

## 第十二章-消息断点

本章将重点介绍 Windows 消息。

下面的引言简要的描述了 Windows 消息的概念:

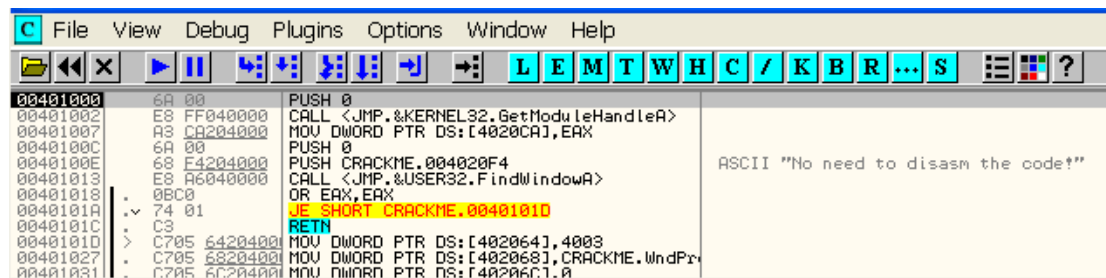
Windows 消息被广泛用于各种事件的通知上面,如果你想操作窗口或者控件(UI 元素其实也是一种窗口,如:按钮,编辑框,工具栏,树形控件等)的话,给它发送消息即可。消息也可以来至于其他应用程序。你也可以通过消息来实现系统通知,移动鼠标,按下键盘上的某键等操作。

正如我们前面所讨论的,OD 中大部分的 API 函数我们可以使用普通 CC 断点来下断,但是少数检测 CC 断点的情况,使用消息断点会更加有效。消息断点在内核调试器 SoftICE 中也称为 BMSG。

Windows 窗口程序至少有一个消息循环,消息循环有特定的 API 函数构成,最常见的是 GetMessage 和 DispatchMessage 函数,有的消息循环也会用到其他的 API 函数。想要深入了解 Windows 的消息的话,可以参考下面的链接中的教程”理解消息循环”(该教程的中文版见附件):

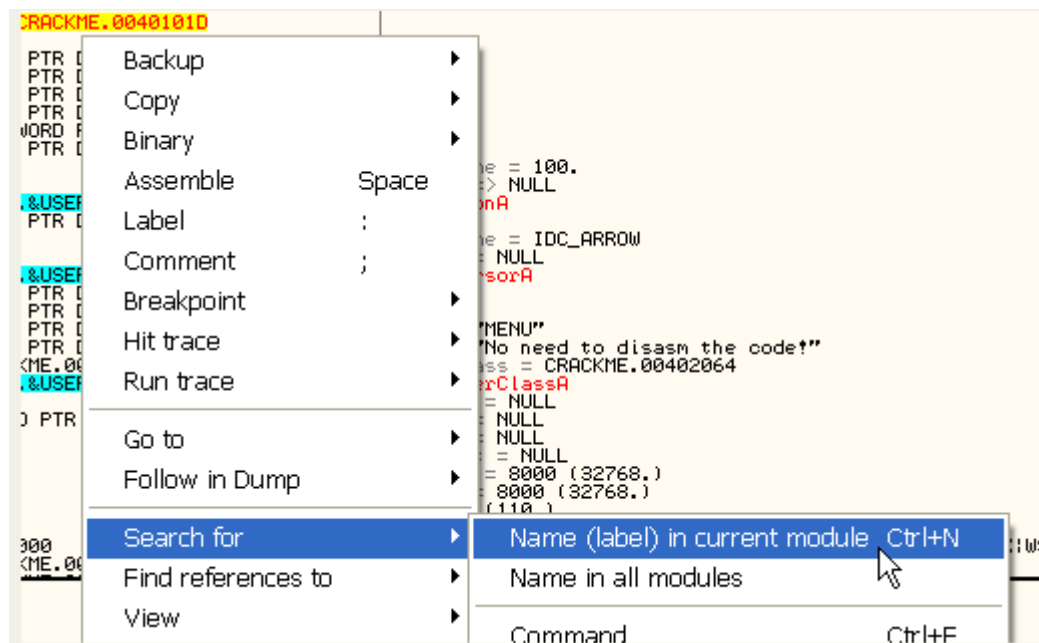
[http://winprog.org/tutorial/message\\_loop.html](http://winprog.org/tutorial/message_loop.html)

让我们来看一个简单的例子:用 OD 加载 CrueHead`'s 的 CrackMe。



首先我们尝试第一种提取序列号的方法,然后再来尝试消息断点提取序列号的方法。我们来看看导入到程序的 API 函数,看看有没有获取输入文本的函数。

在反汇编窗口中单击鼠标右键选择-Search-Name(label) in current module。

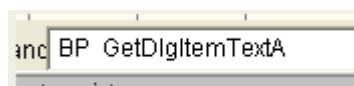


获取编辑框中文本我们通常使用的 API 是 GetDlgItemTextA 或者 GetWindowTextA。当然,也可以使用 Unicode 版的 API 函数 GetDlgItemTextW 或者 GetWindowTextW,再者,也可以发送消息直接获取编辑框中文本。但是,不要指望从 GetDlgItemTextA 或者 GetWindowTextA 下手获取一些保护强度比较高的编辑框控件中的文本。但是我们还是先来看看这种方法吧。

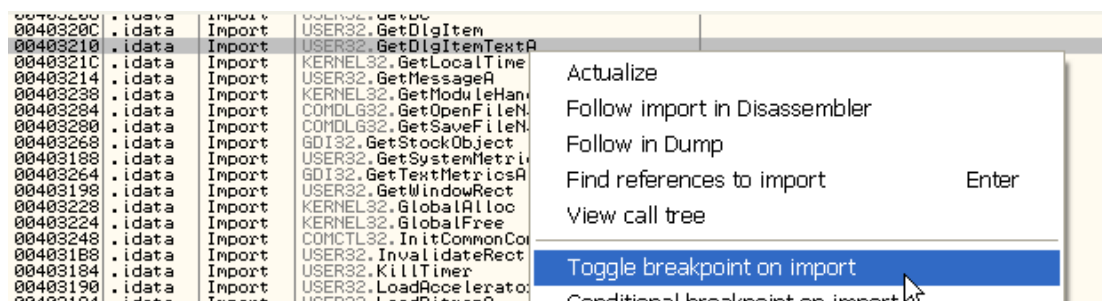
0040326C	.idata	Import	GDI32.EndPage
00403200	.idata	Import	USER32.EndPaint
00403240	.idata	Import	KERNEL32.ExitProcess
00403204	.idata	Import	USER32.FindWindowA
00403208	.idata	Import	USER32.GetDC
0040320C	.idata	Import	USER32.GetDlgItem
00403210	.idata	Import	USER32.GetDlgItemTextA
0040321C	.idata	Import	KERNEL32.GetLocalTime
00403214	.idata	Import	USER32.GetMessageA
00403238	.idata	Import	KERNEL32.GetModuleHandleA
00403284	.idata	Import	COMDLG32.GetOpenFileNameA
00403280	.idata	Import	COMDLG32.GetSaveFileNameA
00403260	.idata	Import	GDI32.GetStockObject

尽管该列表中有 GetDlgItemTextA,但是并不意味着这个函数就是用来读取用户输入的用户名和序列号的(可能仅仅是获取用户输入的其他字段的)。有可能作者是故意添加该函数来误导我们的。还是就是,该 API 函数可以通过各种不同的方式来动态加载,不一定要通过导入表。因此,可能 CrackMe 真的使用的是 GetDlgItemTextA,但是我们在导入表中找不到这个函数。为了不把问题复杂化,我们假设 CrackMe 就是使用 GetDlgItemTextA 来获取编辑框中文本的。

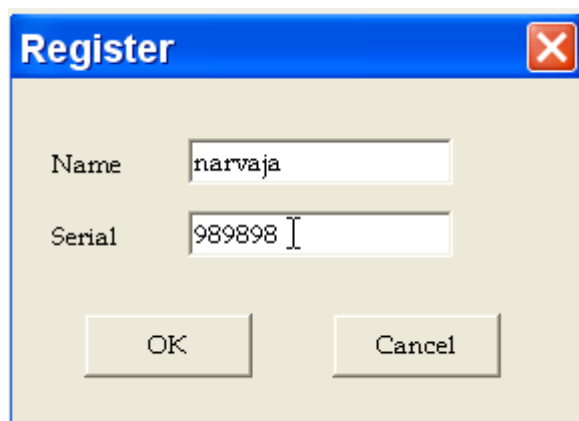
我们在命令栏中使用 BP GetDlgItemTextA 设置断点:



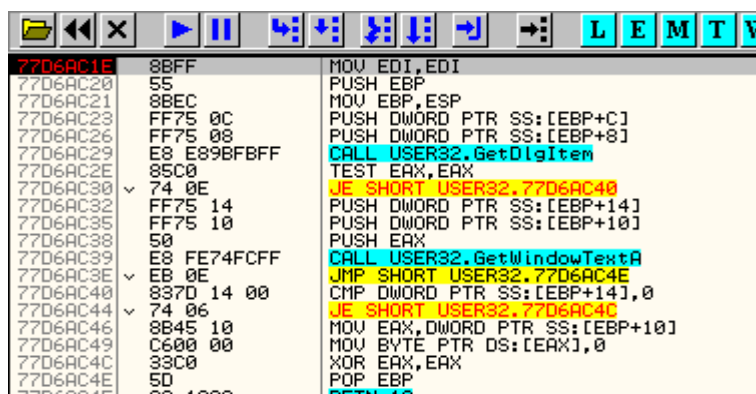
或者



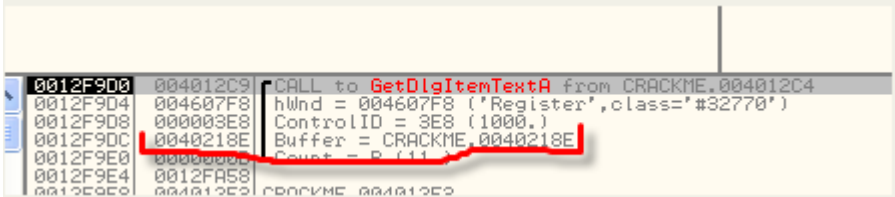
设置了断点以后,运行程序,输入用户名和序列号:



单击 OK,程序断在我们设置的断点处。

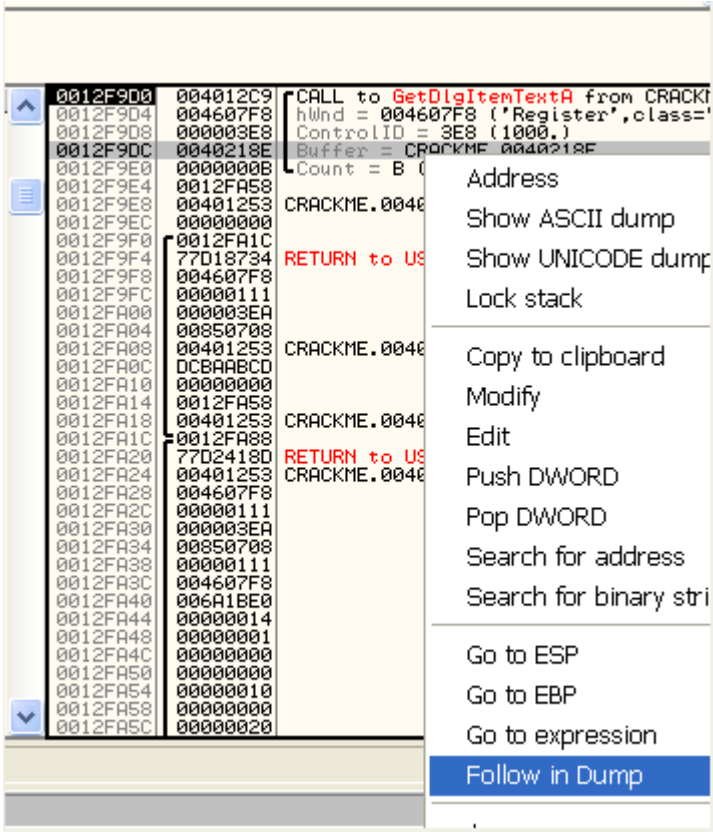


注意堆栈。



可以看到该函数有一个参数是缓冲区,编辑框中的内容会被拷贝至该缓冲区。

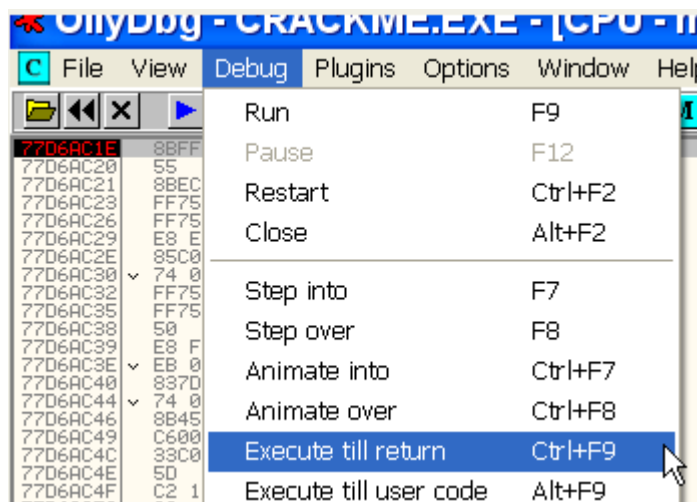
所以,我们在数据窗口中定位到该缓冲区,堆栈窗口中选中该缓冲区参数,单击鼠标右键选择-Follow in Dump。 或者在数据窗口中单击鼠标右键选择-Goto-Expression 输入 40218E。



现在缓冲区还是空的,因为该函数还没有被执行。

Address	Hex dump	ASCII
0040218E	00 00 00 00 00 00 00 00	.....
00402196	00 00 00 00 00 00 00 00	.....
0040219E	00 00 00 00 00 00 00 00	.....
004021A6	00 00 00 00 00 00 00 00	.....
004021B6	00 00 00 00 00 00 00 00	.....
004021BE	00 00 00 00 00 00 00 00	.....
004021C6	00 00 00 00 00 00 00 00	.....
004021CE	00 00 00 00 00 00 00 00	.....
004021D6	00 00 00 00 00 00 00 00	.....
004021DE	00 00 00 00 00 00 00 00	.....
004021E6	00 00 00 00 00 00 00 00	.....
004021EE	00 00 00 00 00 00 00 00	.....
004021F6	00 00 00 00 00 00 00 00	.....
004021FE	00 00 00 00 00 00 00 00	.....
00402206	00 00 00 00 00 00 00 00	.....

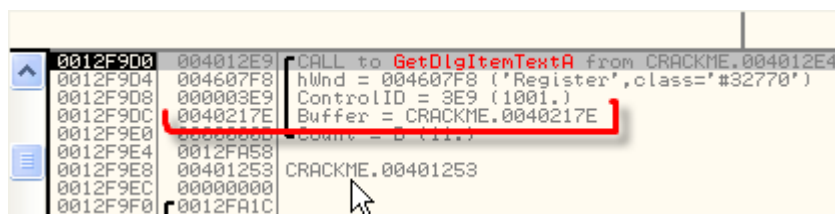
我们通过选择主菜单项 Debug-Execute till return 来执行该函数。



现在缓冲区中保存了我们在注册窗口中输入的用户名。

Address	Hex dump	ASCII
0040218E	6E 61 72 76 61 6A 61 00	narvaja.
00402196	00 00 00 00 00 00 00 00	.....
0040219E	00 00 00 00 00 00 00 00	.....
004021A6	00 00 00 00 00 00 00 00	.....
004021AE	00 00 00 00 00 00 00 00	.....
004021B6	00 00 00 00 00 00 00 00	.....
004021BE	00 00 00 00 00 00 00 00	.....
004021C6	00 00 00 00 00 00 00 00	.....

接着,按 F9 键运行程序,会再次触发我们的断点。



现在,缓冲区参数的地址为 40217E,我们在数据窗口中转到这个地址:

Address	Hex dump	ASCII
0040217E	00 00 00 00 00 00 00 00	.....
00402186	00 00 00 00 00 00 00 00	.....
0040218E	6E 61 72 76 61 6A 61 00	narvaja.
00402196	00 00 00 00 00 00 00 00	.....
0040219E	00 00 00 00 00 00 00 00	.....
004021A6	00 00 00 00 00 00 00 00	.....

我们依然选择主菜单项 Debug-Execute till return,执行到返回。

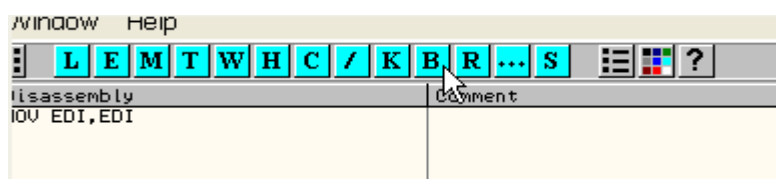
Address	Hex dump	ASCII
0040217E	39 38 39 38 39 38 00 00	989898..
00402186	00 00 00 00 00 00 00 00	.....
0040218E	6E 61 72 76 61 6A 61 00	narvaja.
00402196	00 00 00 00 00 00 00 00	.....
0040219E	00 00 00 00 00 00 00 00	.....

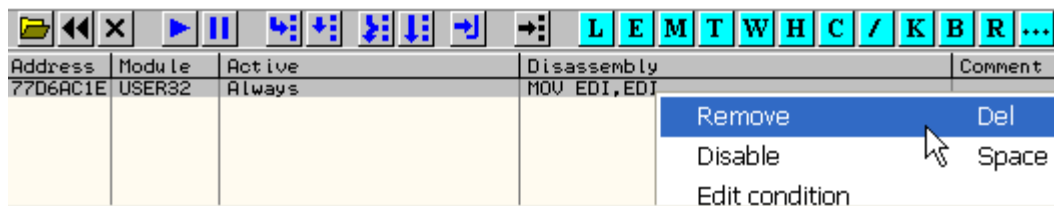
这是我们输入的序列号。

整个过程想必很清楚了,为了找到正确的序列号,在程序获取我们输入数据(这里是用户名和序列号)的时候应该让其中断下来。

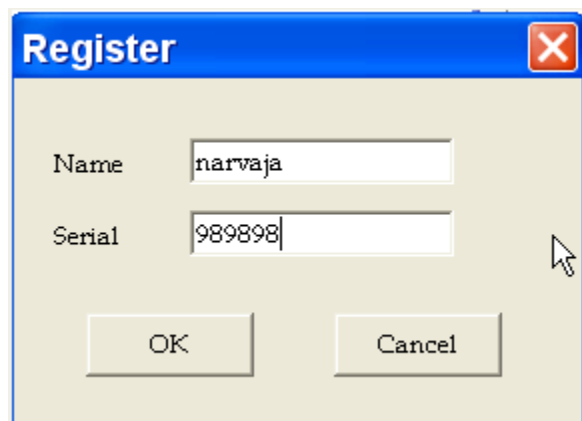
更进一步的分析我们后面再讨论。我们现在再通过消息断点来提取序列号。很多有经验的程序员不使用 API 函数来获取编辑框中文本,而是直接通过发送消息来获取编辑框中的文本。

我们单击工具栏中【B】按钮删除所有的普通 CC 断点。



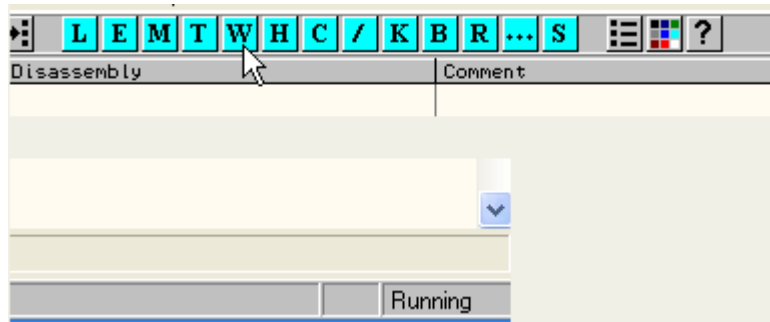


F9 键将程序运行起来,打开注册窗口输入用户名和序列号,但是不要点确定。



消息断点与普通 CC 断点的区别在于,普通 CC 断点在程序启动之前就可以设置,但是对于消息断点来说,只有在窗口创建之后才能够设置消息断点以及拦截消息。

单击工具栏中的【W】按钮打开 Windows 窗口(并不会暂停程序,依然显示的是运行)。



如果【W】按钮弹出的窗口列表为空,你可以单击鼠标右键选择-Actualize。

Handle	Title	Parent	MinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
004907F8	Register	Topmost			14C800C4	00010101	Main	77D3E54F	#32770
00410728	Cancel	004907F8		000003EB	50010000	00000004	Main	77D3B00E	Button
008D0708	Serial	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
00F20736		004907F8		000003E8	50010000	00000204	Main	77D3B3C4	Edit
011A079C	OK	004907F8		000003EA	50010000	00000004	Main	77D3B00E	Button
011E06B2	Name	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
008E07CC		004907F8		000003E9	50010000	00000204	Main	77D3B3C4	Edit
01840738	CrackMe v1.0	Topmost		00D40718	1CCF0000	00000100	Main	00401128	No need to disasm the code!

我们找到 Class(类名)为 Button,Title(标题)为 OK 的窗口。

我们在找到的窗口这一行单击鼠标右键选择-Message breakpoint on ClassProc。

Handle	Title	Parent	WinProc	ID	Style	ExtStyle	Thread	Cl
004907F8	Register	Topmost			14C800C4	00010101	Main	77
00410728	Cancel	004907F8		000003EB	50010000	00000004	Main	77
00800708	Serial	004907F8		0000FFFF	50020000	00000004	Main	77
00F20736		004907F8		000003E8	50010080	00000204	Main	77
011A079C	OK	004907F8						77
011E06B2	Name	004907F8						77
080E07CC		004907F8						77
01840738	CrackMe v1.0	Topmost						00

- Actualize
- Follow ClassProc
- Toggle breakpoint on ClassProc
- Conditional log breakpoint on ClassProc
- Message breakpoint on ClassProc
- Copy to clipboard

### Set breakpoint on WinProc

Messages: Any message

☒ Break on any window  
☐ Break on all windows with same title  
☐ Break on actual window only (invalid in next session)

Never    On message    Pass count (dec.)  
 Pause program: ☐ ☒ 0.  
 Log WinProc arguments: ☐ ☒

☐ Sort messages by name

OK Cancel

在打开的窗口中,我们展开下拉列表选择我们感兴趣的消息类型:

### Set breakpoint on WinProc

Messages: Any message

☒ Break  
☐ Break  
☐ Break

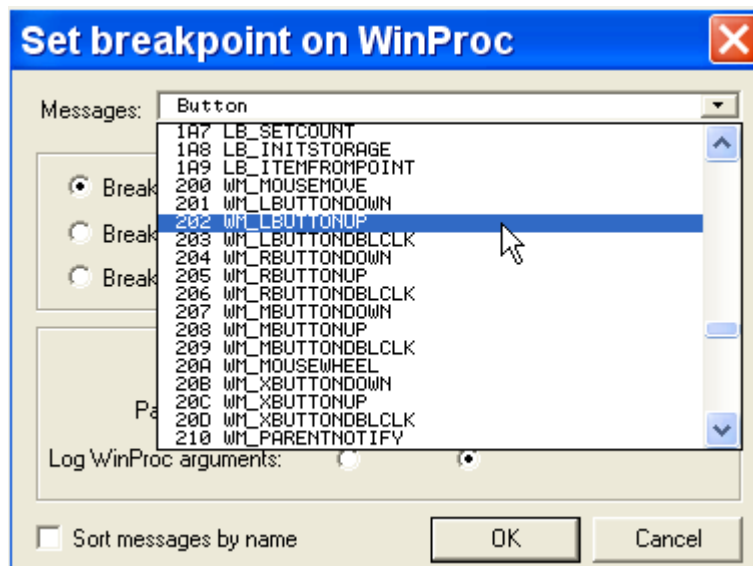
Drawing  
 Scrolling  
 Icon  
 MDI  
 Dialog  
 Menu  
 Text  
 Mouse  
 Keyboard  
 Clipboard  
 Edit control  
 Static control  
 Button  
 Combo box  
 List box  
 IME  
 User-defined  
 000 WM\_NULL

Pa  
 Log WinProc arguments: ☐ ☒

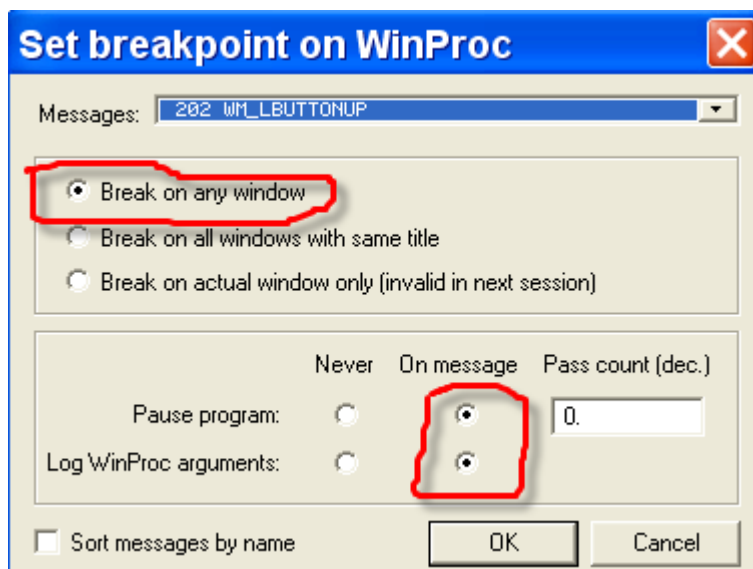
☐ Sort messages by name

OK Cancel

下拉列表显示有静态文本控件,按钮控件,鼠标,剪贴板等类型的消息,如果你不知道需要拦截什么消息的的话,选择第一项 Any Message 即可。这里我们关注消息属于 Button(按钮)这一项。当我们单击鼠标左键的时候,系统会发送 WM\_LBUTTONDOWN 消息(L 代表左边)。当我们松开鼠标左键的时候,系统会发送 WM\_LBUTTONUP 消息。我们设置了消息断点以后,当我们松开鼠标左键的时候,窗口会收到值为 0x202 的消息。



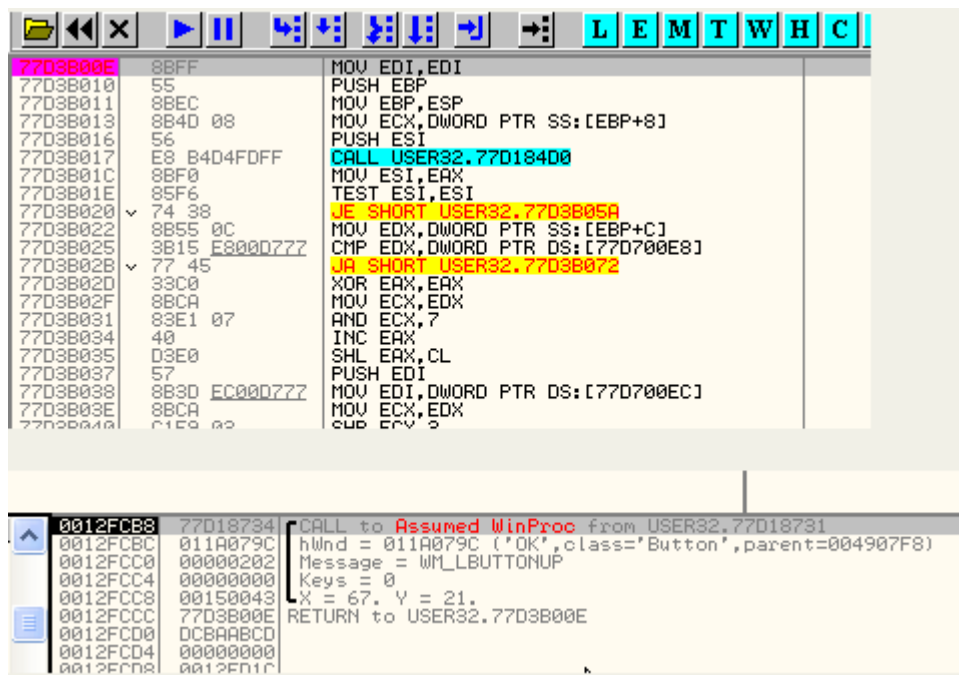
如下:



我们选择值为 0x202 的 WM\_LBUTTONDOWN 消息。并且选择 Break on any window(当前程序的任何窗口接收到该消息都中断),以及 Pause program(中断程序),还要选中下面的 Log WinProc arguments(记录消息过程函数的参数值)。

Handle	Title	Parent	MinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
004907F8	Register	Topmost			14C800C4	00010101	Main	77D3E54F	#32770
00410728	Cancel	004907F8		000003EB	50010000	00000004	Main	77D3E54F	Button
00000708	Serial	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
00F20736		004907F8		000003E8	50010000	00000204	Main	77D3E5C4	Edit
011A079C	OK	004907F8		000003EA	50010000	00000004	Main	77D3E5C4	Button
011E06B2	Name	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
000007CC		004907F8		000003E9	50010000	00000204	Main	77D3E5C4	Edit
01840738	CrackMe v1.0	Topmost		00D40718	1CCF0000	00000100	Main	00401128	No need to disasm the code!

我们可以看到我们选择的 Any window(任意窗口)包括了 OK(确定),Cancel(取消)按钮。我们单击 OK。



到了这里,很多新手可能会犯迷糊,因为我们触发的消息断点断在了一段陌生的代码中(不属于主程序的代码)。实际上,要回到主程序的代码处也很容易。

003C0000	00002000				Map	R	R	
003D0000	00004000				Priv	RW	RW	
003E0000	00002000				Map	R	R	
003F0000	00004000				Priv	RW	RW	
00400000	00001000	CRACKME	PE header		Imag	R	RWE	
00401000	00001000	CRACKME	CODE	code	Imag	R	RWE	
00402000	00001000	CRACKME	DATA	data	Imag	R	RWE	
00403000	00001000	CRACKME	.idata	imports	Imag	R	RWE	
00404000	00001000	CRACKME	.edata	exports	Imag	R	RWE	
00405000	00001000	CRACKME	.reloc	relocations	Imag	R	RWE	
00406000	00002000	CRACKME	.rsrc	resources	Imag	R	RWE	
00410000	00007000				Map	R	R	E

我们知道主程序的代码是 401000 开头的这个区段,我们选中这个区段,单击鼠标右键选择-Set memory breakpoint on access。

003D0000	00004000				Priv	RW	RW	
003E0000	00002000				Map	R	R	
003F0000	00004000				Priv	RW	RW	
00400000	00001000	CRACKME	PE header		Imag	R	RWE	
00401000	00001000	CRACKME	CODE	code	Imag	R	RWE	
00402000	00001000	CRACKME	DATA	data	Imag	R	RWE	
00403000	00001000	CRACKME	.idata	imports	Imag	R	RWE	
00404000	00001000	CRACKME	.edata	exports	Imag	R	RWE	
00405000	00001000	CRACKME	.reloc	relocations	Imag	R	RWE	
00406000	00002000	CRACKME	.rsrc	resources	Imag	R	RWE	
00410000	00007000				Map	R	R	E
004D0000	00002000				Map	R	R	
004E0000	00103000				Map	R	R	
005F0000	000CC000				Map	R	R	
008F0000	00010000				Map	R	R	
00CF0000	00050000				Map	R	R	
00D40000	00001000				Map	R	R	
062D0000	00001000	SSSensor	PE header		Imag	R	RWE	
062D1000	00009000	SSSensor	code		Imag	R	RWE	
062DA000	00002000	SSSensor	.rdata	imports,exp	Imag	R	RWE	
062DC000	00005000	SSSensor	.data	data	Imag	R	RWE	
062E1000	00001000	SSSensor	.shared		Imag	R	RWE	
062E2000	00001000	SSSensor	.rsrc	resources	Imag	R	RWE	
062E3000	00002000	SSSensor	.reloc	relocations	Imag	R	RWE	
58C30000	00001000	COMCTL32	PE header		Imag	R	RWE	

Right-click context menu options:

- Actualize
- View in Disassembler (Enter)
- Dump in CPU
- Dump
- Search (Ctrl+B)
- Set break-on-access (F2)
- Set memory breakpoint on access (highlighted)
- Set memory breakpoint on write

F9 键运行起来,不一会儿程序就断下来了。



不要清除内存访问断点,我们按 F9 键运行,我们发现单步执行了一行,我们继续 F9 单步。

我们一直 F9,直到 RET 返回,然后我们又回到了 401253 处,我们继续 F9,然后就跳转到了我们感兴趣的通过 GetDlgItemTextA 获取用户名和序列号代码的附近。对不对,我们再一次定位到了我们感兴趣的代码,这一次我们并没有直接给 API 下断点。

如果应用程序并不是通过 API 函数来获取用户输入的序列号的话,我们可以通过消息断点来定位,这是消息断点的优点。

为了让我们确定的时候,程序能断下来,我们单击鼠标右键选择-Breakpoint-Remove memory breakpoint 来删除内存访问断点。

我们单击工具栏中【B】按钮打开断点列表窗口,单击鼠标右键选择-Remove 删除所有消息断点。

Address	Module	Active	Disassembly
77D3B00E	USER32	Log "<WinProc>"	MOV EDI,EDI

我们再次运行程序,打开注册窗口,但是这次我们不输入任何东西。

Register

Name

Serial

OK

Cancel

单击工具栏中的[W]按钮打开口列表。

Handle	Title	Parent	WinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
004907F8	Register	Topmost			14C800C4	00010101	Main	77D3E54F	#32770
00410728	Cancel	004907F8		000003EB	50010000	00000004	Main	77D3B00E	Button
00000708	Serial	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
00F20736		004907F8		000003E8	50010000	00000204	Main	77D3B3C4	Edit
011A079C	OK	004907F8		000003EA	50010001	00000004	Main	77D3B00E	Button
011E06E2	Name	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
000E07CC		004907F8		000003E9	50010000	00000204	Main	77D3B3C4	Edit
01840738	CrackMe v1.0	Topmost		00D40718	1CCF0000	00000100	Main	00401128	No need to disasm the code!

重复之前的步骤,但是这次我们选择:

Set breakpoint on WinProc

Messages:

Any message

0F5 BM\_CLICK

0F6 BM\_GETIMAGE

0F7 BM\_SETIMAGE

100 WM\_KEYDOWN

101 WM\_KEYUP

102 WM\_CHAR

103 WM\_DEADCHAR

104 WM\_SYSKEYDOWN

105 WM\_SYSKEYUP

106 WM\_SYSCHAR

107 WM\_SYSDEADCHAR

109 WM\_WNT\_CONVERTREQUESTEX

10A WM\_CONVERTREQUEST

10B WM\_CONVERTRESULT

10C WM\_INTERIM

10D WM\_IME\_STARTCOMPOSITION

10E WM\_IME\_ENDCOMPOSITION

10F WM\_IME\_COMPOSITION

Break

Break

Break

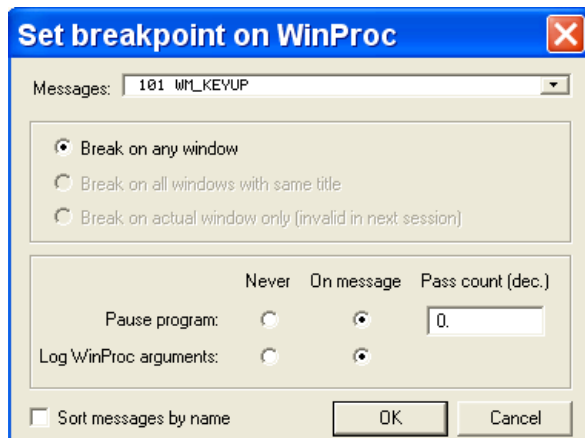
Log WinProc arguments:

Sort messages by name

OK

Cancel

值为 0x101 的 WM\_KEYUP 消息-当按下键盘上面的某个键的时候产生该消息。

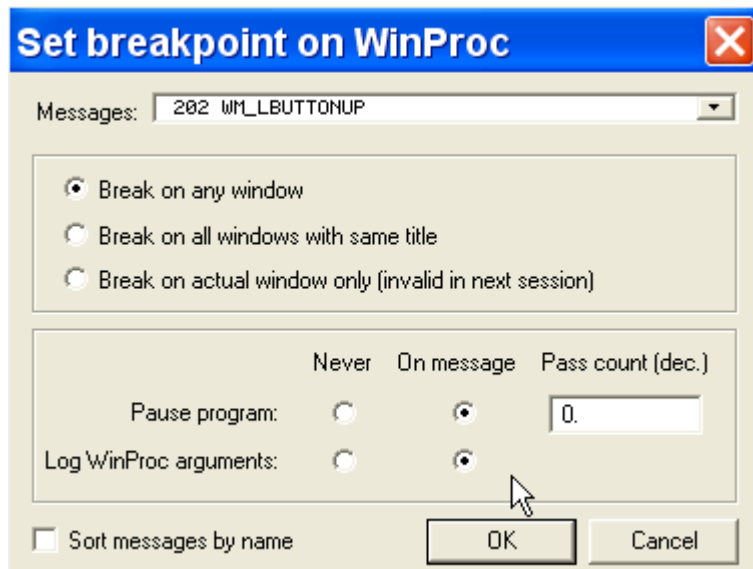


Handle	Title	Parent	WinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
004907F8	Register	Topmost			14C800C4	00010101	Main	77D3E54F	#32770
00410728	Cancel	004907F8		000003EB	50010000	00000004	Main	77D3B00E	Button
008D0708	Serial	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
00F20736		004907F8		000003EB	50010000	00000204	Main	77D3B3C4	Edit
011A079C	OK	004907F8		000003EB	50010001	00000004	Main	77D3B00E	Button
011E06B2	Name	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
000E07CC		004907F8		000003EB	50010000	00000204	Main	77D3B3C4	Edit
01840738	CrackMe v1.0	Topmost		00D40718	1CCF0000	00000100	Main	00401128	No need to c

当我们输入用户名的第一个字符的时候,消息断点并没有触发,因为当前程序并没有通过这种方式来获取用户输入的用户名和序列号,但是有些程序员喜欢通过这种方式来获取用户输入的信息,我们不妨考虑一下这种可能性。

现在我们有疑问,我们只想在 OD 中记录下程序接受到消息,但是不希望程序中断下来。我们该怎么做呢? 我可以使用另一种形式的消息断点。

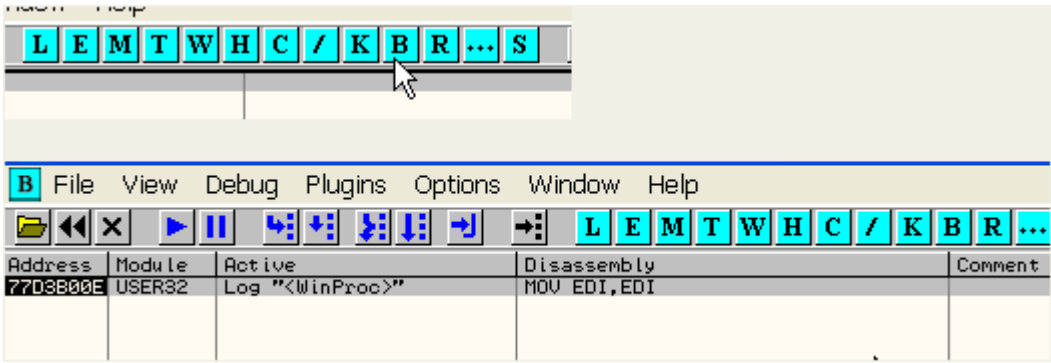
我们选择工具栏中的【B】按钮打开断点窗口,删除所有断点。然后设置一个针对于值为 0x202 的 WM\_LBUTTONDOWN 的消息断点。接着我们注册窗口中的 OK 按钮。



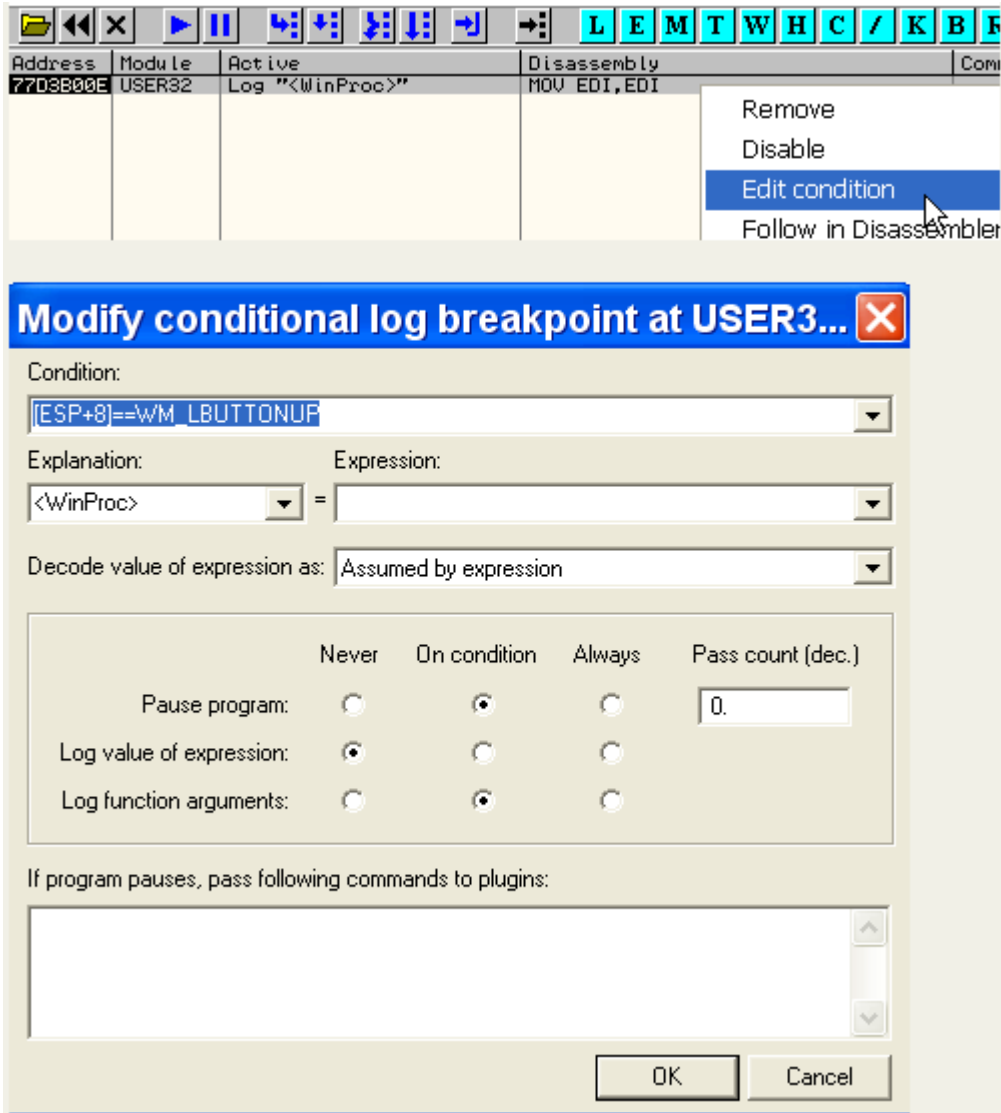
77D3B00E	8BFF	MOV EDI,EDI
77D3B010	55	PUSH EBP
77D3B011	8BEC	MOV EBP,ESP
77D3B013	8B4D 08	MOV ECX,DWORD PTR SS:[EBP+8]
77D3B016	56	PUSH ESI
77D3B017	E8 B404FDFF	CALL USER32.77D18400
77D3B01C	8BF0	MOV ESI,EAX
77D3B01E	85F6	TEST ESI,ESI
77D3B020	74 38	JE SHORT USER32.77D3B050
77D3B022	8B55 0C	MOV EDX,DWORD PTR SS:[EBP+C]
77D3B025	3B15 E800D777	CMP EDX,DWORD PTR DS:[77D700E8]
77D3B028	77 45	JAE SHORT USER32.77D3B072
77D3B02D	33C0	XOR EAX,EAX

我们设置的内存断点触发了。现在我们来改进一下,让其能捕捉所有的消息并且记录到日志中。

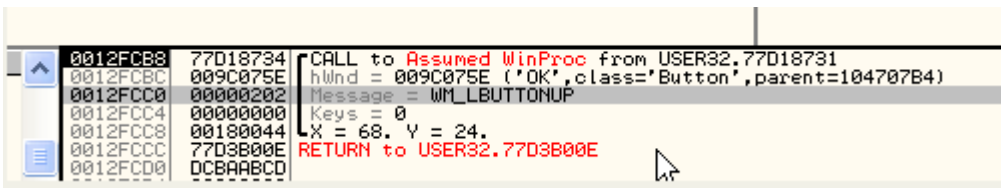
我们单击工具栏中的[B]按钮打开断点列表。



在我们设置的消息断点这一行上单击鼠标右键选择-Edit condition.

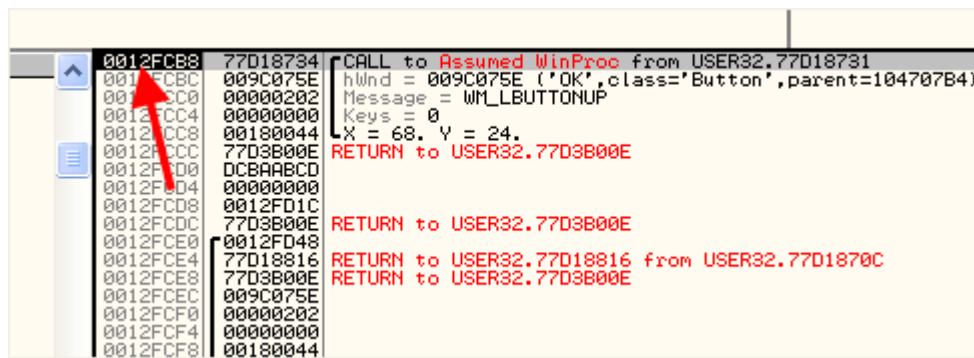


我们可以看到消息断点实际上也是一个条件断点,当前条件为[ESP + 8] == 0x202,即 WM\_LBUTTONDOWN。我们看看堆栈的情况:

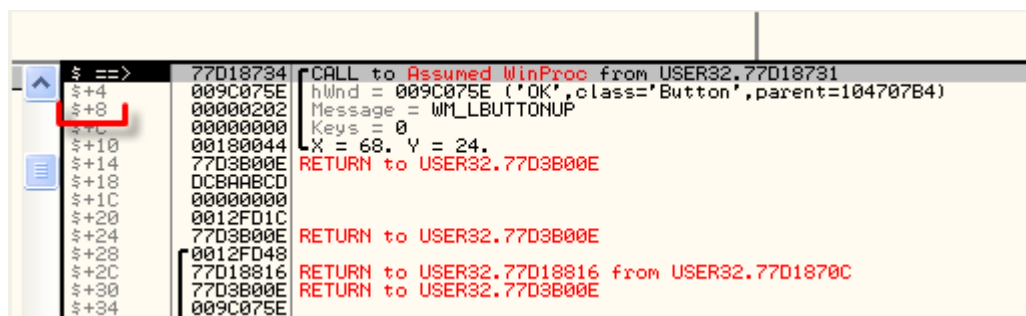


ESP+8 存放的值为 0x202,正好触发消息断点。

如果你不清楚[ESP+8],请双击栈顶地址:

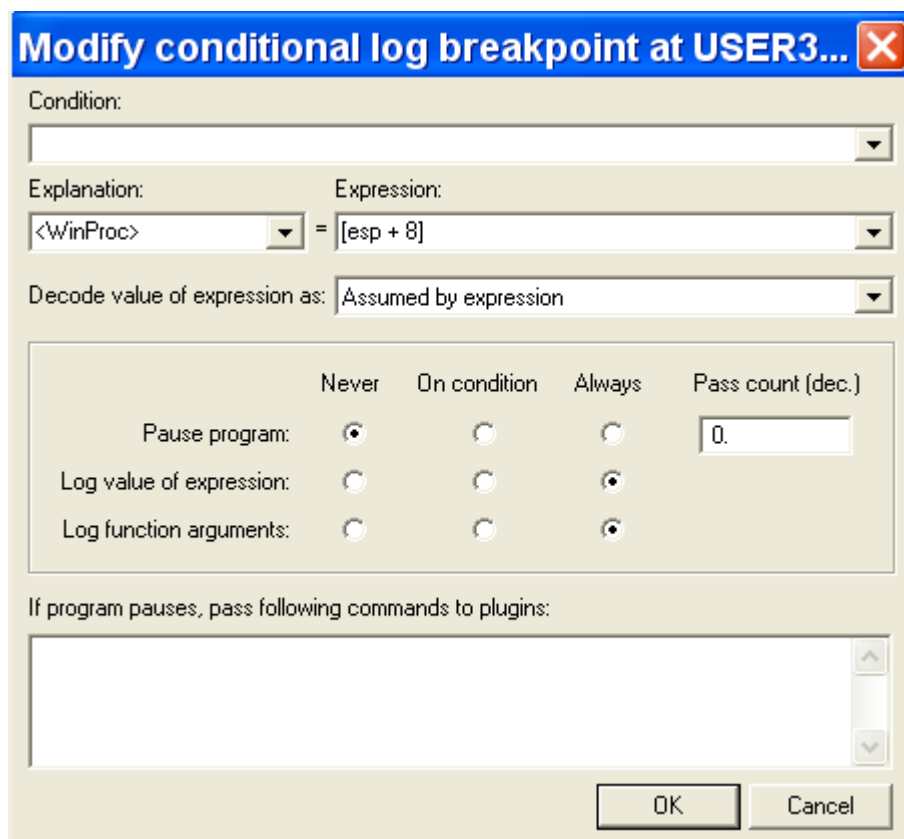


我们看看栈顶:



现在栈址是相对 ESP 显示的,\$+8 相当于 ESP+8。

[ESP+8]的值对应的就是消息的值,我们现在想在日志中记录当前消息,所以改变条件断点参数如下:



Expression 编辑框我们填上[ESP + 8],然后 Pause program(中断程序)选择 Never(不中断),Log value of expression(记录表达式的值)

选择 Always(总是记录),Log function arguments(记录函数参数)也选择 Always。

单击 OK,然后打开注册窗口输入用户名和序列号,单击 OK,接着来看看日志中的消息:

```
77D3B00E COND: <WinProc> = 00000087
77D3B00E CALL to Assumed WinProc from USER32.77D18731
      hWnd = 00F60728 ('OK',class='Button',parent=01270778)
      Message = WM_GETDLGCODE
      wParam = 0
      lParam = 0
77D3B00E COND: <WinProc> = 00000087
77D3B00E CALL to Assumed WinProc from USER32.77D18731
      hWnd = 041507DA ('Cancel',class='Button',parent=01270778)
      Message = WM_GETDLGCODE
      wParam = 0
      lParam = 0
77D3B00E COND: <WinProc> = 000000F4
77D3B00E CALL to Assumed WinProc from USER32.77D18731
      hWnd = 00F60728 ('OK',class='Button',parent=01270778)
      Message = BM_SETSTYLE
      Style = BS_DEFPUSHBUTTON
      Redraw = TRUE
77D3B00E COND: <WinProc> = 00000201
77D3B00E CALL to Assumed WinProc from USER32.77D18731
      hWnd = 00F60728 ('OK',class='Button',parent=01270778)
      Message = WM_LBUTTONDOWN
      Keys = MK_LBUTTON
      X = 46, Y = 3
77D3B00E COND: <WinProc> = 00000007
77D3B00E CALL to Assumed WinProc from USER32.77D18731
      hWnd = 00F60728 ('OK',class='Button',parent=01270778)
      Message = WM_SETFOCUS
      hWndLose = 00A3075E (class='Edit',parent=01270778)
      lParam = 0
77D3B00E COND: <WinProc> = 000000F3
77D3B00E CALL to Assumed WinProc from USER32.77D18731
      hWnd = 00F60728 ('OK',class='Button',parent=01270778)
      Message = BM_SETSTATE
      Highlight = TRUE
      lParam = 0
77D3B00E COND: <WinProc> = 00000200
77D3B00E CALL to Assumed WinProc from USER32.77D18731
      hWnd = 00F60728 ('OK',class='Button',parent=01270778)
      Message = WM_MOUSEMOVE
      Keys = MK_LBUTTON
      X = 47, Y = 10
77D3B00E COND: <WinProc> = 000000F3
77D3B00E CALL to Assumed WinProc from USER32.77D18731
      hWnd = 00F60728 ('OK',class='Button',parent=01270778)
      Message = BM_SETSTATE
      Highlight = TRUE
      lParam = 0
77D3B00E COND: <WinProc> = 00000202
77D3B00E CALL to Assumed WinProc from USER32.77D18731
      hWnd = 00F60728 ('OK',class='Button',parent=01270778)
      Message = WM_LBUTTONUP
      Keys = 0
      X = 47, Y = 10
77D3B00E COND: <WinProc> = 000000F3
77D3B00E CALL to Assumed WinProc from USER32.77D18731
      hWnd = 00F60728 ('OK',class='Button',parent=01270778)
      Message = BM_SETSTATE
      Highlight = FALSE
      lParam = 0
77D3B00E COND: <WinProc> = 00000008
77D3B00E CALL to Assumed WinProc from USER32.77D18731
      hWnd = 00F60728 ('OK',class='Button',parent=01270778)
      Message = WM_KILLFOCUS
      hWndGet = 01270778 ('Register',class='#32770')
      lParam = 0
77D3B00E COND: <WinProc> = 00000002
77D3B00E CALL to Assumed WinProc from USER32.77D18731
      hWnd = 00F60728 ('OK',class='Button',parent=01270778)
```

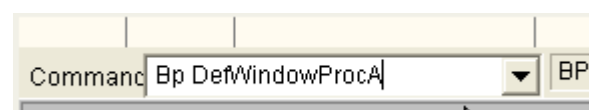
在日志窗口中,我们可以看到首先是值为 0x201 的 WM\_LBUTTONDOWN 消息,然后又是值为 0x202 的 WM\_LBUTTONUP 消息。

没有 WM\_KEYDOWN 和 WM\_KEYUP 消息,因为我们并没有按键盘上的键。

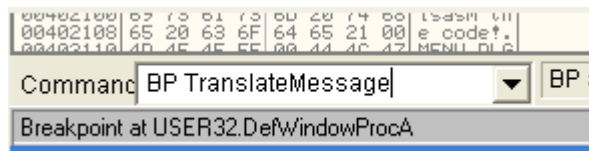
为了记录下程序接收到的(按钮,输入的文本内容)等所有信息,我们可以对消息处理函数 TranslateMessage 或者 DefWindowProcA 设置条件断点。

如果想完整的记录下这两个 API 函数的参数信息,我们可以

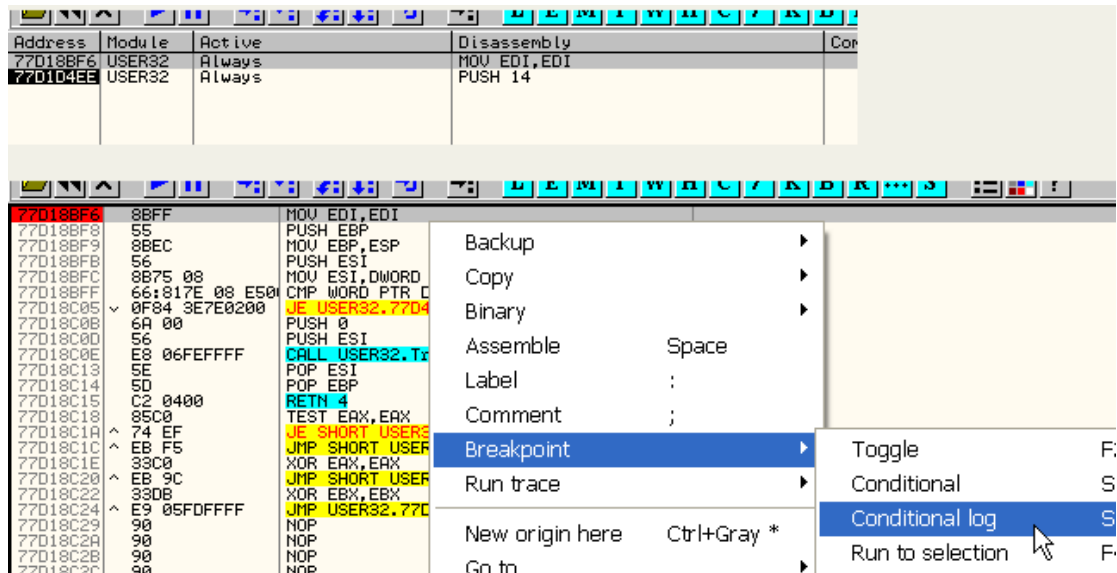
通过命令栏 BP 给 TranslateMessage 和 DefWindowProcA 设置断点。



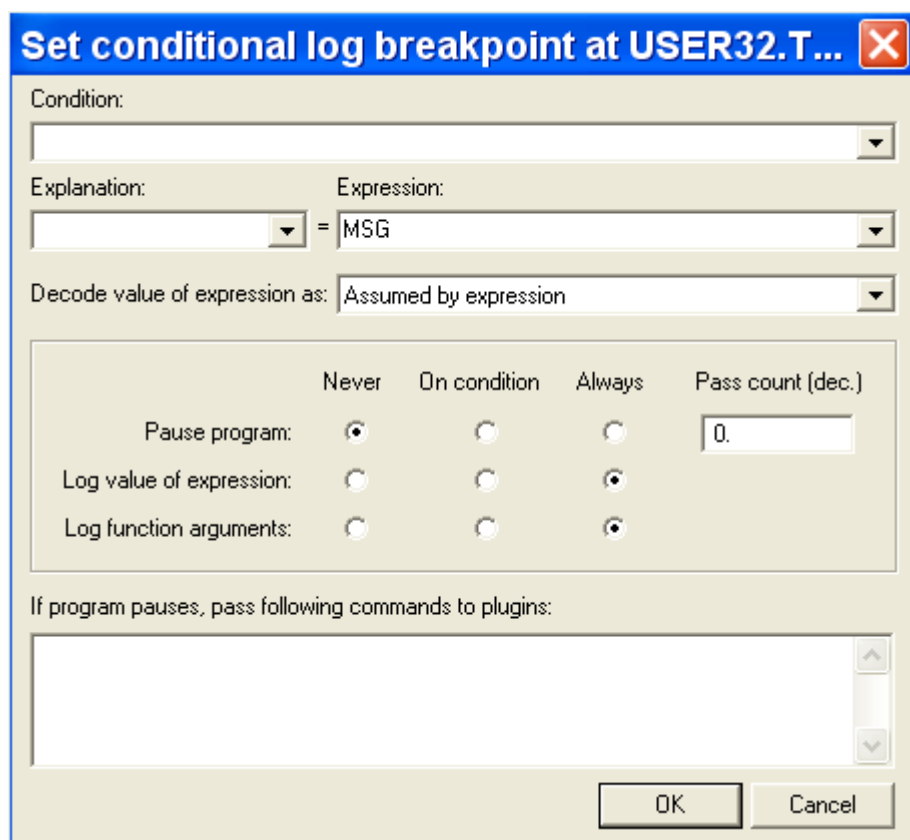
和



这样我们就成功的给这两个 API 函数设置了断点,接下来我们给这两个断点设置条件。我们单击工具栏中的【B】按钮打开断点列表窗口,然后在第一个断点上单击鼠标右键选择-Follow in disassembler。



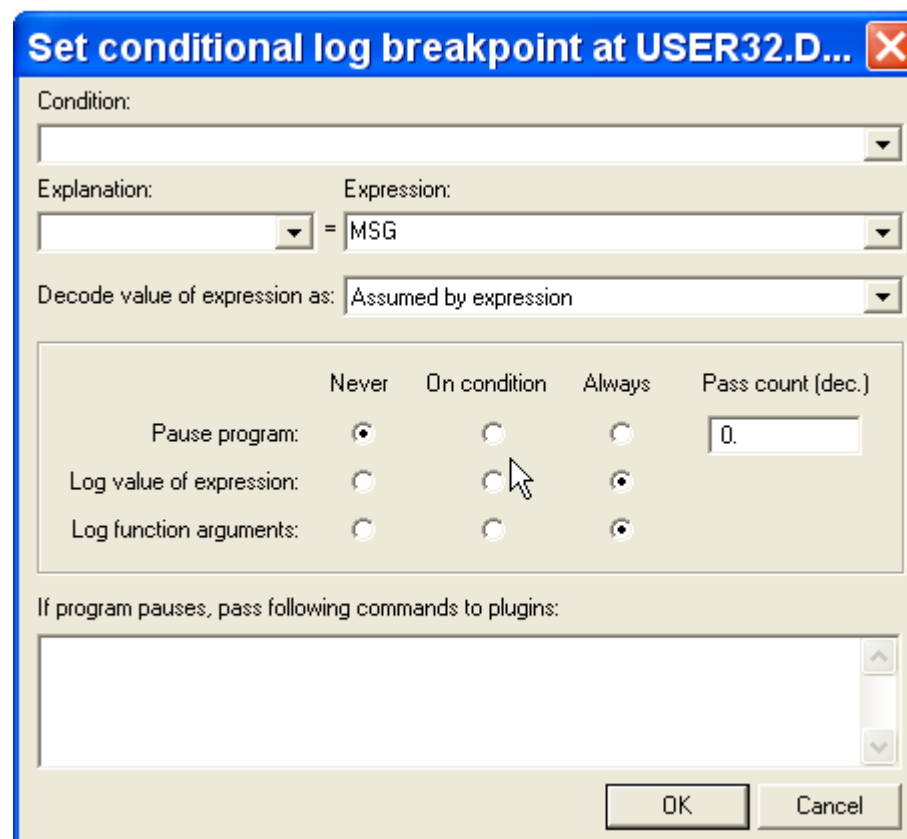
然后断点这一行上单击鼠标右键选择-Conditional log。



Expression 编辑框我们填上 MSG例如:WM\_LBUTTONDOWN 的值为 0x202,那么 MSG 就等于 0x202。

然后 Pause program 选择 Never,Log value of expression 选择 Always,Log function arguments 选择 Always。接着同样的设置第二个

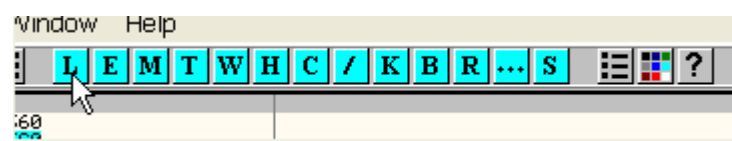
断点。



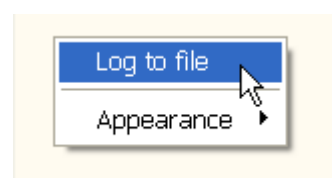
这样,我们的条件断点就设置好了。

Address	Module	Active	Disassembly
77D18BF6	USER32	Log	MOV EDI,EDI
77D1D4EE	USER32	Log	PUSH 14

因为记录的结果可能会很多,所以我们最好是把它们保存到文件中。



在日志窗口中单击鼠标右键选择-Log to file。



添加上文本文件的名称。我们单击运行。



	Message = WM_ENTERIDLE Source = MSGF_MENU hWnd = 031507F8 (class='#32768',parent=00140742) COND: 00000121 CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!') Message = WM_ENTERIDLE Source = MSGF_MENU hWnd = 031507F8 (class='#32768',parent=00140742) COND: 00000121 CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!') Message = WM_ENTERIDLE Source = MSGF_MENU hWnd = 031507F8 (class='#32768',parent=00140742) COND: 00000085 CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!') Message = WM_NCPAINT Region = E0041935 (region) lParam = 0 COND: 00000014 CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!') Message = WM_ERASEBKGD hDC = 9401167D lParam = 0 COND: 00000125 CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!') Message = WM_UNINITMENUPOPUP hMenu = 005406F4 ID = 0 COND: 0000011F CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!') Message = WM_MENUSELECT Item = 0, Flags = MF_BYPOSITION MF_SEPARATOR 3 MF_BITMAP MF_OWNERDRAW MF_POPUP MF_ME hMenu = NULL COND: 00000212 CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!') Message = WM_EXITMENULOOP IsPopup = FALSE lParam = 0 COND: 00000086 CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!') Message = WM_NCACTIVATE Active = FALSE lParam = 0 COND: 00000006 CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!') Message = WM_ACTIVATE WA_INACTIVE Minimized = 0 hWnd = NULL COND: 0000001C CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!') Message = WM_ACTIVATEAPP Activate = FALSE ThreadId = AC0 COND: 00000008 CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!') Message = WM_KILLFOCUS hWndGet = NULL lParam = 0 COND: 0000000F CALL to TranslateMessage from CRACKME.0040118C pMsg = WM_PAINT hw = 140742 ("CrackMe v1.0") COND: 0000000F CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!') Message = WM_PAINT
--	---

现在,我们日志中就记录了窗口过程函数接收到的所有消息。有个这个日志文件,我们就可以在其中挑选我们感兴趣的消息。然后设置相应的消息断点来印证我们的猜想。

在以后的内容你会逐步体会到运行消息断点来去除 NAG 等各种保护机制的方便之处的。

下一章,我们来提取 CrueHead'a 的 CrackMe 的正确序列号。