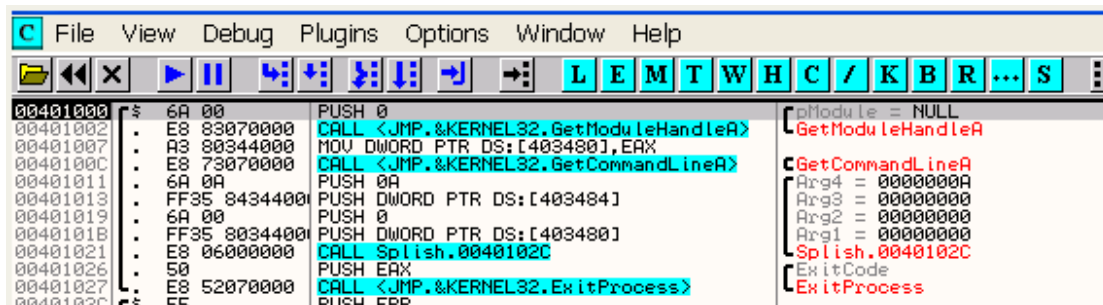


## 第十五章-硬编码序列号寻踪-Part3

我们来接着完成上一章留下那个硬编码 CrackMe 的作业,名字叫”Splish”。

用 OD 加载它。

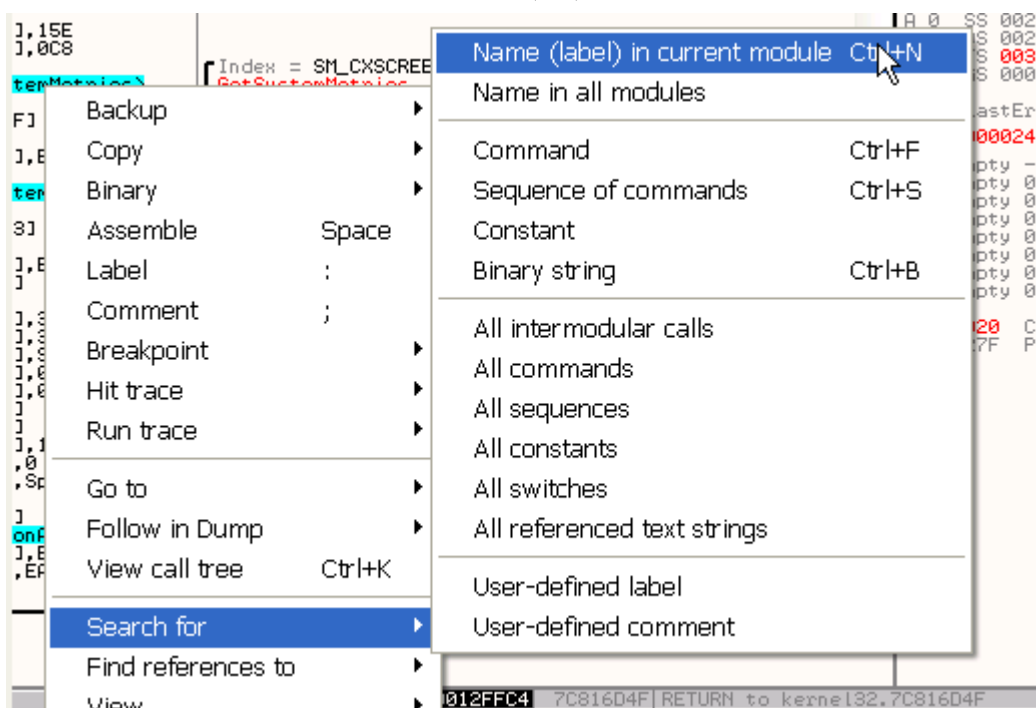


```
00401000 6A 00 PUSH 0
00401002 E8 83070000 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 A3 80344000 MOV DWORD PTR DS:[403480],EAX
0040100C E8 73070000 CALL <JMP.&KERNEL32.GetCommandLineA>
00401011 6A 0A PUSH 0A
00401013 FF35 84344000 PUSH DWORD PTR DS:[403484]
00401019 6A 00 PUSH 0
0040101B FF35 80344000 PUSH DWORD PTR DS:[403480]
00401021 E8 06000000 CALL Splish.0040102C
00401026 50 PUSH EAX
00401027 E8 52070000 CALL <JMP.&KERNEL32.ExitProcess>
0040102C 55 PUSH EBP
```

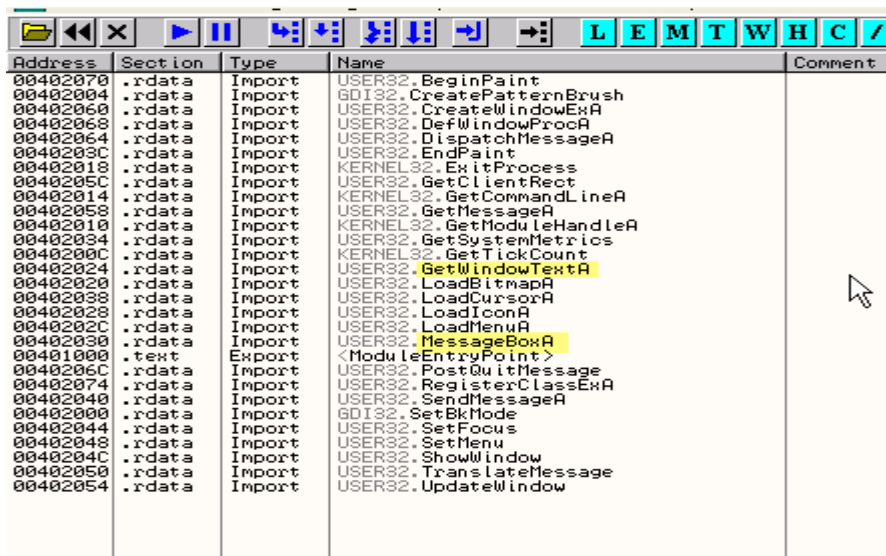
phModule = NULL  
GetModuleHandleA  
GetCommandLineA  
Arg4 = 0000000A  
Arg3 = 00000000  
Arg2 = 00000000  
Arg1 = 00000000  
Splish.0040102C  
ExitCode  
ExitProcess

OD 加载后停在了入口点处。

我们通过在反汇编窗口中点击鼠标右键选择-Search for-Name(label)in current module 查看 API 函数列表。

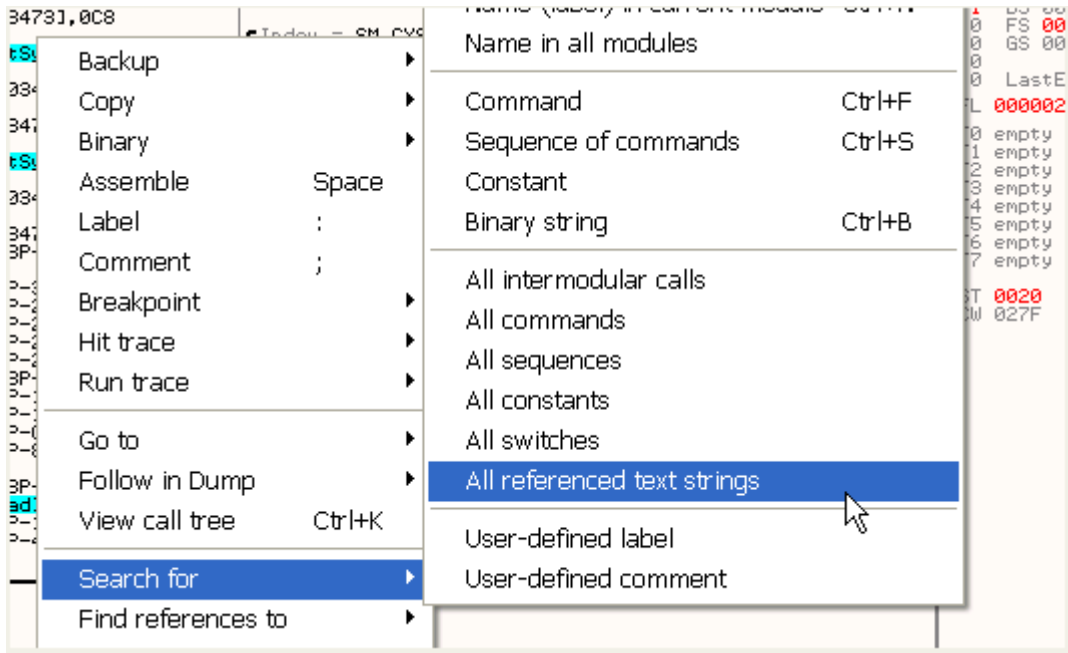


我们可以看到当前模块使用了哪些 API 函数。



Address	Section	Type	Name	Comment
00402070	.rdata	Import	USER32.BeginPaint	
00402004	.rdata	Import	GDI32.CreatePatternBrush	
00402060	.rdata	Import	USER32.CreateWindowExA	
00402068	.rdata	Import	USER32.DefWindowProcA	
00402064	.rdata	Import	USER32.DispatchMessageA	
0040203C	.rdata	Import	USER32.EndPaint	
00402018	.rdata	Import	KERNEL32.ExitProcess	
0040205C	.rdata	Import	USER32.GetClientRect	
00402014	.rdata	Import	KERNEL32.GetCommandLineA	
00402058	.rdata	Import	USER32.GetMessageA	
00402010	.rdata	Import	KERNEL32.GetModuleHandleA	
00402034	.rdata	Import	USER32.GetSystemMetrics	
0040200C	.rdata	Import	KERNEL32.GetTickCount	
00402024	.rdata	Import	USER32.GetWindowTextA	
00402020	.rdata	Import	USER32.LoadBitmapA	
00402038	.rdata	Import	USER32.LoadCursorA	
00402028	.rdata	Import	USER32.LoadIconA	
0040202C	.rdata	Import	USER32.LoadMenuA	
00402030	.rdata	Import	USER32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	
0040206C	.rdata	Import	USER32.PostQuitMessage	
00402074	.rdata	Import	USER32.RegisterClassExA	
00402040	.rdata	Import	USER32.SendMessageA	
00402000	.rdata	Import	GDI32.SetBkMode	
00402044	.rdata	Import	USER32.SetFocus	
00402048	.rdata	Import	USER32.SetMenu	
0040204C	.rdata	Import	USER32.ShowWindow	
00402050	.rdata	Import	USER32.TranslateMessage	
00402054	.rdata	Import	USER32.UpdateWindow	

我们可以看到使用了 GetWindowTextA 来获取序列号,MessageBoxA 来提示序列号正确或者错误。我们可以给这两个 API 设置断点,这里我们先来看看该程序的字符串列表。



单击鼠标右键选择-Search for-All referenced text strings。

Address	Disassembly	Text string
00401000	PUSH 0	(Initial CPU selection)
00401005	MOV DWORD PTR SS:[EBP-8],Splish.0040300A	ASCII "OurWindow"
0040110B	PUSH Splish.0040300A	ASCII "Splish, Splash"
00401110	PUSH Splish.00403000	ASCII "OurWindow"
004011B1	PUSH Splish.00403006	ASCII "Hard Coded:"
004011B6	PUSH Splish.00403060	ASCII "static"
004011DC	PUSH Splish.00403092	ASCII "Name:"
004011E1	PUSH Splish.00403060	ASCII "static"
00401207	PUSH Splish.00403098	ASCII "Serial:"
0040120C	PUSH Splish.00403060	ASCII "static"
00401237	PUSH Splish.00403058	ASCII "edit"
0040126A	PUSH Splish.00403058	ASCII "edit"
00401290	PUSH Splish.00403058	ASCII "edit"
004012D9	PUSH Splish.00403020	ASCII "Check Hardcoded"
004012DE	PUSH Splish.00403019	ASCII "button"
0040130F	PUSH Splish.00403030	ASCII "Name/Serial Check"
00401314	PUSH Splish.00403019	ASCII "button"
00401353	ASCII "HardCoded",0	
0040138E	ASCII "Congratulations,"	
0040139E	ASCII " you got the har"	
004013AE	ASCII "d coded serial",0	
004013BF	PUSH Splish.0040300A	ASCII "Splish, Splash"
004013C4	PUSH Splish.0040138E	ASCII "Congratulations, you got the hard coded serial"
004013D4	PUSH Splish.0040300A	ASCII "Splish, Splash"
004013D9	PUSH Splish.00403067	ASCII "Sorry, please try again."
00401433	PUSH Splish.0040300A	ASCII "Splish, Splash"
00401438	PUSH Splish.004030D9	ASCII "Your mission is to disable the Splash Screen, find the hardcode"
0040147F	ASCII "Splash_Class",0	
0040148C	PUSH Splish.00403000	ASCII "MyBmp"
004014D0	MOV DWORD PTR SS:[EBP-8],Splish.0040147F	ASCII "Splash_Class"
00401528	PUSH Splish.0040300A	ASCII "Splish, Splash"
0040152D	PUSH Splish.0040147F	ASCII "Splash_Class"
00401600	PUSH Splish.0040300A	ASCII "Splish, Splash"
00401605	PUSH Splish.004030A0	ASCII "Please enter your name."
00401695	PUSH Splish.0040300A	ASCII "Splish, Splash"
0040169A	PUSH Splish.0040306B	ASCII "Please enter your serial number."
004016CF	PUSH Splish.0040300A	ASCII "Splish, Splash"
004016D4	PUSH Splish.00403042	ASCII "Good job, now keygen it."
004016E4	PUSH Splish.0040300A	ASCII "Splish, Splash"
004016E9	PUSH Splish.00403067	ASCII "Sorry, please try again."

这里我们可以看到提示输入了正确硬编码序列号的字符串-"Gongratulations, you got the hard coded serial"。我们在这个字符串上面双击鼠标左键,就可以来到引用了该字符串的代码处。

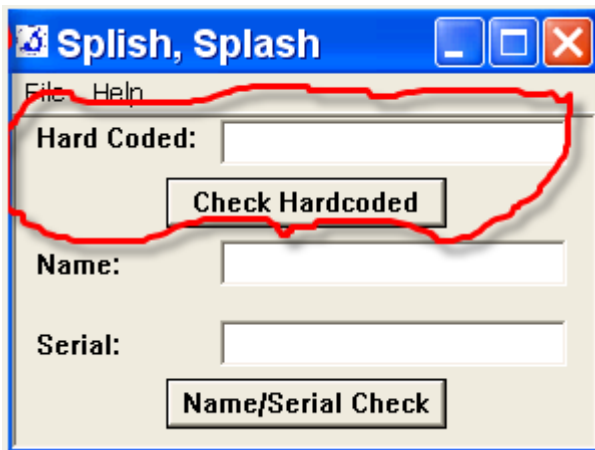
我们可以看到来到了验证序列号的代码块了。

00401351	EB 04	JMP SHORT Splish.00401350	
00401353	48 61 72 64	ASCII "HardCoded",0	Count = 20 (32.)
0040135D	6A 20	PUSH 20	Buffer = Splish.00403215
0040135F	68 15324000	PUSH Splish.00403215	hWnd = NULL
00401364	FF35 90344001	PUSH DWORD PTR DS:[4034901]	GetWindowTextA
0040136A	E8 B0300000	CALL <JMP.&USER32.GetWindowTextA>	
0040136F	8D05 53134001	LEA EAX,DWORD PTR DS:[401353]	
00401375	8D1D 15324001	LEA EBX,DWORD PTR DS:[403215]	
0040137B	0038 00	CMP BYTE PTR DS:[EAX],0	
0040137E	74 0C	JE SHORT Splish.0040138C	
00401380	8A08	MOV CL,BYTE PTR DS:[EAX]	
00401382	8A13	MOV DL,BYTE PTR DS:[EBX]	
00401384	38D1	CMP CL,DL	
00401386	75 4A	JNZ SHORT Splish.004013D2	
00401388	40	INC EAX	
00401389	40	INC EBX	
0040138A	EB EF	JMP SHORT Splish.0040137B	
0040138C	EB 2F	JMP SHORT Splish.0040138D	
0040138E	43 6F 6E 67	ASCII "Congratulations,"	Style = MB_OK!MB_APPLMODAL
0040139E	20 79 6F 75	ASCII " you got the har"	Title = "Splish, Splash"
004013A4	64 20 63 6F	ASCII "d coded serial",0	Text = "Congratulations, you got the hard coded serial"
004013B0	6A 00	PUSH 0	hOwner = NULL
004013BF	68 0A304000	PUSH Splish.0040300A	MessageBoxA
004013C4	68 8E134000	PUSH Splish.0040138E	
004013C9	6A 00	PUSH 0	
004013CB	E8 78030000	CALL <JMP.&USER32.MessageBoxA>	
004013D0	EB 13	JMP SHORT Splish.004013E5	
004013D2	6A 00	PUSH 0	Style = MB_OK!MB_APPLMODAL
004013D4	68 0A304000	PUSH Splish.0040300A	Title = "Splish, Splash"
004013D9	68 67304000	PUSH Splish.00403067	Text = "Sorry, please try again."
004013DE	6A 00	PUSH 0	hOwner = NULL
004013E0	E8 63030000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
004013E5	61	POPAD	
004013E6	EB 73	JMP SHORT Splish.0040145B	
004013E8	6643F8 02	CMP AX,2	
004013EC	75 1B	JNZ SHORT Splish.00401409	

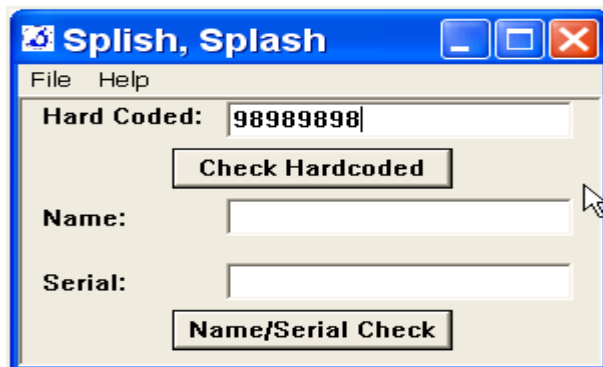
可以看到使用 GetWindowTextA 获取用户输入的序列号,然后使用 MessageBoxA 来提示用户输入的序列号正确与否。我们在获取用户输入的序列号 GetWindowTextA 的调用处设置一个断点。

00401351	EB 0A	JMP SHORT Splish.0040135D	
00401353	48 61 72 64	ASCII "HardCoded",0	Count = 20 (32.)
0040135D	6A 20	PUSH 20	Buffer = Splish.00403215
0040135F	68 15324000	PUSH Splish.00403215	hWnd = NULL
00401364	FF35 90344001	PUSH DWORD PTR DS:[4034901]	GetWindowTextA
0040136A	E8 B0300000	CALL <JMP.&USER32.GetWindowTextA>	
0040136F	8D05 53134001	LEA EAX,DWORD PTR DS:[401353]	
00401375	8D1D 15324001	LEA EBX,DWORD PTR DS:[403215]	
0040137B	0038 00	CMP BYTE PTR DS:[EAX],0	
0040137E	74 0C	JE SHORT Splish.0040138C	
00401380	8A08	MOV CL,BYTE PTR DS:[EAX]	
00401382	8A13	MOV DL,BYTE PTR DS:[EBX]	
00401384	38D1	CMP CL,DL	
00401386	75 4A	JNZ SHORT Splish.004013D2	
00401388	40	INC EAX	
00401389	40	INC EBX	
0040138A	EB EF	JMP SHORT Splish.0040137B	
0040138C	EB 2F	JMP SHORT Splish.0040138D	
0040138E	43 6F 6E 67	ASCII "Congratulations,"	Style = MB_OK!MB_APPLMODAL
0040139E	20 79 6F 75	ASCII " you got the har"	Title = "Splish, Splash"
004013A4	64 20 63 6F	ASCII "d coded serial",0	Text = "Congratulations, you got the hard coded serial"
004013B0	6A 00	PUSH 0	hOwner = NULL
004013BF	68 0A304000	PUSH Splish.0040300A	MessageBoxA
004013C4	68 8E134000	PUSH Splish.0040138E	
004013C9	6A 00	PUSH 0	
004013CB	E8 78030000	CALL <JMP.&USER32.MessageBoxA>	
004013D0	EB 13	JMP SHORT Splish.004013E5	
004013D2	6A 00	PUSH 0	Style = MB_OK!MB_APPLMODAL
004013D4	68 0A304000	PUSH Splish.0040300A	Title = "Splish, Splash"
004013D9	68 67304000	PUSH Splish.00403067	Text = "Sorry, please try again."
004013DE	6A 00	PUSH 0	hOwner = NULL
004013E0	E8 63030000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
004013E5	61	POPAD	
004013E6	EB 73	JMP SHORT Splish.0040145B	
004013E8	6643F8 02	CMP AX,2	
004013EC	75 1B	JNZ SHORT Splish.00401409	

按 F9 键运行 CrackMe。



为了找到上面的硬编码序列号,我们随便输入一个错误的序列号,然后单击 Check Hardcoded 按钮。



我们可以看到断在了之前设置的断点处。

00401353	>	48 61 72 64	ASCII "HardCoded",0	
0040135D	>	6A 20	PUSH 20	Count = 20 (32.)
0040135F	.	68 15324000	PUSH Splish.00403215	Buffer = Splish.00403215
00401364	.	FF35 90344001	PUSH DWORD PTR DS:[4034901]	hWnd = 000B0680 (class='Edit',parent=00150674)
0040136F	.	E8 BB030000	CALL <JMP.<USER32.GetWindowTextA>	GetWindowTextA
0040136F	.	8D05 53134001	LEA EAX,DWORD PTR DS:[401353]	
00401375	.	8D1D 15324001	LEA EBX,DWORD PTR DS:[403215]	
0040137D	>	903C 00	CMP BYTE PTR DS:[EAX],0	
00401382	.	74 0C	JE SHORT Splish.0040138C	
00401388	.	8A08	MOV CL,BYTE PTR DS:[EAX]	
0040138D	.	8D13	MOV DI,BYTE PTR DS:[EAX]	

我们先来看看堆栈中的情况,Buffer 参数指向 403215 地址开始的内存单元,用于保存用户输入的序列号。

0012FC58	000B0680	hWnd = 000B0680 (class='Edit',parent=00150674)
0012FC5C	00403215	Buffer = Splish.00403215
0012FC60	00000020	Count = 20 (32.)
0012FC64	0012FCEC	
0012FC68	00401178	Splish.00401178

我们在数据窗口中定位到该缓冲区

0012FC58	000B0680	hWnd = 000B0680 (class='Edit',parent=00150674)
0012FC5C	00403215	Buffer = Splish.00403215
0012FC60	00000020	Count = 20 (32.)
0012FC64	0012FCEC	
0012FC68	00401178	Splish.00401178

Address	Hex dump	ASCII
00403215	00 00 00 00 00 00 00 00	.....
0040321D	00 00 00 00 00 00 00 00	.....
00403225	00 00 00 00 00 00 00 00	.....
0040322D	00 00 00 00 00 00 00 00	.....
00403235	00 00 00 00 00 00 00 00	.....
0040323D	00 00 00 00 00 00 00 00	.....
00403245	00 00 00 00 00 00 00 00	.....

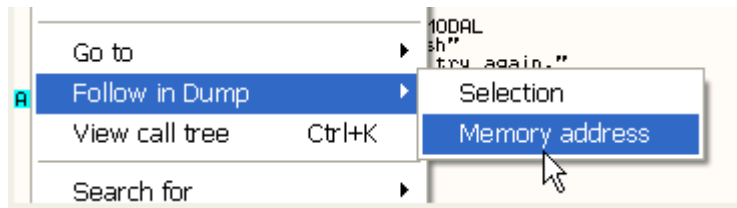
该缓冲区会保存用户输入的序列号。我们按 F8 键单步执行该 API 函数。

Address	Hex dump	ASCII
00403215	39 38 39 38 39 38 39 38	98989898
0040321D	00 00 00 00 00 00 00 00	.....
00403225	00 00 00 00 00 00 00 00	.....
0040322D	00 00 00 00 00 00 00 00	.....

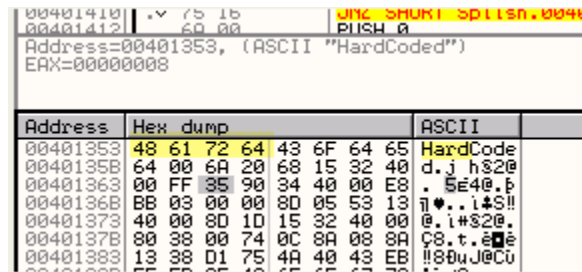
由于单击 F8 键,该 API 函数得以执行,所以序列号保存到了该缓冲区中。

0040136F	.	8D05 53134001	LEA EAX,DWORD PTR DS:[401353]	
00401375	.	8D1D 15324001	LEA EBX,DWORD PTR DS:[403215]	
0040137B	>	903C 00	CMP BYTE PTR DS:[EAX],0	
0040137E	.	74 0C	JE SHORT Splish.0040138C	
00401388	.	8A08	MOV CL,BYTE PTR DS:[EAX]	
0040138D	.	8D13	MOV DI,BYTE PTR DS:[EAX]	

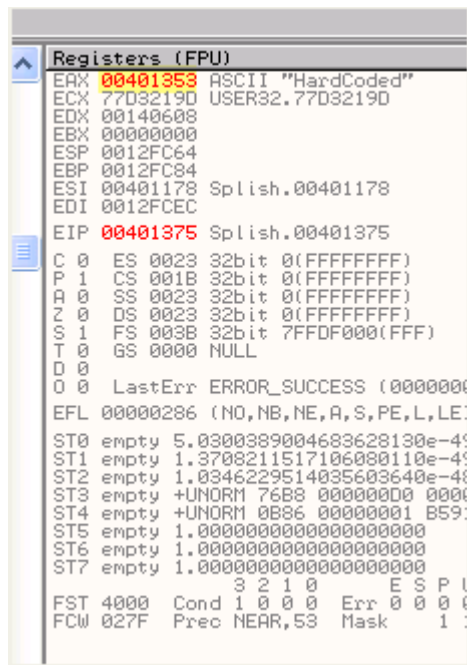
下一条指令会将 401353 保存到 EAX 中(记住,LEA 指令并不是移动指定地址内存单元中的内容,而是移动方括号中的值,这里是 401353)。我们通过在指令上面单击鼠标右键选择-Follow in Dump-Memory address 来在数据窗口中定位到 401353 这个地址。



401353 指向字符串"HardCoded",按 F7 键单步执行 LEA 指令。



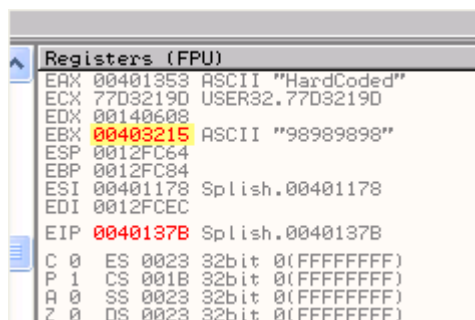
解释窗口中也提示 401353 这个地址指向字符串"HardCoded",执行 LEA 指令后,EAX 保存了 401353 这个地址。



接下来的 LEA 指令,EBX 会保存 403215 这个地址。



按 F7 键单步,EBX 保存 403215。



OD 中显示,403215 这个地址指向字符串”98989898”,也就是我们之前输入的错误序列号,我们像之前一样在数据窗口中转到 403215 这个地址。



可以看到 403215 指向的内存单元中保存了我们输入的错误序列号。

Address	Hex dump	ASCII
00403215	39 38 39 38 39 38 39 38	98989898
0040321D	00 00 00 00 00 00 00 00	.....
00403225	00 00 00 00 00 00 00 00	.....
0040322D	00 00 00 00 00 00 00 00	.....
00401375	8D 1D 15 32 40 00	LEA EBX,DWORD PTR DS:[403215]
0040137B	74 0C	CMP BYTE PTR DS:[EAX],0
0040137E	8A 08	JE SHORT Splish.0040138C
00401380	8B 08	MOV CL,BYTE PTR DS:[EAX]
00401382	8B 08	MOV CL,BYTE PTR DS:[EAX]

下一条指令检查 EAX=401353 这个地址指向内存单元的第一个字节是否为零。

Registers (FPU)	
EAX	00401353 ASCII "HardCoded"
ECX	77D3219D USER32.77D3219D
EDX	00140608
EBX	00403215 ASCII "98989898"
ESP	0012FC64
EBP	0012FC84
ESI	00401178 Splish.00401178
EDI	0012FCEC
EIP	0040137B Splish.0040137B

我们可以在数据窗口中转到 401353 这个地址,可以看到保存了字符串”HardCoded”。

00401412	60 00	PUSH 0
DS:[00401353]=48 ('H')		
Jump from 0040138A		
Address	Hex dump	ASCII
00401353	48 61 72 64 43 6F 64 65	HardCode
0040135B	64 0A 6A 20 6A 15 32 40	d.i h&20

这里第一个字节是 48,解释窗口中提示该 ASCII 码对应的字符是'H',是”HardCoded”字符串的第一个字节,不为零。

EIP	0040137
C 0	ES 002
P 1	CS 001
A 0	SS 002
Z 0	DS 002
S 0	FS 003
T 0	GS 000
D 0	
O 0	TestEr
EFL	0000020

由于'H'不等于零,所以零标志位置 0,JE 不会跳转(记住,JE 指令当零标志位 Z 置 1 的时候跳转)。

继续单击 F7 键单步。

0040137B	74 0C	JE SHORT Splish.0040138C
0040137E	8A 08	MOV CL,BYTE PTR DS:[EAX]
00401380	8B 08	MOV DL,BYTE PTR DS:[EBX]
00401382	3B 01	CMP CL,DL
00401384	75 4A	JNZ SHORT Splish.004013D2
00401386	40	INC EAX
00401388	43	INC EBX
0040138A	EB EF	JMP SHORT Splish.0040137B
0040138C	EB 2F	JMP SHORT Splish.004013B0
0040138E	43 6F 6E 67	ASCII "Congratulations,"
00401390	20 79 6F 75	ASCII " you got the har"
00401392	64 20 63 6F	ASCII "d coded serial",0
00401394	6A 00	PUSH 0
00401396	68 0A304000	PUSH Splish.00403000
00401398	68 8E134000	PUSH Splish.0040138E
0040139A	6A 00	PUSH 0
0040139C	E8 78030000	CALL <JMP.&USER32.MessageBoxA>
0040139E	EB 13	JMP SHORT Splish.004013E5
004013A0	6A 00	PUSH 0
004013A2	68 0A304000	PUSH Splish.00403000
004013A4	68 67304000	PUSH Splish.00403067
004013A6	6A 00	PUSH 0
004013A8	E8 63030000	CALL <JMP.&USER32.MessageBoxA>
004013AA	61	POPAD

Style = MB\_OK!MB\_APPLMODAL  
Title = "Splish, Splash"  
Text = "Congratulations, you got the hard coded serial"  
hOwner = NULL  
MessageBoxA

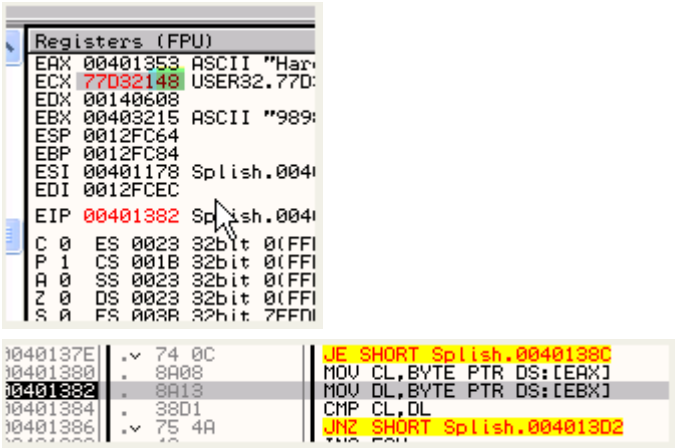
Style = MB\_OK!MB\_APPLMODAL  
Title = "Splish, Splash"  
Text = "Sorry, please try again."  
hOwner = NULL  
MessageBoxA

可以清楚的看到该指令将 EAX 指向内存单元(即字符串”HardCoded”)的第一个字节,这里是 48,保存到 CL 寄存器中,接下来一条指令将 EBX 指向的内存单元(即我们输入的错误序列号)的第一个字节保存到 DL 寄存器中。接着比较这两个字节是否相等,如果不

相等,就跳转到 4013D2 地址处,弹出消息框提示”Sorry, please try again.”。

下图中验证了这一点。

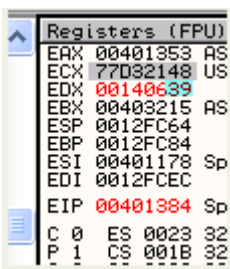
按 F7 键,CL 寄存器的值为 48。



接下来一行 DL 寄存器保存我们输入的错误序列号的第一个字节。



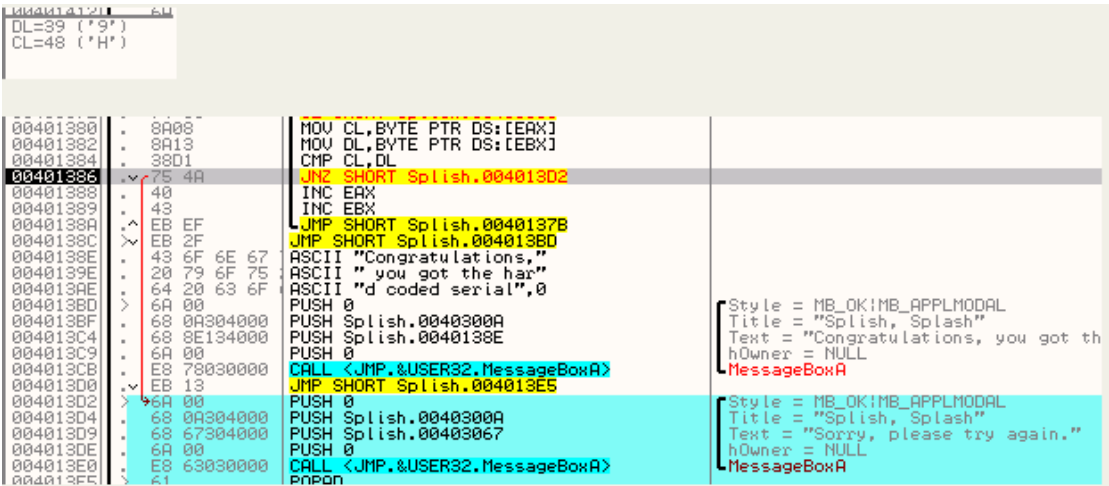
按下 F7 键,可以看到 DL 的值为 39。



现在比较 CL 和 DL 的值。

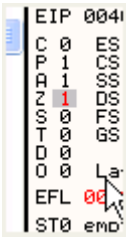


OD 解释窗口中,字符’9’对应的 ASCII 码值为 39,即我们输入的错误序列号的第一个字符与字符’H’对应的 ASCII 码值为 48,即硬编码序列号”HardCoded”的第一个字符,进行比较。





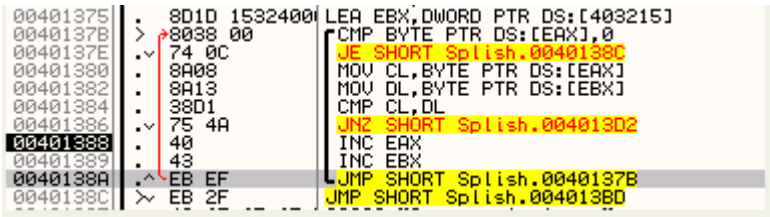
可以看到由于它们不相等,JNZ 指令就会跳转到提示错误信息的代码处。如果它们相等的话,跳转将不会发生,我们可以通过双击零标志位 Z 来修改其值,让跳转不发生。



现在零标志位 Z 置 1 了,表示比较的两个字节是相等的。

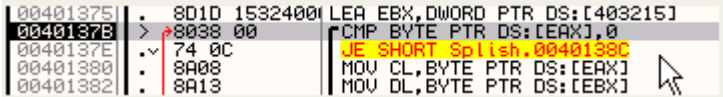


接下来可以看到 EAX,EBX 的值递增 1,然后 JMP 指令又跳转回了循环的开始。

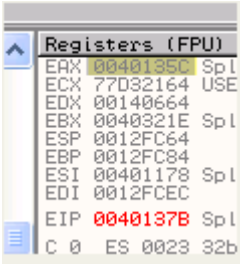


EAX 现在指向字符串”HardCoded”的第二个字节,我们可以看到会逐个字符依次比较,直到 EAX 指向的字节值为 0 为止(字符串结束标志为 0)。

EAX,EBX 分别加 1 以后将比较第二个字节,如果相等,则继续循环比较第 3 个字节,当”HardCoded”字符串都比较完毕了,CL 和 DL 还是相等的时候,就不会跳转到提示错误的消息框代码处。



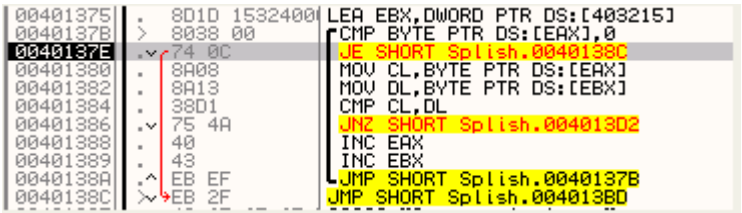
由于现在比较完了”HardCoded”的所有字符,已经到了字符串的末尾,值为 0。



这里都为零,检测序列号结束。

Address	Hex dump	ASCII
00401354	61 72 64 43 6F 64 65 64	ardCoded
0040135C	00 6A 20 68 15 32 40 00	.j h320.
00401364	FF 35 90 34 40 00 E8 BB	5e40.b1

接着 JZ 指令跳出循环。





现在我们到了显示正确消息框的代码块处。(由于我们修改零标志位 Z 的值)

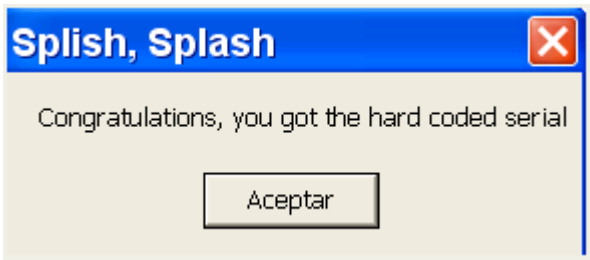
00401389	. 43	INC EBX	
0040138A	. EB EF	JMP SHORT Splish.0040137B	
0040138C	. EB 2F	JMP SHORT Splish.0040138D	
0040138E	. 43 6F 6E 67	ASCII "Congratulations,"	
0040139E	. 20 79 6F 75	ASCII " you got the har"	
0040139F	. 64 20 63 6F	ASCII "d coded serial",0	
004013BD	. 6A 00	PUSH 0	
004013BF	. 68 0A304000	PUSH Splish.0040300A	
004013C4	. 68 8E134000	PUSH Splish.0040138E	
004013C9	. 6A 00	PUSH 0	
004013CB	. E8 70030000	CALL <JMP.&USER32.MessageBoxA>	
004013D0	. EB 13	JMP SHORT Splish.004013E5	

Style = MB\_OK!MB\_APPLMODAL  
Title = "Splish, Splash"  
Text = "Congratulations, you got the hard coded serial"  
hOwner = NULL  
MessageBoxA

每次 CL 与 DL 进行比较的时候,我们通过修改零标志位 Z 的值,让程序以为它们两个是相等的,从而提示正确的消息框。不管怎么说,我们现在已经知道了正确的序列号是"HardCoded"(要注意字母的大小写)



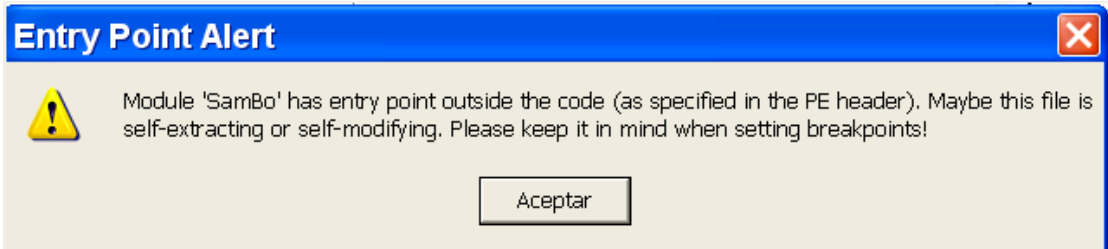
删除之前设置的所有断点,然后按下 Check Hardcoded 按钮。



弹出 Congratulations,you got the hard coded serial 正确序列号的消息框。

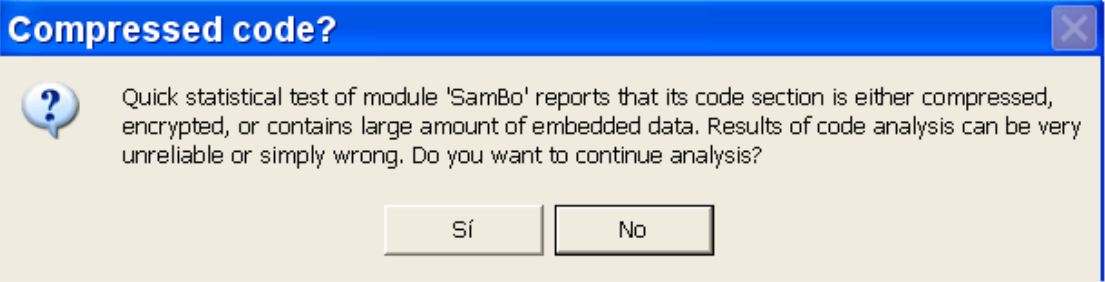
好了,现在来看最后一个硬编码序列号的 CrackMe 例子,第 16 章我们将探讨下一个话题。

这个 CrackMe 与之前稍稍有点不同,名字叫做 SamBo,我们用 OD 加载它。

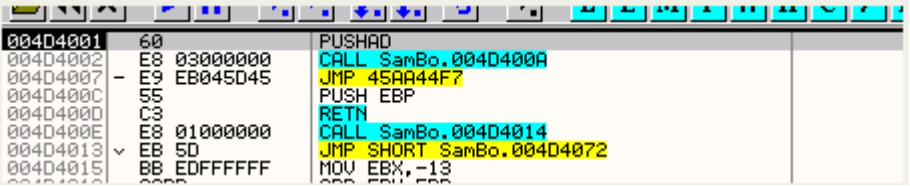


这个窗口提示说“该程序入口点不在代码段,可能是自解压或者自修改文件,这样文件我们称之为被加壳或者被压缩。我们后面会深入探讨壳,这里我们先还是用 OD 加载它,尽管是加过壳的,我们还是可以尝试找到序列号的。

我们同意 OD 警告,将到达入口点处。

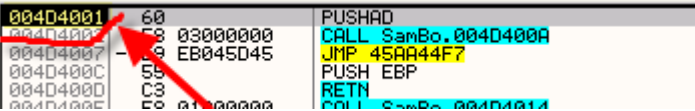


紧接着弹出一个窗口提示“代码段可能被压缩,加密,或者包含大量嵌入数据。代码分析可能是不可靠或者完全错误的。您仍要继续分析吗?”因为程序会在自己脱壳后继续运行原程序代码,所以我们选择 NO。

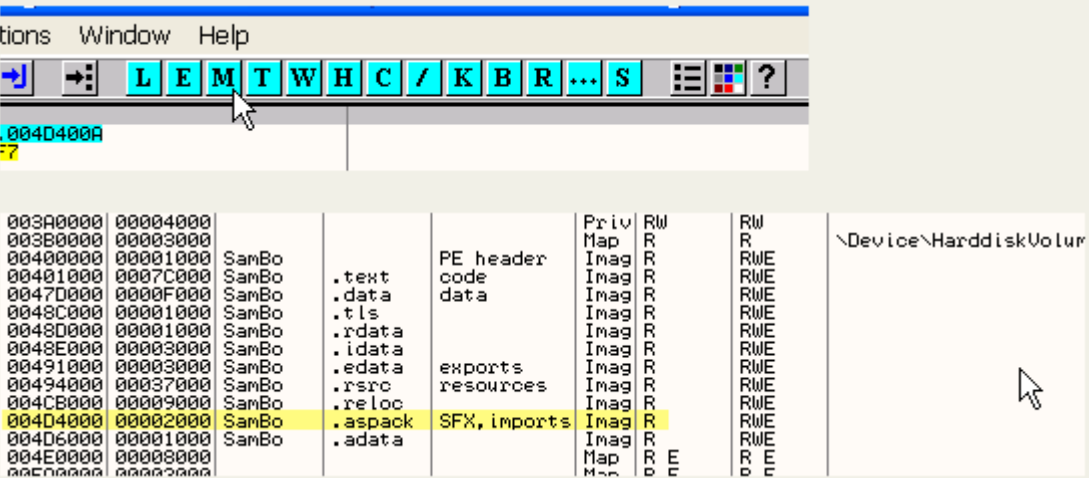


我们可以看到该 CrackMe 并没有像未加壳之前一样停在.text 节的 401000 处。

现在的入口点是 4D4001。

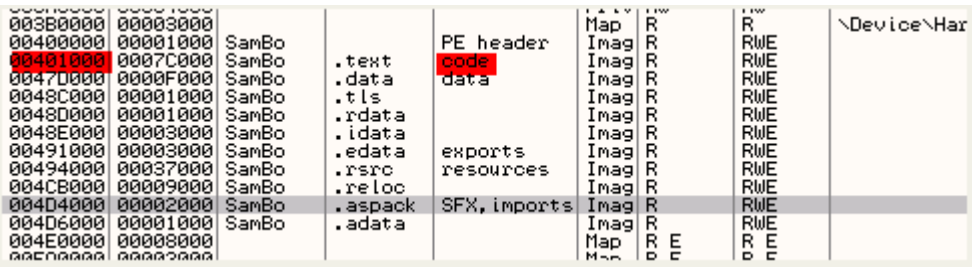


让我们来看看该程序的区段的情况,我们可以选择菜单项中的 View-Memory 或者单击工具栏中 M 按钮。



我们可以看到该程序入口点所属区段起始地址为 4D4000,大小为 2000(十六进制),入口点为 4D4001。

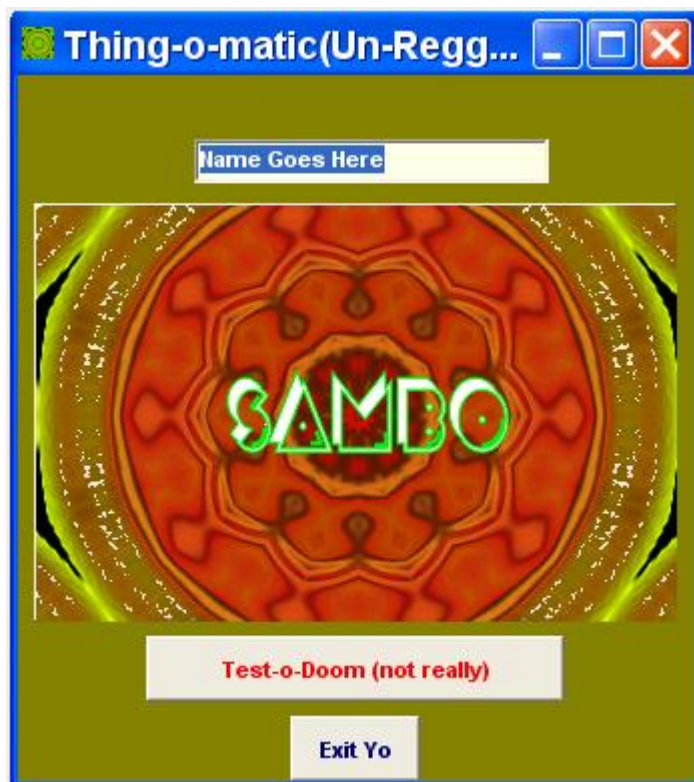
这也是为什么 OD 提示入口点在.text 节之外的缘故。



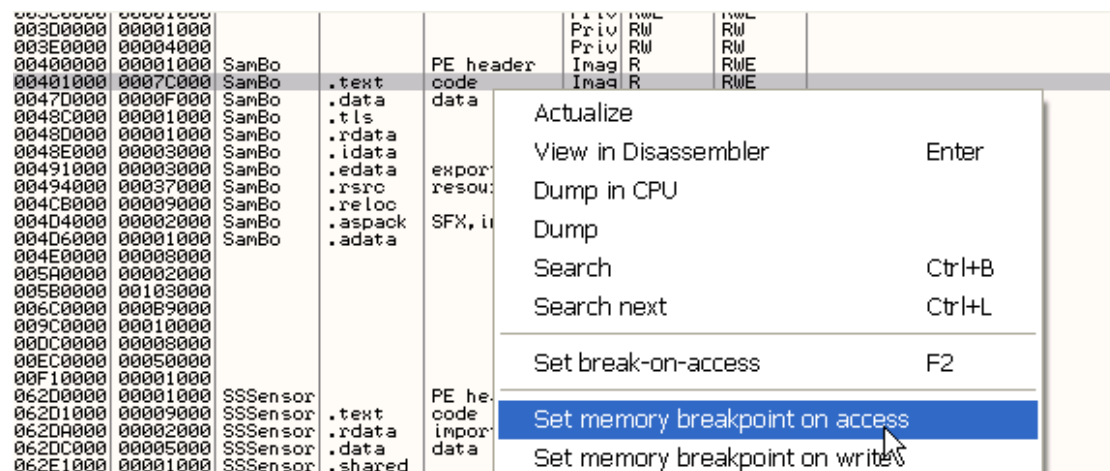
.text 区段开始于 401000,OD 提示包含 code,我们当前的入口点 4D4001 属于另外一个区段,OD 提示该程序可能包含大量嵌入数据。

从当前入口点开始解密其他区段,然后会跳转到真正的入口点处,开始执行原程序。

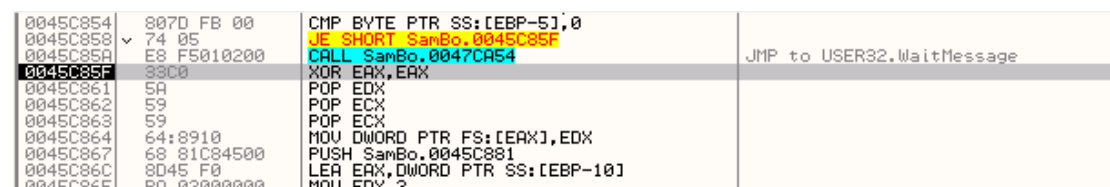
我们现在按 F9 键运行起来。



此时弹出了一个 CrackMe 窗口,等待用户输入序列号,我们知道该程序已经在内存中解密区段结束了,现在在执行代码段中的代码,我们在.text 区段上设置一个内存访问断点,让程序在执行代码段中的代码的中断下来。



当我们打开 CrackMe 窗口的时候,这个时候 OD 断在了代码段中。



我们可以看到程序正在解密内存中各种区段,我们来分析一下代码。单击鼠标右键选择-Analysis-Analyse code。



```

0045C82B . 8B45 FC      MOV EAX,DWORD PTR SS:[EBP-4]
0045C82E . E8 29F9FFFF  CALL SamBo.0045C15C
0045C833 . E8 E46FFDFF  CALL SamBo.0043381C
0045C838 . E8 A1FA0100  CALL SamBo.0047C2DE
0045C83D . 8B15 906C480 MOV EDX,DWORD PTR DS:[486C90]
0045C843 . 3B02         CMP EAX,DWORD PTR DS:[EDX]
0045C845 . 75 0D        JNZ SHORT SamBo.0045C854
0045C847 . E8 9827FCFF  CALL SamBo.0041EFE4
0045C84C . 84C0         TEST AL,AL
0045C84E . 74 04        JE SHORT SamBo.0045C854
0045C850 . C645 FB 00  MOV BYTE PTR SS:[EBP-5],0
0045C854 . 807D FB 00  CMP BYTE PTR SS:[EBP-5],0
0045C858 . 74 05        JE SHORT SamBo.0045C85F
0045C85A . E8 F5010200  CALL SamBo.0047CA54
0045C85F . 33C0        XOR EAX,EAX
0045C861 . 5A          POP EDX
0045C862 . 59          POP ECX
0045C863 . 59          POP ECX
0045C864 . 64:8910     MOV DWORD PTR FS:[EAX],EDX
0045C867 . 68 81C84500 PUSH SamBo.0045C881
0045C86C . 8D45 F0     LEA EAX,DWORD PTR SS:[EBP-10]
0045C86F . BA 02000000 MOV EDX,2
0045C874 . E8 EB72FDFF  CALL SamBo.00433B64
0045C879 . C3          RETN
0045C87A . E9 E96EFDFF JMP SamBo.00433768
0045C87F . EB EB       JMP SHORT SamBo.0045C86C
0045C881 . 5F          POP EDI
0045C882 . 5E          POP ESI
0045C883 . 5B          POP EBX
0045C884 . 8BE5       MOV ESP,EBP
0045C886 . 5D          POP EBP
0045C887 . C3          RETN
0045C888 . 53          PUSH EBX
0045C889 . 56          PUSH ESI
0045C88A . 57          PUSH EDI
0045C88B . 8BFA       MOV EDI,EDX
0045C88D . A1 047C4800 MOV EAX,DWORD PTR DS:[487C04]
0045C892 . E8 C903FFFF  CALL SamBo.00459C60

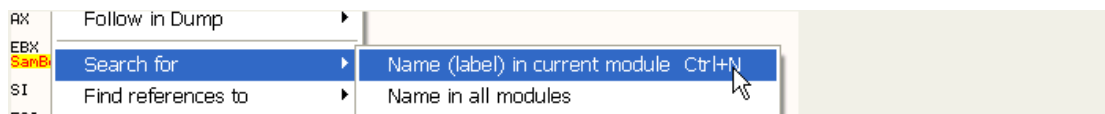
```

GetCurrentThreadId  
SamBo.00487458

WaitMessage

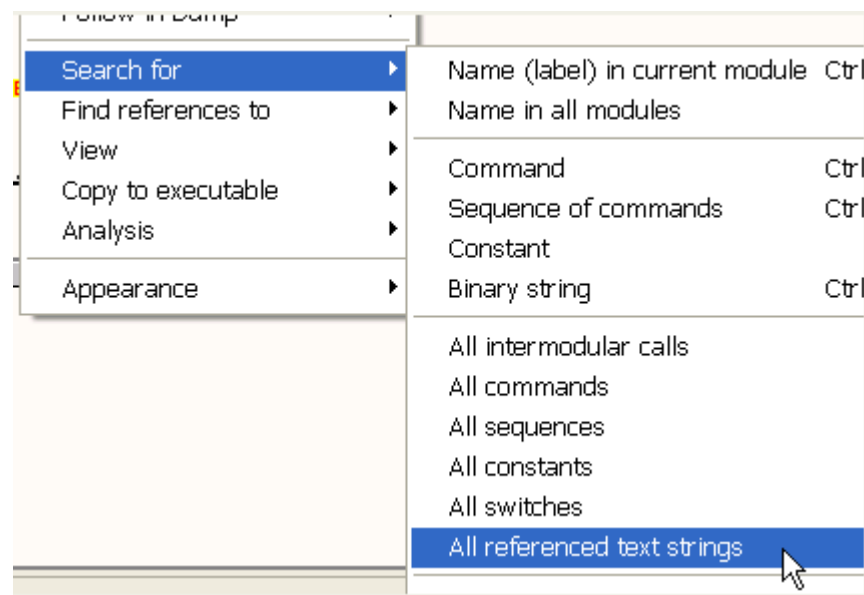
可以看到现在 OD 展示给我们的是完美分析后的代码。

现在我们位于代码段,我们查看当前模块使用了哪些 API 函数-但是我们在程序刚刚加载的时候不能这么做,因为那个时候我们查看当前模块使用的 API 的话是查看的壳所在模块使用的 API 函数,现在就不一样了。



Address	Section	Type	Name	Comment
004050BA	.aspack	Import	oleaut32.#35	
0040506C	.text	Export	@\$xp\$26Shdocvw_t lb@TcppWebBrowser	
00419960	.text	Export	@\$xp\$26Shdocvw_t lb@TcppWebBrowser	
00402274	.text	Export	@\$xp\$28Shdocvw_t lb@TcppShell@Windows	
0040FC68	.text	Export	@\$xp\$28Shdocvw_t lb@TcppShell@Windows	
00402198	.text	Export	@\$xp\$29Shdocvw_t lb@TcppShell@UIHelper	
0040FBCC	.text	Export	@\$xp\$29Shdocvw_t lb@TcppShell@UIHelper	
0040380C	.text	Export	@\$xp\$29Shdocvw_t lb@TcppWebBrowser_V1	
004113BC	.text	Export	@\$xp\$29Shdocvw_t lb@TcppWebBrowser_V1	
00401EE0	.text	Export	@\$xp\$32Shdocvw_t lb@TcppScriptErrorList	
0040F908	.text	Export	@\$xp\$32Shdocvw_t lb@TcppScriptErrorList	
0040A14	.text	Export	@\$xp\$32Shdocvw_t lb@TcppInternetExplorer	
00410560	.text	Export	@\$xp\$32Shdocvw_t lb@TcppInternetExplorer	
00401D98	.text	Export	@\$xp\$33Shdocvw_t lb@TcppSearchAssistantOC	
0040F8E8	.text	Export	@\$xp\$33Shdocvw_t lb@TcppSearchAssistantOC	
004023BC	.text	Export	@\$xp\$34Shdocvw_t lb@TcppShell@BrowserWindow	
0040F05C	.text	Export	@\$xp\$34Shdocvw_t lb@TcppShell@BrowserWindow	
00401FD4	.text	Export	@\$xp\$36Shdocvw_t lb@TShell@FavoritesNameSpace	
0040FA78	.text	Export	@\$xp\$36Shdocvw_t lb@TShell@FavoritesNameSpace	
00411B94	.text	Export	@\$Shdocvw_ock@FInitialize	
00411B7C	.text	Export	@\$Shdocvw_ock@Initialize	
00406738	.text	Export	@\$Unit10@FInitialize	
00406738	.text	Export	@\$Unit10@Initialize	
004059AA	.aspack	Import	user32.ActivateKeyboardLayout	
004059A2	.aspack	Import	gdi32.BitBlt	
004050B2	.aspack	Import	ole32.CoCreateInstance	
0047D098	.data	Export	_CPPDebugHook	
00486E9C	.data	Export	_Form1	
004013C5	.text	Export	_GetExceptDllInfo	
00404F60	.aspack	Import	kernel32.GetModuleHandleA	
00404F5C	.aspack	Import	kernel32.GetProcAddress	
0040509A	.aspack	Import	comctl32.InageList_Add	
00404F64	.aspack	Import	kernel32.LoadLibraryA	
00404001	.aspack	Export	<ModuleEntryPoint>	
00405092	.aspack	Import	advapi32.RegCloseKey	
0040F37C	.text	Export	@\$Shdocvw_ock@Register\$qqrv	
0047D6F8	.data	Export	@\$Shdocvw_t lb@TcppScriptErrorList@	
00481E44	.data	Export	@\$Shdocvw_t lb@TcppScriptErrorList@	
00401F68	.text	Export	@\$Shdocvw_t lb@TcppScriptErrorList@bctr\$qqrp18Classes@TComponent	
0040FA08	.text	Export	@\$Shdocvw_t lb@TcppScriptErrorList@bctr\$qqrp18Classes@TComponent	
0040E8B8	.text	Export	@\$Shdocvw_t lb@TcppScriptErrorList@BeforeDestruction\$qqrv	
0040E7D4	.text	Export	@\$Shdocvw_t lb@TcppScriptErrorList@Connect\$qqrv	
0040EACC	.text	Export	@\$Shdocvw_t lb@TcppScriptErrorList@ConnectTo\$qqr91%TComInterface\$28Shd	1 argument
0040E9D0	.text	Export	@\$Shdocvw_t lb@TcppScriptErrorList@Disconnect\$qqrv	
0040E5D0	.text	Export	@\$Shdocvw_t lb@TcppScriptErrorList@GetDefaultInterface\$qv	
0040E5F4	.text	Export	@\$Shdocvw_t lb@TcppScriptErrorList@GetDunk\$qqrv	
0040EBEC	.text	Export	@\$Shdocvw_t lb@TcppScriptErrorList@InitServerData\$qqrv	
0047D93C	.data	Export	@\$Shdocvw_t lb@TcppInternetExplorer@	
00402188	.data	Export	@\$Shdocvw_t lb@TcppInternetExplorer@	
00403000	.text	Export	@\$Shdocvw_t lb@TcppInternetExplorer@bctr\$qqrp18Classes@TComponent	
004108F0	.text	Export	@\$Shdocvw_t lb@TcppInternetExplorer@bctr\$qqrp18Classes@TComponent	
00408838	.text	Export	@\$Shdocvw_t lb@TcppInternetExplorer@BeforeDestruction\$qqrv	
004084EC	.text	Export	@\$Shdocvw_t lb@TcppInternetExplorer@Connect\$qqrv	
0040884C	.text	Export	@\$Shdocvw_t lb@TcppInternetExplorer@ConnectTo\$qqr83%TComInterface\$24Shd	1 argument
00408750	.text	Export	@\$Shdocvw_t lb@TcppInternetExplorer@Disconnect\$qqrv	
00408208	.text	Export	@\$Shdocvw_t lb@TcppInternetExplorer@GetDefaultInterface\$qv	
0040822C	.text	Export	@\$Shdocvw_t lb@TcppInternetExplorer@GetDunk\$qqrv	

比较糟糕的是 API 函数列表中显示是一些比较陌生的字符串。那我们就来看一下字符串列表,尝试寻找一点蛛丝马迹。



我们又看到了一个糟糕的字符串列表。

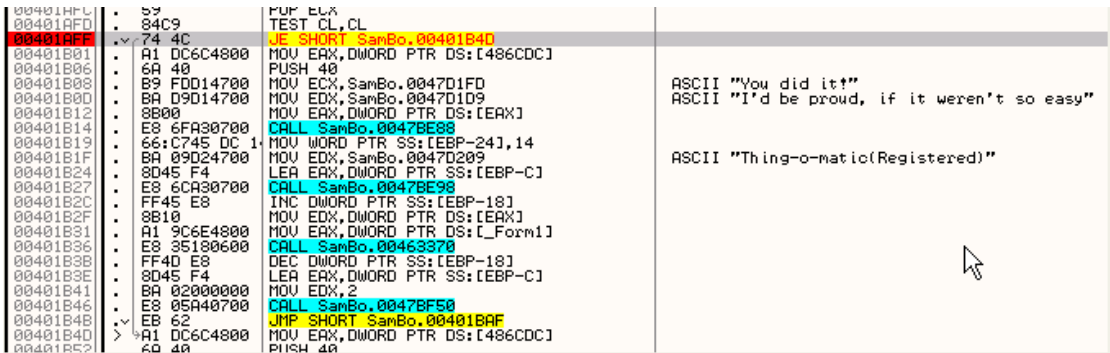
Address	Disassembly	Text string
0040101F	ASCII "oSG",0	
00401043	ASCII "00_6",0	
00401049	ASCII "0LZG",0	
00401068	ASCII "pwG",0	
0040106E	DD SamBo.00477CA4	ASCII "hLRH"
0040107A	DD SamBo.00478018	ASCII "hDRH"
0040101C	MOV EDX,SamBo.0047D0C1	ASCII "tSamBo!"
00401744	ASCII "Sysutils::Except"	
00401754	ASCII "ion",0	
004017F8	ASCII "Exception &",0	
00401834	ASCII "System::AnsiStri"	
00401844	ASCII "ng",0	
00401912	DD SamBo.00400000	ASCII "M2P"
00401934	ASCII "System::TObject",0	
0040195C	ASCII "Exception *",0	
00401A24	ASCII "TForm1 *",0	
00401A36	MOV ECX,SamBo.0047D1C5	ASCII "Thnx to Crackmes.de"
00401A48	MOV EDX,SamBo.0047D1AD	ASCII "SamBo's First Crackme"
00401B08	MOV ECX,SamBo.0047D1FD	ASCII "You did it!"
00401B0D	MOV EDX,SamBo.0047D1D9	ASCII "I'd be proud, if it weren't so easy"
00401B1F	MOV EDX,SamBo.0047D209	ASCII "Thing-o-matic(Registered)"
00401B54	MOV ECX,SamBo.0047D233	ASCII "VUP"
00401B59	MOV EDX,SamBo.0047D223	ASCII "VAV YOU GOT IT!"
00401B6C	MOV ECX,SamBo.0047D253	ASCII "JUST JOKING!"
00401B71	MOV EDX,SamBo.0047D237	ASCII "nope, actually it was wrong"
00401B83	MOV EDX,SamBo.0047D260	ASCII "Thing-o-matic(Shareware)"
00401BFC	ASCII "TForm *",0	
00401C10	ASCII "AnsiString *",0	
00401C2E	DD SamBo.00400000	ASCII "M2P"
00401C50	ASCII "Forms::TForm",0	
00401CA8	ASCII "TForm1",0	
00401CCF	ASCII "TForm1",0	

我们可以看到"You did it!"-用于提示成功的字符串之一。

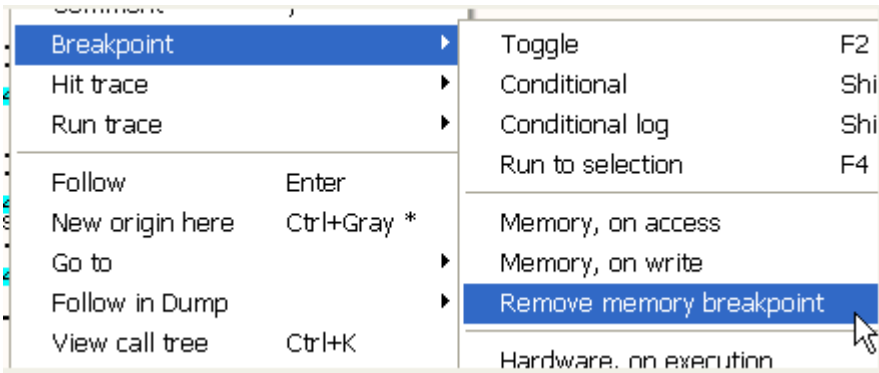
30401AF7	E8 54A40700	CALL SamBo.0047BF50	SamBo.0047BF50
30401AFC	59	POP ECX	
30401AFD	84C9	TEST CL,CL	
30401AFF	74 4C	JS SHORT SamBo.00401B4D	
30401B01	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
30401B06	6A 40	PUSH 40	
30401B08	B9 FDD14700	MOV ECX,SamBo.0047D1FD	ASCII "You did it!"
30401B0D	BA D9D14700	MOV EDX,SamBo.0047D1D9	ASCII "I'd be proud, if it weren't so easy"
30401B12	8B00	MOV EAX,DWORD PTR DS:[EAX]	
30401B14	E8 6FA30700	CALL SamBo.0047BE98	
30401B19	66 C745 DC 1	MOV WORD PTR SS:[EBP-24],14	
30401B1F	BA 09D24700	MOV EDX,SamBo.0047D209	ASCII "Thing-o-matic(Registered)"
30401B24	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
30401B27	E8 6CA30700	CALL SamBo.0047BE98	
30401B2C	FF45 E8	INC DWORD PTR SS:[EBP-18]	
30401B2F	8B10	MOV EDX,DWORD PTR DS:[EAX]	
30401B31	A1 9C6E4800	MOV EAX,DWORD PTR DS:[_Form1]	
30401B36	E8 35180600	CALL SamBo.00463370	
30401B38	FF4D E8	DEC DWORD PTR SS:[EBP-18]	
30401B3E	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
30401B41	BA 02000000	MOV EDI,2	
30401B46	E8 05A40700	CALL SamBo.0047BF50	
30401B48	EB 62	JMP SHORT SamBo.00401BAF	
30401B4D	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
30401B52	6A 40	PUSH 40	
30401B54	B9 38D24700	MOV ECX,SamBo.0047D233	ASCII "VUP"
30401B59	BA 23D24700	MOV EDX,SamBo.0047D223	ASCII "VAV YOU GOT IT!"
30401B5E	8B00	MOV EAX,DWORD PTR DS:[EAX]	
30401B60	E8 23A30700	CALL SamBo.0047BE98	
30401B65	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
30401B6A	6A 10	PUSH 10	
30401B6C	B9 53D24700	MOV ECX,SamBo.0047D253	ASCII "JUST JOKING!"
30401B71	BA 37D24700	MOV EDX,SamBo.0047D237	ASCII "nope, actually it was wrong"
30401B76	8B00	MOV EAX,DWORD PTR DS:[EAX]	
30401B78	E8 0BA30700	CALL SamBo.0047BE98	
30401B7D	66 C745 DC 2	MOV WORD PTR SS:[EBP-24],20	
30401B83	BA 60D24700	MOV EDX,SamBo.0047D260	ASCII "Thing-o-matic(Shareware)"
30401B88	8D45 F0	LEA EAX,DWORD PTR SS:[EBP-10]	



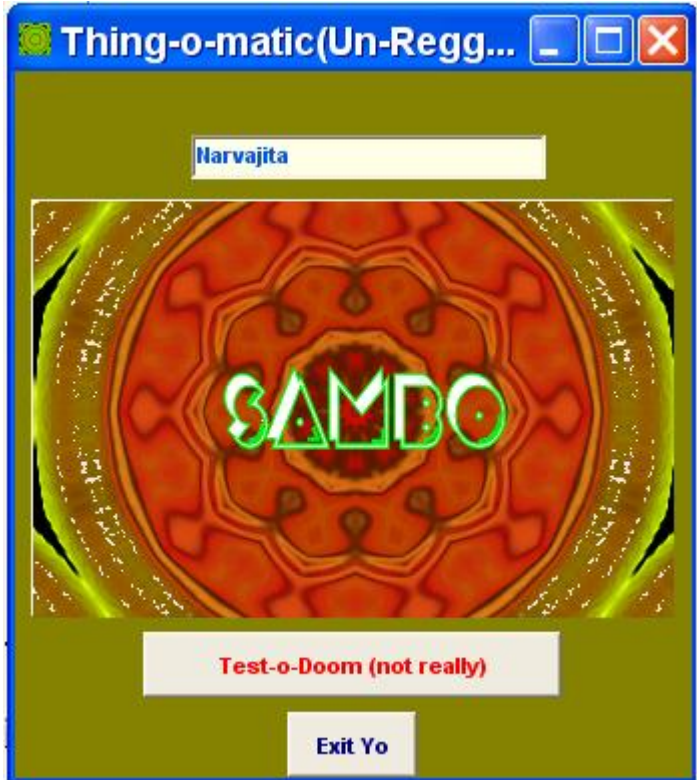
这里有一个比较指令和一个条件跳转指令跳转到提示成功的字符串代码块处,附近还有提示错误的代码块,但是并不是通过调用 MessageBoxA 函数来提示。



我们在该条件跳转指令上面设置一个断点以便来验证是不是一个关键的跳转,并且通过单击鼠标右键选择-Breakpoint-Remove memory breakpoint 来删除之前设置的内存断点。



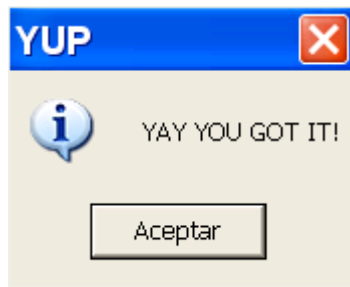
接着运行程序。



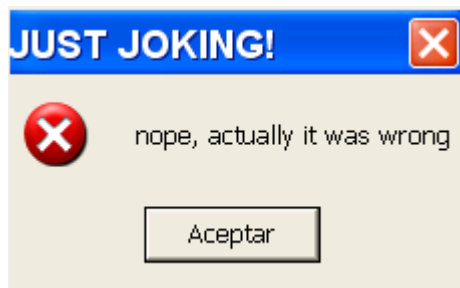
我们随便输入一个错误的序列号,这里我们输入“Narvajita”,然后按下 Test-o-Doom 按钮。

00401AFD	84C9	TEST CL,CL	
00401AFF	74 4C	JE SHORT SamBo.00401B40	
00401B01	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B06	6A 40	PUSH 40	
00401B08	B9 FDD14700	MOV ECX,SamBo.0047D1FD	ASCII "You did it!"
00401B0D	BA 09D14700	MOV EDX,SamBo.0047D1D9	ASCII "I'd be proud, if it weren't so easy"
00401B12	8B00	MOV EAX,DWORD PTR DS:[EAX]	
00401B14	E8 6FA30700	CALL SamBo.0047BE88	
00401B19	66:C745 DC 1	MOV WORD PTR SS:[EBP-24],14	
00401B1F	BA 09D24700	MOV EDX,SamBo.0047D209	ASCII "Thing-o-matic(Registered)"
00401B24	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00401B27	E8 6CA30700	CALL SamBo.0047BE98	
00401B2C	FF45 E8	INC DWORD PTR SS:[EBP-18]	
00401B2F	8B10	MOV EDX,DWORD PTR DS:[EAX]	
00401B31	A1 9C6E4800	MOV EAX,DWORD PTR DS:[_Form1]	
00401B36	E8 35180600	CALL SamBo.00463370	
00401B3B	FF4D E8	DEC DWORD PTR SS:[EBP-18]	
00401B3E	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00401B41	BA 02000000	MOV EDX,2	
00401B46	E8 05A40700	CALL SamBo.0047BF50	
00401B48	EB 62	JMP SHORT SamBo.00401BAF	
00401B4D	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B52	6A 40	PUSH 40	
00401B54	B9 33D24700	MOV ECX,SamBo.0047D233	ASCII "YUP"

看到跳转将会实现,我们直接按 F9 键。



然后弹出一个提示成功的窗口,我们单击接受按钮。



接着弹出一个消息框提示“开个玩笑,序列号错误”。我们单击接受按钮又回到了等待用户输入序列号的窗口。我们再次按下 Test-o-Doom 按钮。

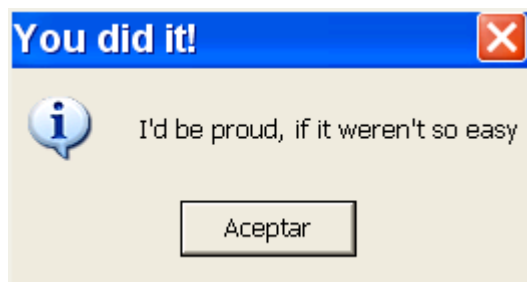
00401AF7	E8 54A40700	CALL SamBo.0047BF50	CALL SamBo.0047BF50
00401AFC	59	POP ECX	
00401AFD	84C9	TEST CL,CL	
00401B01	74 4C	JE SHORT SamBo.00401B40	
00401B06	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B08	6A 40	PUSH 40	
00401B0D	B9 FDD14700	MOV ECX,SamBo.0047D1FD	ASCII "You did it!"
00401B0E	BA 09D14700	MOV EDX,SamBo.0047D1D9	ASCII "I'd be proud, if it weren't so easy"
00401B12	8B00	MOV EAX,DWORD PTR DS:[EAX]	
00401B14	E8 6FA30700	CALL SamBo.0047BE88	
00401B19	66:C745 DC 1	MOV WORD PTR SS:[EBP-24],14	
00401B1F	BA 09D24700	MOV EDX,SamBo.0047D209	ASCII "Thing-o-matic(Registered)"
00401B24	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00401B27	E8 6CA30700	CALL SamBo.0047BE98	
00401B2C	FF45 E8	INC DWORD PTR SS:[EBP-18]	
00401B2F	8B10	MOV EDX,DWORD PTR DS:[EAX]	
00401B31	A1 9C6E4800	MOV EAX,DWORD PTR DS:[_Form1]	
00401B36	E8 35180600	CALL SamBo.00463370	
00401B3B	FF4D E8	DEC DWORD PTR SS:[EBP-18]	
00401B3E	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00401B41	BA 02000000	MOV EDX,2	
00401B46	E8 05A40700	CALL SamBo.0047BF50	
00401B48	EB 62	JMP SHORT SamBo.00401BAF	
00401B4D	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B52	6A 40	PUSH 40	
00401B54	B9 33D24700	MOV ECX,SamBo.0047D233	ASCII "YUP"
00401B59	BA 02000000	MOV EDX,2	ASCII "YAY YOU GOT IT!"

我们修改影响跳转标志位,看看会不会弹出提示序列号正确的消息框。



00401AFD	84C9	TEST CL,CL	
00401AFF	74 4C	JE SHORT SamBo.00401B4D	
00401B01	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B06	6A 40	PUSH 40	
00401B08	B9 F0D14700	MOV ECX,SamBo.0047D1F0	ASCII "You did it!"
00401B0D	BA D9D14700	MOV EDX,SamBo.0047D1D9	ASCII "I'd be proud, if it weren't so easy"
00401B12	3B00	MOV EAX,DWORD PTR DS:[EAX]	
00401B14	E8 6FA30700	CALL SamBo.0047BE88	
00401B19	66:C745 DC 1	MOV WORD PTR SS:[EBP-24],14	
00401B1F	BA 09D24700	MOV EDX,SamBo.0047D209	
00401B24	3D45 F4	LEA EDX,DWORD PTR SS:[EBP-C]	ASCII "Thing-o-matic(Registered)"
00401B27	E8 6CA30700	CALL SamBo.0047BE98	
00401B2C	FF45 E8	INC DWORD PTR SS:[EBP-18]	
00401B2F	3B10	MOV EDX,DWORD PTR DS:[EAX]	
00401B31	A1 9C6E4800	MOV EAX,DWORD PTR DS:[_Form1]	
00401B36	E8 35180600	CALL SamBo.00463370	
00401B3B	FF4D E8	DEC DWORD PTR SS:[EBP-18]	
00401B3E	3D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00401B41	BA 02000000	MOV EDX,2	
00401B46	E8 05A40700	CALL SamBo.0047BF50	
00401B4B	EB 62	JMP SHORT SamBo.00401BAF	
00401B4D	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B52	60 40	DISU 40	

我们在零标志位 Z 上面双击改变其值,然后运行起来。



弹出提示序列号正确的消息框,那么这个跳转就是决定序列号正确与否的关键跳转。我们再来看看这个关键跳转。

00401AFD	84C9	TEST CL,CL	
00401AFF	74 4C	JE SHORT SamBo.00401B4D	
00401B01	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B06	6A 40	PUSH 40	

这里的 TEST CL,CL 指令判断 CL 是否等于零,在判断之前还一个 CALL 指令,我们在这个 CALL 指令处设置一个断点。

00401AF2	BA 02000000	MOV EDX,2	
00401AF7	E8 54A40700	CALL SamBo.0047BF50	
00401AFC	59	POP ECX	
00401AFD	84C9	TEST CL,CL	
00401AFF	74 4C	JE SHORT SamBo.00401B4D	
00401B01	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	

我们把程序运行起来,然后来到主窗口再次按下 Test-o-Doom 按钮,断在了 CALL 指令处。

我们看看堆栈中的参数情况。

0012FB30	00000000	Arg1 = 00000000
0012FB34	0017C048	
0012FB38	00DC4394	
0012FB3C	00DC2578	
0012FB40	0012FD90	Pointer to next SEH record
0012FB44	00471F93	SE handler
0012FB48	0047D3AC	SamBo.0047D3AC
0012FB4C	0012FB34	
0012FB50	00000000	
0012FB54	002B0410	UNICODE "and"
0012FB58	0012FB68	
0012FB5C	00000000	
0012FB60	00DC4394	
0012FB64	00000000	
0012FB68	00000040	
0012FB6C	00000000	
0012FB70	00DC5654	ASCII "Narvajita"
0012FB74	0012FB98	
0012FB78	004648DA	RETURN to SamBo.004648DA
0012FB7C	00DC4394	
0012FB80	004707DD	RETURN to SamBo.004707DD from SamBo.00464870
0012FB84	004707C7	RETURN to SamBo.004707C7 from SamBo.004331F0
0012FB88	00DC4394	
0012FB8C	004706FA	SamBo.004706FA

堆栈中我们可以看到我们刚刚输入的错误序列号,我们通过单击鼠标右键选择-Follow in Dump 在数据窗口中定位到这个字符串。

0012FB54	002B0410	UNIC	Go to expression	Ctrl
0012FB58	0012FB68			
0012FB5C	00000000		Follow in Dump	
0012FB60	00DC4394			
0012FB64	00000000			
0012FB68	00000040		Appearance	
0012FB6C	00000000			
0012FB70	00DC5654	ASCII		
0012FB74	0012FB98			

Address	Hex dump	ASCII
00DC5654	4E 61 72 76 61 6A 69 74	Narvajit
00DC565C	61 00 6D 20 30 7A 48 00	a.m 0zH.
00DC5664	30 7A 48 00 9C 29 00 00	0zH.0zH.0zH.0zH.
00DC566C	31 35 35 36 35 35 35 00	1556555.
00DC5674	30 7A 48 00 30 7A 48 00	0zH.0zH.
00DC567C	88 29 00 00 67 2D 6F 2D	0zH.0zH.
00DC5684	6D 61 74 69 63 28 53 68	0zH.0zH.
00DC568C	61 72 65 77 61 72 65 29	0zH.0zH.
00DC5694	00 00 00 00 30 7A 48 00	0zH.0zH.
00DC569C	30 7A 48 00 64 29 00 00	0zH.0zH.

我们可以看到一个字符串“1556555”,可能是正确的序列号,我们来验证一下它到底是不是正确的序列号。

我们对这个错误序列号设置一个内存访问断点,如果这个CALL中将我们输入的错误序列号与正确的序列号进行比较的话,就会断下来。

Address	Hex dump	ASCII
00DC5654	4E 61 72 76 61 6A 69 74	Narvajit
00DC565C	61 00 6D 20 30 7A 48 00	a.m 0zH.
00DC5664	30 7A 48 00 9C 29 00 00	0zH.0zH.0zH.0zH.
00DC566C	31 35 35 36 35 35 35 00	1556555.
00DC5674	30 7A 48 00 30 7A 48 00	0zH.0zH.
00DC567C	88 29 00 00 67 2D 6F 2D	0zH.0zH.
00DC5684	6D 61 74 69 63 28 53 68	0zH.0zH.
00DC568C	61 72 65 77 61 72 65 29	0zH.0zH.
00DC5694	00 00 00 00 30 7A 48 00	0zH.0zH.
00DC569C	30 7A 48 00 64 29 00 00	0zH.0zH.

我们将光标拖选中错误的序列号,然后单击鼠标右键选择-Breakpoint-Memory,on access。

Address	Hex dump	ASCII
00DC5654	4E 61 72 76 61 6A 69 74	Narvajit
00DC565C	61 00 6D 20 30 7A 48 00	a.m 0zH.
00DC5664	30 7A 48 00 9C 29 00 00	0zH.0zH.0zH.0zH.
00DC566C	31 35 35 36 35 35 35 00	1556555.
00DC5674	30 7A 48 00 30 7A 48 00	0zH.0zH.
00DC567C	88 29 00 00 67 2D 6F 2D	0zH.0zH.
00DC5684	6D 61 74 69 63 28 53 68	0zH.0zH.
00DC568C	61 72 65 77 61 72 65 29	0zH.0zH.
00DC5694	00 00 00 00 30 7A 48 00	0zH.0zH.
00DC569C	30 7A 48 00 64 29 00 00	0zH.0zH.
00DC56A4	15 4F 41 00 00 00 00 00	0zH.0zH.
00DC56AC	00 00 00 00 00 00 00 00	0zH.0zH.

此时,如果我们运行起来,如果这个CALL中访问我们错误序列号的话,OD将中断下来。

我们按下F9键。

我们可以看到并没有中断下来,说明比较还在前面,所以我们可以考虑反复通过上面的方式在还要靠前的地方设置断点,或者我们可以以程序获取用户输入的序列号作为入手点,但是这里并没有使用GetWindowTextA函数,但是我们知道有将虚拟键消息转换为字符消息的函数。

我们删除设置的内存断点。

Comment	;
Breakpoint	Toggle F9
Hit trace	Conditional SI
Run trace	Conditional log SI
Go to	Memory, on access
Follow in Dump	Memory, on write
View call tree Ctrl+K	Remove memory breakpoint

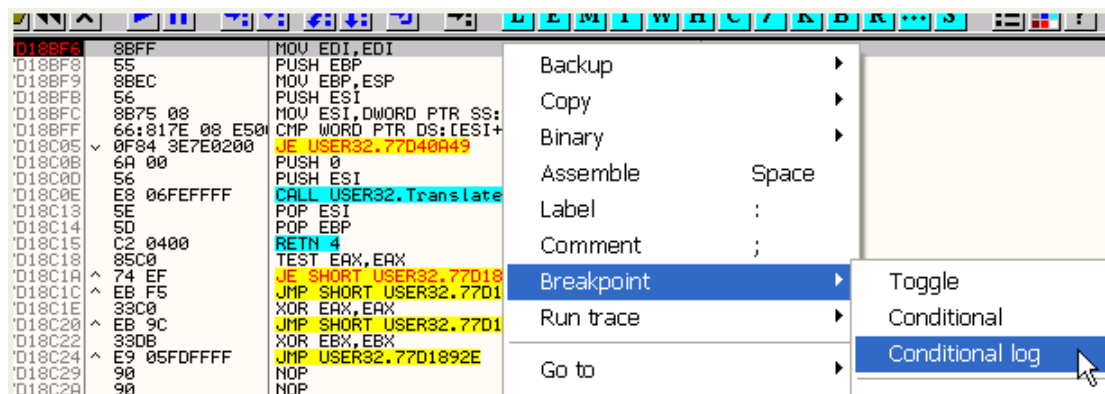
我们通过在命令栏中输入BP TranslateMessage给TranslateMessage函数设置一个断点。

Command	BP TranslateMessage
---------	---------------------

运行起来。

77D18BF6	8BFF	MOV EDI,EDI
77D18BF8	55	PUSH EBP
77D18BF9	8BEC	MOV EBP,ESP
77D18BFB	56	PUSH ESI
77D18BFC	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]
77D18BFF	66:817E 08 E50	CMP WORD PTR DS:[ESI+8],0E5
77D18C05	0F84 3E7E0200	JE USER32.77D40A49
77D18C08	6A 00	PUSH 0
77D18C0D	56	PUSH ESI
77D18C0E	E8 06FEFFFF	CALL USER32.TranslateMessageEx

断在了该 API 函数入口处,我们在断在这一行上面单击鼠标选择-Breakpoint-Conditional log



Set conditional log breakpoint at USER32.T...

Condition:  
MSG==202

Explanation:  
MSG

Decode value of expression as: Assumed by expression

	Never	On condition	Always	Pass count (dec.)
Pause program:	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	0.
Log value of expression:	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	
Log function arguments:	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	

If program pauses, pass following commands to plugins:

OK Cancel

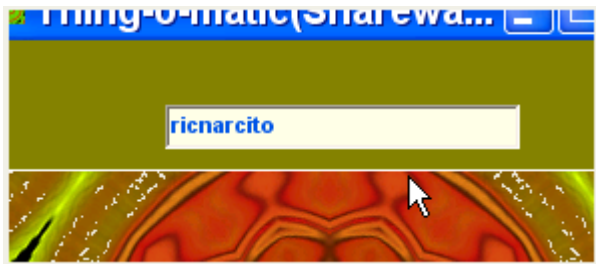
我们设置条件为 MSG == 202 即 WM\_LBUTTONDOWN。

并且设置中断程序方式为条件中断,以及总是记录表达式的值以及函数的参数值。

77D18BF6	8BFF	MOV EDI,EDI
77D18BF8	55	PUSH EBP
77D18BF9	8BEC	MOV EBP,ESP
77D18BFB	56	PUSH ESI
77D18BFC	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]
77D18BFF	66:817E 08 E50	CMP WORD PTR DS:[ESI+8],0
77D18C05	0F84 3E7E0200	JE USER32.77D40A49
77D18C08	6A 00	PUSH 0
77D18C0D	56	PUSH ESI
77D18C0E	E8 06FEFFFF	CALL USER32.TranslateMess
77D18C13	5E	POP ESI
77D18C14	5D	POP EBP

这里,可以看到一个粉红色的条件断点,我们运行起来。

我们来到主窗口,由于之前的序列号还在内存中,为了不和之前的混淆,我们输入一个不同的序列号。

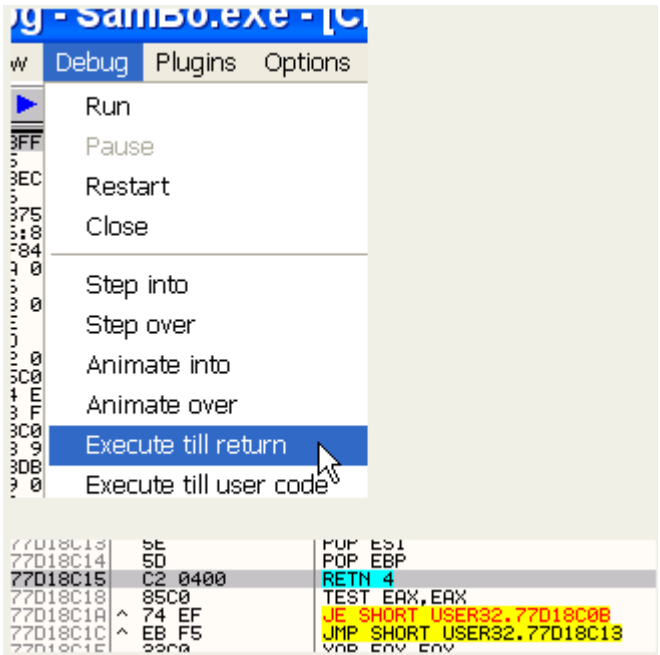


我们通过单击按钮触发条件断点。

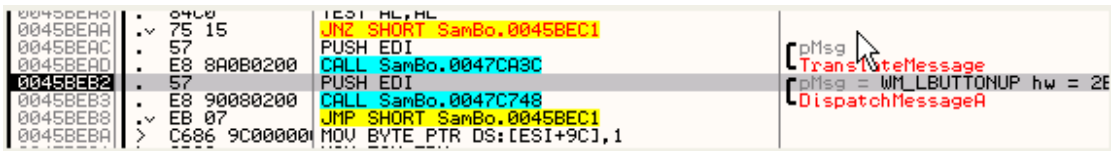


当 WM\_LBUTTONDOWN 消息到达的时候,断了下来。

我们通过选择菜单中 Debug-Execute till return 执行到返回。

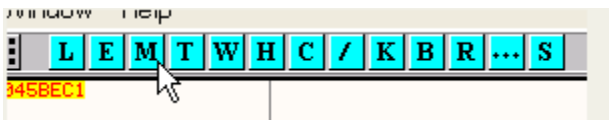


我们单击 F7 键单步返回到主程序模块中。

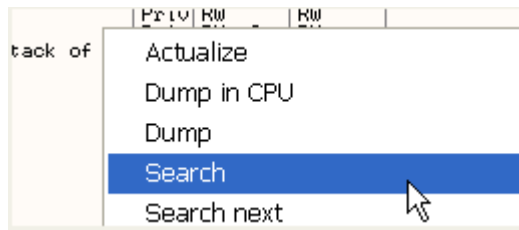


现在我们有两种选择,第一种-我们对 .text 节设置内存访问断点,然后 F9 键运行起来看看哪里在进行序列号的比较,这对于我们来说是比较困难的,我们可以看到,代码量有点大。

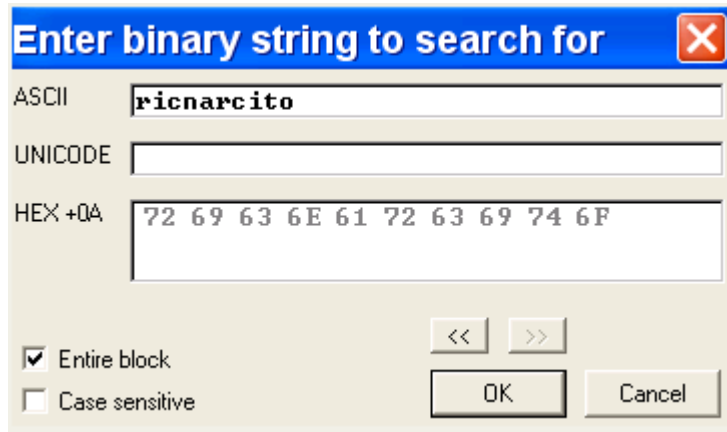
第二种选择-就是在内存搜索一下我们输入的序列号,我们单击工具栏中的 M 按钮打开内存窗口。



这里在对应内存块中搜索的功能,我们在当前内存块上面单击鼠标右键选择-Search。



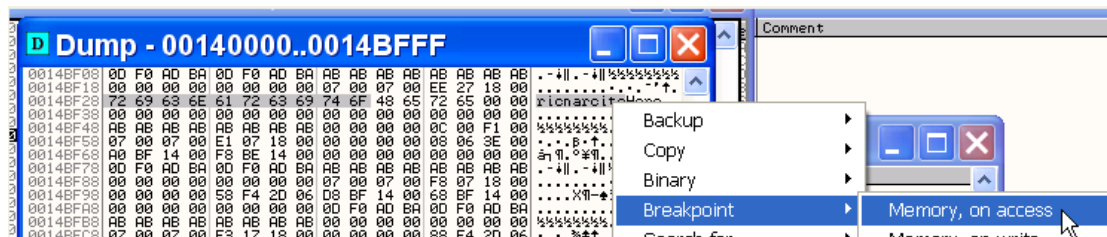
在弹出的窗口的 ASCII 编辑框中输入我们的错误序列号。



我们勾选上 Entire block(这个内存块),然后按下 OK。

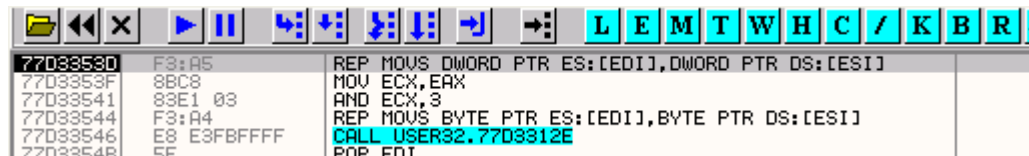


我们还可以使用 CTRL + L 来看看有没有其他内存区域有该错误序列号,我们发现当前区段中没有,其他区段中也没有找到错误序列号了。

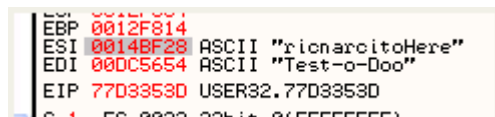


我们对错误序列号设置内存访问断点,当程序拿错误序列号进行比较的时候就会断下来。

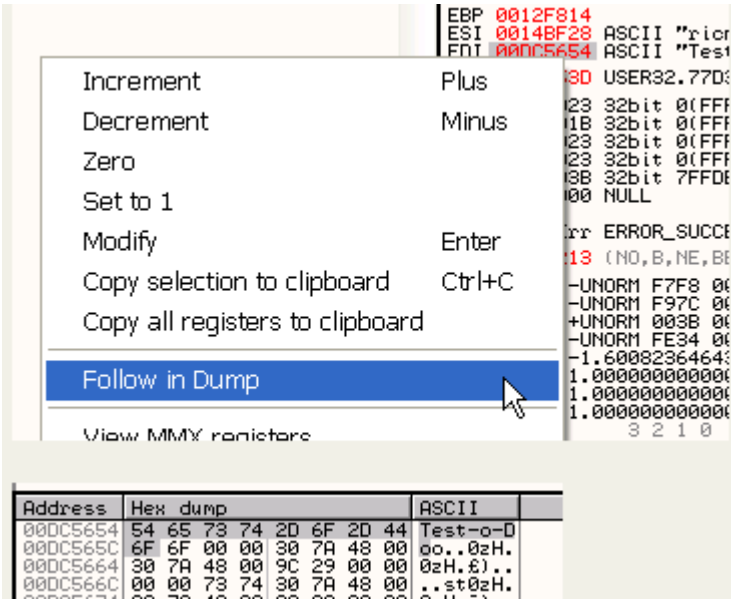
我们按下 F9 键运行起来。



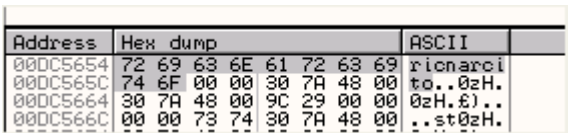
这里首先将错误序列号拷贝一块内存区域,然后紧接着有一个 CALL。



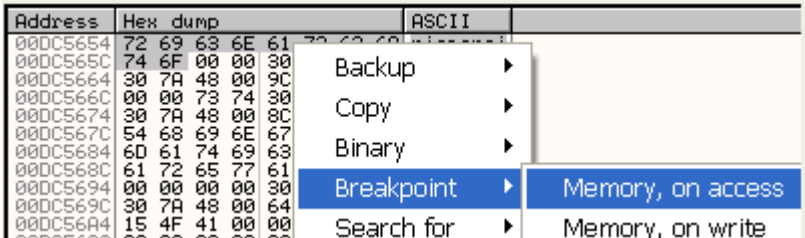
我们知道 REP MOVSB 指令会将 ESI 指向内存单元的内容拷贝到 EDI 指向的内存单元中,所以我们通过在 EDI 寄存器上面单击鼠标右键选择-Follow in Dump 在数据窗口中定位到 EDI 指向的内存单元。



这里 REP MOVSB 将进行序列号拷贝,我们按下 F8 执行该指令。



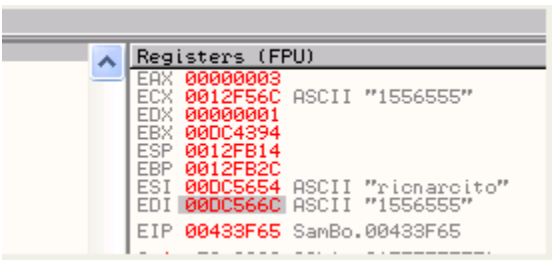
当我们一直 F8 单步执行到下面的 CALL 指令处时,错误序列号拷贝完毕,我们继续在该错误序列号上面设置内存访问断点。



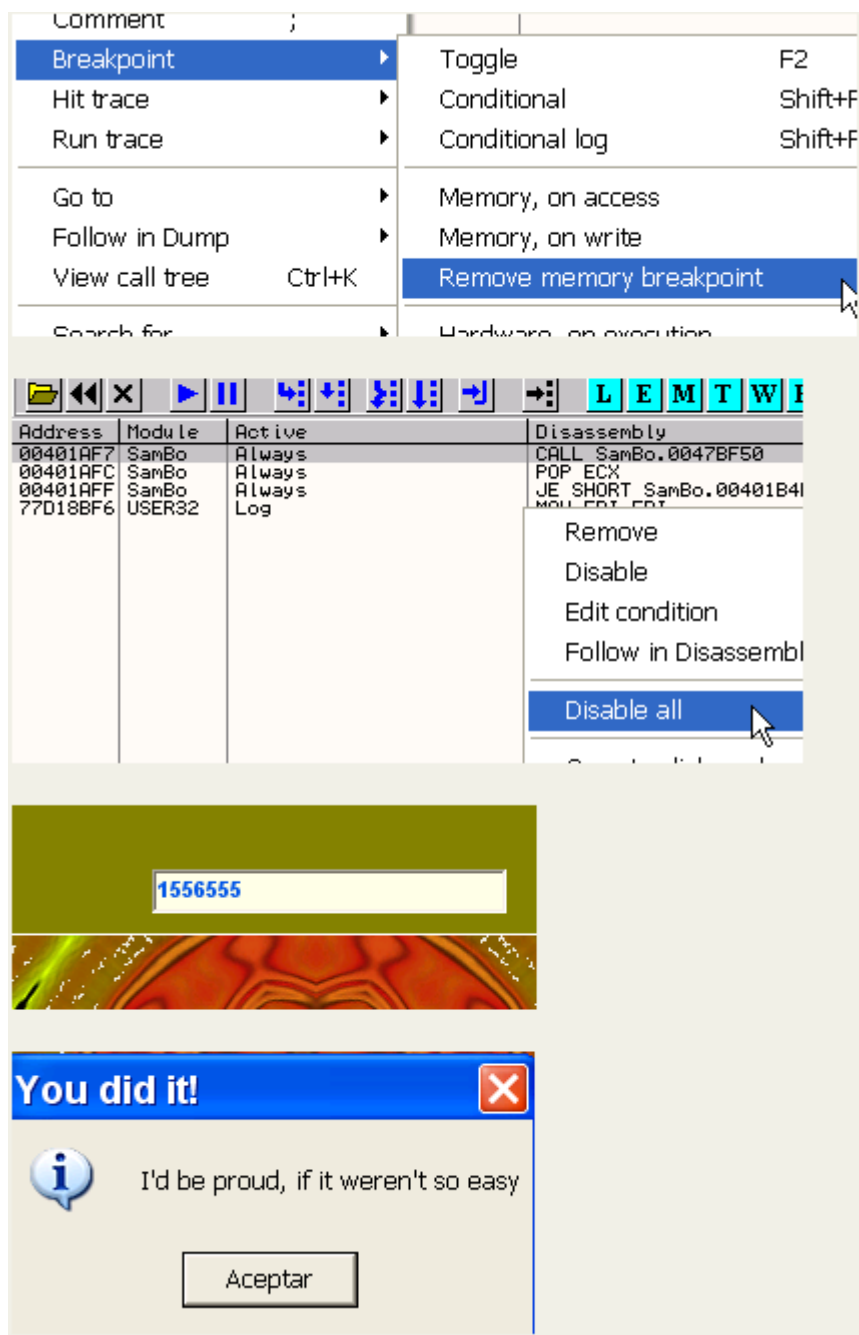
我们运行起来,断了下来。



正在进行比较,嘿嘿。



ESI 指向我们输入的错误序列号,EDI 指向正确的序列号"1556555",现在我们清楚所有断点。



弹出提示序列号正确的消息框。

下一章我们将分析用户名和序列号生成算法。



