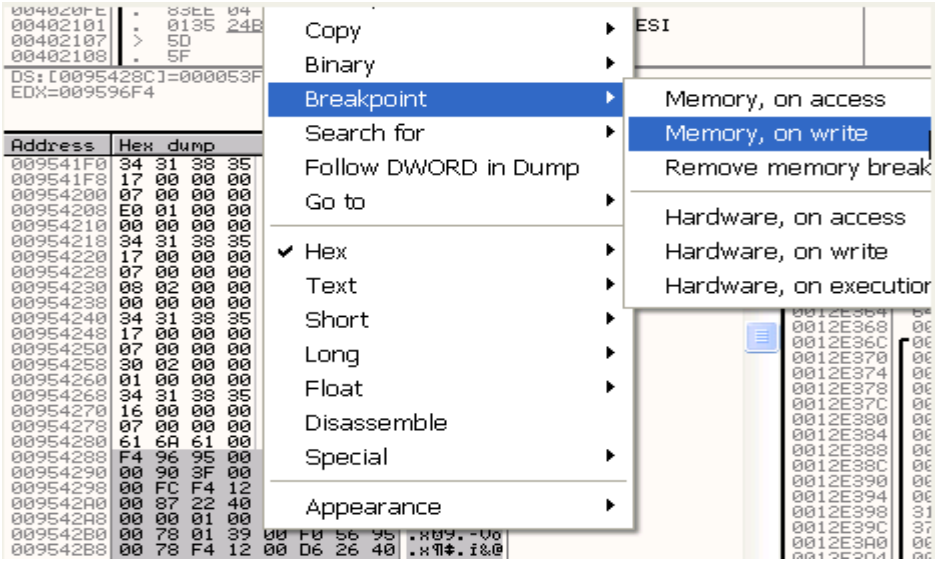
可以看到,这个的 CrackMe 还是有一点小难度的,不过我们还是可以在一个合适的时间点跟踪到我们输入的错误序列号的位置,并且通过设置内存访问断点来定位到正确的序列号。

另一种方法可以从内存写入断点切入。



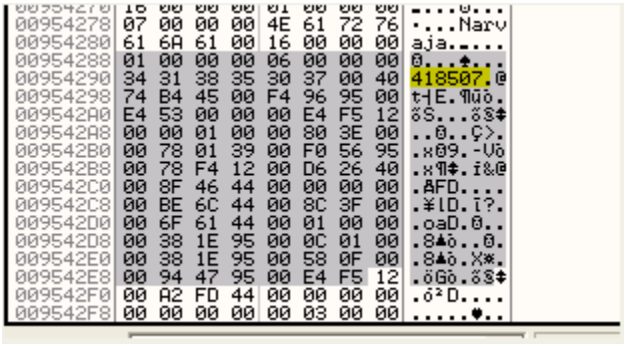
我们输入第 5 个字节。

我们对下一个字节可能会写入的内存单元设置内存写入断点。

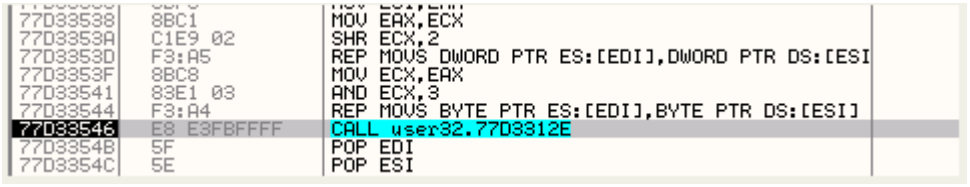


为了输入下一个字符,我们运行起来。

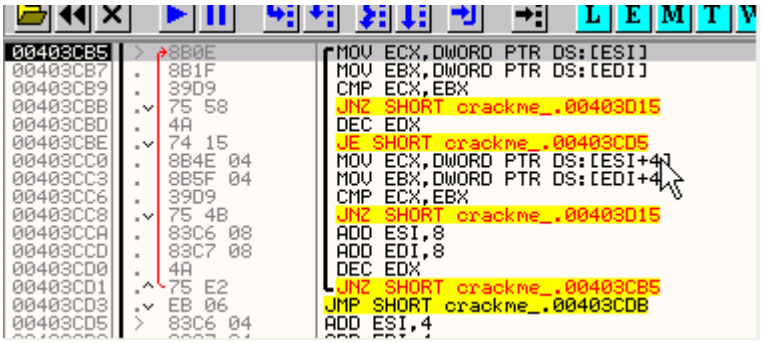
这里可能是正确的序列号。



继续运行。



断在了拷贝错误序列号处,我们可以继续设置内存断点定位到序列号比较代码处。



可以看到,这种方式稍微复杂一点。

好了,留下 CrueHead'a 的第二个 CrackMe 给大家练习。