

## 第十二章：一个难啃的 NOOB 例子

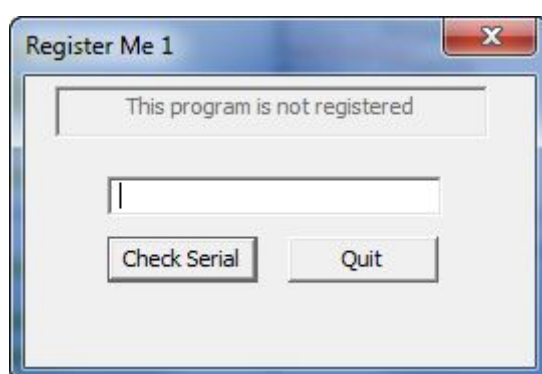
### 一、简介

本章我们将研究一个有点更具挑战性的程序。它叫 **ReverseMe1**，我写的。我也会讨论一个 Olly 的插件“ASCII 码表”。可以在[工具](#)页下载它。这个 **ReverseMe** 是用来说明为什么 LAME 补丁方式通常就是那么 lame（烂）的一个极好的例子。

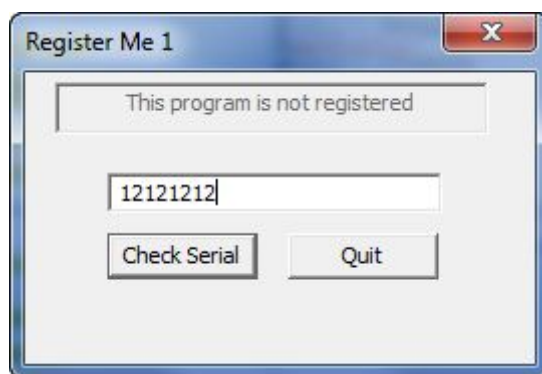
你可以在[教程](#)页下载相关文件及本文的 PDF 版。

### 二、准备开始

运行下程序看看：



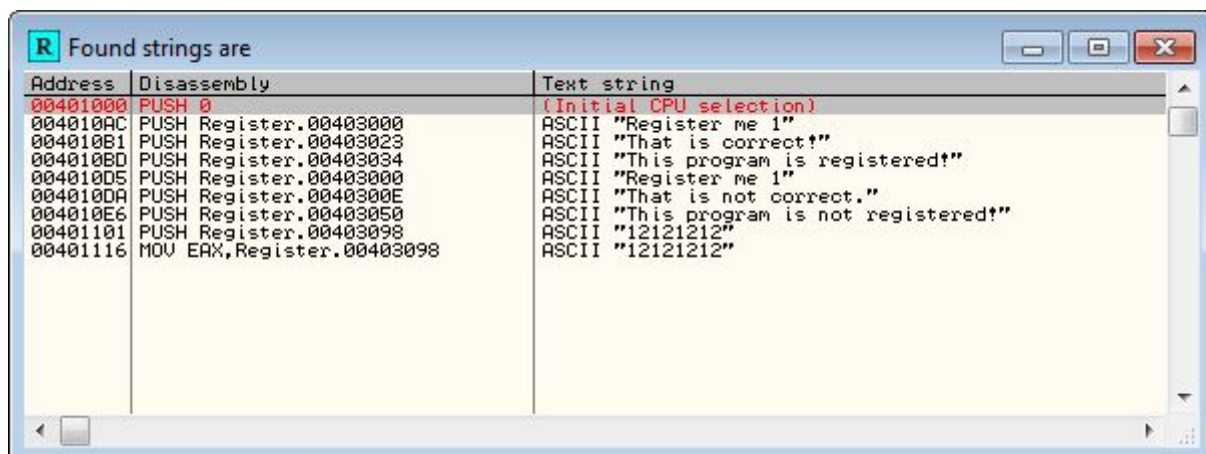
我们能看到它说还没有注册，需要序列号。那就给它一个：



点“Check Serial”：



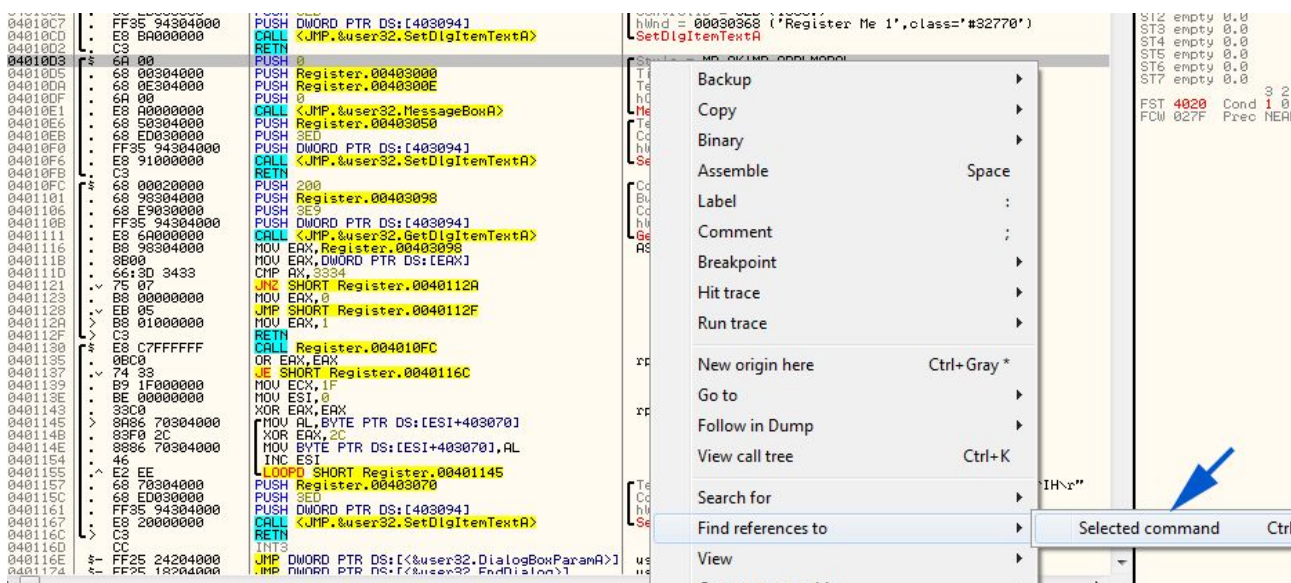
我们看到我们是错的（再一次）！Ollly 载入应用，用咱们信得过的“搜索字符串”：



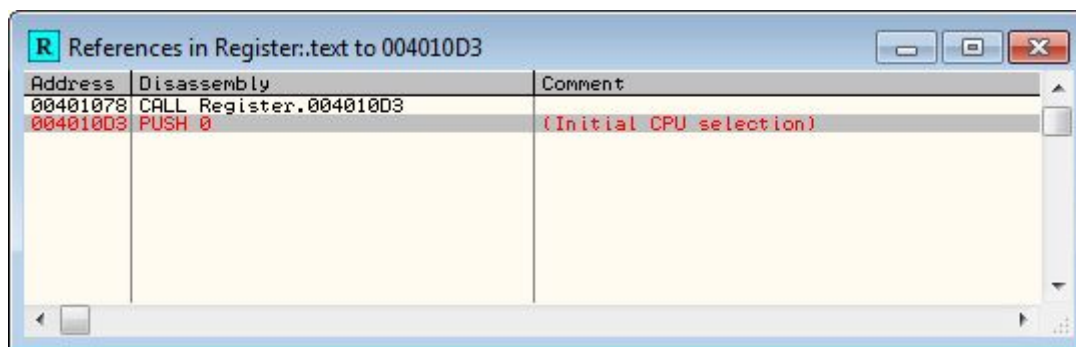
好哇，看起来前途光明呀。咱们来检查下 “That is not correct” 字符串：



咱们来到了问题的核心。因为每一个都是单独的方法，我们需要看看哪里调用了它们，所以咱们要这么做：



Ollly 弹出了 References 窗口：



我们能够看到有一个对该函数的调用。咱们双击它，看看它是啥样的：

00401063	E8 94000000	CALL Register.004010FC	
00401068	0BC0	OR EAX,EAX	rpport4.75CB1B9C
0040106A	75 0C	JNZ SHORT Register.00401078	
0040106C	E8 39000000	CALL Register.004010AA	
00401071	E8 BA000000	CALL Register.00401130	
00401076	EB 2C	JMP SHORT Register.004010A4	
00401078	E8 56000000	CALL Register.004010D3	
0040107D	EB 25	JMP SHORT Register.004010A4	
0040107F	817D 10 EC030000	CMP [ARG.3],SEC	
00401086	75 1C	JNZ SHORT Register.004010A4	
00401088	6A 00	PUSH 0	[Result = 0 hWnd = 0252006C EndDialog
0040108A	FF75 08	PUSH [ARG.1]	
0040108D	E8 E2000000	CALL <JMP.&user32.EndDialog>	
00401092	EB 10	JMP SHORT Register.004010A4	
00401094	837D 0C 10	CMP [ARG.2],10	
00401098	75 0A	JNZ SHORT Register.004010A4	
0040109A	6A 00	PUSH 0	[Result = 0 hWnd = 0252006C EndDialog rpport4.75CB1B9C
0040109C	FF75 08	PUSH [ARG.1]	
0040109F	E8 D0000000	CALL <JMP.&user32.EndDialog>	
004010A4	33C0	XOR EAX,EAX	
004010A6	C9	LEAVE	
004010A7	C2 1000	RETN 10	
004010AA	6A 00	PUSH 0	[Style = MB_OK!MB_APPLMODAL Title = "Register me 1" Text = "That is correct!" hOwner = NULL MessageBoxA
004010AC	68 00304000	PUSH Register.00403000	[Text = "This program is registered!" ControlID = 3ED (1005.) hWnd = 00030368 ('Register Me 1',class='#32770') SetDlgItemTextA
004010B1	68 23304000	PUSH Register.00403023	
004010B6	6A 00	PUSH 0	
004010B8	E8 C9000000	CALL <JMP.&user32.MessageBoxA>	
004010BD	68 34304000	PUSH Register.00403034	
004010C2	68 ED030000	PUSH 3ED	
004010C7	FF35 94304000	PUSH DWORD PTR DS:[403094]	
004010CD	E8 8A000000	CALL <JMP.&user32.SetDlgItemTextA>	
004010D2	C3	RETN	
004010D3	6A 00	PUSH 0	[Style = MB_OK!MB_APPLMODAL Title = "Register me 1" Text = "That is not correct." hOwner = NULL MessageBoxA
004010D5	68 00304000	PUSH Register.00403000	[Text = "This program is not registered!" ControlID = 3ED (1005.) hWnd = 00030368 ('Register Me 1',class='#32770') SetDlgItemTextA
004010DA	68 0E304000	PUSH Register.0040300E	
004010DF	6A 00	PUSH 0	
004010E1	E8 A0000000	CALL <JMP.&user32.MessageBoxA>	
004010E6	68 50304000	PUSH Register.00403050	
004010EB	68 ED030000	PUSH 3ED	
004010F0	FF35 94304000	PUSH DWORD PTR DS:[403094]	
004010F6	E8 91000000	CALL <JMP.&user32.SetDlgItemTextA>	
004010FB	C3	RETN	
004010FC	68 00020000	PUSH 200	[Count = 200 (512.) Buffer = Register.00403098 ControlID = 3E9 (1001.) hWnd = 00030368 ('Register Me 1',class='#32770') GetDlgItemTextA ASCII "12121212"
00401101	68 98304000	PUSH Register.00403098	
00401106	68 E9030000	PUSH 3E9	
0040110B	FF35 94304000	PUSH DWORD PTR DS:[403094]	
00401111	E8 6A000000	CALL <JMP.&user32.GetDlgItemTextA>	
00401116	B8 98304000	MOV EAX,Register.00403098	
0040111B	8B00	MOV EAX,DWORD PTR DS:[EAX]	
0040111D	66:3D 3433	CMPSX 3433	
00401121	75 07	JNZ SHORT Register.0040112A	
00401123	B8 00000000	MOV EAX,0	

这里，我们能看到坏消息是在 401078 处调用的，并且我们马上就能看到 40106A 处有个跳转指令跳到这里：



<pre> 00401063 . E8 94000000 CALL Register.004010FC 00401068 . 00C0 OR EAX,EAX 0040106A . 75 0C JNZ SHORT Register.00401078 0040106C . E8 39000000 CALL Register.004010AA 00401071 . E8 BA000000 CALL Register.00401130 00401076 . EB 2C JMP SHORT Register.004010A4 00401078 . E8 56000000 CALL Register.004010D3 0040107A . EB 25 JMP SHORT Register.004010A4 0040107C . 817D 10 EC030000 CMP [ARG_3],ECX 0040107E . 75 1C JNZ SHORT Register.004010A4 00401080 . 6A 00 PUSH 0 00401082 . FF75 08 PUSH [ARG_1] 00401084 . E8 E2000000 CALL &lt;JMP.&amp;user32.EndDialog&gt; 00401086 . EB 10 JMP SHORT Register.004010A4 00401088 . 837D 0C 10 CMP [ARG_2],10 0040108A . 75 0A JNZ SHORT Register.004010A4 0040108C . 6A 00 PUSH 0 0040108E . FF75 08 PUSH [ARG_1] 00401090 . E8 D0000000 CALL &lt;JMP.&amp;user32.EndDialog&gt; 00401092 . 33C0 XOR EAX,EAX 00401094 . C9 LEAVE 00401096 . C2 1000 RETN 10 00401098 . 6A 00 PUSH 0 0040109A . 68 00304000 PUSH Register.00403000 0040109C . 68 23304000 PUSH Register.00403023 0040109E . 6A 00 PUSH 0 004010A0 . CALL &lt;JMP.&amp;user32.MessageBoxA&gt; 004010A2 . PUSH Register.00403034 004010A4 . 68 C9000000 PUSH 3E9 004010A6 . 68 34304000 PUSH DWORD PTR DS:[403094] 004010A8 . FF35 94304000 CALL &lt;JMP.&amp;user32.SetDlgItemTextA&gt; 004010AA . E8 BA000000 CALL &lt;JMP.&amp;user32.SetDlgItemTextA&gt; 004010AC . C3 RETN 004010AE . 6A 00 PUSH 0 004010B0 . 68 00304000 PUSH Register.00403000 004010B2 . 68 0E304000 PUSH Register.0040300E 004010B4 . 6A 00 PUSH 0 004010B6 . CALL &lt;JMP.&amp;user32.MessageBoxA&gt; 004010B8 . PUSH Register.00403050 004010BA . 68 ED030000 PUSH 3ED 004010BC . 68 34304000 PUSH DWORD PTR DS:[403094] 004010BE . FF35 94304000 CALL &lt;JMP.&amp;user32.SetDlgItemTextA&gt; 004010C0 . E8 91000000 CALL &lt;JMP.&amp;user32.SetDlgItemTextA&gt; 004010C2 . C3 RETN 004010C4 . 68 00020000 PUSH 200 004010C6 . 68 98304000 PUSH Register.00403098 004010C8 . 68 E9030000 PUSH 3E9 004010CA . 68 34304000 PUSH DWORD PTR DS:[403094] 004010CC . FF35 94304000 CALL &lt;JMP.&amp;user32.SetDlgItemTextA&gt; 004010CE . E8 6A000000 CALL &lt;JMP.&amp;user32.GetDlgItemTextA&gt; 004010D0 . 8B 98304000 MOV EAX,Register.00403098 004010D2 . 3B00 CMP EAX,DWORD PTR DS:[EAX] 004010D4 . 66:3D 3433 CMP AX,3334 004010D6 . 75 0C JNZ SHORT Register.00401078 </pre>	<pre> t4.75CB1B9C [Result = 0 hWnd = 0252006C EndDialog  Style = MB_OK!MB_APPLMODAL Title = "Register me 1" Text = "That is correct!" hOwner = NULL MessageBoxA Text = "This program is registered!" ControlID = 3ED (1005.) hWnd = 00030368 ('Register Me 1',class='#32770') SetDlgItemTextA  Style = MB_OK!MB_APPLMODAL Title = "Register me 1" Text = "That is not correct." hOwner = NULL MessageBoxA Text = "This program is not registered!" ControlID = 3ED (1005.) hWnd = 00030368 ('Register Me 1',class='#32770') SetDlgItemTextA  Count = 200 (512.) Buffer = Register.00403098 ControlID = 3E9 (1001.) hWnd = 00030368 ('Register Me 1',class='#32770') GetDlgItemTextA ASCIIZ "12121212" </pre>
--	--

向上滚几行，我们就能看到有一个 **CALL**，用来检测 程序/比较/跳转，和我们前面看到的一样。从这里我们能够猜到，主要的检测程序是在 **4010FC**，**401063** 处调用了它。在返回后，**EAX** 寄存器被检测其值是否是 **0**，如果不是就跳到坏消息。

00401059	75 24	JNZ SHORT Register.0040107F	
0040105B	8B 45 08	MOV EAX, [ARG.1]	
0040105E	A3 94304000	MOV DWORD PTR DS:[403094],EAX	
00401063	E8 94000000	CALL Register.004010FC	<b>This is the main call..</b>
00401068	0BC0	OR EAX,EAX	
0040106A	75 0C	JNZ SHORT Register.00401078	
0040106C	E8 39000000	CALL Register.004010AA	
00401071	E8 BA000000	CALL Register.00401130	
00401076	EB 2C	JMP SHORT Register.004010A4	
00401078	E8 56000000	CALL Register.004010D3	
0040107D	EB 25	JMP SHORT Register.004010A4	
0040107F	81 7D 10 EC030000	CMPL [ARG.3],SEC	
00401086	75 1C	JNZ SHORT Register.004010A4	
00401088	6A 00	PUSH 0	[Result = 0 hWnd = 0252006C EndDialog
0040108A	FF 75 08	PUSH [ARG.1]	
0040108D	E8 E2000000	CALL <JMP.&user32.EndDialog>	
00401092	EB 10	JMP SHORT Register.004010A4	
00401094	83 7D 0C 10	CMPL [ARG.2],10	
00401098	75 0A	JNZ SHORT Register.004010A4	
0040109A	6A 00	PUSH 0	[Result = 0 hWnd = 0252006C EndDialog rport4.75CB1B9C
0040109C	FF 75 08	PUSH [ARG.1]	
0040109F	E8 D0000000	CALL <JMP.&user32.EndDialog>	
004010A4	33 C0	XOR EAX,EAX	
004010A6	C9	LEAVE	
004010A7	C2 1000	RETN 10	
004010AA	6A 00	PUSH 0	[Style = MB_OK MB_APPLMODAL Title = "Register me 1" Text = "That is correct!" hOwner = NULL MessageBoxA
004010AC	68 00304000	PUSH Register.00403000	
004010B1	68 23304000	PUSH Register.00403023	
004010B6	6A 00	PUSH 0	
004010B8	E8 C9000000	CALL <JMP.&user32.MessageBoxA>	
004010BD	68 34304000	PUSH Register.00403034	
004010C2	68 ED030000	PUSH 3ED	[Text = "This program is registered!" ControlID = 3ED (1005.) hWnd = 00030368 ('Register Me 1',class='SetDlgItemTextA
004010C7	FF 35 94304000	PUSH DWORD PTR DS:[403094]	
004010CD	E8 BA000000	CALL <JMP.&user32.SetDlgItemTextA>	
004010D2	C3	RETN	
004010D3	6A 00	PUSH 0	[Style = MB_OK MB_APPLMODAL Title = "Register me 1" Text = "That is not correct." hOwner = NULL MessageBoxA
004010D8	68 00304000	PUSH Register.00403000	
004010DB	68 0040300E	PUSH Register.0040300E	
004010E0	6A 00	PUSH 0	
004010E2	CALL <JMP.&user32.MessageBoxA>		
004010E8	68 00304000	PUSH Register.00403000	
004010EB	68 ED030000	PUSH 3ED	[Text = "This program is not registered!" ControlID = 3ED (1005.) hWnd = 00030368 ('Register Me 1',class='SetDlgItemTextA
004010F0	FF 35 94304000	PUSH DWORD PTR DS:[403094]	
004010F6	E8 91000000	CALL <JMP.&user32.SetDlgItemTextA>	
004010FB	C3	RETN	
004010FC	68 00020000	PUSH 200	[Count = 200 (512.) Buffer = Register.00403098 ControlID = 3E9 (1001.) hWnd = 00030368 ('Register Me 1',class='SetDlgItemTextA ASCII "12121212"
00401101	68 98304000	PUSH Register.00403098	
00401106	68 E9030000	PUSH 3E9	
0040110B	FF 35 94304000	PUSH DWORD PTR DS:[403094]	
00401111	E8 6A000000	CALL <JMP.&user32.GetDlgItemTextA>	
00401116	B8 98304000	MOV EAX,Register.00403098	
0040111B	8B 00	MOV EAX,DWORD PTR DS:[EAX]	
0040111D	66 3D 3433	CMPL AX,3334	
00401121	75 07	JNZ SHORT Register.0040112A	
00401123	B8 00000000	MOV EAX,0	
00401128	EB 05	JMP SHORT Register.0040112F	
0040112A	B8 01000000	MOV EAX,1	
0040112F	C3	RETN	
00401130	E8 C7FFFFFF	CALL Register.004010FC	
00401135	0BC0	OR EAX,EAX	rport4.75CB1B9C
00401137	74 33	JE SHORT Register.0040116C	

测试下我们的假设，在 40106A 处设置断点，然后重启应用。在输入一个序列号以后（我输入的还是“12121212”），我们断在了调用序列号校验的那个 CALL 后面的跳转处：

0040105E	A3 94304000	MOV DWORD PTR DS:[403094],EAX
00401063	E8 94000000	CALL Register.004010FC
00401068	0BC0	OR EAX,EAX
0040106A	75 0C	JNZ SHORT Register.00401078
0040106C	E8 39000000	CALL Register.004010AA
00401071	E8 BA000000	CALL Register.00401130
00401076	EB 2C	JMP SHORT Register.004010A4
00401078	E8 56000000	CALL Register.004010D3
0040107D	EB 25	JMP SHORT Register.004010A4
0040107F	81 7D 10 EC030000	CMPL [ARG.3],SEC
00401086	75 1C	JNZ SHORT Register.004010A4
00401088	6A 00	PUSH 0
0040108A	FF 75 08	PUSH [ARG.1]
0040108D	E8 E2000000	CALL <JMP.&user32.EndDialog>
00401092	EB 10	JMP SHORT Register.004010A4
00401094	83 7D 0C 10	CMPL [ARG.2],10
00401098	75 0A	JNZ SHORT Register.004010A4
0040109A	6A 00	PUSH 0

现在咱们帮 Olly 走正确的路，所以我们不能让跳转实现（直接到调用好消息的 CALL 那）：

```

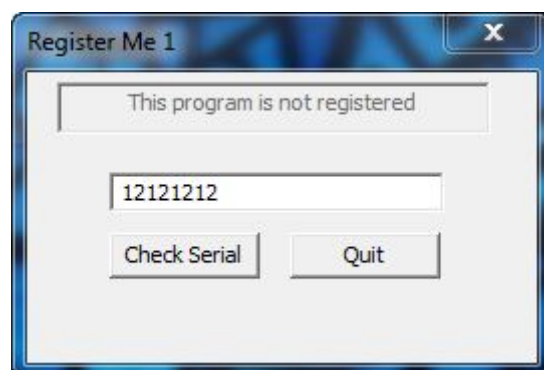
C 0 ES 002
P 0 CS 001
A 0 SS 002
Z 1 DS 002
S 0 FS 003
T 0 GS 000
D 0
O 0 LastErr

```

点一下运行：



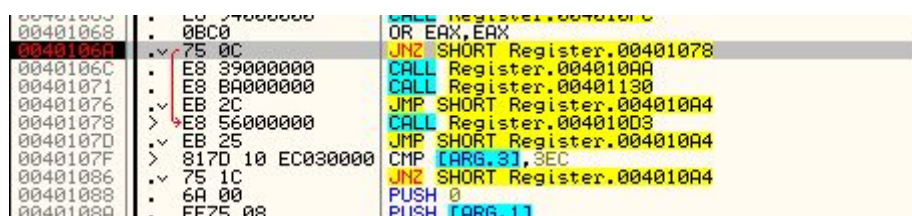
耶，so easy（妈妈再也不用担心我破不了了）!!点 OK:



噢，++%\$@，这里他爸的发生了啥，你个阿西吧\$\$\$%^#!!!!!!很明显，我们的程序没有注册成功。这说明我们肯定错过了啥。

### 三、进一步分析

重启应用，输入序列号，让 Olly 再次断在 40106A:



看看这个，如果我们阻止 Olly 跳到坏消息那，直接执行 40106C 的那个 CALL，就是调用 4010AA。沿着那条路往下走，我们能看到它是相当的标准：它弹出一个显示 “That is not correct” 的消息框，然后将主窗口的标签修改成 “This program is registered!”。



00401068	0BC0	OR EAX,EAX	
00401069	75 0C	JNZ SHORT Register.00401078	
0040106C	E8 39000000	CALL Register.0040109A	
00401071	E8 B0000000	CALL Register.00401130	
00401076	EB 2C	JMP SHORT Register.004010A4	
00401078	E8 56000000	CALL Register.004010D3	
0040107D	EB 25	JMP SHORT Register.004010A4	
0040107F	817D 10 EC030000	CMP [ARG.3],SEC	
00401086	75 1C	JNZ SHORT Register.004010A4	
00401088	6A 00	PUSH 0	
0040108A	FF75 08	PUSH [ARG.1]	
0040108D	E8 E2000000	CALL <JMP.&user32.EndDialog>	[Result = 0 hWnd = 000503B2 ('Register Me 1',class='#32770') EndDialog
00401092	EB 10	JMP SHORT Register.004010A4	
00401094	837D 0C 10	CMP [ARG.2],10	
00401098	75 0A	JNZ SHORT Register.004010A4	
0040109A	6A 00	PUSH 0	
0040109C	FF75 08	PUSH [ARG.1]	
0040109F	E8 D0000000	CALL <JMP.&user32.EndDialog>	[Result = 0 hWnd = 000503B2 ('Register Me 1',class='#32770') EndDialog
004010A4	33C0	XOR EAX,EAX	
004010A6	C9	LEAVE	
004010A7	C2 1000	RETN 10	
004010AA	6A 00	PUSH 0	
004010AC	68 00304000	PUSH Register.00403000	[Style = MB_OK MB_APPLMODAL Title = "Register me 1"
004010B1	68 23304000	PUSH Register.00403023	Text = "That is correct!" hOwner = NULL
004010B6	6A 00	PUSH 0	MessageBoxA
004010B8	E8 C9000000	CALL <JMP.&user32.MessageBoxA>	Text = "This program is registered!"
004010BD	68 34304000	PUSH Register.00403034	ControlID = 3ED (1005.)
004010C2	68 ED030000	PUSH 3ED	hWnd = 000503B2 ('Register Me 1',class='#32770')
004010C7	FF35 94304000	PUSH DWORD PTR DS:[403094]	SetDlgItemTextA
004010CD	E8 BA000000	CALL <JMP.&user32.SetDlgItemTextA>	
004010D2	C3	RETN	

等等!一旦我们从那个 CALL 返回了,在 401071 还有另一个 CALL 等着咱:

00401068	0BC0	OR EAX,EAX	
00401069	75 0C	JNZ SHORT Register.00401078	
0040106C	E8 39000000	CALL Register.0040109A	
00401071	E8 B0000000	CALL Register.00401130	
00401076	EB 2C	JMP SHORT Register.004010A4	
00401078	E8 56000000	CALL Register.004010D3	
0040107D	EB 25	JMP SHORT Register.004010A4	
0040107F	817D 10 EC030000	CMP [ARG.3],SEC	
00401086	75 1C	JNZ SHORT Register.004010A4	
00401088	6A 00	PUSH 0	
0040108A	FF75 08	PUSH [ARG.1]	
0040108D	E8 E2000000	CALL <JMP.&user32.EndDialog>	[Result = 0 hWnd = 000503B2 ('Register Me 1',class='#32770') EndDialog
00401092	EB 10	JMP SHORT Register.004010A4	
00401094	837D 0C 10	CMP [ARG.2],10	
00401098	75 0A	JNZ SHORT Register.004010A4	
0040109A	6A 00	PUSH 0	
0040109C	FF75 08	PUSH [ARG.1]	
0040109F	E8 D0000000	CALL <JMP.&user32.EndDialog>	[Result = 0 hWnd = 000503B2 ('Register Me 1',class='#32770') EndDialog
004010A4	33C0	XOR EAX,EAX	
004010A6	C9	LEAVE	
004010A7	C2 1000	RETN 10	
004010AA	6A 00	PUSH 0	
004010AC	68 00304000	PUSH Register.00403000	[Style = MB_OK MB_APPLMODAL Title = "Register me 1"
004010B1	68 23304000	PUSH Register.00403023	Text = "That is correct!" hOwner = NULL
004010B6	6A 00	PUSH 0	MessageBoxA
004010B8	E8 C9000000	CALL <JMP.&user32.MessageBoxA>	Text = "This program is registered!"
004010BD	68 34304000	PUSH Register.00403034	ControlID = 3ED (1005.)
004010C2	68 ED030000	PUSH 3ED	hWnd = 000503B2 ('Register Me 1',class='#32770')
004010C7	FF35 94304000	PUSH DWORD PTR DS:[403094]	SetDlgItemTextA
004010CD	E8 BA000000	CALL <JMP.&user32.SetDlgItemTextA>	
004010D2	C3	RETN	
004010D3	6A 00	PUSH 0	
004010D5	68 00304000	PUSH Register.00403000	[Style = MB_OK MB_APPLMODAL Title = "Register me 1"

该 CALL 调用的是 401130,所以咱们看看那个子程序。首先,我们注意它调用了 SetDlgItemTextA,不过有一个看起来很奇怪的字符串。咱们来一行一行的执行。401130 有个 CALL 调用了 4010FC。往上看,我们看到这是一个序列号校验子程序。然后 EAX 自身做了 OR 操作看是否为 0,如果不是,它执行了许多看起来很怪异的玩意儿:

0040112F	C3	RETN	
00401130	E8 C7FFFFFF	CALL Register.004010FC	Call to serial check
00401135	0BC0	OR EAX,EAX	
00401137	74 33	JE SHORT Register.0040116C	If EAX != 0...
00401139	B9 1F000000	MOV ECX,1F	
0040113E	BE 00000000	MOV ESI,0	
00401143	33C0	XOR EAX,EAX	
00401145	8A86 70304000	MOV AL,BYTE PTR DS:[ESI+403070]	
00401148	83F0 2C	XOR EAX,2C	
0040114E	8886 70304000	MOV BYTE PTR DS:[ESI+403070],AL	
00401152	4E	INC ESI	
00401154	75 00	LOOPD SHORT Register.00401145	
00401156	6A 00	PUSH 0	
00401158	68 00304000	PUSH Register.00403000	
0040115D	68 34304000	PUSH Register.00403034	
00401162	E8 20000000	CALL <JMP.&user32.SetDlgItemTextA>	Text = "xDE_\x0C^CK^MA\x0CE_\x0CBCX\x0C^IKE^XI^IH\r" ControlID = 3ED (1005.) hWnd = 000503B2 ('Register Me 1',class='#32770') SetDlgItemTextA
0040116C	C3	RETN	

到目前为止,我们从这些收集到的信息是,在我们给程序打了补丁后它显示了好消息,然后另一个 CALL 执行了,在这个 CALL 里,又有一个 CALL 再次执行了序列号校验子程序,对结果做了同样的分析。这是一个备份检测点!现在我们来看看如果我们在这个备份检测点失败的话会怎样(这里我们是可以让它检测失败的,因为我们只给那个跳转打了补丁):





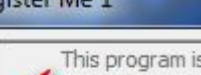
```

00401133  MOV ECX, 1F000000
00401137  MOV ESI, 0
00401139  MOV EAX, 3300
0040113E  MOV EAX, 70304000
00401143  XOR EAX, 2C
00401145  MOV BYTE PTR DS:[ESI+403070], AL
00401148  MOV BYTE PTR DS:[ESI+403070], AL
0040114E  INC ESI
00401154  LOOPD SHORT Register.00401145
00401155  PUSH Register.00403070
00401157  PUSH 3ED
0040115C  PUSH DWORD PTR DS:[403094]
00401161  CALL JMP.&user32.SetDlgItemTextA
00401167  RETN
0040116C  INT3
0040116D  JMP DWORD PTR DS:[&user32.ShowDialogParamA]
0040116E  user32.ShowDialogParamA
00401174  JMP DWORD PTR DS:[&user32.EndDialog]
00401176  user32.EndDialog
0040117A  JMP DWORD PTR DS:[&user32.GetDlgItem]
0040117D  user32.GetDlgItem
0040117F  JMP DWORD PTR DS:[&user32.GetDlgItemTextA]
00401182  user32.GetDlgItemTextA
00401186  JMP DWORD PTR DS:[&user32.MessageBoxA]
00401189  user32.MessageBoxA
0040118C  JMP DWORD PTR DS:[&user32.SetDlgItemTextA]
0040118F  user32.SetDlgItemTextA
00401192  JMP DWORD PTR DS:[&user32.SetFocus]
00401195  user32.SetFocus
00401198  JMP DWORD PTR DS:[&kernel32.ExitProcess]
0040119B  kernel32.ExitProcess
0040119E  JMP DWORD PTR DS:[&kernel32.GetModuleHandleA]
004011A0  kernel32.GetModuleHandleA
004011A4  DB 00
004011A5  DB 00
004011A6  DB 00
004011A7  DB 00
004011A8  DB 00
004011A9  DB 00
004011AA  DB 00
004011AB  DB 00
004011AC  DB 00
004011AD  DB 00
004011AE  DB 00
004011AF  DB 00
004011B0  DB 00
004011B1  DB 00
004011B2  DB 00
004011B3  DB 00
004011B4  DB 00
004011B5  DB 00
004011B6  DB 00
004011B7  DB 00
004011B8  DB 00
004011B9  DB 00
004011BA  DB 00
004011BB  DB 00
004011BC  DB 00
004011BD  DB 00
004011BE  DB 00
004011BF  DB 00
004011C0  DB 00
004011C1  DB 00
004011C2  DB 00
004011C3  DB 00
004011C4  DB 00
004011C5  DB 00
004011C6  DB 00
004011C7  DB 00
004011C8  DB 00
004011C9  DB 00
004011CA  DB 00
004011CB  DB 00
004011CC  DB 00
004011CD  DB 00
004011CE  DB 00
004011CF  DB 00
004011D0  DB 00
004011D1  DB 00
004011D2  DB 00
004011D3  DB 00
004011D4  DB 00
004011D5  DB 00
004011D6  DB 00
004011D7  DB 00
004011D8  DB 00
004011D9  DB 00
004011DA  DB 00
004011DB  DB 00
004011DC  DB 00
004011DD  DB 00
004011DE  DB 00
004011DF  DB 00
004011E0  DB 00
004011E1  DB 00
004011E2  DB 00
004011E3  DB 00
004011E4  DB 00
004011E5  DB 00
004011E6  DB 00
004011E7  DB 00
004011E8  DB 00
004011E9  DB 00
004011EA  DB 00
004011EB  DB 00
004011EC  DB 00
004011ED  DB 00
004011EE  DB 00
004011EF  DB 00
004011F0  DB 00
004011F1  DB 00
004011F2  DB 00
004011F3  DB 00
004011F4  DB 00
004011F5  DB 00
004011F6  DB 00
004011F7  DB 00
004011F8  DB 00
004011F9  DB 00
004011FA  DB 00
004011FB  DB 00
004011FC  DB 00
004011FD  DB 00
004011FE  DB 00
004011FF  DB 00
00401200  DB 00
00401201  DB 00
00401202  DB 00
00401203  DB 00
00401204  DB 00
00401205  DB 00
00401206  DB 00
00401207  DB 00
00401208  DB 00
00401209  DB 00
0040120A  DB 00
0040120B  DB 00
0040120C  DB 00
0040120D  DB 00
0040120E  DB 00
0040120F  DB 00
00401210  DB 00
00401211  DB 00
00401212  DB 00
00401213  DB 00
00401214  DB 00
00401215  DB 00
00401216  DB 00
00401217  DB 00
00401218  DB 00
00401219  DB 00
0040121A  DB 00
0040121B  DB 00
0040121C  DB 00
0040121D  DB 00
0040121E  DB 00
0040121F  DB 00
00401220  DB 00
00401221  DB 00
00401222  DB 00
00401223  DB 00
00401224  DB 00
00401225  DB 00
00401226  DB 00
00401227  DB 00
00401228  DB 00
00401229  DB 00
0040122A  DB 00
0040122B  DB 00
0040122C  DB 00
0040122D  DB 00
0040122E  DB 00
0040122F  DB 00
00401230  DB 00
00401231  DB 00
00401232  DB 00
00401233  DB 00
00401234  DB 00
00401235  DB 00
00401236  DB 00
00401237  DB 00
00401238  DB 00
00401239  DB 00
0040123A  DB 00
0040123B  DB 00
0040123C  DB 00
0040123D  DB 00
0040123E  DB 00
0040123F  DB 00
00401240  DB 00
00401241  DB 00
00401242  DB 00
00401243  DB 00
00401244  DB 00
00401245  DB 00
00401246  DB 00
00401247  DB 00
00401248  DB 00
00401249  DB 00
0040124A  DB 00
0040124B  DB 00
0040124C  DB 00
0040124D  DB 00
0040124E  DB 00
0040124F  DB 00
00401250  DB 00
00401251  DB 00
00401252  DB 00
00401253  DB 00
00401254  DB 00
00401255  DB 00
00401256  DB 00
00401257  DB 00
00401258  DB 00
00401259  DB 00
0040125A  DB 00
0040125B  DB 00
0040125C  DB 00
0040125D  DB 00
0040125E  DB 00
0040125F  DB 00
00401260  DB 00
00401261  DB 00
00401262  DB 00
00401263  DB 00
00401264  DB 00
00401265  DB 00
00401266  DB 00
00401267  DB 00
00401268  DB 00
00401269  DB 00
0040126A  DB 00
0040126B  DB 00
0040126C  DB 00
0040126D  DB 00
0040126E  DB 00
0040126F  DB 00
00401270  DB 00
00401271  DB 00
00401272  DB 00
00401273  DB 00
00401274  DB 00
00401275  DB 00
00401276  DB 00
00401277  DB 00
00401278  DB 00
00401279  DB 00
0040127A  DB 00
0040127B  DB 00
0040127C  DB 00
0040127D  DB 00
0040127E  DB 00
0040127F  DB 00
00401280  DB 00
00401281  DB 00
00401282  DB 00
00401283  DB 00
00401284  DB 00
00401285  DB 00
00401286  DB 00
00401287  DB 00
00401288  DB 00
00401289  DB 00
0040128A  DB 00
0040128B  DB 00
0040128C  DB 00
0040128D  DB 00
0040128E  DB 00
0040128F  DB 00
00401290  DB 00
00401291  DB 00
00401292  DB 00
00401293  DB 00
00401294  DB 00
00401295  DB 00
00401296  DB 00
00401297  DB 00
00401298  DB 00
00401299  DB 00
004
```

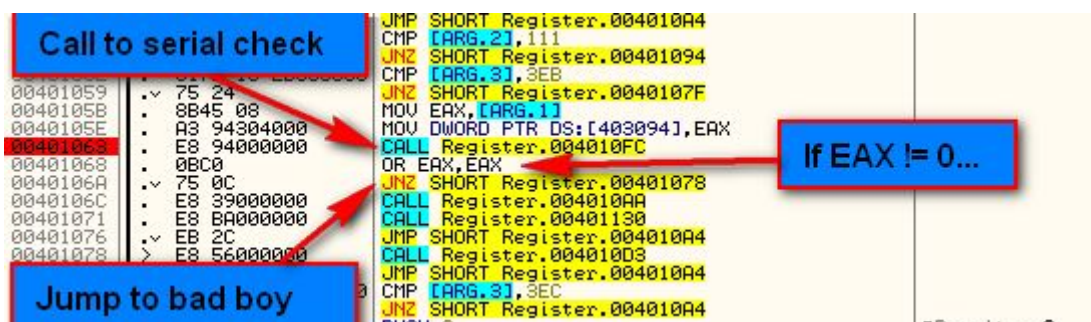
```
00401143 | . 3344 | XCH EHX, EHX |
00401146 | . 8A86 70304000 | MOV AL, BYTE PTR DS:[ESI+403070] |
00401148 | . 83F0 2C | XOR EAX, 2C |
0040114E | . 8886 70304000 | MOV BYTE PTR DS:[ESI+403070], AL |
00401154 | . 46 | INC ESI |
00401155 | ^ E2 EE | LOOPD SHORT Register.00401145 |
00401157 | . 68 70304000 | PUSH Register.00403070 |
0040115C | . 68 ED030000 | PUSH 3ED |
00401161 | . FF35 94304000 | PUSH DWORD PTR DS:[403094] |
00401167 | . C8 20000000 | CALL <JMP.&user32.SetDlgItemTextA> |
0040116D | . CC | RETN |
0040116E | . CC | INT3 |
0040116E | $- FF25 24204000 | JMP DWORD PTR DS:[<&user32.DialogBoxParamA>] | user32.DialogBoxParamA
```

可以看到这个字符串变成了传递给 **SetDlgItemTextA** 的值, 事实上用之前在那里的坏消息替换了已注册的好消息:

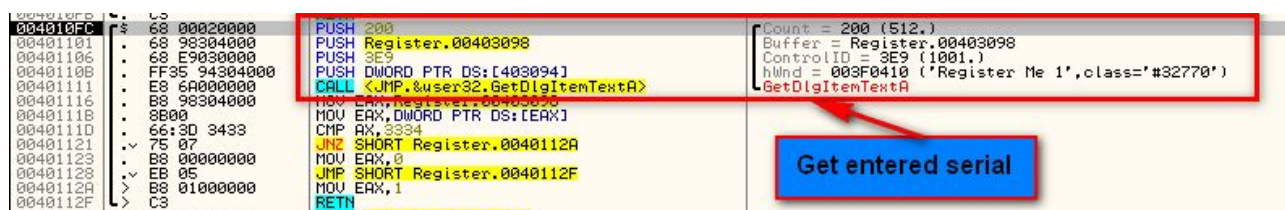
00401154	46	EE	LOORD SHORT Register, 00401145	
00401155	68	70304000	PUSH Register, 00403070	
00401157	68	ED030000	PUSH 3ED	
0040115C	68	FF35 94304000	PUSH DWORD PTR DS:[403094]	
00401161	68	20000000	CALL <user32.SetDlgItemTextA>	Text = "This program is not registered!" ControlID = 3ED (1005.) hwnd = 000030B2 ('Register Me 1', class='32770') SetDlgItemTextA
00401166	C3		RETN	
00401168	4E		INT3	
0040116E	55	FF25 24204000	JMP DWORD PTR DS:[<user32.ShowDialogParamA>]	user32.ShowDialogParamA



所以，现在我们知道，给该应用打补丁的巧妙的方法是进入到序列号检测子程序，确保它总是返回正确的值，因为它不只是在第一次检测时被调用，而且在显示成功后再次被调用。再提醒你一下，序列号检测的相关 **CALL** 被调用，然对 **eax** 进行 0 测试。如果不是 0，就跳到坏消息，所以我们想让子程序返回 0！然后，序列号检测子程序再次被调用，如果它再次返回 0，那么我们的第二次检测就通过了：



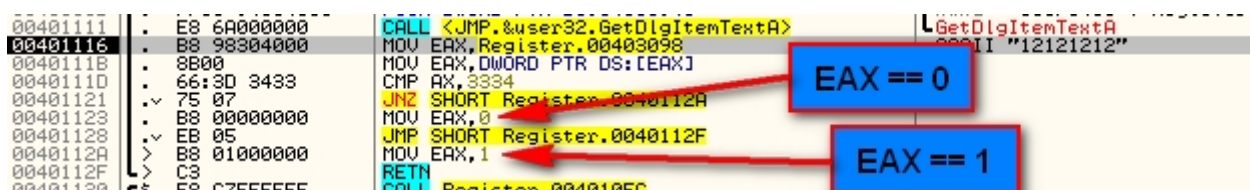
那么，咱们去序列号检测子程序那，看看能对它做些什么。子程序的开始调用了 **GetDlgItemTextA**，我们猜它就是获取我们输入的序列号。你可以在 401101（它指向的是放置文本的 **buffer**）的参数上右键，在数据窗口中跟随它：



我们单步步过 **GetDlgItemTextA** 指令后，就能在 **buffer** 中看到我们的序列号了：

Address	Hex dump	ASCII
00403098	31 32 31 32 00 00 00 00 00 00 00 00 00 00 00 00	12121212.....
004030A8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004030B8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004030C8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004030D8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004030E8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004030F8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00403108	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00403118	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00403128	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00403138	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

在它被保存到 **buffer** 后，该 **buffer** 的起始地址被拷贝到 **EAX** 中，随后该地址中的内容被拷贝到 **EAX** 中。就是将我们密码的前四个字节拷贝到 **EAX** 中。然后这几个字节与 3334 进行比较，如果不匹配，**EAX** 就被填充为 1（坏消息），否则就填充为 0（好消息）：



我们可以看到，做主要决定的是 401121 的 **JNZ** 指令：



00401118	8B00	MOV EAX,DWORD PTR DS:[EAX]
0040111D	66:3D 3433	CMP AX,3334
00401121	75 07	JNZ SHORT Register.0040112A
00401123	B8 00000000	MOV EAX,0
00401128	EB 05	JMP SHORT Register.0040112F
0040112A	B8 01000000	MOV EAX,1
0040112F	C3	RETN

这一行决定了在返回前，EAX 到底是 0 还是 1。所以我们要做的就是保证 EAX 总是等于 0:

004010EB	68 ED030000	PUSH 3ED	ControlID = 3ED (1005.)
004010F0	FF35 94304000	PUSH DWORD PTR DS:[403094]	hWnd = 003B0D44 ('Register Me 1',class='#32770')
004010F6	E8 91000000	CALL <JMP.&user32.SetDlgItemTextA>	
004010FB	C3	RETN	
004010FC	68 00020000	PUSH 200	
00401101	68 98304000	PUSH Register.00403098	
00401106	68 E9300000	PUSH 3E9	
0040110B	FF35 94304000	PUSH DWORD PTR DS:[403094]	
00401111	68 6A000000	CALL <JMP.&user32.GetDlgItemTextA>	
00401116	B8 98304000	MOV EAX,Register.00403098	
0040111B	8B00	MOV EAX,DWORD PTR DS:[EAX]	
0040111D	66:3D 3433	CMP AX,3334	
00401121	90	NOP	
00401122	90	NOP	
00401123	B8 00000000	MOV EAX,0	
00401128	EB 05	JMP SHORT Register.0040112F	
0040112A	B8 01000000	MOV EAX,1	
0040112F	C3	RETN	
00401130	E8 C7FFFFFF	CALL Register.004010FC	
00401135	0BC0	OR EAX,EAX	

Assemble at 00401122

☒ Fill with NOP's

Assemble Cancel

所以现在，代码将总是直接给 EAX 赋 0 值，然后直接跳转到返回处。运行下程序看看：



注意在对序列号检测子程序调用后，我们自然而然的就跳转到好消息那了：

0040112F	C3	RETN	
00401130	E8 C7FFFFFF	CALL Register.004010FC	
00401135	0BC0	OR EAX,EAX	
00401137	74 33	JE SHORT Register.0040116C	We now jump!!!
00401139	B9 1F000000	MOV ECX,1F	
0040113E	BE 00000000	MOV ESI,0	
00401143	33C0	XOR EAX,EAX	
00401145	8A86 70304000	MOV AL,BYTE PTR DS:[ESI+403070]	
00401148	83F0 2C	XOR EAX,2C	
0040114E	8886 70304000	MOV BYTE PTR DS:[ESI+403070],AL	
00401154	46	INC ESI	
00401155	E2 EE	LOOPD SHORT Register.00401145	
00401157	68 70304000	PUSH Register.00403070	
0040115C	68 ED030000	PUSH 3ED	
00401161	FF35 94304000	PUSH DWORD PTR DS:[403094]	
00401167	E8 20000000	CALL <JMP.&user32.SetDlgItemTextA>	
0040116C	C3	RETN	
0040116D	CC	INT3	

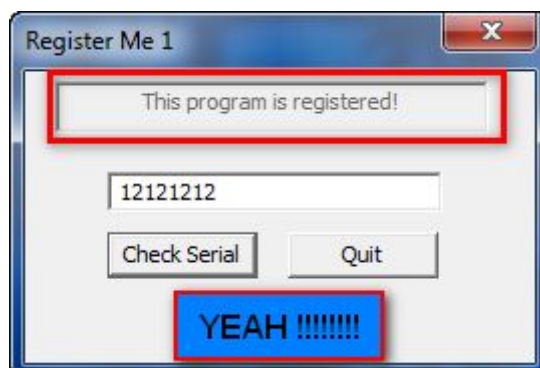
Text = "x0E\x0C\^CK^MA\x0CE\x0CBCX\x0C^IKE\_XI^IH\r"

ControlID = 3ED (1005.)

hWnd = 003B0D44 ('Register Me 1',class='#32770')

SetDlgItemTextA

在第二个检测点，我们也跳到了好消息那：



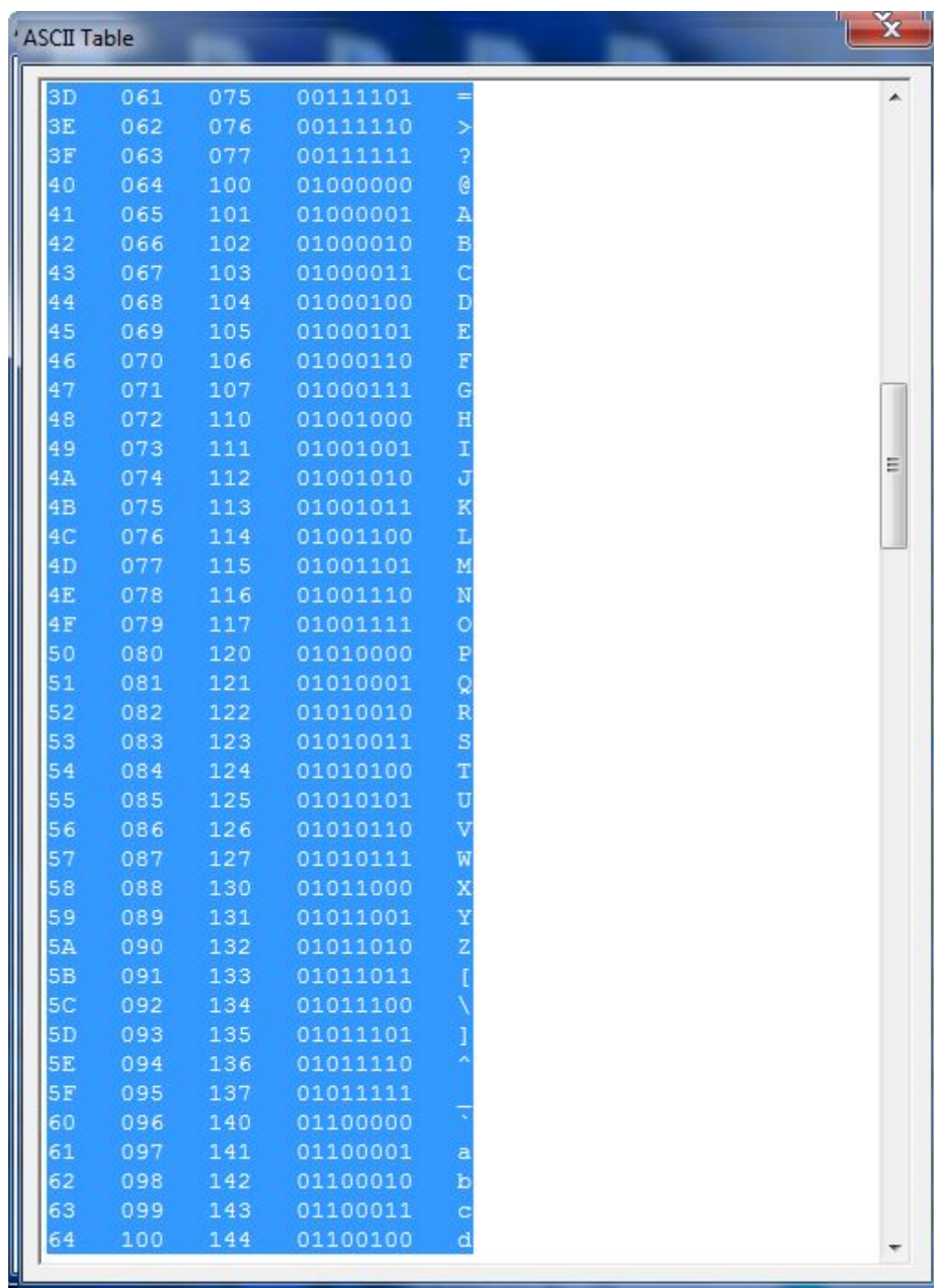


我们现在已经找到了一个注册该程序的补丁，无论你输入什么序列号都行

祝贺你！

#### 四、ASCII 码表插件

你需要做的一件事是找出密码是什么（或对密码有什么样的要求）。给你些帮助，下载并安装“Ascii Table”插件，将其拷贝到插件目录。重启 Olly 后，选择“plugin” -> “Ascii table”就会显示一个表格。尽管它还有很多地方需要改进，不过它能让你快速查询 ASCII 值：



3D	061	075	001111101	=
3E	062	076	001111110	>
3F	063	077	001111111	?
40	064	100	010000000	@
41	065	101	010000001	A
42	066	102	010000010	B
43	067	103	010000011	C
44	068	104	010000100	D
45	069	105	010000101	E
46	070	106	010000110	F
47	071	107	010000111	G
48	072	110	010010000	H
49	073	111	010010001	I
4A	074	112	010010010	J
4B	075	113	010010011	K
4C	076	114	010010100	L
4D	077	115	010010101	M
4E	078	116	010010110	N
4F	079	117	010010111	O
50	080	120	010100000	P
51	081	121	010100001	Q
52	082	122	010100010	R
53	083	123	010100011	S
54	084	124	010100100	T
55	085	125	010100101	U
56	086	126	010100110	V
57	087	127	010100111	W
58	088	130	010110000	X
59	089	131	010110001	Y
5A	090	132	010110010	Z
5B	091	133	010110011	[
5C	092	134	010110100	\
5D	093	135	010110101	]
5E	094	136	010110110	^
5F	095	137	010110111	_
60	096	140	011000000	ˆ
61	097	141	011000001	a
62	098	142	011000010	b
63	099	143	011000011	c
64	100	144	011000100	d



\*\*\*如果有人想要主动更新或重做这个插件，我将永远感激。第一，那些文本不应该被选中，也不应该可编辑（我为什么要编辑 **ASCII** 码表？）。第二，让窗口大小可变真是件好事。如果有人做了，请告诉我，我欠你一辈子。\*\*\*