

第十三章：破解一个真正的程序

一、简介

本章我们打算不训练了，咱们来破解一个真正的程序。这个程序有个时间限制，过了这个时间，这个程序就不能用了。我们准备给它打补丁，让它认为是注册过的。目标文件在下载中有（我没有提及程序的名字，因为教程的目的不是为了拿到一个“破解版”程序，只是为了学习）。与所有的商业软件一样，如果你真的打算用它们，你真的应该考虑购买它。人们在软件中投入了大量的时间，他们应该得到补偿。为了不让这个系列教程成为关于“获得破解版软件”的东东，我试着找了一个没有人真想要的程序，所以我下载了这个软件，它是上周 **Download.com** 中拥有最少下载量的软件。作为一个完全诚实的人，在本章中破解了这个程序以后，我很喜欢这个程序，所以我买了一个注册码，现在在我心安理得的用它（译者注：作者真是活雷锋，其实咱们都是搞技术的，或多或少都写过代码，尊重软件作者，为他们的劳动付费，其实就是尊重自己。实在不愿意花钱的，就用免费替代软件行了，我一般喜欢用开源免费软件。多说了几句哈）。只是告诉你，你 cannot 通过下载量来判断一个应用。

你可以在[教程](#)页下载相关文件及本文的 PDF 版。

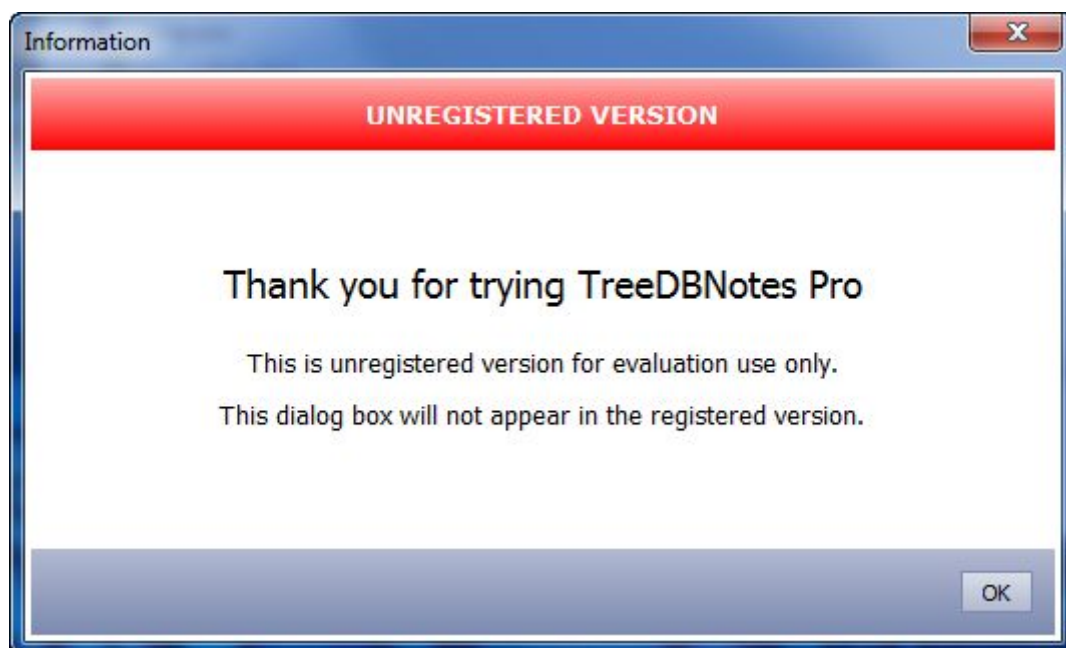
好，咱们继续...

二、研究该应用

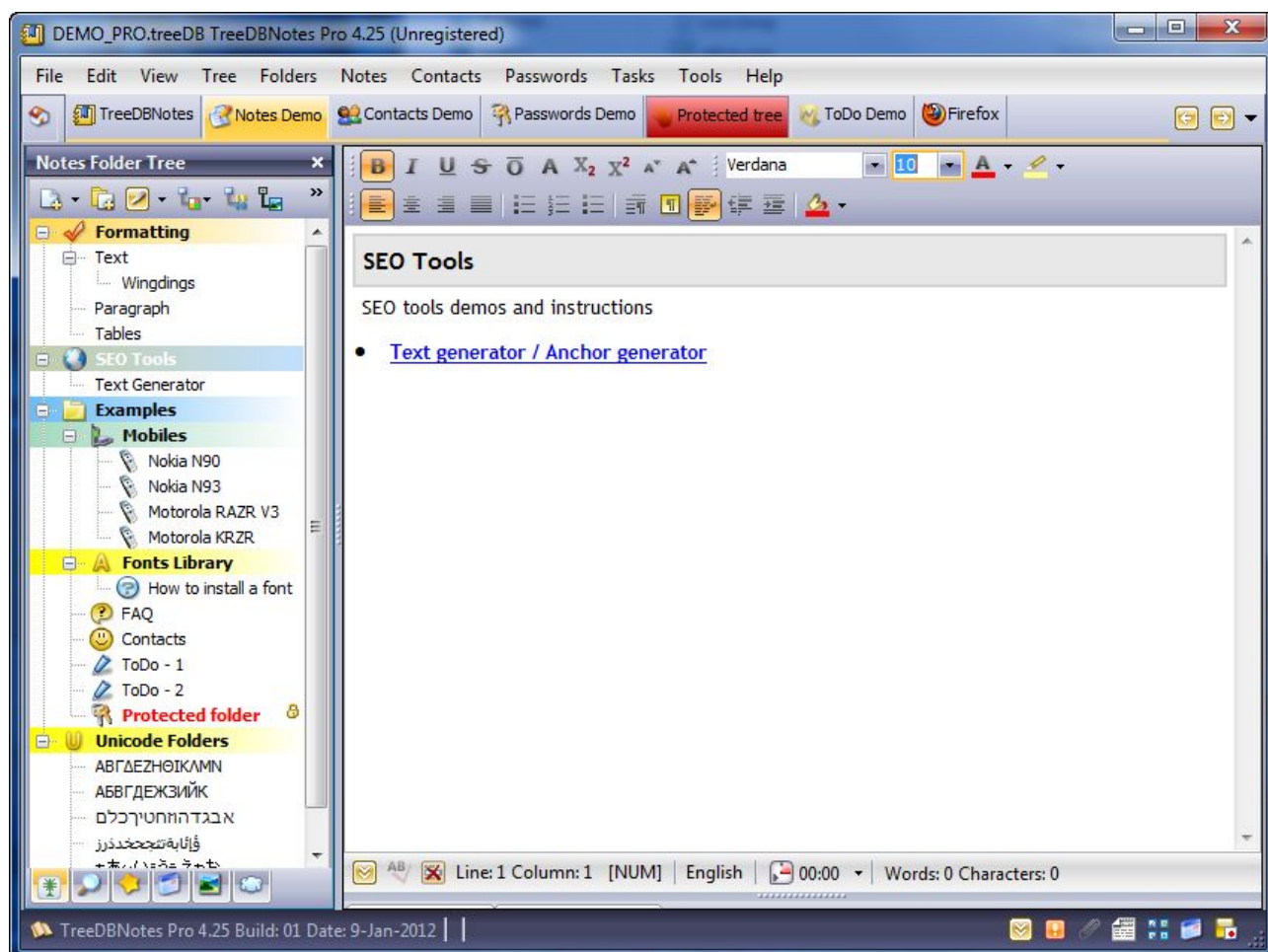
首先安装软件。安装完成后，会弹出下面这个窗口：



让“Run the app”保持勾选状态，看看会遇到什么：



好吧，看起来不是很好啊。我们注意到这里有几个字符串可能会有帮助，“unregistered”、“evaluation”、“registered”等等。点 OK，然后弹出主界面：

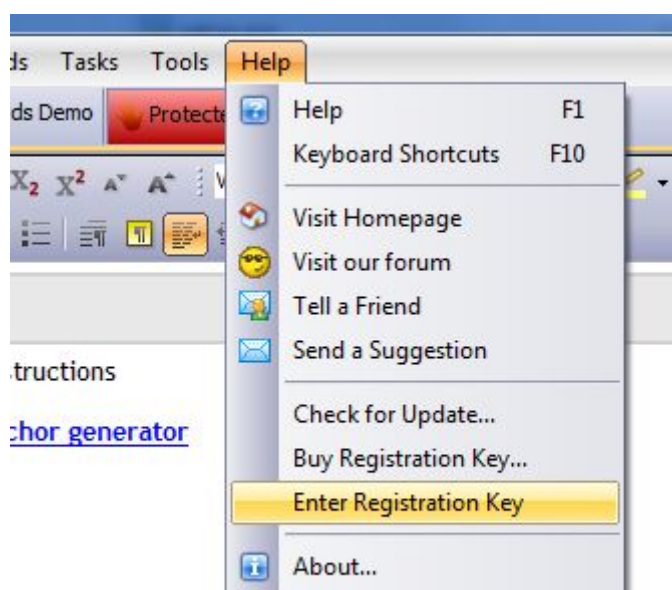


注意，标题栏中显示的是“unregistered”。通常，我注意一个程序的另一个地方就是它的关于对话框。它通常都包含有字符串，以及用于逆向的思路。

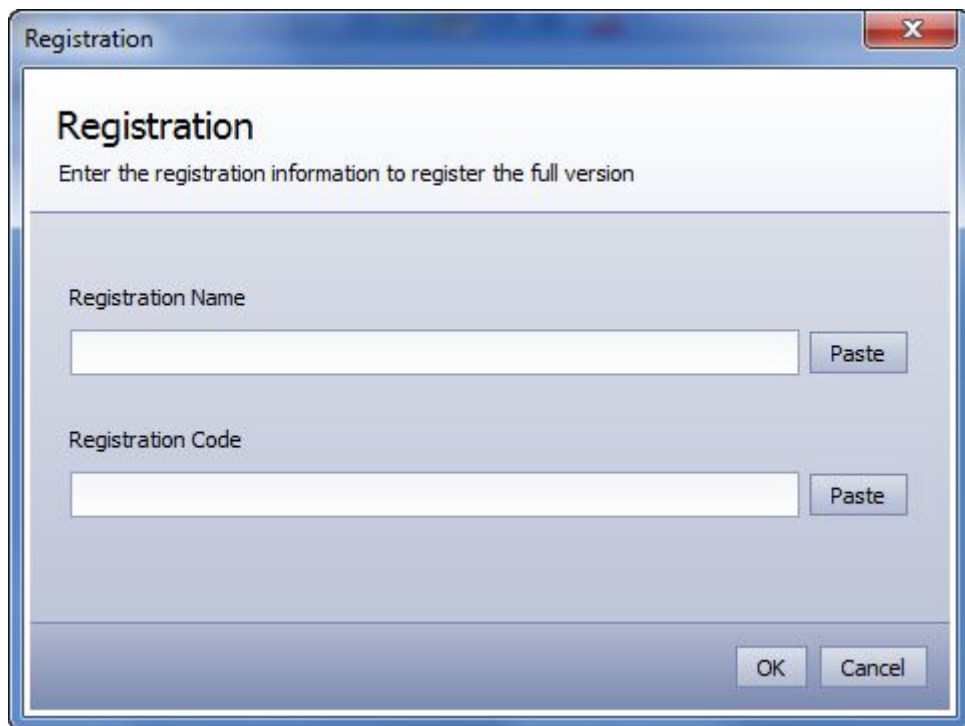
这时候，我们寻找关键字、可识别的方法调用，以及类似的东西。这样的工作你做的越多，就会有更多的线索。



这里我们又看见了“unregistered”。我通常做的下一件事是，找找看有没有什么地方用来输入注册码。如果“搜索字符串”这招不好用的话，那么对于渗透来说这是一个好入手点：



下面是输入注册码的地方：



Registration

Enter the registration information to register the full version

Registration Name

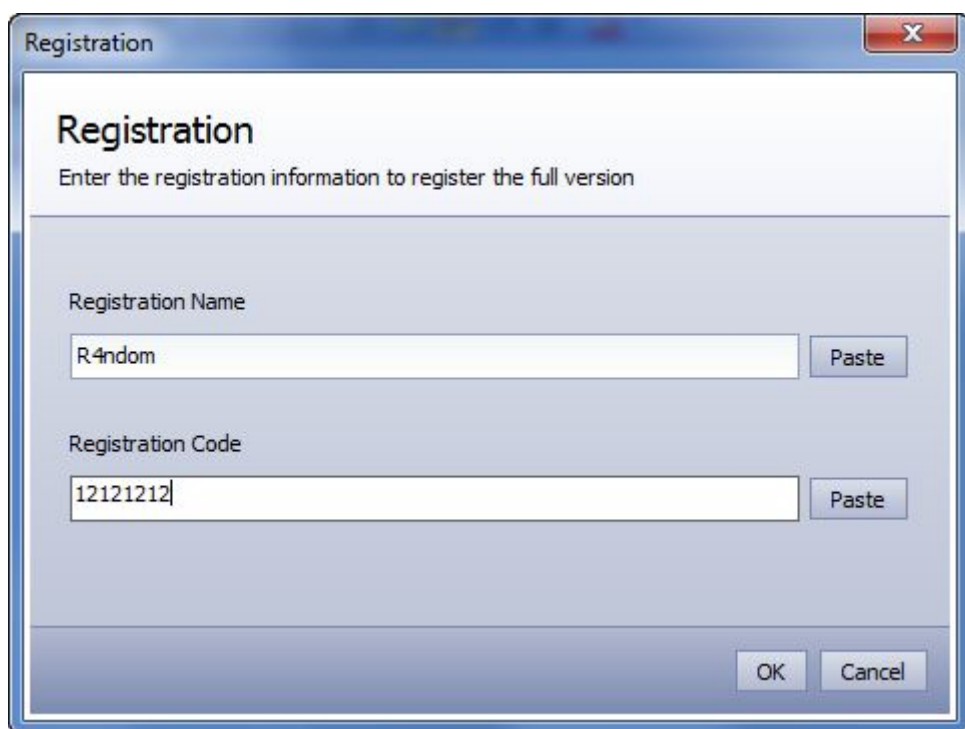
Paste

Registration Code

Paste

OK Cancel

输入一个试试，看看什么情况：



Registration

Enter the registration information to register the full version

Registration Name

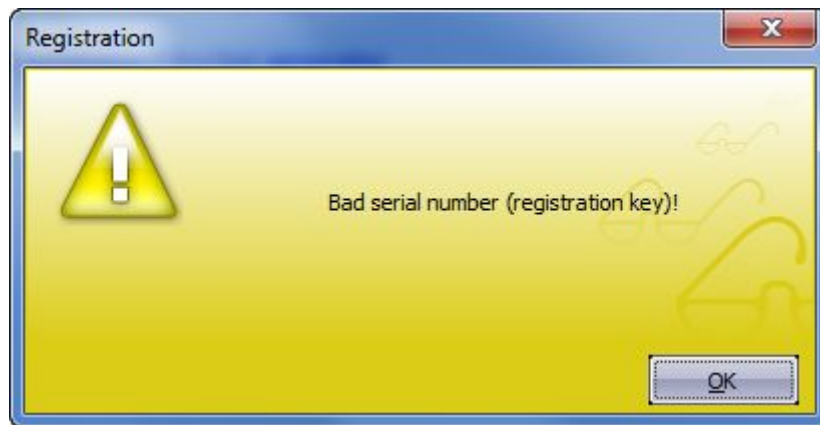
Paste

Registration Code

Paste

OK Cancel

点击 OK:



唉！我好像从来就没有输对过😅。好吧，对于我们当前搜集到的信息来说，我们有一个相当好的方法，Ollly 载入程序：

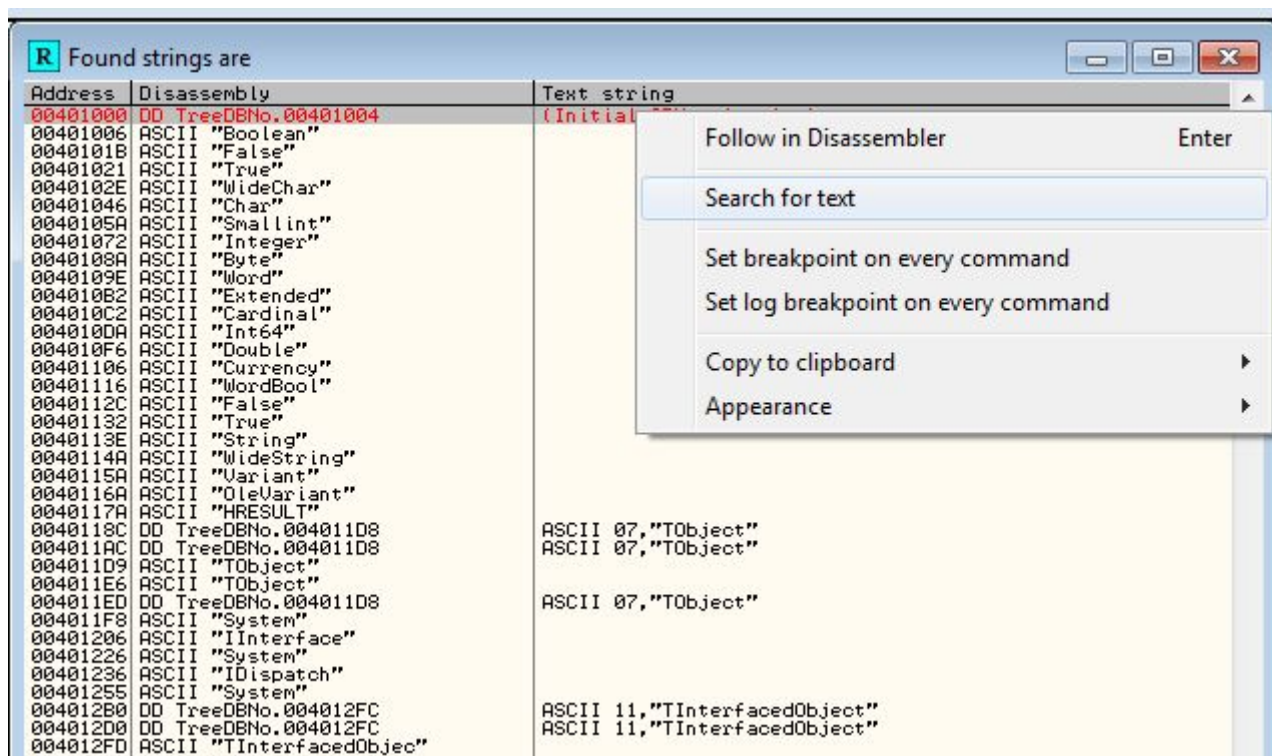
009F6098	\$ 55	PUSH EBP	
009F6099	8BEC	MOV EBP,ESP	
009F609B	83C4 F0	ADD ESP,-10	
009F609E	53	PUSH EBX	
009F609F	B8 204F9F00	MOV EAX,TreeDBNo.009F4F20	
009F60A4	E8 CB16A1FF	CALL TreeDBNo.00407774	
009F60A9	8B1D 0006A200	MOV EBX,DWORD PTR DS:[A206D0]	TreeDBNo.00A21BF8
009F60AF	33C9	XOR ECX,ECX	
009F60B1	B2 01	MOV DL,1	
009F60B3	A1 7CE29700	MOV EAX,DWORD PTR DS:[97E27C]	
009F60B8	E8 275EAAFF	CALL TreeDBNo.0049BEE4	
009F60BD	8B15 5405A200	MOV EDX,DWORD PTR DS:[A20554]	TreeDBNo.00A26764
009F60C3	8902	MOV DWORD PTR DS:[EDX],EAX	kernel32.BaseThreadInit
009F60C5	A1 5405A200	MOV EAX,DWORD PTR DS:[A20554]	
009F60CA	8B00	MOV EAX,DWORD PTR DS:[EAX]	
009F60CC	E8 68A1AAFF	CALL TreeDBNo.004A023C	
009F60D1	A1 5405A200	MOV EAX,DWORD PTR DS:[A20554]	
009F60D6	8B00	MOV EAX,DWORD PTR DS:[EAX]	
009F60D8	8B10	MOV EDX,DWORD PTR DS:[EAX]	
009F60DA	FF92 80000000	CALL DWORD PTR DS:[EDX+80]	
009F60E0	8B03	MOV EAX,DWORD PTR DS:[EBX]	
009F60E2	E8 45D8AAFF	CALL TreeDBNo.004A392C	
009F60E7	8B03	MOV EAX,DWORD PTR DS:[EBX]	
009F60E9	BA CC619F00	MOV EDI,TreeDBNo.009F61CC	ASCII "TreeDBNotes"
009F60EE	E8 31D4AAFF	CALL TreeDBNo.004A3524	
009F60F3	8B00 7C09A200	MOV ECX,DWORD PTR DS:[A2097C]	TreeDBNo.00A2632C
009F60F9	8B03	MOV EAX,DWORD PTR DS:[EBX]	
009F60FB	8B15 04AF8E00	MOV EDX,DWORD PTR DS:[8EAFD4]	TreeDBNo.008EB020
009F6101	E8 3ED8AAFF	CALL TreeDBNo.004A3944	
009F6106	8B00 0CFFA100	MOV ECX,DWORD PTR DS:[A1FF0C]	TreeDBNo.00A26874
009F610C	8B03	MOV EAX,DWORD PTR DS:[EBX]	
009F610E	8B15 98229D00	MOV EDI,DWORD PTR DS:[9D2298]	TreeDBNo.009D22E4
009F6114	E8 2B08AAFF	CALL TreeDBNo.004A3944	
009F6119	8B00 0000A200	MOV ECX,DWORD PTR DS:[A20000]	TreeDBNo.00A267A4
009F611F	8B03	MOV EAX,DWORD PTR DS:[EBX]	
009F6121	8B15 00999800	MOV EDI,DWORD PTR DS:[9899D0]	TreeDBNo.00989A1C
009F6127	E8 18D8AAFF	CALL TreeDBNo.004A3944	
009F612C	8B00 F800A200	MOV ECX,DWORD PTR DS:[A200F8]	TreeDBNo.00A248F0
009F6132	8B03	MOV EAX,DWORD PTR DS:[EBX]	
009F6134	8B15 58148B00	MOV EDI,DWORD PTR DS:[8B1458]	TreeDBNo.008B14A4
009F613A	E8 05D8AAFF	CALL TreeDBNo.004A3944	
009F613F	8B00 1403A200	MOV ECX,DWORD PTR DS:[A20314]	TreeDBNo.00A263E4
009F6145	8B03	MOV EAX,DWORD PTR DS:[EBX]	
009F6147	8B15 28FB9100	MOV EDI,DWORD PTR DS:[91FB28]	TreeDBNo.0091FB74
009F614D	E8 F2D7AAFF	CALL TreeDBNo.004A3944	
009F6152	8B00 5802A200	MOV ECX,DWORD PTR DS:[A20258]	TreeDBNo.00A263F0
009F6158	8B03	MOV EAX,DWORD PTR DS:[EBX]	
009F615A	8B15 B8019200	MOV EDI,DWORD PTR DS:[9201B8]	TreeDBNo.00920204
009F6160	E8 DF07AAFF	CALL TreeDBNo.004A3944	
009F6165	8B00 5CFFA100	MOV ECX,DWORD PTR DS:[A1FF5C]	TreeDBNo.00A26360
009F616B	8B03	MOV EAX,DWORD PTR DS:[EBX]	
009F616D	8B15 ACE78E00	MOV EDI,DWORD PTR DS:[8EE7AC]	TreeDBNo.008EE7F8
009F6173	E8 CCD7AAFF	CALL TreeDBNo.004A3944	
009F6178	8B00 2802A200	MOV ECX,DWORD PTR DS:[A20228]	TreeDBNo.00A26358
009F617E	8B03	MOV EAX,DWORD PTR DS:[EBX]	
009F6180	8B15 A8E68E00	MOV EDI,DWORD PTR DS:[8EE6A8]	TreeDBNo.008EE6F4
009F6186	E8 B9D7AAFF	CALL TreeDBNo.004A3944	
009F618B	8B00 340BA200	MOV ECX,DWORD PTR DS:[A20B34]	TreeDBNo.00A2676C
009F6191	8B03	MOV EAX,DWORD PTR DS:[EBX]	
009F6193	8B15 E8E99700	MOV EDI,DWORD PTR DS:[97EAE8]	TreeDBNo.0097EB34
009F6199	E8 A6D7AAFF	CALL TreeDBNo.004A3944	
009F619E	A1 0000A200	MOV EAX,DWORD PTR DS:[A20000]	
009F61A3	8B00	MOV EAX,DWORD PTR DS:[EAX]	
009F61A5	E8 1E71FAFF	CALL TreeDBNo.0099D2C8	
009F61AA	A1 5405A200	MOV EAX,DWORD PTR DS:[A20554]	
009F61AF	8B00	MOV EAX,DWORD PTR DS:[EAX]	
009F61B1	E8 7EA0AAFF	CALL TreeDBNo.004A0234	
009F61B6	8B03	MOV EAX,DWORD PTR DS:[EBX]	
009F61B8	E8 07D800FF	CALL TreeDBNo.00403904	

你可能已经注意到了，这看起来和我们已经见过的大部分应用有点不一样。看起来有辣么多的 CALL 指令，没有那些典型的 Windows 设置的玩意儿（像

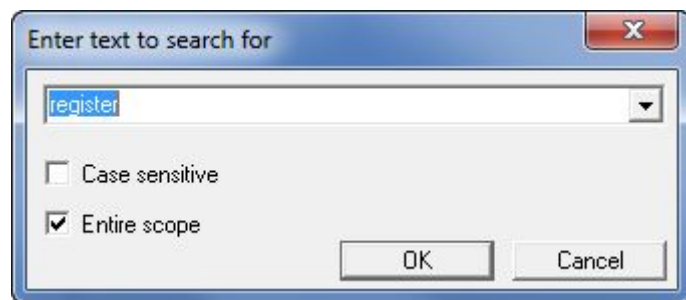
RegisterClass...)。这是一个好的标志，说明程序是用 Delphi 写的。Delphi 在程序中会使用大量的 CALL。我们可以通过运行一个 ID 程序来确定，不过我打算在后面的教程中讨论。也有一些专门的工具用来处理 Delphi 程序，不过幸运星是本章我们不需要用专用工具（虽然我们会接触到它们😄）。

三、寻找补丁

试试咱们的字符串搜索。右键，选择“Search for” -> “All referenced text strings”，将会弹出搜索窗口。滚动到顶部然后右键，选择“Search for text”：



弹出文本搜索对话框。现在我们注意到“registration”和“registered”很早就用到了，所以咱们就搜它们。通常在这种情况下，因为是第一次搜索，我会搜“regist”，因为包含了这两个单词，而且也从来没有让我失望过（我猜没有多少程序会使用单词‘register’😄）。不要勾选‘Case sensitive’，选中‘Entire scope’，然后点 OK：



第一个命中的看起来没啥前途，按 CTRL+L 继续搜：

Found strings are		
Address	Disassembly	Text string
004A2002	MOV EDI,TreeDBNo.004A207C	ASCII "Default"
004A207C	ASCII "Default",0	
004A2334	ASCII "TApplication",0	
004A241C	PUSH TreeDBNo.004A2524	ASCII "MAINICON"
004A2524	ASCII "MAINICON",0	
004A2FAF	MOV EAX,TreeDBNo.004A32AC	ASCII "voltest3.dll"
004A2FD0	PUSH TreeDBNo.004A32BC	ASCII "RegisterAutomation"
004A32AC	ASCII "voltest3.dll",0	
004A32BC	ASCII "RegisterAutomati"	
004A32CC	ASCII "on",0	
004A3D90	ASCII ".,",0	
004A4E01	PUSH TreeDBNo.004A4E24	ASCII "User32.dll"
004A4E11	PUSH TreeDBNo.004A4E30	ASCII "SetLayeredWindowAttributes"
004A4E24	ASCII "User32.dll",0	
004A4E30	ASCII "SetLayeredWindow"	
004A4E40	ASCII "Attributes",0	
004A4F00	PUSH TreeDBNo.004A4F1C	ASCII "TaskbarCreated"
004A4F1C	ASCII "TaskbarCreated",0	
004A4F2C	ASCII "need dictionary",0	
004A4F3C	ASCII "stream end",0	
004A4F4C	ASCII "file error",0	
004A4F58	ASCII "stream error",0	
004A4F68	ASCII "data error",0	

注意，这次找到的就是我们第一次搜到的。因为第一次命中的是在字符串被压到堆栈的地方，第二次才是字符串“RegisterAutomation”在内存中的真实的数据。因为在第二列中没有指令所以可以分辨出来，反而有个 ASCII 字样。你遇到的大多数字符串都有两个版本，一个是字符串被访问的地方，另一个就是字符串真正所在的地方：

Found strings are		
Address	Disassembly	Text string
004A2002	MOV EDI,TreeDBNo.004A207C	ASCII "Default"
004A207C	ASCII "Default",0	
004A2334	ASCII "TApplication",0	
004A241C	PUSH TreeDBNo.004A2524	ASCII "MAINICON"
004A2524	ASCII "MAINICON",0	
004A2FAF	MOV EAX,TreeDBNo.004A32AC	ASCII "voltest3.dll"
004A2FD0	PUSH TreeDBNo.004A32BC	ASCII "RegisterAutomation"
004A32AC	ASCII "voltest3.dll",0	
004A32BC	ASCII "RegisterAutomati"	
004A32CC	ASCII "on",0	
004A3D90	ASCII ".,",0	
004A4E01	PUSH TreeDBNo.004A4E24	ASCII "User32.dll"
004A4E11	PUSH TreeDBNo.004A4E30	ASCII "SetLayeredWindowAttributes"
004A4E24	ASCII "User32.dll",0	
004A4E30	ASCII "SetLayeredWindow"	
004A4E40	ASCII "Attributes",0	
004A4F00	PUSH TreeDBNo.004A4F1C	ASCII "TaskbarCreated"
004A4F1C	ASCII "TaskbarCreated",0	
004A4F2C	ASCII "need dictionary",0	
004A4F3C	ASCII "stream end",0	
004A4F4C	ASCII "file error",0	
004A4F58	ASCII "stream error",0	
004A4F68	ASCII "data error",0	

如果你再按一次 CTRL+L，我们会遇到另一个没前途的字符串。一直按 CTRL+L 直到来到下面这个地方：

Found strings are		
Address	Disassembly	Text string
009A9F9F	PUSH TreeDBNo.009AA134	ASCII "sNotes"
009A9FAB	MOV EAX,TreeDBNo.009AA144	ASCII "IDFolder"
009A9FC4	MOV EDX,TreeDBNo.009AA158	ASCII "ID"
009A9FF1	MOV EDX,TreeDBNo.009AA164	ASCII "NoteData"
009AA040	MOV EDX,TreeDBNo.009AA134	ASCII "sNotes"
009AA134	ASCII "sNotes",0	
009AA144	ASCII "IDFolder",0	
009AA158	ASCII "ID",0	
009AA164	ASCII "NoteData",0	
009AABA9	MOV EDX,TreeDBNo.009AABCC	UNICODE "TreeDBNotes Pro 4.25 (Registered)"
009AABB8	MOV EDX,TreeDBNo.009AAC14	UNICODE "TreeDBNotes Pro 4.25 (Unregistered)"
009AABCC	UNICODE "TreeDBNo"	
009AABDC	UNICODE "tes Pro "	
009AABEC	UNICODE "4.25 (Re"	
009AABFC	UNICODE "gistered"	
009AAC0C	UNICODE ")",0	
009AAC14	UNICODE "TreeDBNo"	
009AAC24	UNICODE "tes Pro "	
009AAC34	UNICODE "4.25 (Un"	
009AAC44	UNICODE "register"	
009AAC54	UNICODE "ed)",0	
009AACD8	PUSH TreeDBNo.009AAD70	UNICODE ": "
009AD3A5	MOV EAX,TreeDBNo.009AD5A0	UNICODE ": "

这回看起来好多了。它将会在程序启动过程中的某个时刻出现，它会检测我们有没有注册，然后根据检测的结果来决定窗口的标题栏显示注册还是没注册。这是我们开始干活的好地方。双击有“registered”的那行，咱们就会跳到相应的代码那：

009AAB84	0F94C2	SETD DL	
009AAB87	8B83 20F0000	MOV EAX,DWORD PTR DS:[EBX+F20]	
009AAB92	E8 9A3BAEFF	CALL TreeDBNo.0048E72C	kernel32.762BED6C
009AAB93	5D	POP EBX	kernel32.762BED6C
009AAB94	C3	RETN	
009AAB95	8D40 00	LEA EAX,DWORD PTR DS:[EAX]	
009AAB98	55	PUSH EBP	
009AAB99	8BEC	MOV EBP,ESP	
009AAB9B	53	PUSH EBX	
009AAB9C	8BD0	MOV EBX,EDX	TreeDBNo.<ModuleEntryPoint>
009AAB9E	80B8 B8150000 00	CMP BYTE PTR DS:[EAX+15B8],0	
009AAB9F	74 0F	JE SHORT TreeDBNo.009AABB6	
009AAB97	8BC3	MOV EAX,EBX	
009AABA9	BA CCB3A00	MOV EDI,TreeDBNo.009AABCC	UNICODE "TreeDBNotes Pro 4.25 (Registered)"
009AABAE	E8 55A9A5FF	CALL TreeDBNo.00405508	kernel32.762BED6C
009AABB3	5B	POP EBX	kernel32.762BED6C
009AABB4	5D	POP EBP	
009AABB5	C3	RETN	
009AABB6	8BC3	MOV EAX,EBX	
009AABB8	BA 14AC3A00	MOV EDI,TreeDBNo.009AAC14	UNICODE "TreeDBNotes Pro 4.25 (Unregistered)"
009AABB0	E8 46A9A5FF	CALL TreeDBNo.00405508	kernel32.762BED6C
009AABC2	5B	POP EBX	kernel32.762BED6C
009AABC3	5D	POP EBP	kernel32.762BED6C
009AABC4	C3	RETN	
009AABC5	00	DB 00	
009AABC6	00	DB 00	
009AABC7	00	DB 00	
009AABC8	42	DB 42	CHAR 'B'
009AABC9	00	DB 00	
009AABCA	00	DB 00	
009AABCB	00	DB 00	
009AABCC	5400 7200 6500 65	UNICODE "TreeDBNo"	
009AABDC	7400 6500 7300 20	UNICODE "tes Pro "	
009AABEC	3400 2E00 3200 39	UNICODE "4.25 (Re"	
009AABFC	6700 6900 7300 74	UNICODE "gistered"	
009AAC0C	2900 0000	UNICODE ")",0	
009AAC10	46	DB 46	CHAR 'F'
009AAC11	00	DB 00	
009AAC12	00	DB 00	
009AAC13	00	DB 00	
009AAC14	5400 7200 6500 65	UNICODE "TreeDBNo"	
009AAC24	7400 6500 7300 20	UNICODE "tes Pro "	
009AAC34	3400 2E00 3200 39	UNICODE "4.25 (Un"	
009AAC44	7200 6500 6700 63	UNICODE "register"	
009AAC54	6500 6400 2900 00	UNICODE "ed)",0	
009AAC5C	55	PUSH EBP	
009AAC5D	8BEC	MOV EBP,ESP	
009AAC5E	5D	POP EBP	

首先我们能看到字符串是在 9AABA9 那，还能看到字符串存储在内存的 9AABCC 处。第二，要注意到是两个字符串是在同一个方法中，在它们的上面有个一个条件跳转。点击 9AABA5 处的条件跳转：

009AAB98	53	PUSH EBX	TreeDBNo.<ModuleEntryPoint>
009AAB9C	8BDA	MOV EBX,EDX	
009AAB9E	80B8 B8150000 00	CMP BYTE PTR DS:[EAX+15B8],0	
009AABA5	74 0F	JE SHORT TreeDBNo.009AABB6	
009AABA7	8BC3	MOV EAX,EBX	
009AABA9	BA CCAB9A00	MOV EDX,TreeDBNo.009AABCC	UNICODE "TreeDBNotes Pro 4.25 (Registered)"
009AABAE	E8 55A9A5FF	CALL TreeDBNo.00405508	
009AABB3	5B	POP EBX	kernel32.7747ED6C
009AABB4	5D	POP EBP	kernel32.7747ED6C
009AABB5	C3	RETN	
009AABB6	8BC3	MOV EAX,EBX	
009AABB8	BA 14AC9A00	MOV EDX,TreeDBNo.009AAC14	UNICODE "TreeDBNotes Pro 4.25 (Unregistered)"
009AABB0	E8 46A9A5FF	CALL TreeDBNo.00405508	
009AABC2	5B	POP EBX	kernel32.7747ED6C
009AABC3	5D	POP EBP	kernel32.7747ED6C
009AABC4	C3	RETN	
009AABC5	00	DB 00	
009AABC6	00	DB 00	
009AABC7	00	DB 00	
009AABC8	42	DB 42	CHAR 'B'

我们能够看到如果结果相等，就跳到“Unregistered”那里。很明显，不能让它跳。咱们在 JE 指令那设置一个 BP，启动应用：

009AAB94	C3	RETN	
009AAB95	8D40 00	LEA EAX,DWORD PTR DS:[EAX]	
009AAB98	55	PUSH EBP	
009AAB99	8BEC	MOV EBP,ESP	
009AAB9B	53	PUSH EBX	
009AAB9C	8BDA	MOV EBX,EDX	TreeDBNo.<ModuleEntryPoint>
009AAB9E	80B8 B8150000 00	CMP BYTE PTR DS:[EAX+15B8],0	
009AABA5	74 0F	JE SHORT TreeDBNo.009AABB6	
009AABA7	8BC3	MOV EAX,EBX	
009AABA9	BA CCAB9A00	MOV EDX,TreeDBNo.009AABCC	UNICODE "TreeDBNotes Pro 4.25 (Registered)"
009AABAE	E8 55A9A5FF	CALL TreeDBNo.00405508	
009AABB3	5B	POP EBX	kernel32.7747ED6C
009AABB4	5D	POP EBP	kernel32.7747ED6C
009AABB5	C3	RETN	
009AABB6	8BC3	MOV EAX,EBX	
009AABB8	BA 14AC9A00	MOV EDX,TreeDBNo.009AAC14	UNICODE "TreeDBNotes Pro 4.25 (Unregistered)"
009AABB0	E8 46A9A5FF	CALL TreeDBNo.00405508	
009AABC2	5B	POP EBX	kernel32.7747ED6C
009AABC3	5D	POP EBP	kernel32.7747ED6C
009AABC4	C3	RETN	
009AABC5	00	DB 00	
009AABC6	00	DB 00	
009AABC7	00	DB 00	

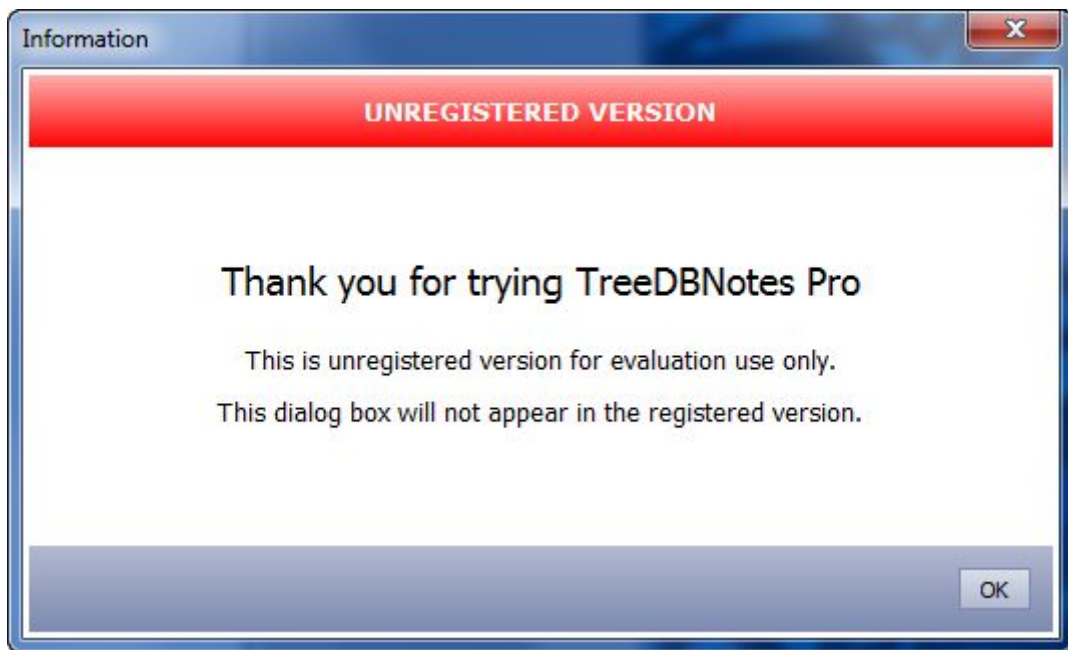
Oilly 就会断在那行，你会发现我们就要跳到坏消息那了。咱们得修改下：

```

C 0 ES 0023 32bi
P 1 CS 001B 32bi
A 0 SS 0023 32bi
Z 0 DS 0023 32bi
S 0 FS 003B 32bi
T 0 GS 0000 NULL
O 0 LastErr ERRO

```

运行程序。Oilly 会再次断在同一行，并准备跳到坏消息那。咱们再次就 0 标志位置 0，然后运行程序。又来了一遍，清除 0 标志位后，我们最终得到如下的反馈：



所以那样做是不起作用的。给那个检测点打补丁不会注册成功，如果你点 OK 并再一次将标志位置 0，你会发现主窗口的 “unregistered” 没有了：



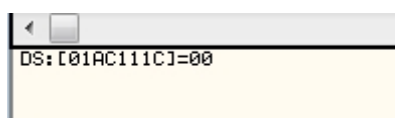
那么，我们知道至少没有跟踪错。我们准备做的是步入到下一“层”，做深入的研究。重启应用，然后断在了我们的断点处，咱们再多做些研究：

009AAB95	8D40 00	LEA EAX,DWORD PTR DS:[EAX]	
009AAB98	55	PUSH EBP	
009AAB99	8BEC	MOV EBP,ESP	
009AAB9B	53	PUSH EBX	
009AAB9C	8BDA	MOV EBX,EDX	
009AAB9E	80B8 B8150000 00	CMP BYTE PTR DS:[EAX+15B8],0	
009AAB9E	74 0F	JE SHORT TreeDBNo.009AABB6	
009AABA7	8BC3	MOV EAX,EBX	
009AABA9	BA CCAB9A00	MOV EDX,TreeDBNo.009AABCC	UNICODE "TreeDBNotes Pro 4.25 (Registered)"
009AABAE	E8 55A9A5FF	CALL TreeDBNo.00405508	
009AABB3	5B	POP EBX	042233A8
009AABB4	5D	POP EBP	042233A8
009AABB5	C3	RETN	
009AABB6	8BC3	MOV EAX,EBX	
009AABB8	BA 14AC9A00	MOV EDX,TreeDBNo.009AAC14	UNICODE "TreeDBNotes Pro 4.25 (Unregistered)"
009AABB0	E8 46A9A5FF	CALL TreeDBNo.00405508	
009AABC2	5B	POP EBX	042233A8
009AABC3	5D	POP EBP	042233A8
009AABC4	C3	RETN	
009AABC5	00	DB 00	
009AABC6	00	DB 00	

在比较的前面并没有 CALL，但是在 JE 指令的前面 9AAB9E 处有个比较：

CMP BYTE PTR DS: [EAX+15B8],0

所以，这个比较的结果决定了我们是注册还是未注册。EAX+15B8 只是一个内存地址，在这里它是一个全局变量，因为是以 DS: 打头的。我们希望这是程序检测注册与否的唯一一个检测点。如果不是的话，我们就需要找出其他检测点的位置。点选比较指令，就可以看到 EAX+15B8i 是什么：



在地址上右键，选择“Follow in dump”：

Address	Hex	dump	ASCII
01AC111C	00	00 00 00 00 FF FF FF FF 00 00 00 00 00 00 00 00
01AC112C	00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 000...0...
01AC113C	00	00 00 00 00 00 00 00 00 01 00 00 00 01 00 00 0023".....
01AC114C	0A	00 00 00 00 00 00 00 00 A8 33 22 04 00 00 00 002φ◆.....
01AC115C	00	00 00 00 00 E4 ED FE 04 00 00 00 00 00 00 00m\$◆ro\$◆
01AC116C	00	00 00 00 00 00 00 00 00 10 6D 24 04 F4 6F 24 04	±B%00- ◆4◆.....
01AC117C	08	42 AC 01 38 BF 20 04 34 C8 1E 04 00 00 00 00	%...&↓\$0"=\$0.900
01AC118C	26	00 00 00 88 19 A4 01 FC CD A4 01 0C 39 A7 01	◆L00dJ%0.....
01AC119C	04	C0 A6 01 64 FB AB 01 00 00 00 00 00 00 00&...0H0.dJ%0
01AC11AC	00	00 00 00 26 00 00 00 94 C7 47 00 64 FB AB 01H.dJ%0N...%zI.
01AC11BC	00	00 00 00 00 00 00 00 00 00 00 00 00 00 00	dJ%0[.0.....
01AC11CC	14	29 48 00 64 FB AB 01 4E 00 00 00 58 7A 49 000.....
01AC11DC	64	FB AB 01 08 00 5B 03 00 00 00 00 B1 03 00 000.....
01AC11EC	00	00 00 00 00 00 00 00 00 01 00 00 00 0A 00 000.....
01AC11FC	00	00 00 00 14 00 00 FF 01 00 00 00 00 00 000.....
01AC120C	00	00 00 00 00 00 00 00 00 0C 00 00 00 04 00 00N...%zI.dJ%0
01AC121C	00	00 00 00 4E 00 00 00 58 7A 49 00 64 FB AB 010.....
01AC122C	08	00 3D 02 00 00 00 73 02 00 00 00 00 00 000.....
01AC123C	01	00 00 00 01 00 00 0A 00 00 00 00 00 00 000.....

你的地址几乎肯定和我的不一样。这个没事。跟着教程，将我的地址替换成你的地址，一样跑的好好的。

这里我们能看到该地址被用于检测注册与否，就是 1AC111C 处（至少我这里是这样的）的第一个 00。意思是如果此内存位置的内容是除了 0 以外的任何数据，那么就假定我们已经注册。这也意味着，程序中可能有别的子程序检查该内存位置，这就是为什么主窗口显示“Registered”，而程序的其他部分知道我们没有注册。因为我们只是在检查了内存内容后绕过了这个子程序的自然流程，其他子程序的检测却没有绕过。

首先，咱们把这个内存地址设置为非 0，那么我们知道至少这个子程序将会按照我们想要的方式工作。在比较那行（9AAB9E）设置一个断点，将其他断点删掉。重启应用后 Olly 就断下来了。在比较那行上右键，选择“Follow in dump”->“Memory location”，因为 Olly 会在我们重启应用的时候重置数据窗口。你可能已经注意到了，比较指令检查的内存地址这次变了：

Address	Hex dump	ASCII
01B9111C	00 00 00 00 FF FF FF FF 00 00 00 00 00 00 00 00
01B9112C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01B9113C	00 00 00 00 00 00 00 00 01 00 00 00 01 00 00 000.....
01B9114C	0A 00 00 00 00 00 00 00 A8 33 22 04 00 00 00 00d3.....
01B9115C	00 00 00 00 E4 ED FE 04 00 00 00 00 00 00 00 0024.....
01B9116C	00 00 00 00 00 00 00 00 10 6D 24 04 F4 6F 24 04m\$4.....
01B9117C	08 42 B9 01 38 BF 20 04 34 C8 1E 04 00 00 00 00	8E08744=0.940
01B9118C	26 00 00 00 88 19 B1 01 FC CD B1 01 0C 39 B4 01	%...2400000.940
01B9119C	04 C0 B3 01 64 FB B8 01 00 00 00 00 00 00 00 00	% 0d70.....
01B911AC	00 00 00 00 26 00 00 00 94 C7 47 00 64 FB B8 01	...%...0170
01B911BC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01B911CC	14 29 48 00 64 FB B8 01 4E 00 00 00 58 7A 49 00	4)H.d70N...%zI.
01B911DC	64 FB B8 01 08 00 58 03 00 00 00 00 B1 03 00 00	d700.[]...s0...
01B911EC	00 00 00 00 00 00 00 00 01 00 00 00 0A 00 00 000.....
01B911FC	00 00 00 00 14 00 00 FF 01 00 00 00 00 00 00 000.....
01B9120C	00 00 00 00 00 00 00 00 0C 00 00 00 04 00 00 000.....
01B9121C	00 00 00 00 4E 00 00 00 58 7A 49 00 64 FB B8 01	...N...%zI.d700
01B9122C	08 00 3D 02 00 00 00 00 73 02 00 00 00 00 00 00	8.=0.....s0.....
01B9123C	01 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00	0.....

第一次是 1AC111C，现在是 01B9111C。你的和我的会不一样，你只需要注意第二次的就行，存储 已注册/未注册 标志的内存地址不同。

点击数据窗口中的“00”(在我的数据窗口中是 1B9111C)，右键选择“Binary”->“Edit”：

咱们输入 01：

注意数据窗口中的内容已经更新了：

Address	Hex dump	ASCII
01B9111C	01 00 00 00 FF FF FF FF 00 00 00 00 00 00 00 00	0...
01B9112C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01B9113C	00 00 00 00 00 00 00 00 01 00 00 00 01 00 00 000...0...
01B9114C	0A 00 00 00 00 00 00 00 A8 33 22 04 00 00 00 0003"0...
01B9115C	00 00 00 00 E4 ED FE 04 00 00 00 00 00 00 00 00Σφ◆...
01B9116C	00 00 00 00 00 00 00 00 10 6D 24 04 F4 6F 24 04m\$◆fo\$◆
01B9117C	08 42 B9 01 38 BF 20 04 34 C8 1E 04 00 00 00 00	тВ 08γ ◆4\$▲◆...
01B9118C	26 00 00 00 88 19 B1 01 FC CD B1 01 0C 39 B4 01	%....ē└┐0"=0.9I0
01B9119C	04 C0 B3 01 64 FB B8 01 00 00 00 00 00 00 00 00	◆└┐0d┐0.└┐0.└┐0
01B911AC	00 00 00 00 26 00 00 00 94 C7 47 00 64 FB B8 01&...0└┐G.d┐0
01B911BC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01B911CC	14 29 48 00 64 FB B8 01 4E 00 00 00 58 7A 49 00	т)H.d┐0N...XzI.
01B911DC	64 FB B8 01 08 00 5B 03 00 00 00 00 B1 03 00 00	d┐0C.C.└┐0.└┐0.
01B911EC	00 00 00 00 00 00 00 00 01 00 00 00 0A 00 00 000.....
01B911FC	00 00 00 00 14 00 00 FF 01 00 00 00 00 00 00 00т...0.....
01B9120C	00 00 00 00 00 00 00 00 0C 00 00 00 04 00 00 00◆.....
01B9121C	00 00 00 00 4F 00 00 00 58 7A 49 00 64 FB B8 01N...XzI.d┐0

继续运行程序直到再一次断下来。你会发现内存中的内容又变回了 0，我们将再一次跳到坏消息那。这意味着程序的某个地方，做了第二次检测并将注册与否的标志重置为 0。我们需要做的就是找出在哪里重置的，确保不会再被重置。要这样做的话，在该内存位置设置一个硬件断点，当程序向该内存位置写数据时让 Olly 断下来。之所以选择“写”，是因为某个地方向该内存写了 0。

重启应用直到它断下来。右键比较指令，选择“Follow in dump”，因为 Olly 又重置了数据窗口。用二进制编辑方法将第一个内存位置修改为 01。注意它现在的地址又换了：

009AAB5E

80B8 B8150000 00

CMP BYTE PTR DS:[EAX+15B8],0

009AAB5F

74 0F

JE SHORT TreeDBNo.009AAB66

009AAB67

8BC3

MOV EAX,EBX

009AAB68

BA CCAB9A00

MOV EDX,TreeDBNo.009AABCC

009AAB69

E8 55A9A5FF

CALL TreeDBNo.00405508

009AAB6A

5B

POP EBX

009AAB6B

5D

POP EBP

009AAB6C

C3

RETN

009AAB6D

8BC3

MOV EAX,EBX

009AAB6E

BA 14AC9A00

MOV EDX,TreeDBNo.009AAC14

009AAB6F

E8 46A9A5FF

CALL TreeDBNo.00405508

009AAB70

5B

POP EBX

009AAB71

5D

POP EBP

009AAB72

C3

RETN

009AAB73

00

DB 00

009AAB74

00

DB 00

009AAB75

00

DB 00

009AAB76

00

DB 00

009AAB77

42

DB 42

Unicode "TreeDBNotes"

01B8FB64

01B8FB64

Unicode "TreeDBNotes"

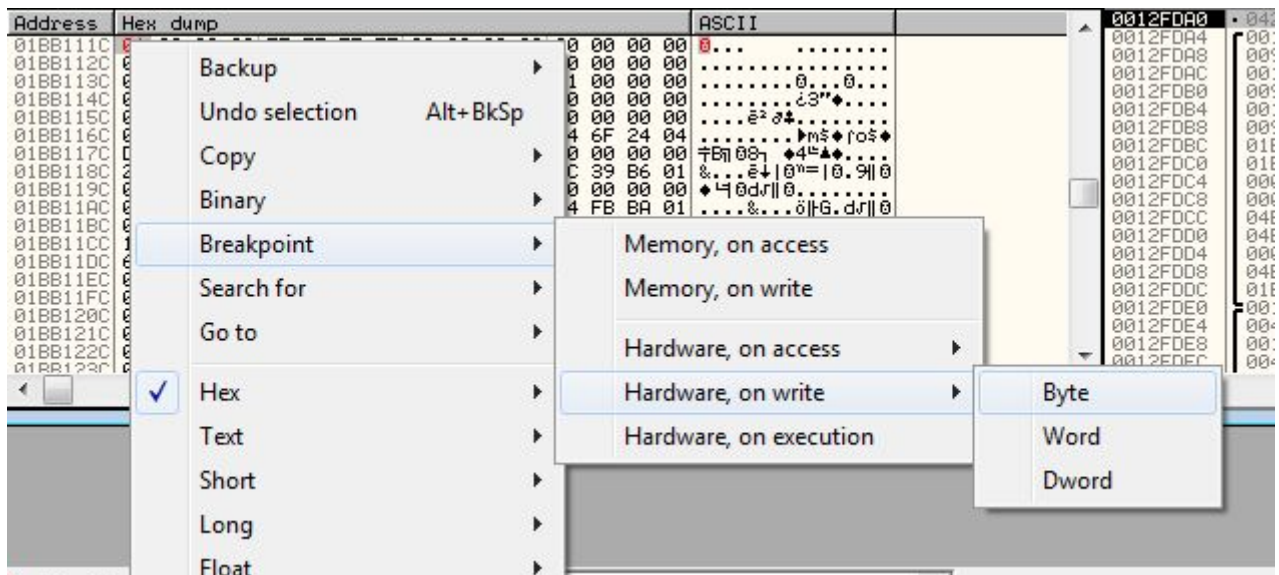
01B8FB64

01B8FB64

CHAR 'B'

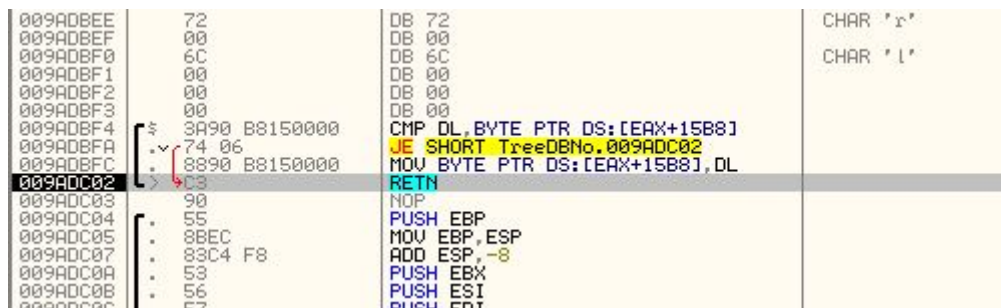
Address	Hex dump	ASCII
01B9111C	01 00 00 00 FF FF FF FF 00 00 00 00 00 00 00 00	0...
01B9112C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01B9113C	00 00 00 00 00 00 00 00 01 00 00 00 01 00 00 000...0...
01B9114C	0A 00 00 00 00 00 00 00 A8 33 22 04 00 00 00 0003"0...
01B9115C	01 00 00 00 E4 ED FE 04 00 00 00 00 00 00 00 00	0.....Σφ◆...
01B9116C	00 00 00 00 00 00 00 00 10 6D 24 04 F4 6F 24 04m\$◆fo\$◆
01B9117C	08 42 B9 01 38 BF 20 04 34 C8 1E 04 00 00 00 00	тВ 08γ ◆4\$▲◆...
01B9118C	8C 11 B9 01 8C 11 B9 01 24 00 00 00 0C 39 B4 01	└┐0└┐0└┐0\$...9I0
01B9119C	04 C0 B3 01 64 FB B8 01 C8 A8 01 05 2C A9 01 05	◆└┐0d┐0\$20\$..r0\$
01B911AC	24 00 00 00 27 00 00 00 94 C7 47 00 64 FB B8 01	\$...\$...0└┐G.d┐0
01B911BC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01B911CC	14 29 48 00 64 FB B8 01 4E 00 00 00 58 7A 49 00	т)H.d┐0N...XzI.
01B911DC	64 FB B8 01 08 00 5B 03 00 00 00 00 B1 03 00 00	d┐0C.C.└┐0.└┐0.
01B911EC	00 00 00 00 00 00 00 00 01 00 00 00 0A 00 00 000.....
01B911FC	00 00 00 00 14 00 00 FF 01 00 00 00 00 00 00 00т...0.....
01B9120C	00 00 00 00 00 00 00 00 0C 00 00 00 04 00 00 00◆.....
01B9121C	00 00 00 00 4E 00 00 00 58 7A 49 00 64 FB B8 01N...XzI.d┐0

右键数据窗口中的第一个值，选择“Breakpoint” -> “Hardware, on write” -> “byte”：



在逆向一个程序时，我通常留在硬件断点，因为它们很难被应用检测到。我选择“byte”是因为我们想追踪的就一个字节。

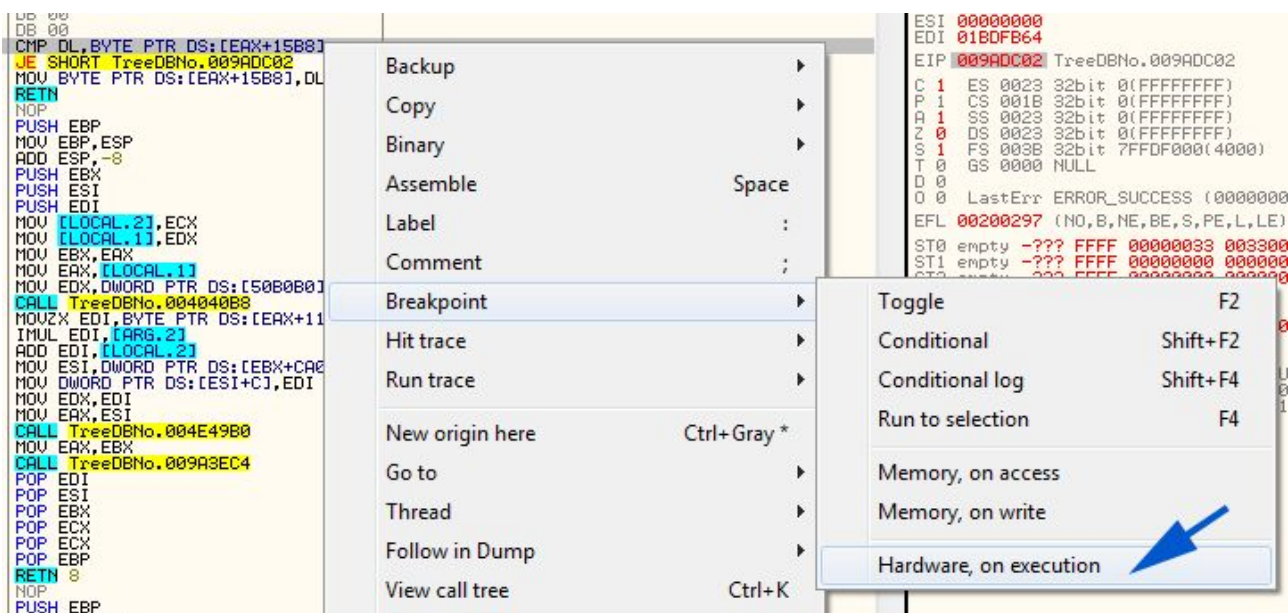
运行程序。Olly 会再次断在普通断点，你会发现我们输入的 01 仍然在那里，所以到目前为止还不错。再运行，Olly 会断在一个新地方：



如果你看 OllyDbg 左下角的话，会发现我们断在了硬件断点。

四、给程序打补丁

现在，咱们来研究研究这块代码。第一条指令是将 DL 与我们刚才编辑的内存内容进行比较，如果相等就跳到 9ADC02，然后就返回了。如果不相等，就将 DL 的内容存储到我们编辑的内存中。我们已经知道了 DL 等于 0，因为我们看到内存中的值从 01 变成了 00。所以这基本上就是另一个注册检测点，并且如果它检测失败就会将 已注册/未注册 标志置 0。如果成功，就什么都不做。现在咱们将硬件断点删掉，选择“Debug” -> “Hardware breakpoints”（译者注：这里的 Debug 指的是菜单中的），将硬件断点删除。咱们在 9ADBF4 处设置另一个硬件断点，这样我们可以在该段代码运行前断下来：



你或许会纳闷，我为什么不在这里设置一个普通断点。因为我先试过了！Oilly 根本就不会断下来好嘛！有几个愿意可能会导致该问题的发生：这段是多态代码，所以我们的 BP 丢了，程序检测到软件断点所以把它删了，断点在一个 Oilly 不会自动追踪的区块...。不管怎样，就是这么个结果。我们需要设置硬件断点而不是软件断点。不保证硬件断点一定管用，因为软件有可能专门对它们进行检测。不过它是一个更可靠的设置断点的方法，所以通常来说还是比较好用的。

在后面的章节中我们会更多的学习反调试技巧。

重启应用，我们会再次断在新的硬件断点处：

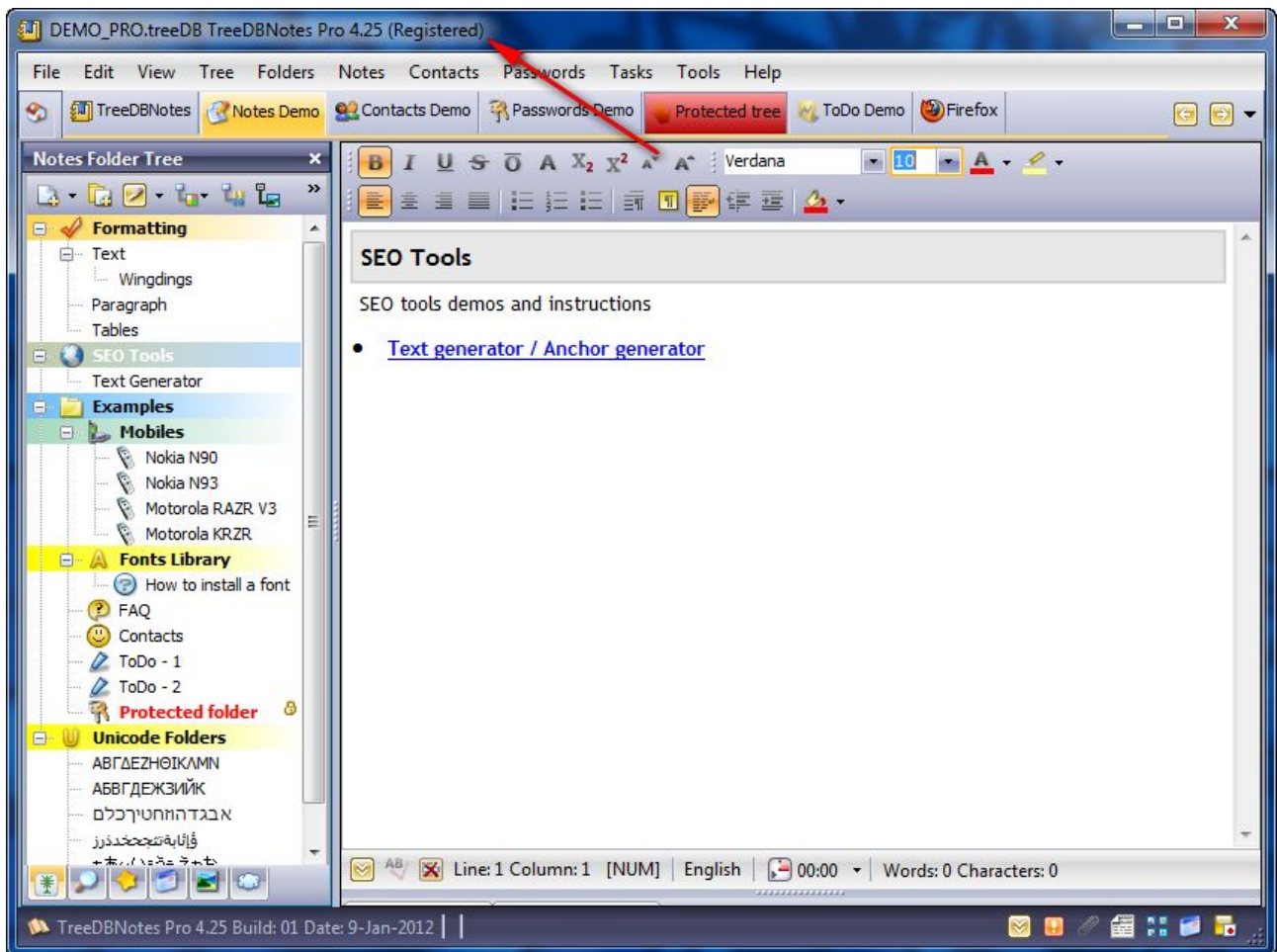
009ADB2	00	DB 00	
009ADB3	00	DB 00	
009ADB4	3A90 B8150000	CMP DL, BYTE PTR DS:[EAX+15B8]	
009ADB5	74 06	JE SHORT TreeDBNo.009ADC02	
009ADB6	8890 B8150000	MOV BYTE PTR DS:[EAX+15B8], DL	
009ADB7	C3	RETN	
009ADB8	90	NOP	
009ADB9	55	PUSH EBP	
009ADB0	8BEC	MOV EBP, ESP	

好，现在咱们来思考思考啊。这个子程序是在咱们原来的断点前面被调用。这个子程序检测我们是否注册，如果没有就将[EAX+15B8]地址处的内容设置为0，如果注册了就置为01（或者任何非0的数据）。然后我们原来的子程序被调用，就是那个在窗口标题中输出“Registered”或“Unregistered”的子程序，它也是根据内存中的数据是0还是1来决定输出。如果我们确保任何时候只要该子程序运行时那个内存位置中都是1，那么任何其他子程序来检测内存中内容时看到的都只能是1，也就是认为我们已经注册了。

如果我们只是将子程序修改成总是在内存中的合适位置放置1的话会怎么样？咱们来试试看。

下一个问题就是怎么做最简单。好，我们已经有了在9ADBFC处被用某些值(DL)填充的内存位置，所以我们只需要在上面的某个地方将DL改成1。问题是将DL改成1需要在当前指令的长度上加一个字节，这样做会覆盖RETN

现在我们注册成功了!!! 继续运行程序（打开一个 demo 文件），Olly 会在注册子程序中断下来几次，不过每次它都会走正确的路。不久你就会看到主窗口：



你会看到我们仍然是已注册状态。点击显示关于对话框：



恭喜你！你已经成功的完成了第一次破解😁。

别忘了将它保存到磁盘。打开硬件断点窗口（“**Debug**” -> “**Hardware breakpoints**”），点断点边上的 **Follow** 按钮。然后我们就来到了我们打补丁的地方。选中所有我们修改过的行，右键选择 “**Copy to executable**”。在弹出的窗口中右键，选择 “**Save to disk**”。以原来的名字保存它。现在退出 Olly，运行程序体验它的全部，以及注册成功的骄傲与自豪!!!