

## R4ndom's Tutorial #13: Cracking a Real Program

by R4ndom on Jul.12, 2012, under Beginner, Reverse Engineering, Tutorials

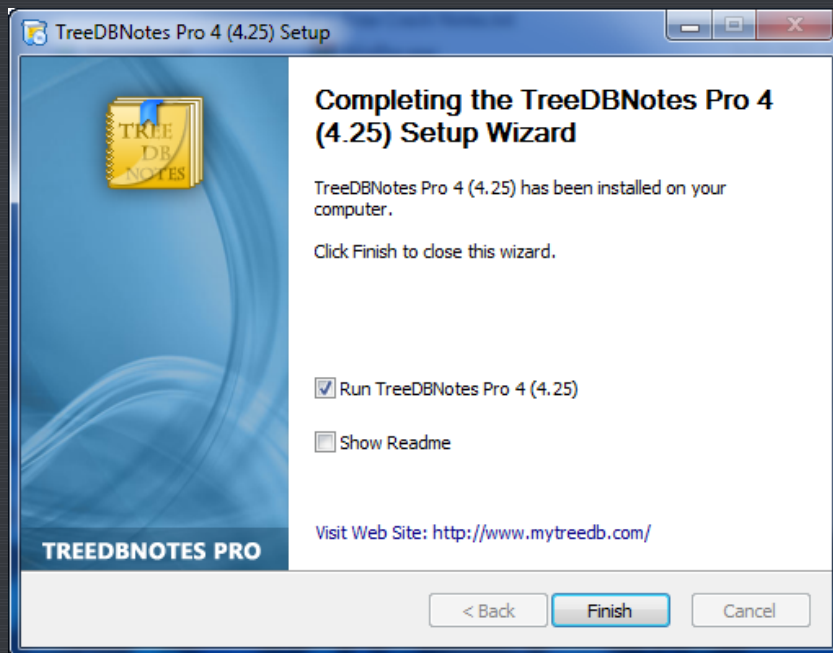
### Introduction

In this tutorial we are going to take off the training wheels and crack a real program. This program has a time restriction, and after this time, it will not work anymore. We are going to patch it to think it is registered. The target is included in this download (I am not stating the name of the program as the purpose of this tutorial is not to get a 'cracked' program but to learn how to do it.) Like all commercial programs, if you plan on using them, you really should consider buying it. People put a great deal of time into apps and they deserve to be compensated. In an attempt to not make this series about 'getting cracked software', I tried to get a program that no one would really want, so I downloaded this app, which had the least amount of downloads last week on Download.com. To be totally honest, after cracking the program in this tutorial, I liked it so much I paid for the registration and now use the app legitimately. Just goes to show you you can't judge an app by it's downloads.

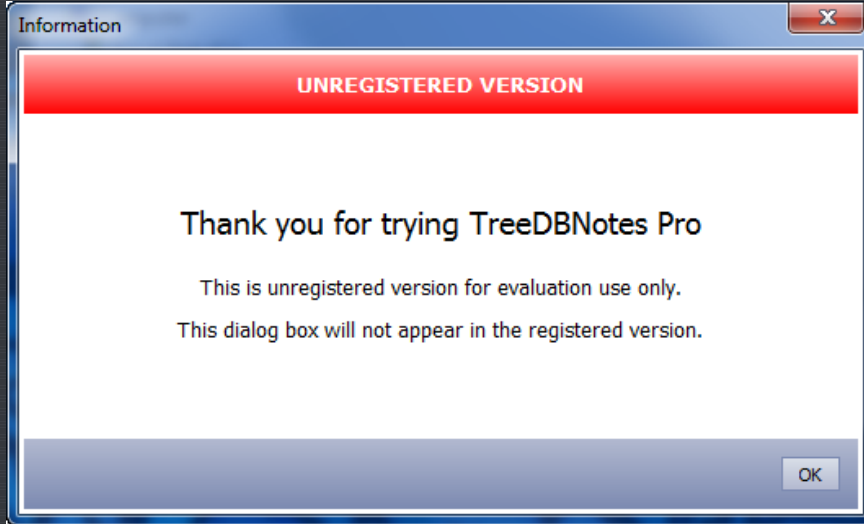
Well, on with the show...

### Studying the App

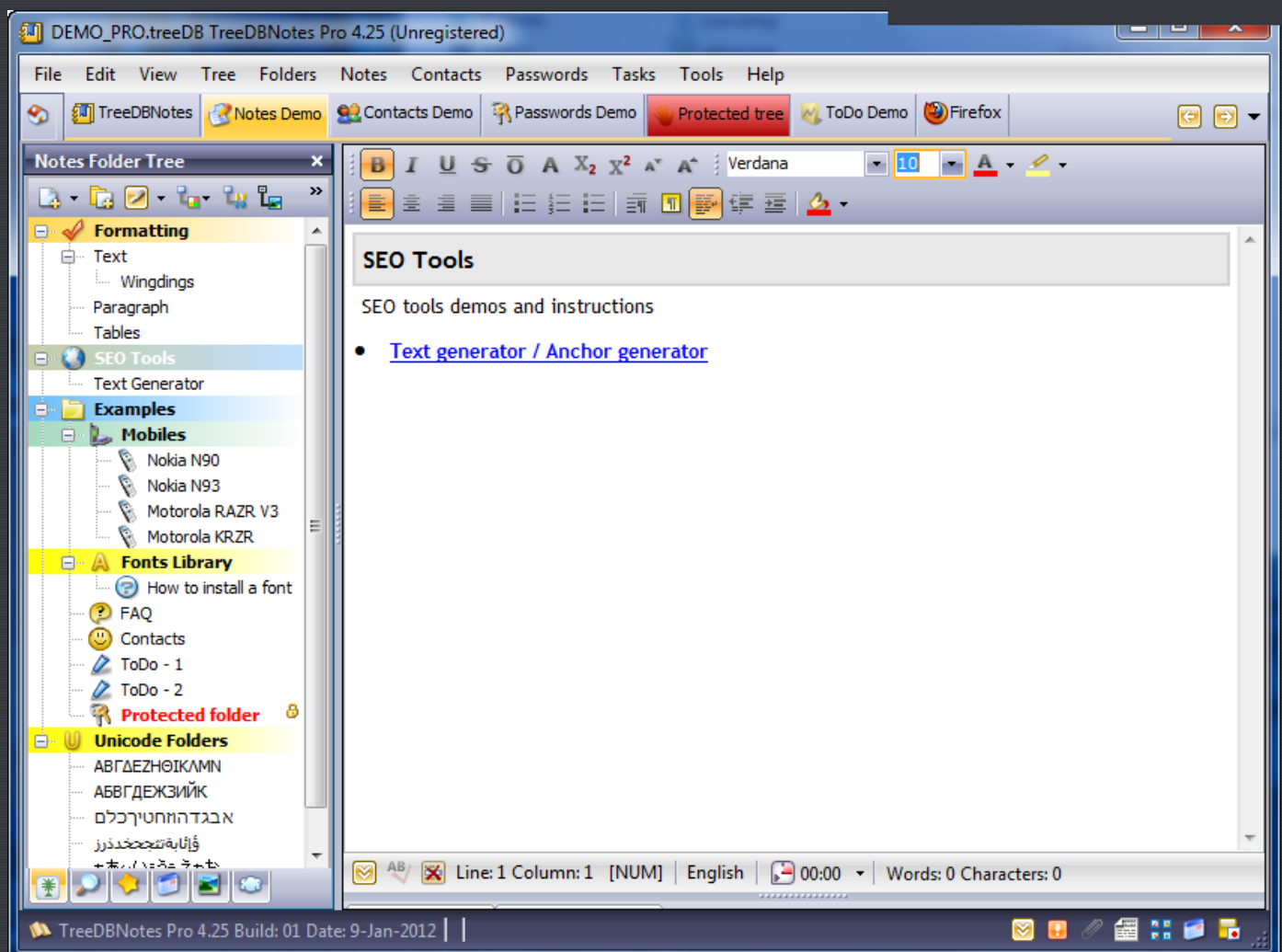
Go ahead and install the app. After completion, the following screen comes up:



Let's leave the "Run the app" checked and see what we're dealing with:



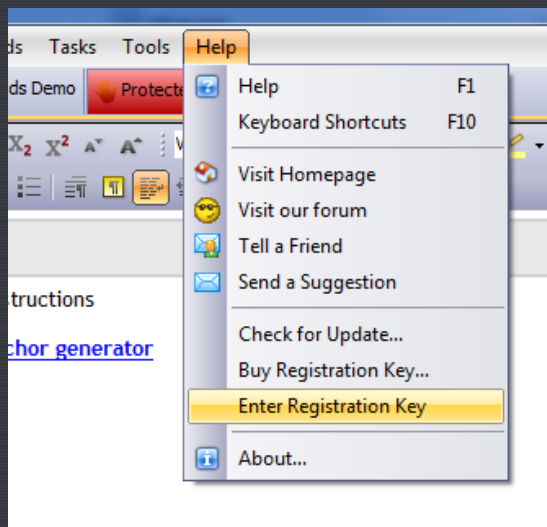
Well that's not very nice. Here we notice some strings that could be a potential help; "unregistered", "evaluation", "registered" etc. Click OK and we get to the main screen:



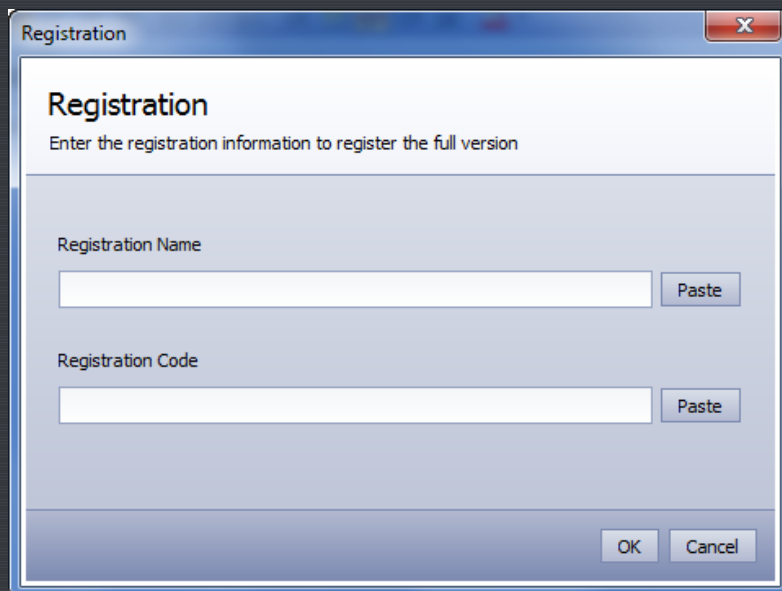
Notice it says "unregistered" at the top in the title bar. Usually, another place I look in an app is the about screen. A lot of times it will contain strings and or ideas for reversing. During this phase, we are looking for keywords, recognizable method calls, stuff like that. The more you do this the more clues will jump out at you:



Here we see the word “unregistered” again. The next thing I usually look for is if there is a way to enter a registration code. This is a good starting point for penetration if the “search for strings” trick doesn’t work:



and here we see an option to enter a reg code:



Let’s try one and see what happens:

Registration

Registration

Enter the registration information to register the full version

Registration Name

R4ndom

Paste

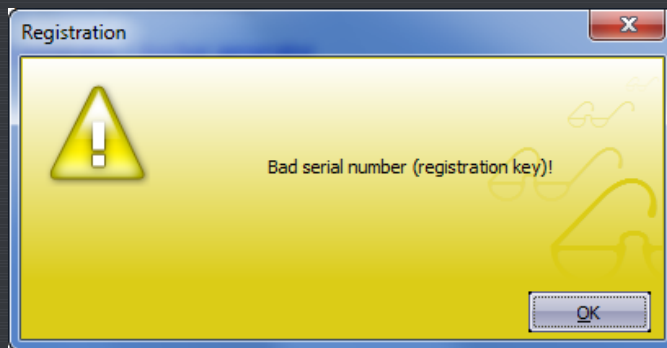
Registration Code

12121212

Paste

OK Cancel

Click OK:



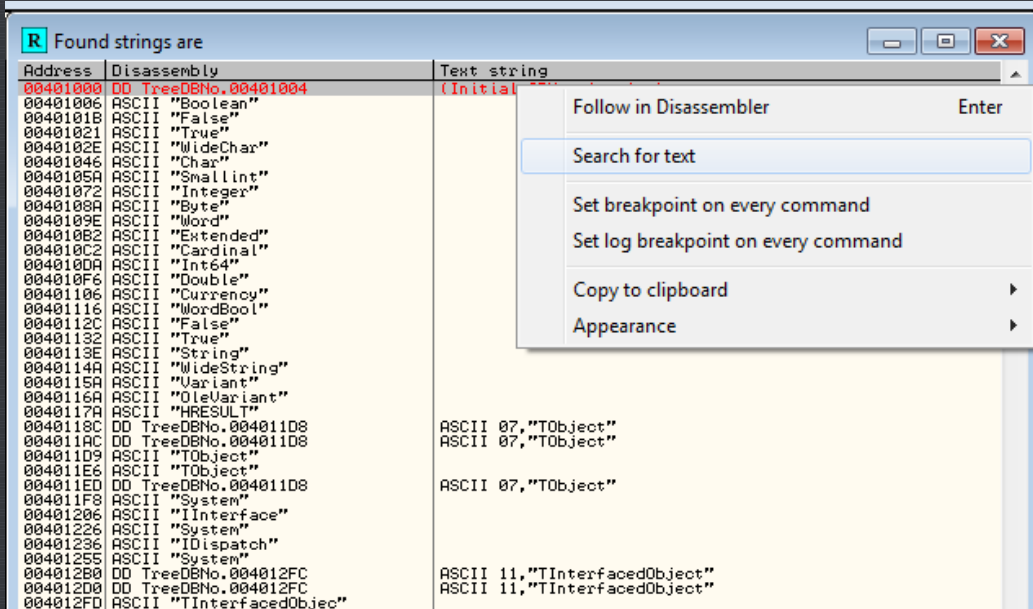
Bummer. I never seem to get this part right 😞. Alright, we have a pretty good idea as to what we have at our disposal, so let's load it up in Olly:

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:[A20600]	TreeDBNo.00A21BF8
009F60AF	33C9	XOR	ECX,ECX	
009F60B1	B2 01	MOV	DL,1	
009F60B3	A1 7CE23700	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 CC613F00	MOV	ECX,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 040F8E00	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	EDX,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 00399800	MOV	EDX,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	EDX,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	EDX,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	EDX,DWORD PTR DS:[9201B8]	TreeDBNo.00920204
009F6160	E8 0FD7AAFF	CALL	TreeDBNo.004A3944	
009F6165	8B00 5CFFA100	MOV	ECX,DWORD PTR DS:[A1FF5C]	TreeDBNo.00A26360
009F6168	8B03	MOV	EAX,DWORD PTR DS:[EBX]	
009F616D	8B15 ACE78E00	MOV	EDX,DWORD PTR DS:[8EE7AC]	TreeDBNo.008EE7F8
009F6173	E8 CC07AAFF	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	EDX,DWORD PTR DS:[8EE6A8]	TreeDBNo.008EE6F4
009F6186	E8 B9D7AAFF	CALL	TreeDBNo.004A3944	
009F6188	8B00 340BA200	MOV	ECX,DWORD PTR DS:[A20B34]	TreeDBNo.00A2676C
009F6191	8B03	MOV	EAX,DWORD PTR DS:[EBX]	
009F6193	8B15 E8EA9700	MOV	EDX,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.0099D02C8	
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.004A09F4	

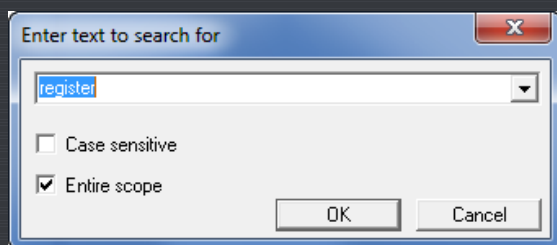
You may notice that this looks a little different than most of the apps we've looked at so far; there seem to be an awful lot of CALL instructions, without the typical Windows setup stuff (like RegisterClass...). This is a good sign that the program was written in Delphi. Delphi uses a TON of calls all over the place. We can tell for sure by running an ID program, but we'll get into that in a future tutorial. There are also specialized tools for dealing with Delphi programs, but fortunately we do not need to use them in this tutorial (we will get to them though 😊)

## Finding the Patches

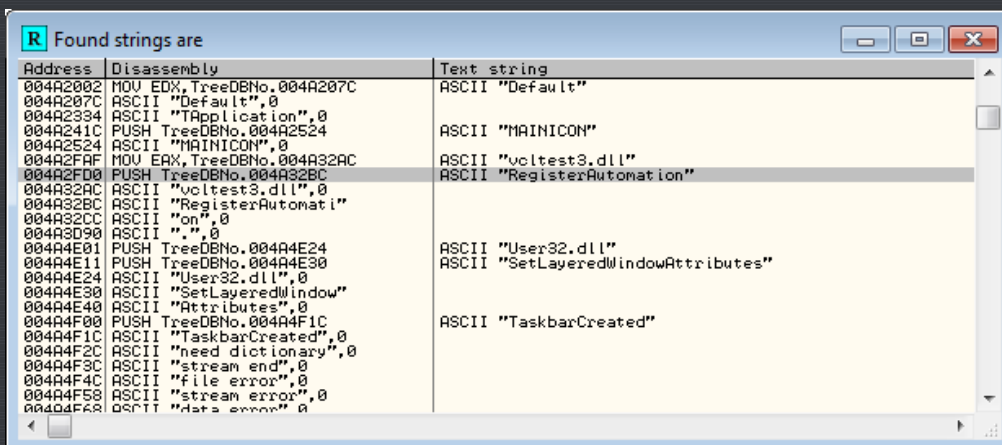
Let's try our string search. Right-click, choose "Search for" -> "All referenced text strings" and the search window will open. Scroll to the top and right click. Choose "Search for text":



and the search for text window opens. Now, I noticed that the word "registration" and "registered" were used a lot earlier, so let's search for them. Usually in this case, as my first search, I will search for "regist" as this covers both "Registration" and "registered", and I've never gotten a false positive from this (I guess not a lot of programs use the word "registrat" in their programs 😊). Make sure "Case sensitive" is unclicked and "Entire scope" IS clicked and hit OK:



The first hit we get doesn't seem promising, so hit ctrl-L to go to the next occurrence:



Notice that this occurrence is just the actual data of the first hit we had. This is because the first hit was where the string "RegisterAutomation" was pushed on to the stack, and the second occurrence is the actual data in memory for the string "RegisterAutomation". You can tell because there is no instruction for it in the second column, and instead it says ASCII. Most strings you come across will have two version of it, the one where the string is accesses, and one where the sting actually resides:

Address	Disassembly	Text string
004A2002	MOV EDX,TreeDBNo.004A207C	ASCII "Default"
004A207C	ASCII "Default",0	
004A2334	ASCII "Application",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	

If you hey ctrl-L again, we will come to another not very promising looking string. Keep hitting ctrl-L until we come to the following:

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	
009AAB89	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	
009AAC58	PUSH TreeDBNo.009AAD70	UNICODE ": "
009AAC5C	MOV ECX,TreeDBNo.009AD50C	UNICODE " "

Now that looks a lot better. It would appear that at some point in the programs starting up sequence, it checks if we art registered or not, and depending on the results, it fills the title bar of the window with either the registered or unregistered string. This is a good place to start. Double click on the "registered" version and we will jump to the code:

009AAB84	• 0F94C2	SETB DL	
009AAB87	• 8B83 280F0000	MOV EAX,DWORD PTR DS:[EBX+F28]	
009AAB8D	• E8 9A3BAEFF	CALL TreeDBNo.0048E72C	kernel132.762BED6C
009AAB92	• 5B	POP EBX	kernel132.762BED6C
009AAB93	• 5D	POP EBP	
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	kernel132.762BED6C
009AABB4	• 5D	POP EBP	kernel132.762BED6C
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	kernel132.762BED6C
009AABC3	• 5D	POP EBP	kernel132.762BED6C
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 38	UNICODE "4.25 (Re"	
009AABFC	• 6700 6900 7300 74	UNICODE "gistered"	
009AAC0C	• 2900 0000	UNICODE ")",0	
009AAC10	• 46	DB 46	
009AAC11	• 00	DB 00	CHAR 'F'
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 38	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	• 5B	POP EBX	



First notice we can see where the string is used at address 9AABA9, and we can also see where the string is stored in memory at address 9AABBC. Secondly, notice that both strings are in the same method and a conditional jump is above them. Clicking on that conditional jump at address 9AABA5:

009AAB77	8BEC	MOV EBX,ESI	
009AAB9B	53	PUSH EBX	
009AAB9C	8BD9	MOV EBX,EDX	
009AAB9E	80B8 B8150000 00	CMP BYTE PTR DS:[EAX+15B8],0	TreeDBNo.<ModuleEntryPoint>
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	
009AABB2	5B	POP EBX	kernel32.7747ED6C
009AABB3	5D	POP EBP	kernel32.7747ED6C
009AABB4	C3	RETN	
009AABC5	00	DB 00	
009AABC6	00	DB 00	
009AABC7	00	DB 00	
009AABC8	42	DB 42	CHAR 'B'

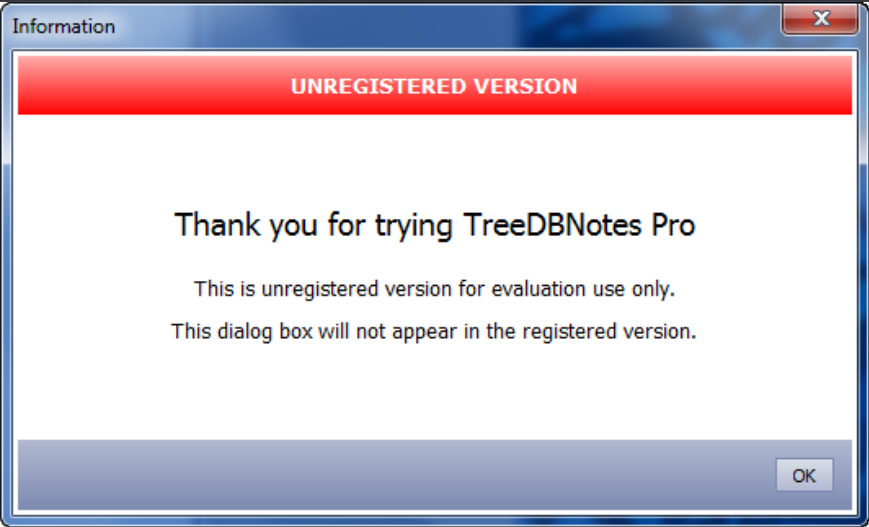
we can see that if the result is equal, we will jump to the “unregistered” version of the string. We obviously don’t want this to happen. Let’s place a BP on this JE instruction and start the app:

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	8BD9	MOV EBX,EDX	
009AAB9E	80B8 B8150000 00	CMP BYTE PTR DS:[EAX+15B8],0	TreeDBNo.<ModuleEntryPoint>
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	
009AABB2	5B	POP EBX	kernel32.7747ED6C
009AABB3	5D	POP EBP	kernel32.7747ED6C
009AABB4	C3	RETN	
009AABC5	00	DB 00	
009AABC6	00	DB 00	
009AABC7	00	DB 00	

Olly will break at this line and you will notice that we are going to jump to the bad boy. Let’s change that:

C	0	ES	0023	32b
E	1	CS	001B	32b
A	0	SS	0023	32b
Z	0	DS	0023	32b
S	0	FS	003B	32b
T	0	GS	0000	NULL
D	0			
O	0	LastErr	ERROR	

and run the app. Olly will then break at this same line again, wanting to jump to the bad boy. Let’s change it again by zeroing out the zero register and hitting run. This will happen one more, and clearing out the zero flag, we finally get some feedback:



So that didn’t work. So patching that one check does not make us unregistered, although if you click OK and zero out the flag one more time, you will notice that it does take off the “unregistered” title of the main window:





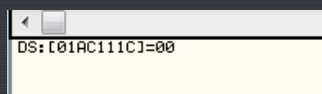
So at least we know we're on the right track. What we are going to have to do is step this up to the next 'level' and investigate a little further. Re-start the app so that we break at our breakpoint and let's investigate a little more:

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	
009AAB9F	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	042233A8
009AABB3	5B	POP EBX	042233A8
009AABB4	5D	POP EBP	
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	042233A8
009AABC2	5B	POP EBX	042233A8
009AABC3	5D	POP EBP	042233A8
009AABC4	C3	RETN	
009AABC5	00	DB 00	
009AABC6	00	DB 00	

There is no call before the compare, but before the JE instruction there is a compare at address 9AAQB9E:

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

So, based on the outcome of this compare, we are either registered or we're not. EAX+15B8 is just a memory address, in this case a global variable as it starts with DS:. What we hope is that this is the only check that the app is registered or not. If it is not, we will need to go find out where else the app checks for registration status. Clicking on the compare instruction shows us what EAX+15B8 is:



So right click on this address and choose "Follow in dump":

Address	Hex dump	ASCII
01AC111C	00 00 00 00 FF FF FF FF 00 00 00 00 00 00 00 00	.....
01AC1120	00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00	.....0.....
01AC113C	00 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00	.....0.....
01AC114C	0A 00 00 00 00 00 00 00 A8 33 22 04 00 00 00 00	.....23".....
01AC115C	00 00 00 00 E4 ED FE 04 00 00 00 00 00 00 00 00	.....2000.....
01AC116C	00 00 00 00 00 00 00 00 10 6D 24 04 F4 6F 24 04	.....ms+ros+
01AC117C	D8 42 AC 01 38 BF 20 04 34 C8 1E 04 00 00 00 00	7B%087 4%0.....
01AC118C	26 00 00 00 88 19 A4 01 FC CD A4 01 0C 39 A7 01	...e+r0"=0.900
01AC119C	04 C0 A6 01 64 FB AB 01 00 00 00 00 00 00 00 00	...0dr%0.....
01AC11AC	00 00 00 00 26 00 00 00 94 C7 47 00 64 FB AB 01	...&...0 G.dr%0
01AC11BC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01AC11CC	14 29 48 00 64 FB AB 01 4E 00 00 00 58 7A 49 00	0H.dr%0N...XzI.
01AC11DC	64 FB AB 01 08 00 5B 03 00 00 00 81 03 00 00 00	dr%00.0.....
01AC11EC	00 00 00 00 00 00 01 00 00 00 0A 00 00 00 00	.....0.....
01AC11FC	00 00 00 00 14 00 00 FF 01 00 00 00 00 00 00 00	.....0.....
01AC120C	00 00 00 00 00 00 00 00 0C 00 00 00 04 00 00 00	.....
01AC121C	00 00 00 00 4E 00 00 00 58 7A 49 00 64 FB AB 01	...N...XzI.dr%0
01AC122C	00 00 3D 02 00 00 00 00 73 02 00 00 00 00 00 00	0.=0...s0.....
01AC123C	01 00 00 00 01 00 00 00 0A 00 00 00 00 00 00 00	0...0.....

\*\*\*Your address will almost certainly be different than mine. That is OK. Just follow along and replace your address with mine and it will run fine \*\*\*

Here we can see the address that is checked for being registered or not; it is the first 00 at address 1AC111C (on my computer at least). That means that if the contents of this memory location were to be anything other than zero, this routine would assume we were registered. This also means that there are probably other routines in the app that check this memory location which is why the main screen shows "Registered" while another part of the app knows we're not. Since we only bypassed this routine's natural flow after checking the memory contents, any other routine that checks it was not bypassed.

First things first, let's set this memory address to non-zero so we know that at least this routine will always work the way we want. Set a breakpoint on the compare line (9AABA5) and delete our other BP. Re-start the app and Olly will break. Right click on the compare line and choose "Follow in dump" -> "Memory location" as Olly reset our dump window when we restarted. One thing you may notice is that the memory address that the compare instruction checks is different this time:

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 00	.....0...0...
01B9114C	00 00 00 00 00 00 00 00 A8 33 22 04 00 00 00 00	.....23"♦...
01B9115C	00 00 00 00 E4 ED FE 04 00 00 00 00 00 00 00 00	....2♦♦♦.....
01B9116C	00 00 00 00 00 00 00 00 10 6D 24 04 F4 6F 24 04	.....♦m\$♦fo\$♦
01B9117C	D8 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	%...8480"=8.940
01B9119C	04 C0 B3 01 64 FB B8 01 00 00 00 00 00 00 00	♦4 0dγ0.....
01B911AC	00 00 00 00 26 00 00 00 94 C7 47 00 64 FB B8 01	....%...8 G.dγ0
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	γ)H.dγ0H...%zI.
01B911DC	64 FB B8 01 08 00 5B 03 00 00 00 00 B1 03 00 00	dγ00.L♦.....
01B911EC	00 00 00 00 00 00 00 00 01 00 00 00 0A 00 00 00	.....0.....
01B911FC	00 00 00 00 14 00 00 FF 01 00 00 00 00 00 00	.....γ.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 01	....N...%zI.dγ0
01B9122C	08 00 3D 02 00 00 00 00 73 02 00 00 00 00 00	8.=0...s0.....
01B9123C	01 00 00 00 01 00 00 00 00 00 00 00 00 00 00	0.0.....

My first one was 1AC111C and it is now 1B9111C. Yours will be different than mine, but just notice that the second time through, the memory address that stores the registered/not-registered flag is different.

Click on the "00" in the dump (at 1B9111C in my dump), right click and choose "Binary" -> "Edit":

Edit data at 01D4111C

ASCII

UNICODE

?

HEX +00

00

☐ Keep size

OK

Cancel

Let's enter 01:

Edit data at 009F7000

ASCII

0

UNICODE

?

HEX +01

01

☐ Keep size

OK

Cancel

and notice it has been updated in the dump:

Address	Hex dump	ASCII
01B9111C	01 00 00 00 FF FF FF FF 00 00 00 00 00 00 00 00	0...
01B91120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....0.....
01B9113C	00 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00	.....0.....
01B9114C	0A 00 00 00 00 00 00 00 A8 33 22 04 00 00 00 00	.....23".....
01B9115C	00 00 00 00 E4 ED FE 04 00 00 00 00 00 00 00 00	.....200.....
01B9116C	00 00 00 00 00 00 00 00 10 6D 24 04 F4 6F 24 04	.....msos
01B9117C	08 42 B9 01 38 BF 20 04 34 C8 1E 04 00 00 00 00	8B087 400.....
01B9118C	26 00 00 00 88 19 B1 01 FC CD B1 01 0C 39 B4 01	.....0.90
01B9119C	04 C0 B3 01 64 FB B8 01 00 00 00 00 00 00 00 00	.....00000000
01B911AC	00 00 00 00 26 00 00 00 94 C7 47 00 64 FB B8 01	.....00G.d00
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 58 7A 49 00	00H.d00N...%zI.
01B911DC	64 FB B8 01 08 00 5B 03 00 00 00 00 B1 03 00 00	d000.L0...000..
01B911EC	00 00 00 00 00 00 00 00 01 00 00 00 0A 00 00 00	.....0.....
01B911FC	00 00 00 00 14 00 00 FF 01 00 00 00 00 00 00 00	.....0.....
01B9120C	00 00 00 00 00 00 00 00 0C 00 00 00 04 00 00 00	.....0.....
01B9121C	00 00 00 00 4E 00 00 00 58 7A 49 00 64 FB B8 01	.....N...%zI.d000

No go ahead and run it till we break again. You will notice that the memory contents have changed back to zeroes and that we are now going to jump to the bad boy again. This means that somewhere in the app, a secondary check was done that reset our registered flag back to zero. What we need to do is find where this is being set and make sure it doesn't happen. To do this, we want to set a hardware breakpoint on this memory location to tell Olly to stop whenever the app writes to this location. We want to chose 'write' because somewhere a zero is being written to this address.

Re-start the app and run it until we break. Right click the compare and choose "Follow in dump" again as Olly has reset the dump window. Binary edit the first memory location to 01. Notice it's now at a different memory address:

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 EDI,TreeDBNo.009AABCC	Unicode "TreeDBNotes"
009AABAE	E8 55A9A5FF	CALL TreeDBNo.00405508	
009AABB3	5B	POP EBX	01B8FB64
009AABB4	5D	POP EBP	01B8FB64
009AABB5	C3	RETN	
009AABB6	8BC3	MOV EAX,EBX	
009AABB8	BA 14AC9A00	MOV EDI,TreeDBNo.009AAC14	Unicode "TreeDBNotes"
009AABB9	E8 46A9A5FF	CALL TreeDBNo.00405508	
009AABC2	5B	POP EBX	01B8FB64
009AABC3	5D	POP EBP	01B8FB64
009AABC4	C3	RETN	
009AABC5	00	DB 00	
009AABC6	00	DB 00	
009AABC7	00	DB 00	
009AABC8	42	DB 42	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	.....0.....
01B9113C	00 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00	.....0.....
01B9114C	0A 00 00 00 00 00 00 00 A8 33 22 04 00 00 00 00	.....23".....
01B9115C	01 00 00 00 E4 ED FE 04 00 00 00 00 00 00 00 00	.....200.....
01B9116C	00 00 00 00 00 00 00 00 10 6D 24 04 F4 6F 24 04	.....msos
01B9117C	08 42 B9 01 38 BF 20 04 34 C8 1E 04 00 00 00 00	8B087 400.....
01B9118C	8C 11 B9 01 8C 11 B9 01 24 00 00 00 0C 39 B4 01	8C11B9012400000C39B401
01B9119C	04 C0 B3 01 64 FB B8 01 C8 A8 01 05 2C A9 01 05	.....00000000
01B911AC	24 00 00 00 27 00 00 00 94 C7 47 00 64 FB B8 01	.....00G.d000
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 58 7A 49 00	00H.d00N...%zI.
01B911DC	64 FB B8 01 08 00 5B 03 00 00 00 00 B1 03 00 00	d000.L0...000..
01B911EC	00 00 00 00 00 00 00 00 01 00 00 00 0A 00 00 00	.....0.....
01B911FC	00 00 00 00 14 00 00 FF 01 00 00 00 00 00 00 00	.....0.....
01B9120C	00 00 00 00 00 00 00 00 0C 00 00 00 04 00 00 00	.....0.....
01B9121C	00 00 00 00 4E 00 00 00 58 7A 49 00 64 FB B8 01	.....N...%zI.d000

Then right-click on the first value in the dump that we edited and choose"Breakpoint" -> Hardware, on write" -> "byte":

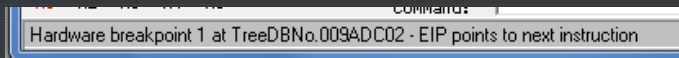
Address	Hex dump	ASCII
01BB111C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0...
01BB112C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....0.....
01BB113C	00 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00	.....0.....
01BB114C	00 00 00 00 00 00 00 00 A8 33 22 04 00 00 00 00	.....23".....
01BB115C	00 00 00 00 E4 ED FE 04 00 00 00 00 00 00 00 00	.....200.....
01BB116C	00 00 00 00 00 00 00 00 10 6D 24 04 F4 6F 24 04	.....msos
01BB117C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01BB118C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01BB119C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01BB11AC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01BB11BC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01BB11CC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01BB11DC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01BB11EC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01BB11FC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01BB120C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01BB121C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01BB122C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01BB123C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

When reverse engineering an app, I generally stay with hardware breakpoints as they are harder for the app to detect. I selected "byte" as it's only the one byte we want to track.

Now run the app. Olly will break at our normal breakpoint again, and you can see that the 01 value we entered is still there, so so far so good. Run it again and Olly will break in a new section:

009ADBEE	72	DB 72	CHAR 'r'
009ADBEF	00	DB 00	
009ADBFB	6C	DB 6C	CHAR 'l'
009ADBF0	00	DB 00	
009ADBF1	00	DB 00	
009ADBF2	00	DB 00	
009ADBF3	00	DB 00	
009ADBF4	3A90 B8150000	CMP DL, BYTE PTR DS:[EAX+15B8]	
009ADBF5	74 06	JE SHORT TreeDBNo.009ADC02	
009ADBF6	8890 B8150000	MOV BYTE PTR DS:[EAX+15B8], DL	
009ADBF7	C3	RETN	
009ADC02	90	NOP	
009ADC03	55	PUSH EBP	
009ADC04	8BEC	MOV EBP, ESP	
009ADC05	83C4 F8	ADD ESP, -8	
009ADC06	53	PUSH EBX	
009ADC07	56	PUSH ESI	
009ADC08	57	PUSH EDI	

If you look in the bottom left corner of the OllyDBG window, you will see that we broke on our hardware breakpoint:



## Patching the App

Now, let's study this code. the first instruction compared DL with the memory contents of our edited address. If they are equal, we jump to 9ADC02, which simply returns. If they are not the same, we store the contents of DL into our memory location. We already know that DL equals zero because we saw the memory location change from our 01 back to 00. So this is basically another registration check, and if it fails if puts a zero in the registered/not-registered flag. If it doesn't fail, it leaves it alone. Now let's remove our hardware breakpoint "Debug" -> "Hardware breakpoints" and delete it, and let's place another hardware breakpoint at address 9ADBF4 so that we can break before this routine has run:

DB 00  
DB 00  
CMP DL, BYTE PTR DS:[EAX+15B8]  
JE SHORT TreeDBNo.009ADC02  
MOV BYTE PTR DS:[EAX+15B8], DL  
RETN  
NOP  
PUSH EBP  
MOV EBP, ESP  
ADD ESP, -8  
PUSH EBX  
PUSH ESI  
PUSH EDI  
MOV [LOCAL.2], ECX  
MOV [LOCAL.1], EDX  
MOV EBX, EAX  
MOV EAX, [LOCAL.1]  
MOV EDX, DWORD PTR DS:[50B0B0]  
CALL TreeDBNo.004040B8  
MOVZX EDI, BYTE PTR DS:[EAX+11]  
IMUL EDI, [ARG.2]  
ADD EDI, [LOCAL.2]  
MOV ESI, DWORD PTR DS:[EBX+CA0]  
MOV DWORD PTR DS:[ESI+C], EDI  
MOV EDX, EDI  
MOV EAX, ESI  
CALL TreeDBNo.004E49B0  
MOV EAX, EBX  
CALL TreeDBNo.009A3EC4  
POP EDI  
POP ESI  
POP EBX  
POP ECX  
POP ECX  
POP EBP  
RETN 8  
NOP  
PUSH EBP

Backup

Copy

Binary

Assemble

Label

Comment

Breakpoint

Hit trace

Run trace

New origin here

Go to

Thread

Follow in Dump

View call tree

Toggle

Conditional

Conditional log

Run to selection

Memory, on access

Memory, on write

Hardware, on execution

ESI 00000000  
EDI 01BDFB64  
EIP 009ADC02 TreeDBNo.009ADC02  
C 1 ES 0023 32bit 0(FFFFFFFF)  
P 1 CS 001B 32bit 0(FFFFFFFF)  
A 1 SS 0023 32bit 0(FFFFFFFF)  
Z 0 DS 0023 32bit 0(FFFFFFFF)  
S 1 FS 003B 32bit 7FFDF000(4000)  
T 0 GS 0000 NULL  
D 0  
O 0 LastErr ERROR\_SUCCESS (00000000)  
EFL 00200297 (NO, B, NE, BE, S, PE, L, LE)  
ST0 empty -??? FFFF 00000033 003300  
ST1 empty -??? FFFF 00000000 000000  
ST2 empty 000 FFFF 00000000 000000

Now you may wonder why I didn't just put a regular breakpoint on this. It is because I tried that first! But Olly would not break on it. There are several reasons that could cause this; this code changes polymorphically, so our BP is lost, there is a check in the app for a software breakpoint and the app removes it, the breakpoint is in a section that Olly will not track automatically... It happens. If it does, we need to set a hardware breakpoint on it instead. There are no guarantees that a HW breakpoint will work, as the app may specifically check for these as well, but it is a more robust way of placing a breakpoint, so it usually works.

\*\*\* We will be going over anti-debug tricks more in future tutorials\*\*\*

Now restart the app and we will again break at our new hardware breakpoint:

009ADBF2	00	DB 00	
009ADBF3	00	DB 00	
009ADBF4	3A90 B8150000	CMP DL, BYTE PTR DS:[EAX+15B8]	
009ADBF5	74 06	JE SHORT TreeDBNo.009ADC02	
009ADBF6	8890 B8150000	MOV BYTE PTR DS:[EAX+15B8], DL	
009ADC02	C3	RETN	
009ADC03	90	NOP	
009ADC04	55	PUSH EBP	
009ADC05	8BEC	MOV EBP, ESP	

OK, now let's think for a minute. This routine is called before our original break. This routine checks if we are registered or not and puts a zero in the memory address pointed to by [EAX+15B8] if it is not, and a 01 (or any non-zero) if it is. Then our old routine is called, the one that either prints "Registered" or "Unregistered" on the title of the window based on if this memory location contains a 0 or 1. So if we make sure a 1 is put into that memory location every time this routine is run, then any other routines will check that memory location and see that it is a 1 and think that we're registered.

What would happen if we just change this routine to always put a 01 into the proper memory location? Let's try it.

Now the next question is what's the easiest way to do that. Well, we have the memory location already being populated with something (DL) at address 9ADBFC, so we could just change the DL to a one. The problem with this is that changing the DL to a one will add a byte to the length of this instruction, and this will overwrite our RETN statement. What about if we replace the compare and jump instructions and instead just load 01 into DL. That way, on the last line, DL will be moved into our memory location! So here's what we do- highlight the two compare and jump instructions:

009ADBFB	00	DB 00	
009ADBFC	r# 3A90 B8150000	CMP DL, BYTE PTR DS:[EAX+15B8]	
009ADBFD	74 06	JE SHORT TreeDBNo.009ADC02	
009ADBE0	8890 B8150000	MOV BYTE PTR DS:[EAX+15B8], DL	
009ADBE1	C3	RETN	
009ADBE2	90	NOP	
009ADBE3	55	PUSH EBP	

Then right-click and choose "Binary" -> "Fill with NOPs":

The screenshot shows a debugger window with assembly code. A right-click context menu is open over the assembly code, and a sub-menu is open for the 'Binary' option. The sub-menu contains 'Fill with 00's', 'Fill with NOPs', and 'Binary copy'. A blue arrow points to 'Fill with NOPs'.

Which gives us this:

009ADBFB	00	DB 00	
009ADBFC	00	DB 00	
009ADBFD	90	NOP	
009ADBE0	90	NOP	
009ADBE1	90	NOP	
009ADBE2	90	NOP	
009ADBE3	90	NOP	
009ADBE4	90	NOP	
009ADBE5	90	NOP	
009ADBE6	90	NOP	
009ADBE7	90	NOP	
009ADBE8	90	NOP	
009ADBE9	90	NOP	
009ADBEA	90	NOP	
009ADBEB	90	NOP	
009ADBE4	8890 B8150000	MOV BYTE PTR DS:[EAX+15B8], DL	
009ADBE5	C3	RETN	
009ADBE6	90	NOP	
009ADBE7	55	PUSH EBP	
009ADBE8	8BEC	MOV EBP, ESP	
009ADBE9	83C4 F8	ADD ESP, -8	

This step isn't required, but it makes it a lot easier to see what you're doing.

Now click on the first NOP at address 9ADBFD and hit the space bar. This will bring up the assemble window. Then enter MOV DL, 1:

The screenshot shows a debugger window with assembly code. An 'Assemble at 009ADBFD' dialog box is open, showing the assembly instruction 'mov dl, 1' and a checked checkbox labeled 'Fill with NOP's'. The 'Assemble' button is highlighted.

Click Assemble then Cancel. That gives us this:

009ADB2	00	DB 00	
009ADB3	00	DB 00	
009ADB4	B2 01	MOV DL,1	
009ADB5	90	NOP	
009ADB6	90	NOP	
009ADB7	90	NOP	
009ADB8	90	NOP	
009ADB9	90	NOP	
009ADB0	90	NOP	
009ADB1	90	NOP	
009ADB2	90	NOP	
009ADB3	8890 B8150000	MOV BYTE PTR DS:[EAX+15B8],DL	
009ADB4	C3	RET	
009ADB5	90	NOP	
009ADB6	55	PUSH EBP	
009ADB7	8BEC	MOV EBP,ESP	

Now, whenever this routine is called, a one will be put into the memory flag instead of a zero. Since we are still paused on the first line of this routine, you can single step to see DL being loaded with 1, and then the 1 being put into the memory address (you may need to go to the proper address in your dump as Olly has probably reset it again). Now run the app and Olly will break at our original breakpoint:

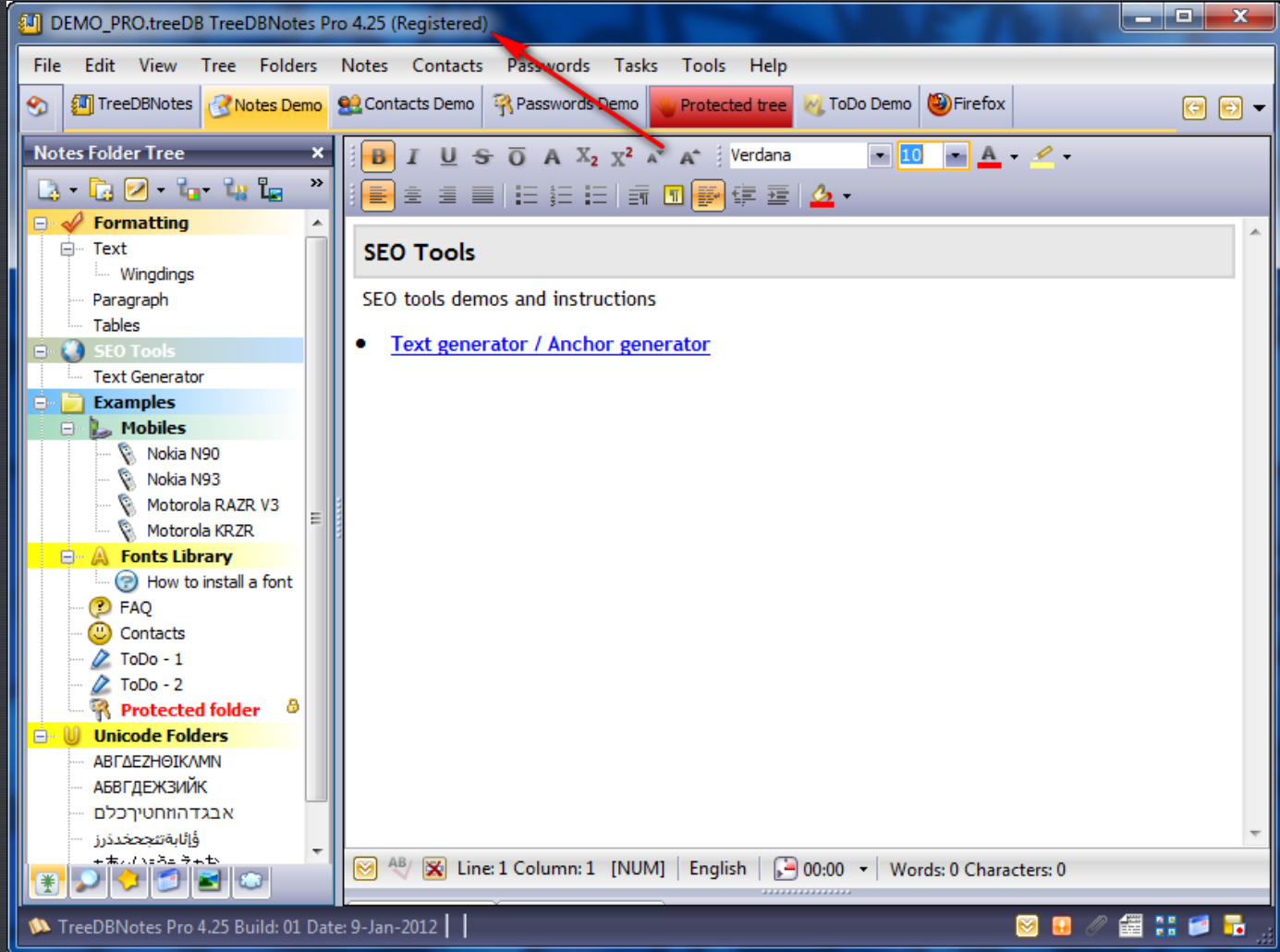
009AAB95	8D40 00	LEA EAX,DWORD PTR DS:[EAX]	
009AAB96	55	PUSH EBP	
009AAB97	8BEC	MOV EBP,ESP	
009AAB98	53	PUSH EBX	
009AAB99	8BD9	MOV EBX,EDX	
009AABA0	80B8 B8150000 00	CMP BYTE PTR DS:[EAX+15B8],0	
009AABA1	74 0F	JS SHORT TreeDBNo.009AABB6	
009AABA2	8BC3	MOV EAX,EBX	
009AABA3	BA C0B9A000	MOV EDI,TreeDBNo.009AABC0	Unicode "TreeDBNotes Pro 4.25 (Registered)"
009AABA4	E8 55A9A5FF	CALL TreeDBNo.00405508	042233A8
009AABA5	5B	POP EBX	042233A8
009AABA6	5D	POP EBP	
009AABA7	C3	RET	
009AABA8	8BC3	MOV EAX,EBX	
009AABA9	BA 14AC9A00	MOV EDI,TreeDBNo.009AAC14	Unicode "TreeDBNotes Pro 4.25 (Unregistered)"
009AABAA	E8 46A9A5FF	CALL TreeDBNo.00405508	042233A8
009AABAB	5B	POP EBX	042233A8
009AABAC	5D	POP EBP	
009AABAD	C3	RET	
009AABAE	00	DB 00	

and we can see that we are going to fall through to the correct string. Go ahead and keep running and we will break in our modified registration check routine, and it will put a 01 into our address again as we planned. This will go back and forth a couple times until finally:



We are now registered!!!! Go ahead and run the program (open a demo file) and Olly will break several more times in our registration routine, but each time it will go the right way. Soon you will get to the main screen:





and you will see that we are still registered. Clicking on the about screen shows:



Congratulations. You have patched your first crack 😊

Don't forget to save it back to disk. Open the Hardware breakpoints window ("Debug" -> "Hardware breakpoints") and click the Follow button on our BP. That will take us to our patch. Highlight everything we changed, right-click and select "Copy to executable". The right-click in the new window and select "Save to disk". Save it as the original file name. Now quit Olly and run the app and experience it is all it registered glory!!!!



