第二十八章-Visual Basic 程序的破解-Part3

破解 VB 程序的又一手法

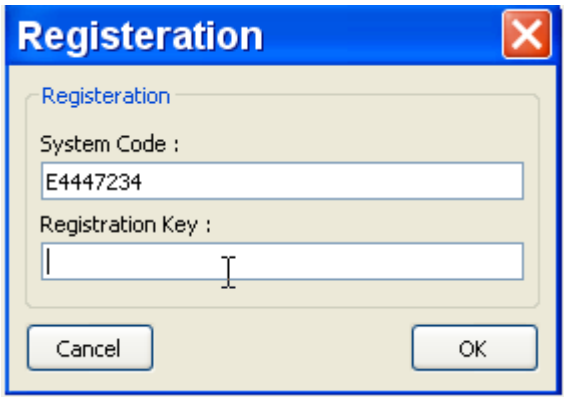
本章我们的实现对象是 CrackMe2。

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。

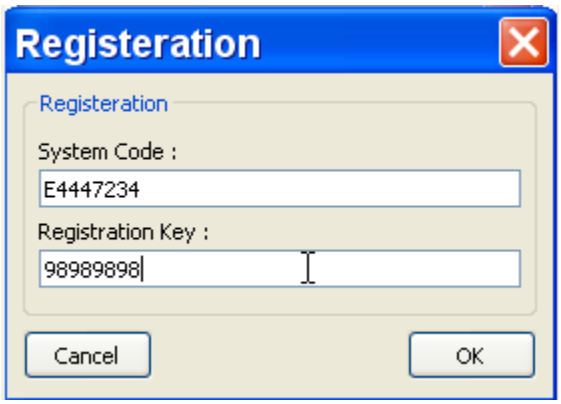
要找到这个 CrackMe 的序列号很简单,关键是如何剔除这个 NAG 窗口,通过 4C 法我们可以轻松的剔除掉 NAG 窗口,这里我就不再赘述了,大家可以自行尝试。这里我们来介绍另外一种方式。

首先我们来看看正确的序列号是多少。

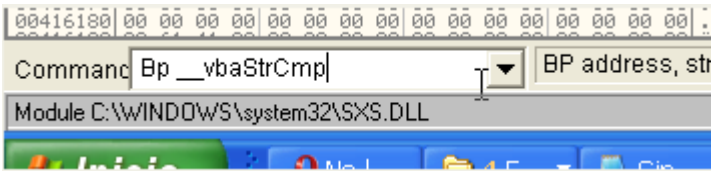
我们单击 Register 按钮。



出现了注册窗口,我们随便输入一个错误的序列号。



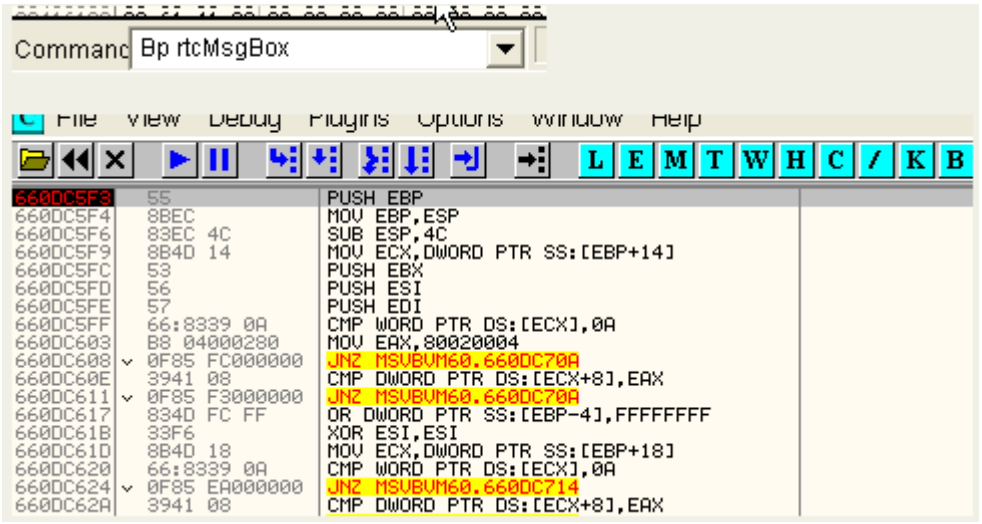
首先我们来看看是否是用__vbaStrCmp 这个函数来进行字符串比较的。



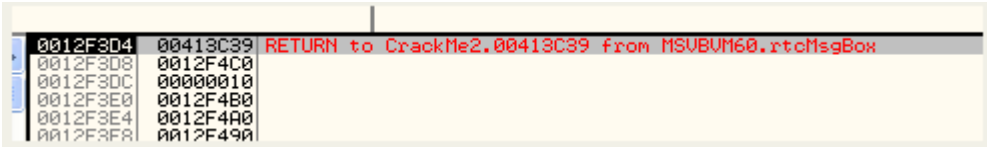
我们在命令栏中输入 bp __vbaStrCmp。

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。

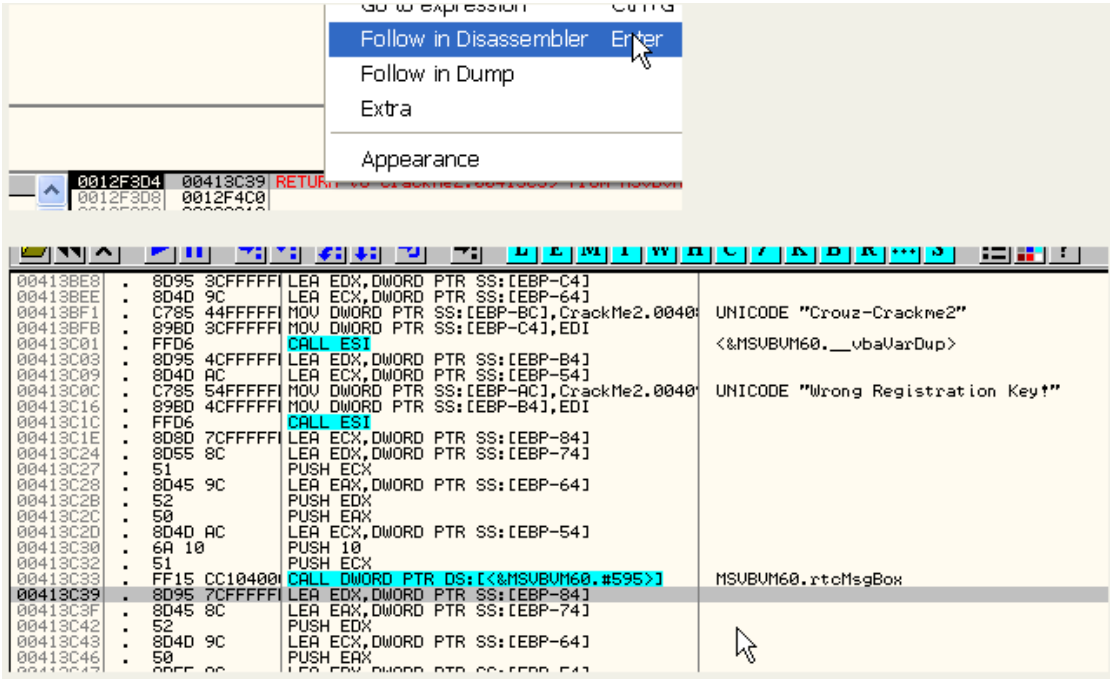
我们在命令栏中输入 bp rtcMsgBox。



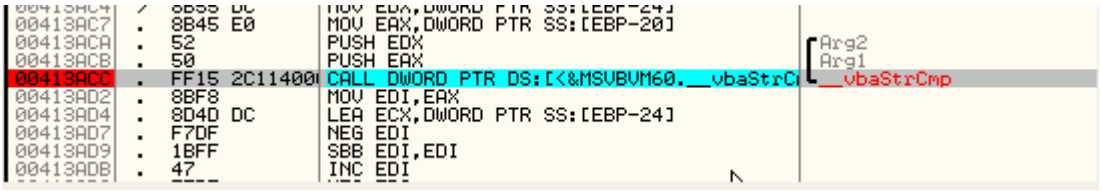
我们单击 OK 按钮。断在了 rtcMsgBox 的入口处。我猜是提示输入的序列号错误。我们来到堆栈窗口看看调用来至于哪里。



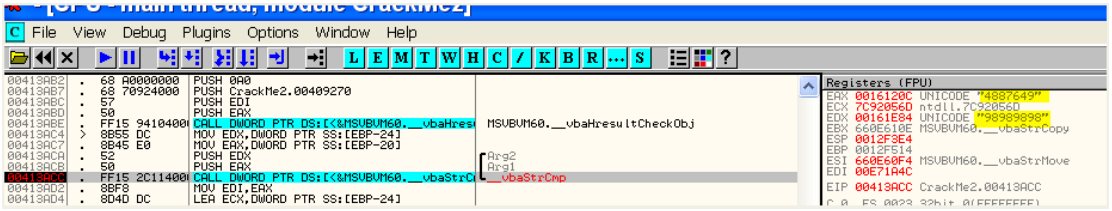
我们在反汇编窗口中定位到该返回地址处。



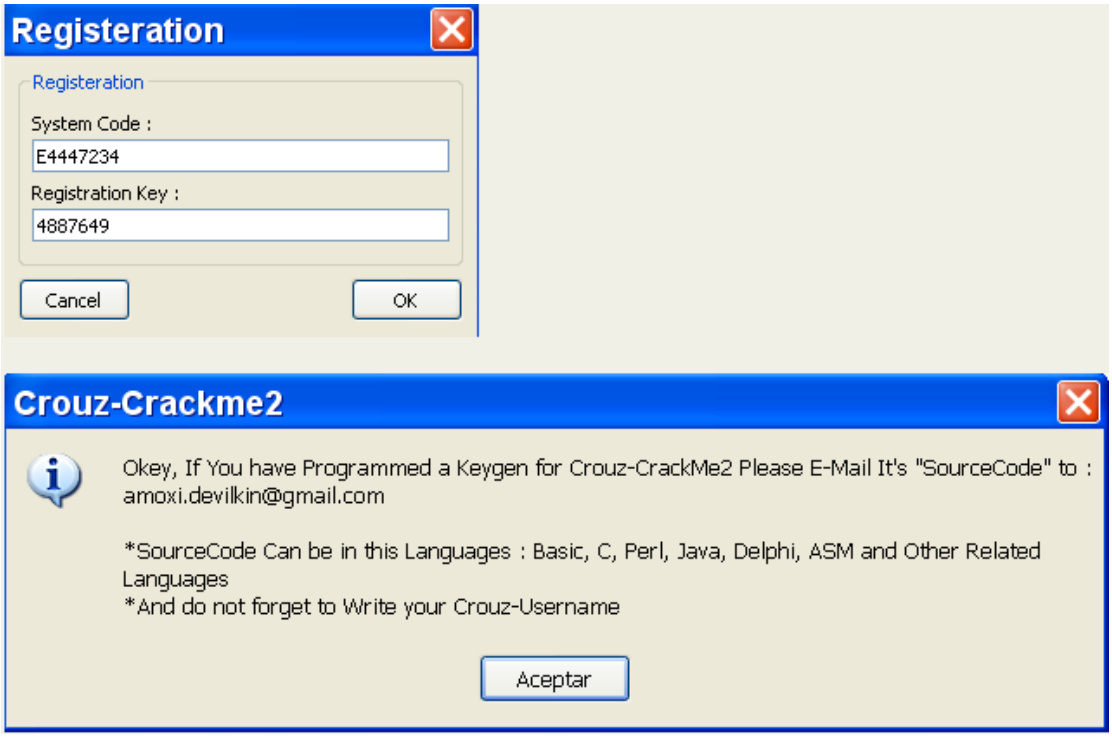
好了,现在我们来到返回地址处,我们看看前面几行有没有调用字符串比较之类的 API 函数。



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。

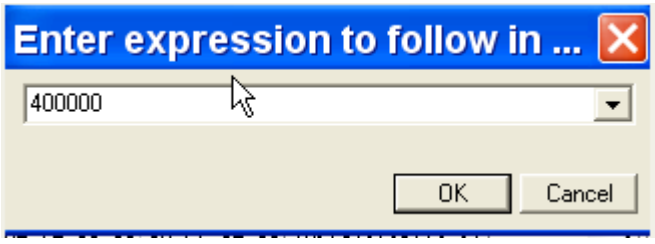


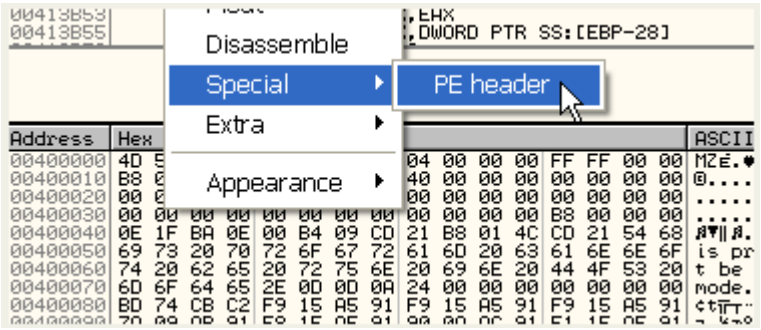
断了下来,我们可以看到正在进行字符串的比较,其中一个是我们输入的错误序列号。另一个我这里是 4887649,我们删除之前的断点,在注册窗口中输入这个字符串看看是否是正确的序列号。



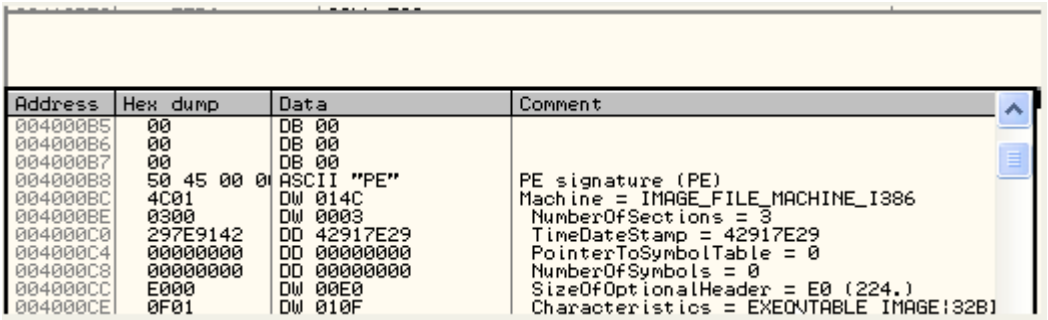
我们可以看到的确是正确的序列号。

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。

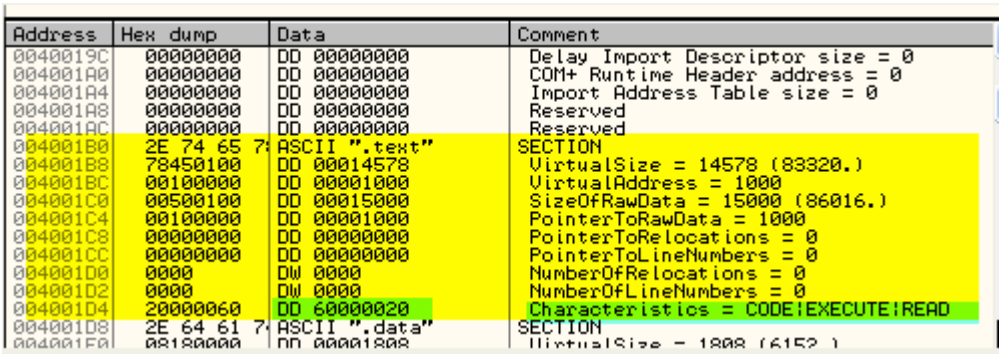




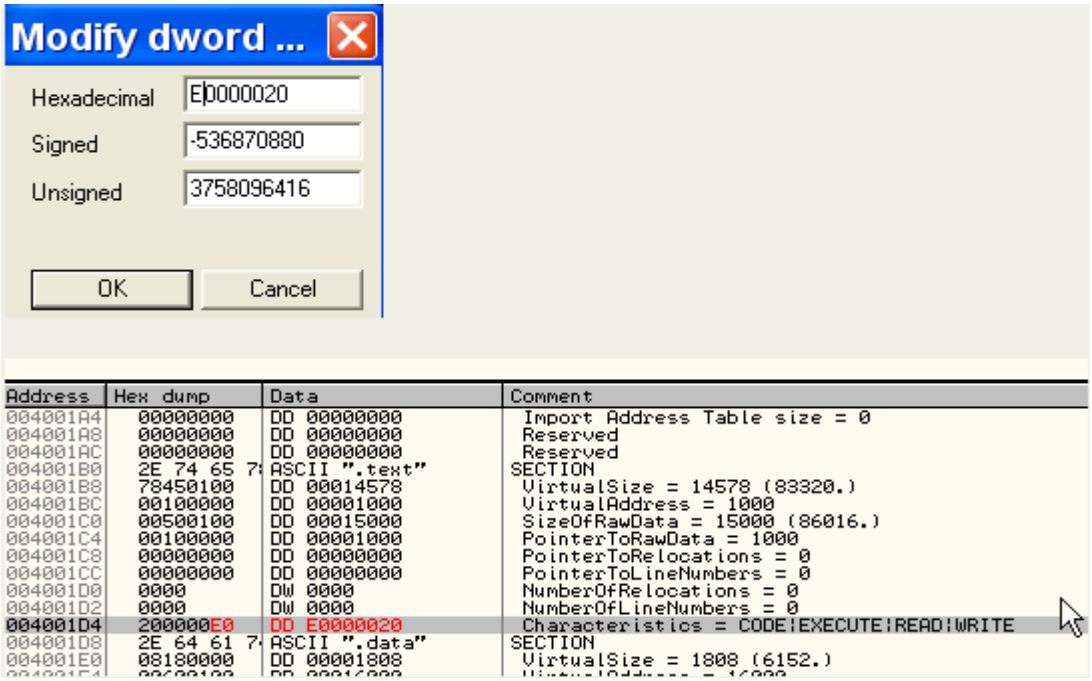
我们切换为 PE header 模式显示。



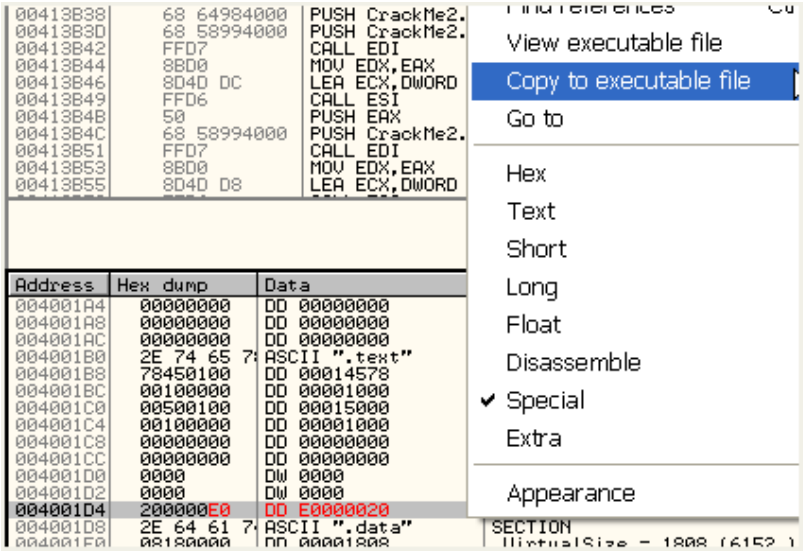
往下我们可以看到 PE Signature 标志,我们继续往下定位到第一个区段。

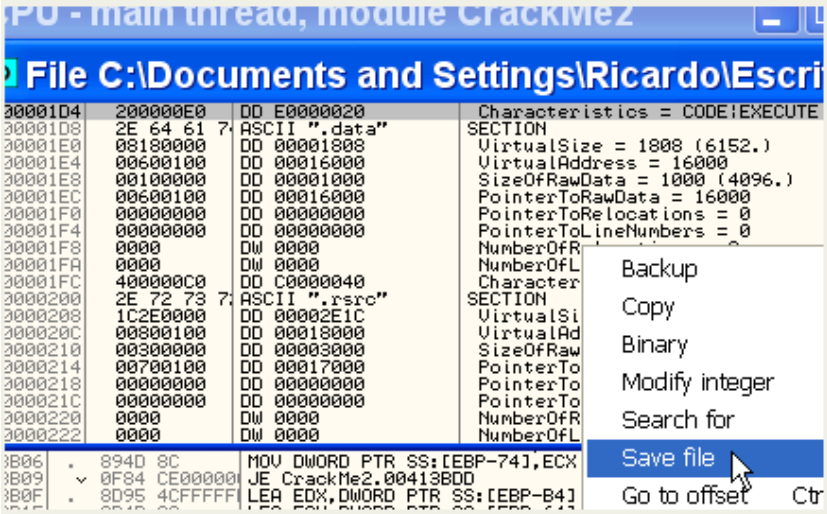


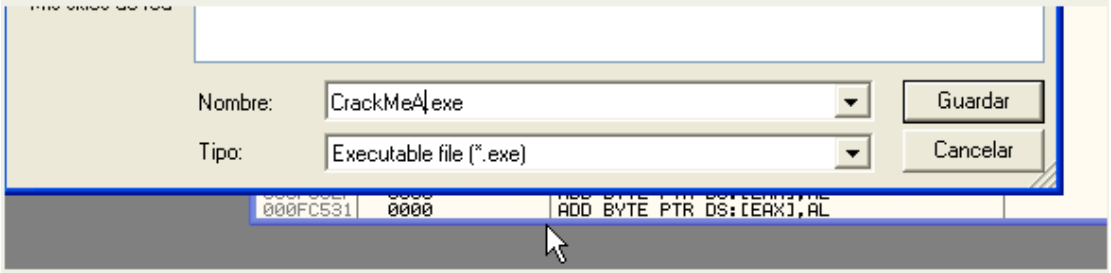
这里我们可以看到 Characteristics 字段,我们将为其修改为 E0000020 的话,就可以写入代码段了。



好了,现在我们将刚刚所做的修改保存到文件。



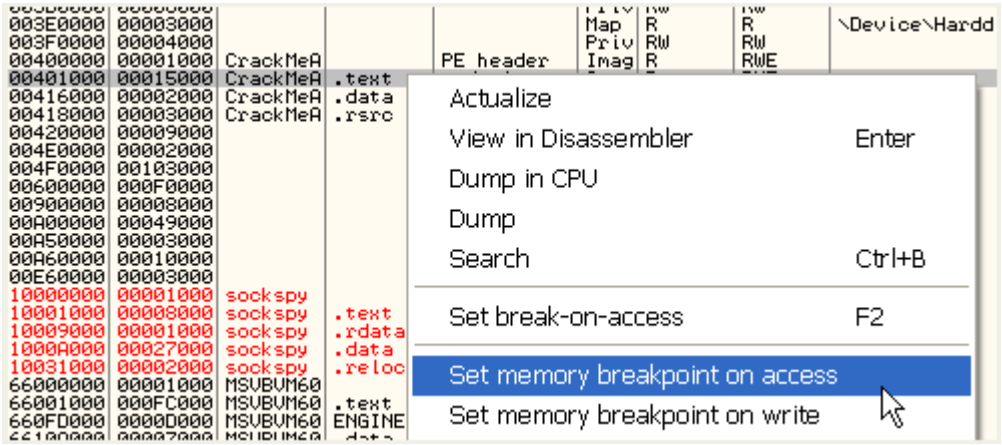




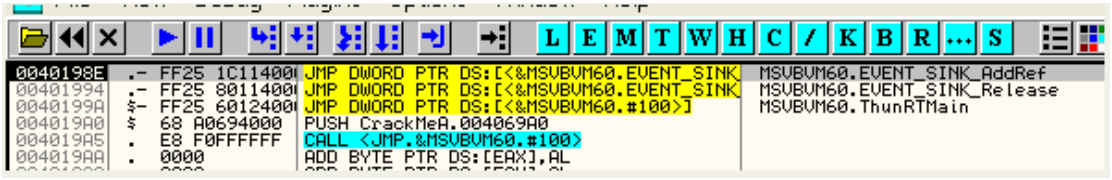
另存为 CrackMeA.exe

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

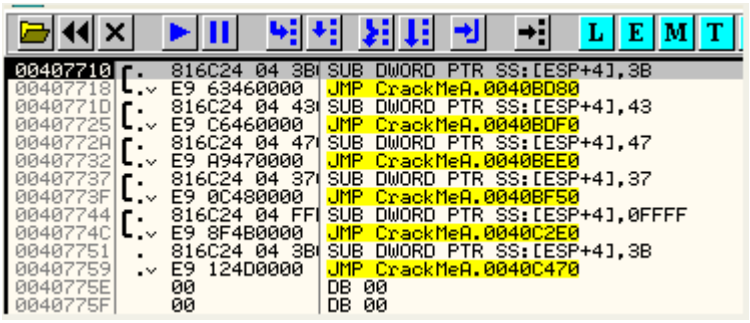
接着给代码段设置内存访问断点(实际上是内存执行断点)。



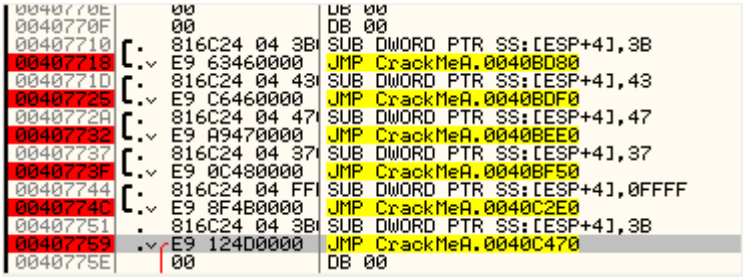
我们多按几次 F9 键运行起来就可以来到这里。



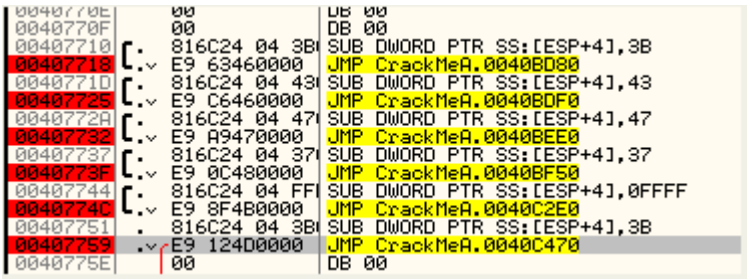
我们继续按 F9 键运行,停在了熟悉的多分支处,将会转向去执行程序的不同部分。



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。

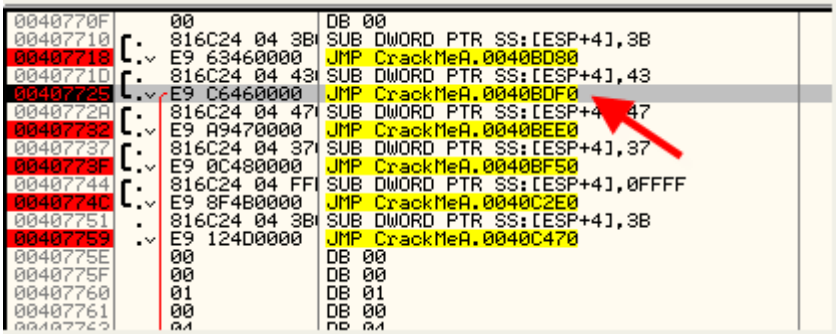


我们来看看断在其他 JMP 指令之前会不会弹出 NAG 窗口,运行起来。

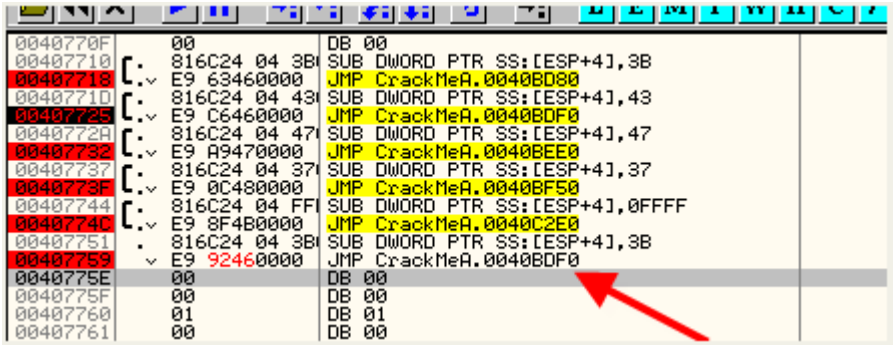


我们可以看到断在了 407759 处,将跳转到 40C470 处,并没有弹出 NAG 窗口,我们继续运行。

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。



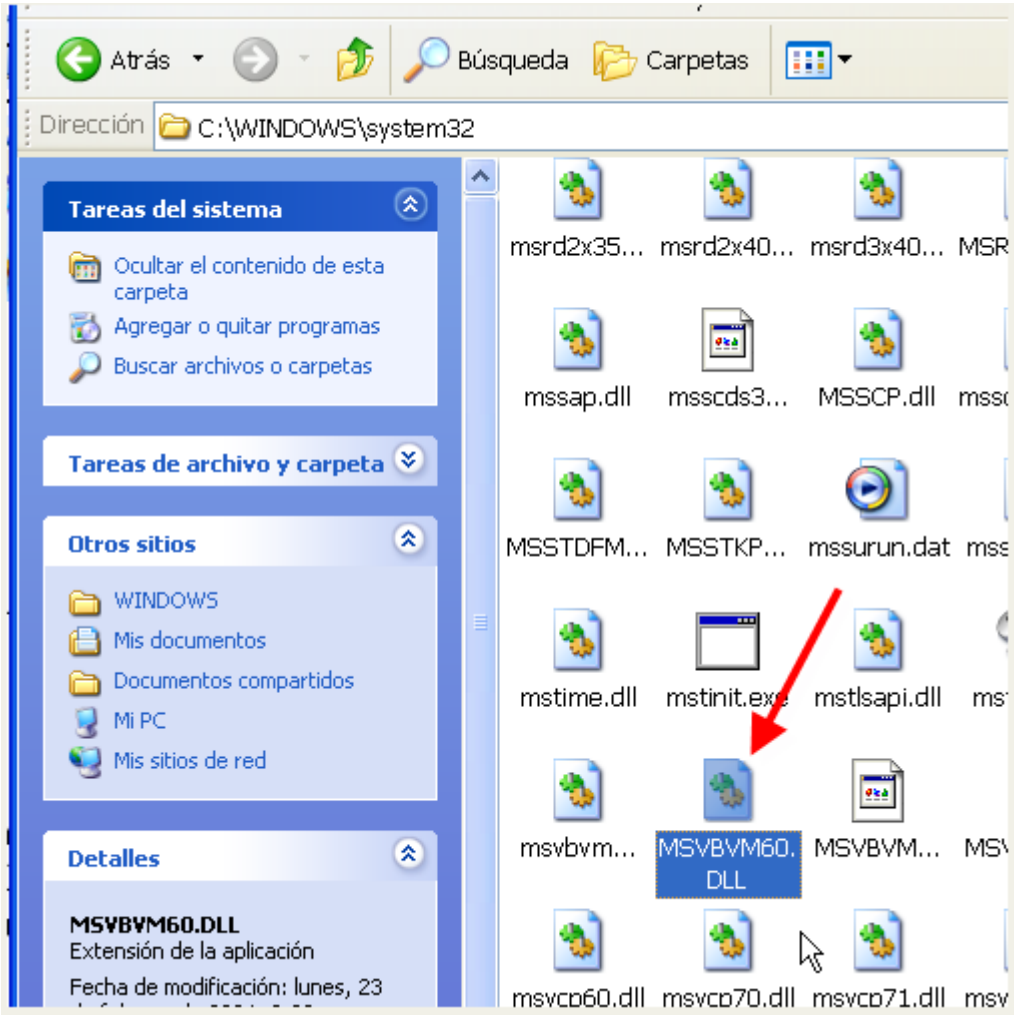
我们将 407759 处的 JMP 40C4F0 修改为 JMP 40BDF0 看看会发生什么。

我们保存到修改到文件,然后直接运行起来。

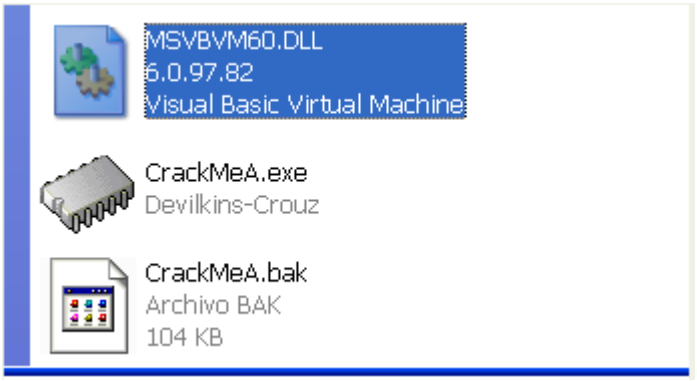
我们可以看到刚刚做的修改并没有剔除掉 NAG 窗口,但是原本是需要我们单击 NAG 窗口的 Register 按钮才会弹出注册窗口的,现在是 NAG 窗口和注册窗口一起弹了出来。

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。

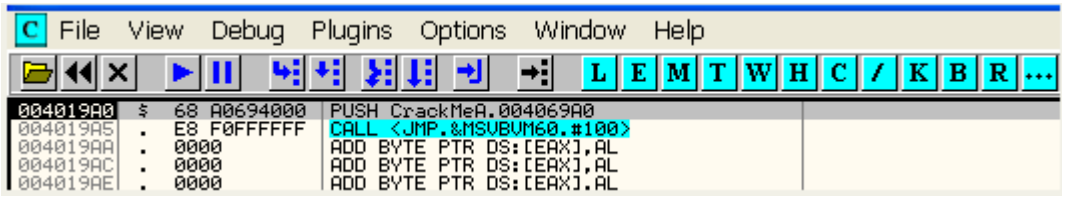


首先将 CrackMeA 做个备份。

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,如果当前目录下面有就直接加载,如果当期目录下面没

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。

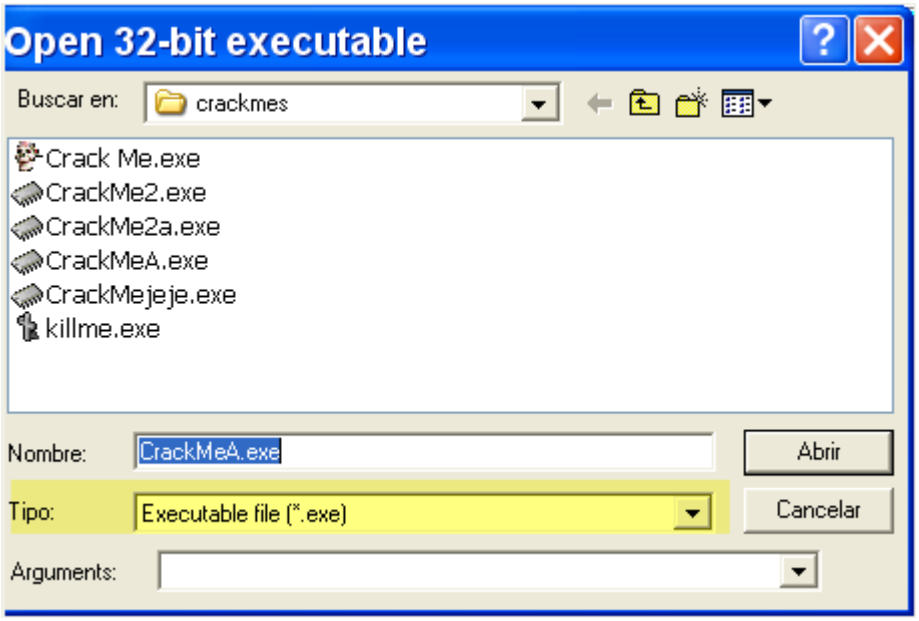


我们单击工具栏中 E 按钮打开模块列表窗口,看看是加载的当前目录下的 MSVBVM60.dll 还是 system32 下的。

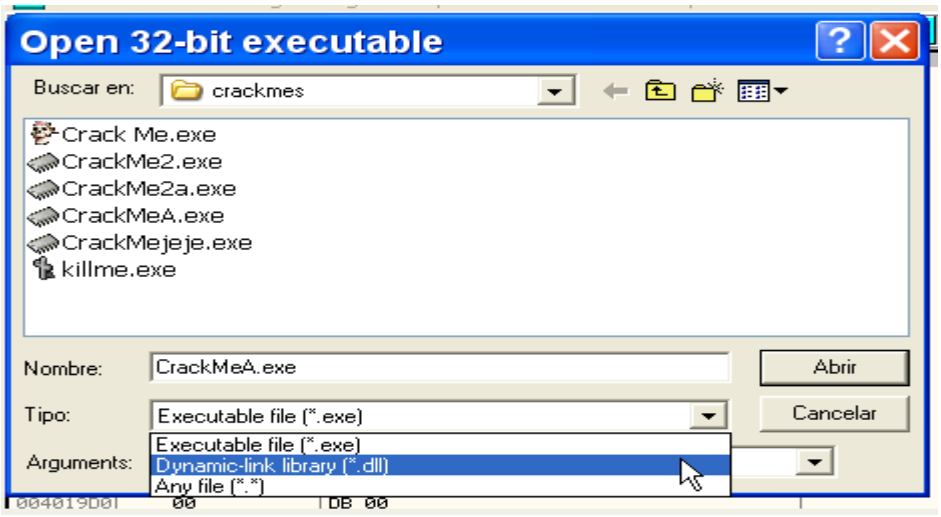
我们可以看到加载是当前目录下的 MSVBVM60.dll。

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:

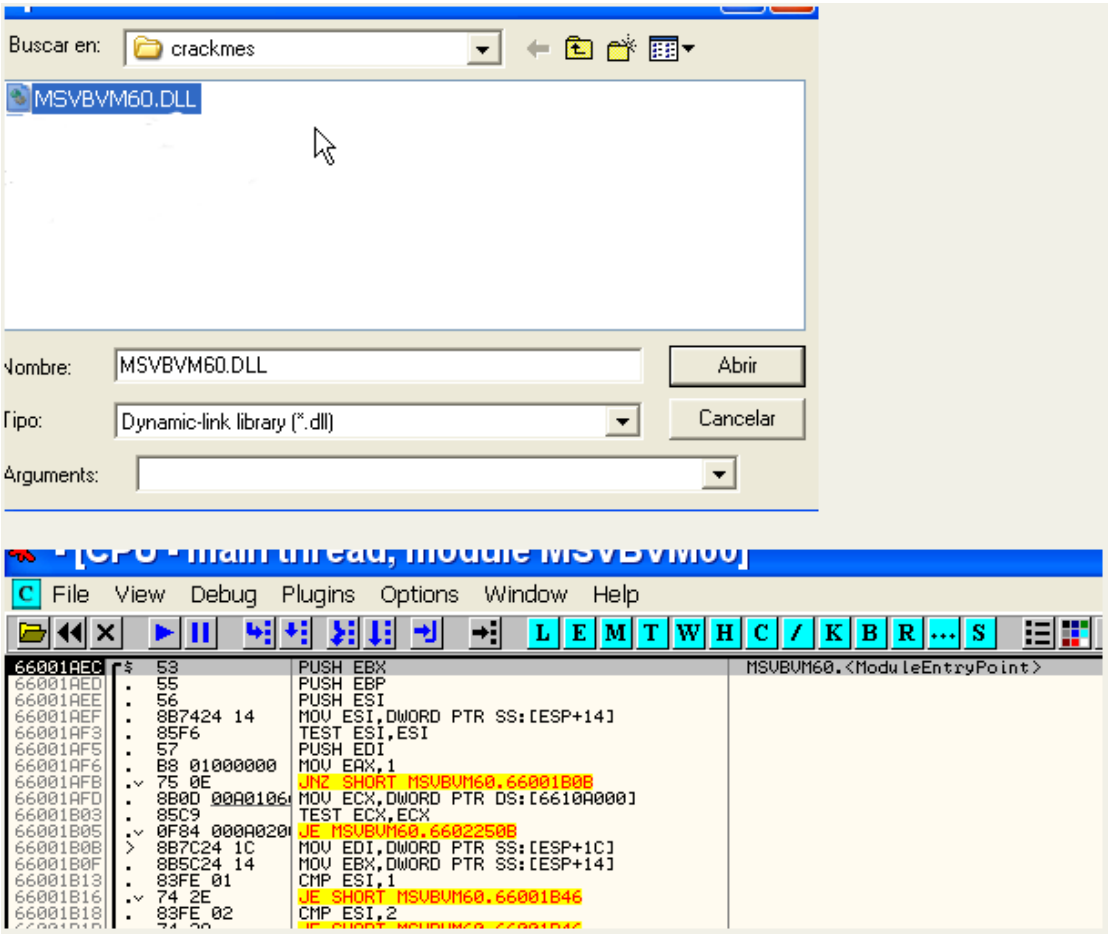
我们打开一个 OD 然后单击菜单栏中 Open 选项。



默认显示的 exe 文件,我们将文件类型改为 DLL。

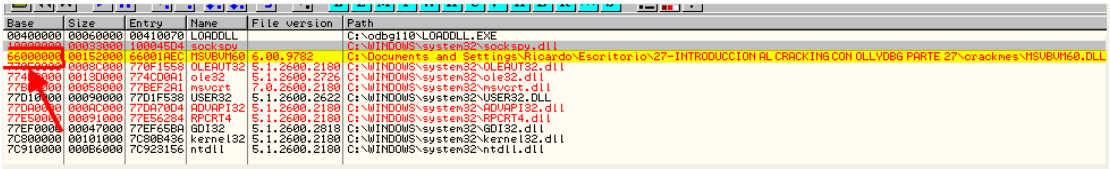


将下拉选项选中 Dynamic-Link library(*.dll)就能打开 DLL 文件了,我们打开 MSVBVM60.dll。



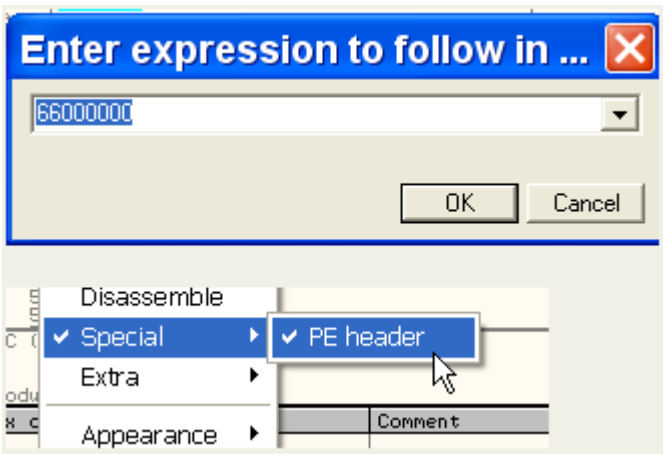
我们停在了该动态库的入口点处。

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。

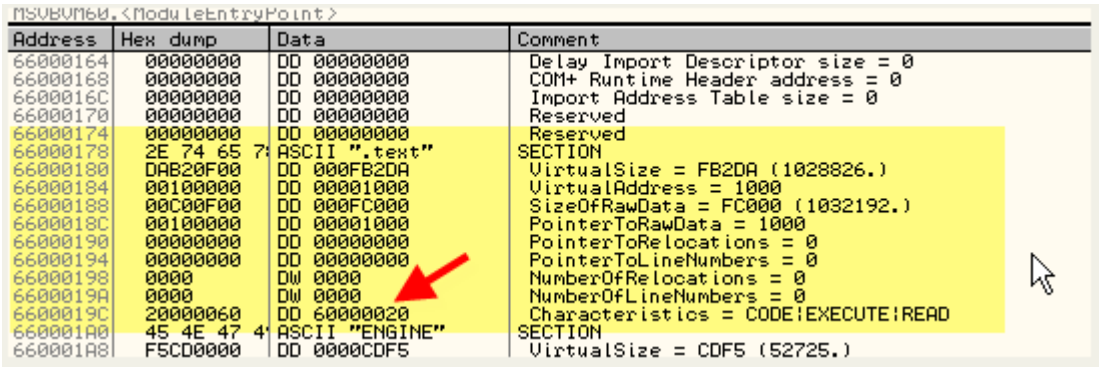


我这里 MSVBVM60.dll 的基地址是 66000000。可能跟你机器上的不一样。

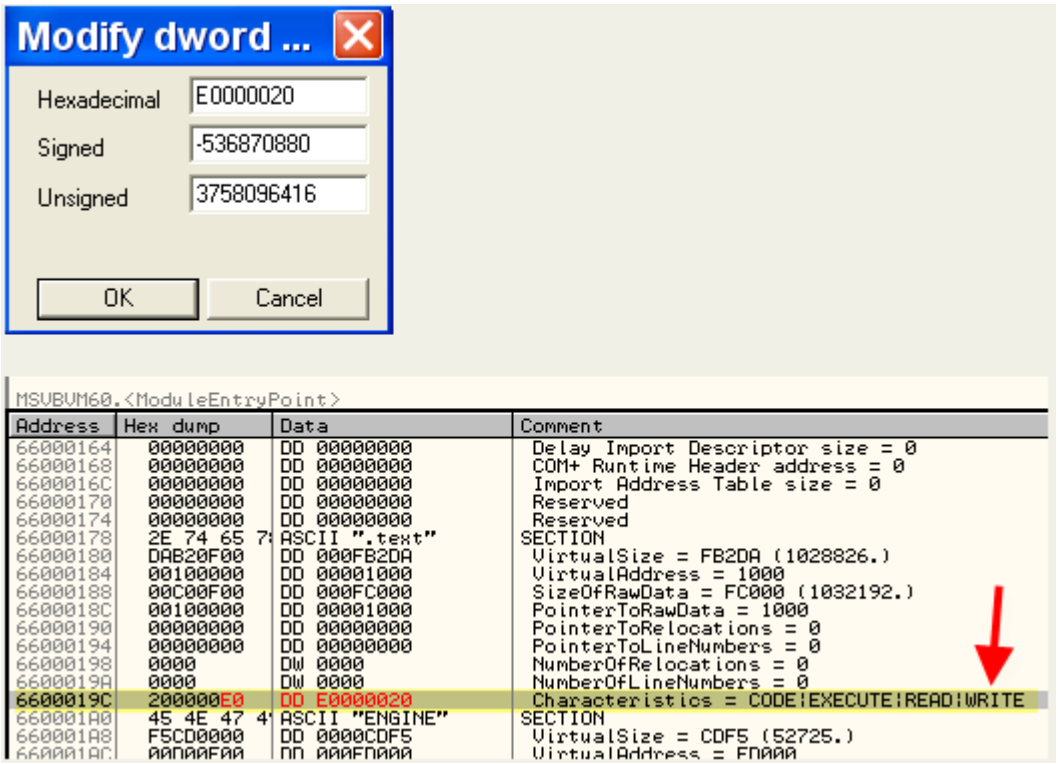
我们在数据窗口中定位到这个地址。



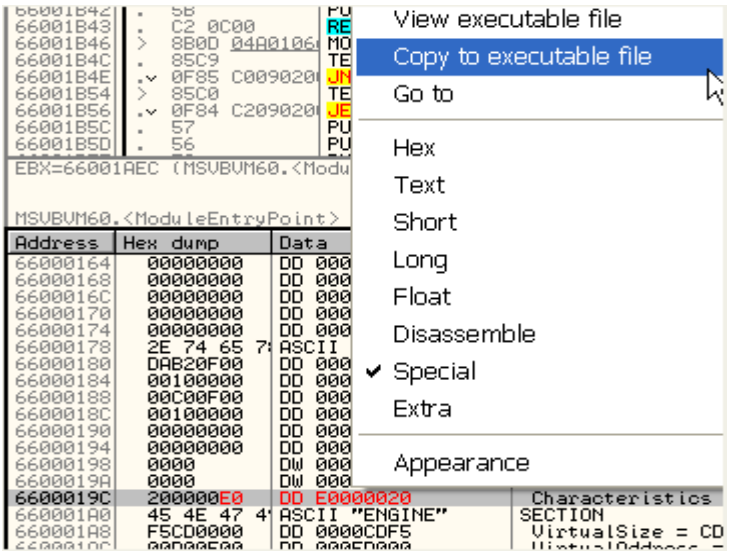
定位到头部以后我们切换到 PE header 模式显示,接着往下拉到.text 区段。

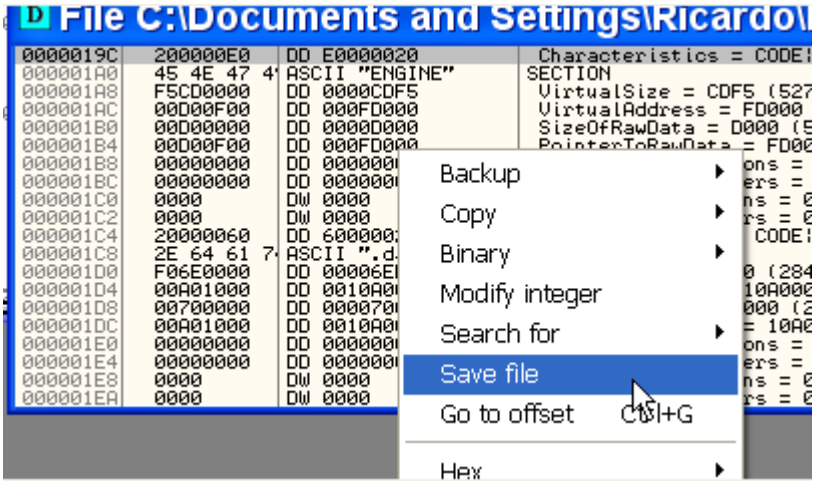


将 Characteristics 字段的值修改为 E0000020,让代码段具有写权限。



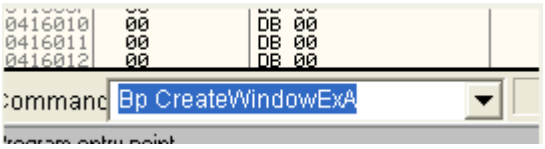
保存修改到文件。



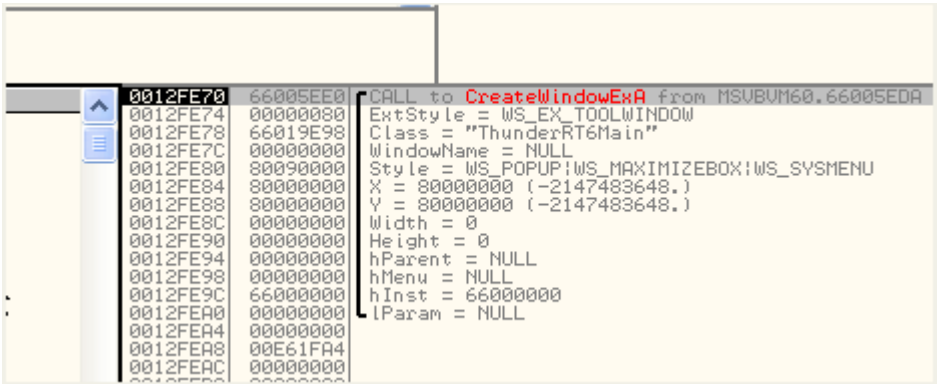


这里我们不重命名,因为重命名后 CrackMe 就不会加载了。

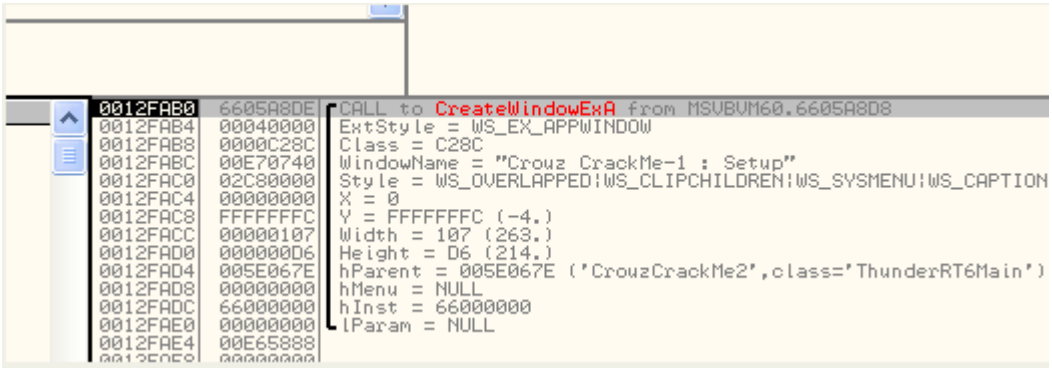
我们用 Patch 过的 OllyDbg 加载 CrackMeA,由于注册窗口和 NAG 窗口是一起弹出来的,所以我们给创建窗口的 API 函数 CreateWindowExA 设置一个断点。



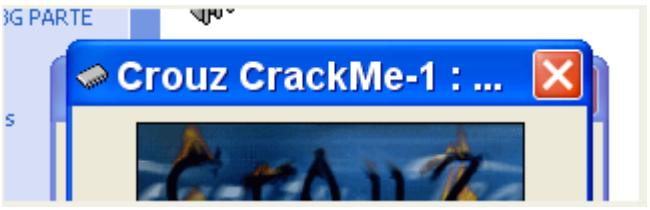
运行起来。



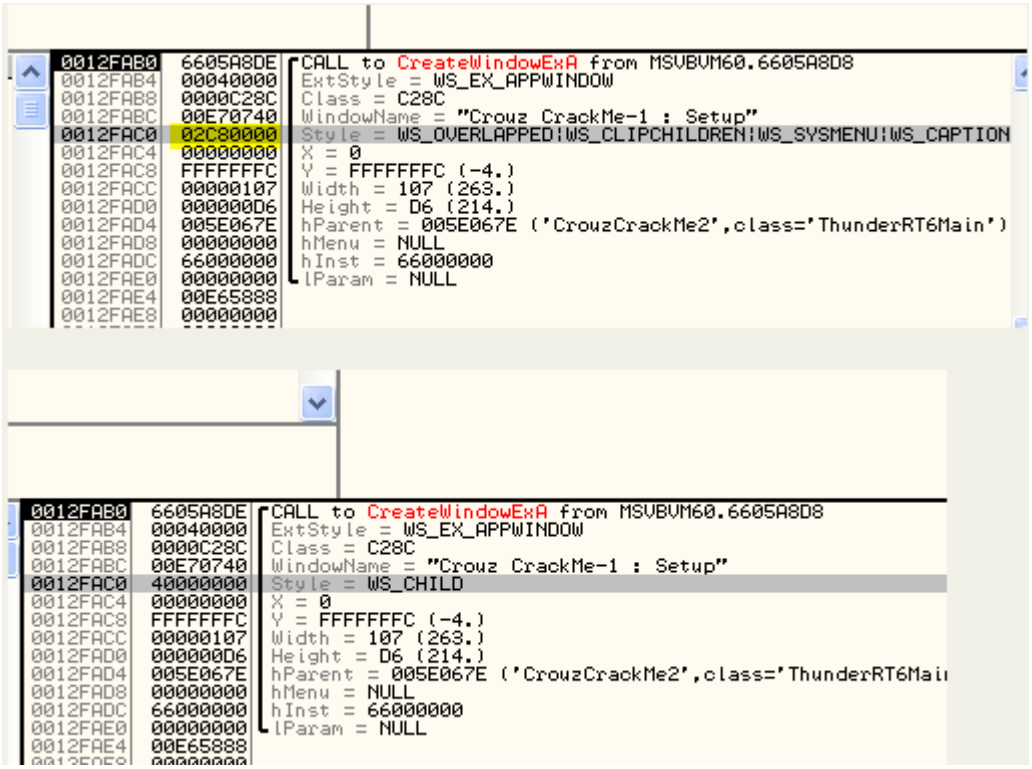
由于会创建子窗口所以会断下来几次,我多按 F9 键几次,直到创建 NAG 窗口为止。



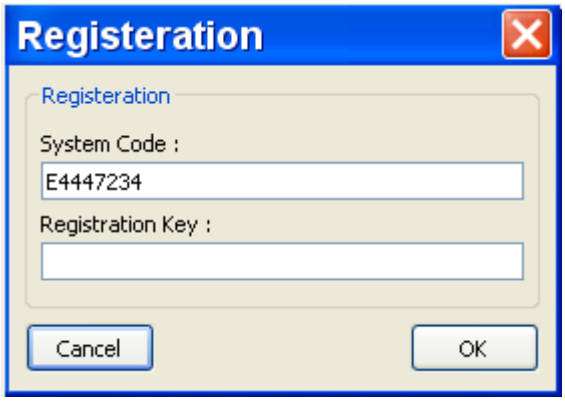
判断是否是创建 NAG 窗口很简单,根据窗口标题名就可以很容易的看出来。



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。

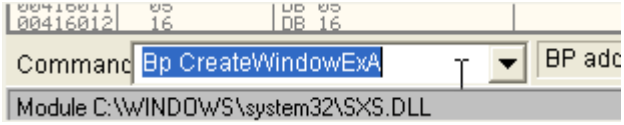


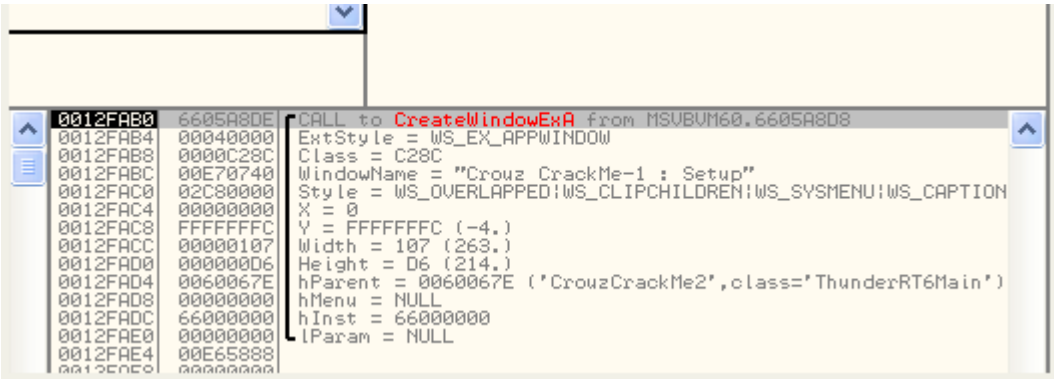
我们可以看到已经将窗口风格修改为 WS_CHILD,你可以尝试输入不同的值,现在我们可以删除所有断点运行起来。



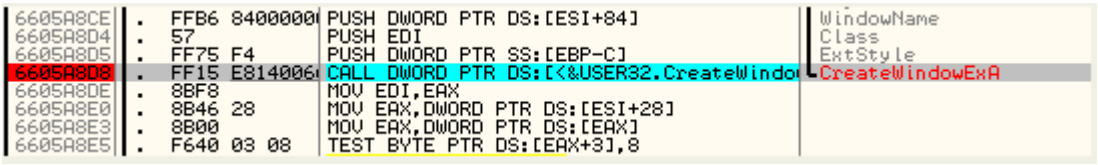
我们可以看到只弹出了一个注册窗口,并没有弹出 NAG 窗口。如果不修改 MSVBVM60.dll 想要剔除这个 NAG 窗口就很复杂了,我们手工从系统目录复制一个 MSVBVM60.dll 将其置于与目标程序同一目录,通过修改这个文件并不会影响到其他应用程序,嘿嘿。

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。

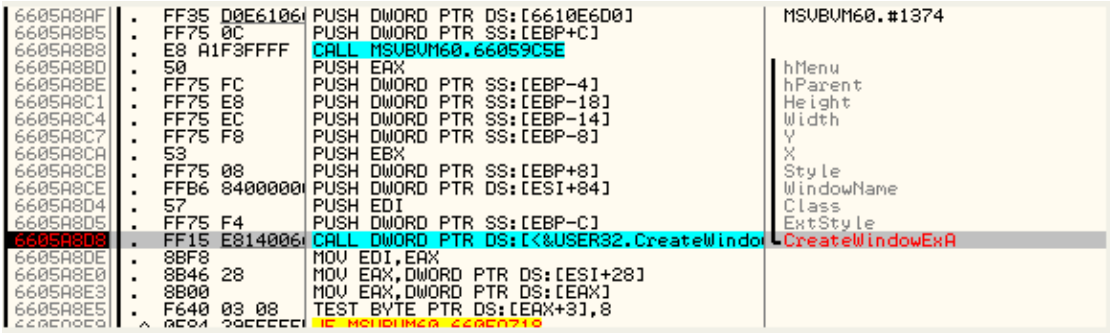




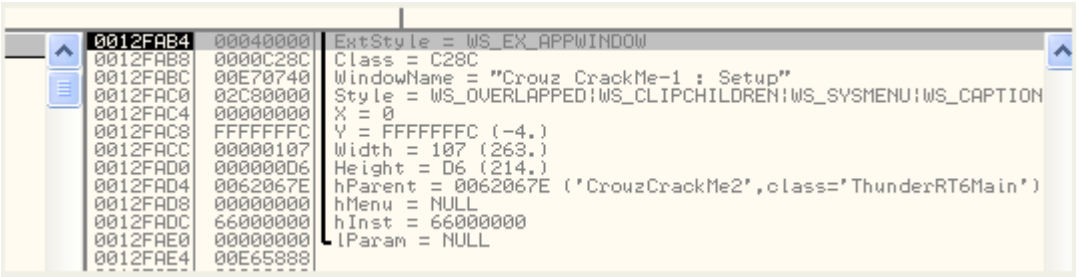
从堆栈中的信息我们可以看出调用来至于 MSVBVM60.DLL,我们定位到调用处。



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。



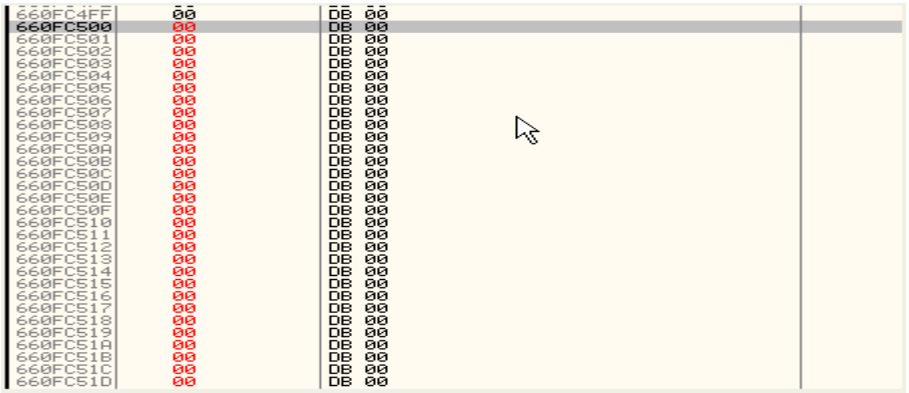
断了下来,我们看看堆栈的情况。



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。

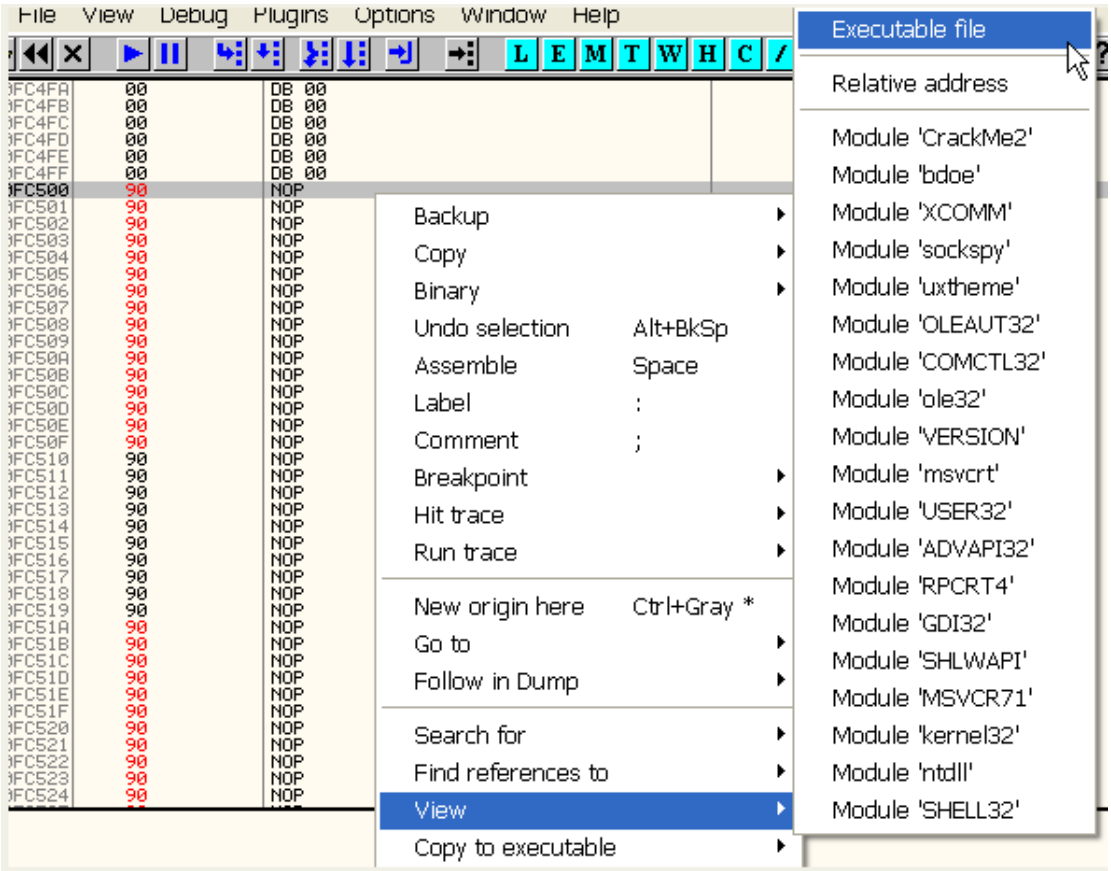
更加不幸的是,注册窗口的创建的也是调用的此处。

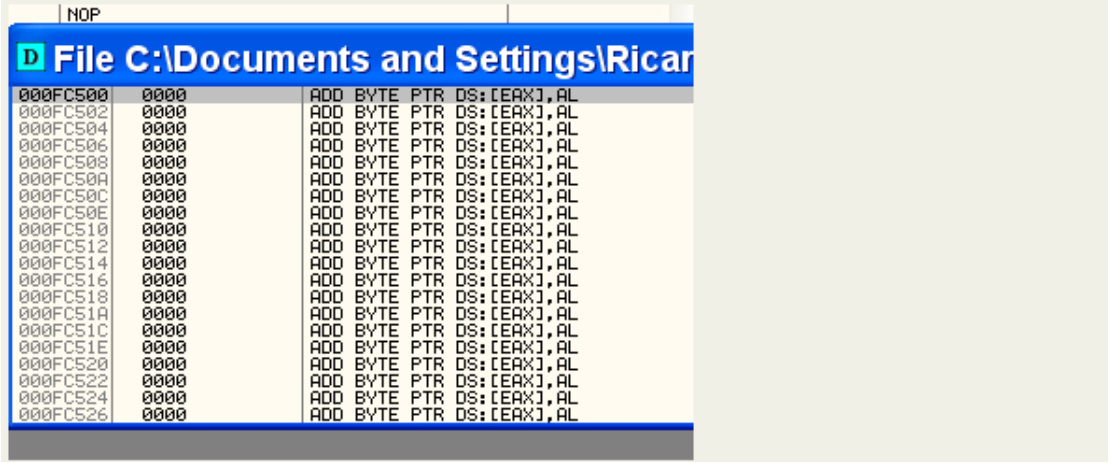
现在我们需要找到一块空闲的区域写入 JMP 地址,我们会发现代码段的最后有一块空闲的区域。



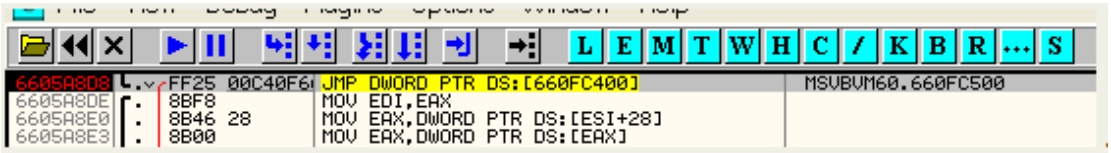
在我的机器上,这块空闲的区域在代码段的最后,我们可以看到很多零,首先我们来验证一下看看这块区域是否可用。

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。

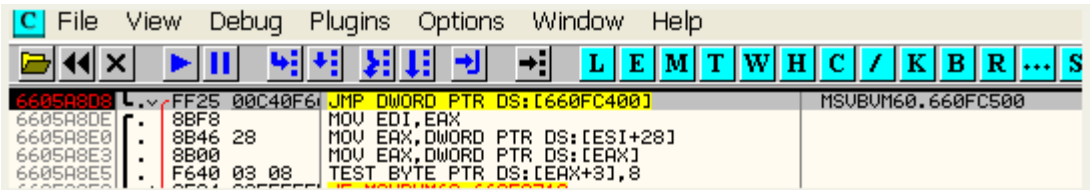




好了,可以看到该空闲区域是在文件中是存在的,我们现在可以插入代码了。如果我们不检查的话,很可能所插入的代码不能保存到文件,因为可能这部分零只存在于内存中,并不是可执行文件的一部分。(我们在后面讲到脱壳的时候会详细解释)

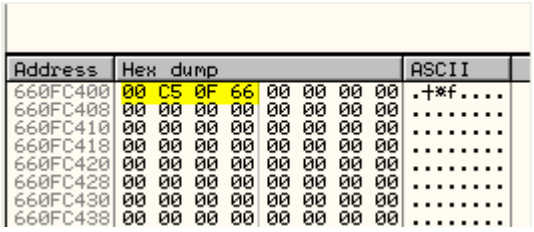


首先我们将 6605A8D8 处的 CALL CreateWindowExA 修改为 JMP 660FC500(你的机器上这个地址由你来选定),让其跳往刚刚的空闲区域。

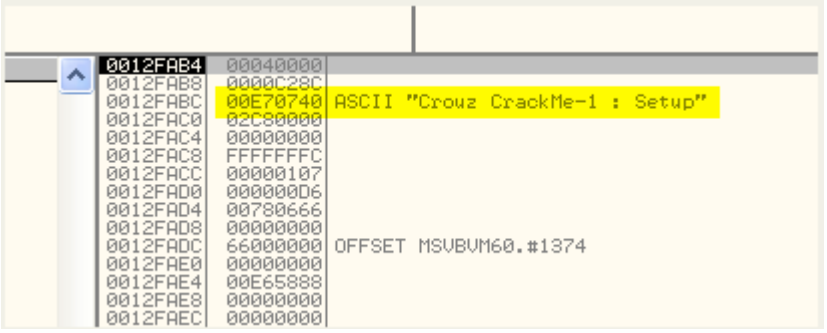


这里我们构造一个间接跳转,这样能够确保跟之前的 CALL CreateWindowExA 指令一样长,这样就不至于修改到后面 MOV EDI,EAX 等指令,代码也就能正常的执行。如果覆盖了 MOV EDI,EAX 的任何一个字节的话,调用下面空白区域的代码返回接着执行,可能会出错。

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。

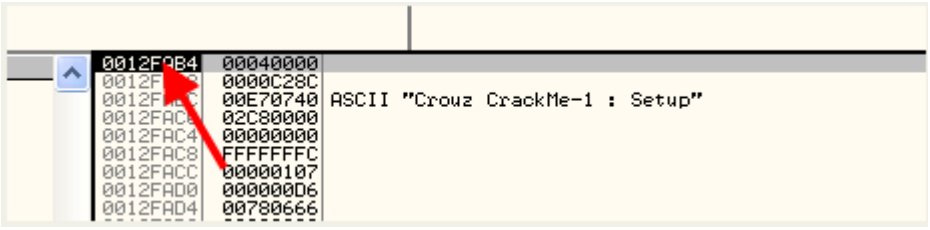


我们在 660FC400 内存单元中写入 660FC500-我们将植入代码的首地址。

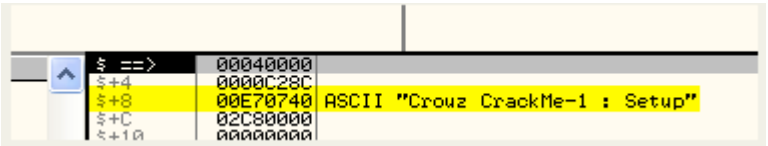


我们接着来看看堆栈中给 CreateWindowExA 传入的参数,如窗口名称。

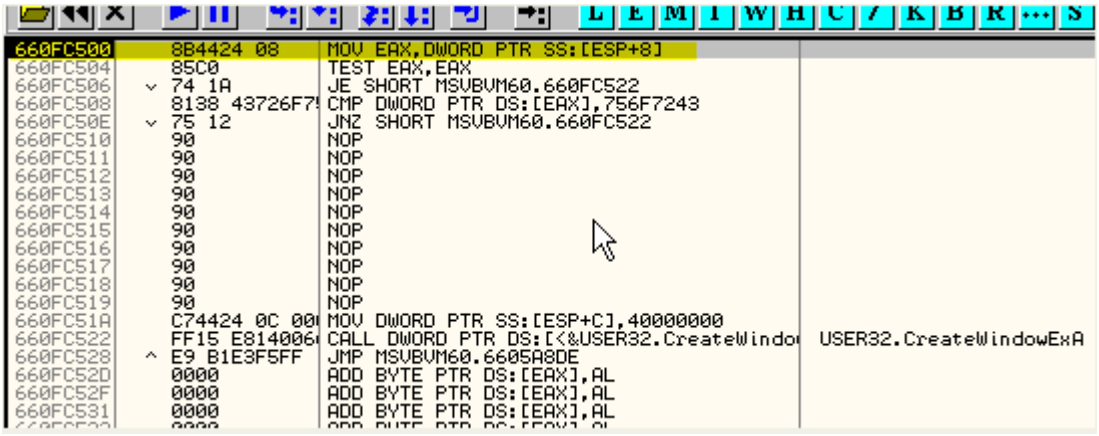
我们必须确保第一次调用的时候存在这个名称,因为有时候调用,窗口名称是为空的,如果我们不做检查的话,就会出错。



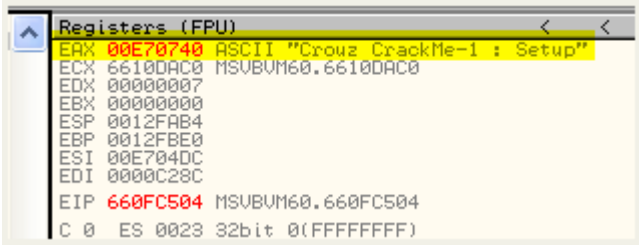
我们在堆栈地址上双击将显示方式切换为偏移形式。



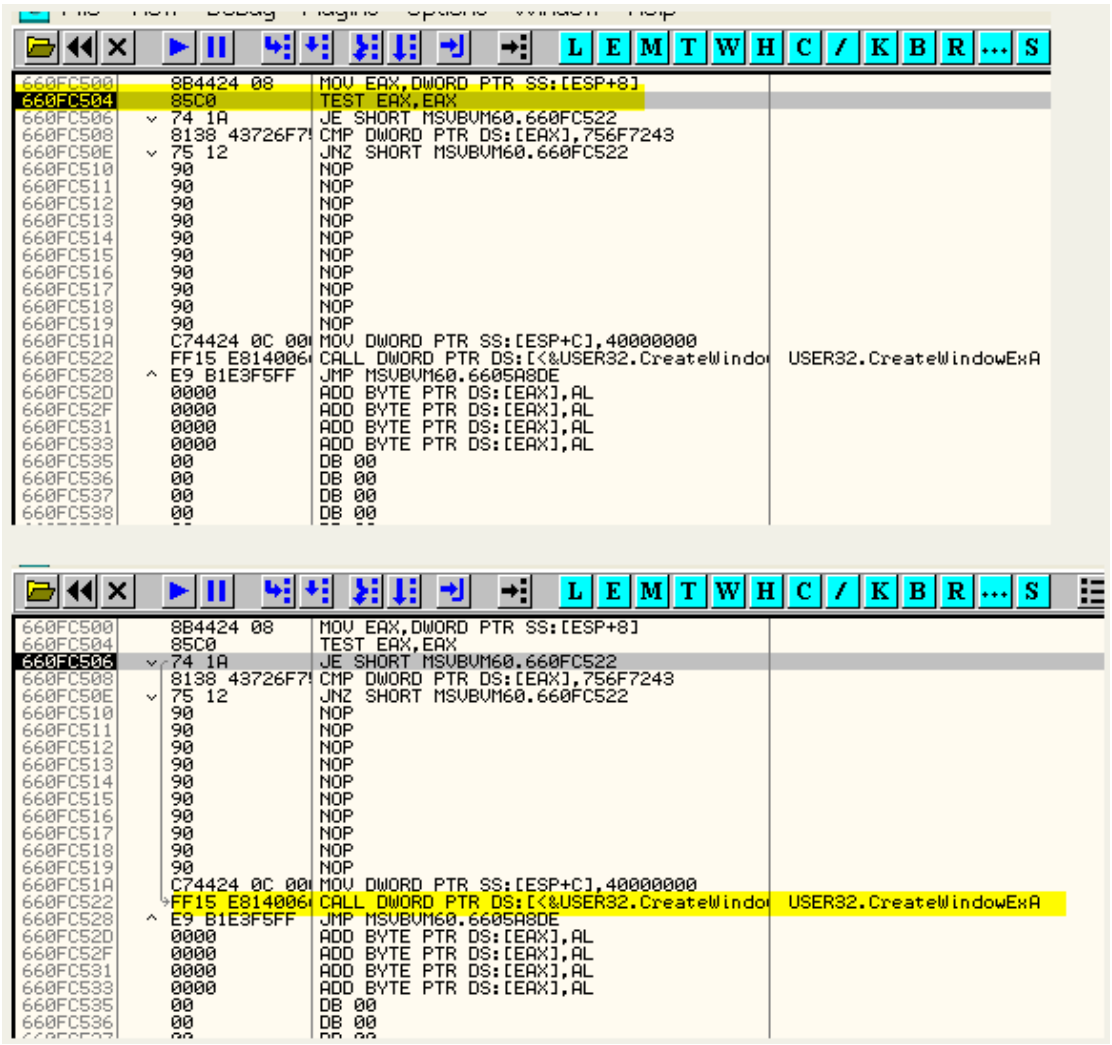
这说明 ESP+8 处保存的是窗口名称的首地址,我们将其保存到 EAX 寄存器中。



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。

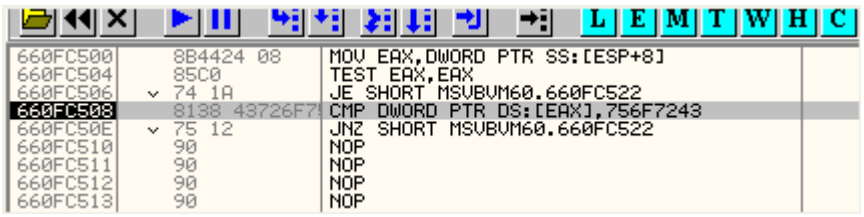


现在 EAX 中保存了窗口名称的首地址,接着我们在下一行中检查 EAX 是否为零,如果为零说明该窗口可能没有标题名称。

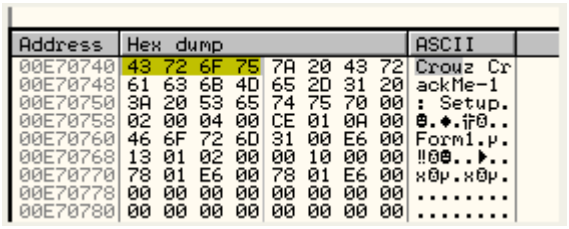


这里如果 EAX 为零,就不需要修改窗口的样式了,继续调用 CreateWindowExA 即可。

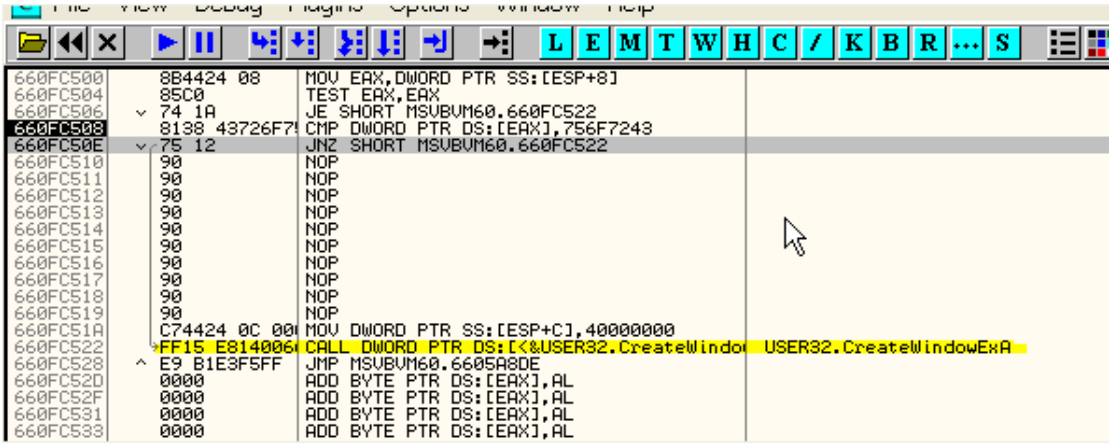
如果 EAX 不为零,则需要做进一步的判断。



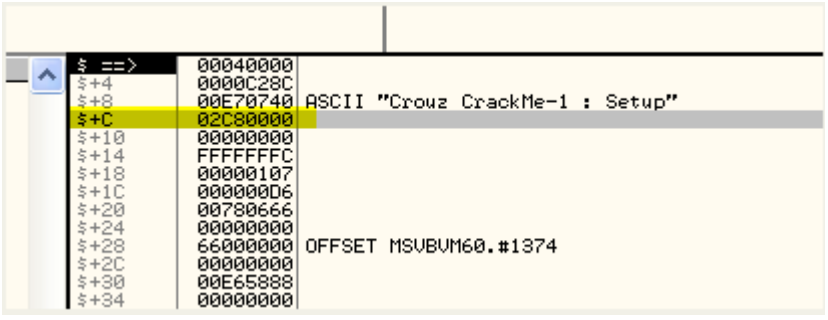
这里我们判断窗口标题名称的前 4 个字节是否为 756F7243(也就是 NAG 窗口标题 Crouz CrackMe-1 : Setup 的前 4 个字节)。



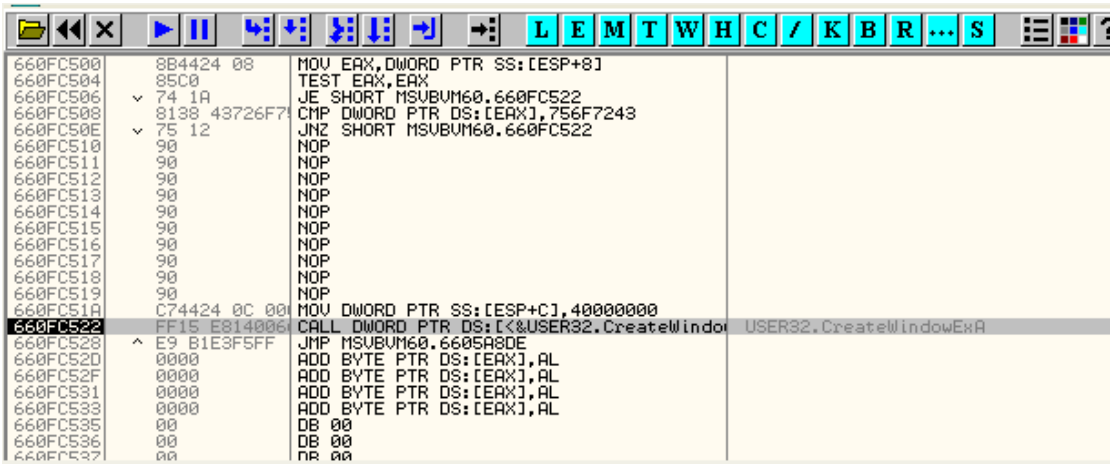
这里如果判断出的确是 NAG 窗口,那么继续往下执行修改窗口样式,如果不是 NAG 窗口的话,就不修改窗口样式,直接调用 CALL CreateWindowExA。



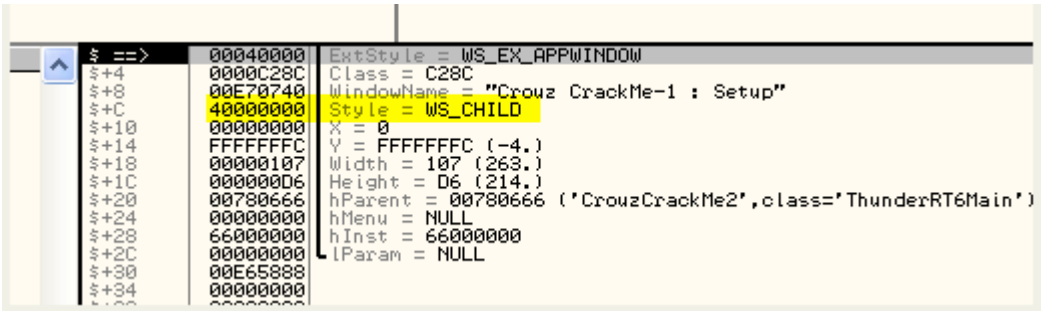
这里我们可以看到如果不是 NAG 窗口的标题的话,就直接跳转到 CALL CreateWindowExA 处,并不修改窗口样式。如果是 NAG 窗口的话,就将 ESP + C 指向的窗口样式参数值修改为 40000000(子窗口样式)。



当我们执行到这一行

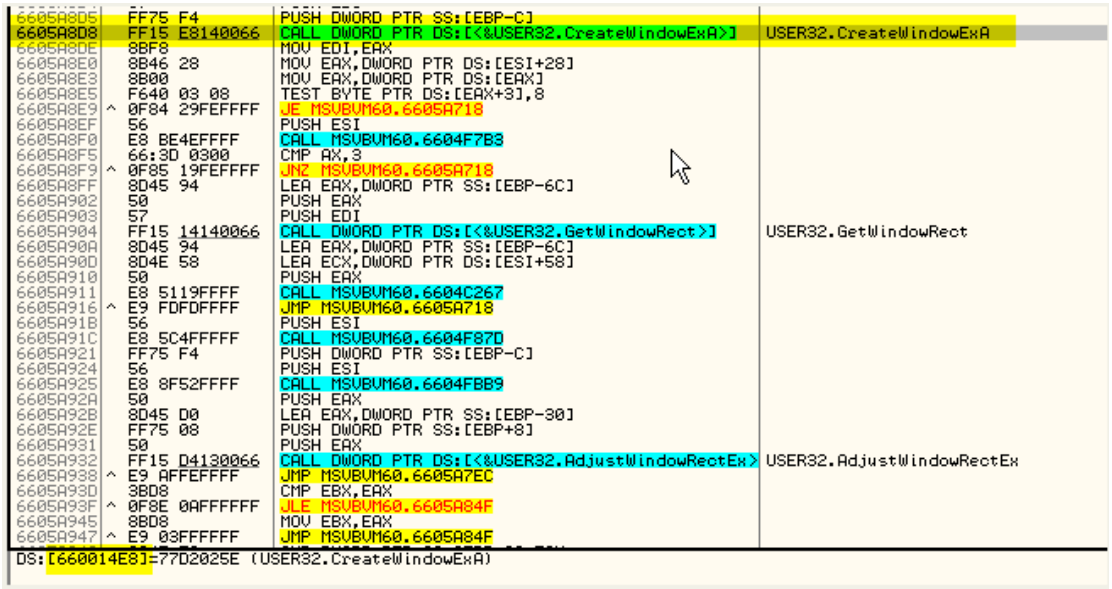


可以看到调用是 CreateWindowExA,我们来看看参数情况:



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。



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+C],40000000
660FC522    FF15 E8140060   CALL DWORD PTR DS:[<&USER32.CreateWindo  USER32.CreateWindowExA
660FC528  ^ E9 B1E3F5FF     JMP MSVBVM60.6605A8DE
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
```

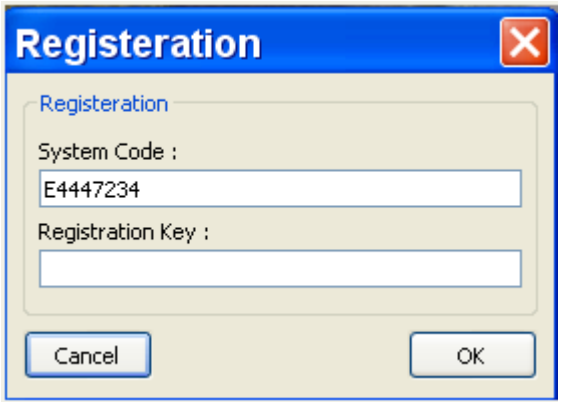
调用完 CreateWindowExA 以后我们已经返回到原来的下一行处继续执行 MOV EDI,EAX。

```
6605A8D5    FF75 F4         PUSH DWORD PTR SS:[EBP-C]
6605A8D8    FF25 00C40F66   JMP DWORD PTR DS:[660FC400]   MSVBVM60.660FC500
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
```

接着我们运行起来。

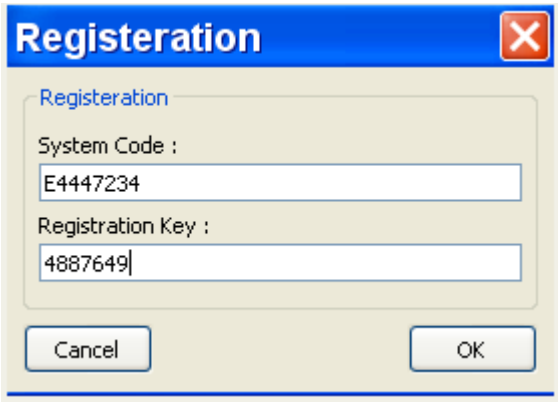
我们可以看到只弹出了注册窗口,没有出现 NAG 创建。总结一下我们需要的处理-检查窗口标题是否为 NAG 窗口,如果是就改变其窗口标题。

我们保存所有修改到文件直接运行起来看看效果。

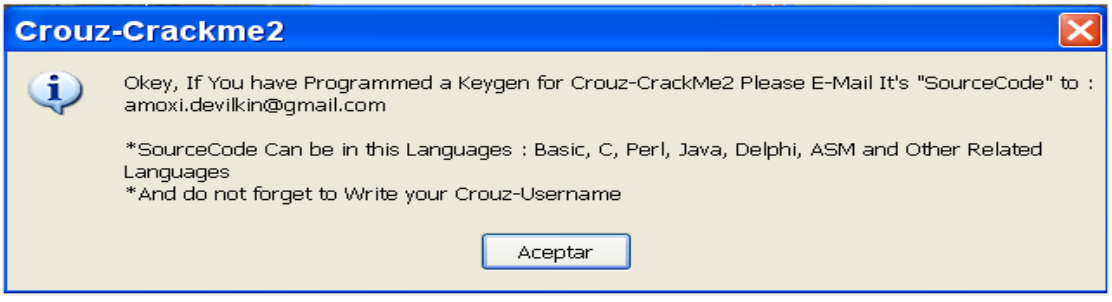


我们可以看到同样是直接弹出了注册窗口,没有出现 NAG 窗口,说明我们 Patch 的 CrackMe 和 MSVBVM60.DLL 成功剔除了 NAG 窗口。

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)。