

第二十八章-Visual Basic 程序的破解-Part3

破解 VB 程序的又一手法

本章我们的实现对象是 CrackMe2。

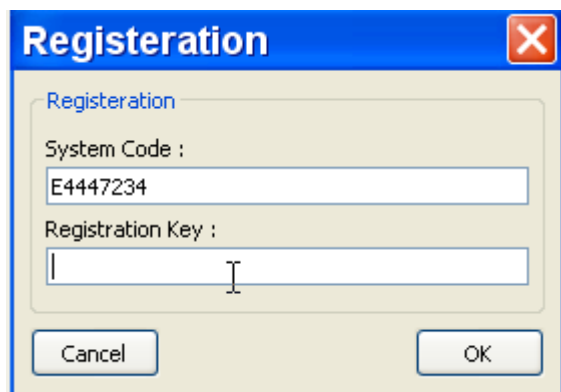


运行起来我们可以看到一个 NAG 窗口,通过上一章 4C 法我们可以很容易的剔除掉这个 NAG 窗口,其中与序列号相关的一部分涉及到了 PCODE,等我们介绍到了 PCODE 的时候再来讨论。

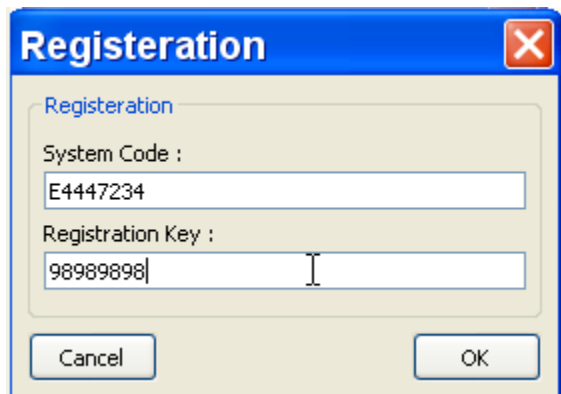
要找到这个 CrackMe 的序列号很简单,关键是如何剔除这个 NAG 窗口,通过 4C 法我们可以轻松的剔除掉 NAG 窗口,这里我就不再赘述了,大家可以自行尝试。这里我们来介绍另外一种方式。

首先我们来看看正确的序列号是多少。

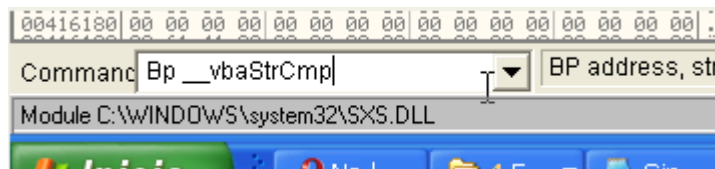
我们单击 Register 按钮。



出现了注册窗口,我们随便输入一个错误的序列号。



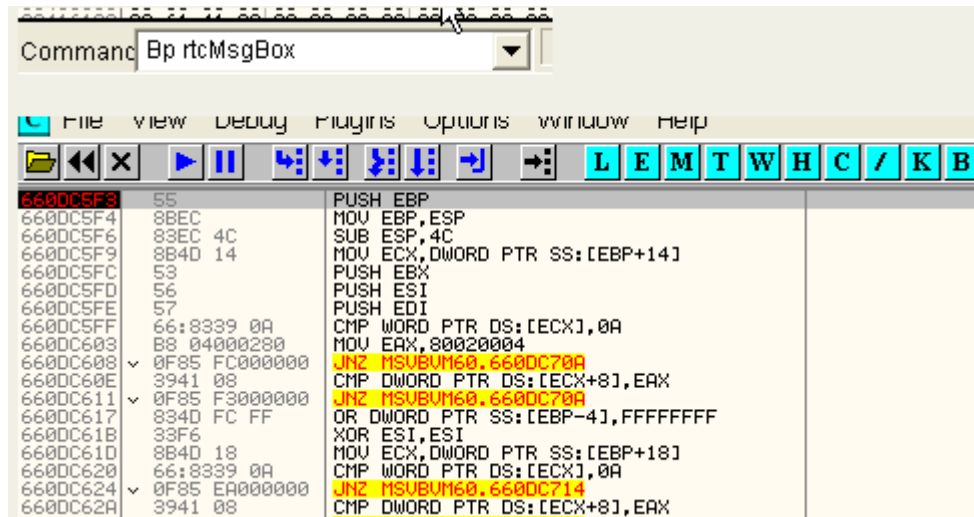
首先我们来看看是否是用 `__vbaStrCmp` 这个函数来进行字符串比较的。



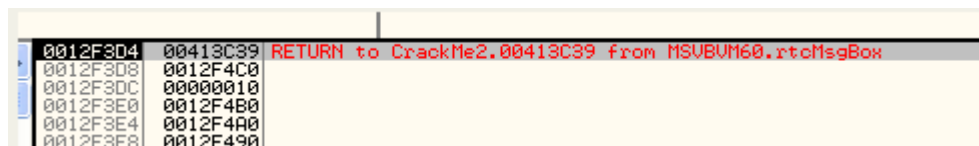
我们在命令栏中输入 bp __vbaStrCmp。

接着按 F9 键运行起来,我们会发现会断下来很多次,所以我们可以换种思路,从错误提示入手,看看能不能在弹出错误提示框的时候断下来,那么序列号的比较应该就在附近了。现在我们删除刚刚对 __vbaStrCmp 设置的断点,然后给弹出消息框的函数 rtcMsgBox 设置断点。

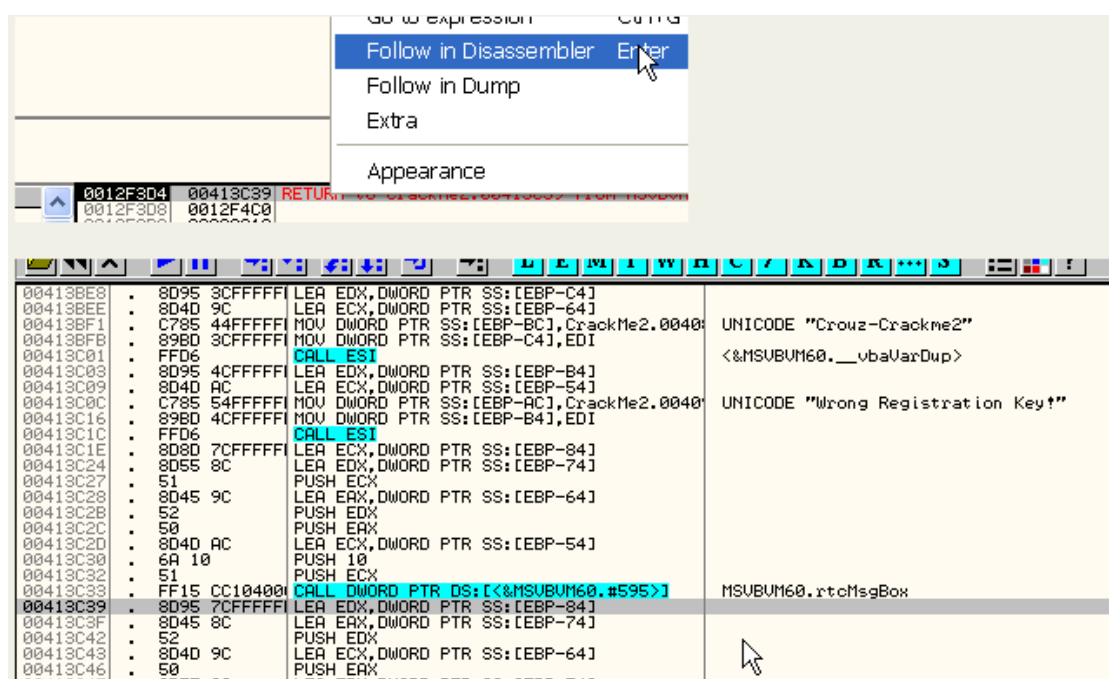
我们在命令栏中输入 bp rtcMsgBox。



我们单击 OK 按钮。断在了 rtcMsgBox 的入口处。我猜是提示输入的序列号错误。我们来到堆栈窗口看看调用来自于哪里。



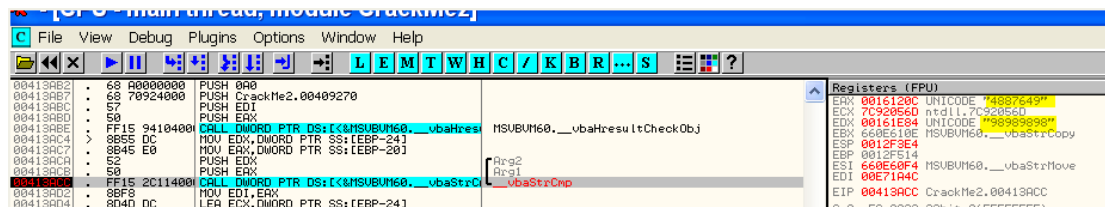
我们在反汇编窗口中定位到该返回地址处。



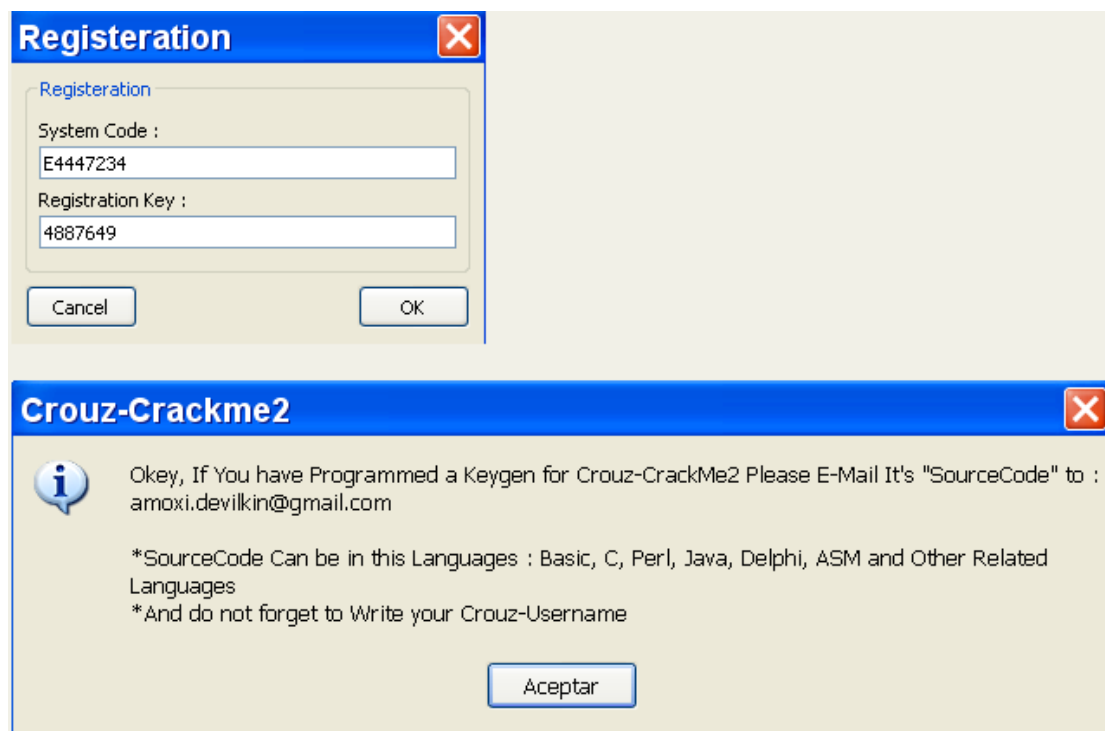
好了,现在我们来到返回地址处,我们看看前面几行有没有调用字符串比较之类的 API 函数。

00413AC4	8B55 DC	MOV EAX,DWORD PTR SS:[EBP-24]
00413AC7	8B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]
00413ACA	52	PUSH EDX
00413ACB	50	PUSH EAX
00413ACC	FF15 2C114000	CALL DWORD PTR DS:[<&MSUBUM60,___vbaStrCmp]
00413AD2	8BF8	MOV EDI,EAX
00413AD4	8D4D DC	LEA ECX,DWORD PTR SS:[EBP-24]
00413AD7	F7DF	NEG EDI
00413AD9	1BFF	SBB EDI,EDI
00413ADB	47	INC EDI

我们可以看到前面的确有进行字符串比较的 API 函数,我们给这处调用设置一个断点,我们按 F9 键运行起来,验证一下该处是不是我们要找的。

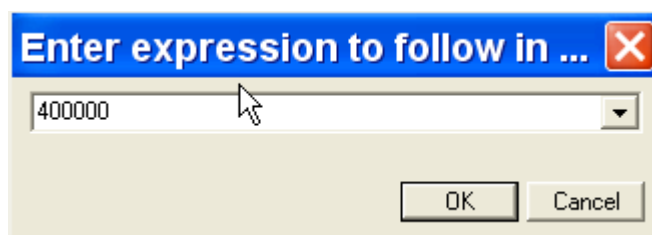


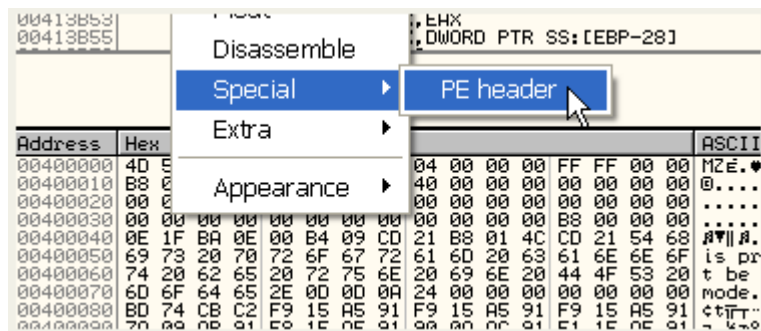
断了下来,我们可以看到正在进行字符串的比较,其中一个是我们输入的错误序列号。另一个我这里是 4887649,我们删除之前的断点,在注册窗口中输入这个字符串看看是否是正确的序列号。



我们可以看到的的确是正确的序列号。

接下来,我们来尝试剔除这个 NAG 窗口。由于我们可能需要修改这个 CrackMe,所以我们需要给这个 CrackMe 的代码段赋予写权限,我们首先在数据窗口中定位 400000 地址处。





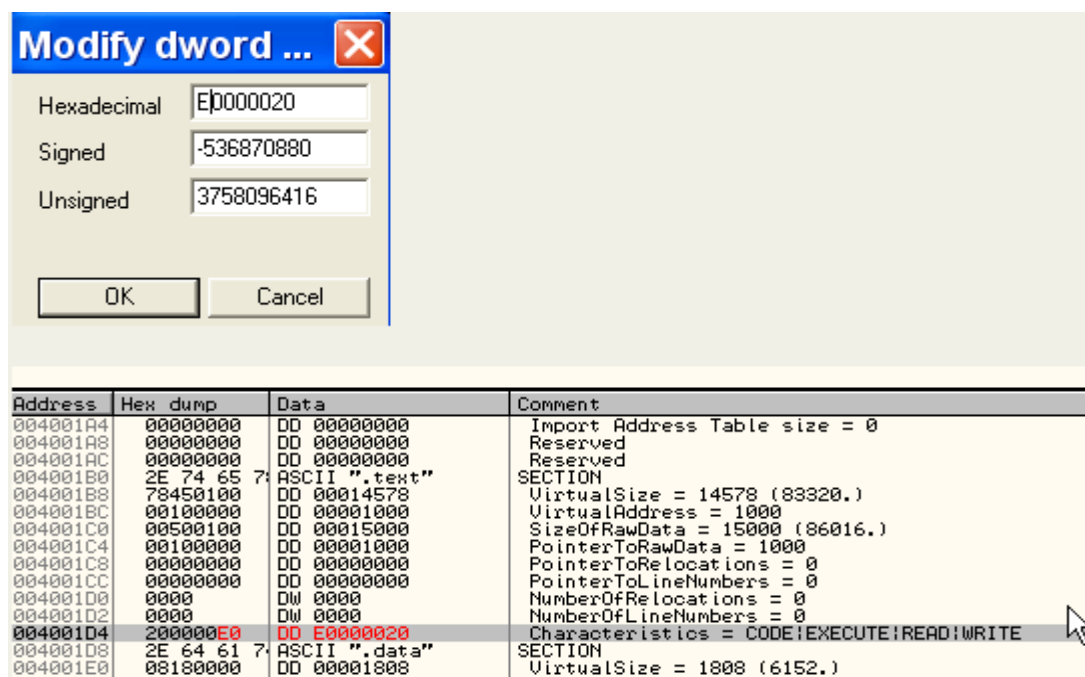
我们切换为 PE header 模式显示。

Address	Hex dump	Data	Comment
004000B5	00	DB 00	
004000B6	00	DB 00	
004000B7	00	DB 00	
004000B8	50 45 00 00	ASCII "PE"	PE signature (PE)
004000BC	4C01	DW 014C	Machine = IMAGE_FILE_MACHINE_I386
004000BE	0300	DW 0003	NumberOfSections = 3
004000C0	297E9142	DD 42917E29	TimeDateStamp = 42917E29
004000C4	00000000	DD 00000000	PointerToSymbolTable = 0
004000C8	00000000	DD 00000000	NumberOfSymbols = 0
004000CC	E000	DW 00E0	SizeOfOptionalHeader = E0 (224.)
004000CE	0F01	DW 010F	Characteristics = EXEQTABLE IMAGE!32B1

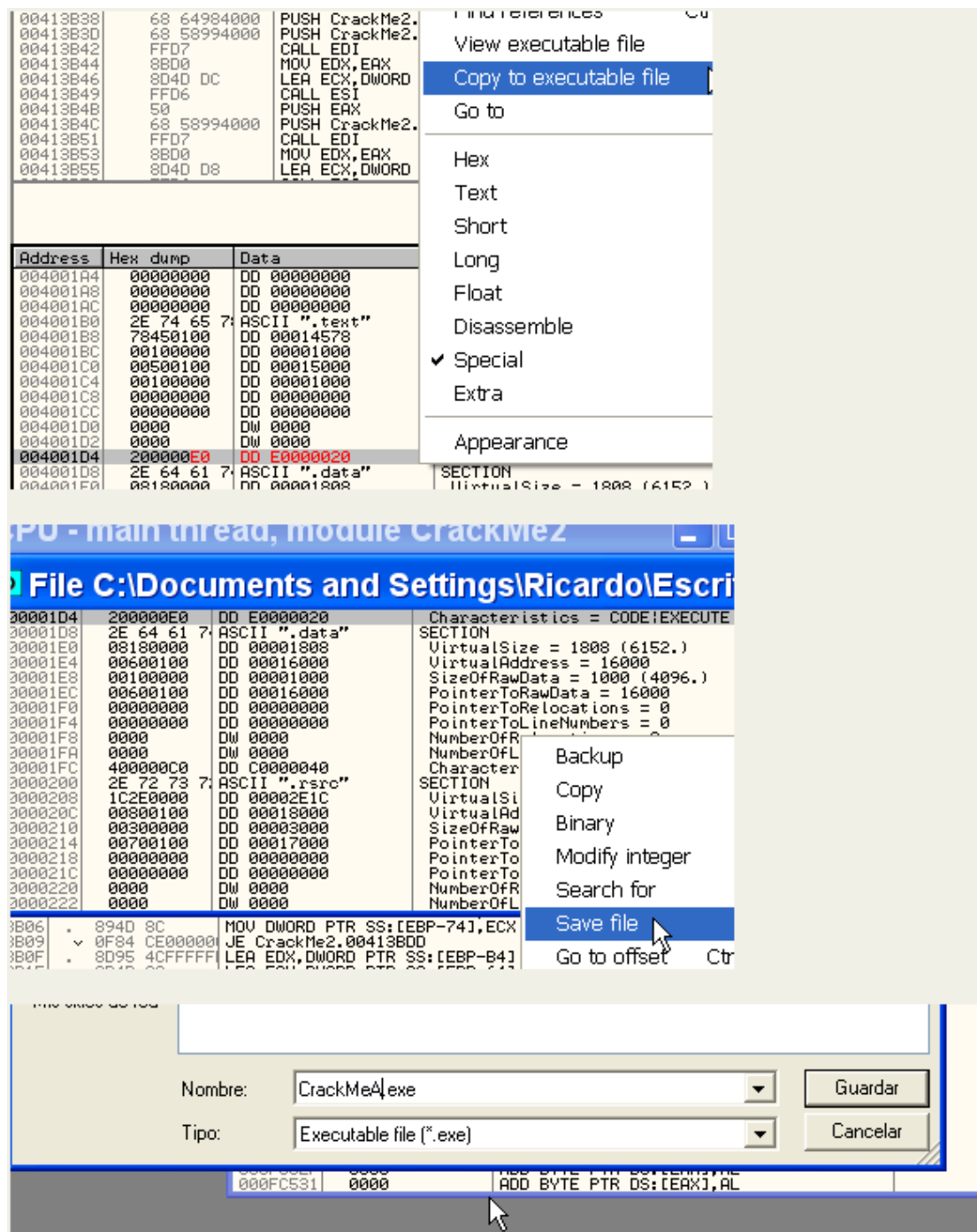
往下我们可以看到 PE Signature 标志,我们继续往下定位到第一个区段。

Address	Hex dump	Data	Comment
0040019C	00000000	DD 00000000	Delay Import Descriptor size = 0
004001A0	00000000	DD 00000000	COM+ Runtime Header address = 0
004001A4	00000000	DD 00000000	Import Address Table size = 0
004001A8	00000000	DD 00000000	Reserved
004001AC	00000000	DD 00000000	Reserved
004001B0	2E 74 65 74	ASCII ".text"	SECTION
004001B8	78450100	DD 00014578	VirtualSize = 14578 (83320.)
004001BC	00100000	DD 00001000	VirtualAddress = 1000
004001C0	00500100	DD 00015000	SizeOfRawData = 15000 (86016.)
004001C4	00100000	DD 00001000	PointerToRawData = 1000
004001C8	00000000	DD 00000000	PointerToRelocations = 0
004001CC	00000000	DD 00000000	PointerToLineNumbers = 0
004001D0	0000	DW 0000	NumberOfRelocations = 0
004001D2	0000	DW 0000	NumberOfLineNumbers = 0
004001D4	20000000	DD 60000020	Characteristics = CODE!EXECUTE!READ
004001D8	2E 64 61 74	ASCII ".data"	SECTION
004001DC	00180000	DD 00001800	VirtualSize = 1800 (6152.)

这里我们可以看到 Characteristics 字段,我们将为其修改为 E0000020 的话,就可以写入代码段了。



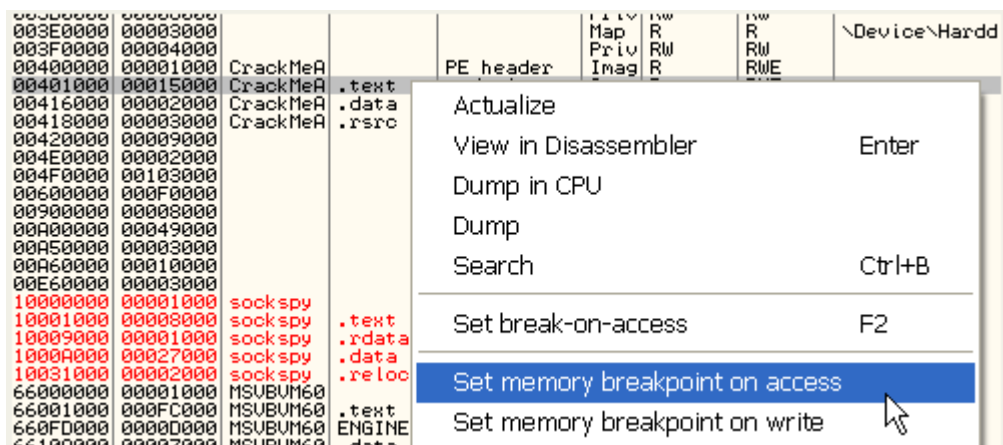
好了,现在我们将刚刚所做的修改保存到文件。



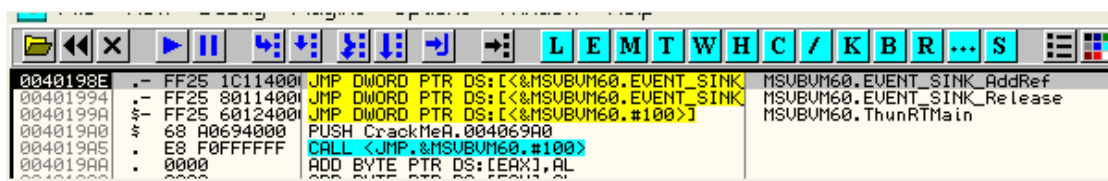
另存为 CrackMeA.exe

我们用 olly_parcheado_para_vb 这个 Patch 过的 OD 来加载这个 CrackMeA。

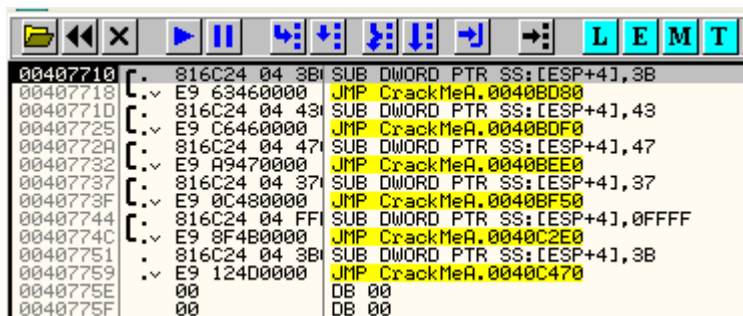
接着给代码段设置内存访问断点(实际上是内存执行断点)。



我们多按几次 F9 键运行起来就可以来到这里。

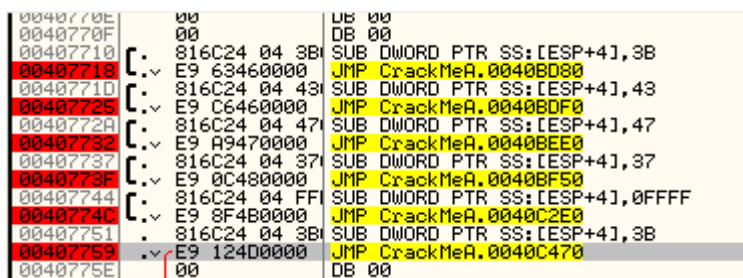


我们继续按 F9 键运行,停在了熟悉的多分支处,将会转向去执行程序的不同部分。

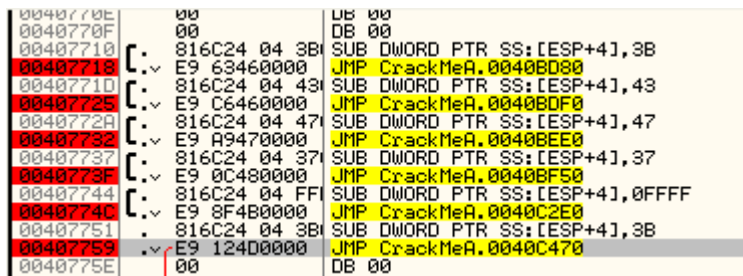


第一次,停在了 407710 处,将 JMP 到 40BD80 地址处-程序将执行的第一部分,我们来看看再次停在其他 JMP 分支之前会不会弹出

NAG 窗口,我们删除之前设置的内存访问断点,接着给该多分支的各个 JMP 指令处都设置断点。



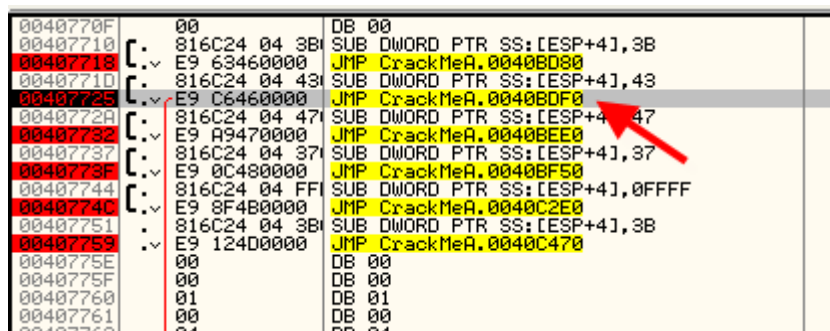
我们来看看断在其他 JMP 指令之前会不会弹出 NAG 窗口,运行起来。



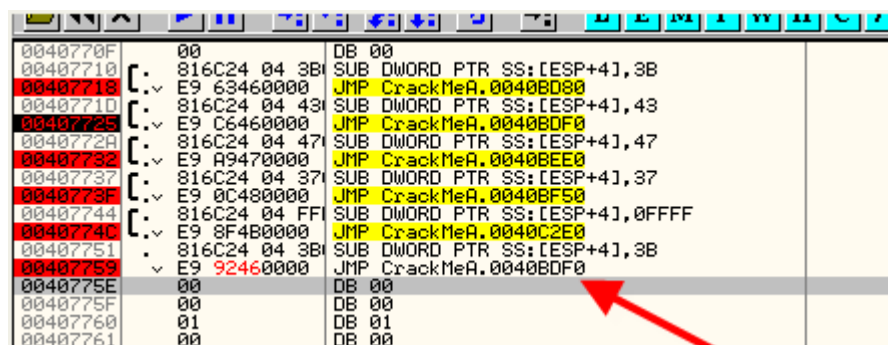
我们可以看到断在了 407759 处,将跳转到 40C470 处,并没有弹出 NAG 窗口,我们继续运行。



弹出了 NAG 窗口,说明是在 40C470 这个分支中弹出的 NAG 窗口,我们单击 Register 按钮。



断在了第二个 JMP 处,所以说 NAG 是在跳转到 40BDF0 之前弹出的,我们直接跳过 NAG 窗口。



我们将 407759 处的 JMP 40C4F0 修改为 JMP 40BDF0 看看会发生什么。

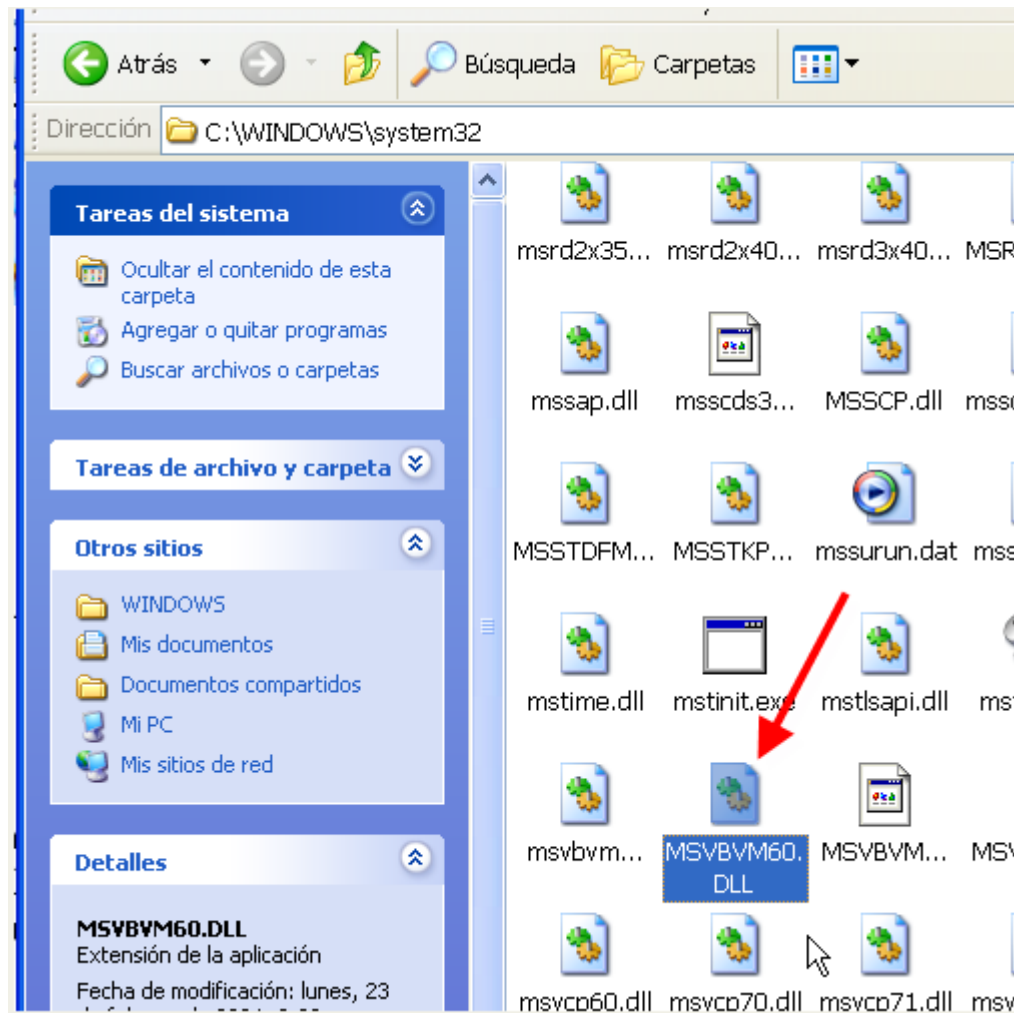
我们保存到修改到文件,然后直接运行起来。



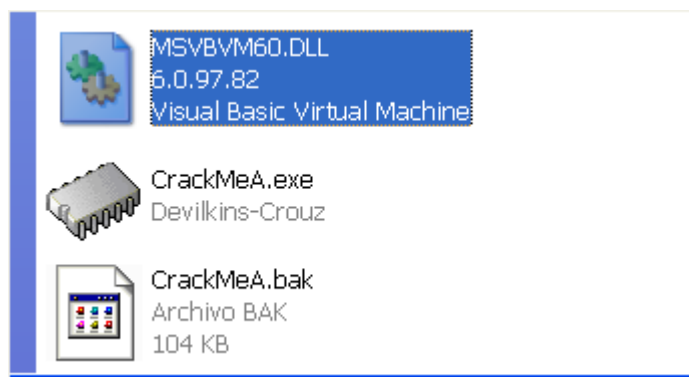
我们可以看到刚刚做的修改并没有剔除掉 NAG 窗口,但是原本是需要我们单击 NAG 窗口的 Register 按钮才会弹出注册窗口的,现在是 NAG 窗口和注册窗口一起弹了出来。

我们刚刚修改 JMP 指令并没有完全解决问题,别无选择了,我们只能修改 VB 的 DLL 的,很多人说我们不能修改系统库文件,会导致其他程序不能用的,实际上我们可以将 VB 的库文件拷贝至跟待破解的目标文件同一个目录即可,那么修改了的 VB 库文件只会的当前文件夹的目标文件起作用,操作系统里的其他应用程序还是会继续使用 system32 目录下的 VB 库文件,两者不会冲突。

那么我们需要拷贝那个 VB 库文件呢?



我们到 system32 目录下拷贝 MSVBVM60.dll 这个文件。

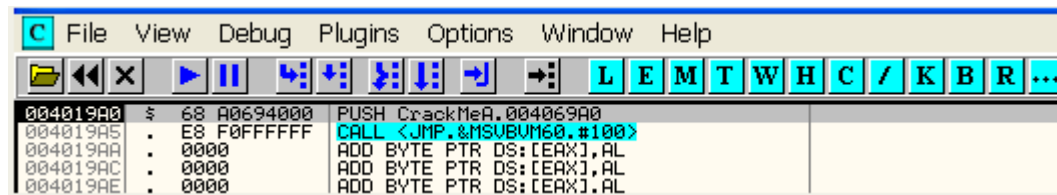


首先将 CrackMeA 做个备份。

目标文件如果需要加载特定的库文件的话,首先会搜索当前目录下有没有,如果当前目录下没有就直接加载,如果当期目录下没有

有,就会继续前往 system32 目录下搜索,我们当前就是这种情况。

我们用 OllyDbg 加载该 CrackMeA,这里我们使用原版的 OllyDbg,因为 Patch 过的那个 OD 添加代码有时候会失败。



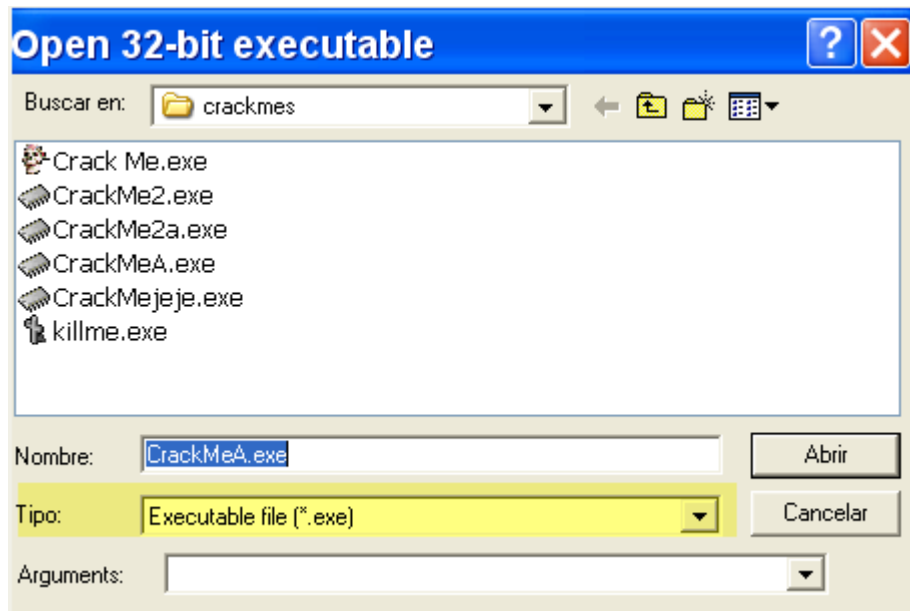
我们单击工具栏中 E 按钮打开模块列表窗口,看看是加载的当前目录下的 MSVBVM60.dll 还是 system32 下的。



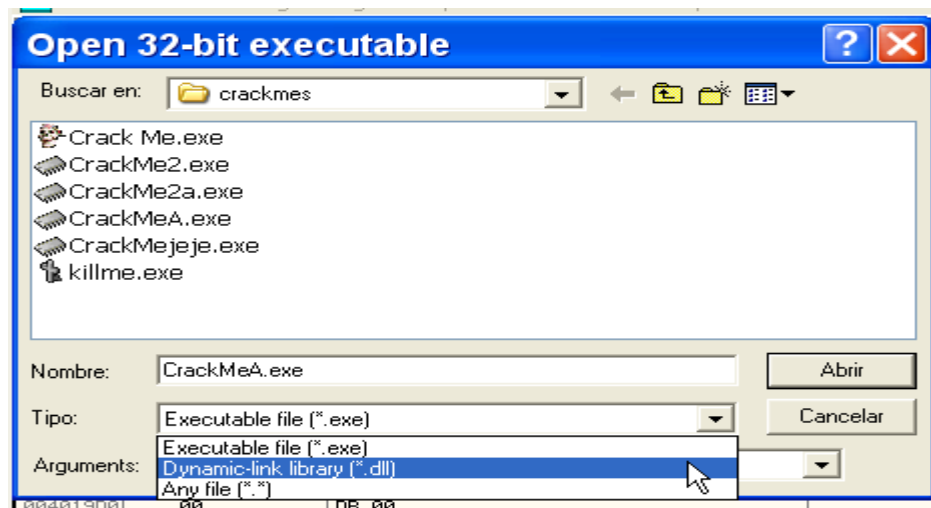
我们可以看到加载是当前目录下的 MSVBVM60.dll。

好了,现在我们需要确保对 MSVBVM60.dll 的代码段有写权限,我们做如下操作:

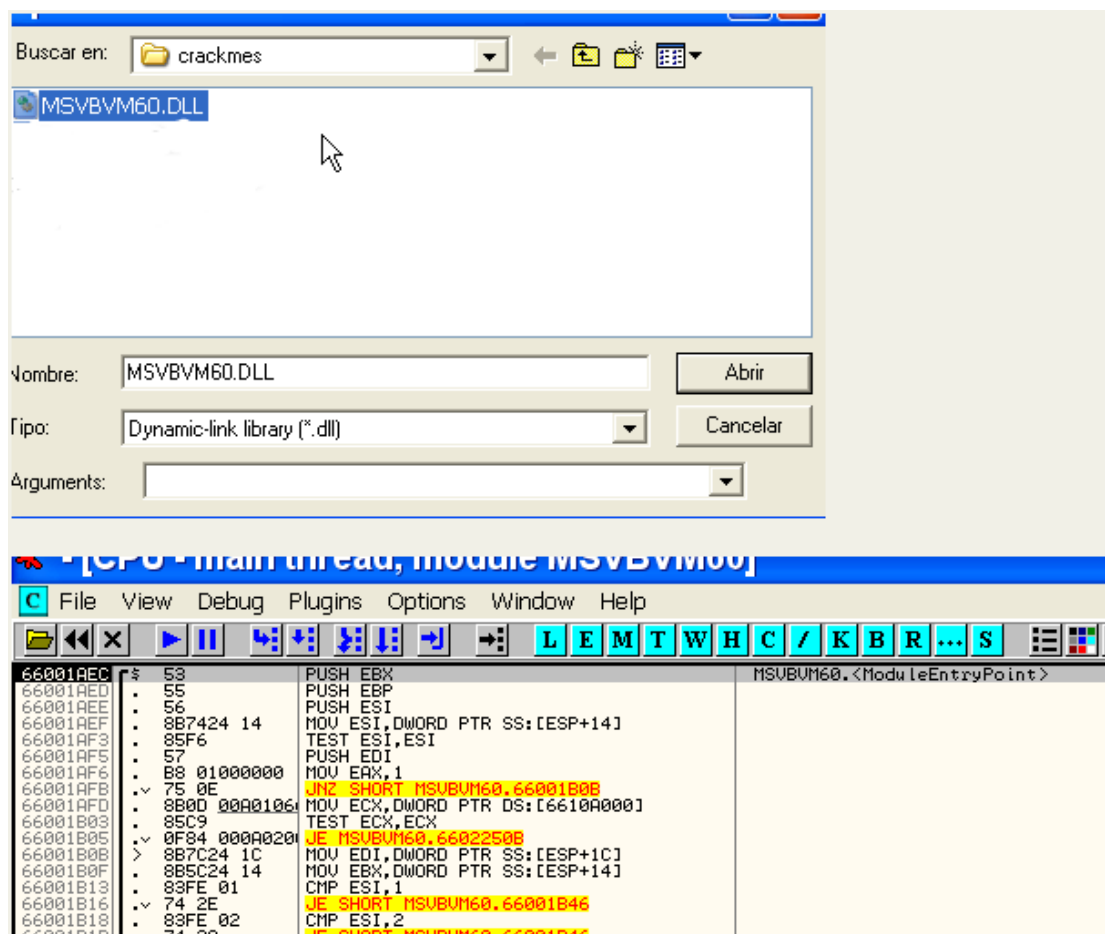
我们打开一个 OD 然后单击菜单栏中 Open 选项。



默认显示的 exe 文件,我们将文件类型改为 DLL。



将下拉选项选中 Dynamic-Link library(*.dll)就能打开 DLL 文件了,我们打开 MSVBVM60.dll。



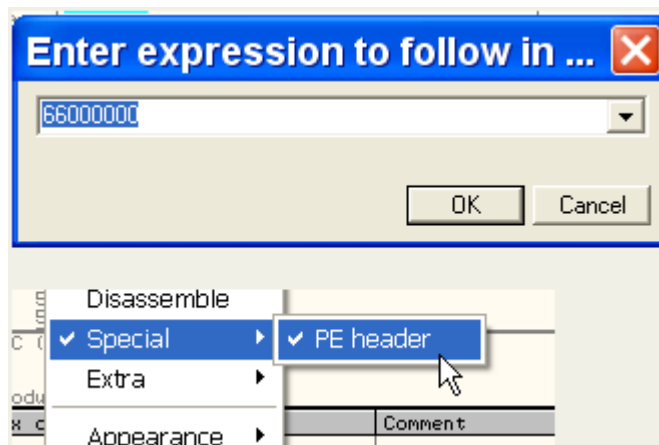
我们停在了该动态库的入口点处。

现在我们定位该动态库的头部(并不是像定位 CrackMeA 的头部那样直接定位到 400000 地址处),我们来看看加载了哪些模块,以及映像基址是多少。我们单击工具栏中的 E 按钮。

Base	Size	Entry	Name	File version	Path
00400000	00000000	00410070	LODDLL		C:\odbc\lib\LODDLL.EXE
7C901234	00000000	10000000	lockspv		C:\WINDOWS\system32\lockspv.dll
66000000	00152000	66001AEC	MSUBVM60	6.00.9782	C:\Documents and Settings\Usuario\Inicio\27-INTRODUCCION AL CRACKING CON OLLYDBG PARTE 27\crackmes\MSUBVM60.DLL
77D10000	00000000	77D10000	OLEAUT32	5.1.2600.2180	C:\WINDOWS\system32\OLEAUT32.dll
774D0000	00130000	774D0000	ole32	5.1.2600.2726	C:\WINDOWS\system32\ole32.dll
77B10000	00050000	77B10000	nsrvrt	5.1.2600.2180	C:\WINDOWS\system32\ntsvrt.dll
77010000	00090000	77010000	USER32	5.1.2600.2622	C:\WINDOWS\system32\USER32.dll
770A0000	000A0000	770A0000	ADUAP132	5.1.2600.2180	C:\WINDOWS\system32\ADUAP132.dll
77E50000	00091000	77E50000	RPCRT4	5.1.2600.2180	C:\WINDOWS\system32\RPCRT4.dll
77F10000	00047000	77F10000	GDI32	5.1.2600.2818	C:\WINDOWS\system32\GDI32.dll
7C800000	00101000	7C800000	kernel32	5.1.2600.2180	C:\WINDOWS\system32\kernel32.dll
7C910000	00060000	7C910000	ntdll	5.1.2600.2180	C:\WINDOWS\system32\ntdll.dll

我这里 MSVBVM60.dll 的基地址是 66000000。可能跟你机器上的不一样。

我们在数据窗口中定位到这个地址。



定位到头部以后我们切换到 PE header 模式显示,接着往下拉到 .text 区段。

Address	Hex dump	Data	Comment
66000164	00000000	DD 00000000	Delay Import Descriptor size = 0
66000168	00000000	DD 00000000	COM+ Runtime Header address = 0
6600016C	00000000	DD 00000000	Import Address Table size = 0
66000170	00000000	DD 00000000	Reserved
66000174	00000000	DD 00000000	Reserved
66000178	2E 74 65 74	ASCII ".text"	SECTION
66000180	DAB20F00	DD 000FB2DA	VirtualSize = FB2DA (1028826.)
66000184	00100000	DD 00001000	VirtualAddress = 1000
66000188	00C00F00	DD 000FC000	SizeOfRawData = FC000 (1032192.)
6600018C	00100000	DD 00001000	PointerToRawData = 1000
66000190	00000000	DD 00000000	PointerToRelocations = 0
66000194	00000000	DD 00000000	PointerToLineNumbers = 0
66000198	0000	DW 0000	NumberOfRelocations = 0
6600019A	0000	DW 0000	NumberOfLineNumbers = 0
6600019C	20000020	DD 00000020	Characteristics = CODE!EXECUTE!READ
660001A0	45 4E 47 44	ASCII "ENGINE"	SECTION
660001A8	F5CD0000	DD 0000CDF5	VirtualSize = CDF5 (52725.)

将 Characteristics 字段的值修改为 E0000020,让代码段具有写权限。

Modify dword ...

Hexadecimal

Signed

Unsigned

OK Cancel

Address	Hex dump	Data	Comment
66000164	00000000	DD 00000000	Delay Import Descriptor size = 0
66000168	00000000	DD 00000000	COM+ Runtime Header address = 0
6600016C	00000000	DD 00000000	Import Address Table size = 0
66000170	00000000	DD 00000000	Reserved
66000174	00000000	DD 00000000	Reserved
66000178	2E 74 65 74	ASCII ".text"	SECTION
66000180	DAB20F00	DD 000FB2DA	VirtualSize = FB2DA (1028826.)
66000184	00100000	DD 00001000	VirtualAddress = 1000
66000188	00C00F00	DD 000FC000	SizeOfRawData = FC000 (1032192.)
6600018C	00100000	DD 00001000	PointerToRawData = 1000
66000190	00000000	DD 00000000	PointerToRelocations = 0
66000194	00000000	DD 00000000	PointerToLineNumbers = 0
66000198	0000	DW 0000	NumberOfRelocations = 0
6600019A	0000	DW 0000	NumberOfLineNumbers = 0
6600019C	200000E0	DD E0000020	Characteristics = CODE!EXECUTE!READ!WRITE
660001A0	45 4E 47 44	ASCII "ENGINE"	SECTION
660001A8	F5CD0000	DD 0000CDF5	VirtualSize = CDF5 (52725.)
660001AC	00000000	DD 00000000	VirtualAddress = F0000

保存修改到文件。

66001B42 5B PU

66001B43 C2 0C00 RE

66001B46 8B00 04A0106 MO

66001B4C 85C9 TE

66001B4E 0F85 C009020 UN

66001B54 85C0 TE

66001B56 0F84 C209020 JE

66001B5C 57 PU

66001B5D 56 PU

EBX=66001AEC (MSUBUM60.<ModuleEntryPoint>)

MSUBUM60.<ModuleEntryPoint>

Address	Hex dump	Data	Comment
66000164	00000000	DD 000	
66000168	00000000	DD 000	
6600016C	00000000	DD 000	
66000170	00000000	DD 000	
66000174	00000000	DD 000	
66000178	2E 74 65 74	ASCII	
66000180	DAB20F00	DD 000	
66000184	00100000	DD 000	
66000188	00C00F00	DD 000	
6600018C	00100000	DD 000	
66000190	00000000	DD 000	
66000194	00000000	DD 000	
66000198	0000	DW 000	
6600019A	0000	DW 000	
6600019C	200000E0	DD E0000020	Characteristics
660001A0	45 4E 47 44	ASCII "ENGINE"	SECTION
660001A8	F5CD0000	DD 0000CDF5	VirtualSize = CD
660001AC	00000000	DD 00000000	VirtualAddress =

View executable file

Copy to executable file

Go to

Hex

Text

Short

Long

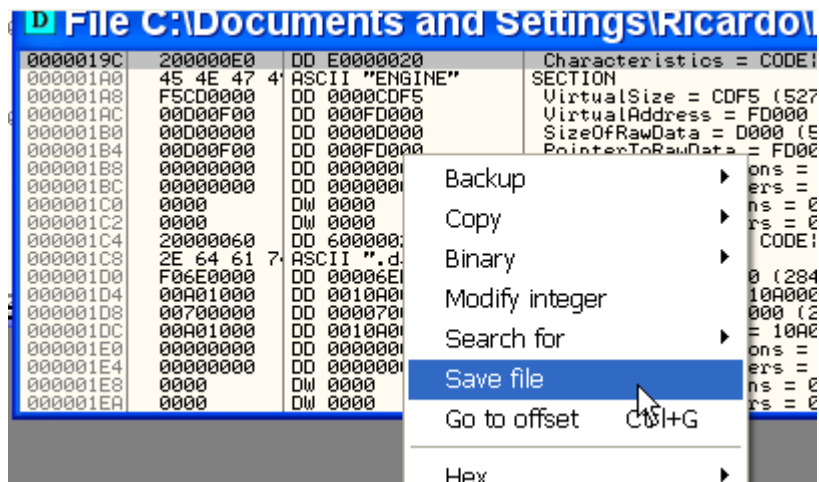
Float

Disassemble

Special

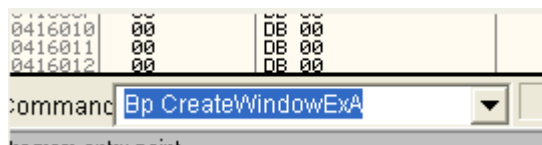
Extra

Appearance

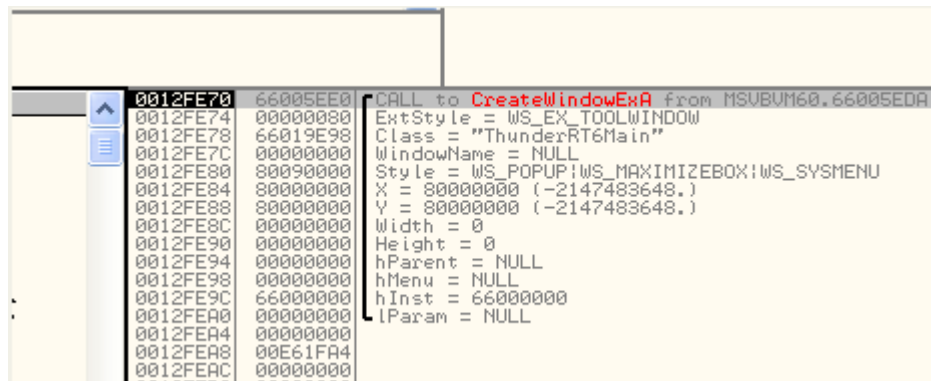


这里我们不重命名,因为重命名后 CrackMe 就不会加载了。

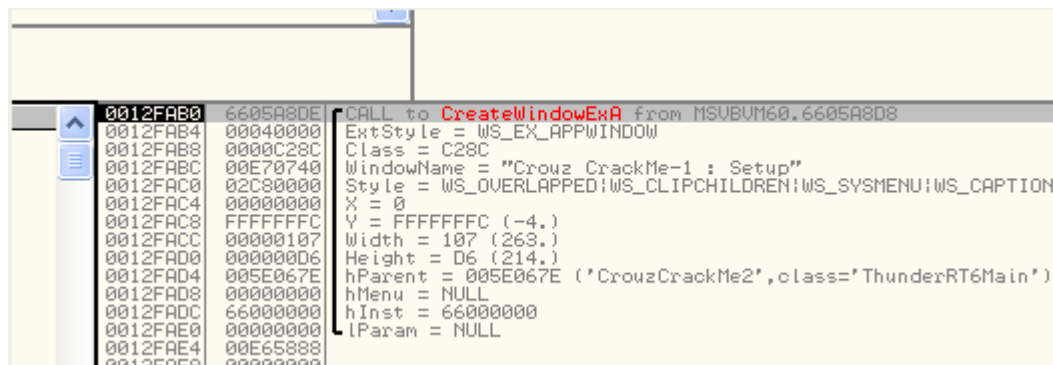
我们用 Patch 过的 OllyDbg 加载 CrackMeA,由于注册窗口和 NAG 窗口是一起弹出来的,所以我们给创建窗口的 API 函数 CreateWindowExA 设置一个断点。



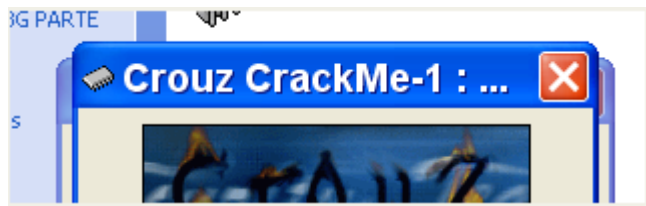
运行起来。



由于会创建子窗口所以会断下来几次,我多按 F9 键几次,直到创建 NAG 窗口为止。



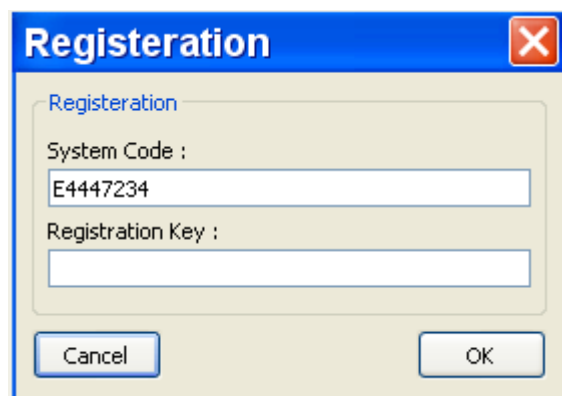
判断是否是创建 NAG 窗口很简单,根据窗口标题名就可以很容易的看出来。



可以看到现在正在创建 NAG 窗口,我们将其窗口风格值修改为 40000000(子窗口风格),看看会发生什么。

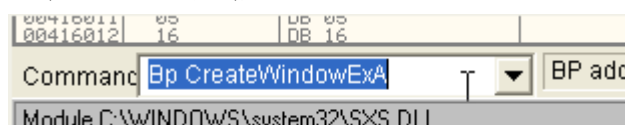


我们可以看到已经将窗口风格修改为 WS_CHILD,你可以尝试输入不同的值,现在我们可以删除所有断点运行起来。



我们可以看到只弹出了一个注册窗口,并没有弹出 NAG 窗口。如果不修改 MSVBVM60.dll 想要剔除这个 NAG 窗口就很复杂了,我们手工从系统目录复制一个 MSVBVM60.dll 将其置于与目标程序同一目录,通过修改这个文件并不会影响到其他应用程序,嘿嘿。

好了,我们重复之前的步骤,依然还是先给 CreateWindowExA 这个函数设置一个断点。



0012FAB0	6605A8DE	CALL to CreateWindowExA from MSVBVM60.6605A8D8
0012FAB4	00040000	ExtStyle = WS_EX_APPWINDOW
0012FAB8	0000C28C	Class = C28C
0012FABC	00E70740	WindowName = "Crouz CrackMe-1 : Setup"
0012FAC0	02C80000	Style = WS_OVERLAPPED WS_CLIPCHILDREN WS_SYSMENU WS_CAPTION
0012FAC4	00000000	X = 0
0012FAC8	FFFFFFFC	Y = FFFFFFFC (-4.)
0012FACC	00000107	Width = 107 (263.)
0012FAD0	000000D6	Height = D6 (214.)
0012FAD4	0060067E	hParent = 0060067E ('CrouzCrackMe2',class='ThunderRT6Main')
0012FAD8	00000000	hMenu = NULL
0012FADC	66000000	hInst = 66000000
0012FAE0	00000000	lParam = NULL
0012FAE4	00E65888	
0012FAE8	00000000	

从堆栈中的信息我们可以看出调用来自于 MSVBVM60.DLL,我们定位到调用处。

6605A8CE	FFB6 84000000	PUSH DWORD PTR DS:[ESI+84]	WindowName
6605A8D4	57	PUSH EDI	Class
6605A8D5	FF75 F4	PUSH DWORD PTR SS:[EBP-C]	ExtStyle
6605A8D8	FF15 E8140066	CALL DWORD PTR DS:[&USER32.CreateWindowExA]	CreateWindowExA
6605A8DE	8BF8	MOV EDI,EAX	
6605A8E0	8B46 28	MOV EAX,DWORD PTR DS:[ESI+28]	
6605A8E3	8B00	MOV EAX,DWORD PTR DS:[EAX]	
6605A8E5	F640 03 08	TEST BYTE PTR DS:[EAX+3],8	

我们清除掉 CreateWindowExA 入口处的断点,接着给 6605A8D8 处的 CALL CreateWindowExA 设置一个断点。我们重新启动程序,接着运行起来。

6605A8AF	FF35 D0E61066	PUSH DWORD PTR DS:[6610E6D0]	MSVBVM60.#1374
6605A8B5	FF75 0C	PUSH DWORD PTR SS:[EBP+C]	
6605A8B8	E8 A1F3FFFF	CALL MSVBVM60.66059C5E	
6605A8BD	50	PUSH EAX	hMenu
6605A8BE	FF75 FC	PUSH DWORD PTR SS:[EBP-4]	hParent
6605A8C1	FF75 E8	PUSH DWORD PTR SS:[EBP-18]	Height
6605A8C4	FF75 EC	PUSH DWORD PTR SS:[EBP-14]	Width
6605A8C7	FF75 F8	PUSH DWORD PTR SS:[EBP-8]	Y
6605A8CA	53	PUSH EBX	X
6605A8CB	FF75 08	PUSH DWORD PTR SS:[EBP+8]	Style
6605A8CE	FFB6 84000000	PUSH DWORD PTR DS:[ESI+84]	WindowName
6605A8D4	57	PUSH EDI	Class
6605A8D5	FF75 F4	PUSH DWORD PTR SS:[EBP-C]	ExtStyle
6605A8D8	FF15 E8140066	CALL DWORD PTR DS:[&USER32.CreateWindowExA]	CreateWindowExA
6605A8DE	8BF8	MOV EDI,EAX	
6605A8E0	8B46 28	MOV EAX,DWORD PTR DS:[ESI+28]	
6605A8E3	8B00	MOV EAX,DWORD PTR DS:[EAX]	
6605A8E5	F640 03 08	TEST BYTE PTR DS:[EAX+3],8	
6605A8E8	95C4 305FFFFF	JE MSVBVM60.66059C70	

断了下来,我们看看堆栈的情况。

0012FAB4	00040000	ExtStyle = WS_EX_APPWINDOW
0012FAB8	0000C28C	Class = C28C
0012FABC	00E70740	WindowName = "Crouz CrackMe-1 : Setup"
0012FAC0	02C80000	Style = WS_OVERLAPPED WS_CLIPCHILDREN WS_SYSMENU WS_CAPTION
0012FAC4	00000000	X = 0
0012FAC8	FFFFFFFC	Y = FFFFFFFC (-4.)
0012FACC	00000107	Width = 107 (263.)
0012FAD0	000000D6	Height = D6 (214.)
0012FAD4	0062067E	hParent = 0062067E ('CrouzCrackMe2',class='ThunderRT6Main')
0012FAD8	00000000	hMenu = NULL
0012FADC	66000000	hInst = 66000000
0012FAE0	00000000	lParam = NULL
0012FAE4	00E65888	

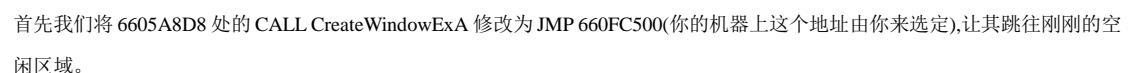
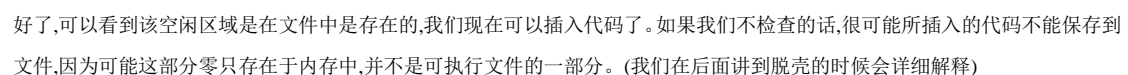
我们可以看到第一次断下来就是创建的 NAG 窗口,可能其他窗口的创建是其他地方调用的 CreateWindowExA。

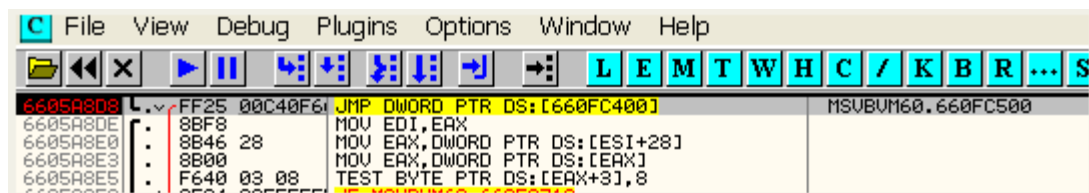
更加不幸的是,注册窗口的创建的也是调用的此处。

现在我们需要找到一块空闲的区域写入 JMP 地址,我们会发现代码段的最后有一块空闲的区域。

660FC4FF	00	DB 00
660FC500	00	DB 00
660FC501	00	DB 00
660FC502	00	DB 00
660FC503	00	DB 00
660FC504	00	DB 00
660FC505	00	DB 00
660FC506	00	DB 00
660FC507	00	DB 00
660FC508	00	DB 00
660FC509	00	DB 00
660FC50A	00	DB 00
660FC50B	00	DB 00
660FC50C	00	DB 00
660FC50D	00	DB 00
660FC50E	00	DB 00
660FC50F	00	DB 00
660FC510	00	DB 00
660FC511	00	DB 00
660FC512	00	DB 00
660FC513	00	DB 00
660FC514	00	DB 00
660FC515	00	DB 00
660FC516	00	DB 00
660FC517	00	DB 00
660FC518	00	DB 00
660FC519	00	DB 00
660FC51A	00	DB 00
660FC51B	00	DB 00
660FC51C	00	DB 00
660FC51D	00	DB 00

我们在这块区域上单击鼠标右键选择-View-Executable file。我们可以看到准备保存到 exe 中的区域。





这里我们构造一个间接跳转,这样能够确保跟之前的 CALL CreateWindowExA 指令一样长,这样就不至于修改到后面 MOV EDI,EAX 等指令,代码也就能正常的执行。如果覆盖了 MOV EDI,EAX 的任何一个字节的话,调用下面空白区域的代码返回接着执行,可能会出错。

我们将 660FC500 这个空闲区域的地址保存到 660FC400 内存单元中,这样 JMP [660FC400]跟 JMP 660FC500 就是等效的了。

Address	Hex dump	ASCII
660FC400	00 C5 0F 66 00 00 00 00	.+*f....
660FC408	00 00 00 00 00 00 00 00
660FC410	00 00 00 00 00 00 00 00
660FC418	00 00 00 00 00 00 00 00
660FC420	00 00 00 00 00 00 00 00
660FC428	00 00 00 00 00 00 00 00
660FC430	00 00 00 00 00 00 00 00
660FC438	00 00 00 00 00 00 00 00

我们在 660FC400 内存单元中写入 660FC500-我们将植入代码的首地址。

0012FAB4	00040000	
0012FAB8	0000C28C	
0012FABC	00E70740	ASCII "Crouz CrackMe-1 : Setup"
0012FAC0	02C80000	
0012FAC4	00000000	
0012FAC8	FFFFFFFC	
0012FACC	00000107	
0012FAD0	00000006	
0012FAD4	00780666	
0012FAD8	00000000	
0012FADC	66000000	OFFSET MSUBUM60.#1374
0012FAE0	00000000	
0012FAE4	00E65888	
0012FAE8	00000000	
0012FAEC	00000000	

我们接着来看看堆栈中给 CreateWindowExA 传入的参数,如窗口名称。

我们必须确保第一次调用的时候存在这个名称,因为有时候调用,窗口名称是为空的,如果我们不做检查的话,就会出错。

0012FAB4	00040000	
0012FAB8	0000C28C	
0012FABC	00E70740	ASCII "Crouz CrackMe-1 : Setup"
0012FAC0	02C80000	
0012FAC4	00000000	
0012FAC8	FFFFFFFC	
0012FACC	00000107	
0012FAD0	00000006	
0012FAD4	00780666	

我们在堆栈地址上双击将显示方式切换为偏移形式。

\$ ==>	00040000	
\$+4	0000C28C	
\$+8	00E70740	ASCII "Crouz CrackMe-1 : Setup"
\$+C	02C80000	
\$+10	00000000	

这说明 ESP+8 处保存的是窗口名称的首地址,我们将其保存到 EAX 寄存器中。

660FC500	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]	
660FC504	85C0	TEST EAX,EAX	
660FC506	74 1A	JE SHORT MSUBUM60.660FC522	
660FC508	8138 43726F7	CMP DWORD PTR DS:[EAX],756F7243	
660FC50E	75 12	JNZ SHORT MSUBUM60.660FC522	
660FC510	90	NOP	
660FC511	90	NOP	
660FC512	90	NOP	
660FC513	90	NOP	
660FC514	90	NOP	
660FC515	90	NOP	
660FC516	90	NOP	
660FC517	90	NOP	
660FC518	90	NOP	
660FC519	90	NOP	
660FC51A	C74424 0C 00	MOV DWORD PTR SS:[ESP+C],40000000	
660FC522	FF15 E814006	CALL DWORD PTR DS:[<&USER32.CreateWindow	USER32.CreateWindowExA
660FC528	E9 B1E3F5FF	JMP MSUBUM60.6605A8DE	
660FC52D	0000	ADD BYTE PTR DS:[EAX],AL	
660FC52F	0000	ADD BYTE PTR DS:[EAX],AL	
660FC531	0000	ADD BYTE PTR DS:[EAX],AL	
660FC533	0000	ADD BYTE PTR DS:[EAX],AL	

我们单步跟踪到 660FC500 处,看看具体的细节。我们按 F8 单步,可以看到窗口名称的首地址被保存到了 EAX 中。

Registers (FPU)	
EAX	00E70740 ASCII "Crouz CrackMe-1 : Setup"
ECX	6610DAC0 MSUBUM60.6610DAC0
EDX	00000007
EBX	00000000
ESP	0012FAB4
EBP	0012FBE0
ESI	00E704DC
EDI	0000C28C
EIP	660FC504 MSUBUM60.660FC504
C 0	ES 0023 32bit 0(FFFFFFFF)

现在 EAX 中保存了窗口名称的首地址,接着我们在下一行中检查 EAX 是否为零,如果为零说明该窗口可能没有标题名称。

660FC500	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]	
660FC504	85C0	TEST EAX,EAX	
660FC506	74 1A	JE SHORT MSUBUM60.660FC522	
660FC508	8138 43726F7	CMP DWORD PTR DS:[EAX],756F7243	
660FC50E	75 12	JNZ SHORT MSUBUM60.660FC522	
660FC510	90	NOP	
660FC511	90	NOP	
660FC512	90	NOP	
660FC513	90	NOP	
660FC514	90	NOP	
660FC515	90	NOP	
660FC516	90	NOP	
660FC517	90	NOP	
660FC518	90	NOP	
660FC519	90	NOP	
660FC51A	C74424 0C 00	MOV DWORD PTR SS:[ESP+C],40000000	
660FC522	FF15 E814006	CALL DWORD PTR DS:[<&USER32.CreateWindow	USER32.CreateWindowExA
660FC528	E9 B1E3F5FF	JMP MSUBUM60.6605A8DE	
660FC52D	0000	ADD BYTE PTR DS:[EAX],AL	
660FC52F	0000	ADD BYTE PTR DS:[EAX],AL	
660FC531	0000	ADD BYTE PTR DS:[EAX],AL	
660FC533	0000	ADD BYTE PTR DS:[EAX],AL	
660FC535	00	DB 00	
660FC536	00	DB 00	
660FC537	00	DB 00	
660FC538	00	DB 00	

这里如果 EAX 为零,就不需要修改窗口的样式了,继续调用 CreateWindowExA 即可。

如果 EAX 不为零,则需要做进一步的判断。

660FC500	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]	
660FC504	85C0	TEST EAX,EAX	
660FC506	74 1A	JE SHORT MSUBUM60.660FC522	
660FC508	8138 43726F7	CMP DWORD PTR DS:[EAX],756F7243	
660FC50E	75 12	JNZ SHORT MSUBUM60.660FC522	
660FC510	90	NOP	
660FC511	90	NOP	
660FC512	90	NOP	
660FC513	90	NOP	

这里我们判断窗口标题名称的前 4 个字节是否为 756F7243(也就是 NAG 窗口标题 Crouz CrackMe-1 : Setup 的前 4 个字节)。

Address	Hex dump	ASCII
00E70740	43 72 6F 75 7A 20 43 72	Crouz Cr
00E70748	61 63 68 4D 65 2D 31 20	ackMe-1
00E70750	3A 20 53 65 74 75 70 00	: Setup.
00E70758	02 00 04 00 CE 01 0A 00	0.0.0.0.
00E70760	46 6F 72 6D 31 00 E6 00	Form1.p.
00E70768	13 01 02 00 00 10 00 00	!!00.0.0.
00E70770	78 01 E6 00 78 01 E6 00	x0p.x0p.
00E70778	00 00 00 00 00 00 00 00
00E70780	00 00 00 00 00 00 00 00

这里如果判断出的确是 NAG 窗口,那么继续往下执行修改窗口样式,如果不是 NAG 窗口的话,就不修改窗口样式,直接调用 CALL CreateWindowExA。

660FC500	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]	
660FC504	85C0	TEST EAX,EAX	
660FC506	74 1A	JE SHORT MSUBUM60.660FC522	
660FC508	8138 43726F7	CMP DWORD PTR DS:[EAX],756F7243	
660FC50E	75 12	JNZ SHORT MSUBUM60.660FC522	
660FC510	90	NOP	
660FC511	90	NOP	
660FC512	90	NOP	
660FC513	90	NOP	
660FC514	90	NOP	
660FC515	90	NOP	
660FC516	90	NOP	
660FC517	90	NOP	
660FC518	90	NOP	
660FC519	90	NOP	
660FC51A	C74424 0C 00	MOV DWORD PTR SS:[ESP+C],40000000	
660FC522	FF15 E814000	CALL DWORD PTR DS:[&USER32.CreateWindowExA]	USER32.CreateWindowExA
660FC528	E9 B1E3F5FF	JMP MSUBUM60.6605A8DE	
660FC52D	0000	ADD BYTE PTR DS:[EAX],AL	
660FC52F	0000	ADD BYTE PTR DS:[EAX],AL	
660FC531	0000	ADD BYTE PTR DS:[EAX],AL	
660FC533	0000	ADD BYTE PTR DS:[EAX],AL	

这里我们可以看到如果不是 NAG 窗口的标题的话,就直接跳转到 CALL CreateWindowExA 处,并不修改窗口样式。如果是 NAG 窗口的话,就将 ESP+C 指向的窗口样式参数值修改为 40000000(子窗口样式)。

\$ =>	00040000	
\$+4	0000C28C	
\$+8	00E70740	ASCII "Crouz CrackMe-1 : Setup"
\$+C	02C80000	
\$+10	00000000	
\$+14	FFFFFFFC	
\$+18	00000107	
\$+1C	000000D6	
\$+20	00780666	
\$+24	00000000	
\$+28	66000000	OFFSET MSUBUM60.#1374
\$+2C	00000000	
\$+30	00E65888	
\$+34	00000000	

当我们执行到这一行

660FC500	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]	
660FC504	85C0	TEST EAX,EAX	
660FC506	74 1A	JE SHORT MSUBUM60.660FC522	
660FC508	8138 43726F7	CMP DWORD PTR DS:[EAX],756F7243	
660FC50E	75 12	JNZ SHORT MSUBUM60.660FC522	
660FC510	90	NOP	
660FC511	90	NOP	
660FC512	90	NOP	
660FC513	90	NOP	
660FC514	90	NOP	
660FC515	90	NOP	
660FC516	90	NOP	
660FC517	90	NOP	
660FC518	90	NOP	
660FC519	90	NOP	
660FC51A	C74424 0C 00	MOV DWORD PTR SS:[ESP+C],40000000	
660FC522	FF15 E8140066	CALL DWORD PTR DS:[<&USER32.CreateWindowExA]	USER32.CreateWindowExA
660FC528	E9 B1E3F5FF	JMP MSUBUM60.6605A8DE	
660FC52D	0000	ADD BYTE PTR DS:[EAX],AL	
660FC52F	0000	ADD BYTE PTR DS:[EAX],AL	
660FC531	0000	ADD BYTE PTR DS:[EAX],AL	
660FC533	0000	ADD BYTE PTR DS:[EAX],AL	
660FC535	00	DB 00	
660FC536	00	DB 00	
660FC537	00	DB 00	

可以看到调用是 CreateWindowExA,我们来看看参数情况:

\$ ==>	00040000	ExtStyle = WS_EX_APPWINDOW
\$+4	0000C28C	Class = C28C
\$+8	00E70740	WindowName = "Crowz CrackMe-1 : Setup"
\$+C	40000000	Style = WS_CHILD
\$+10	00000000	X = 0
\$+14	FFFFFFFC	Y = FFFFFFFC (-4.)
\$+18	00000107	Width = 107 (263.)
\$+1C	000000D6	Height = D6 (214.)
\$+20	00780666	hParent = 00780666 ('CrowzCrackMe2',class='ThunderRT6Main')
\$+24	00000000	hMenu = NULL
\$+28	66000000	hInst = 66000000
\$+2C	00000000	lParam = NULL
\$+30	00E65888	
\$+34	00000000	

我们可以看到 ESP + C 指向的窗口样式参数值被修改为了 40000000 即 WS_CHILD。

接下来我们调用 CreateWindowExA 了,我们需要知道 CreateWindowExA 的入口地址,好的,那么我们来看看原先的调用是怎样的。

6605A805	FF75 F4	PUSH DWORD PTR SS:[EBP-C]	
6605A808	FF15 E8140066	CALL DWORD PTR DS:[<&USER32.CreateWindowExA>]	USER32.CreateWindowExA
6605A80E	8BF8	MOV EDI,EAX	
6605A810	8B46 28	MOV EAX,DWORD PTR DS:[ESI+28]	
6605A813	8B00	MOV EAX,DWORD PTR DS:[EAX]	
6605A815	F640 03 08	TEST BYTE PTR DS:[EAX+3],8	
6605A819	0F84 29FEFFFF	JE MSUBUM60.6605A718	
6605A81F	56	PUSH ESI	
6605A820	E8 BE4EFFFF	CALL MSUBUM60.6604F7B3	
6605A825	66 3D 0300	CMP AX,3	
6605A829	0F85 19FEFFFF	JNZ MSUBUM60.6605A718	
6605A82F	8D45 94	LEA EAX,DWORD PTR SS:[EBP-6C]	
6605A831	50	PUSH EAX	
6605A833	57	PUSH EDI	
6605A835	FF15 14140066	CALL DWORD PTR DS:[<&USER32.GetWindowRect>]	USER32.GetWindowRect
6605A838	8D4C 94	LEA EAX,DWORD PTR SS:[EBP-6C]	
6605A83A	8D4E 58	LEA ECX,DWORD PTR DS:[ESI+58]	
6605A83C	50	PUSH EAX	
6605A83E	E8 5119FFFF	CALL MSUBUM60.6604C267	
6605A841	E9 FDFDFFFF	JMP MSUBUM60.6605A718	
6605A843	56	PUSH ESI	
6605A845	E8 5C4FFFFF	CALL MSUBUM60.6604F87D	
6605A848	FF75 F4	PUSH DWORD PTR SS:[EBP-C]	
6605A84A	56	PUSH ESI	
6605A84C	E8 8F52FFFF	CALL MSUBUM60.6604F8B9	
6605A84F	50	PUSH EAX	
6605A851	8D45 D0	LEA EAX,DWORD PTR SS:[EBP-30]	
6605A853	FF75 08	PUSH DWORD PTR SS:[EBP+8]	
6605A855	50	PUSH EAX	
6605A857	FF15 D4130066	CALL DWORD PTR DS:[<&USER32.AdjustWindowRectEx>]	USER32.AdjustWindowRectEx
6605A85A	E9 AFFEFFFF	JMP MSUBUM60.6605A7EC	
6605A85D	3B08	CMP EBX,EAX	
6605A85F	0F8E 0AFFFFFF	JLE MSUBUM60.6605A84F	
6605A861	3B08	MOV EBX,EAX	
6605A863	E9 03FFFFFF	JMP MSUBUM60.6605A84F	
DS:[660014E8]=77D2025E (USER32.CreateWindowExA)			

我们可以看到这是一个间接调用,它是读取 660014E8 内存单元中的值然后再调用的,实际上就是:

CALL [660014E8]

OD 的提示窗口中会提示 CreateWindowExA 的实际入口地址是多少,那么这里为什么要使用间接调用呢?是为了能够在任何一个机器上都能运行,我们在后面脱壳章节中介绍 IAT 的时候会更深入的解释。

660FC519	90	NOP	
660FC51A	C74424 0C 00	MOV DWORD PTR SS:[ESP+C1],40000000	
660FC522	FF15 E8140064	CALL DWORD PTR DS:[<&USER32.CreateWindow	USER32.CreateWindowExA
660FC528	E9 B1E3F5FF	JMP MSUBVM60.6605A8DE	
660FC52D	0000	ADD BYTE PTR DS:[EAX],AL	
660FC52F	0000	ADD BYTE PTR DS:[EAX],AL	
660FC531	0000	ADD BYTE PTR DS:[EAX],AL	
660FC533	0000	ADD BYTE PTR DS:[EAX],AL	
660FC535	00	DB 00	
660FC536	00	DB 00	

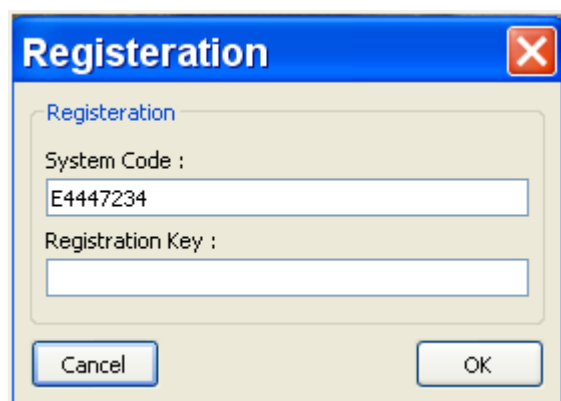
调用完 CreateWindowExA 以后我们已经返回到原来的下一行处继续执行 MOV EDI,EAX。

6605A8D9	57	PUSH EDI	
6605A8DE	FF75 F4	PUSH DWORD PTR SS:[EBP-C1]	
6605A8E3	FF25 00C40F64	JMP DWORD PTR DS:[660FC400]	MSUBVM60.660FC500
6605A8DE	8BF8	MOV EDI,EAX	
6605A8E0	8B46 28	MOV EAX,DWORD PTR DS:[ESI+28]	
6605A8E3	8B00	MOV EAX,DWORD PTR DS:[EAX]	
6605A8E5	F640 03 08	TEST BYTE PTR DS:[EAX+3],8	
6605A8E9	0F84 29FFFFFF	JE MSUBVM60.66050718	

接着我们运行起来。

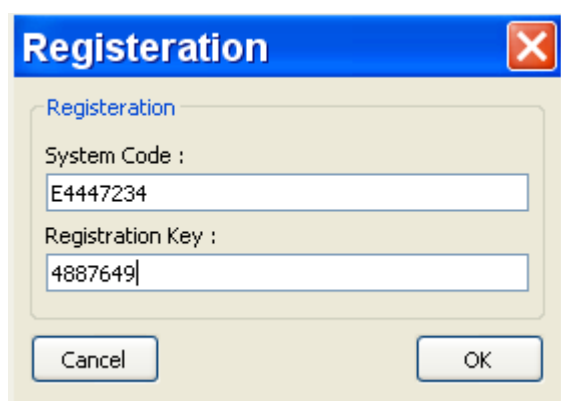
我们可以看到只弹出了注册窗口,没有出现 NAG 创建。总结一下我们需要的处理-检查窗口标题是否为 NAG 窗口,如果是就改变其窗口标题。

我们保存所有修改到文件直接运行起来看看效果。

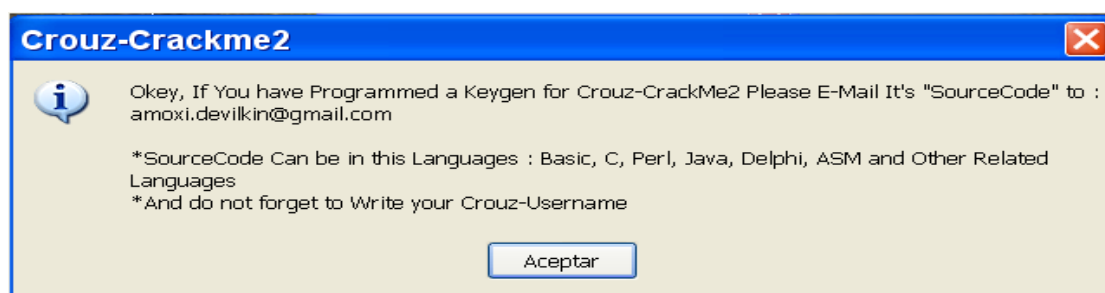


我们可以看到同样是直接弹出了注册窗口,没有出现 NAG 窗口,说明我们 Patch 的 CrackMe 和 MSVBVM60.DLL 成功剔除了 NAG 窗口。

接着我们来输入正确的序列号。



单击 OK。



我们可以看到提示序列号正确,说明我们成功搞定了这个 VB CrackMe。

下面你可以自己尝试剔除掉这个名为 CrackMe 的 NAG 窗口(必须的库 MSVBVM50.DLL)。