

سری آموزش های R4NDOM-فارسی

قسمت ۱۳- کرک یک برنامه ی واقعی

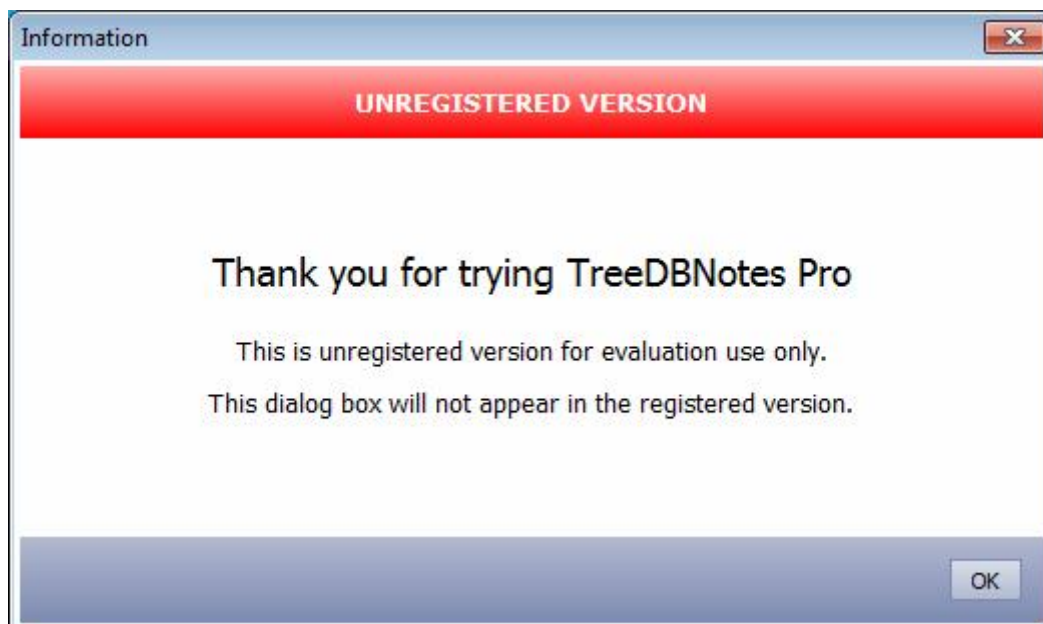
در این قسمت ما قصد داریم یک برنامه ی تجاری را کرک کنیم(ضمیمه شده) این برنامه محدودیت زمانی دارد و پس از به پایان رسیدن زمان دیگر کار نمیکند مال بیاید شروع کنیم :

بررسی عملکرد برنامه:

ابتدا برنامه را نصب کنید و در انتها :



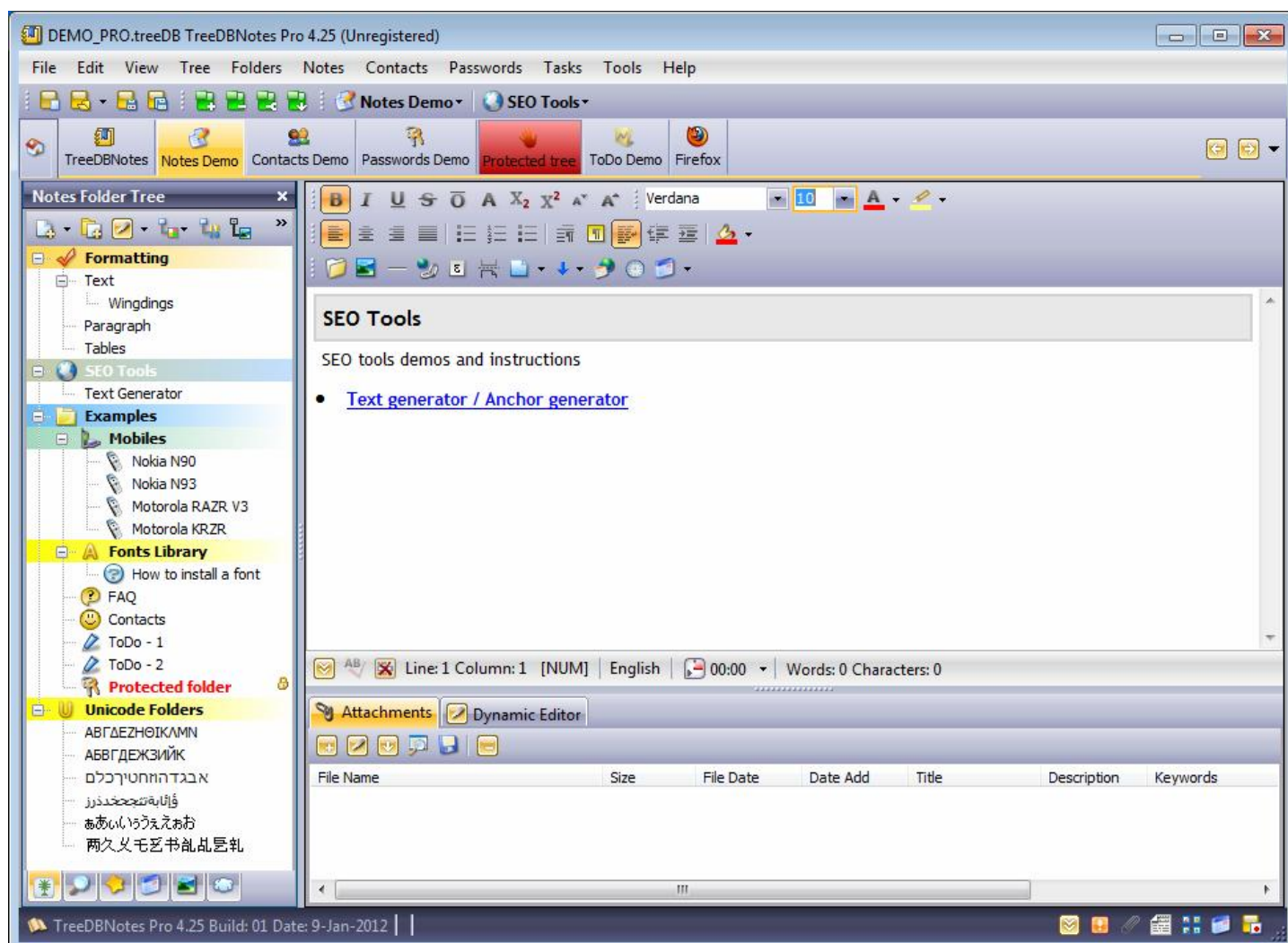
بسیار خوب برنامه را اجرا کنید :



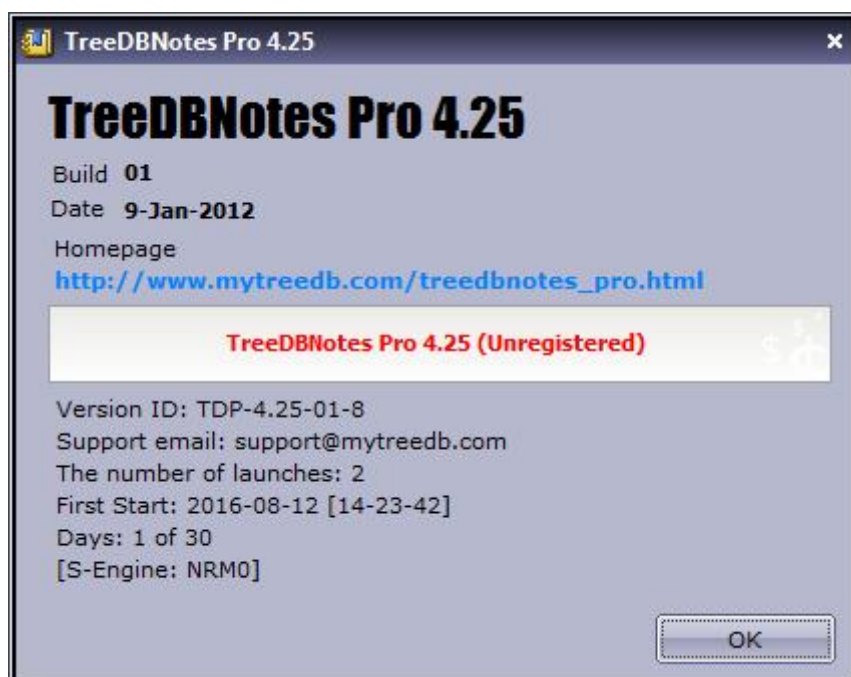
بسیار خوب، البته این فیلی خوشایند نیست، در این صفحه رشته های مانند: evaluation, registered, unregistered را میبینید، بروی Ok کلیک کنید تا برنامه اجرا شود:



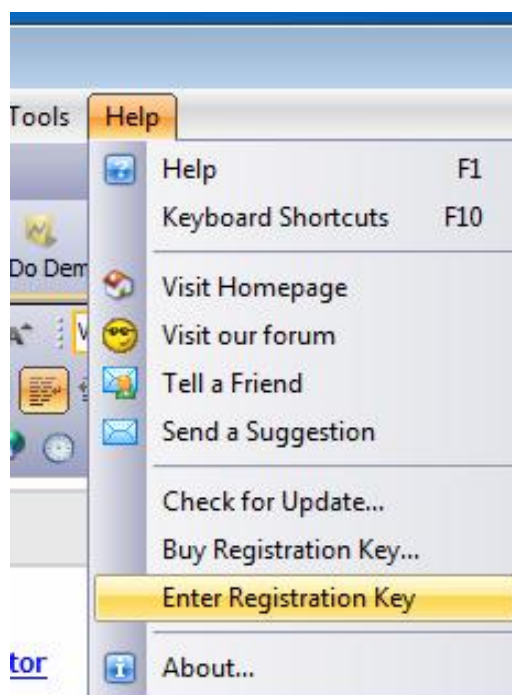
بروی (Open demo file) کلیک کنید تا وارد پنجره ی اصلی شویم :



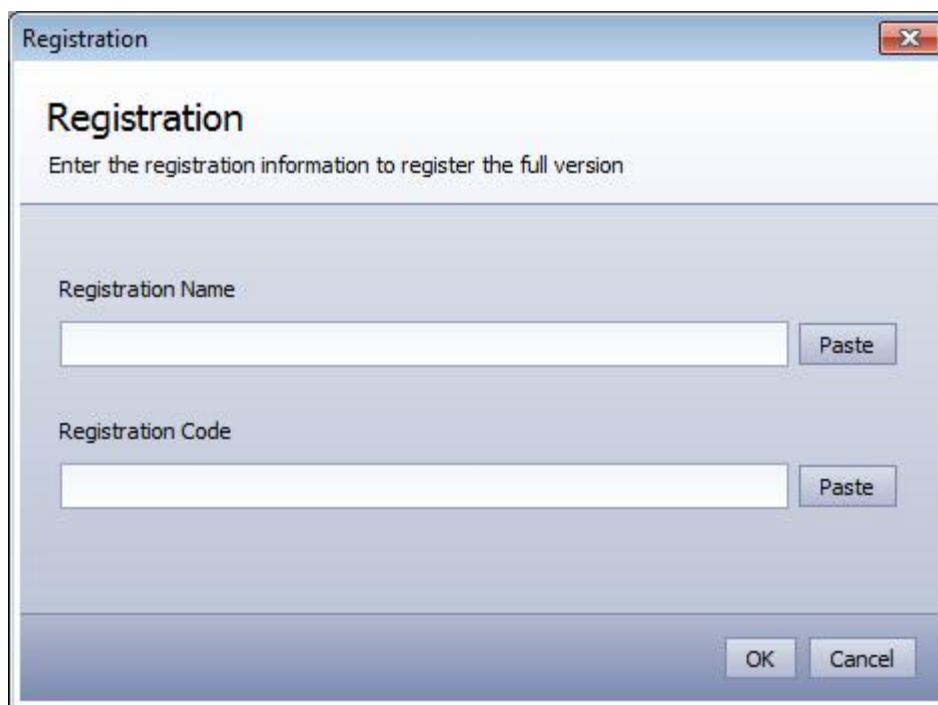
به تایتل برنامه نگاه کنید میبینید که به نوشته شده **Unregistered**، محل های دیگری هم برای تست محدودیت های برنامه وجود دارد مانند پنجره ی **About**، اساسا هرچیزی که به ما سرنگ بدهد مهم است، حال به پنجره ی About از منوی Help میرویم :



همانطور که میبینید اینجا هم دوباره رشته ی **Unregistered** را میبینیم، باز به منوی Help نگاه کنید :



سرف نخ فوبی است اینطور نیست؟ این سرنگ ها موقعی بارزش تر میشوند که مثلا تکنیک **“search for strings”** جواب ندهد
حال بروی **Enter Registryon Key** کلیک کنید:



Registration

Enter the registration information to register the full version

Registration Name

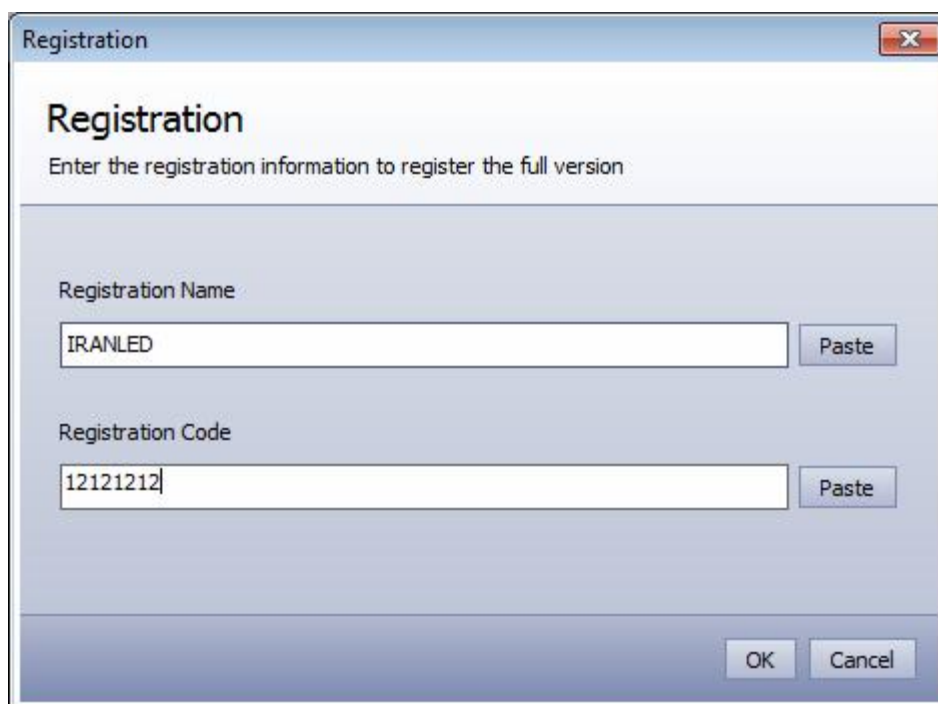
Paste

Registration Code

Paste

OK Cancel

مال طبق معمول همیشه یک نام و کد را وارد کنید و کلید OK را بفشارید :



Registration

Enter the registration information to register the full version

Registration Name

Paste

Registration Code

Paste

OK Cancel

و بعد از کلیک بروی OK:



انتظارش را داشتیم ،نه؟بسیار خوب ،برنامه را درون Olly باز کنید :

```

[*G.P.U* - main thread, module TreeDBNo]
File View Debug Plugins Options Window Help Tools BreakPoint->
Ln E Me Th Ws Hs Cp Pa St Br Re Tr Sc ?

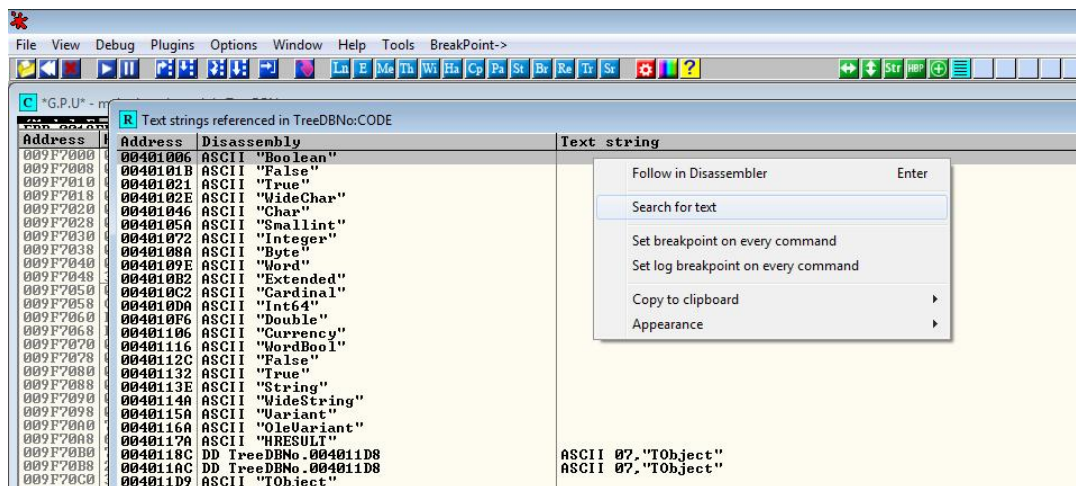
<ModuleEr $ 55 PUSH EBP
009F6099 . 8BEC MOV EBP,ESP
009F609B . 83C4 F0 ADD ESP,-10
009F609E . 53 PUSH EBX
009F609F . B8 204F9F00 MOV EAX,TreeDBNo.009F4F20
009F60A4 . E8 CB16A1FF CALL TreeDBNo.00407774
009F60A9 . 8B1D D006A200 MOV EBX,DWORD PTR [A206D0] TreeDBNo.00A21BF8
009F60AF . 33C9 XOR ECX,ECX
009F60B1 . B2 01 MOV DL,1
009F60B3 . A1 7CE29700 MOV EAX,DWORD PTR [97E27C]
009F60B8 . E8 275EAAFF CALL TreeDBNo.0049BEE4
009F60BD . 8B15 5405A200 MOV EDX,DWORD PTR [A20554] TreeDBNo.00A26764
009F60C3 . 8902 MOV DWORD PTR [EDX],EAX
009F60C5 . A1 5405A200 MOV EAX,DWORD PTR [A20554]
009F60CA . 8B00 MOV EAX,DWORD PTR [EAX]
009F60CC . E8 6BA1AAFF CALL TreeDBNo.004A023C
009F60D1 . A1 5405A200 MOV EAX,DWORD PTR [A20554]
009F60D6 . 8B00 MOV EAX,DWORD PTR [EAX]
009F60D8 . 8B10 MOV EDX,DWORD PTR [EAX]
009F60DA . FF92 80000000 CALL DWORD PTR [EDX+80]
009F60E0 . 8B03 MOV EAX,DWORD PTR [EBX]
009F60E2 . E8 45D8AAFF CALL TreeDBNo.004A392C
009F60E7 . 8B03 MOV EAX,DWORD PTR [EBX]
009F60E9 . BA CC619F00 MOV EDX,TreeDBNo.009F61CC ASCII "TreeDBNotes"
009F60EE . E8 31D4AAFF CALL TreeDBNo.004A3524 TreeDBNo.00A2632C
009F60F3 . 8B0D 7C09A200 MOV ECX,DWORD PTR [A2097C] TreeDBNo.00A2632C
009F60F9 . 8B03 MOV EAX,DWORD PTR [EBX] TreeDBNo.008EB020
009F60FB . 8B15 D4AF8E00 MOV EDX,DWORD PTR [8EAFD4] TreeDBNo.008EB020
009F6101 . E8 3ED8AAFF CALL TreeDBNo.004A3944 TreeDBNo.00A26874
009F6106 . 8B0D 0CFFA100 MOV ECX,DWORD PTR [A1FF0C] TreeDBNo.00A26874
009F610C . 8B03 MOV EAX,DWORD PTR [EBX]
009F610E . 8B15 98229D00 MOV EDX,DWORD PTR [9D2298] TreeDBNo.009D22E4
009F6114 . E8 2BD8AAFF CALL TreeDBNo.004A3944 TreeDBNo.00A267A4
009F6119 . 8B0D 0000A200 MOV ECX,DWORD PTR [A20000] TreeDBNo.00A267A4
009F611F . 8B03 MOV EAX,DWORD PTR [EBX]
009F6121 . 8B15 D0999800 MOV EDX,DWORD PTR [9899D0] TreeDBNo.00989A1C
009F6127 . E8 18D8AAFF CALL TreeDBNo.004A3944 TreeDBNo.00A248F0
009F612C . 8B0D F800A200 MOV ECX,DWORD PTR [A200F8] TreeDBNo.00A248F0
009F6132 . 8B03 MOV EAX,DWORD PTR [EBX]
009F6134 . 8B15 58148B00 MOV EDX,DWORD PTR [8B1458] TreeDBNo.008B14A4
009F613A . E8 05D8AAFF CALL TreeDBNo.004A3944 TreeDBNo.00A263E4
009F613F . 8B0D 1403A200 MOV ECX,DWORD PTR [A20314] TreeDBNo.00A263E4
009F6145 . 8B03 MOV EAX,DWORD PTR [EBX]
009F6147 . 8B15 28FB9100 MOV EDX,DWORD PTR [91FB28] TreeDBNo.0091FB74
009F614D . E8 F2D7AAFF CALL TreeDBNo.004A3944 TreeDBNo.00A263F0
009F6152 . 8B0D 5802A200 MOV ECX,DWORD PTR [A20258] TreeDBNo.00A263F0
009F6158 . 8B03 MOV EAX,DWORD PTR [EBX]
009F615A . 8B15 B8019200 MOV EDX,DWORD PTR [9201B8] TreeDBNo.00920204
009F6160 . E8 DFD7AAFF CALL TreeDBNo.004A3944 TreeDBNo.00A26360
009F6165 . 8B0D 5CFFA100 MOV ECX,DWORD PTR [A1FF5C] TreeDBNo.00A26360
009F616B . 8B03 MOV EAX,DWORD PTR [EBX]
009F616D . 8B15 ACE78E00 MOV EDX,DWORD PTR [8EE7AC] TreeDBNo.008EE7F8
009F6173 . E8 CCD7AAFF CALL TreeDBNo.004A3944 TreeDBNo.00A26358
009F6178 . 8B0D 2802A200 MOV ECX,DWORD PTR [A20228] TreeDBNo.00A26358
009F617E . 8B03 MOV EAX,DWORD PTR [EBX]
009F6180 . 8B15 A8E68E00 MOV EDX,DWORD PTR [8EE6A8] TreeDBNo.008EE6F4
009F6186 . E8 B9D7AAFF CALL TreeDBNo.004A3944 TreeDBNo.00A2676C
009F618B . 8B0D 340BA200 MOV ECX,DWORD PTR [A20B34] TreeDBNo.00A2676C
009F6191 . 8B03 MOV EAX,DWORD PTR [EBX]
009F6193 . 8B15 E8EA9700 MOV EDX,DWORD PTR [97EAE8] TreeDBNo.0097EB34
009F6199 . E8 A6D7AAFF CALL TreeDBNo.004A3944
009F619E . A1 0000A200 MOV EAX,DWORD PTR [A20000]
009F61A3 . 8B03 MOV EAX,DWORD PTR [EBX]

```

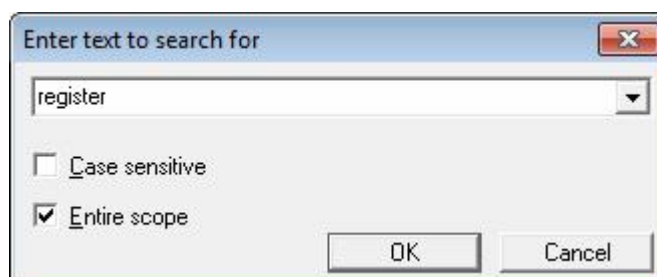
شاید در نگاه اول کمی گیج کننده بیاید چون تا به حال چنین چیزی ندیده ایم ، **Call** های متعدد بدون هیچ گونه توضیحی ، این نشانه هائی از این است که این برنامه با زبان **Delphi** نوشته شده است ، البته در آموزش های بعدی روش تشخیص مفصلا بحث خواهد شد همچنین روش آنالیز برنامه های دلفی ، اما فوشبفتانه در این مثال نیازی نیست

جستجو

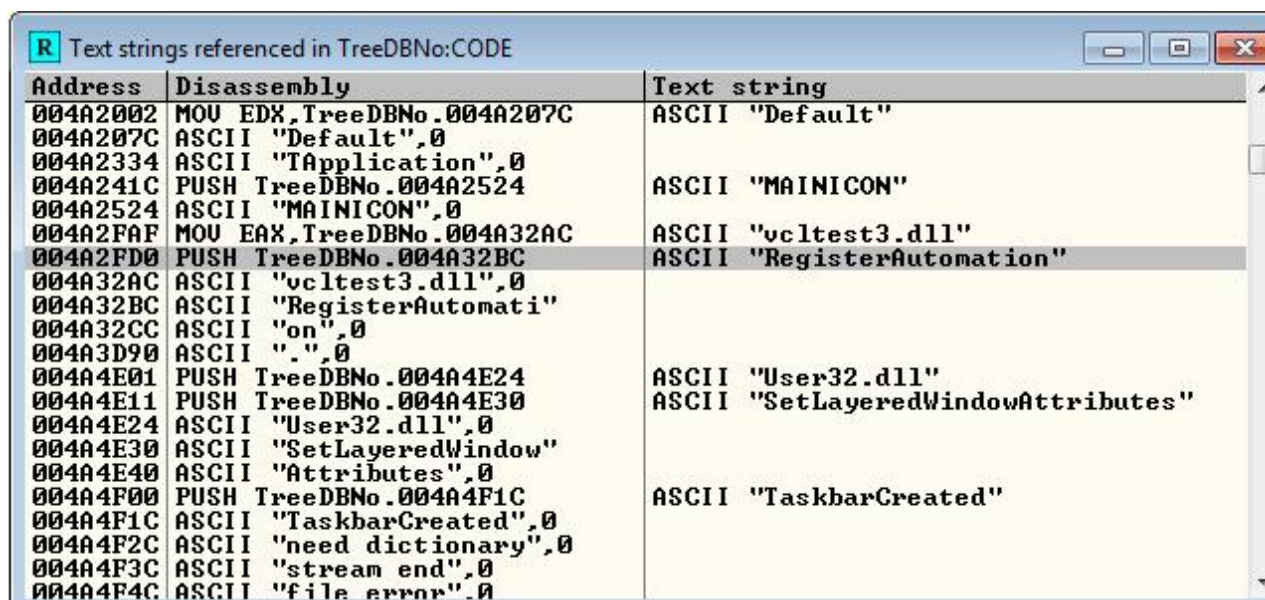
ابتدا بیایید رشته های بکار رفته در برنامه را از طریق **"All referenced text strings"** -> **"Search for"** ببینیم ، بعد از نمایش رشته ها متوجه فراوانی آن ها میشویم و برای سرعت بخشیدن به کار رشته ی مورد نظر را جستجو میکنیم :



معمولا جستجو را با کلماتی مانند Registerd یا Registraion شروع میکنیم ، اما من معمولا از رشته ی Register استفاده میکنم چون در این حالت هم Registration و هم Registerd پوشش داده خواهند شد بسیار خوب :



مطمئن شوید که چک باکس Entries cope در حالت انتخاب شده و **"Case sensitive"** غیر فعال باشد :



این اولین نتیجه ی جستجو است و به نظر به درد نمیخورد با فشردن کلید **Ctrl+L** جستجو را ادامه میدهیم:

R Text strings referenced in TreeDBNo:CODE		
Address	Disassembly	Text string
004A2002	MOV EDX,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 "vcltest3.dll"
004A2FD0	PUSH TreeDBNo.004A32BC	ASCII "RegisterAutomation"
004A32AC	ASCII "vcltest3.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	

یکی دیگر ☺ آیا میدانید فرق این دو چیست؟ هر دو یک رشته ی یکسان هستند با این تفاوت که اولی "RegisterAutomation" درون پشته قرار گرفته و دومی "RegisterAutomation" یک داده ی واقعی در برنامه است. در ستون دوم عبارت ASCII را میبینید، باید بدانید که اکثرا از هر رشته ۲ تا است، اولی برای دستیابی به آن و دومی محل واقعی آن است، بسیار خوب دوباره کلید **Ctrl+L** را چند باری بفشارید تا به رشته زیر برسید:

R Text strings referenced in TreeDBNo:CODE		
Address	Disassembly	Text string
009A9FC4	MOV EDX,TreeDBNo.009AA158	ASCII "ID"
009A9FF1	MOV EDX,TreeDBNo.009AA164	ASCII "NoteData"
009AA04D	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 "	

بهتر شد، به نظر میرسد که این محلی است که تاتیل برنامه از آن استفاده میکند و میتواند نقطه ی شروع خوبی باشد بروی آن دبل کلیک کنید تا به محل آن برویم:

009AAB94	C3	RET	
009AAB95	8D40 00	LEA EAX,DWORD PTR [EAX]	
009AAB98	55	PUSH EBP	
009AAB99	8BEC	MOV EBP,ESP	
009AAB9B	53	PUSH EBX	
009AAB9C	8BDA	MOV EBX,EDX	
009AAB9E	80B8 B8150000	CMP BYTE PTR [EAX+15B81],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	
009AABB4	5D	POP EBP	
009AABB5	C3	RET	
009AABB6	8BC3	MOV EAX,EBX	
009AABB8	BA 14AC9A00	MOV EDX,TreeDBNo.009AAC14	UNICODE "TreeDBNotes Pro 4.25 <Unregistered>"
009AABBD	E8 46A9A5FF	CALL TreeDBNo.00405508	
009AABC2	5B	POP EBX	
009AABC3	5D	POP EBP	
009AABC4	C3	RET	
009AABC5	00	DB 00	
009AABC6	00	DB 00	
009AABC7	00	DB 00	
009AABC8	42	DB 42	
009AABC9	00	DB 00	CHAR 'B'
009AABCA	00	DB 00	

همانطور که در تصویر بالا میبینید اولین چیزی که میبینیم این است که رشته در آدرس 9AABA9 مورد استفاده قرار گرفته و دومین چیز این است که این رشته در آدرس حافظه ی 9AABCC ذخیره شده است ،حالا به خط 9AABA5 دقت کنید :

009AAB95	8BEC	MOV EBX,ESI	
009AAB96	53	PUSH EBX	
009AAB97	8BDA	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)"
009AABBD	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'

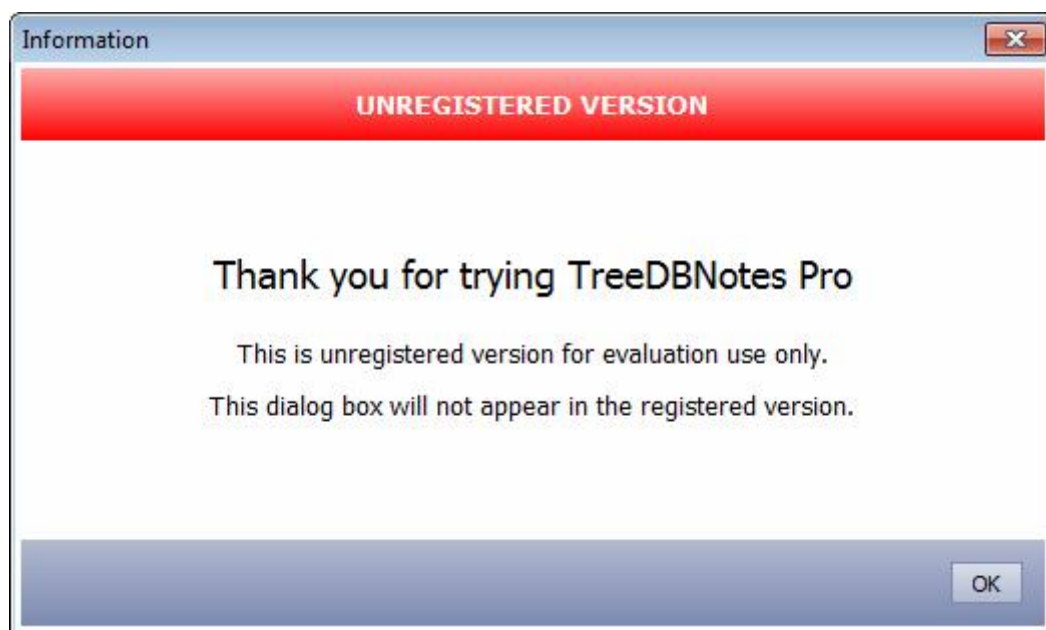
یک دستور العمل پرشی میبینیم که به این معنی است که اگر نتیجه برابر باشد ما به قسمت "unregistered" پرش خواهیم کرد از آنجایی که ما نمیخواهیم چنین اتفاقی رخ دهد بروی این دستورالعمل یک BP گذاشته و برنامه را اجرا میکنیم :

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	
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)"
009AABBD	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	

همانطور که انتظار داشتیم Olly به روی دستور العمل JE متوقف شده ما اکنون در آستانه ی ارجاع به Bad Boy هستیم ، با تغییر پرچم Z این ارجاع با Good Boy تعویض خواهد شد :

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
D	0			
O	0	LastError	ERRO	

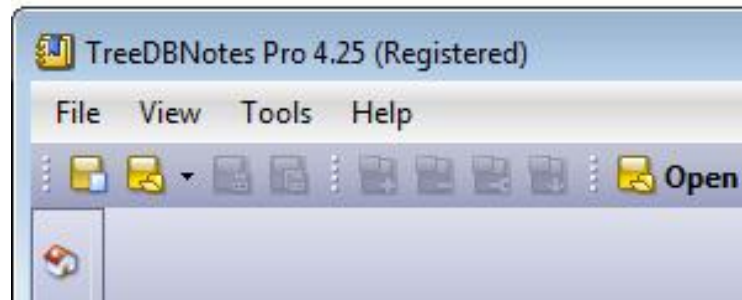
و برنامه را با کلید F9 اجرا کنید، میبینید که olly مجدداً بروی همان آدرس متوقف شده که میخواهد به bad Boy پرش کند، دوباره پرچم Z را تغییر دهید و برنامه را با F9 اجرا کنید، اگر olly دوباره متوقف شد، باز مقدار پرچم Z را تغییر دهید و برنامه را با F9 اجرا کنید :



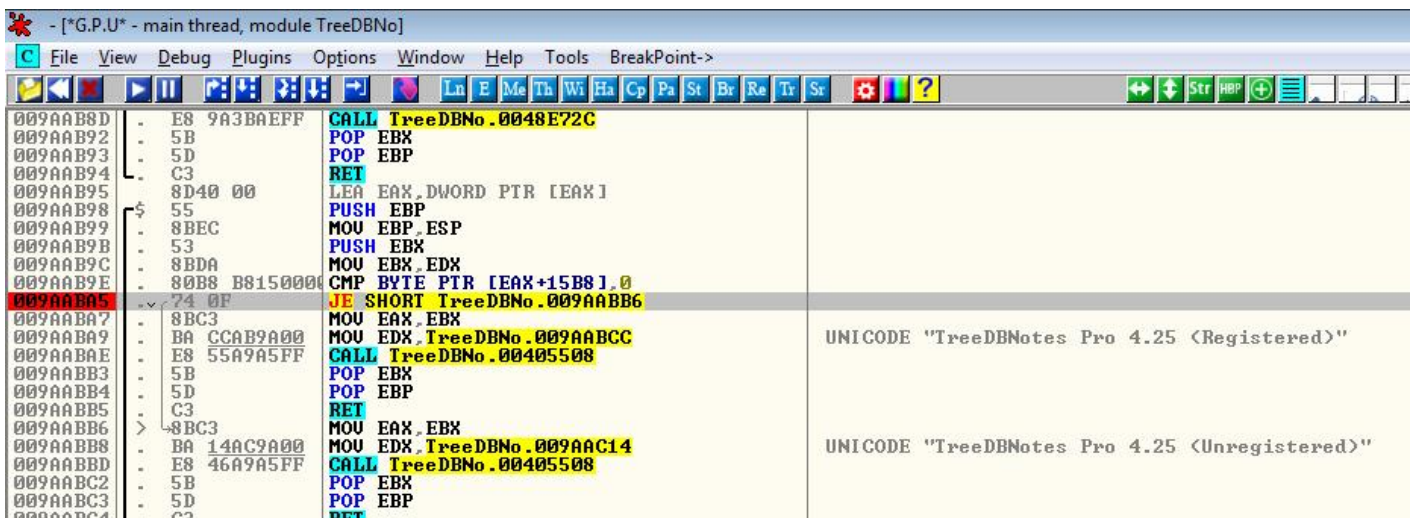
میبیند که پچ ما بروی این قسمت جواب نداده، مال اگر کلید Ok را بفشارید olly باز در همان آدرس چندین بار متوقف میشود و شما هم به همان تعداد پرچم Z را تغییر دهید تا به سمت Good Boy پرش کنید و برنامه را با F9 اجرا کنید :



بروی Close کلیک کنید و باز olly متوقف میشود و باز هم مقدار پرچم Z را در صورت نیاز تغییر دهید تا موقعی که برنامه بالا بیاید:



بسیار خوب، برنامه را ری استارت کنید و به آدرس قبلی باز گردید :



اگر متوجه شده باشید که هیچ فراخوانی قبل از آدرس **9AABA5** وجود ندارد ولی در آدرس **9AAB9E** یک مقایسه وجود دارد:

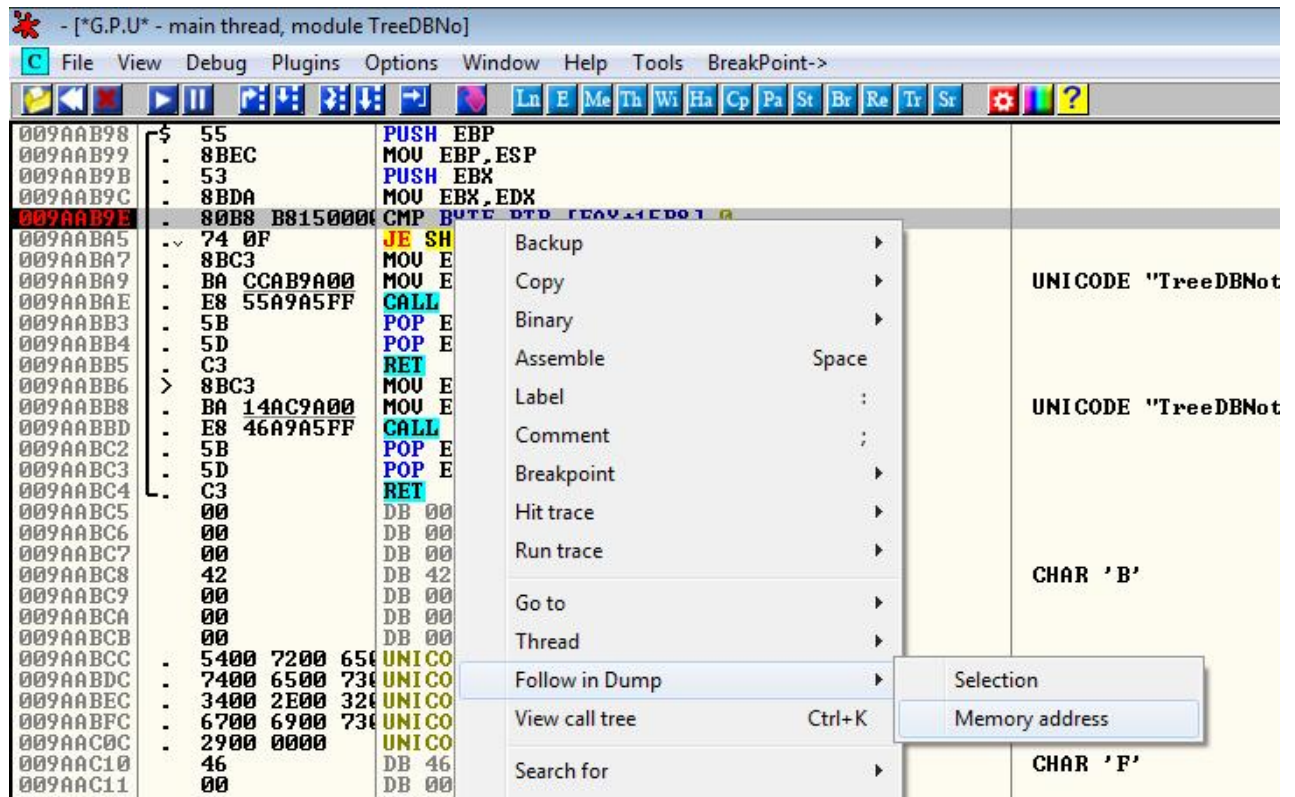
CMP BYTE PTR DS:[EAX+15B8],0

این دستور چک میکند که آیا برنامه رجیستر شده یا نه ، یک بایت از آدرس حافظه ی [EAX+15B8] واقع در DS را که متغیری است از نوع Global که وظیفه ی چک کردن وضعیت رجیستر شدن برنامه را به عهده دارد ،بیاید ببینیم در آدرس [EAX+15B8] چه مقداری وجود دارد ، با راست کلیک کردن بروی قط مذکور و **Flow in Dump**:

Address	Hex dump	ASCII
01BB0F28	00 00 00 00 FF FF FF FF
01BB0F30	00 00 00 00 00 00 00 00
01BB0F38	00 00 00 00 00 00 00 00
01BB0F40	00 00 00 00 00 00 00 00
01BB0F48	00 00 00 00 00 00 00 00
01BB0F50	01 00 00 00 01 00 00 00
01BB0F58	00 00 00 00 00 00 00 00

نکته: آدرس یقیناً در سیستم شما متفاوت خواهد بود

همانطور که میبینید اولین بایت 00 از آدرس 01BB0F28 مسئول وضعیت رجیستر شدن را به عهده دارد، به این معنی که اگر مقدارش هر چیزی بجز صفر باشد برنامه رجیستر خواهد شد
 بسیار خوب بباید بروی فضا 9AAB9E یک BP بگذارید و بقیه ی BP ها را بردارید، بعد برنامه را ری استارت کنید، همانطور که مشاهده میکنید Olly متوقف شده است حال بروی فضا مذکور راست کلیک کرده "Follow in dump":



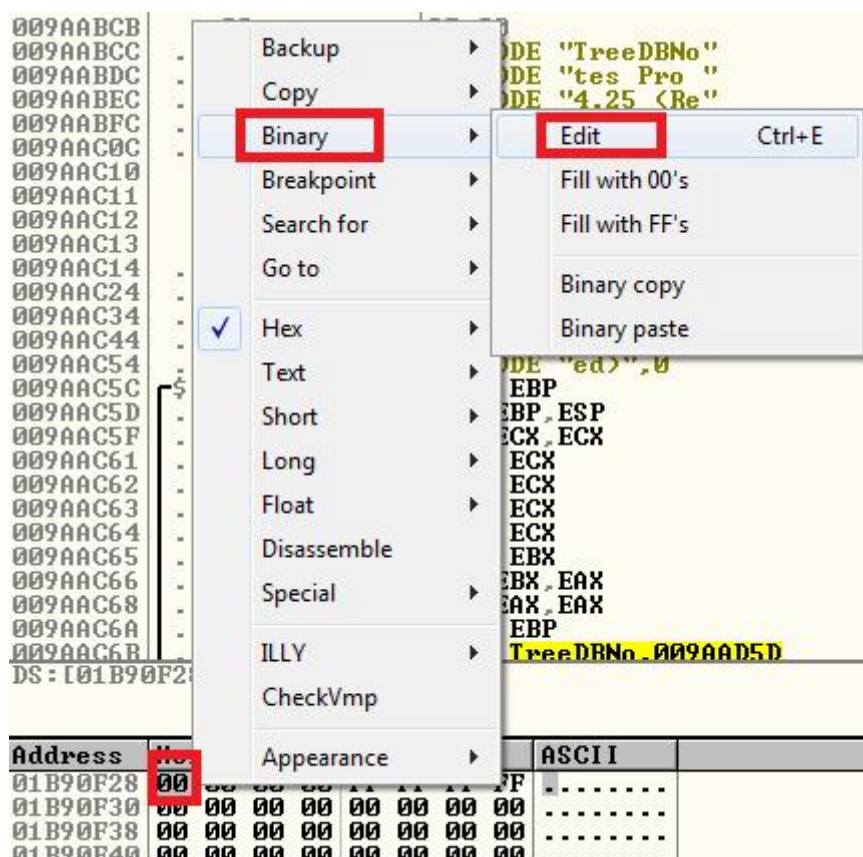
و در پنجره ی Dump:

009AAC6B | 68 5DAD9A00 | PUSH TreeDBNo.009AAB9E
 DS: [01B90F28]=00

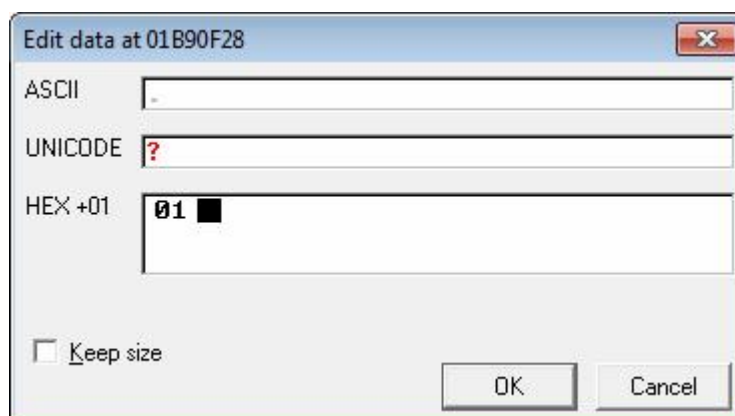
Address	Hex dump	ASCII
01B90F28	00 00 00 00 FF FF FF FF
01B90F30	00 00 00 00 00 00 00 00
01B90F38	00 00 00 00 00 00 00 00
01B90F40	00 00 00 00 00 00 00 00
01B90F48	00 00 00 00 00 00 00 00
01B90F50	01 00 00 00 01 00 00 00
01B90F58	0A 00 00 00 00 00 00 00
01B90F60	E4 4D B6 01 00 00 00 00	.M.....
01B90F68	00 00 00 00 20 EF 44 04D.
01B90F70	00 00 00 00 00 00 00 00
01B90F78	00 00 00 00 00 00 00 00
01B90F80	7C 6B 98 03 C8 6E 98 03	ik...n..
01B90F88	BC 93 BB 01 14 15 95 03
01B90F90	E4 6C 93 03 00 00 00 00	.l.....
01B90F98	26 00 00 00 B0 19 B1 01	&.....
01B90FA0	2C CE B1 01 0C 78 B3 01x..
01B90FA8	04 C0 B3 01 70 F9 B8 01p....
01B90FB0	00 00 00 00 00 00 00 00
01B90FB8	00 00 00 00 26 00 00 00&....
01B90FC0	94 C7 47 00 70 F9 B8 01	..G.p....
01B90FC8	00 00 00 00 00 00 00 00
01B90FD0	00 00 00 00 00 00 00 00

M1 M2 M3 M4 M5 Command:
 Memory Window 1 Start&0;x1B90F28 End&0;x1B90F27 Size&0;x0 Value

بله ☺ همانطور که میبینید این آدرس با آدرس قبلی متفاوت هست (در سیستم شما هم همینطور است) اگر فاطرتان باشد آدرس اولی 01BB0F28 بود و این آدرس 01B90F28 است ، درواقع هر بار که برنامه را ری استارت کنید این آدرس تغییر خواهد کرد، حال بروی آدرس 01B90F28 (همون که جدیده ☺) در پنجره ی دامپ راست کلیک کنید :



و بعد مقدار 00 را به 01 تغییر دهید :



سپس کلید Ok را بفشارید تا تغییرات اعمال شود :

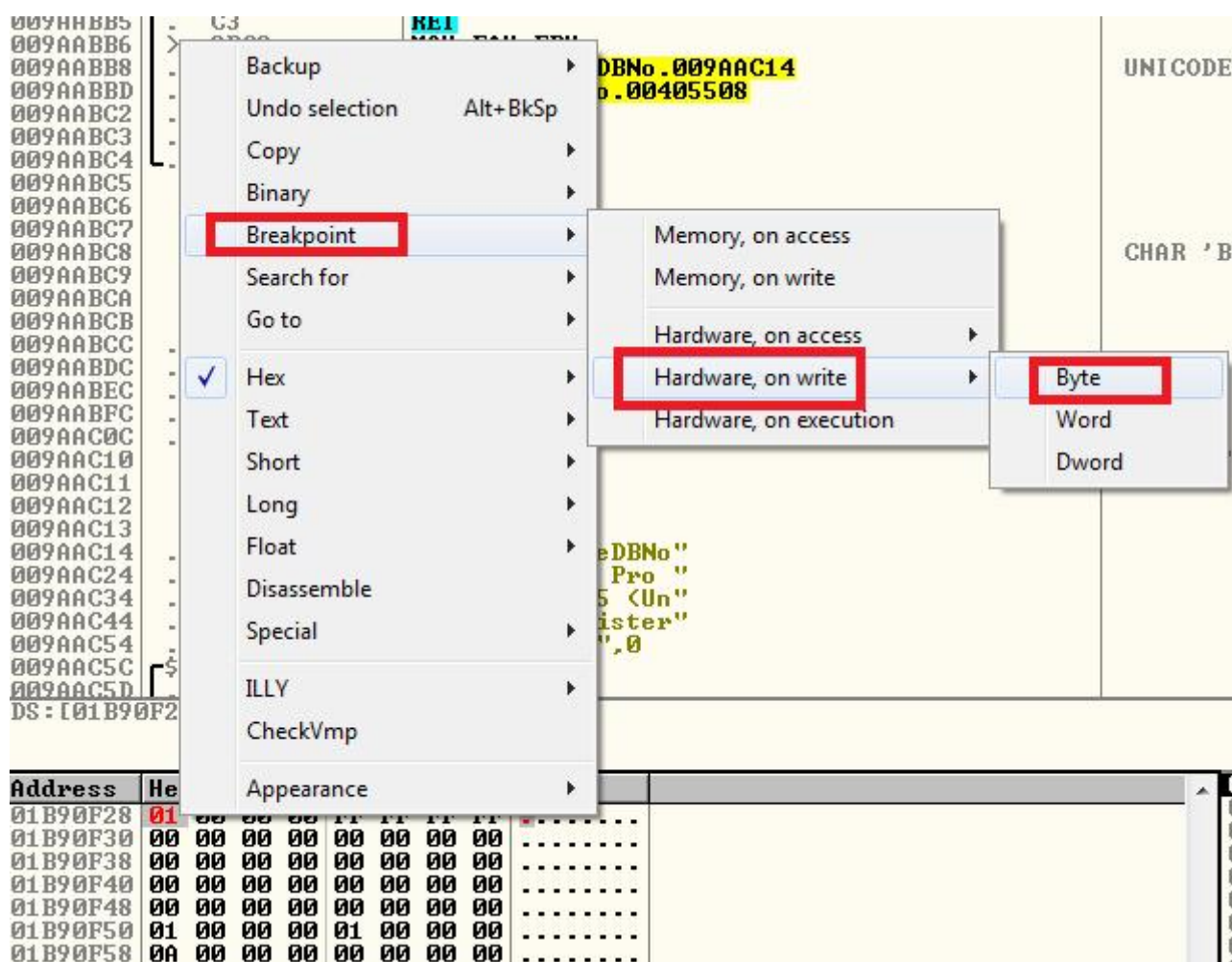
Address	Hex dump	ASCII
01B90F28	01 00 00 00 FF FF FF FF
01B90F30	00 00 00 00 00 00 00 00
01B90F38	00 00 00 00 00 00 00 00
01B90F40	00 00 00 00 00 00 00 00
01B90F48	00 00 00 00 00 00 00 00
01B90F50	01 00 00 00 01 00 00 00
01B90F58	0A 00 00 00 00 00 00 00

حالا یک بار کلید F8 را بفشارید میبینید که پرش به سمت Good Boy میرود چرا که مقدار [EAX+15B8] دیگر صفر نیست ، سپس با فشردن کلید F9 برنامه را اجرا کنید ، بله ، برنامه دوباره در همان آدرس قبلی 9AAB9E متوقف شده و مقدار آدرس 01B90F28 دوباره به 00 تغییر کرد ☹️. حال به Bad Boy میرویم ☺️ ، اما این تغییر مجدد 00 به 01 یعنی یک جای دیگر از برنامه (روتین چک مجدد دارد) ☺️

این روتین دوباره مقدار 01 را به 00 برمیگرداند ، پس باید این قسمت پیدا شود ، اما پطوری؟ سناریو اینگونه است ، اگر این چک دوم قادر است که بایت ما را پیدا کند و آن را به 00 تغییر دهد ، پس ما هم میتوانیم موقعیت آن را هنگامی که میخواهد بایت ما را صفر کند پیدا کنیم ☺️ برای این کار ما نیاز به یک hardware breakpoint داریم . که در آدرس حافظه ی 01B90F28 باید بروی بایت اول که الان 00 شده قرار دهیم ، این کار به olly میگوید موقعی که هرجای برنامه میخواهد در این آدرس چیزی بنویسد ، آن را متوقف کن ☺️

پس از یافتن olly به محل قسمتی میرود که قصد داشته بایت ما را به 00 تغییر دهد:

اگر کماکان در آدرس 01B90F28 (در سیستم شما متفاوت است) هستید و از آن عبور نکرده اید (در غیر اینصورت برنامه را ری استارت کنید و دوباره بروی Follow in Dump کلیک کنید و مقدار 00 را به 01 تغییر دهید) سپس بروی آدرس مورد نظر (در بایت اول 01) راست کلیک کنید و:



طبق تصویر بالا عمل کنید، اما باید بگوییم ما از خصوصیت **Byte** و **Write** در نقطه ی توقف سفت افزاری، استفاده کرده ایم چون میخواهیم ببینیم کجای برنامه میفواهد **یک بایت 01** واقع در آدرس **1B90F28** را با مقدار **00 جایگزین** کند و همچنین ما از نقطه ی توقف سفت افزاری استفاده کرده ایم چون برای برنامه خیلی مشکل است که بتواند آن را شناسایی کند، حال برنامه را باکلید F9 اجرا میکنیم :

009ADBFC	00	DB 00
009ADBFD	00	DB 00
009ADBF4	\$ 3A90 B8150000	CMP DL, BYTE PTR [EAX+15B8]
009ADBF5	74 06	JE SHORT TreeDBNo.009ADC02
009ADBF6	8890 B8150000	MOV BYTE PTR [EAX+15B8], DL
009ADC02	C3	RET
009ADC03	90	NOP

همانطور که میبینید olly در این آدرس متوقف شده و در پایین میبینید که علت این توقف نقطه ی توقف سفت افزاری ما بوده:



و همینطور در پنجره ی دامپ میبینید که مقدار بایت ما برابر با **00** شده :

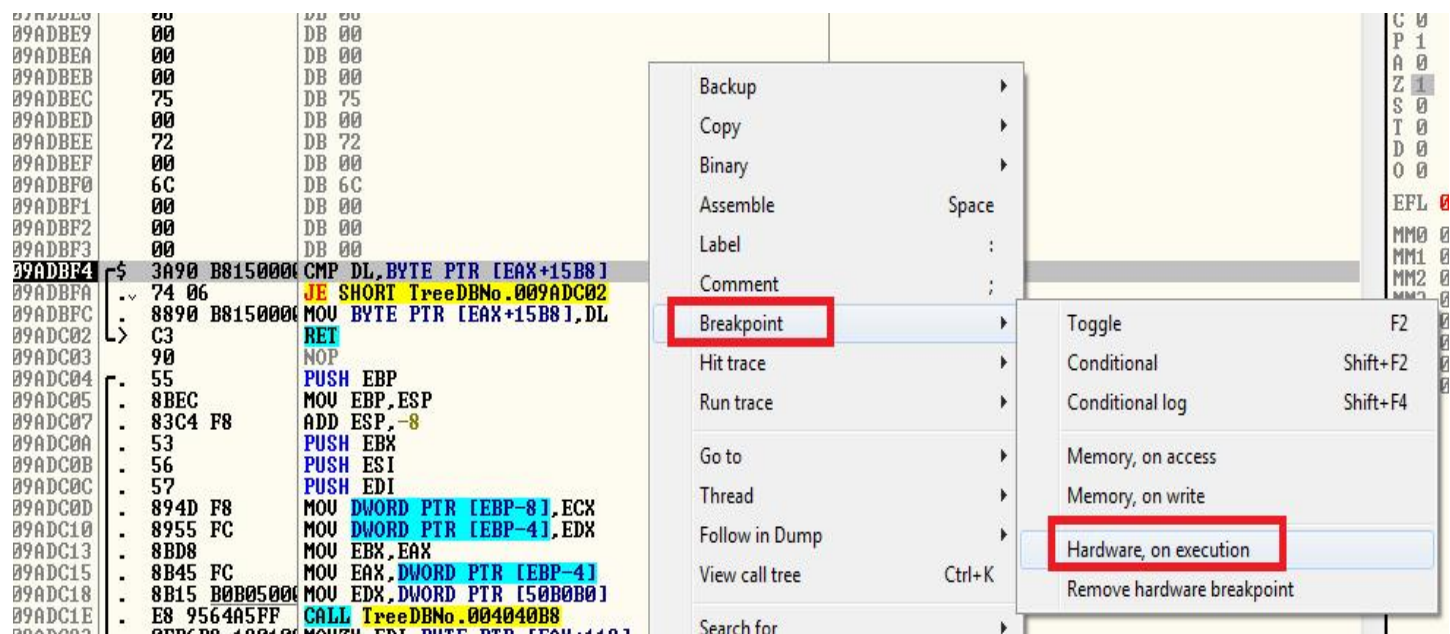
01B90F28	00 00 00 00	FF FF FF FF
01B90F30	00 00 00 00	00 00 00 00
01B90F38	00 00 00 00	00 00 00 00
01B90F40	00 00 00 00	00 00 00 00
01B90F48	00 00 00 00	00 00 00 00
01B90F50	01 00 00 00	01 00 00 00
01B90F58	0A 00 00 00	00 00 00 00
01B90F60	E4 4D B6 01	00 00 00 00	.M.....
01B90F68	01 00 00 00	20 EF 44 04D.
01B90F70	00 00 00 00	00 00 00 00
01B90F78	00 00 00 00	00 00 00 00
01B90F80	7C 6B 98 03	C8 6E 98 03	ik...n..
01B90F88	BC 93 BB 01	14 15 95 03
01B90F90	E4 6C 93 03	00 00 00 00	.l.....
01B90F98	98 0F B9 01	98 0F B9 01
01B90FA0	24 00 00 00	0C 78 B3 01	\$.x..
01B90FA8	04 C0 B3 01	70 F9 B8 01	...p...
01B90FB0	C4 AB 47 04	04 64 48 04	..G...dH.
01B90FB8	24 00 00 00	27 00 00 00	\$.

طریقه ی پچ :

اگر بفواهم برنامه را پچ کنیم لازم است محل توقف سفت افزاری را کمی مطالعه کنیم :

009ADBFC	00	DB 00
009ADBFD	00	DB 00
009ADBF4	\$ 3A90 B8150000	CMP DL, BYTE PTR [EAX+15B8]
009ADBF5	74 06	JE SHORT TreeDBNo.009ADC02
009ADBF6	8890 B8150000	MOV BYTE PTR [EAX+15B8], DL
009ADC02	C3	RET
009ADC03	90	NOP

در فضا 9ADBFC مقدار یک بایت از آدرس [EAX+15B8] (یادتان هست؟) که الان 00 است را با DL مقایسه میکند و در فضا بعد اگر مساوی بود به فضا 9ADC02 پرش میکنیم و در فضا بعد مقدار DL را به آدرس [EAX+15B8] منتقل میکند و فضا بعد هم یک دستور ساده ی بازگشت است، قبل از هر چیز ابتدا نقطه ی توقف سخت افزاری را پاک کنید برای اینکار از منوی -> "Debug" "Hardware breakpoints" را انتخاب و آن را Delete کنید، و اما مقدار DL=0 است چون ما در آدرس حافظه ایی 1B90F28 این مقدار را دیدیم که از یک به صفر تغییر کرد، پس پرش به سمت bad Boy صورت میگیرد، حالا دوباره نیاز به یک نقطه ی توقف سخت افزاری داریم که روی آدرس 9ADBFC گذاشته و قصد داریم قبل از اینکه در آدرس حافظه مقدار صفر را مقایسه کند متوقف شویم:



برنامه را اجرا کنید، اوه، متوقف نشد، شاید تعجب کنید که دلیل آن چیست، باید بگویم دلایل متعددی میتواند داشته باشد یکی از آن دلایل تغییر کد به صورت فصوصیت polymorphically (عکس العمل متفاوت از دو رویداد مشترک) در آینده میبشترفند های آنتی دیباگ این مسئله بررسی فواهد شد، حالا برنامه را ری استارت کنید و با کلید F9 برنامه را اجرا کنید:



برنامه در آدرسی که نقطه ی توقف سخت افزاری گذاشته بودیم متوقف شد. حال ما قبل از اینکه مرکتی انجام بشود در ممل مضمور داریم این فضا یک میکند که آیا ما رجیستر شده ایم یا نه، اگر نشده و بودیم مقدار صفر را در آدرس حافظه ی [EAX+15B8] میگذارد. و اگر رجیستر شده بودیم مقدار 01 یا هر مقدار دیگری بجز 00 را در آن آدرس میگذارد، پس اگر ما کاری کنیم که همیشه مقدار 01 را درون آدرس [EAX+15B8] بگذاریم مشکل حل میشود، حالا سوال بعدی این است که چه روشی بهترین گزینه است؟ فب همانطور که میدانیم مقدار 00 در آدرس [EAX+15B8] درون DL ریفته میشود، و در فضا 9ADBFC مقدار DL در آدرس حافظه ایی [EAX+15B8] قرار میگیرد، پس بهتر است که از قبل DL دارای مقدار 01 باشد © برای این کار ما به دو فضا اولی نیاز نداریم پس من آنها را هایلایت میکنم :

009ADBF2	00	DB 00
009ADBF3	00	DB 00
009ADBF4	3A90 B8150000	CMP DL, BYTE PTR [EAX+15B8]
009ADBFA	74 06	JE SHORT TreeDBNo.009ADC02
009ADBFC	8890 B8150000	MOV BYTE PTR [EAX+15B8], DL
009ADC02	C3	RET

و راست کلیک میکنم سپس: “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 the 'Binary' option is selected. A sub-menu is open, showing options like 'Edit', 'Fill with 00's', 'Fill with NOPs', 'Binary copy', and 'Binary paste'. The 'Fill with NOPs' option is highlighted.

و سپس:

009ADBF2	00	DB 00
009ADBF3	00	DB 00
009ADBF4	90	NOP
009ADBF5	90	NOP
009ADBF6	90	NOP
009ADBF7	90	NOP
009ADBF8	90	NOP
009ADBF9	90	NOP
009ADBFA	90	NOP
009ADBFB	90	NOP
009ADBFC	8890 B8150000	MOV BYTE PTR [EAX+15B8], DL
009ADC02	C3	RET
009ADC03	90	NOP

حالا بروی دستور العمل NOP در آدرس 9ADBF4 دبل کلیک کنید و دستور زیر را وارد کنید:

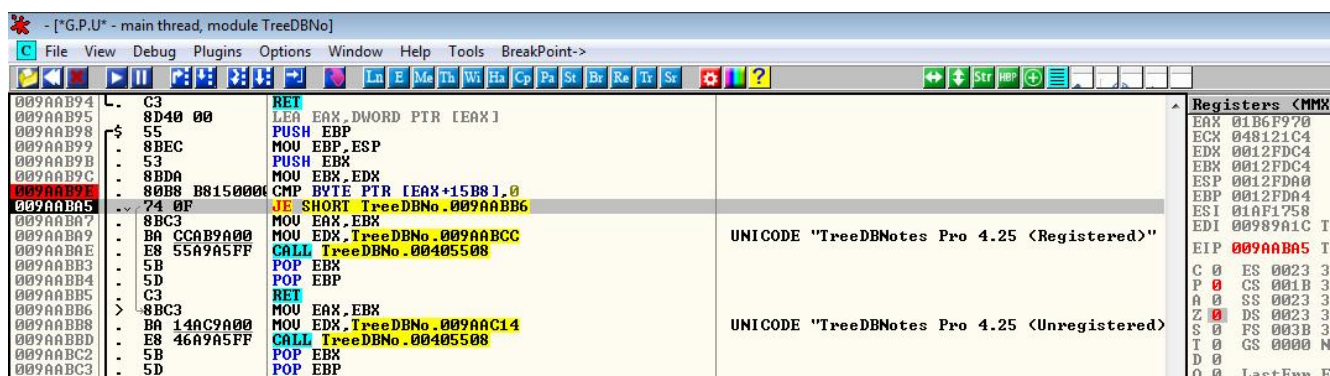
The screenshot shows a debugger window with assembly code. A dialog box titled 'Assemble at 009ADBF4' is open, showing the assembly code 'MOV DL,1'. The 'Fill with NOPs' checkbox is checked. The 'Assemble' button is highlighted.

وبعد دکمه ی Assemble را بفشارید و مطمئن شوید Fill with NOP's فعال باشد و در آخر دکمه ی Cancel را فشارید:

009ADBFB2	00	DB 00
009ADBFB3	00	DB 00
009ADBFB4	B2 01	MOV DL,1
009ADBFB6	90	NOP
009ADBFB7	90	NOP
009ADBFB8	90	NOP
009ADBFB9	90	NOP
009ADBFA0	90	NOP
009ADBFB0	90	NOP
009ADBFC0	L> 8890 B8150000	MOV BYTE PTR [EAX+15B8],DL
009ADC02	C3	RET
009ADC03	90	NOP

حالا هر موقع که این روتین فراخوانی بشود مقدار 1 درون DL حاضر است و در خط 9ADBFC این مقدار درون آدرس [EAX+15B8] ریخته و نتیجه این میشود که برنامه ریجیستر میشود

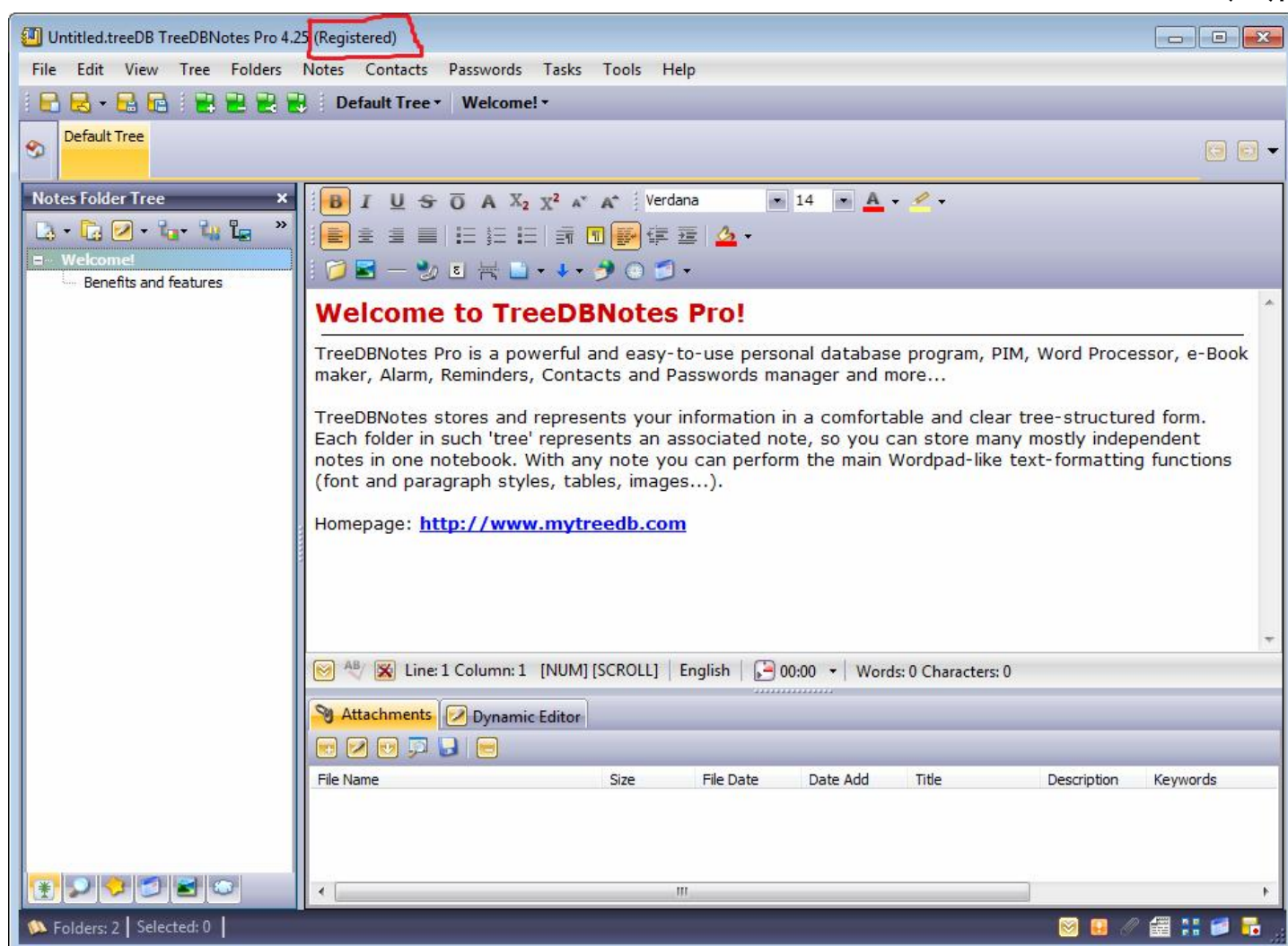
حالا کلید F9 را بفشارید ،میبیند که olly در روتین اصلی متوقف شده ،کلید F8 را یکبار بفشارید ، بله به نظر همه چیز درست است حالا کلید F9 را بفشارید تا برنامه کاملاً اجرا شود:



البته بخاطر نقاط توقف تا بارگذاری کامل حدود ۴ بار متوقف میشویم که پس از فشردن کلید F9 در هر بار توقف :



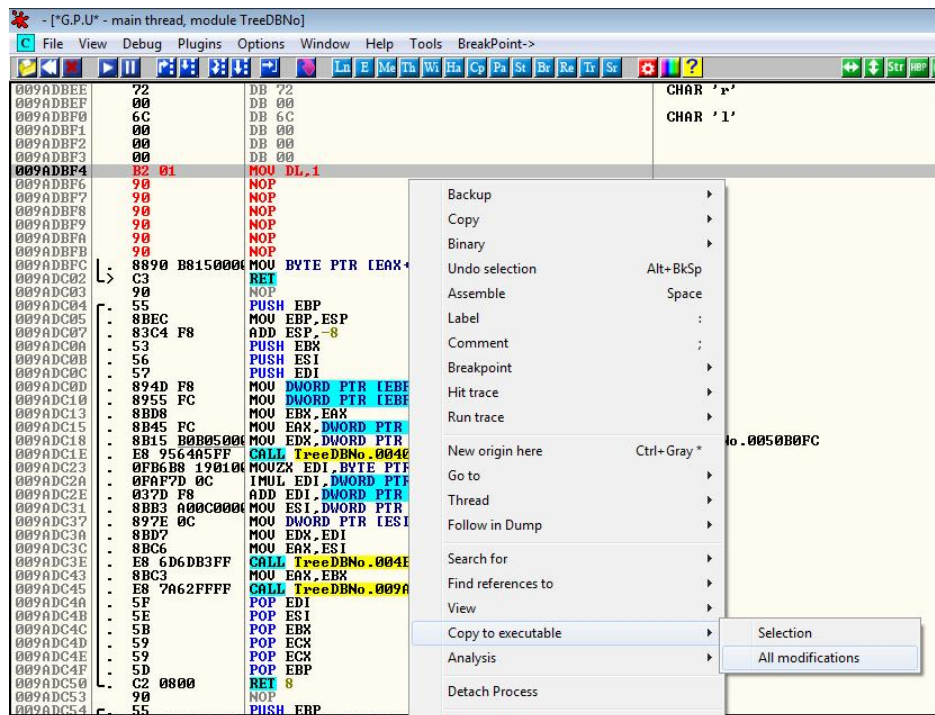
سپس بروی Close کلیک کنید هر موقع که olly متوقف شده (در نقاط توقف) با فشردن کلید F9 از آن عبور کنید تا برنامه کاملاً اجرا شود:



و در پنجره ی About میبینیم که برنامه رجیستر شده :



تبریک ☺ شما اولین برنامه ی واقعی را کرک کردید فقط یادتان نرود که نتیجه ی کارها را ذخیره کنید:



و همینطور Hardware breakpoints را از قسمت ("Debug" -> "Hardwarebreakpoints") پاک کنید :

