

第十三章-硬编码序列号寻踪-Part1

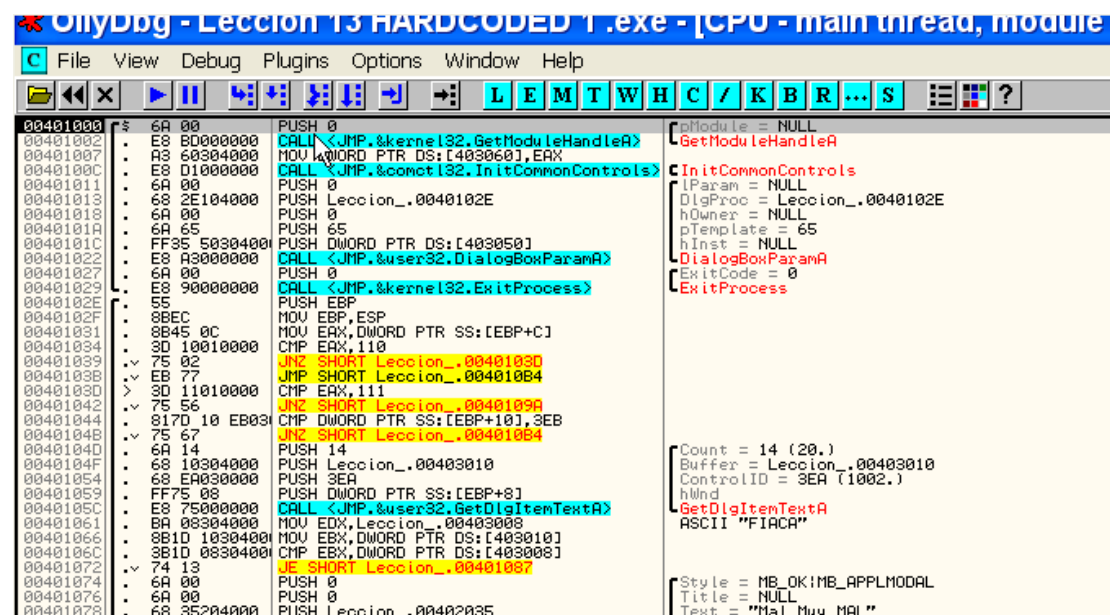
我觉得寻找序列号是最难的工作之一,特别是当我们遇到了强力的加密算法的时候,找序列号就更难了,我们先从简单的情况开始,慢慢的延伸到复杂的情况,逐步锻炼我们寻找序列号的能力。

这个部分我们专门讨论硬编码序列号,所谓的硬编码序列号就是不依赖于用户名来生成,总是由固定的字符和数字构成的固定不变的字符串,这种序列号通常比较容易找到,因此,我们从这种类型开始介绍,先看几个简单的,然后再看复杂的。

我们一

共有 4 个练习的例子,本章是两个简单的例子,下一章是两个复杂点的例子。

首先第一个最简单的 CrackMe 名为 Leccion_13_HARDCODED_1(原名称西班牙文,英文名称译为:Lesson_13_Hardcoded_1),我们用 OD 加载它。



```
00401000 6A 00 PUSH 0
00401002 E8 BD000000 CALL <JMP.<kernel32.GetModuleHandleA>
00401007 A3 60304000 MOV [DWORD PTR DS:[403060]],EAX
0040100C E8 01000000 CALL <JMP.<kernel32.InitCommonControls>
00401011 6A 00 PUSH 0
00401013 68 2E104000 PUSH Leccion_.0040102E
00401018 6A 00 PUSH 0
0040101A 6A 65 PUSH 65
0040101C FF35 50304000 PUSH DWORD PTR DS:[403050]
00401022 E8 A3000000 CALL <JMP.<user32.ShowDialogBoxParamA>
00401027 6A 00 PUSH 0
00401029 E8 90000000 CALL <JMP.<kernel32.ExitProcess>
0040102E 55 PUSH EBP
0040102F 8BEC MOV EBP,ESP
00401031 8B45 0C MOV EAX,DWORD PTR SS:[EBP+C]
00401034 3D 10010000 CMP EAX,110
00401039 75 02 JNZ SHORT Leccion_.0040103D
0040103B EB 77 JMP SHORT Leccion_.004010B4
0040103D 3D 11010000 CMP EAX,111
00401042 75 56 JNZ SHORT Leccion_.0040109A
00401044 817D 10 EB03 CMP DWORD PTR SS:[EBP+10],3EB
00401048 75 67 JNZ SHORT Leccion_.004010B4
0040104D 6A 14 PUSH 14
0040104F 68 10304000 PUSH Leccion_.00403010
00401054 68 EA030000 PUSH 3EA
00401059 FF75 08 PUSH DWORD PTR SS:[EBP+8]
0040105C E8 75000000 CALL <JMP.<user32.GetDlgItemTextA>
00401061 BA 08304000 MOV EDI,Leccion_.00403008
00401066 8B1D 10304000 MOV EBX,DWORD PTR DS:[403010]
0040106C 3B1D 08304000 CMP EBX,DWORD PTR DS:[403008]
00401072 74 13 JZ SHORT Leccion_.004010B7
00401074 6A 00 PUSH 0
00401076 6A 00 PUSH 0
00401078 68 35204000 PUSH Leccion_.00402035
```

```
[pModule = NULL
GetModuleHandleA

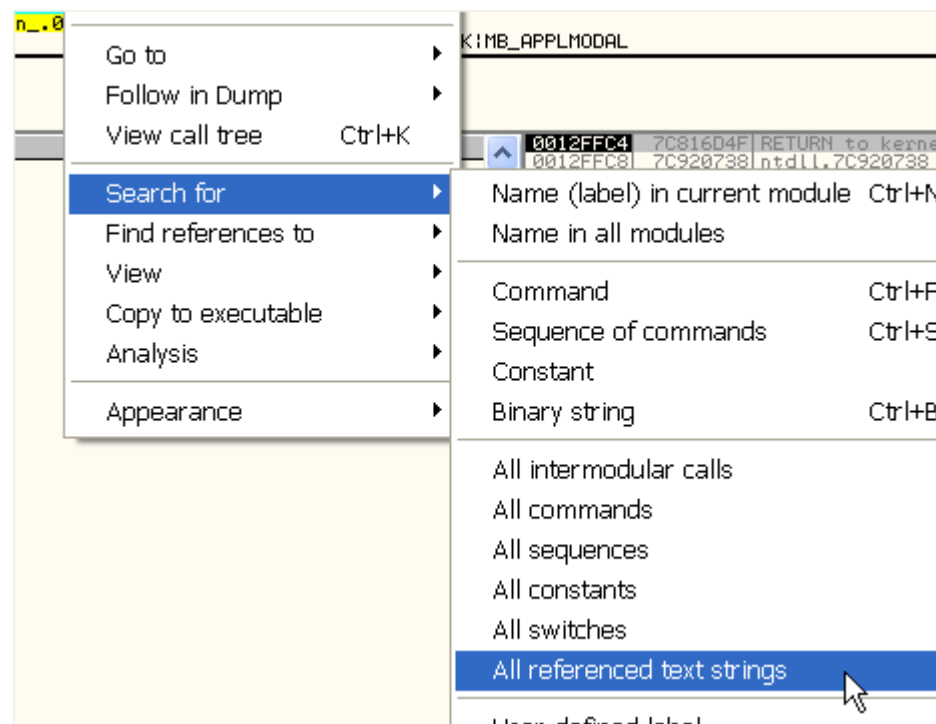
InitCommonControls
iParam = NULL
DlgProc = Leccion_.0040102E
hOwner = NULL
pTemplate = 65
hInst = NULL
DialogBoxParamA
ExitCode = 0
ExitProcess

Count = 14 (20.)
Buffer = Leccion_.00403010
ControlID = 3EA (1002.)
hWnd
GetDlgItemTextA
ASCII "FIACA"

Style = MB_OK!MB_APPLMODAL
Title = NULL
Text = "Mal Muu MAI"
```

作为最简单的硬编码序列号的例子,正确的序列号是作为全局字符串出现在程序中的,我们一起来看看。

想要找到所有的全局字符串,我们在反汇编窗口中单击鼠标右键选择-Search for-All referenced text strings。



结果如下:

Address	Disassembly	Text string
00401000	PUSH 0	(Initial CPU selection)
00401061	MOV EDX,Leccion_.00403008	ASCII "FIACA"
00401078	PUSH Leccion_.00402035	ASCII "Mal Muy MAL"

(Mal Muy MAL:西班牙文译为:错误)

这里,我们可以看到“FIACA”这个字符串,这个字符串可能就是序列号,但是我们不推荐在右侧的字符串列表中肉眼定位序列号,原因有二:

- 1.我们这个程序,只有两行,但是有的程序,有成千上万行,尝试肉眼逐一查找简直就是疯了,虽然当前这种情况,我们一眼就可以看出来正确的序列号,但是如果有千上万行的话,肉眼看就是不可取的了。
- 2.有些程序故意放置一些假的序列号在字符串列表中,当我们把这个序列号当做正确的序列号就恰恰中了作者的陷阱。较为妥当的做法是检查序列号的正确性。

首先通过在反汇编窗口中单击鼠标右键选择-Search for-Name(label) in current module 来查看 API 函数列表。

CMP DWORD PTR SS:[JNZ SHORT Leccion_.00401014
PUSH 14
PUSH Leccion_.00401014
PUSH 3EA
PUSH DWORD PTR SS:[CALL <JMP.>user32.
MOV EDX,Leccion_.0
MOV EBX,DWORD PTR
CMP EBX,DWORD PTR
JE SHORT Leccion_.
PUSH 0
PUSH 0
PUSH Leccion_.0040
PUSH DWORD PTR SS:[CALL <JMP.>user32.
JMP SHORT Leccion_.
PUSH 0

AssemblySpace
Label:
Comment;
Breakpoint
Hit trace
Run trace
Go to
Follow in Dump
View call treeCtrl+K

Search for
Find references to
View...

(20.)
Leccion_.00403010
= 3EA (1002.)
TextA
CA"
OK!MB_APPLMODAL
LL
l Muy MAL"
A
OK!MB_APPLMODAL

Name (label) in current module Ctrl+N
Name in all modules

程序中用了一些 API 函数,其中有几个我们比较熟悉。

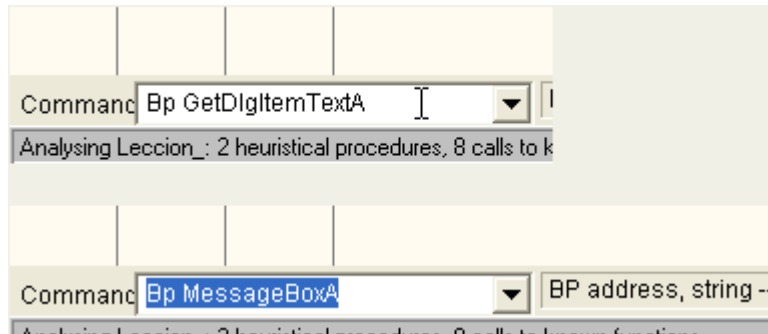
Address	Section	Type	Name	Comment
00402018	.rdata	Import	user32.DialogBoxParamA	
00402020	.rdata	Import	user32.EndDialog	
0040200C	.rdata	Import	kernel32.ExitProcess	
0040201C	.rdata	Import	user32.GetDlgItemTextA	
00402008	.rdata	Import	kernel32.GetModuleHandleA	
00402000	.rdata	Import	comctl32.InitCommonControls	
00402014	.rdata	Import	user32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	

GetDlgItemTextA 获取用户输入的序列号,MessageBoxA 提示输入的序列号正确,错误与否。我们给这两个 API 函数设置断点。

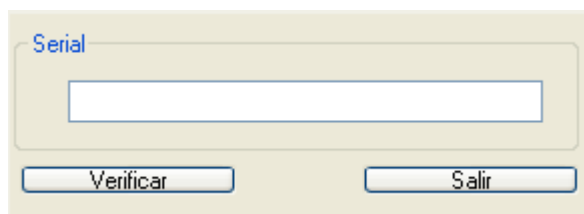
00402018	.rdata	Import	user32.DialogBoxParamA	
00402020	.rdata	Import	user32.EndDialog	
0040200C	.rdata	Import	kernel32.ExitProcess	
0040201C	.rdata	Import	user32.GetDlgItemText	
00402008	.rdata	Import	kernel32.GetModuleHandleA	
00402000	.rdata	Import	comctl32.InitCommonControls	
00402014	.rdata	Import	user32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	

Actualize
Follow import in Disassembler
Follow in Dump
Find references to import
View call tree
Toggle breakpoint on import
Conditional breakpoint on import
Conditional log breakpoint on import

单击鼠标右键选择-Toggle breakpoint on import 或者在命令栏中输入 bp GetDlgItemTextA,bp MessageBoxA。

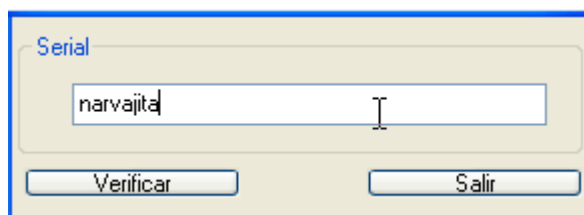


好了,按 F9 键,运行 CrackMe。



(Verficar 西班牙文译为:验证,Salir 译为:取消)

我们在序列号窗口中随便输入一个人名,比如说 narvajita。



单击 Verificar(验证)按钮,可以看到断在了我们刚刚设置的断点处。

```

77D6AC1E 8BFF      MOV EDI,EDI
77D6AC20 55        PUSH EBP
77D6AC21 8BEC      MOV EBP,ESP
77D6AC23 FF75 0C   PUSH DWORD PTR SS:[EBP+C]
77D6AC26 FF75 08   PUSH DWORD PTR SS:[EBP+8]
77D6AC29 E8 E89BFBF CALL user32.GetDlgItem
77D6AC2E 85C0      TEST EAX,EAX
77D6AC30 74 0E     JE SHORT user32.77D6AC40
77D6AC32 FF75 14   PUSH DWORD PTR SS:[EBP+14]
77D6AC35 FF75 10   PUSH DWORD PTR SS:[EBP+10]
77D6AC38 50        PUSH EAX
77D6AC39 E8 FE74FCFF CALL user32.GetWindowTextA
77D6AC3E EB 0E     JMP SHORT user32.77D6AC4E
77D6AC40 837D 14 00 CMP DWORD PTR SS:[EBP+14],0
77D6AC44 74 06     JE SHORT user32.77D6AC4C
77D6AC46 8B45 10   MOV EAX,DWORD PTR SS:[EBP+10]
77D6AC49 C600 00   MOV BYTE PTR DS:[EAX],0
77D6AC4C 33C0      XOR EAX,EAX
77D6AC4E 5D        POP EBP
77D6AC4F C2 1000   RETN 10
77D6AC52 90        NOP
77D6AC53 90        NOP
77D6AC54 90        NOP
77D6AC55 90        NOP

```

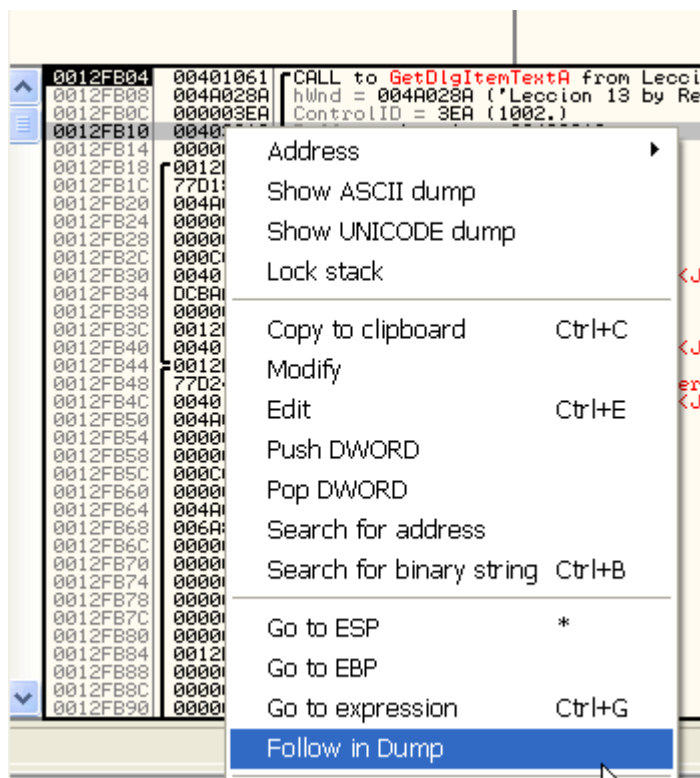
从堆栈中可以看出我们断在了 GetDlgItemTextA 处,该函数用于获取用户输入的序列号,并且该函数的 Buffer 参数是用于存放获取到的序列号的缓冲区首地址,这里,缓冲区的首地址为 403010。

```

0012FB04 00401061 CALL to GetDlgItemTextA from Leccion_.004
0012FB08 004A028A hWnd = 004A028A ('Leccion 13 by RedH@wK')
0012FB0C 000003EA ControlID = 3EA (1002.)
0012FB10 00403010 Buffer = Leccion_.00403010
0012FB14 00000014 Count = 14 (20.)
0012FB18 0012FB44
0012FB1C 77D18734 RETURN to user32.77D18734

```

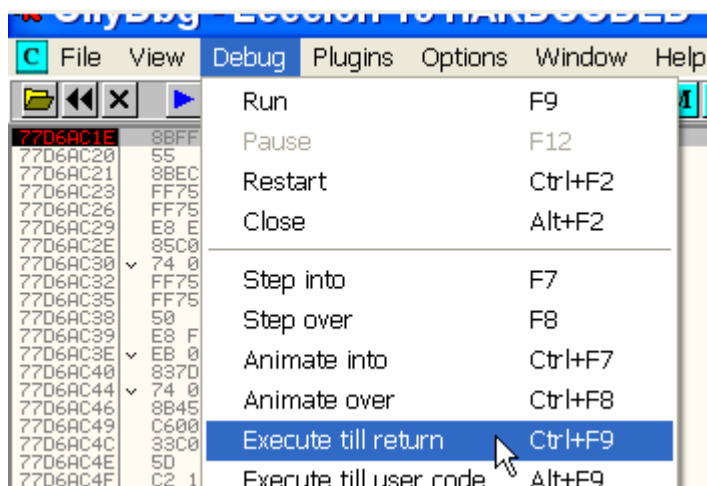
我们在该参数上单击鼠标右键选择-Follow in Dump 定位到缓冲区在数据窗口中的位置。



当前缓冲区为空,因为该 API 函数还没有执行。

Address	Hex dump	ASCII
00403010	00 00 00 00 00 00 00 00
00403018	00 00 00 00 00 00 00 00
00403020	00 00 00 00 00 00 00 00
00403028	00 00 00 00 00 00 00 00
00403030	00 00 00 00 00 00 00 00
00403038	00 00 00 00 00 00 00 00
00403040	00 00 00 00 00 00 00 00
00403048	00 00 00 00 00 00 00 00
00403050	00 00 00 00 00 00 00 00
00403058	00 00 00 00 00 00 00 00

我们选择主菜单中项 Debug-Execute till return 执行到函数返回。



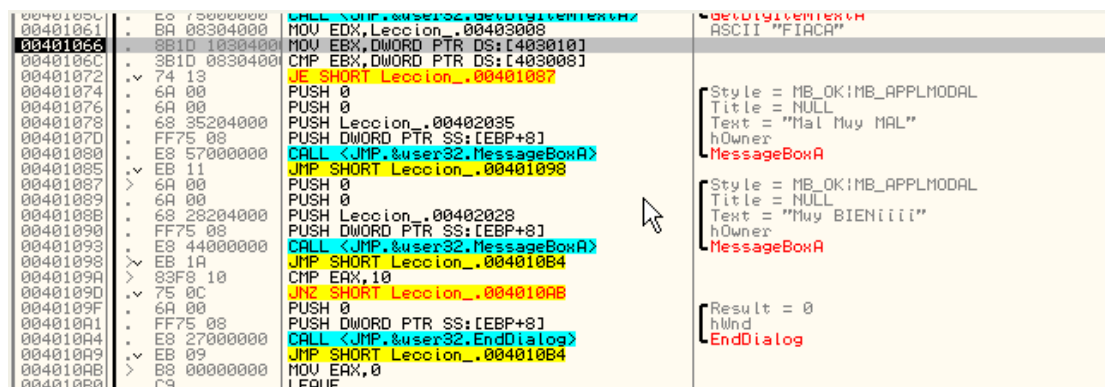
现在我们到了 RET 指令处。

Address	Hex dump	Disassembly
77D6AC1E	8BFF	MOV EDI,EDI
77D6AC20	55	PUSH EBP
77D6AC21	8BEC	MOV EBP,ESP
77D6AC23	FF75 0C	PUSH DWORD PTR SS:[EBP+C]
77D6AC26	FF75 08	PUSH DWORD PTR SS:[EBP+8]
77D6AC29	E8 E99BF8FF	CALL user32.GetDlgItem
77D6AC2E	85C0	TEST EAX,EAX
77D6AC30	74 0E	JE SHORT user32.77D6AC40
77D6AC32	FF75 14	PUSH DWORD PTR SS:[EBP+14]
77D6AC35	FF75 10	PUSH DWORD PTR SS:[EBP+10]
77D6AC38	50	PUSH EAX
77D6AC39	E8 FE74FCFF	CALL user32.GetWindowTextA
77D6AC3E	EB 0E	JMP SHORT user32.77D6AC4E
77D6AC40	837D 14 00	CMP DWORD PTR SS:[EBP+14],0
77D6AC44	74 06	JE SHORT user32.77D6AC4C
77D6AC46	8B45 10	MOV EAX,DWORD PTR SS:[EBP+10]
77D6AC49	C600 00	MOV BYTE PTR DS:[EAX],0
77D6AC4C	33C0	XOR EAX,EAX
77D6AC4E	5D	POP EBP
77D6AC4F	C2 1000	RETN 10
77D6AC52	90	NOP
77D6AC53	90	NOP
77D6AC54	90	NOP
77D6AC55	90	NOP

该函数将用户输入的序列号保存到了缓冲区中。

Address	Hex dump	ASCII
00403010	6E 61 72 76 61 6A 69 74	narvajit
00403018	61 00 00 00 00 00 00 00	a.....
00403020	00 00 00 00 00 00 00 00
00403028	00 00 00 00 00 00 00 00
00403030	00 00 00 00 00 00 00 00
00403038	00 00 00 00 00 00 00 00

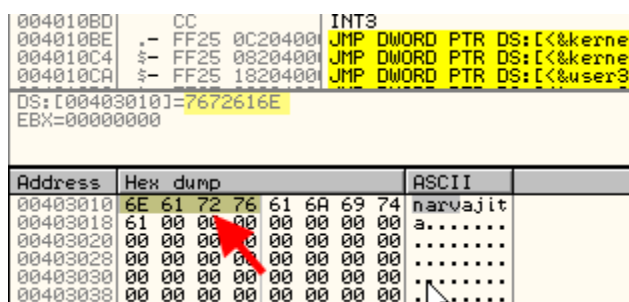
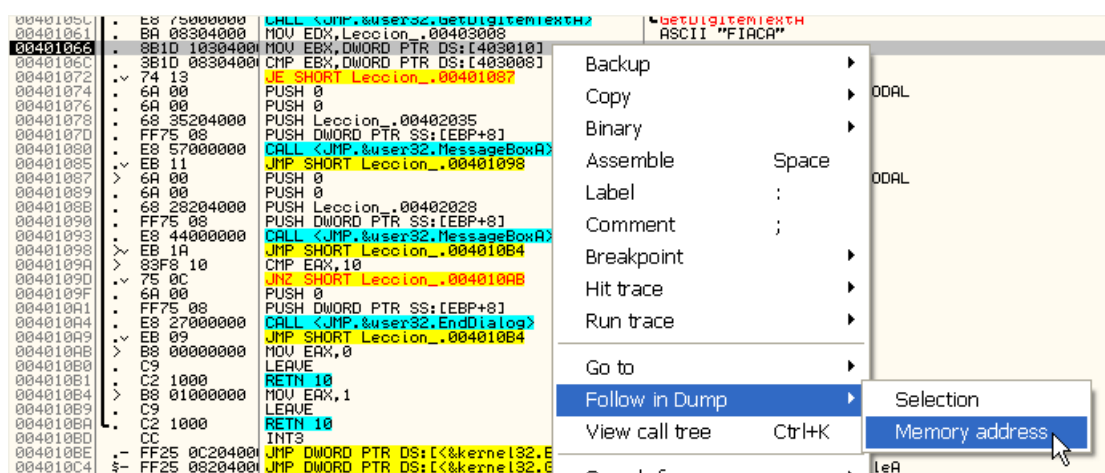
我们按 F7 键单步返回到主程序中。



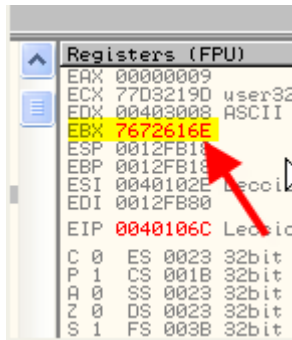
这里我们可以看到比较和条件跳转指令,两个分支分别为提示 Mal Muy MAL(错误)的消息框和提示 Muy BIEN(正确)的消息框。

很明显,401087 是其中一个分支。通常我们可以通过修改程序流程来达到爆破的目的,但是这里我们是要找到序列号,所以我们还是一起来看看程序是怎么比较的吧。

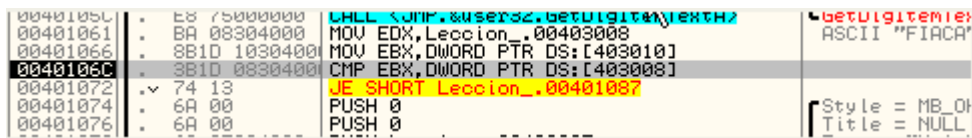
401066 地址处的指令将 403010 地址处的 DWORD 内容保存到 EBX 中,我们在该语句上单击鼠标右键选择-Follow in Dump-Memory address 定位 403010 在数据窗口中的位置。



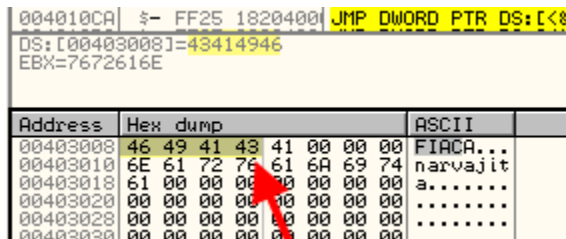
提示窗口中我们可以看到 7672616E,在 403010 开始的内存单元中是倒序存放的,(小端存储我们前面章节已经介绍过),这 4 个字节是错误的序列号,被保存到 EBX 中。



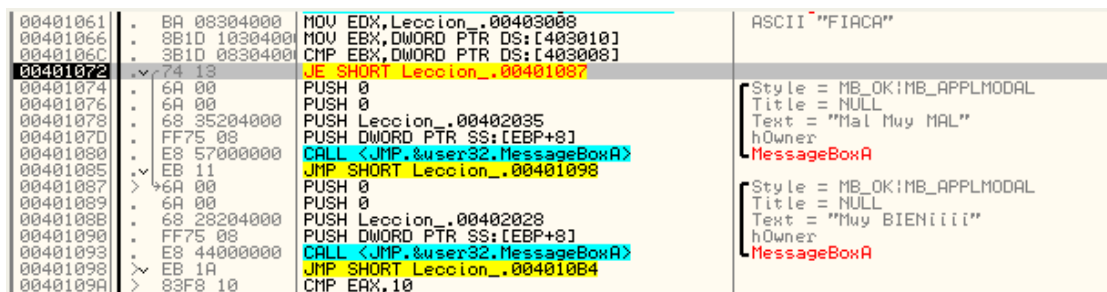
按 F7 键单步,来到下一条指令处。



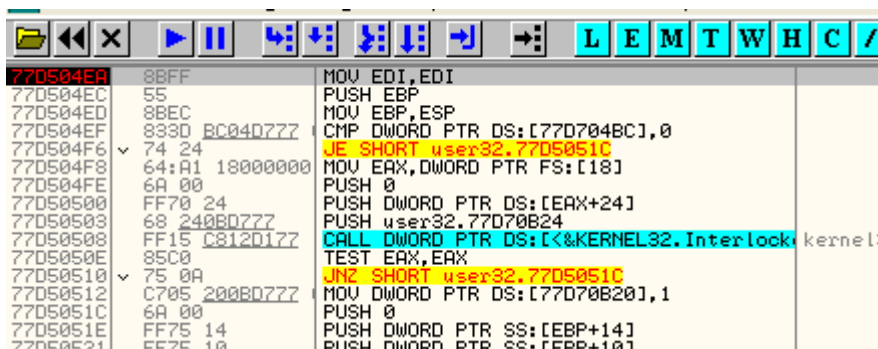
这里,我们看到该指令将 EBX(包含了我们刚刚输入的错误序列号的前 4 个字节)的值与 403008 内存单元中的内容进行比较。跟之前操作一样我们在数据窗口中转到 403008 处。



我们可以看到该指令将“FIACA”的前 4 个字节与我们输入的序列号的前 4 个字节进行比较,如果它们是相等的,则零标志位 Z 被置 1,然后跳转到提示“Muy BIEN”(正确的)的消息框处,如果不相等的话,就提示“Mal Muy MAL”(错误的)消息框。



很明显它们不相等,所以会直接提示“Mal Muy MAL”(错误的)消息框。我们按 F9 键运行。



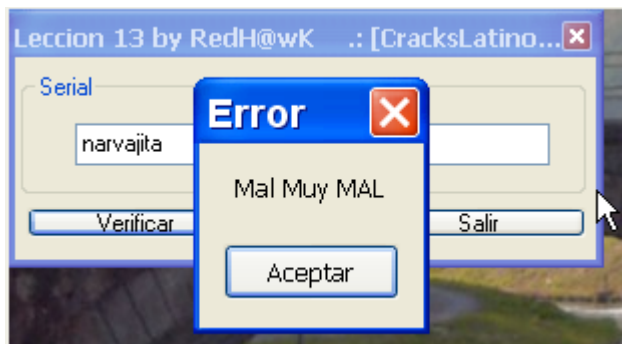
我们断在了 MessageBoxA 处。


```

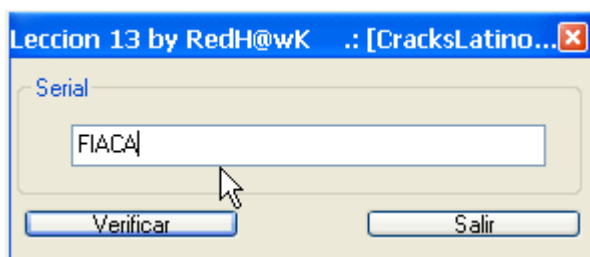
0012FB04 00401085 CALL to MessageBoxA from Leccion_.00401080
0012FB08 004A028A hOwner = 004A028A ('Leccion 13 by RedH@wK  ..: [...',class='#32770')
0012FB0C 00402035 Text = "Mal Muy MAL"
0012FB10 00000000 Title = NULL
0012FB14 00000000 Style = MB_OK!MB_APPLMODAL
0012FB18 0012FB44
0012FB1C 77D18734 RETURN to user32.77D18734
0012FB20 004A028A

```

从参数我们可以看出是提示“Mal Muy MAL”(错误)。我们按 F9 键运行起来。



正如你所看到的,弹出了错误消息提示框,我们单击“Aceptar”(确定)按钮,然后输入正确的序列号“FIACA”。



单击“Verificar”(验证)按钮,并重复上面的步骤,再次到达比较指令为止。

```

00401061 . BA 08304000 MOV EDX,Leccion_.00403008
00401066 . 8B1D 10304000 MOV EBX,DWORD PTR DS:[403010]
0040106C . 3B1D 08304000 CMP EBX,DWORD PTR DS:[403008]
00401072 . 74 13 JE SHORT Leccion_.00401087
00401074 . 6A 00 PUSH 0
00401076 . 6A 00 PUSH 0

```

跟之前一样,EBX 的值与 403008 内存单元的内容进行比较。

DS:[00403008]=43414946
EBX=43414946

Address	Hex dump	ASCII
00403008	46 49 41 43 41 00 00 00	FIACA...
00403010	46 49 41 43 41 00 69 74	FIACA.it
00403018	61 00 00 00 00 00 00 00	a.....
00403020	00 00 00 00 00 00 00 00

根据提示窗口来看,它们是相等的,零标志位置 1,我们按 F7 键单步执行。

```

EIP 00401072 Leccion_.004010
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (
EFL 00000246 (NO,NB,E,BE,NS,
ST0 empty -8.154345916481978
ST1 empty +INOPM 5760.000000

```

可以看到零标志位置 1,所以 JE 指令将跳转。

00401061	. BA 08304000	MOV EDI,Leccion_.00403008	ASCII "FIACA"
00401066	. 8B1D 10304000	MOV EBX,DWORD PTR DS:[4030101]	
0040106C	. 3B1D 08304000	CMP EBX,DWORD PTR DS:[4030081]	
00401072	74 13	JE SHORT Leccion_.00401087	
00401074	6A 00	PUSH 0	Style = MB_OK!MB_APPLMODAL
00401076	6A 00	PUSH 0	Title = NULL
00401078	68 35204000	PUSH Leccion_.00402035	Text = "Mal Muy MAL"
0040107D	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401080	E8 57000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401085	EB 11	JMP SHORT Leccion_.00401098	
00401087	6A 00	PUSH 0	Style = MB_OK!MB_APPLMODAL
00401089	6A 00	PUSH 0	Title = NULL
0040108B	68 28204000	PUSH Leccion_.00402028	Text = "Muy BIENiiii"
00401090	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401093	E8 44000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401098	EB 1A	JMP SHORT Leccion_.004010B4	
0040109A	83F8 10	CMP EAX,10	
0040109D	75 0C	JNZ SHORT Leccion_.004010AB	

跳转到了“Muy BIEN”(正确)的消息框处。我们按 F9 键运行。

0012FB04	00401098	CALL to MessageBoxA from Leccion_.00401093
0012FB08	004A028A	hOwner = 004A028A ('Leccion 13 by RedH@wK
0012FB0C	00402028	Text = "Muy BIENiiii" ;: [...',class='#32770')
0012FB10	00000000	Title = NULL
0012FB14	00000000	Style = MB_OK!MB_APPLMODAL
0012FB18	0012FB44	
0012FB1C	77D18734	RETURN to user32.77D18734
0012FB20	004A028A	
0012FB24	00000111	
0012FB28	00000000	

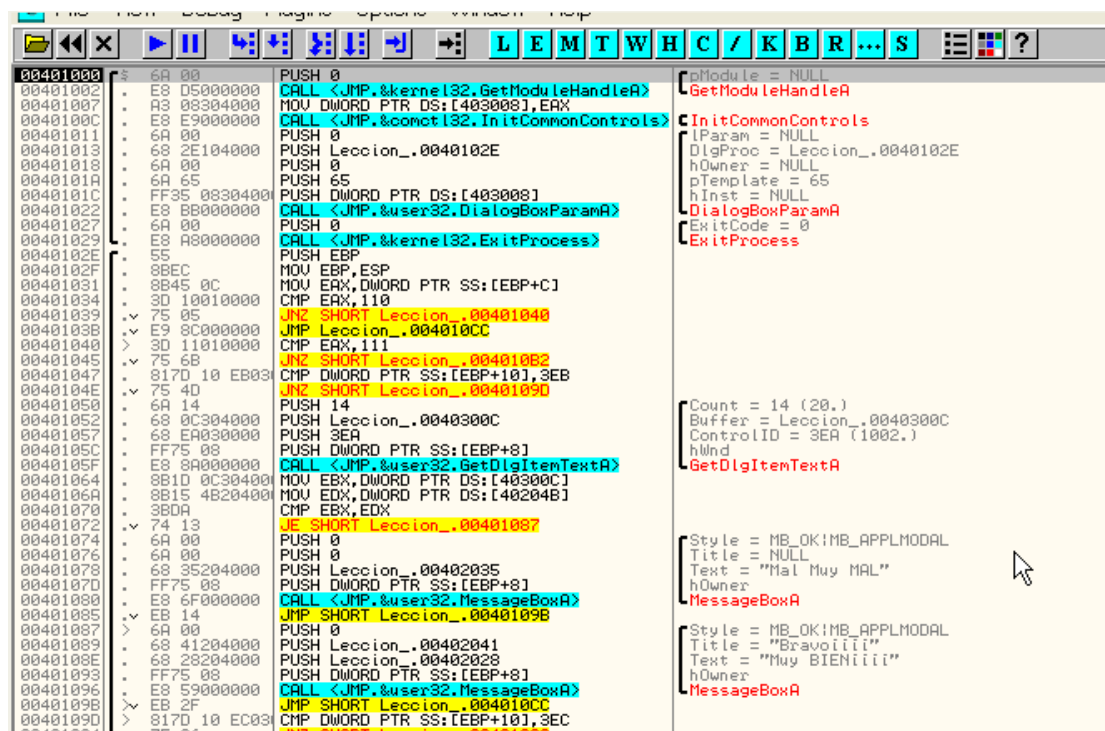
根据堆栈窗口中该 API 函数的参数可以看出将弹出提示正确的消息框。



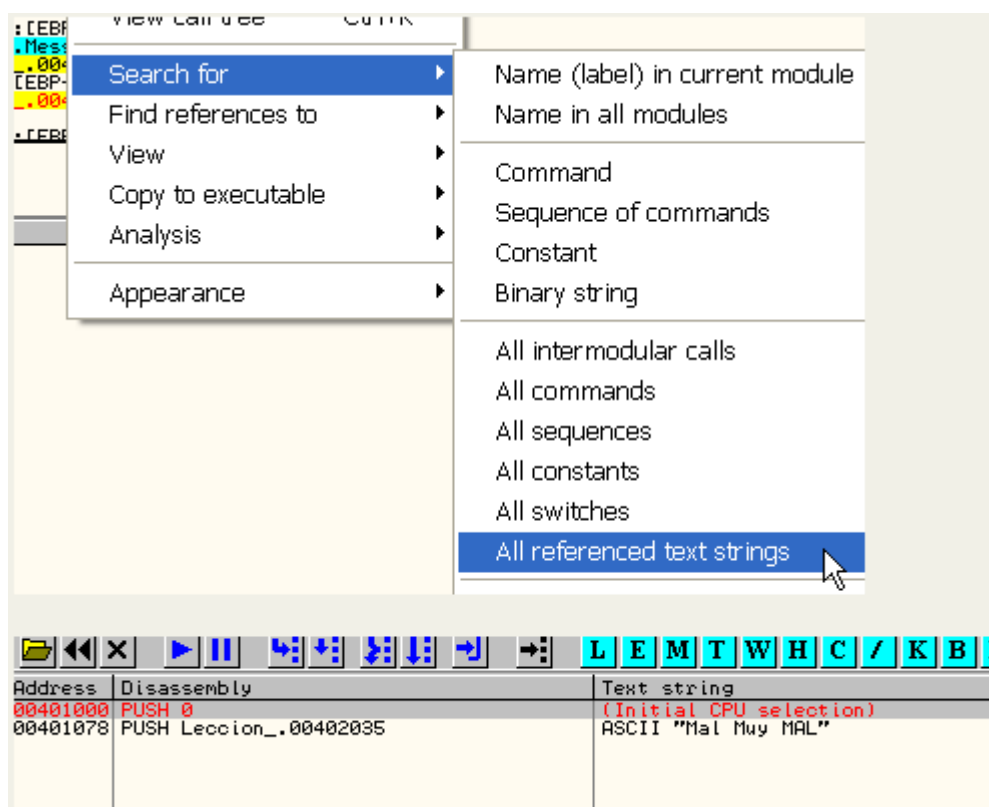
虽然作者忘了修改窗口的标题,但是我们还是找到了正确的序列号,这个硬编码序列号的例子很简单。

这个例子是我朋友 Redhawk 写的,哈哈,他有点懒,连正确提示框的标题都没有改。接下来一个 CrackMe 的提示框的标题做了修改。

我们用 OD 打开“Leccion 13 HARDCODED 2”。



这个例子和刚才那个例子非常的相似,但是正确的序列号并不在字符串列表窗口中。



并没有“FIACA”,嘿嘿。

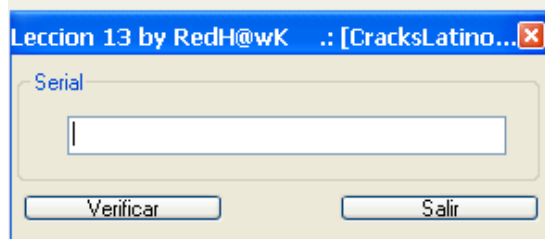
是的,正确的序列号并不是“FIACA”

我们只能来到比较指令处。

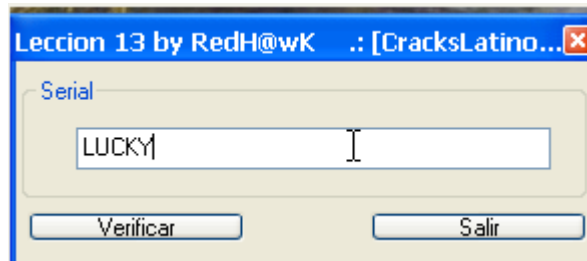
0040105F	E8 8A000000	CALL <JMP.&user32.GetDlgItemTextA>	hWnd
00401064	8B1D 0C304000	MOV EBX,DWORD PTR DS:[40300C]	GetDlgItemTextA
0040106A	8B15 4B204000	MOV EDX,DWORD PTR DS:[40204B]	
00401070	3BD8	CMP EBX,EDX	
00401072	74 13	JE SHORT Leccion_.00401087	
00401074	6A 00	PUSH 0	Style = MB_OK;MB_APPLMODAL
00401076	6A 00	PUSH 0	Title = NULL
00401078	68 35204000	PUSH Leccion_.00402035	Text = "Mal Muy MAL"
0040107D	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401080	E8 6F000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401085	EB 14	JMP SHORT Leccion_.0040109B	
00401087	6A 00	PUSH 0	Style = MB_OK;MB_APPLMODAL
00401089	68 41204000	PUSH Leccion_.00402041	Title = "Bravo!!!!"
0040108E	68 28204000	PUSH Leccion_.00402028	Text = "Muy BIEN!!!!"
00401093	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401096	E8 59000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
0040109B	EB 2F	JMP SHORT Leccion_.004010CC	
0040109D	817D 10 EC000000	CMP DWORD PTR SS:[EBP+10],3EC	

我们在 401064 处下个断点,然后运行起来。

00401057	68 EA030000	PUSH 3EA	ControlID = 3EA (1002.)
0040105C	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd
0040105F	E8 8A000000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401064	8B1D 0C304000	MOV EBX,DWORD PTR DS:[40300C]	
0040106A	8B15 4B204000	MOV EDX,DWORD PTR DS:[40204B]	
00401070	3BD8	CMP EBX,EDX	
00401072	74 13	JE SHORT Leccion_.00401087	
00401074	6A 00	PUSH 0	Style = MB_OK;MB_APPLMODAL
00401076	6A 00	PUSH 0	Title = NULL
00401078	68 35204000	PUSH Leccion_.00402035	Text = "Mal Muy MAL"
0040107D	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401080	E8 6F000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401085	EB 14	JMP SHORT Leccion_.0040109B	
00401087	6A 00	PUSH 0	Style = MB_OK;MB_APPLMODAL
00401089	68 41204000	PUSH Leccion_.00402041	Title = "Bravo!!!!"
0040108E	68 28204000	PUSH Leccion_.00402028	Text = "Muy BIEN!!!!"
00401093	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401096	E8 59000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
0040109B	EB 2F	JMP SHORT Leccion_.004010CC	
0040109D	817D 10 EC000000	CMP DWORD PTR SS:[EBP+10],3EC	
004010A1	75 26	JNZ SHORT Leccion_.004010B0	



我们随便输入一个错误的序列号,这里我们输入 LUCKY。



单击“Verificar”(验证)。

0040105C	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd
0040105F	E8 8A000000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401064	8B1D 0C304000	MOV EBX,DWORD PTR DS:[40300C]	
0040106A	8B15 4B204000	MOV EDX,DWORD PTR DS:[40204B]	
00401070	3BD8	CMP EBX,EDX	
00401072	74 13	JE SHORT Leccion_.00401087	
00401074	6A 00	PUSH 0	Style = MB_OK;MB_APPLMODAL
00401076	6A 00	PUSH 0	Title = NULL
00401078	68 35204000	PUSH Leccion_.00402035	Text = "Mal Muy MAL"
0040107D	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner

我们可以看到断在了我们刚刚设置的断点处,我们在数据窗口中定位 40300C 内存单元。

004010A8	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd
004010AB	E8 38000000	CALL <JMP.&user32.EndDialog>	EndDialog
004010B0	EB 10	JMP SHORT Leccion_.004010B0	
DS:[0040300C]=4B43554C EBX=00000000			
Address	Hex dump	ASCII	
0040300C	4C 55 43 4B 59 00 00 00	LUCKY...	
00403014	00 00 00 00 00 00 00 00	
0040301C	00 00 00 00 00 00 00 00	
00403024	00 00 00 00 00 00 00 00	
0040302C	00 00 00 00 00 00 00 00	

该指令 40300C 内存单元中 4 字节的内容保存到 EBX 寄存器中。

Registers (FPU)	
EAX	00000005
ECX	77D3219D user32.77D3219D
EDX	7C91EB94 ntdll.KiFastSystemCallRe
EBX	4B43554C
ESP	0012FB18
EBP	0012FB18
ESI	0040102E Leccion_.0040102E
EDI	0012FB80
EIP	0040106A Leccion_.0040106A
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 0	DS 0023 32bit 0(FFFFFFFF)
S 1	FS 003B 32bit 7FFDE000(FFF)

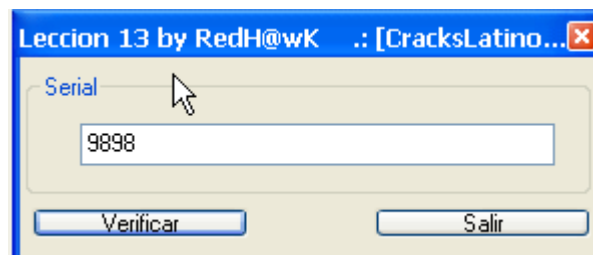
我们按 F7 键单步。

00401057	. 68 EA030000	PUSH 3EA
0040105C	. FF75 08	PUSH DWORD PTR SS:[EBP+8]
0040105F	. E8 8A000000	CALL <JMP.&user32.GetDlgItemTextA>
00401064	. 8B1D 0C304000	MOV EBX,DWORD PTR DS:[40300C]
0040106A	. 8B15 4B204000	MOV EDX,DWORD PTR DS:[40204B]
00401070	. 3BD8	CMP EBX,EDX
00401072	. 74 13	JE SHORT Leccion_.00401087
00401074	. 6A 00	PUSH 0

该指令将 40204B 内存单元中内容保存到 EDX 寄存器中,我们来看看 40204B 内存单元的内容是什么。

Address	Hex dump	ASCII
0040204B	39 38 39 38 00 00 00 00	9898....
00402053	00 AC 20 00 00 00 00 00	.%.....
0040205B	00 00 00 00 00 EE 20 00-
00402063	00 08 20 00 00 B8 20 00	. .@ .
0040206B	00 00 00 00 00 00 00 00

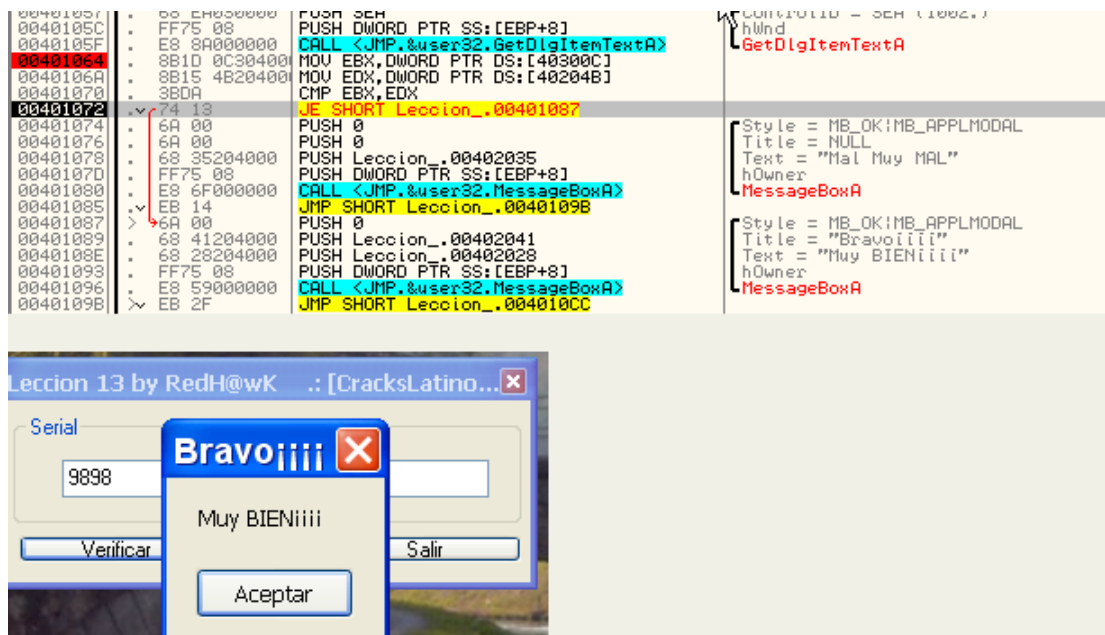
这里存放着 4 个字节的字符串“9898”,当我们输入的序列号前 4 个字节与“9898”相等的话,就跳转到正确的消息框处。不相等的话,就提示错误的消息框。我们重启 OD 输入正确的序列号“9898”。



单击“Verificar”(验证)。

0040105C	. FF75 08	PUSH DWORD PTR SS:[EBP+8]
0040105F	. E8 8A000000	CALL <JMP.&user32.GetDlgItemTextA>
00401064	. 8B1D 0C304000	MOV EBX,DWORD PTR DS:[40300C]
0040106A	. 8B15 4B204000	MOV EDX,DWORD PTR DS:[40204B]
00401070	. 3BD8	CMP EBX,EDX
00401072	. 74 13	JE SHORT Leccion_.00401087
00401074	. 6A 00	PUSH 0
00401076	. 6A 00	PUSH 0
00401078	. 68 35204000	PUSH Leccion_.00402035
0040107D	. FF75 08	PUSH DWORD PTR SS:[EBP+8]
00401080	. E8 6F000000	CALL <JMP.&user32.MessageBoxA>
00401085	. EB 14	JMP SHORT Leccion_.0040109B
00401087	. 6A 00	PUSH 0
00401089	. 68 41204000	PUSH Leccion_.00402041
0040108E	. 68 28204000	PUSH Leccion_.00402028
00401093	. FF75 08	PUSH DWORD PTR SS:[EBP+8]
00401096	. E8 59000000	CALL <JMP.&user32.MessageBoxA>
0040109B	. EB 2F	JMP SHORT Leccion_.004010B1
0040109D	. 317D 10 EC03	CMP DWORD PTR SS:[EBP+10]
004010A4	. 75 26	JNZ SHORT Leccion_.004010B1
004010A6	. 6A 00	PUSH 0
004010A8	. FF75 08	PUSH DWORD PTR SS:[EBP+8]
004010AB	. E8 38000000	CALL <JMP.&user32.EndDialog>
004010B0	. EB 10	JMP SHORT Leccion_.004010B1

当前 EBX 和 EDX 的值都是 38393839,对应字符串“9898”,所以会跳转到提示“Muy BIEN”(正确的)的消息框处。



我们可以看到,在这里我朋友把正确提示框的标题改成了”Bravo”(恭喜),哈哈,我们又找到了正确的序列号。

接下来的章节,我们将增加难度-两个相对难一点的硬编码序列号的 CrackMe。

这里给一个练习的小例子,第 3 个 CrackMe,名为”Mielecrackme1.zip”。

提供一点思路,关键 API 函数-lstrcmpA,这个 CrackMe 会直接进行字符串的比较。你定位到了该函数以后,看看堆栈中函数参数情况。

接下来的第 14 章,我们再来介绍这个两个相对难一点的硬编码的例子,这里,大家先练习一下这第 3 个 CrackMe 吧。