

## 第八章：参考引用框架

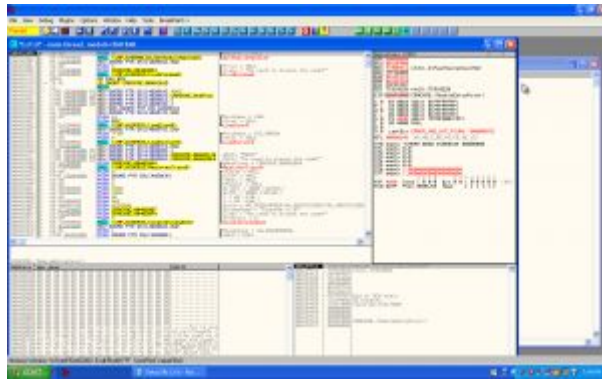
### 一、简介

我们现在要研究的 crackme，相比来说更具挑战性。它就是 Crackme3.exe。咱们也会学习几个新技巧。

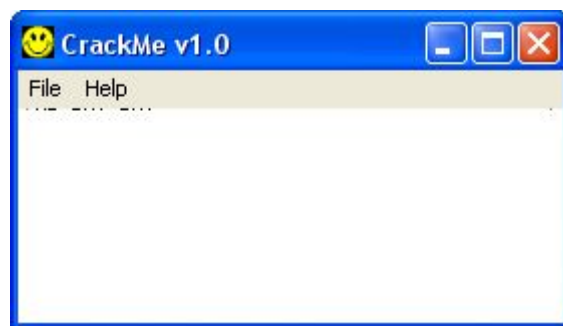
你可以在[教程](#)页下载相关文件以及本文的 PDF 格式版本。

### 二、探究二进制文件

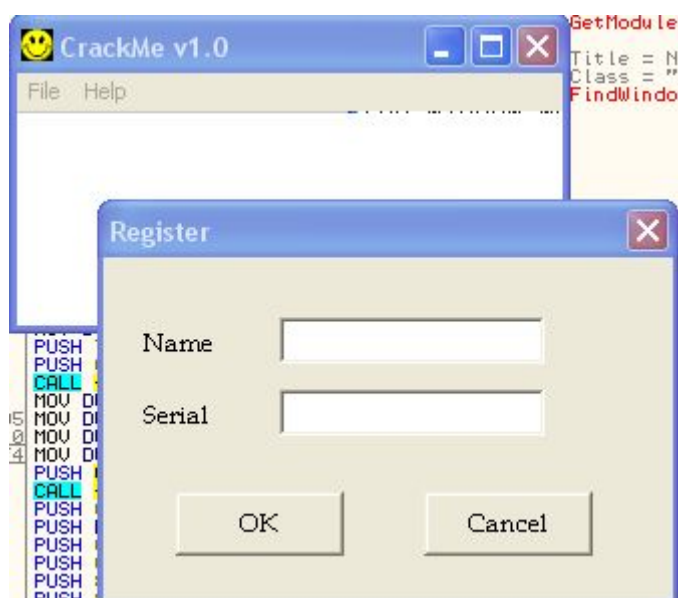
启动 Ollymp 并载入 crackme。它会载入、分析并暂停在第一行：(p1)



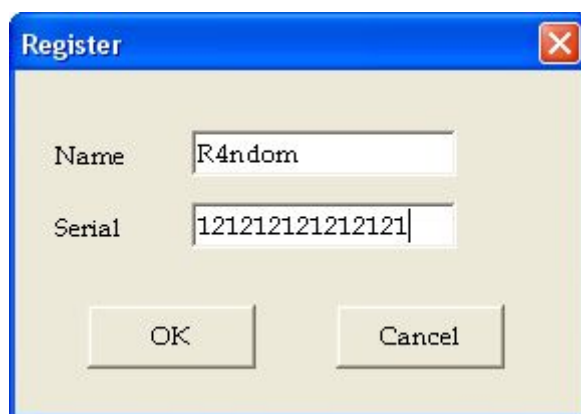
运行下程序看看什么样：(p2)



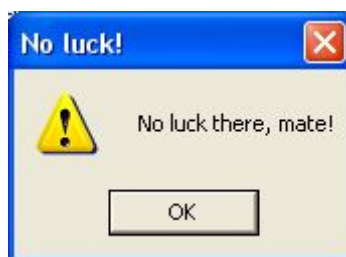
好吧，没啥东西。选择 “Help” ->” Register”：(p3)



现在咱们来到了某个地方。奇怪了，怎么和我们的 FAKE 那么像😅。试着输入用户名和序列号看看程序有什么反应：(p4)



嗯。这回弹出了一个显示坏消息的对话框：(p5)



有时候，对于一个比较小的程序，我喜欢向下多翻几页看看有没有什么有意思的东西。我向下翻了大概 6 页，然后我看到了一些相当有趣的东西：(p6)

```
0040130A C8 000000 ENTER 0,0
0040130E 53 PUSH EBX
0040130F 56 PUSH ESI
00401310 57 PUSH EDI
00401311 817D 0C 11010000 CMP [ARG.2],111
00401318 74 12 JE SHORT CRACKME.0040132C
0040131A 837D 0C 10 CMP [ARG.2],10
0040131E 74 15 JE SHORT CRACKME.00401335
00401320 B8 00000000 MOV EAX,0
00401325 5F POP EDI
00401326 5E POP ESI
00401327 5B POP EBX
00401328 C9 LEAVE
00401329 C2 1000 RETN 10
0040132C 817D 10 F2030000 CMP [ARG.3],3F2
00401333 75 11 JNB SHORT CRACKME.00401346
00401335 6A 00 PUSH 0
00401337 FF75 08 PUSH [ARG.1]
0040133A E8 73010000 CALL <JMP.&USER32.EndDialog>
0040133F B8 01000000 MOV EAX,1
00401344 EB DF JMP SHORT CRACKME.00401325
00401346 B8 00000000 MOV EAX,0
0040134B EB D8 JMP SHORT CRACKME.00401325
0040134F 6A 30 PUSH 30
00401354 68 29214000 PUSH CRACKME.00402129
0040135A 68 34214000 PUSH CRACKME.00402134
00401359 FF75 08 PUSH [ARG.1]
0040135C E8 D9000000 CALL <JMP.&USER32.MessageBoxA>
00401361 C3 RETN
00401362 6A 00 PUSH 0
00401364 E8 AD000000 CALL <JMP.&USER32.MessageBeep>
00401369 6A 30 PUSH 30
0040136B 68 60214000 PUSH CRACKME.00402160
00401370 68 69214000 PUSH CRACKME.00402169
00401375 FF75 08 PUSH [ARG.1]
00401378 E8 BD000000 CALL <JMP.&USER32.MessageBoxA>
0040137D C3 RETN
0040137E 8B7424 04 MOV ESI,DWORD PTR SS:[ESP+4]
00401382 56 PUSH ESI
00401383 8A06 MOV AL,BYTE PTR DS:[ESI]
00401385 84C0 TEST AL,AL
00401387 74 13 JE SHORT CRACKME.0040139C
00401389 3C 41 CMP AL,41
0040138B 72 1F JB SHORT CRACKME.0040139C

ntdll.7C910228

kernel32.7C817077
kernel32.7C817077
kernel32.7C817077

Result = 0
hWnd = 00401000
EndDialog

Style = MB_OK!MB_ICONEXCLAMATION!MB_APPLMODAL
Title = "Good work!"
Text = "Great work, mate!\rNow try the next CrackMe!"
hOwner = 00401000
MessageBoxA

BeepType = MB_OK
MessageBeep
Style = MB_OK!MB_ICONEXCLAMATION!MB_APPLMODAL
Title = "No luck!"
Text = "No luck there, mate!"
hOwner = 00401000
MessageBoxA

ntdll.7C910228
```

看看在 MessageBoxA 函数前面的文本。如果你往 MessageBoxA 函数上面的文本的左边看的话，你会看到一条黑线将函数的参数框起来：(p7)

```
[Style = MB_OK!MB_ICONEXCLAMATION!MB_APPLMODAL
Title = "Good work!"
Text = "Great work, mate!\rNow try the next CrackMe!"
hOwner = 00401000
MessageBoxA
```

01ly 给你显示的是准备传递给函数的参数，就是被调用的那个函数的。本例中个，参数 1 是窗口的类型，参数 2 是窗口的标题 (“Good work!”)，参数 3 是窗口显示的文本 (“Great work...”)，参数 4 是窗口所有者的句柄。最后，MessageBoxA 函数被调用。你可以在 MessageBoxA 上右键，选择 “Help on symbolic names” 来查看传递给函数的参数以及返回值。

现在，我们对比着看下紧随其后的那部分：(p8)

```
[Style = MB_OK!MB_ICONEXCLAMATION!MB_APPLMODAL
Title = "No luck!"
Text = "No luck there, mate!"
hOwner = 00401000
MessageBoxA
```

对这两个函数的调用有很大的不同。一个看起来真的不错，而另一个却不是。我想我们大家都承认，我们宁愿要第一个调用。现在咱们要记住

## R4ndom' s Essential Truths About Reversing Data #2: R4ndom 关于逆向数据的必备真言 2:

**2. 大部分的保护机制都可以被绕过，通过修改一个简单的跳转指令来跳转到“好的”代码处，而不是“坏的”代码处（或者避免跳转跳过“好的”代码）。**

如果你看两个函数的上面几行，你会看到几个 jmp 语句，它们决定了你将走哪条路，好的路或者坏的路。99%的应用里的 99%的时间都是这样。窍门就是

找到这个跳转。（当然剩下的 1%要难得多，不过我们不会接触。）我们的例子中，在 401344 和 40134B 有跳转。现在，作为一个已经训练过的逆向工程师，这些跳转很快就被略过（如果你想知道为啥，是因为它们和我们的消息框在不同的函数中，所以它们不会跳过我们的坏消息或跳转至好消息处，后面会讨论这个）。咱们来研究研究它们：（p9）

00401339	E8 73010000	CALL <JMP.&USER32.EndDialog>	EndDialog
0040133F	B8 01000000	MOV EAX,1	
00401344	EB DF	JMP SHORT CRACKME.00401325	
00401346	B8 00000000	MOV EAX,0	
0040134B	EB D8	JMP SHORT CRACKME.00401325	
0040134D	6A 30	PUSH 30	Style = MB_OK!MB_ICONEXCLAMATION!MB_APPLMODAL
0040134F	68 29214000	PUSH CRACKME.00402129	Title = "Good work!"
00401354	68 34214000	PUSH CRACKME.00402134	Text = "Great work, mate!\rNow try the next CrackMe!"
00401359	FF75 08	PUSH [ARG.1]	hOwner = 00401000
0040135C	E8 09000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
00401361	C3	RETN	

首先，点一下 40134B 处的 JMP 指令。会看到一个红线指示该 JMP 将跳到哪，我们看到它走的是一条错误的路！（p10）

0040134E	74 15	JE SHORT CRACKME.00401350	
00401350	B8 00000000	MOV EAX,0	
00401355	5F	ROP F07	kernel32.7C817077 kernel32.7C817077 kernel32.7C817077
0040135E	5E		
00401367	5B		
00401368	C9		
00401369	C2 1000	CMPL [ARG.3],3F2	
0040136C	74 11	JNE SHORT CRACKME.00401346	
0040136E	6A 00	PUSH 0	Result = 0
0040136F	FF75 08	PUSH [ARG.1]	hWnd = 00401000
00401370	E8 73010000	CALL <JMP.&USER32.EndDialog>	EndDialog
0040137F	B8 01000000	MOV EAX,1	
00401384	EB DF	JMP SHORT CRACKME.00401325	
00401386	B8 00000000	MOV EAX,0	
0040138B	EB D8	JMP SHORT CRACKME.00401325	
0040134D	6A 30	PUSH 30	Style = MB_OK!MB_ICONEXCLAMATION!MB_APPLMODAL
0040134F	68 29214000	PUSH CRACKME.00402129	Title = "Good work!"
00401354	68 34214000	PUSH CRACKME.00402134	Text = "Great work, mate!\rNow try the next CrackMe!"
00401359	FF75 08	PUSH [ARG.1]	hOwner = 00401000
0040135C	E8 09000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
00401361	C3	RETN	
00401362	6A 00	PUSH 0	BeepType = MB_OK
00401364	E8 AD000000	CALL <JMP.&USER32.MessageBeep>	MessageBeep
00401369	6A 30	PUSH 30	Style = MB_OK!MB_ICONEXCLAMATION!MB_APPLMODAL
0040136B	68 60214000	PUSH CRACKME.00402160	Title = "No luck!"
00401370	68 69214000	PUSH CRACKME.00402169	Text = "No luck there, mate!"
00401375	FF75 08	PUSH [ARG.1]	hOwner = 00401000
00401378	E8 BD000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
0040137D	C3	RETN	
0040137E	8B7424 04	MOV ESI,DWORD PTR SS:[ESP+4]	ntdll.7C910228
00401382	56	PUSH ESI	
00401383	20AC	CMPL DI, BYTE PTR DS:[ESI]	

它没有跳到我们的好消息那，也没有跳过坏消息，反而往上跳到前面的代码了。我们试试 401344 那个 JMP。这个事实上和那个指向的一样（仍然是错误的路），所以看起来我们的第一个猜测是错的。

顺便说一下，就像我早些说的，老鸟忽视这些跳转的原因是因为 01ly 显示函数的方式。如果你注意看第一列（地址）和第二列（操作码）之间的话，会看到一些粗黑线。这些线是 01ly 放进去的，用来区分函数（有时候 01ly 无法指出函数的起始点和结束点，所以就不会有这些线）：（p11）



```
00401300  B8 01000000 MOV EAX,1
00401305  E9 7AFFFFFF JMP CRACKME.00401284
0040130A  5E ENTER 0,0
0040130E  5F PUSH EBX
00401310  5E PUSH ESI
00401311  5F PUSH EDI
00401312  817D 0C 11010000 CMP [ARG.2],111
00401318  74 12 JE SHORT CRACKME.0040132C
0040131A  837D 0C 10 CMP [ARG.2],10
0040131E  74 15 JE SHORT CRACKME.00401335
00401320  5F 00000000 MOV EAX,0
00401326  5F POP ESI
00401327  5B POP EBX
00401328  C9 LEAVE
00401329  C2 1000 RETN 10
0040132C  817D 10 F2030000 CMP [ARG.3],3F2
00401333  75 11 JNZ SHORT CRACKME.00401346
00401335  6A 00 PUSH 0
00401337  FF75 08 PUSH [ARG.1]
00401339  E8 73010000 CALL <JMP.&USER32.EndDialog>
0040133F  E8 01000000 MOV EAX,1
00401344  EB DF JMP SHORT CRACKME.00401325
00401346  B8 00000000 MOV EAX,0
00401348  EB D8 JMP SHORT CRACKME.00401325
0040134D  6A 30 PUSH 30
0040134F  68 29214000 PUSH CRACKME.00402129
00401354  68 34214000 PUSH CRACKME.00402134
00401359  FF75 08 PUSH [ARG.1]
0040135C  E8 D9000000 CALL <JMP.&USER32.MessageBoxA>
00401361  C3 RETN
00401362  6A 00 PUSH 0
00401364  E8 AD000000 CALL <JMP.&USER32.MessageBeep>
00401369  6A 30 PUSH 30
0040136B  68 60214000 PUSH CRACKME.00402160
00401370  68 63214000 PUSH CRACKME.00402169
00401375  FF75 08 PUSH [ARG.1]
00401378  E8 BD000000 CALL <JMP.&USER32.MessageBoxA>
0040137D  C3 RETN
0040137E  8B7424 04 MOV ESI,DWORD PTR SS:[ESP+4]
00401382  56 PUSH ESI
00401383  56 PUSH ESI
00401384  56 PUSH ESI
00401385  56 PUSH ESI
00401386  56 PUSH ESI
00401387  56 PUSH ESI
00401388  56 PUSH ESI
00401389  56 PUSH ESI
0040138A  56 PUSH ESI
0040138B  56 PUSH ESI
0040138C  56 PUSH ESI
0040138D  56 PUSH ESI
0040138E  56 PUSH ESI
0040138F  56 PUSH ESI
00401390  56 PUSH ESI
00401391  56 PUSH ESI
00401392  56 PUSH ESI
00401393  56 PUSH ESI
00401394  56 PUSH ESI
00401395  56 PUSH ESI
00401396  56 PUSH ESI
00401397  56 PUSH ESI
00401398  56 PUSH ESI
00401399  56 PUSH ESI
```

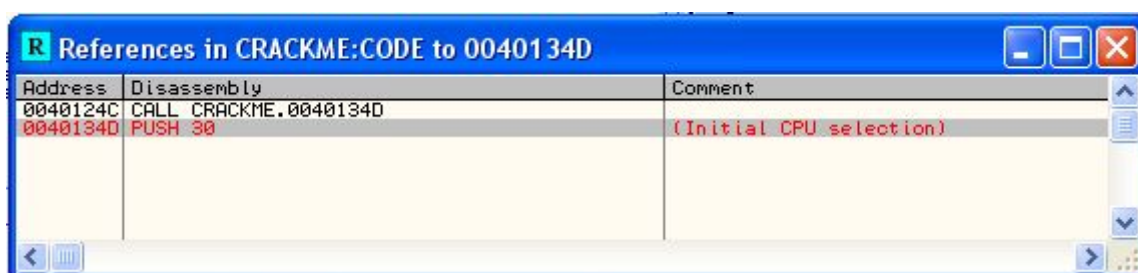
本例中，你可以看到那两个 JMP 是在我们的好消息和坏消息的上面的函数中。因为它不会跳转到好消息或者坏消息处，它们真的对我们没有任何帮助。这也告诉你另一件事，第一个消息框（好消息那个）和坏消息框不是在同一个函数中。这些都告诉我们，这些函数都是在别的地方被调用的，并且在它们被调用之前的某处决定了哪个被调用，是好的还是坏的。咱们看看怎么才能绕过这些干扰.....

### 三、查找参考

在好消息函数的第一行，也就是 40134D 那行上右键。选择 “Find References To” ->” Selected Command”（或者按 Ctrl+R）：（p12）

```
00401320  B8 00000000 MOV EAX,0
00401325  5F POP EDI
00401326  5E POP ESI
00401327  5B POP EBX
00401328  C9 LEAVE
00401329  C2 1000 RETN 10
0040132C  817D 10 F2030000 CMP [ARG.3],3F2
00401333  75 11 JNZ SHORT CRACKME.00401346
00401335  6A 00 PUSH 0
00401337  FF75 08 PUSH [ARG.1]
00401339  E8 73010000 CALL <JMP.&USER32.EndDialog>
0040133F  E8 01000000 MOV EAX,1
00401344  EB DF JMP SHORT CRACKME.00401325
00401346  B8 00000000 MOV EAX,0
00401348  EB D8 JMP SHORT CRACKME.00401325
0040134D  6A 30 PUSH 30
0040134F  68 29214000 PUSH CRACKME.00402129
00401354  68 34214000 PUSH CRACKME.00402134
00401359  FF75 08 PUSH [ARG.1]
0040135C  E8 D9000000 CALL <JMP.&USER32.MessageBoxA>
00401361  C3 RETN
00401362  6A 00 PUSH 0
00401364  E8 AD000000 CALL <JMP.&USER32.MessageBeep>
00401369  6A 30 PUSH 30
0040136B  68 60214000 PUSH CRACKME.00402160
00401370  68 63214000 PUSH CRACKME.00402169
00401375  FF75 08 PUSH [ARG.1]
00401378  E8 BD000000 CALL <JMP.&USER32.MessageBoxA>
0040137D  C3 RETN
0040137E  8B7424 04 MOV ESI,DWORD PTR SS:[ESP+4]
00401382  56 PUSH ESI
00401383  56 PUSH ESI
00401384  56 PUSH ESI
00401385  56 PUSH ESI
00401386  56 PUSH ESI
00401387  56 PUSH ESI
00401388  56 PUSH ESI
00401389  56 PUSH ESI
0040138A  56 PUSH ESI
0040138B  56 PUSH ESI
0040138C  56 PUSH ESI
0040138D  56 PUSH ESI
0040138E  56 PUSH ESI
0040138F  56 PUSH ESI
00401390  56 PUSH ESI
00401391  56 PUSH ESI
00401392  56 PUSH ESI
00401393  56 PUSH ESI
00401394  56 PUSH ESI
00401395  56 PUSH ESI
00401396  56 PUSH ESI
00401397  56 PUSH ESI
00401398  56 PUSH ESI
00401399  56 PUSH ESI
```

弹出“References(参考)”窗口: (p13)



该窗口显示的是 Ollly 能够找到的 CALL 或 JMP 到\*这个\*地址的所有参考 (CALL 和 JMP)。现在, 双击列表中的第一个 (就是那个不是红色的), 然后你就会来到调用这个 (好的) 消息的那行: (p14)

00401243	. 74 07	JE SHORT CRACKME.0040124C	
00401245	. E8 18010000	CALL CRACKME.00401362	
0040124A	. EB 9A	JMP SHORT CRACKME.004011E6	
0040124C	> E8 FC000000	CALL CRACKME.0040134D	
00401251	. EB 93	JMP SHORT CRACKME.004011E6	
00401253	. C8 000000	ENTER 0,0	

在 40124C 那行你可以看到指令 CALL CRACKME.0040134D。40134D 就是好消息对话框的第一行。咱们在这里设置一个断点: (p15)

00401243	. 74 07	JE SHORT CRACKME.0040124C	
00401245	. E8 18010000	CALL CRACKME.00401362	
0040124A	. EB 9A	JMP SHORT CRACKME.004011E6	
0040124C	> E8 FC000000	CALL CRACKME.0040134D	
00401251	. EB 93	JMP SHORT CRACKME.004011E6	
00401253	. C8 000000	ENTER 0,0	

现在, 咱们对另一个函数做相同的操作, 也就是坏消息那个。转到 401362 那行, 就是坏消息函数的第一行, 右键选择“Find References To”->“Selection (or ctrl-R)”。这会再一次调出参考窗口。双击第一条, 我们就会来到调用坏消息的地方: (p16)

00401238	. E8 9B010000	CALL CRACKME.004013D8	
0040123D	. 83C4 04	ADD ESP,4	
00401240	. 58	POP EAX	
00401241	. 3BC3	CMP EAX,EBX	
00401243	. 74 07	JE SHORT CRACKME.0040124C	
00401245	. E8 18010000	CALL CRACKME.00401362	
0040124A	. EB 9A	JMP SHORT CRACKME.004011E6	
0040124C	> E8 FC000000	CALL CRACKME.0040134D	
00401251	. EB 93	JMP SHORT CRACKME.004011E6	
00401253	. C8 000000	ENTER 0,0	
00401257	. 53	PUSH EBX	

有意思的是, 它就在我们刚才设置的断点的上面 2 行! 咱们在这里也设置一个断点: (p17)

0040123D	. 83C4 04	ADD ESP,4	
00401240	. 58	POP EAX	
00401241	. 3BC3	CMP EAX,EBX	
00401243	. 74 07	JE SHORT CRACKME.0040124C	
00401245	. E8 18010000	CALL CRACKME.00401362	
0040124A	. EB 9A	JMP SHORT CRACKME.004011E6	
0040124C	> E8 FC000000	CALL CRACKME.0040134D	
00401251	. EB 93	JMP SHORT CRACKME.004011E6	
00401253	. C8 000000	ENTER 0,0	
00401257	. 53	PUSH EBX	
00401258	. 56	PUSH ESI	

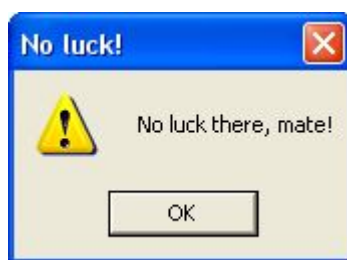
\*\*\*注意，有时候你选中一行然后查找参考，但是一个都没有。导致这个结果的原因有两种：1) 你选择了错误的函数“入口点”，也就是调用这个函数应该 *call* 或 *jump* 其他的地方，但是它们却调用了别的行，有可能就是你选择行的前面或后面那行。选择正确的行来查找参考需要花时间和技巧，不过要坚持下去。2) 代码中没有明显指向这一行的指令。记住，程序运行时有许多数字被动态的操纵，*call* 或 *jump* 指向的地址也不例外。所以，如果 *call* 的地址是动态创建的话，所以 *Olly* 就没有办法提前知道这行会被调用，所以 *Olly* 也就不会将这个参考列出来。关于这个也是有方法的，不过这会我不打算讨论。

现在，如果我们看看这两个 CALL 的附件的话，会看到几个 *jmp* 指令。第一个，401243 的 JE SHORT CRACKME.0040124C。当然，你知道 JE 是啥意思，因为你已经读过汇编语言的书（参见 R. E. T. A. R. D. 规则#1），不过为了证实，我们假定你不知道这个特别的助记符（指令）是啥意思。这就是插件 MnemonicHelp 存在的原因。右键 JE 指令，在上下文菜单中选择“? JE”：(p18)

Intel x86 Instructions		
File Edit Bookmark Options Help		
Contents Index Back Print		
<b>Jcc—Jump if Condition Is Met</b>		
<i>See also</i>		
Opcode	Instruction	Description
77 cb	JA <i>rel8</i>	Jump short if above (CF=0 and ZF=0)
73 cb	JAE <i>rel8</i>	Jump short if above or equal (CF=0)
72 cb	JB <i>rel8</i>	Jump short if below (CF=1)
76 cb	JBE <i>rel8</i>	Jump short if below or equal (CF=1 or ZF=1)
72 cb	JC <i>rel8</i>	Jump short if carry (CF=1)
E3 cb	JCXZ <i>rel8</i>	Jump short if CX register is 0
E3 cb	JECXZ <i>rel8</i>	Jump short if ECX register is 0
74 cb	JE <i>rel8</i>	Jump short if equal (ZF=1)
7F cb	JG <i>rel8</i>	Jump short if greater (ZF=0 and SF=OF)
7D cb	JGE <i>rel8</i>	Jump short if greater or equal (SF=OF)
7C cb	JL <i>rel8</i>	Jump short if less (SF<>OF)
7E cb	JLE <i>rel8</i>	Jump short if less or equal (ZF=1 or SF<>OF)
76 cb	JNA <i>rel8</i>	Jump short if not above (CF=1 or ZF=1)
72 cb	JNAE <i>rel8</i>	Jump short if not above or equal (CF=1)
73 cb	JNB <i>rel8</i>	Jump short if not below (CF=0)
77 cb	JNBE <i>rel8</i>	Jump short if not below or equal (CF=0 and ZF=0)
73 cb	JNC <i>rel8</i>	Jump short if not carry (CF=0)
75 cb	JNE <i>rel8</i>	Jump short if not equal (ZF=0)
7E cb	JNG <i>rel8</i>	Jump short if not greater (ZF=1 or SF<>OF)
7C cb	JNGE <i>rel8</i>	Jump short if not greater or equal (SF<>OF)
7D cb	JNL <i>rel8</i>	Jump short if not less (SF=OF)
7F cb	JNLE <i>rel8</i>	Jump short if not less or equal (ZF=0 and SF=OF)
71 cb	JNO <i>rel8</i>	Jump short if not overflow (OF=0)
7B cb	JNP <i>rel8</i>	Jump short if not parity (PF=0)
79 cb	JNS <i>rel8</i>	Jump short if not sign (SF=0)



这个窗口比较长，因为有大量的跳转指令。如果我们向下看那个“JE”的话，会看到它是“Jump if Equal (ZF = 1)”。意思是如果 0 标志位被置 1 就跳转（或者被比较的两个项目相等）。前面的教程中我们复习过标志位，所以你应该知道，如果被比较的两个对象相等，JE 就会跳转。我们也能够发现，这个 JE 跳过了对坏消息的调用，并且紧随跳转的那条指令是对好消息的调用。如果 JE 没有跳，我们就会调用坏消息。所以，我们想要这个跳转实现，以便我们能够调用好消息。咱们操作下看看。在 JE 指令上设置一个断点，重启（或运行）应用。点击 crackme 中的“Help”->”Register”，输入用户名和序列号，然后点 OK: (p19)



哇！等等！显示了坏消息，并且 Olly 也没有断下来？也就是说 Olly 永远也不会运行到我们的断点！这是咋回事呢。

事实上，这个在逆向工程领域里是可以用得着的😄。我向你保证，每一个专家级逆向工程师/破解者这时候都会想“我错过什么了吗？一个 0xcc 中断？IsDebuggerPresent（译者注：一个 Windows API）？NTFlags？TLS 回调？”，然后白费力气去寻找一些过于复杂的解决方案。但是我们只是初学者，我们只有几个工具可以使用，其中一个就是搜索字符串，那就试试这个吧：(p20)

R Text strings referenced in Crackme3:CODE		
Address	Disassembly	Text string
00401000	Crackme3.004020F4	(Initial CPU selection)
0040100E	PUSH Crackme3.004020F4	ASCII "No need to disasm the code!"
00401077	MOV DWORD PTR DS:[402084],Crackme3.004020F4	ASCII "MENU"
00401081	MOV DWORD PTR DS:[402088],Crackme3.004020F4	ASCII "No need to disasm the code!"
00401087	PUSH Crackme3.004020E7	ASCII "CrackMe v1.0"
0040108C	PUSH Crackme3.004020F4	ASCII "No need to disasm the code!"
004011F7	PUSH Crackme3.0040211F	ASCII "DLG_ABOUT"
00401213	PUSH Crackme3.00402115	ASCII "DLG_REGIS"
00401228	PUSH Crackme3.0040218E	ASCII "R4ndom"
00401233	PUSH Crackme3.0040217E	ASCII "1212121212"
004012B7	PUSH Crackme3.0040218E	ASCII "R4ndom"
004012D7	PUSH Crackme3.0040217E	ASCII "1212121212"
0040134F	PUSH Crackme3.00402129	ASCII "Good work!"
00401354	PUSH Crackme3.00402134	ASCII "Great work, mate!\r\nNow try the next CrackMe!"
0040136B	PUSH Crackme3.00402160	ASCII "No luck!"
00401370	PUSH Crackme3.00402169	ASCII "No luck there, mate!"
004013AF	PUSH Crackme3.00402160	ASCII "No luck!"
004013B4	PUSH Crackme3.00402169	ASCII "No luck there, mate!"

现在，你可以看到一些相当有趣的东西…。有两个“No luck!”坏消息，但是只有一个好消息。也就是说，代码中的某个地方做了检查，如果没有通过就会显示坏消息。这是一个在反逆向工程中非常流行的技术：找一个非常明显的地方放好的/坏的消息，然后添加一个不是那么明显的检测。如果你看看代码



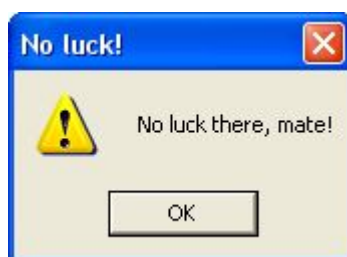
窗口我们的好消息和坏消息所在的位置，你会发现字符串 “No luck!” 是在 40136B，所以我们知道那不是我们要找的字符串。所以咱们双击下 4013AF 那个：(p21)

0040137D	C3	RETN
0040137E	8B7424 04	MOV ESI,DWORD PTR SS:[ESP+4]
00401382	56	PUSH ESI
00401383	8A06	MOV AL,BYTE PTR DS:[ESI]
00401385	84C0	TEST AL,AL
00401387	74 13	JE SHORT Crackme3.0040139C
00401389	3C 41	CMP AL,41
0040138B	72 1F	JB SHORT Crackme3.004013AC
0040138D	3C 5A	CMP AL,5A
0040138F	73 03	JNB SHORT Crackme3.00401394
00401391	46	INC ESI
00401392	EB EF	JMP SHORT Crackme3.00401383
00401394	E8 39000000	CALL Crackme3.004013D2
00401399	46	INC ESI
0040139A	EB E7	JMP SHORT Crackme3.00401383
0040139C	5E	POP ESI
0040139D	E8 20000000	CALL Crackme3.004013C2
004013A2	81F7 78560000	XOR EDI,5678
004013A8	8BC7	MOV EAX,EDI
004013AA	EB 15	JMP SHORT Crackme3.004013C1
004013AC	5E	POP ESI
004013AD	6A 30	PUSH 30
004013AF	68 60214000	PUSH Crackme3.00402160
004013B4	68 60214000	PUSH Crackme3.00402169
004013B9	FF75 08	PUSH [ARG_1]
004013BC	E8 79000000	CALL <JMP.&USER32.MessageBoxA>
004013C1	C3	RETN
004013C2	33FF	XOR EDI,EDI
004013C4	33DB	XOR EBX,EBX
004013C6	8A1E	MOV BL,BYTE PTR DS:[ESI]
004013C8	84DB	TEST BL,BL
004013CA	74 05	JE SHORT Crackme3.004013D1
004013CC	03FB	ADD EDI,EBX
004013CE	46	INC ESI
004013CF	EB F5	JMP SHORT Crackme3.004013C6
004013D1	C3	RETN
004013D2	2C 20	SUB AL,20
004013D4	8B06	MOV BYTE PTR DS:[ESI],AL
004013D6	C3	RETN
004013D7	C3	RETN
004013D8	33C0	XOR EAX,EAX
004013DA	33FF	XOR EDI,EDI
004013DC	33DB	XOR EBX,EBX
004013DE	8B7424 04	MOV ESI,DWORD PTR SS:[ESP+4]

[Style = MB\_OK!MB\_ICONEXCLAMATION!MB\_APPLMODAL  
Title = "No luck!"  
Text = "No luck there, mate!"  
hOwner = 7E418734  
MessageBoxA

这个坏消息是在程序内存中完全不同的区！我认为这个 crackme 是在太简单了！好，咱们深呼吸然后想想 RETARD 规则 #2，找找 比较/跳转。本例中在 4013AA 处有个 JMP，当你点击它的时候，011y 会显示一个箭头刚好跳过了坏消息。看起来前途光明啊…。那就试试吧！在那个 jmp 指令处设置一个断点，重启应用并运行。

\*\*\*你有可能得到错误的消息，就像我们上一章中断点被破坏那样。如果发生了，像上一次那样做就行了。打开 BP 窗口，在你运行程序前重新启用所有的断点：)(p22)



操蛋!!! 好吧，不起作用，所以我猜我们得深入挖掘了。咱们看看代码，试试理解到底是什么个情况（该是你组合阅读大放异彩的时候了😁）：(p23)

00401378	E8 BD000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
0040137D	C3	RETN	
00401382	8B7424 04	MOV ESI,DWORD PTR SS:[ESP+4]	
00401383	56	PUSH ESI	Crackme3.00402188
00401385	8A06	MOV AL,BYTE PTR DS:[ESI]	
00401387	84C0	TEST AL,AL	
00401389	74 13	JE SHORT Crackme3.0040139C	
0040138B	3C 41	CMP AL,41	
0040138D	72 1F	JB SHORT Crackme3.004013AC	
0040138F	3C 5A	CMP AL,5A	
00401391	73 03	JNB SHORT Crackme3.00401394	
00401392	46	INC ESI	Crackme3.00402188
00401394	EB EF	JMP SHORT Crackme3.00401383	
00401396	E8 39000000	CALL Crackme3.004013D2	Crackme3.00402188
00401398	46	INC ESI	Crackme3.0040218E
0040139A	EB E7	JMP SHORT Crackme3.00401383	
0040139C	5E	POP ESI	
0040139E	E8 20000000	CALL Crackme3.004013C2	
004013A0	81F7 78560000	XOR EDI,5678	
004013A2	8BC7	MOV EAX,EDI	
004013A4	EB 15	JMP SHORT Crackme3.004013C1	Crackme3.0040218E
004013AC	5E	POP ESI	
004013AD	6A 30	PUSH 30	
004013AF	68 60214000	PUSH Crackme3.00402160	Title = "No luck!"
004013B1	68 69214000	PUSH Crackme3.00402169	Text = "No luck there, mate!"
004013B3	FF75 08	PUSH [ARG.1]	hOwner = 014103F4 ('CrackMe v1.0',class='No need
004013B5	E8 79000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
004013B7	C3	RETN	
004013C2	33FF	XOR EAX,EAX	

好，我们知道了一件事，因为教程的前面我们学过，就是函数的开始和结束点。图片中你能通过蓝色箭头看到。所以，从函数的起始点开始，有一个循环首先检查 AL 是不是 0 (TEXT AL, AL)，然后循环将 AL 和一组数值 (41, 5a) 进行比较。期间，有一些依赖于 AL 值得跳转。首先，咱们看看到底哪个跳转会调用我们的坏消息 (有一个 JMP 指令刚好在坏消息前面，没有什么可以“空降”直达它。所以，必须有什么东西跳过那个跳转来运行坏消息代码。最有可能的跳转是在 4013AC)。

点一下 4013AC，也就是坏消息框的第一条指令，右键该行选择 “Find References To” -> “Selected Address”。(我知道一旦你点了这行，就会显示一个红色箭头，显示了哪条指令调用了它，但是我们怎么才能知道就没有别的指令调用坏消息呢。找到所有的参考可以帮助我们确定，有可能只有一个。) 然后我们就再次看到了参考窗口：(p24)

References in Crackme3:CODE to 004013AC		
Address	Disassembly	Comment
0040138B	JB SHORT Crackme3.004013AC	
004013AC	POP ESI	(Initial CPU selection)

现在，双击第一个，咱们来看看哪一行正在调用坏消息：(p25)

0040137E	8B7424 04	MOV ESI,DWORD PTR SS:[ESP+4]	
00401382	56	PUSH ESI	Crackme3.00402188
00401383	8A06	MOV AL,BYTE PTR DS:[ESI]	
00401385	84C0	TEST AL,AL	
00401387	74 13	JE SHORT Crackme3.0040139C	
00401389	3C 41	CMP AL,41	
0040138B	72 1F	JB SHORT Crackme3.004013AC	
0040138D	3C 5A	CMP AL,5A	
0040138F	73 03	JNB SHORT Crackme3.00401394	
00401391	46	INC ESI	Crackme3.00402188
00401392	EB EF	JMP SHORT Crackme3.00401383	
00401394	E8 39000000	CALL Crackme3.004013D2	Crackme3.00402188
00401396	46	INC ESI	Crackme3.0040218E
0040139A	EB E7	JMP SHORT Crackme3.00401383	
0040139C	5E	POP ESI	
0040139E	E8 20000000	CALL Crackme3.004013C2	
004013A0	81F7 78560000	XOR EDI,5678	
004013A2	8BC7	MOV EAX,EDI	
004013A4	EB 15	JMP SHORT Crackme3.004013C1	Crackme3.0040218E
004013AC	5E	POP ESI	
004013AD	6A 30	PUSH 30	
004013AF	68 60214000	PUSH Crackme3.00402160	Title = "No luck!"
004013B1	68 69214000	PUSH Crackme3.00402169	Text = "No luck there, mate!"
004013B3	FF75 08	PUSH [ARG.1]	hOwner = 014103F4 ('CrackMe v1.0',class='No need t
004013B5	E8 79000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
004013B7	C3	RETN	



噢，原来是循环中的一个。注意参考窗口中旁边的那个红色的那行（我们现在可以忽略它），只有一个到该地址的参考，所以我们可以保证 40138B 这行是调用坏消息的唯一代码。所以 40138B 的 JB SHORT 4013AC 就是那个罪魁祸首。咱们试着在它上面设置一个 BP，临时修改下看看能否绕过这个坏消息。在 40138B 设断点，重新运行程序：（p26）

00401383	>	8A06	MOV AL, BYTE PTR DS:[ESI]	
00401385	.	84C0	TEST AL, AL	
00401387	>	74 13	JE SHORT Crackme3.0040139C	
00401389	.	3C 41	CMP AL, 41	
0040138B	>	72 1F	JB SHORT Crackme3.004013AC	
0040138D	.	3C 5A	CMP AL, 5A	
0040138F	>	73 03	JNB SHORT Crackme3.00401394	
00401391	.	46	INC ESI	Crackme3.0040218E
00401392	>	EB EF	JMP SHORT Crackme3.00401383	
00401394	.	E8 39000000	CALL Crackme3.004013D2	Crackme3.0040218E
00401399	>	46	INC ESI	
0040139A	.	EB E7	JMP SHORT Crackme3.00401383	Crackme3.0040218E
0040139C	>	5E	POP ESI	Crackme3.0040218E
0040139D	.	E8 20000000	CALL Crackme3.004013C2	
004013A2	.	81F7 78560000	XOR EDI, 5678	
004013A8	.	8BC7	MOV EAX, EDI	
004013AA	>	EB 15	JMP SHORT Crackme3.004013C1	
004013AC	.	5E	POP ESI	Crackme3.0040218E
004013AD	.	6A 30	PUSH 30	
004013AF	.	68 60214000	PUSH Crackme3.00402160	Title = "No luck!"
004013B4	.	68 63214000	PUSH Crackme3.00402169	Text = "No luck there, mate!"
004013B9	.	FF75 08	PUSH [ARG.1]	hOwner = 014203F4 ('CrackMe v1.0', class='No need
004013BC	.	E8 79000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
004013C1	>	C3	RETN	
004013C2	>	33FF	XOR EDI, EDI	

嗯。箭头是灰色的，我们知道在这次的循环迭代中我们没有跳到坏消息那。按下 F9 执行循环体：（p27）

00401383	>	8A06	MOV AL, BYTE PTR DS:[ESI]	
00401385	.	84C0	TEST AL, AL	
00401387	>	74 13	JE SHORT Crackme3.0040139C	
00401389	.	3C 41	CMP AL, 41	
0040138B	>	72 1F	JB SHORT Crackme3.004013AC	
0040138D	.	3C 5A	CMP AL, 5A	
0040138F	>	73 03	JNB SHORT Crackme3.00401394	
00401391	.	46	INC ESI	Crackme3.0040218F
00401392	>	EB EF	JMP SHORT Crackme3.00401383	
00401394	.	E8 39000000	CALL Crackme3.004013D2	Crackme3.0040218F
00401399	>	46	INC ESI	
0040139A	.	EB E7	JMP SHORT Crackme3.00401383	Crackme3.0040218E
0040139C	>	5E	POP ESI	
0040139D	.	E8 20000000	CALL Crackme3.004013C2	
004013A2	.	81F7 78560000	XOR EDI, 5678	
004013A8	.	8BC7	MOV EAX, EDI	
004013AA	>	EB 15	JMP SHORT Crackme3.004013C1	
004013AC	.	5E	POP ESI	Crackme3.0040218E
004013AD	.	6A 30	PUSH 30	
004013AF	.	68 60214000	PUSH Crackme3.00402160	Title = "No luck!"
004013B4	.	68 63214000	PUSH Crackme3.00402169	Text = "No luck there, mate!"
004013B9	.	FF75 08	PUSH [ARG.1]	hOwner = 014203F4 ('CrackMe v1.0', class='No need
004013BC	.	E8 79000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
004013C1	>	C3	RETN	
004013C2	>	33FF	XOR EDI, EDI	

啊！第二次循环时它就要调用坏消息了。好，就让它那么干，看看我们跟踪的对不对。你可能注意到了，如果修改了 0 标志位，跳转仍然实现了。这是因为 JB 指令是跳转指令集中略有不同的那部分，它用进位标志位而不是 0 标志位（别担心，这些你的汇编语言书籍中全都有😄）。所以双击那个进位标志位（“C”）：（p28）

C	0	ES	0
P	1	CS	0
A	0	SS	0
7	0	DS	0

然后那个箭头就会变为灰色：（p29）

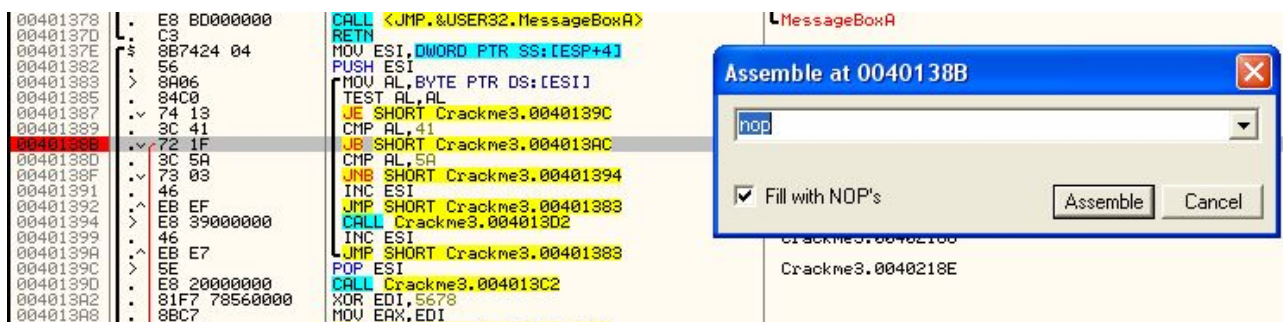


00401387	> 74 13	JE SHORT Crackme3.0040139C	
00401389	> 3C 41	CMP AL,41	
0040138B	> 72 1F	JB SHORT Crackme3.004013AC	
0040138D	> 3C 5A	CMP AL,5A	
0040138F	> 73 03	JNB SHORT Crackme3.00401394	
00401391	> 46	INC ESI	Crackme3.0040218F
00401392	> EB EF	JMP SHORT Crackme3.00401388	
00401394	> E8 39000000	CALL Crackme3.004013D2	Crackme3.0040218F
00401399	> 46	INC ESI	Crackme3.0040218E
0040139A	> EB E7	JMP SHORT Crackme3.00401388	
0040139C	> 5E	POP ESI	Crackme3.0040218E
0040139D	> E8 20000000	CALL Crackme3.004013C2	
004013A2	> 81F7 78560000	XOR EDI,5678	
004013A8	> 8BC7	MOV EAX,EDI	
004013AA	> EB 15	JMP SHORT Crackme3.004013C1	
004013AC	> 5E	POP ESI	Crackme3.0040218E
004013AD	> 6A 30	PUSH 30	
004013AF	> 68 60214000	PUSH Crackme3.00402160	Title = "No luck!"
004013B4	> 68 69214000	PUSH Crackme3.00402169	Text = "No luck there, mate!"
004013B9	> FF75 08	PUSH [ARG_1]	hOwner = 014203F4 ('CrackMe v1.0',class='No
004013BC	> E8 79000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
004013C1	> C3	RETN	
004013C2	> 33FF	XOR EDI,EDI	
004013C4	> 33DB	XOR EBX,EBX	
004013C6	> 9A1E	MOV BL,BYTE PTR DS:[ESI]	

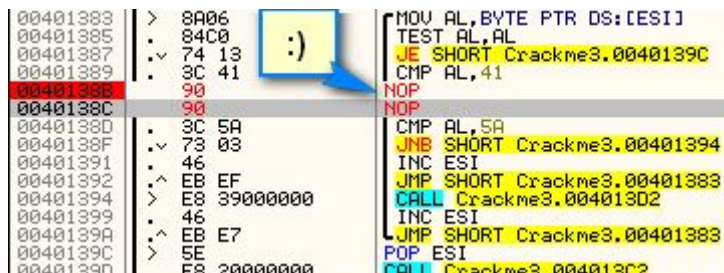
现在我们再次运行循环，看看循环中还有没有调用坏消息的。按 5 次 F9，没有一次调用坏消息。事实上，在第五次 F9 之后，我断在了一个旧断点处，我们首先想到的是补丁：(p30)

00401228	> 68 8E214000	PUSH Crackme3.0040218E	ASCII "RANDOM"
0040122D	> E8 4C010000	CALL Crackme3.0040137E	
00401232	> 50	PUSH EAX	
00401233	> 68 7E214000	PUSH Crackme3.0040217E	ASCII "12121212"
00401238	> E8 9B010000	CALL Crackme3.004013D8	
0040123D	> 83C4 04	ADD ESP,4	
00401240	> 58	POP EAX	Crackme3.0040218E
00401241	> 3BC3	CMP EAX,EBX	
00401243	> 74 07	JE SHORT Crackme3.0040124C	
00401245	> E8 18010000	CALL Crackme3.00401372	
00401249	> EB 9A	JMP SHORT Crackme3.004011E6	
0040124C	> E8 FC000000	CALL Crackme3.0040134D	
00401251	> EB 93	JMP SHORT Crackme3.004011E6	
00401253	> C8 000000	ENTER 0,0	
00401257	> 53	PUSH EBX	Crackme3.00402188
00401258	> 56	PUSH ESI	
00401259	> 57	PUSH EDI	
0040125A	> 817D 0C 10010000	CMP [ARG_2],110	
00401261	> 74 34	JE SHORT Crackme3.00401297	
00401263	> 817D 0C 11010000	CMP [ARG_2],111	
0040126A	> 74 35	JE SHORT Crackme3.004012A1	
0040126C	> 837D 0C 10	CMP [ARG_2],10	
00401270	> 0F84 81000000	JE Crackme3.004012F7	
00401276	> 817D 0C 01020000	CMP [ARG_2],201	
0040127D	> 74 0C	JE SHORT Crackme3.0040128B	
0040127F	> B8 00000000	MOV EAX,0	
00401284	> 5F	POP EDI	Crackme3.0040218E
00401285	> 5E	POP ESI	Crackme3.0040218E
00401286	> 5B	POP EBX	Crackme3.0040218E
00401287	> C9	LEAVE	
00401288	> C2 1000	RETN 10	

那么，这就意味着我们已经成功的绕过了对坏消息的第一个检测，并且回到了原来的检测点。咱们给第一个检测打个补丁，这样就再也不用操心它了，就可以将注意力集中在主要的检测点。回到 40138B 的断点处，我们得想想怎么给它打补丁而不让它跳转到坏消息那。记住，跳转是在第二次循环时实现的，只有在 AL 的值小于 41 时才成立（相关指令是 CMP AL, 31, JB SHORT 4013AC）。如果我们只 NOP 掉这个跳转会怎么样？它就再也不会跳了，我们一点也不用担心它会跳到坏消息那😁：(p31)

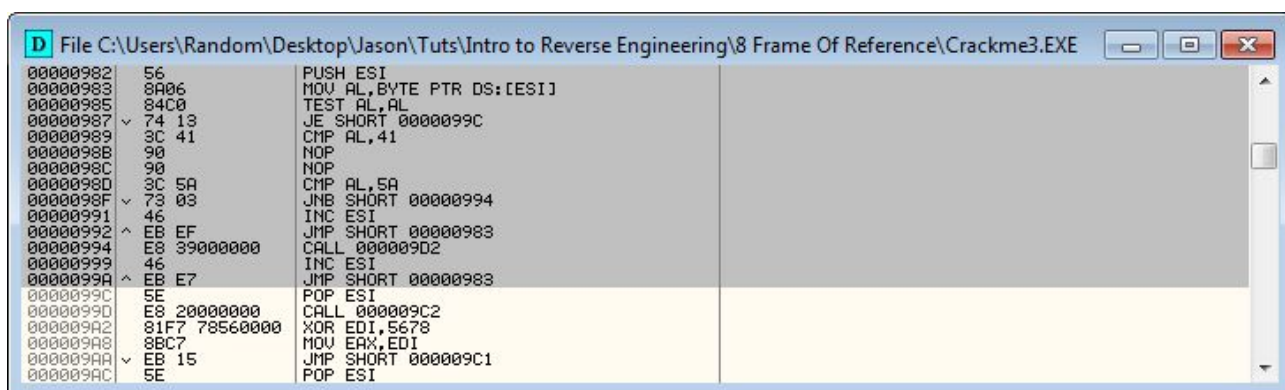


☹️(p32)

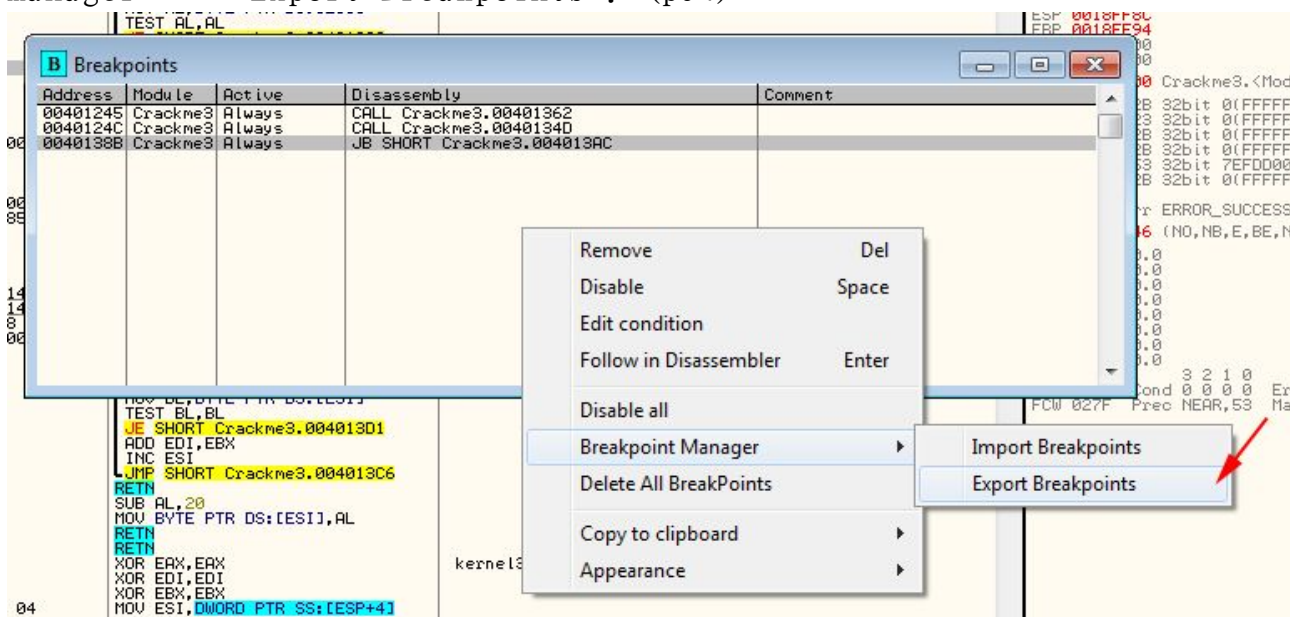


```
00401383 > 8A06 MOV AL, BYTE PTR DS:[ESI]
00401385 . 84C0 TEST AL, AL
00401387 . 74 13 JE SHORT Crackme3.0040139C
00401389 . 3C 41 CMP AL, 41
0040138B 90 NOP
0040138C 90 NOP
0040138D . 3C 5A CMP AL, 5A
0040138F . 73 03 JNB SHORT Crackme3.00401394
00401391 . 46 INC ESI
00401392 . EB EF JMP SHORT Crackme3.00401383
00401394 > E8 39000000 CALL Crackme3.004013D2
00401399 . 46 INC ESI
0040139A . EB E7 JMP SHORT Crackme3.00401383
0040139C > 5E POP ESI
0040139D . E8 20000000 CALL Crackme3.004013C2
```

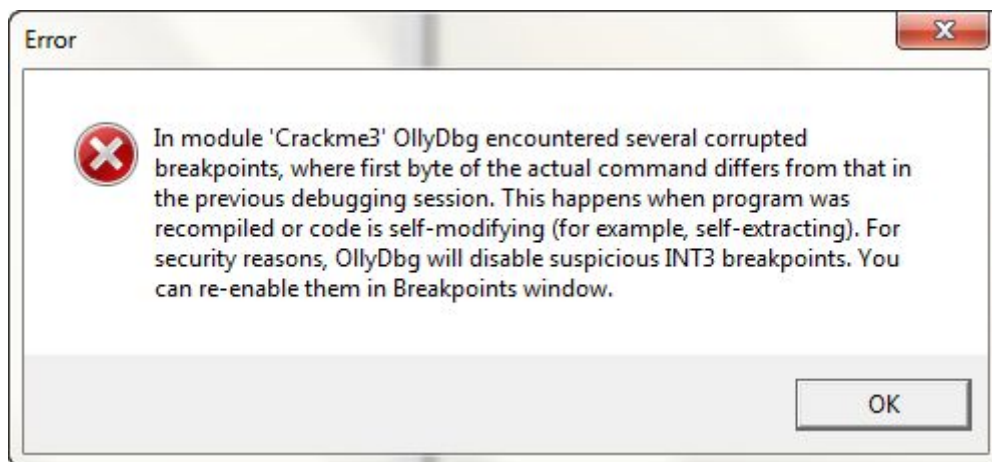
右键，选择“Copy to executable” -> “All modifications”。弹出内存窗口，右键该窗口，选择“Save File”，将其另存为 crackme\_patch1.exe: (p33)



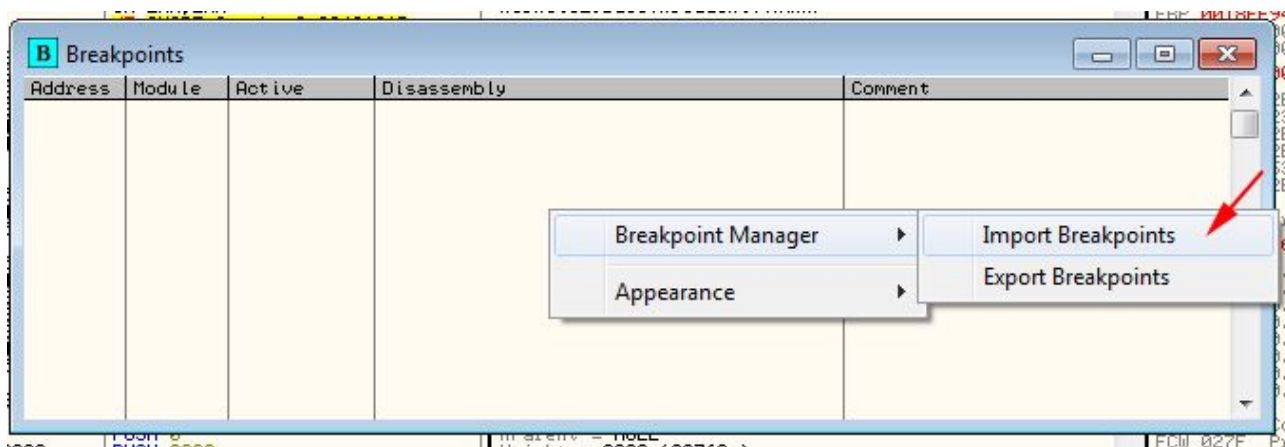
现在，在重新载入刚刚打过补丁版本前，我们要明白所有的补丁、注释和（尤其是）断点都会被删除，因为所有的信息都存储在 Crackme3. udd 这个 UDD 文件中。我们将要打开的 Crackme3\_Patch1，并没有和它相关的 UDD 文件。不过还是有几个好消息的。本文的相关下载中包含有断点管理插件。如果你没有准备好，那就将其拷贝到你的插件目录下，然后重启 Olly. 如果你一开始就安装好了，那你就已经载入了它。现在打开断点窗口，右键并选择“Breakpoint manager” ->” Export Breakpoints”: (p34)



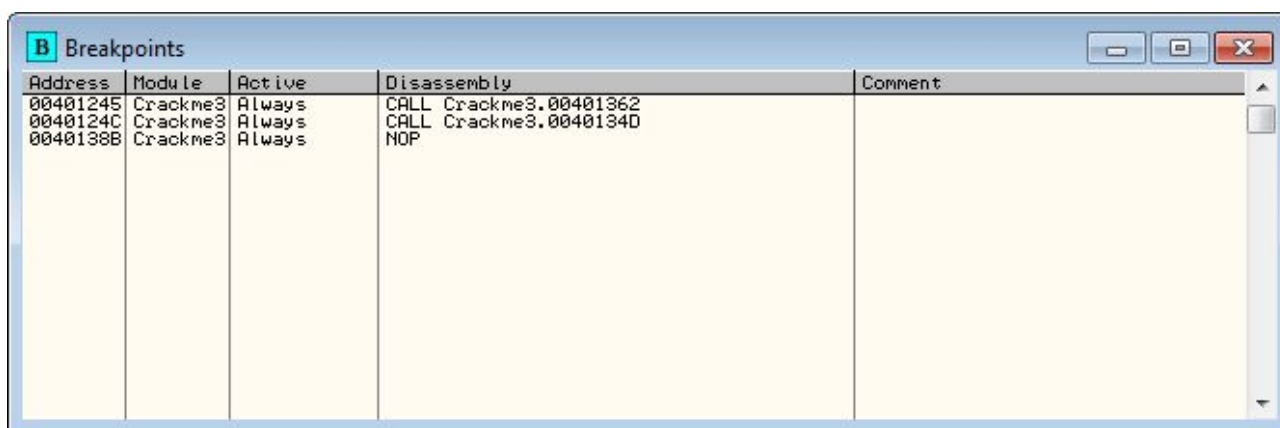
保存文件，因为我们将要将其导入到新文件。现在，将新文件（打过补丁的）载入 OllyDbg。它很可能会弹出一个消息框，告诉你断点被破坏了：（p35）



点 OK 就行了。在我们的打过补丁的程序中打开断点窗口，很可能所有的（或大部分）断点消息了。现在，右键并选择 “Breakpoint Manager” -> “Import breakpoints”：（p36）



现在你会看到我们原来的断点又回来了：（p37）





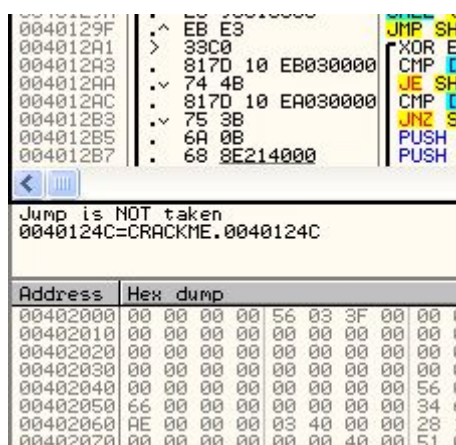
运行程序，Ollly 会断在我们的第一个断点也就是 401243 的 JE 指令处（如果你没有在该行设置断点，那就设一个。译者注：吐一下槽，原作者太操蛋，从来都只打圆括号的左半个，右半个就不管了，我还得自己琢磨把右边的圆括号放哪）。重启应用并运行，你会断在这里：（p38）

```

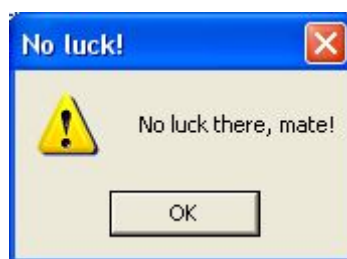
00401240 : 30          FOR EAX
00401241 : 3BC3        CMP EAX,EBX
00401243 : 74 07        JE SHORT CRACKME.0040124C
00401245 : E8 18010000 CALL CRACKME.00401362
0040124A : EB 9A        JMP SHORT CRACKME.004011E6
0040124C : E8 FC000000 CALL CRACKME.0040134D
00401251 : EB 93        JMP SHORT CRACKME.004011E6
00401253 : C8 000000    ENTER 0,0
00401257 : 53          PUSH EBX

```

现在，看那个灰色的箭头，就是从当前暂停行往下到 40124C。因为箭头是灰色的，所以跳转不会实现。你也可以看反汇编窗口与数据窗口中间的那块，它告诉你跳转**没有**实现：（p39）



这意味着，什么都不用做，程序不会跳到第二个调用，会直达第一个调用。第一个调用会跳到坏消息那，所以我们真心不想让它发生。按一下 F8 单步步过。就像 Ollly 告诉我们的，没有跳转，我们当前的位置就在调用坏消息的地方。按一下 F7 单步步入那个 CALL，我们来到了坏消息所在函数的第一条指令处。现在，如果我们按 F9 让程序运行，我们看到的正是意料之中的：（p40）



咱们来看看能不能解决这个😅。重启应用，按 F9 运行之，选择 “Help” -> “Register”，然后输入用户名和序列号。现在，当你按下 OK 按钮时，Ollly 会再次停在我们的第一个断点：（p41）

00401240	30	PUSH EAX	
00401241	3BC3	CMP EAX,EBX	
00401243	74 07	JE SHORT CRACKME.0040124C	
00401245	E8 10010000	CALL CRACKME.00401362	
0040124A	EB 9A	JMP SHORT CRACKME.004011E6	
0040124B	E8 FC000000	CALL CRACKME.0040134D	
00401251	EB 93	JMP SHORT CRACKME.004011E6	
00401253	C8 000000	ENTER 0,0	
00401257	53	PUSH EBX	

这回，咱们来帮助 01ly 走正确的路。浏览下寄存器窗口，注意到 Z 标志位是红色的。嗯，你知道该怎么做了：(p42)

```

C 1 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)
T 0 GS 0000 NULL
D 0
O 0
N 0 LastErr ERROR_SUCCESS (0000)

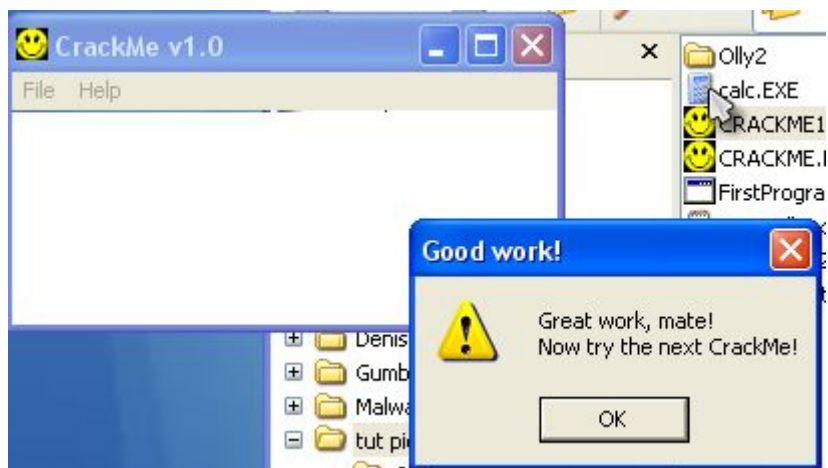
```

注意，箭头是灰色的，显示跳转不会发生，不过现在变红了，在反汇编窗口和数据窗口之间的那个区域已经变成了“Jump will be taken (跳转将会发生)”。我们所做的就是告诉 01ly 去修改标志位，该标志位用来判定两个东西是否相同，为了让它认为它们是相同的。所以现在，我们会跳过对坏消息的调用，转而去调用好消息!!!

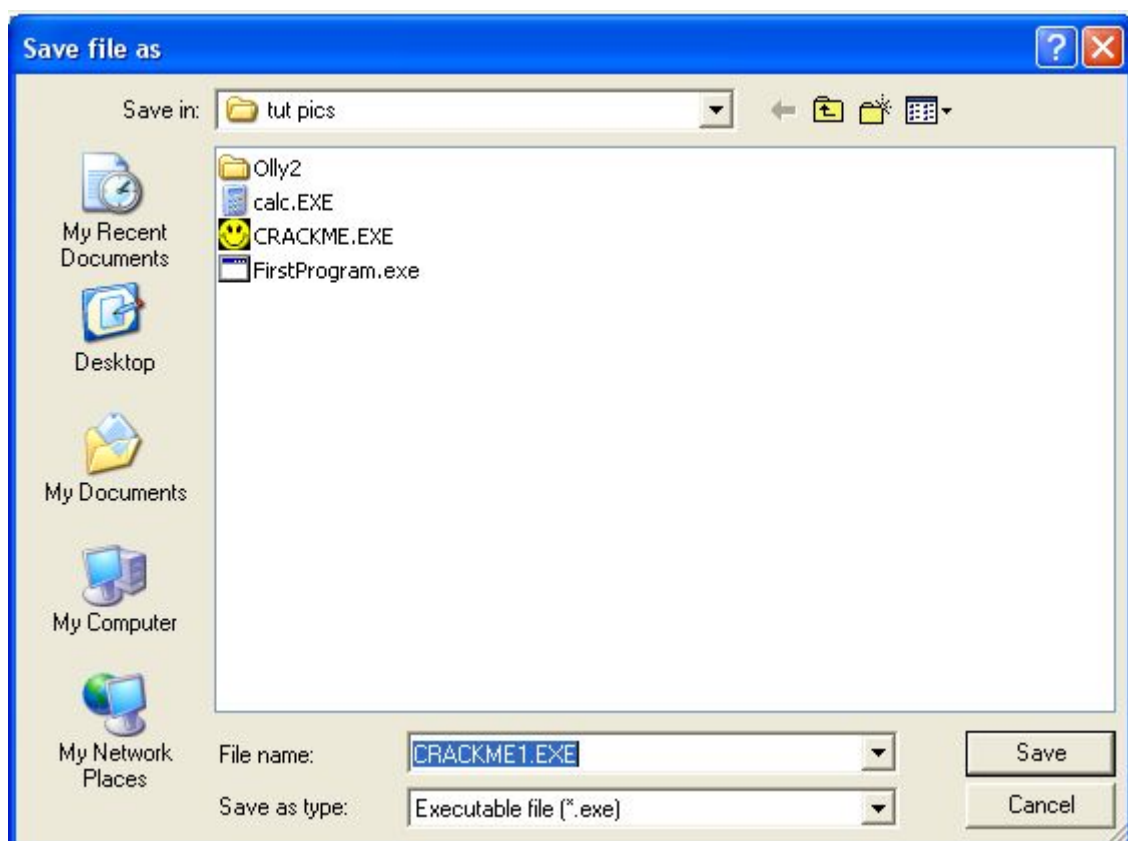
咱们试试。按 F8 执行跳转，再按 F7 单步步入到那个 CALL。现在我们将跳转到好消息的起始处：(p43)

0040134D	6A 30	PUSH 30	Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL
0040134F	68 29214000	PUSH CRACKME.00402129	Title = "Good work!"
00401354	68 34214000	PUSH CRACKME.00402134	Text = "Great work, mate!\r\nNow try the next CrackMe!"
00401359	FF75 08	PUSH [ARG.1]	hOwner = 00400356 ('CrackMe v1.0',class='No need to disasm
0040135C	E8 D9000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
00401361	C3	RETN	
00401362	6A 00	PUSH 0	BeepType = MB_OK
00401364	E8 AD000000	CALL <JMP.&USER32.MessageBeep>	MessageBeep
00401369	6A 30	PUSH 30	Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL
0040136B	68 60214000	PUSH CRACKME.00402160	Title = "No luck!"
00401370	68 69214000	PUSH CRACKME.00402169	Text = "No luck there, mate!"
00401375	FF75 08	PUSH [ARG.1]	hOwner = 00400356 ('CrackMe v1.0',class='No need to disasm
00401378	E8 BD000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
0040137D	C3	RETN	
0040137E	8B7424 04	MOV ESI,DWORD PTR SS:[ESP+4]	CRACKME.0040218E
00401382	56	PUSH ESI	CRACKME.00402188

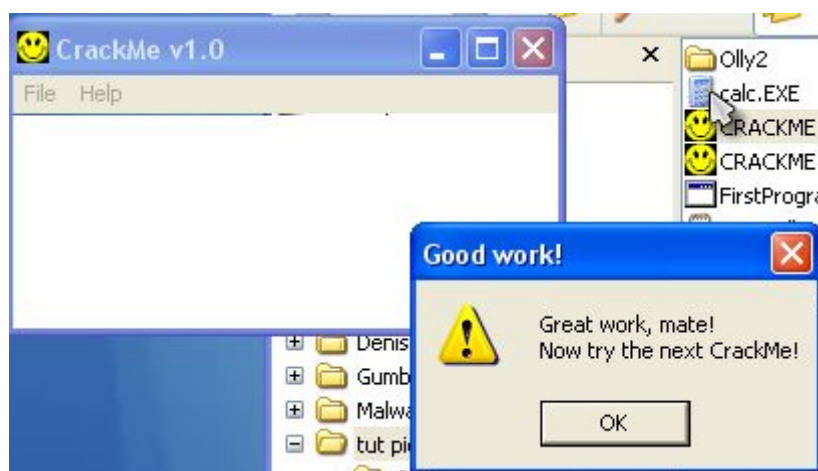
现在，按几次 F8，每按一次就观察一下堆栈窗口。你会看到 MessageBoxA 的参数被压入堆栈，本例中确实是好消息被压入堆栈。只要你单步步过 40135C 处的函数，新的消息对话框就会显示出来。我们已经破解了我们的第一个程序!!! (p44)



现在的问题是，我们只是临时性的修改了标志寄存器，当程序再次运行时，它却不会再次的修改标志位，所以我们还是会得到坏消息。我们需要做的是通过某种方式将修改保存起来，以便于每一次程序运行的时候，我们都能强制它做跳转。这时候补丁要派上用场了。和我们以前做的一样：选中所有已修改的行，右键并选择“Copy to executable”。在弹出的窗口中右键，点击“Save file”。选一个名字，这就是你的打过补丁的版本：(p45)



现在可以关掉数据窗口和 Olly 了。打开你保存的打补丁版本文件夹，运行打过补丁的程序。输入你的信息并验证：(p46)



干的漂亮！你已经真正的破解了一个有些挑战的 crackme。