

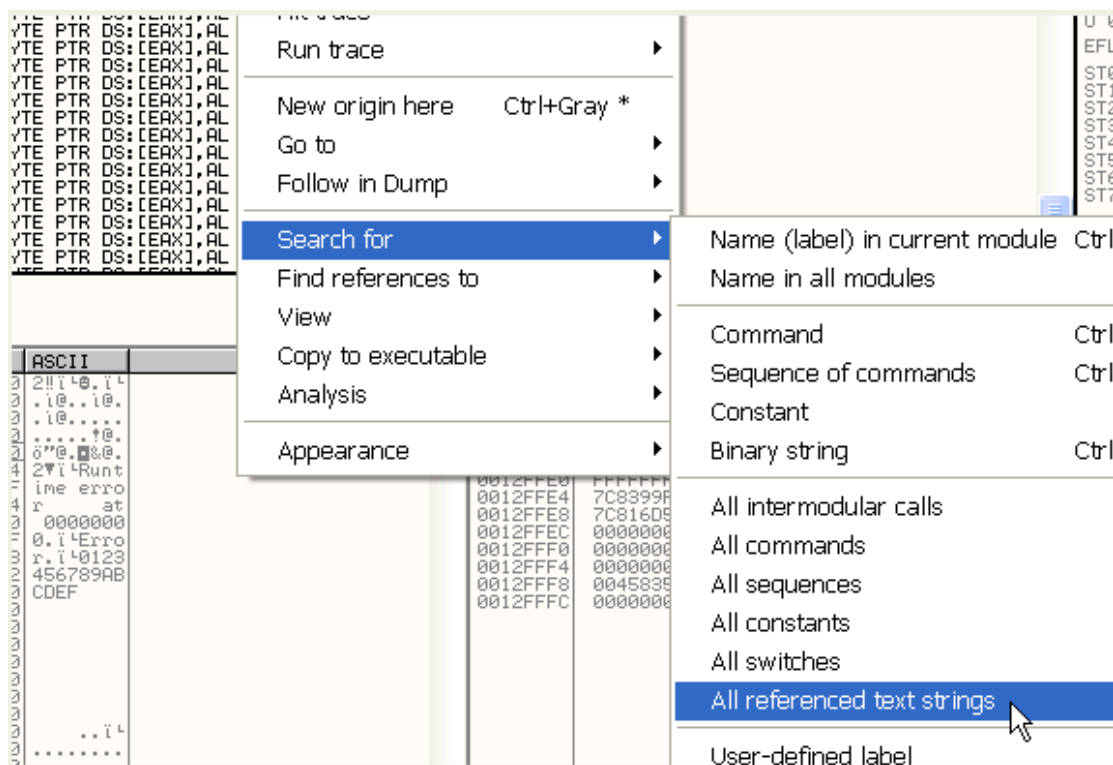
第十八章-序列号生成算法分析-Part3

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

用 OD 加载它。

```
00458354 $ 55 PUSH EBP
00458355 . 8BEC MOV EBP,ESP
00458357 . 83C4 F4 ADD ESP,-0C
0045835A . B8 04824500 MOV EAX,crackme_.00458204
0045835F . E8 CCDBFAFF CALL crackme_.00405F30
00458364 . A1 CCA54500 MOV EAX,DWORD PTR DS:[45A5CC]
00458369 . 8B00 MOV EAX,DWORD PTR DS:[EAX]
0045836B . E8 EC97FEFF CALL crackme_.00441B5C
00458370 . 8B00 9CA64500 MOV ECX,DWORD PTR DS:[45A69C]
00458376 . A1 CCA54500 MOV EAX,DWORD PTR DS:[45A5CC]
0045837B . 8B00 MOV EAX,DWORD PTR DS:[EAX]
0045837D . 8B15 847E4500 MOV EDI,DWORD PTR DS:[457E84]
00458383 . E8 EC97FEFF CALL crackme_.00441B74
00458388 . A1 CCA54500 MOV EAX,DWORD PTR DS:[45A5CC]
0045838D . 8B00 MOV EAX,DWORD PTR DS:[EAX]
0045838F . E8 6098FEFF CALL crackme_.00441BF4
00458394 . E8 2BB4FAFF CALL crackme_.004037C4
00458399 . 8D40 00 LEA EAX,DWORD PTR DS:[EAX]
0045839C . 0000 ADD BYTE PTR DS:[EAX],AL
0045839F . 0000 ORN BYTE PTR DS:[EAX],01
```

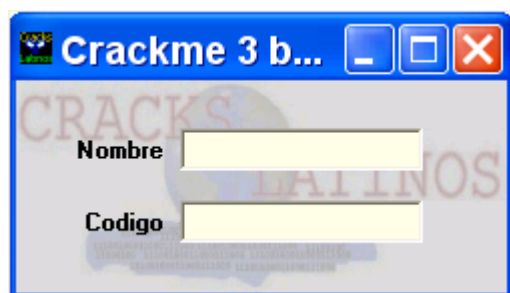
我们断在了入口点处,我们首先看一下 API 函数列表。



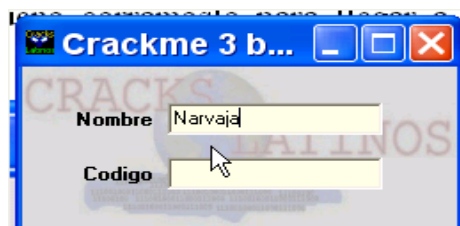
程序中使用的字符串列表如下:

Address	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	
3045C5C4	.idata	Import	kernel32.EnterCriticalSection	
3045C5C0	.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.GetClipboard	
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	

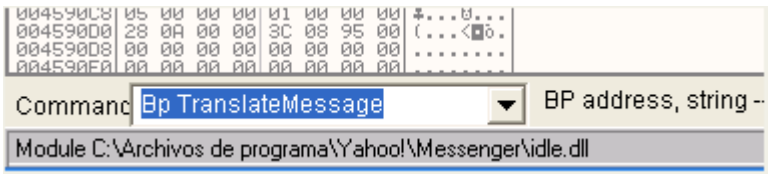
哇,这么多...好吧,我们运行起来,出现了主窗口,可以输入序列号。



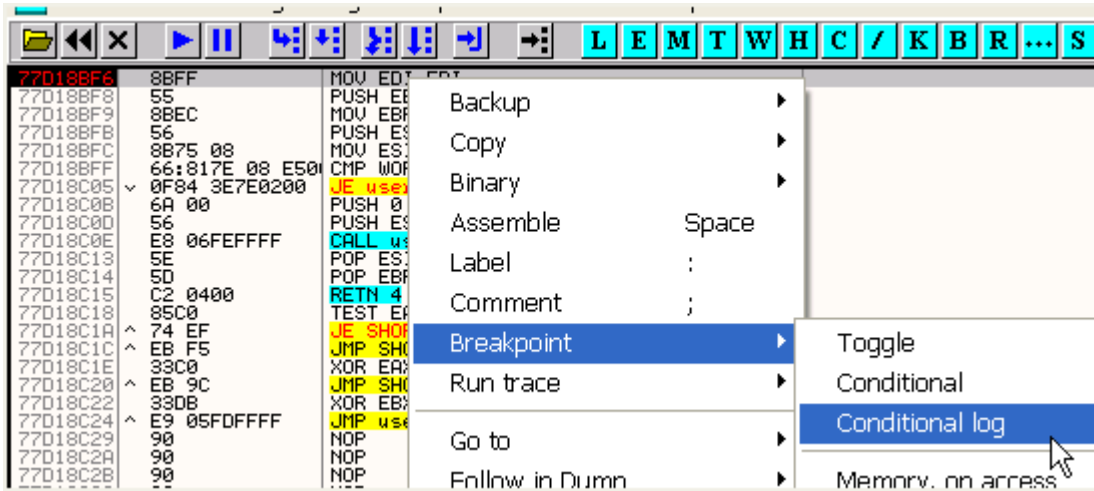
我们可以看到没有注册按钮,我们随便输入一个用户名。



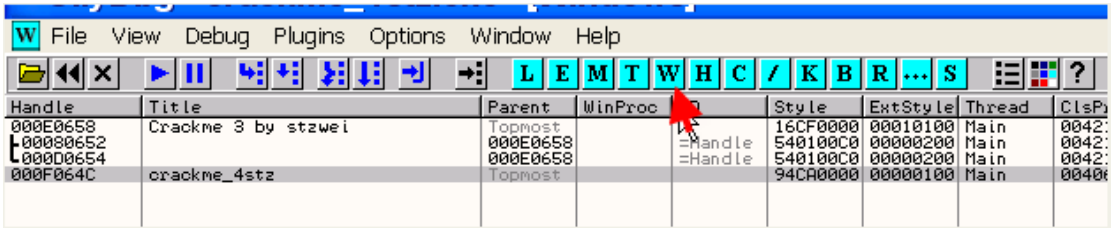
我们需要给 TranslateMessage 这个 API 函数设置条件记录断点,首先给该函数设置一个普通断点。



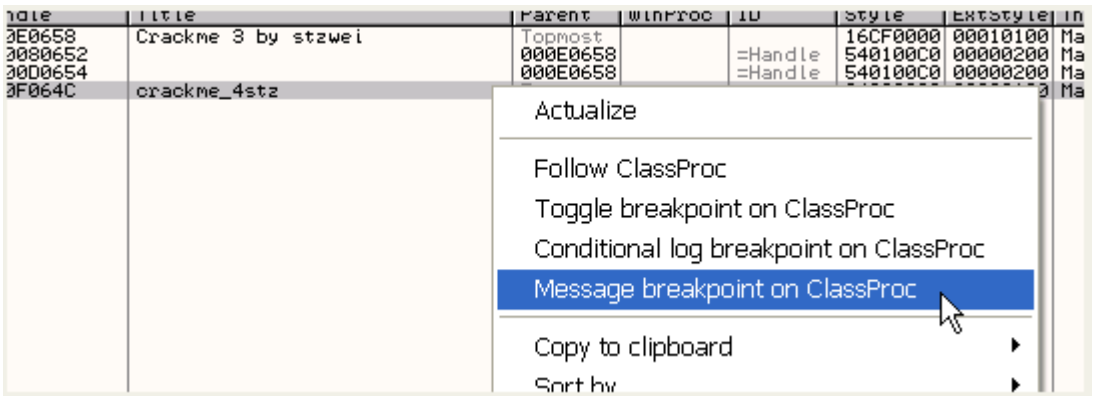
我们回到 CrackMe 的主窗口,继续编辑断点的条件。



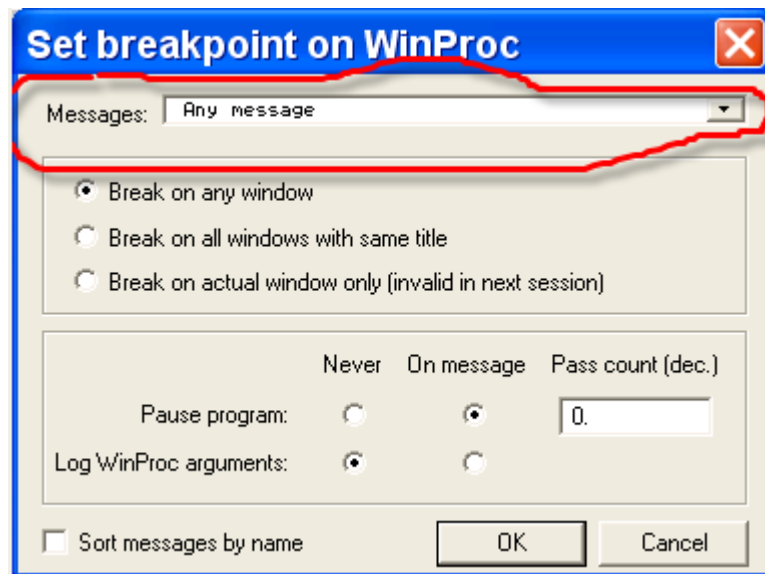
如果你不知道要拦截什么的消息的话,那就先单击工具栏中 W 按钮打开口列表吧。



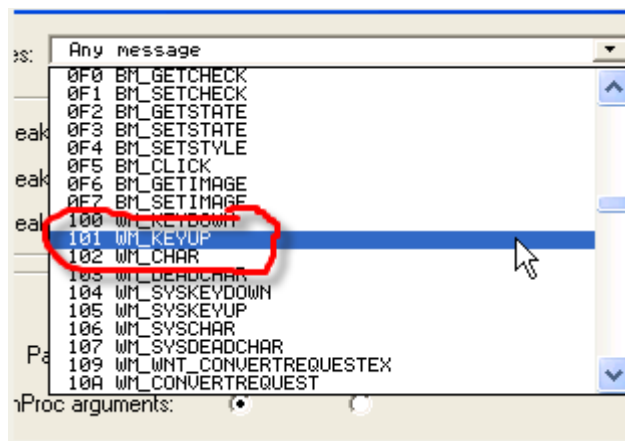
我们找到 CrackMe 的主窗口这行,在上面单击鼠标右键。



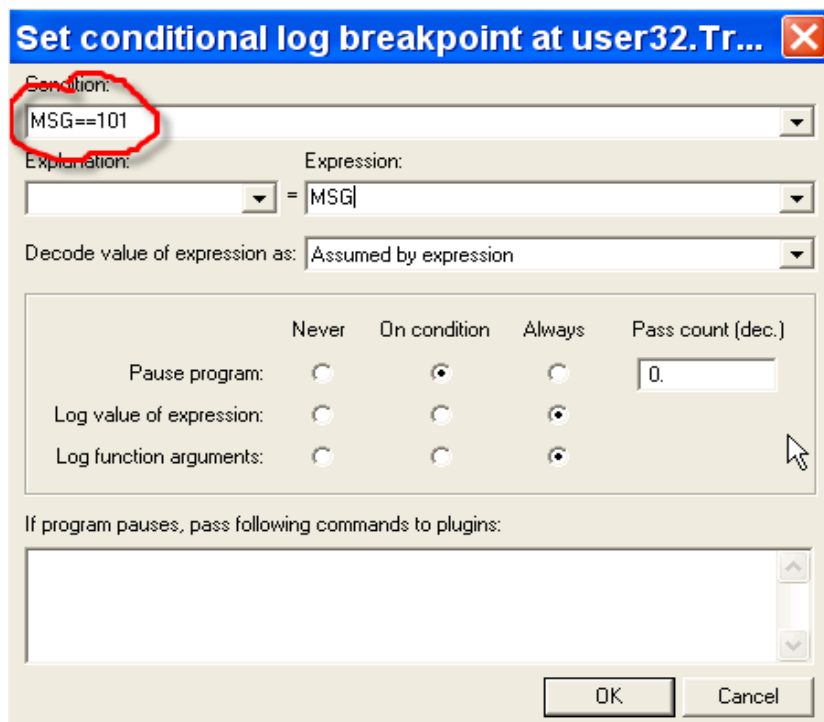
出现了消息断点窗口,我们打开 Messages 消息下拉框。



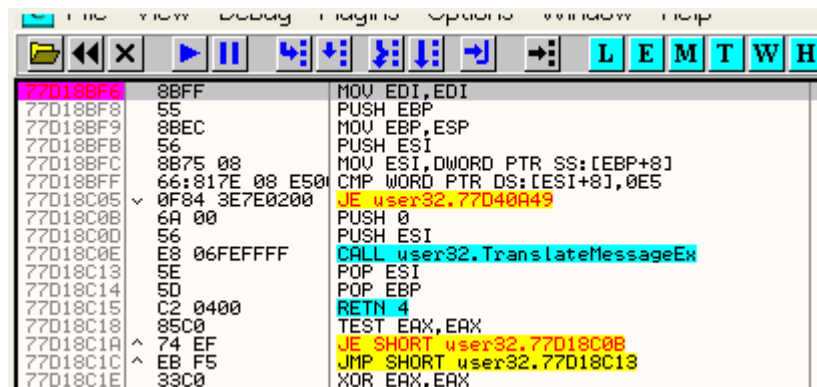
我们找到 WM_KEYUP 消息。



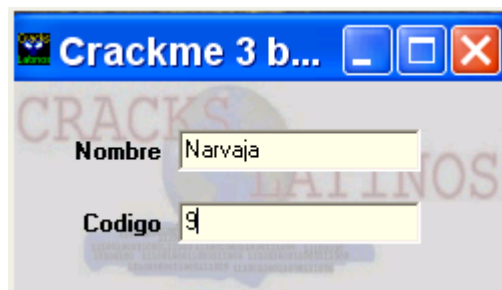
我们可以看到 WM_KEYUP 的值为 101,所以我们关闭该窗口,打开 TranslateMessage 断点所对应的条件记录断点窗口。



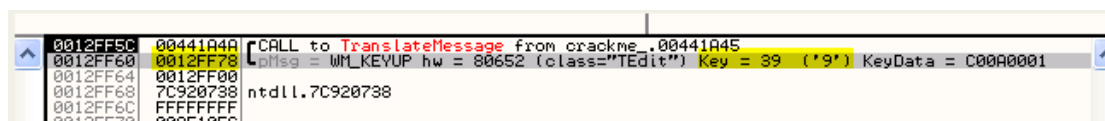
这里,我们设置条件为 MSG == 101(注意是双等号),如果我们没有找到这个值的话,我们写成 MSG == WM_KEYUP 同样可以。
但是我更喜欢写成 MSG == 101 这样形式,按照的自己的习惯来就行了。
单击 OK 以后 TranslateMessage 函数入口处的断点就会变成粉红色。



我们运行起来,输入错误序列号的第一个字母。

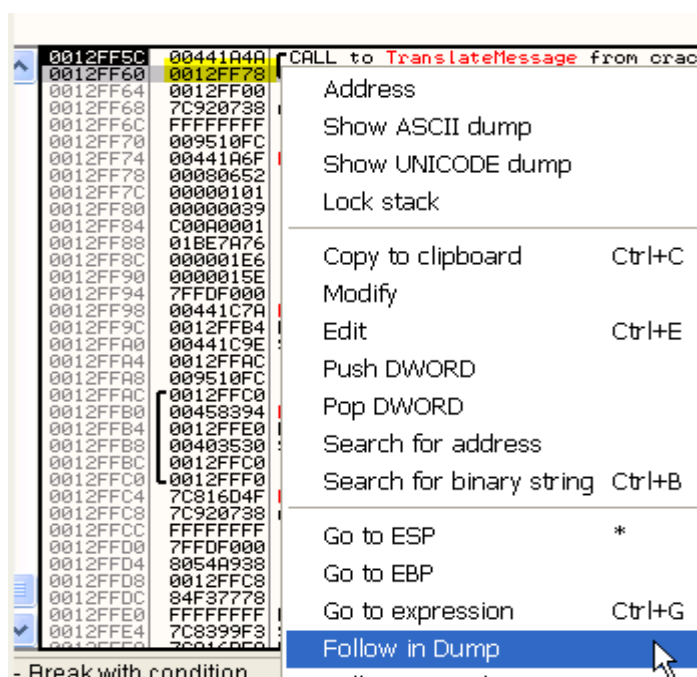


马上就触发刚刚设置的条件断点了。



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'。

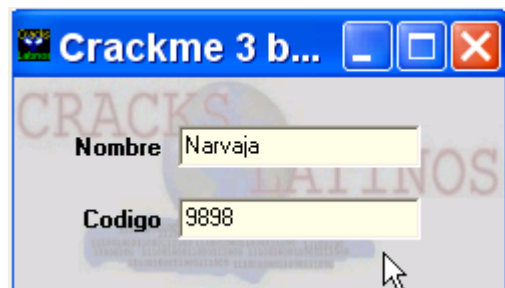
我们在该参数上面单击鼠标右键选择-Follow in Dump 在数据窗口中定位到内存单元。



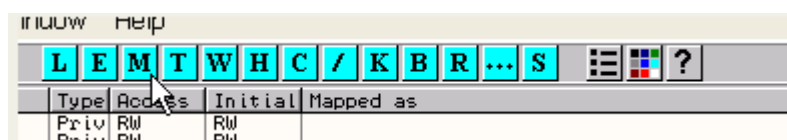
Address	Hex dump	ASCII
0012FF78	52 06 08 00 01 01 00 00	R+..00..
0012FF80	39 00 00 00 01 00 0A C0	9...0...L
0012FF88	76 7A BE 01 E6 01 00 00	vz#0p0..
0012FF90	5E 01 00 00 00 F0 FD 7F	^0...-2^
0012FF98	7D 1C 44 00 04 FF 12 00	>.n.J.±

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。

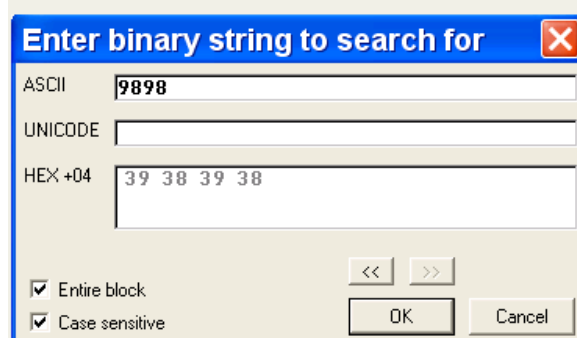
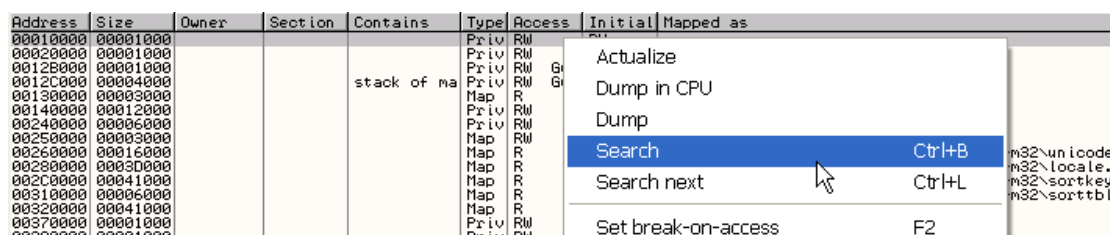
运行起来。



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。

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	Narve0..
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	!0...+
009541D4	00 00 00 00 07 00 00 00
009541DC	4E 61 72 76 B8 01 00 00	Narv00..
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	Narv00..
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	...0...
0095423C	06 00 00 00 34 31 38 35	...4185
00954244	30 37 00 40 16 00 00 00	07.0....
0095424C	01 00 00 00 06 00 00 00	0...+...
00954254	39 38 39 38 39 38 00 00	989898..

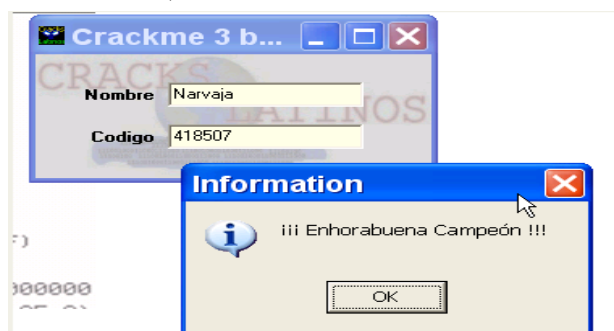
可以看到断下来了,正在进行比较。

00403CB3	> 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	> 75 58	JNZ SHORT crackme_.00403D15
00403CBD	> 4A	DEC EDX
00403CBE	> 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	> 75 48	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	> 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 7FDF000(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。

Address	Hex dump	ASCII
00954198	00 00 00 00 06 00 00 00♣...
009541A0	34 31 38 35 7C 01 00 00	4185i0..
009541A8	17 00 00 00 00 00 00 00	♣.....
009541B0	07 00 00 00 4E 61 72 76	...Narv
009541B8	90 01 00 00 17 00 00 00	e0..♣...
009541C0	00 00 00 00 06 00 00 00♣...
009541C8	34 31 38 35 A4 01 00 00	4185K0..
009541D0	17 00 00 00 00 00 00 00	♣.....
009541D8	07 00 00 00 4E 61 72 76	...Narv
009541E0	B8 01 00 00 17 00 00 00	00..♣...
009541E8	00 00 00 00 06 00 00 00♣...
009541F0	34 31 38 35 CC 01 00 00	4185f0..
009541F8	17 00 00 00 00 00 00 00	♣.....
00954200	07 00 00 00 4E 61 72 76	...Narv
00954208	E0 01 00 00 17 00 00 00	00..♣...
00954210	00 00 00 00 06 00 00 00♣...
00954218	34 31 38 35 F4 01 00 00	418500..
00954220	17 00 00 00 00 00 00 00	♣.....
00954228	07 00 00 00 4E 61 72 76	...Narv
00954230	08 02 00 00 17 00 00 00	00..♣...
00954238	01 00 00 00 06 00 00 00	0...♣...
00954240	34 31 38 35 30 37 00 40	418507..@
00954248	60 1E 95 00 F4 96 95 00	*Δo.000.
00954250	34 54 00 00 39 38 39 38	4T..9898
00954258	00 10 69 00 74 B4 45 00	.ti.t-E.
00954260	F4 96 95 00 20 54 00 00	000. T..
00954268	00 90 3F 00 00 90 3F 00	.e?..e?.
00954270	00 09 1F 40 00 00 00 95	.091F..C0

我们输入第 5 个字节。

009541F0	34 31 38 35 CC 01 00 00	4185f0..
009541F8	17 00 00 00 00 00 00 00	♣.....
00954200	07 00 00 00 4E 61 72 76	...Narv
00954208	E0 01 00 00 17 00 00 00	00..♣...
00954210	00 00 00 00 06 00 00 00♣...
00954218	34 31 38 35 F4 01 00 00	418500..
00954220	17 00 00 00 00 00 00 00	♣.....
00954228	07 00 00 00 4E 61 72 76	...Narv
00954230	08 02 00 00 17 00 00 00	00..♣...
00954238	00 00 00 00 06 00 00 00♣...
00954240	34 31 38 35 1C 02 00 00	4185L0..
00954248	17 00 00 00 00 00 00 00	♣.....
00954250	07 00 00 00 4E 61 72 76	...Narv
00954258	30 02 00 00 17 00 00 00	00..♣...
00954260	01 00 00 00 06 00 00 00	0...♣...
00954268	34 31 38 35 30 37 00 00	418507..
00954270	60 1E 95 00 F4 96 95 00	*Δo.000.
00954278	0C 54 00 00 39 38 39 38	.T..9898
00954280	39 00 20 40 74 B4 45 00	9..@t-E.
00954288	F4 96 95 00 F8 53 00 00	000.°S..
00954290	00 90 3F 00 00 5C 22 40	.e?..\"0
00954298	00 FC F4 12 00 80 22 40	.f0#.C\"0
009542A0	00 87 22 40 00 E4 F5 12	.c\"0.6S♣

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。

004020FE : 83EE 04
00402101 : 0135 24B
00402107 : 5D
00402108 : 5F
DS: [0095428C]=000053F
EDX=009596F4

Address	Hex dump
009541F0	34 31 38 35
009541F8	17 00 00 00
00954200	07 00 00 00
00954208	E0 01 00 00
00954210	00 00 00 00
00954218	34 31 38 35
00954220	17 00 00 00
00954228	07 00 00 00
00954230	08 02 00 00
00954238	00 00 00 00
00954240	34 31 38 35
00954248	17 00 00 00
00954250	07 00 00 00
00954258	30 02 00 00
00954260	01 00 00 00
00954268	34 31 38 35
00954270	16 00 00 00
00954278	07 00 00 00
00954280	61 6A 61 00
00954288	F4 96 95 00
00954290	00 90 3F 00
00954298	00 FC F4 12
009542A0	00 87 22 40
009542A8	00 00 01 00
009542B0	00 78 01 39
009542B8	00 78 F4 12

Copy
Binary
Breakpoint
Search for
Follow DWORD in Dump
Go to
Hex
Text
Short
Long
Float
Disassemble
Special
Appearance

ESI
Memory, on access
Memory, on write
Remove memory break
Hardware, on access
Hardware, on write
Hardware, on execution

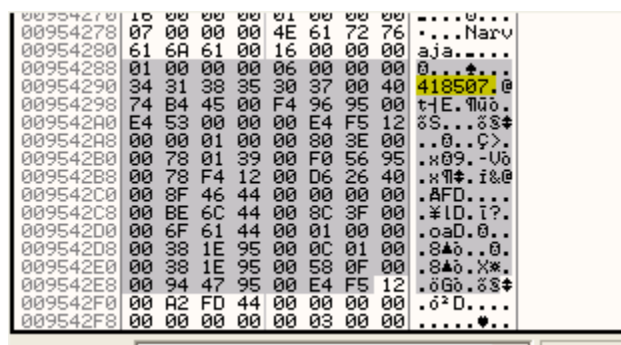
0012E364	00
0012E368	00
0012E36C	00
0012E370	00
0012E374	00
0012E378	00
0012E37C	00
0012E380	00
0012E384	00
0012E388	00
0012E38C	00
0012E390	00
0012E394	00
0012E398	31
0012E39C	37
0012E3A0	00
0012E3A4	00

为了输入下一个字符,我们运行起来。



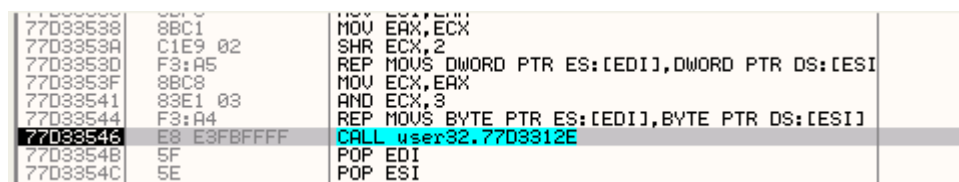
```
File View Debug Plugins Options Window Help
[Icons] [L] [E] [M] [T] [W] [H] [C] [/] [K] [B] [R]
00402811 . F3:A5 REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00402813 . 89C1 MOV ECX,EAX
00402815 . 83E1 03 AND ECX,3
00402818 . 83C6 03 ADD ESI,3
0040281B . 83C7 03 ADD EDI,3
0040281E . F3:A4 REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00402820 . FC CLD
00402821 . 5F POP EDI
00402822 . 5E POP ESI
00402823 . C3 RETN
00402824 . 83EC 08 SUB ESP,8
00402827 . DF3C24 FISTP QWORD PTR SS:[FSP1]
```

这里可能是正确的序列号。



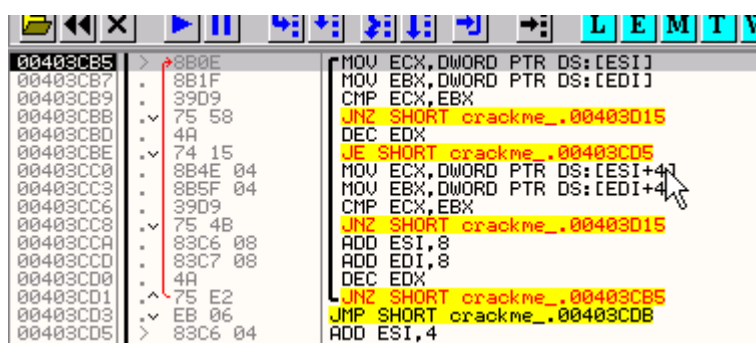
```
00954270 1B 00 00 00 01 00 00 00 .....
00954278 07 00 00 00 4E 61 72 76 ...Narv
00954280 61 6A 61 00 16 00 00 00 aja....
00954288 01 00 00 00 06 00 00 00 0....
00954290 34 31 38 35 30 37 00 40 418507.
00954298 74 B4 45 00 F4 96 95 00 t+E.000.
009542A0 E4 53 00 00 00 E4 F5 12 3S...33#
009542A8 00 00 01 00 00 80 3E 00 ..0..>.
009542B0 00 78 01 39 00 F0 56 95 .x09.-U0
009542B8 00 78 F4 12 00 D6 26 40 .x00.i&0
009542C0 00 8F 46 44 00 00 00 00 .AFD....
009542C8 00 BE 6C 44 00 8C 3F 00 .%ID.I?..
009542D0 00 6F 61 44 00 01 00 00 .oaD.0..
009542D8 00 38 1E 95 00 0C 01 00 .8A0..0.
009542E0 00 38 1E 95 00 58 0F 00 .8A0.%%.
009542E8 00 94 47 95 00 E4 F5 12 .0G0.33#
009542F0 00 A2 FD 44 00 00 00 00 .0^D....
009542F8 00 00 00 00 00 03 00 00 .....0..
```

继续运行。



```
77D33538 . 8BC1 MOV EAX,ECX
77D3353A . C1E9 02 SHR ECX,2
77D3353D . F3:A5 REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
77D3353F . 8BC8 MOV ECX,EAX
77D33541 . 83E1 03 AND ECX,3
77D33544 . F3:A4 REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
77D33546 . E8 E3FBFFFF CALL user32.77D3312E
77D3354B . 5F POP EDI
77D3354C . 5E POP ESI
```

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。



```
[Icons] [L] [E] [M] [T] [W] [H] [C] [/] [K] [B] [R]
00403CB5 . 8B0E MOV ECX,DWORD PTR DS:[ESI]
00403CB7 . 8B1F MOV EBX,DWORD PTR DS:[EDI]
00403CB9 . 39D9 CMP ECX,EBX
00403CBB . 75 58 JNZ SHORT crackme_.00403D15
00403CBD . 4A DEC EDX
00403CBE . 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 . 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 . EB 06 JMP SHORT crackme_.00403CDB
00403CD5 . 83C6 04 ADD ESI,4
```

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。