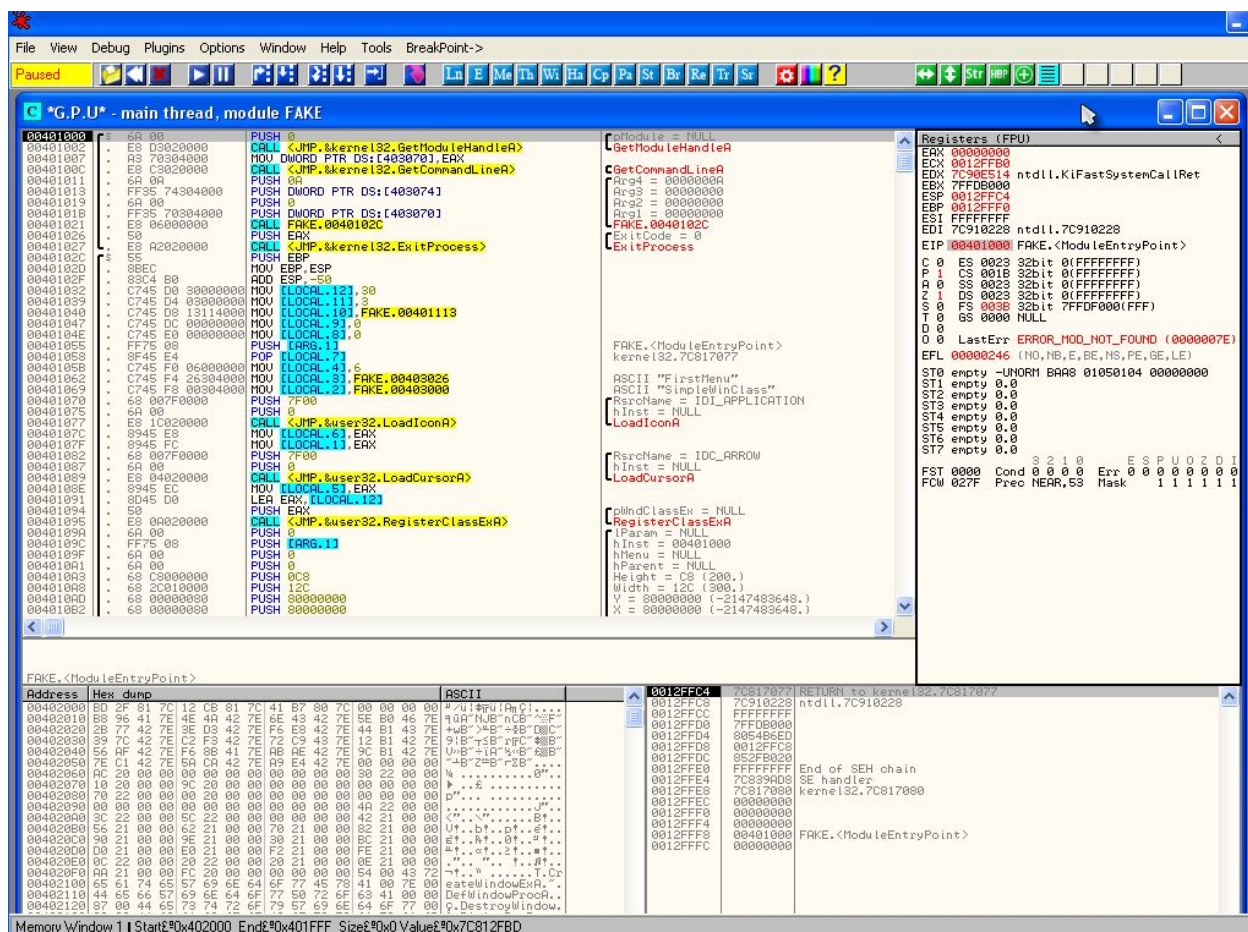


教程五：第一次破解（算是）

一、简介

此次教程通过预览一个 crackme，我们会结束 011y 使用方面剩下的内容。好吧，算是一个 crackme。其实就是我们前面使用的程序，不过被修改成需要序列号注册了，如果输入正确序列号会显示一个好消息，否则显示一个坏消息。我选择这么做而不是用一个完全不同的 crackme，是因为我想要你能够专注于序列号校验程序部分，而不是陷入其他的代码中。下一课我们会研究一个真正的 crackme（我保证）。

此次教程你所需要的就是一个 011yDBG（我的版本或者原始版本都可以），以及一个我改进了的 crackme。顺便说一下，我把改进后的 crackme 叫做“First Assembly Kracking Engine”，或者是 F. A. K. E. 它包含在此次教程的文件下载中。（是的，Gdogg，我知道 Kracking 不是以字母“K”开头的。译者注：如果取 crack 首字母，最后缩写就是 face 了，还有那个 Gdogg 我不造啥意思）我们开始吧。



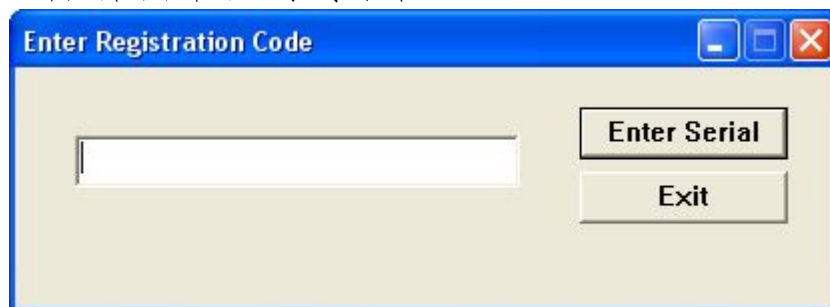
如果你在 011y 中载入了 FAKE.exe 的话，就会发现第一页代码和我们上一次学习用的程序一样。



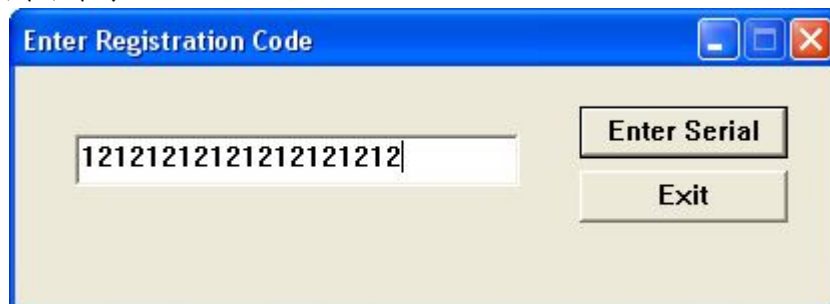
运行下程序，看看它如何工作的是及其重要的。



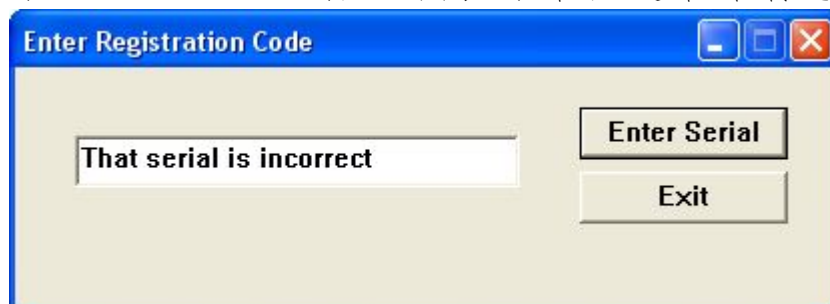
点击注册弹出下面的对话框。



输入序列号。



在点了 Enter Serial 后，出现了下面这个坏消息。



真见鬼！我那么努力的尝试!!! 😊

现在我向你介绍每个新手查找注册校验代码的第一个方法。

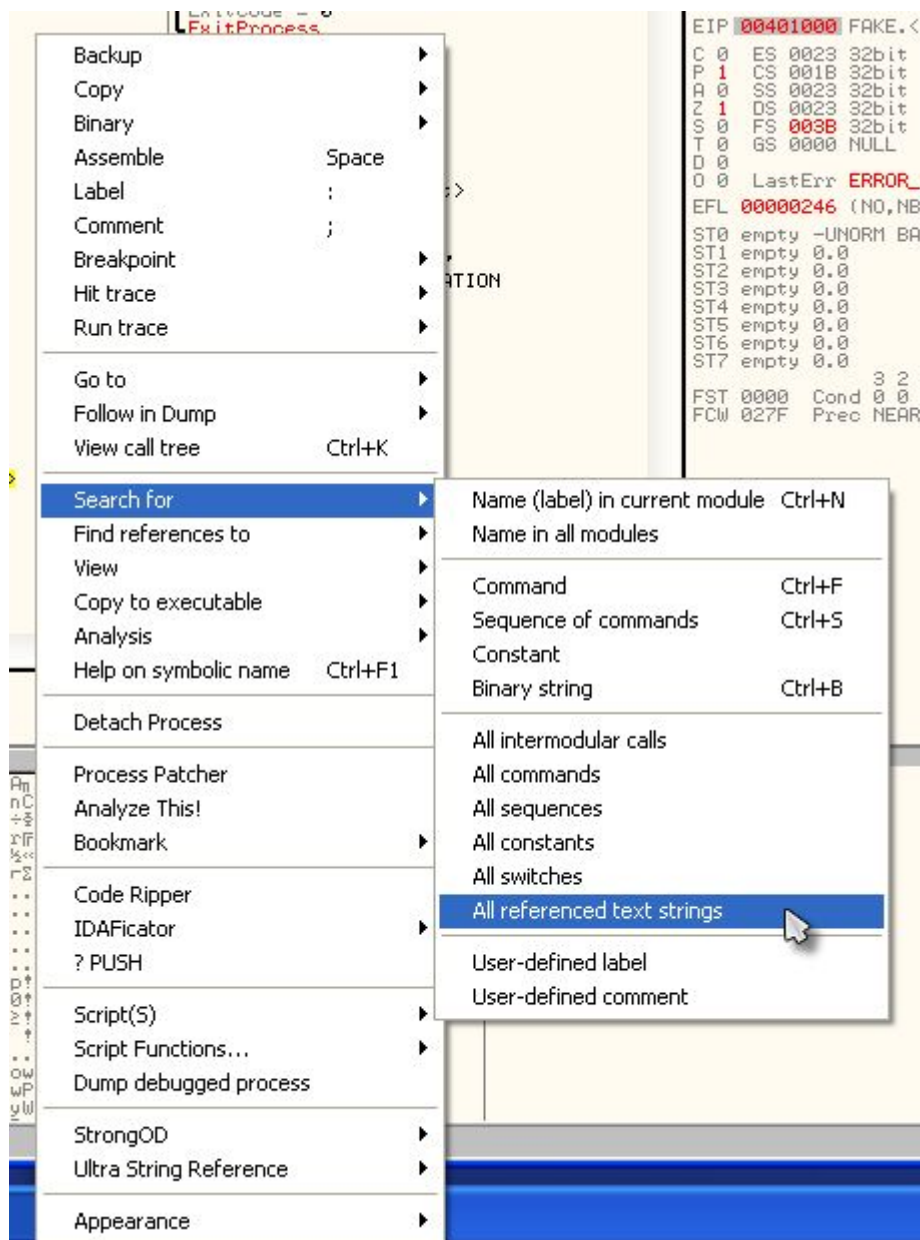
二、搜索所有的文本字符串

先说一下，有许多“老练”的逆向者（或破解者）觉得这个方法已经很少用了。因为这个方法太过于明显了，所以凡是想保护自己的软件不被逆向的人都会让这招失效。这些软件被压缩、保护、加密或修改，只要作者不是一个完完全全的傻子，就会加密字符串以让“Search for strings（搜索字符串）”方法失效。话虽如此，不过我还是发现有许多傻子，这个消息可别告诉任何老鸟，所以我做的第一件事就是检查这个（ps. 这其实也是老鸟做的第一件事）。

基本上，该方法都会涉及到让 Olly 搜索你的程序的内存空间，搜索任何看起来像是 ASCII 或 Unicode 文本字符串。通常，可以立即发现该方法好不好用，会有大量的文本字符串，许多看起来很诱人（比如“Thank you for registering!!!（谢谢注册!!!）”）。或者是有很少的字符串，而且许多都像这样“F07=”。

了解一个二进制文件中是否有合法字符串可以给你一些有价值的信息。比如二进制文件是否通过某种方法被压缩或保护，是否是一个恶意二进制文件（毕竟，“Send all user's passwords to www.badguys.com”这样的句子不会是一个非常负责任的病毒所写吧），甚至二进制文件是用非常少见的语言所写。

咱们来看看具体怎么做。右键反汇编窗口，选择“Search for”->“All Referenced Text Strings”。



然后 OllyDbg 就会搜索程序的内存空间，并显示文本字符串窗口 (Text Strings Window):

R Text strings referenced in FAKE!.text		
Address	Disassembly	Text string
00401039	MOV [LOCAL.11],3	(Initial CPU selection)
00401062	MOV [LOCAL.3],FAKE.00403026	ASCII "FirstMenu"
00401069	MOV [LOCAL.2],FAKE.00403000	ASCII "SimpleWinClass"
004010BC	PUSH FAKE.0040300F	ASCII "Our Main Window-WinAsm"
004010C1	PUSH FAKE.00403000	ASCII "SimpleWinClass"
00401141	PUSH FAKE.00403030	ASCII "MyDialog"
00401222	PUSH FAKE.00403052	ASCII "That serial is correct!!!!"
00401236	PUSH FAKE.00403039	ASCII "That serial is incorrect"

嗯，看起来很有意思吧:) 注意这个列表是真的短，因为这个程序确实非常短小。一般来说，会有数千行字符串。还有，你注意到我注意的了吗：

R Text strings referenced in FAKE!.text		
Address	Disassembly	Text string
00401039	MOV [LOCAL.11],3	(Initial CPU selection)
00401062	MOV [LOCAL.3],FAKE.00403026	ASCII "FirstMenu"
00401069	MOV [LOCAL.2],FAKE.00403000	ASCII "SimpleWinClass"
004010BC	PUSH FAKE.0040300F	ASCII "Our Main Window-WinAsm"
004010C1	PUSH FAKE.00403000	ASCII "SimpleWinClass"
00401141	PUSH FAKE.00403030	ASCII "MyDialog"
00401222	PUSH FAKE.00403052	ASCII "That serial is correct!!!!"
00401236	PUSH FAKE.00403039	ASCII "That serial is incorrect"

前途有望啊。咱们跳到代码那看看有什么：双击 “That serial is correct!!!!” 那一行，Ollly 就会在反汇编窗口显示那一块代码：

```
*G.P.U* - main thread, module FAKE
00401106 . 6A 00 PUSH 0
00401108 . 6A 00 PUSH 0
0040110A . 6A 10 PUSH 10
0040110C . FF75 08 PUSH [ARG.1]
0040110F . E8 C6000000 CALL <JMP.&user32.SendMessageA>
00401114 . JNP SHORT FAKE.00401248
00401116 . CMP EAX,0BB9
00401118 . JNZ SHORT FAKE.00401248
0040111A . 6A 64 PUSH 64
0040111C . 68 78304000 PUSH FAKE.00403078
0040111E . 68 B80B0000 PUSH 0BB8
00401120 . FF75 08 PUSH [ARG.1]
00401122 . E8 85000000 CALL <JMP.&user32.GetDlgItemTextA>
00401124 . MOV EBX,DWORD PTR DS:[403078]
00401126 . CMP BL,61
00401128 . JNZ SHORT FAKE.00401236
0040112A . MOV EBX,DWORD PTR DS:[403079]
0040112C . CMP BL,62
0040112E . JNZ SHORT FAKE.00401236
00401130 . MOV EBX,DWORD PTR DS:[40307A]
00401132 . CMP BL,63
00401134 . JNZ SHORT FAKE.00401236
00401136 . 68 52304000 PUSH FAKE.00403052
00401138 . 68 B80B0000 PUSH 0BB8
0040113A . FF75 08 PUSH [ARG.1]
0040113C . E8 7C000000 CALL <JMP.&user32.SetDlgItemTextA>
0040113E . JNP SHORT FAKE.00401248
00401140 . 68 39304000 PUSH FAKE.00403039
00401142 . 68 B80B0000 PUSH 0BB8
00401144 . FF75 08 PUSH [ARG.1]
00401146 . E8 68000000 CALL <JMP.&user32.SetDlgItemTextA>
00401148 . JNP SHORT FAKE.00401253
0040114A . MOV EAX,0
0040114C . C9 LEAVE
0040114E . C2 1000 RETN 10
00401150 . B8 01000000 MOV EAX,1
00401152 . C9 LEAVE
00401154 . C2 1000 RETN 10
00401156 . JMP DWORD PTR DS:[&user32.CreateWindowExA]
00401158 . JMP DWORD PTR DS:[&user32.DefWindowProcA]
0040115A . JMP DWORD PTR DS:[&user32.DestroyWindow]
0040115C . JMP DWORD PTR DS:[&user32.ShowDialogParamA]
0040115E . JMP DWORD PTR DS:[&user32.DispatchMessageA]
00401160 . JMP DWORD PTR DS:[&user32.EndDialog]

iParam = 0
wParam = 0
Message = WM_CLOSE
hWnd = 401000
SendMessageA

Count = 64 (100.)
Buffer = FAKE.00403078
ControlID = BB8 (3000.)
hWnd = 00401000
GetDlgItemTextA

Text = "That serial is correct!!!!"
ControlID = BB8 (3000.)
hWnd = 00401000
SetDlgItemTextA

Text = "That serial is incorrect"
ControlID = BB8 (3000.)
hWnd = 00401000
SetDlgItemTextA

user32.CreateWindowExA
user32.DefWindowProcA
user32.DestroyWindow
user32.ShowDialogParamA
user32.DispatchMessageA
user32.EndDialog
```

是时候介绍第二条规则了

R4ndom' s Essential Truths About Reversing Data:

R4ndom 关于逆向数据的必备真理：

#2：大多数的保护机制是可以简单的通过修改一个跳转指令来绕过“坏”代码直接跳到“好”代码的。

意思是几乎每一次在坏消息显示之前，就会有某种检查（我们注册了吗？注册码对不对？试用时间过了吗？.....），对比之后就有一个跳转，至于是跳到好消息还是坏消息则依赖于对比的结果。

我们自已来找找看啊...。好消息“This serial is correct!!!!”是从 401222 开始的，向上翻找跳转语句，尤其是

它前面有某种比较（或 CALL）的跳转语句。如果是一个 CALL，可以猜测比较是在 CALL 内部进行的…。我们的例子中，第一个跳转是在 401220 的 JNZ。我在图中加了一个箭头，向你演示了如果跳转成立的话，将会跳到哪去：

00401217	8B1D 7A304000	MOV EBX,DWORD PTR DS:[40307A]	
0040121D	80FB 63	CMP BL,63	
00401220	75 14	JNZ SHORT FAKE.00401236	
00401222	68 52304000	PUSH FAKE.00403052	Text = "That serial is correct!!!!"
00401227	68 B80B0000	PUSH 0BB8	ControlID = BB8 (3000.)
0040122C	FF75 08	PUSH [ARG.1]	hWnd = 00401000
0040122F	E8 7C000000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
00401234	EB 12	JMP SHORT FAKE.00401248	
00401236	68 39304000	PUSH FAKE.00403039	Text = "That serial is incorrect"
0040123B	68 B80B0000	PUSH 0BB8	ControlID = BB8 (3000.)
00401240	FF75 08	PUSH [ARG.1]	hWnd = 00401000
00401243	E8 68000000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
00401248	EB 09	JMP SHORT FAKE.00401253	
0040124D	B8 00000000	MOV EAX,0	

嗯。注意它刚好跳过了我们想要的消息，跳到了我们不想要的消息😞。不过，注意在 JNZ 指令的前面是一个 CMP 指令😁。意思是，这个是 011y 决定显示我们想要还是不想要的消息的关键点。我们再向上翻翻：

G.P.U - main thread, module FAKE			
004011D6	6A 00	PUSH 0	[Param = 0]
004011D8	6A 00	PUSH 0	wParam = 0
004011DA	6A 10	PUSH 10	Message = WM_CLOSE
004011DC	FF75 08	PUSH [ARG.1]	hWnd = 401000
004011DF	E8 C6000000	CALL <JMP.&user32.SendMessageA>	SendMessageA
004011E4	EB 62	JMP SHORT FAKE.00401248	
004011E6	3D B90B0000	CMP EAX,0BB9	
004011ED	75 5B	JNZ SHORT FAKE.00401248	
004011EF	6A 64	PUSH 64	Count = 64 (100.)
004011F4	68 78304000	PUSH FAKE.00403078	Buffer = FAKE.00403078
004011F9	68 B80B0000	PUSH 0BB8	ControlID = BB8 (3000.)
004011FC	FF75 08	PUSH [ARG.1]	hWnd = 00401000
004011FD	E8 85000000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
00401201	8B1D 78304000	MOV EBX,DWORD PTR DS:[403078]	
00401207	80FB 61	CMP BL,61	
0040120A	75 2A	JNZ SHORT FAKE.00401236	
0040120C	8B1D 79304000	MOV EBX,DWORD PTR DS:[403079]	
00401212	80FB 62	CMP BL,62	
00401215	75 1F	JNZ SHORT FAKE.00401236	
00401217	8B1D 7A304000	MOV EBX,DWORD PTR DS:[40307A]	
0040121D	80FB 63	CMP BL,63	
00401220	75 14	JNZ SHORT FAKE.00401236	
00401222	68 52304000	PUSH FAKE.00403052	Text = "That serial is correct!!!!"
00401227	68 B80B0000	PUSH 0BB8	ControlID = BB8 (3000.)
0040122C	FF75 08	PUSH [ARG.1]	hWnd = 00401000
0040122F	E8 7C000000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
00401234	EB 12	JMP SHORT FAKE.00401248	
00401236	68 39304000	PUSH FAKE.00403039	Text = "That serial is incorrect"
0040123B	68 B80B0000	PUSH 0BB8	ControlID = BB8 (3000.)
00401240	FF75 08	PUSH [ARG.1]	hWnd = 00401000
00401243	E8 68000000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
00401248	EB 09	JMP SHORT FAKE.00401253	
0040124A	B8 00000000	MOV EAX,0	
0040124F	C9	LEAVE	
00401250	C2 1000	RETN 10	
00401253	B8 01000000	MOV EAX,1	
00401258	C9	LEAVE	
00401259	C2 1000	RETN 10	
0040125C	FF25 58204000	JMP DWORD PTR DS:[<&user32.CreateWindowExA>]	user32.CreateWindowExA
00401262	FF25 50204000	JMP DWORD PTR DS:[<&user32.DefWindowProcA>]	user32.DefWindowProcA
00401268	FF25 4C204000	JMP DWORD PTR DS:[<&user32.DestroyWindow>]	user32.DestroyWindow
0040126E	FF25 2C204000	JMP DWORD PTR DS:[<&user32.MessageBoxParamA>]	user32.MessageBoxParamA
00401274	FF25 10204000	JMP DWORD PTR DS:[<&user32.DispatchMessageA>]	user32.DispatchMessageA
0040127A	FF25 14204000	JMP DWORD PTR DS:[<&user32.EndDialog>]	user32.EndDialog

在 401212 有另一对 CMP/JNZ，在 401207 有最后一对。凑近点看，你会发现所有的三个跳转都跳过了好消息，跳到了坏消息那。逻辑上，这意味着有三件事被检查，触发任何一个都会命中坏消息。不过，如果我三个跳转都不跳会怎么样？好吧，你会看到我们将空降到好消息那。所以，真正的意思是，如果我们让这

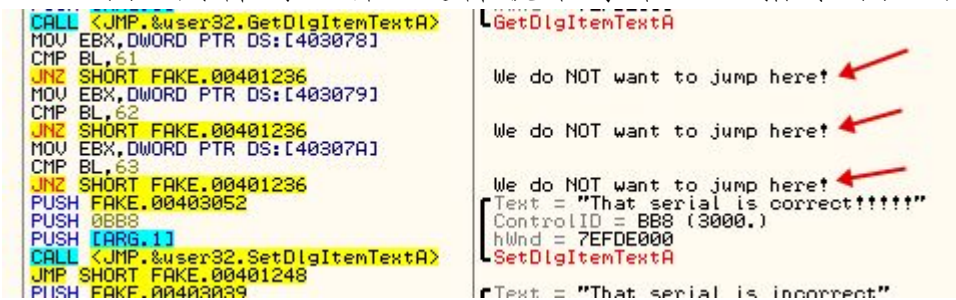
些跳转都不跳，程序会“空降”到好消息那里（译者注：这里作者用的是“fall through”，大概意思是如果将三个 jmp 指令当做一层层的阻碍的话，我们直接穿过这些阻碍到达显示好消息的代码，就是将这三个 jmp 无视掉当作透明的。有些东西可意会，不好言传，所以我将它翻成“空降”）。

我们运行下程序看看它做了什么，不过我先向大家介绍点别的。

三、如何添加注释

注释是很重要的，尤其是在你开始分析错综复杂的代码时。代码本来就很难读，不过有了注释后，我们就可以在非常重要的地方提醒自己。我准备为每个 JNZ 指令添加注释，以此来提醒我们自己什么需要被发生。

要添加注释，你可以双击要添加注释的那行的最后一列（那里，011y 已经放置了类似于“This is the correct serial!!!”这样的其他注释），也可以先选中要添加注释的那行，然后按一下“;”键。好，我们先选中 40120A 那行，然后按一下分号键，接着输入“We do NOT want to jump here!”。现在，给 401215 和 401220 添加同样的注释。这样就给每个 JNZ 指令添加了注释：



现在让我们在 401201 处设置一个断点（在跳转指令前面的其他地方设断点也行）：

004011ED	6A 64	PUSH 64	Count = 64 (100.)
004011EF	68 78304000	PUSH FAKE.00403078	Buffer = FAKE.00403078
004011F4	68 B80B0000	PUSH 0BB8	ControlID = BB8 (3000.)
004011F9	FF75 08	PUSH [ARG.1]	hWnd = 001C05E4 ('Enter Registration Code',
004011FC	E8 85000000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401201	8B1D 78304000	MOV EBX,DWORD PTR DS:[403078]	
00401207	80FB 61	CMP BL,61	
0040120A	75 2A	JNZ SHORT FAKE.00401236	We do NOT want to jump here!
0040120C	8B1D 79304000	MOV EBX,DWORD PTR DS:[403079]	
00401212	80FB 62	CMP BL,62	
00401215	75 1F	JNZ SHORT FAKE.00401236	We do NOT want to jump here!
00401217	8B1D 7A304000	MOV EBX,DWORD PTR DS:[40307A]	
0040121D	80FB 63	CMP BL,63	
00401220	75 14	JNZ SHORT FAKE.00401236	We do NOT want to jump here!
00401222	68 52304000	PUSH FAKE.00403052	Text = "That serial is correct!!!!!"
00401227	68 B80B0000	PUSH 0BB8	ControlID = BB8 (3000.)
0040122C	FF75 08	PUSH [ARG.1]	hWnd = 001C05E4 ('Enter Registration Code',
0040122F	E8 7C000000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
00401234	EB 12	JMP SHORT FAKE.00401248	

让程序跑起来。点击 crackme 上面的 “Register”，输入序列号，再点一下 “Enter serial”。Ollly 就会暂停在断点处：

004011ED	6A 64	PUSH 64	Count = 64 (100.)
004011EF	68 78304000	PUSH FAKE.00403078	Buffer = FAKE.00403078
004011F4	68 B80B0000	PUSH 0BB8	ControlID = BB8 (3000.)
004011F9	FF75 08	PUSH [ARG.1]	hWnd = 001C05E4 ('Enter Registration Code',
004011FC	E8 85000000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401201	8B1D 78304000	MOV EBX,DWORD PTR DS:[403078]	
00401207	80FB 61	CMP BL,61	
0040120A	75 2A	JNZ SHORT FAKE.00401236	We do NOT want to jump here!
0040120C	8B1D 79304000	MOV EBX,DWORD PTR DS:[403079]	
00401212	80FB 62	CMP BL,62	
00401215	75 1F	JNZ SHORT FAKE.00401236	We do NOT want to jump here!
00401217	8B1D 7A304000	MOV EBX,DWORD PTR DS:[40307A]	
0040121D	80FB 63	CMP BL,63	
00401220	75 14	JNZ SHORT FAKE.00401236	We do NOT want to jump here!
00401222	68 52304000	PUSH FAKE.00403052	Text = "That serial is correct!!!!!"
00401227	68 B80B0000	PUSH 0BB8	ControlID = BB8 (3000.)
0040122C	FF75 08	PUSH [ARG.1]	hWnd = 001C05E4 ('Enter Registration Code',
0040122F	E8 7C000000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
00401234	EB 12	JMP SHORT FAKE.00401248	

现在，我们第一个要注意的是我们停止的那行：

MOV EBX, DWORD PTR DS:[403078]

从上一课中我们知道该如何查看该内存地址的内容，在指令上右键，选择 “Follow in Dump” -> “Memory Address”。然后我们就可以在 Ollly 的数据窗口中看到该内存的内容：

Address	Hex	dump	ASCII
00403078	31 32 31 32	31 32 31 32 31 32 31 32 31 32 31 32 31 32 31 32	1212121212121212
00403088	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	12.....
00403098	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030A8	00 00 00 00	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 00 00 00 00
004030C8	00 00 00 00	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 00 00 00 00
004030E8	00 00 00 00	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 00 00 00 00
00403108	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

好，好，好。这不就是我们刚刚输入的序列号嘛。所以，根据这条指令，我们知道了前面四个字节（因为 EAX 是 32 为寄存器）被载入 EBX，也就是 31 32 31 32，用 ASCII 码表示就是 “1212”。按一下 F8 再检查 EBX：

Registers (FPU)		
EAX	00000012	
ECX	74A8008E	user32.74A8008E
EDX	00000030	
EBX	32313231	
ESP	0018F8C4	
EBP	0018F8C4	
ESI	00401178	FAKE.00401178
EDI	00000000	
EIP	00401207	FAKE.00401207

如果你想看看 EBX 中的 ASCII 字符,你可以双击 EBX 寄存器,就会显示几组不同格式的数据,其中一组就是 ASCII:

**为了后面用到,如果你想对不同的寄存器尝试不同的值得话,记住这也是“即时”修改寄存器的一种方法。*

我猜你已经从汇编语言的书中知道了这种方法(我的意思是,来吧!我甚至在工具区上传了一个!!!),我不需要讨论这个,只需要复习下。

四、小端序列

(至少你需要了解这方面内容)

处理器在内存中存储数据是不同的,这依赖于处理器的架构。内存中的数据存储有两种方法:一个叫大端(Big-Endian),另一个叫小端(Little-Endian)。Intel用的是小端,你必须适应这个,否则你会晕头转向的。举个例子:假定一个地址 7E04F172 (是一个 4 字节, 32 位数)。将其按字节拆分,会得到 7E、04、F1、72。现在,人们可能会认为将这些字节存储在内存(假定地址是 1000)中时应该是这样的:

1000::7E

1001::04

1002::F1

1003::72

任何正常人都这样想。但是 Intel 的开发人员比我们这些普通人更聪明，他们决定以一种更加符合逻辑的方法来存储：

1000::72

1001::F1

1002::04

1003::7E

上面的第一个例子是大端序列，意思是数字的最大的那端（以十进制序列形式）在内存中最先被存储。因为 7E000000 比 040000 大，所以第一个字节被存储在第一个位置，第二个字节被存储在第二个位置，以此类推。第二个例子（明显更加的聪明）叫做小端序列，意思是首选存储最小的字节（案例中是 4 号字节），后面依次是第三个字节、第二个字节、第一个字节。因为 72 小于 F100，所以会被先存储。

当你在内存中从一边往另一边看的时候，就会发现用小端而不是它大哥真的很天才。在大端中，数字 7E04F172 看起来像这样：

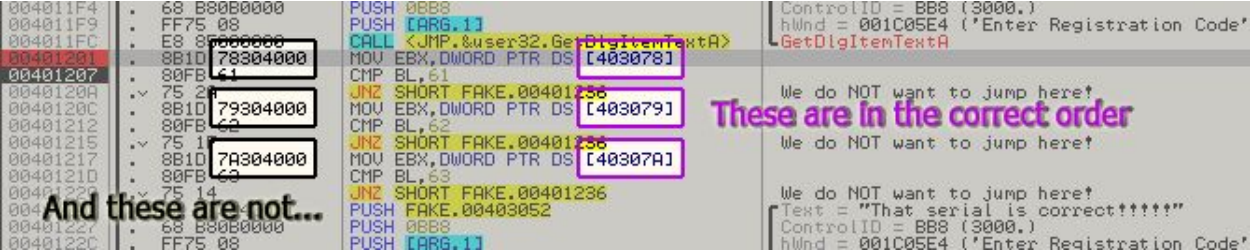
7E04F172

明显很乱。感谢上帝，使用小端的话，同样的数字 7E04F172 看起来更具有逻辑性：

72F1047E

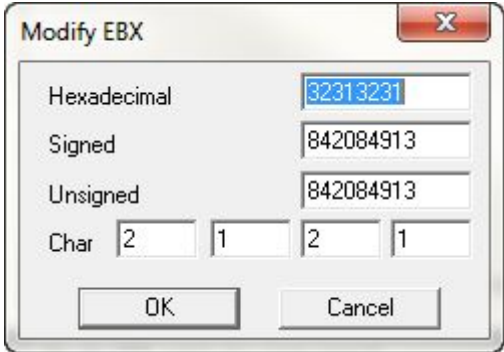
你说啥？蠢的太明显了吧，大端更合理吧。但话又说回来，你又不是 Intel 的半人半神的开发者，所以你甚至不具备弄明白为什么这要优越得多的脑力。无论如何，先不管各种讽刺，这意

味着当你看代码时，无论是磁盘里的还是内存里的，你必须将 4 字节数字反过来。当然，011y 有时已经为你做了这些，这让情况变的更糟了，就像下面的图片这样：



到目前为止，这是我想要说的全部。不过，过会我就会告诉你字节序。

现在，回到我们的寄存器窗口：



注意，十六进制的是小端序列(应该是 31323132), 那个 Char 是向后的，因为我的序列号是以 1212 开头的，而不是 2121。相信我，你会用到这些的。

现在看看下一条指令：

CMP BL, 61

这是一个很明显的比较语句，比较 BL 的值，也就是 EBX 寄存器的第一个字节与 61 (hex) 进行比较。我们真的没什么线索来了解这是什么意思，所以我们单步步过它。最后我们来到了第一条 JNZ 指令：

JNZ SHORT FAKE.401236

这里我们回想一下，我们可以看到我们前面做的注释，就是我们不想让这个跳转实现。这里提醒一下，JNZ 的意思是非 0 的时候跳转。所以，这两行的意思是“如果 BL 的值不等于 61h，就跳转到坏消息”。我们可以清楚的看到 EBX 寄存器的右边的字节（BL）不是 61h，而是 31h。我们已经卡在这了，那个跳转会实现的，但是我们又非常的不想要它实现😞。

等等！Ollly 是一个“动态”的调试器，我们应该可以动态的实现跳转！好吧，因为你很可能已经读了汇编语言书籍中关于标志位的整个章节，所以我不准备讨论这个。

五、CPU 标志位

前面的章节中我们简要的讨论了标志位，我也确实不准备深入的探讨这个问题，因为我确信你的汇编语言书籍的目录中有一个“F”章节。标志位可以让处理器知道某条指令的输出是什么。在 Intel 库中有大量的指令可以影响到标志位，不过最重要（至少对于逆向来说）的是“compare(比较)”指令。基本上，CPU 比较两个项目之后，会根据它们的相互关系属性（相同？一个大？一个小？）来设置标志位，再根据这些标志位来执行相应的跳转语句。这其实是表达 IF THEN 语句的非常奇特的方式。例如，在高级语言中有如下代码：

```
if( serialNumber == 3 )
    dontShowNag();
else
    showNag();
```

用伪汇编语言来表示，同样的指令应该类似下面的代码：

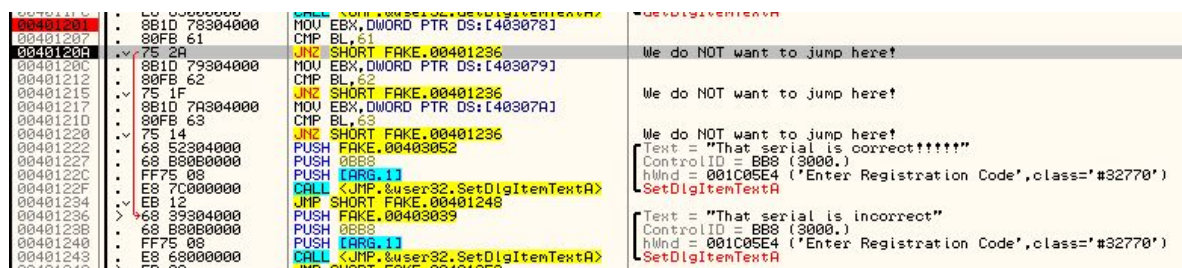
```
compare serialNumber with 3
    jump (if they are equal) to dontShowNag();
    jump to showNag();
```

用真正的汇编表示有可能像这样：

MOV EAX, addressOfSerialNumber CMP EAX, 3 JE addressOfDontShowNag JMP adressOfShowNag

首先，EAX 中存储着我们的序列号。下一步，它和“3”进行比较。如果等于 3 就跳到 dontShowNag()。如果不等于 3，就跳过 JE（如果相等就跳转——jump if equal）指令，执行 JMP（JuMP）指令。不管任何标志位，自动跳到 showNag()。

重要的标志位（对于我们来说）有 0 标志位和进位标志位，在 011y 中分别显示为“Z”和“C”。基本上，通过修改两个标志位中的一个，我们就可以阻止（或者强制）程序中的任何跳转，



```
00401207: 8B1D 78304000 MOV EBX,DWORD PTR DS:[403078]
00401208: 80FB 61 CMP BL,61
0040120A: 75 2A JNZ SHORT FAKE.00401236
0040120C: 8B1D 79304000 MOV EBX,DWORD PTR DS:[403079]
00401212: 80FB 62 CMP BL,62
00401215: 75 1F JNZ SHORT FAKE.00401236
00401217: 8B1D 7A304000 MOV EBX,DWORD PTR DS:[40307A]
0040121D: 80FB 63 CMP BL,63
00401220: 75 14 JNZ SHORT FAKE.00401236
00401222: 68 52304000 PUSH FAKE.00403052
00401227: 68 B8B00000 PUSH 0BB8
0040122C: FF75 08 CALL <JMP.>user32.SetDlgItemTextA
0040122F: EB 12 JMP SHORT FAKE.00401248
00401234: 68 39304000 PUSH FAKE.00403039
00401238: 68 B8B00000 PUSH 0BB8
00401240: FF75 08 CALL <JMP.>user32.SetDlgItemTextA
00401243: EB 08 JMP SHORT FAKE.00401255
```

就像我们下面要介绍的：

在暂停的那行（第一个 JNZ），通过那个红色的箭头，我们可以看到 011y 准备执行这个跳转。如果该跳转不会被执行，这条线就会显示灰色。如果你没有用我所用的 011y，就不会有这个箭头，这样的话你可以看反汇编窗口和数据窗口中间的那一块，011y 会告诉你跳转会不会被执行。本例中，会有如下显示：



```
00401253: 68 01000000 MOV EAX,DWORD PTR DS:[00000001]
00401258: C9 LEAVE
00401259: C2 1000 RETN
0040125C: FF25 58204000 JMP 58204000
00401262: FF25 5A204000 JMP 5A204000
```

Jump is taken
00401236=FAKE.00401236

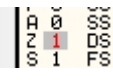
Address	Hex	dump
00403078	31 32 31 32 31 32 31 32 31	
00403088	31 32 00 00 00 00 00 00 00	
00403098	00 00 00 00 00 00 00 00 00	
004030A8	00 00 00 00 00 00 00 00 00	
004030B8	00 00 00 00 00 00 00 00 00	

现在我们知道，如果不做点什么的话，011y 就会执行该跳转。那我们就干点什么吧。看看寄存器窗口，找到“Z”标志位：

```

EBP 0018F8C4
ESI 00401178 FAKE.00401178
EDI 00000000
EIP 0040120A FAKE.0040120A
C 1 ES 002B 32bit 0(FFFFFFFF)
P 0 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 0 DS 002B 32bit 0(FFFFFFFF)
S 1 FS 0053 32bit 7EFD0000(FFF)
T 0 GS 002B 32bit 0(FFFFFFFF)
O 0

```

注意那有个 0。意思是，在 61h 和 BL 的内容 (31h) 之间的比较结果是 0，或者叫 false，所以它们不相等。现在我们明白了为什么 不是 0 就跳转 指令会跳转了，因为就目前来说，0 标志位没有被置位，所以它是“非 0”。现在，双击零标志位后面的那个 0，它就会变成一个 1：。然后注意看那个箭头，变成灰色了 (OllDb 也会提示跳转不成立)：

00401207	80FB 61	CMP BL, 61	
0040120A	75 2A	JNZ SHORT FAKE.00401236	We do NOT want to jum
0040120C	8B1D 79304000	MOV EBX, DWORD PTR DS:[403079]	
00401212	80FB 62	CMP BL, 62	
00401215	75 1F	JNZ SHORT FAKE.00401236	We do NOT want to jum
00401217	8B1D 7A304000	MOV EBX, DWORD PTR DS:[40307A]	
0040121D	80FB 63	CMP BL, 63	
00401220	75 14	JNZ SHORT FAKE.00401236	We do NOT want to jum
00401222	68 52304000	PUSH FAKE.00403052	Text = "That serial i
00401227	68 B80B0000	PUSH 0BB8	ControlID = BB8 (3000
0040122C	FF75 08	PUSH [ARG.1]	hWnd = 001C05E4 ('Ent
0040122F	E8 7C000000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
00401234	EB 12	JMP SHORT FAKE.00401248	
00401236	68 39304000	PUSH FAKE.00403039	Text = "That serial i
0040123B	68 B80B0000	PUSH 0BB8	ControlID = BB8 (3000
00401240	FF75 08	PUSH [ARG.1]	hWnd = 001C05E4 ('Ent
00401243	E8 68000000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
00401248	EB 09	JMP SHORT FAKE.00401253	

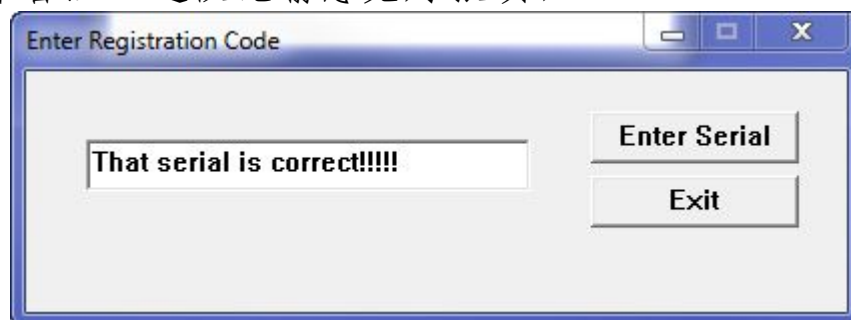
我们已经修改了 OllDb 的标志位，同时我们也修改了程序的行为 😊。大哥继续，按下 F8 (你已经学会了)，我们不会执行该跳转: 0。我们现在来到了看起来一样的代码段，除了 EBX 中存储的是我们序列号的第二个字符，它将会与 62h 进行比较而不是 61h:

JNZ SHORT FAKE.00401236	We do NOT want to jump here!
MOV EBX, DWORD PTR DS:[403079]	
CMP BL, 62	
JNZ SHORT FAKE.00401236	We do NOT want to jump here!
MOV EBX, DWORD PTR DS:[40307A]	

我们知道我们序列号的第二个数字并不是 62h，现在知道该怎么做了吧。F8 直到 JNZ 语句，双击零标志位，继续下去!!! 你会跳过那个 JNZ 指令。快要成功了!!! 最后一个是将我们序列号的第三个数字与 63h 进行比较。我们序列号的第三个数字是 31h，所以该跳转正常来说是要执行的。继续，你知道该怎么做的。跳过了第三个跳转，我们来到了 401222:

00401220	75 14	JNZ SHORT FAKE.00401236	We do NOT want to jump here!
00401222	68 52304000	PUSH FAKE.00403052	Text = "That serial is correct!!!!"
00401227	68 B8000000	PUSH 0BB8	ControlID = BB8 (3000.)
0040122C	FF75 08	PUSH [ARG.1]	hWnd = 001C05E4 ('Enter Registration Code',class='#32770')
0040122F	E8 7C000000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
00401234	E8 12	JMP SHORT FAKE.00401248	Text = "That serial is incorrect"
00401236	68 39304000	PUSH FAKE.00403053	ControlID = BB8 (3000.)
0040123B	68 B8000000	PUSH 0BB8	hWnd = 001C05E4 ('Enter Registration Code',class='#32770')
00401240	FF75 08	PUSH [ARG.1]	SetDlgItemTextA
00401243	E8 68000000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA

你的心是不是开始扑通扑通的了，因为我认为我们都知道接下来会发生什么。在我们和救世主之间再也没有跳转了，所以无论你是单步步过下面的指令（如果你喜欢留悬念的话）还是直接运行程序（如果你和我一样不能忍受悬念的话），我们终会到达天堂（译者注：这段比喻感觉好猥琐）：



六、家庭作业

我知道你不喜欢，这一章已经让大家很兴奋了，不过我会以两件事来结束本章。第一个是：

R4ndom' s Essential Truths About Reversing Data:

R4ndom 关于逆向数据的必备真理：

#3：仅仅读教程，你是学不会逆向工程的。你必须亲自操作，而且必须大量的实践。

根据新规则，我会留一些作业。你的任务，你应该已经接受了，找出序列号。意思是，在让 JNZ 跳转不实现的情况下，你必须在序列号文本框输入的内容是什么？在不以任何方式修改应

用程序的情况下，在输入正确的序列号以后，程序显示了 “That Serial is Correct!!!!!!”，你就知道你找对了。

ps. 如果你需要提示，你可以点击这个[链接](#)。