# 第二十七章-Visual Basic 程序的破解-Part2

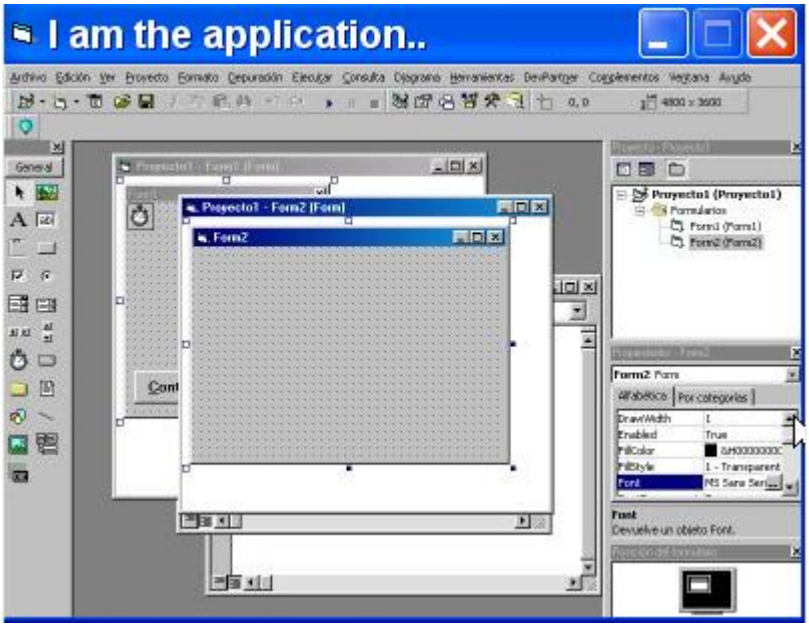
**Visual Basic 程序破解续**

本章我们继续讨论 VB 应用程序破解的话题,上一章我们解决的是一个带 NAG 窗口的 CrackMe。我们是用那个 patch 过的 OD 来弄的,本章我们也会用到,但是本章我们还会介绍另一种更加简便,更加省时的方法。首先我们还是用那个 patch 过的 OD 来分析,弄清楚了具体原理以后,我们再来介绍那种简便的方法。

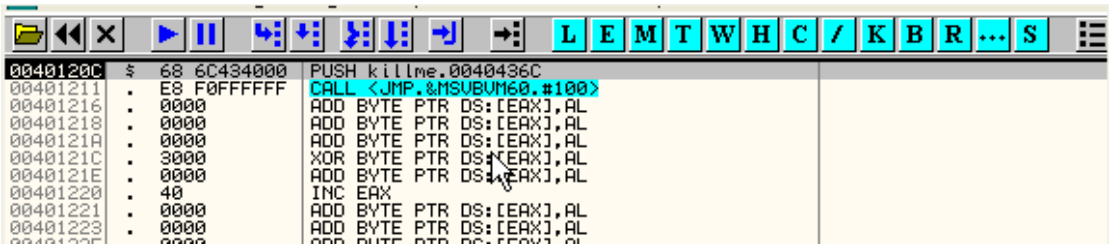
本章我们实验的对象名称叫做 killme,这个程序启动的时候会弹出一个烦人的 NAG 窗口,我们需要想办法把它剔除掉。

我们直接运行起来的话,可以看到弹出一个 NAG 窗口。


我们可以看到 Continue 按钮是灰色的(不可用),当定时器由 5 减少到 0 后,Continue 按钮就被激活了,我们单击 Continue 按钮。



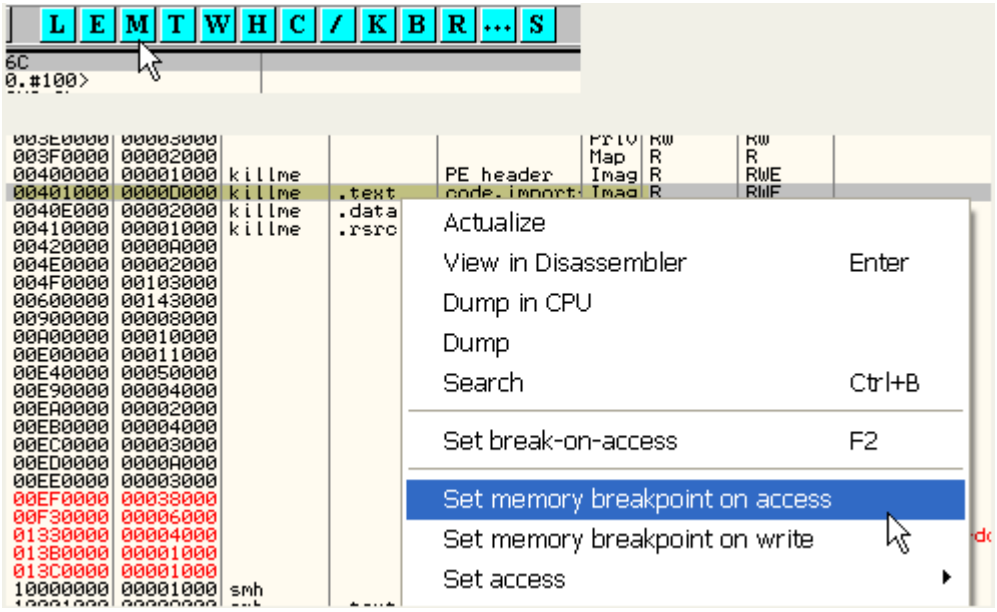

NAG 关闭了,并且弹出主程序窗口,我们用 OD 加载该程序。



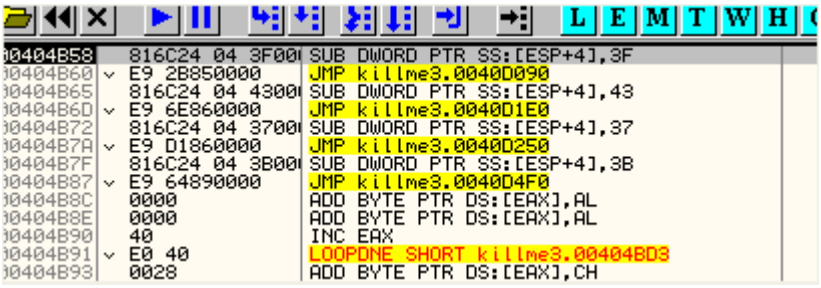

我们按 F9 键运行起来。


定时器时间到了后,Continue 按钮就被激活了。

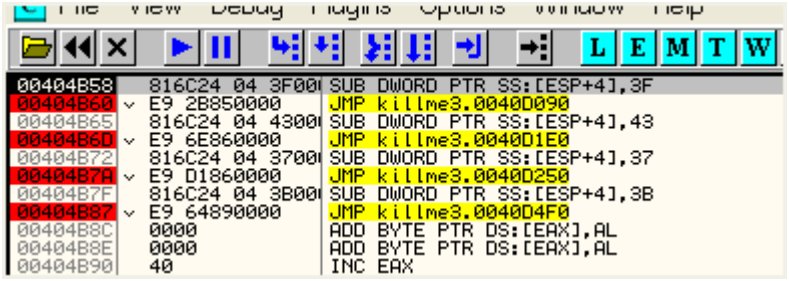
我们可以想象一下,NAG 窗口关闭后,代码中应该会有一个无条件跳转到主窗口代码执行,所以我们给代码段设置内存访问断点(该内存访问断点实际上只是内存执行断点),接着单击 Continue 按钮。



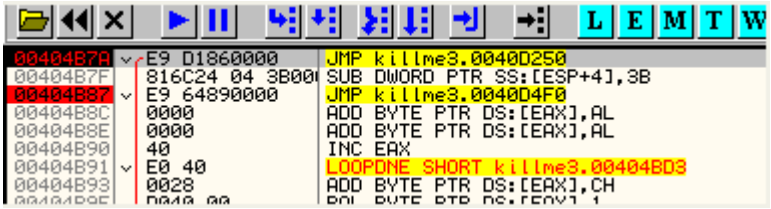

单击 Continue 按钮后,我们断在了这里。



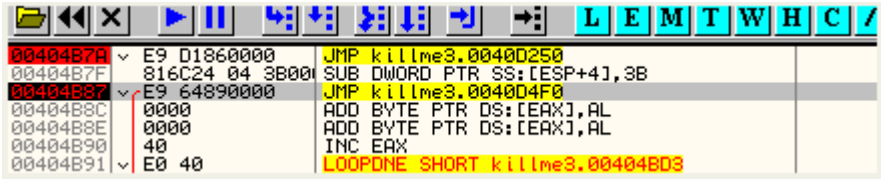

我们可以看到 404B60 处 JMP 指令会跳转到 40D090 地址处开始执行,也就是说主窗口程序实际上是从 40D090 处开始执行的,这里说主窗口程序可能不太准确,因为我们可以看到这里有好几处 JMP 分支会跳转到程序的不同部分执行。我们给这几处 JMP 指令分别设置断点,然后来看看每个分支跳转分别在什么情况下被触发。



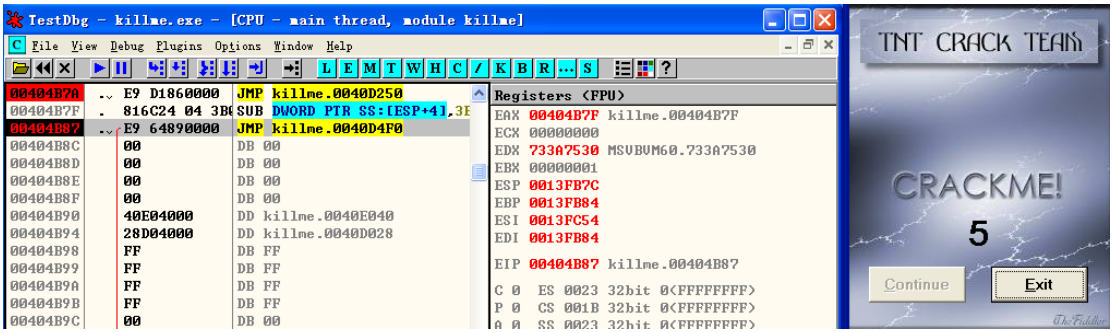

我们重新启动该程序。



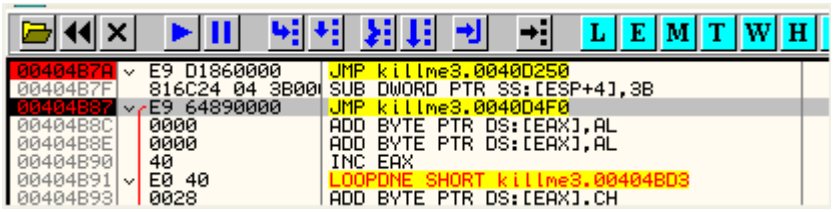

刚刚我们看到了单击 NAG 窗口上的 Continue 按钮后,会执行 JMP 40D090 这个分支。现在我们重新启动程序后,断在了 JMP 40D250 这个分支处。NAG 窗口还没有显示。我们继续运行起来。




我们可以看到断在了 JMP 40D4F0 这个分支处。这个时候 NVG 窗口已经显示出来了,上面显示了数字 5。

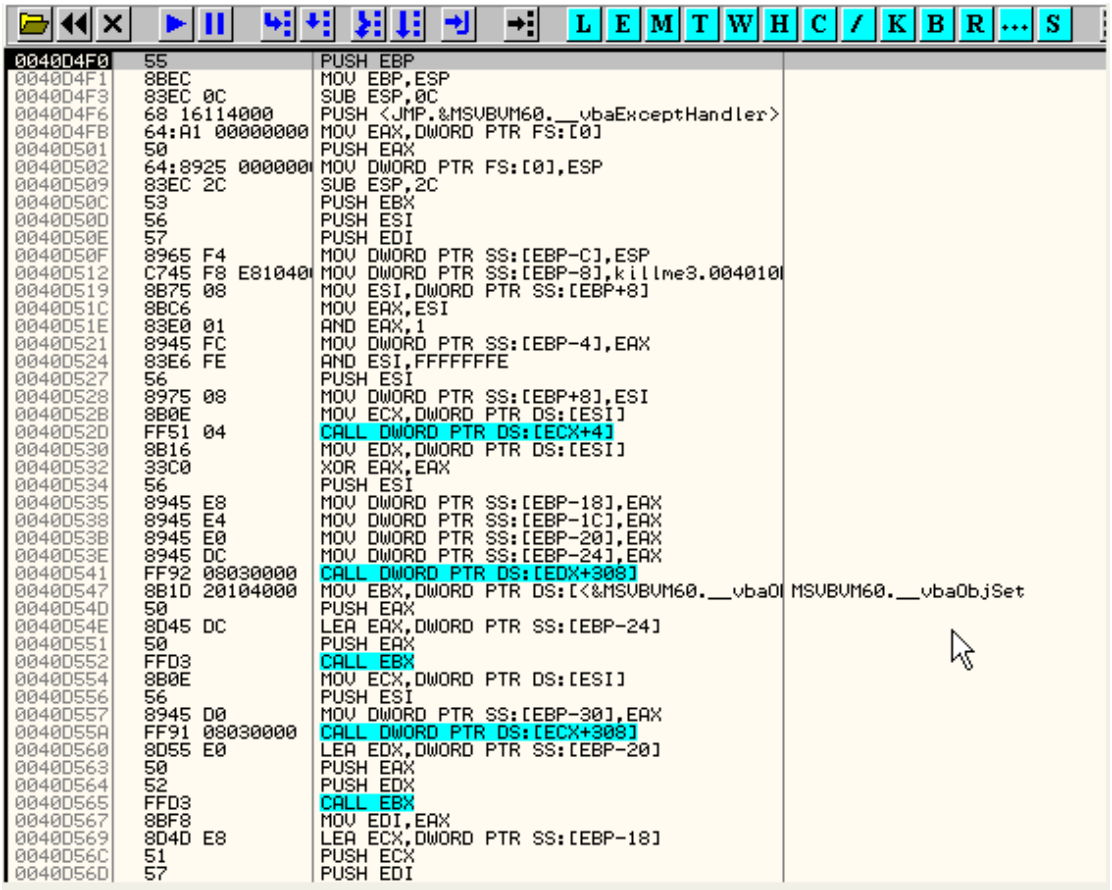



我们继续运行。



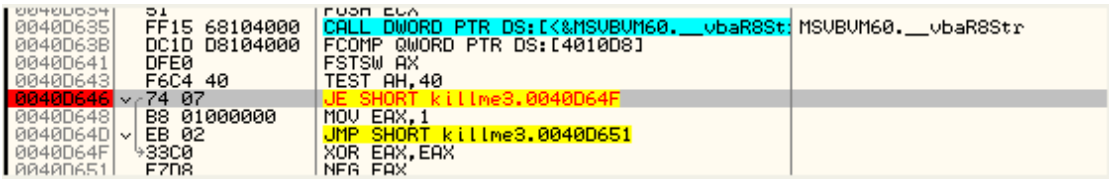

又断在了 JMP 40D4F0 这个分支处。这个时候 NVG 窗口上的数字变为了 4。我们会发现 JMP 40D4F0 这个分支一共会断下来 5 次。也就是说,这个分支过程与定时器相关。

让我们来跟进到这个分支中一探究竟。

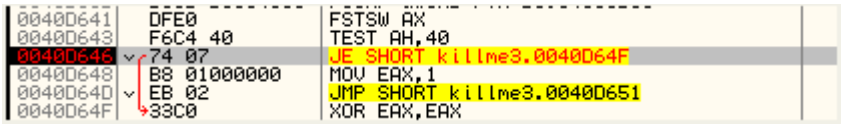



没看出什么特别的地方,我们继续按 F8 键跟踪。

我们跟到了一处条件跳转 JE 40D64F 处。我们来分析一下看看这个条件跳转是不是决定定时器停止并激活 Continue 按钮的关键跳转。




上面是一个浮点比较指令,由于上下文的线索并不太多,所以我们暂时还看不出它的作用是什么。我们给这个 JE 40D64F 设置一个断点,接着运行起来。




断在了 JE 40D64F 这个分支处,并且该跳转将成立,我们多运行几次会发现,计时器减至 0 之前这个跳转一直都是成立的,只有当定

时器显示为 0 时,该跳转才不成立。所以我们可以得知,该条件跳转是用来判断定时器是否为 0 的。我们将这一行 NOP 掉看看会发生什么。

```
0040D63B   DC1D D8104000   FCOMP QWORD PTR DS:[4010D8]
0040D641   DFE0            FSTSW AX
0040D643   F6C4 40         TEST AH,40
0040D646   90              NOP
0040D647   90              NOP
0040D648   B8 01000000     MOV EAX,1
0040D64D   EB 02           JMP SHORT killme3.0040D651
0040D64F   33C0            XOR EAX,EAX
0040D651   F7D8            NEG EAX
0040D653   8D4D E8         LEA ECX,DWORD PTR SS:[EBP-18]
```

好了,已经 NOP 掉了,这样这里就不会跳转到 40D64F 处了,我们运行起来。


我们可以看到,定时器并没有减至 0,而 Continue 按钮已经被激活了。所以我们对于该条件跳转是用来判断定时器是否为 0 的假设是成立的。好了,现在我们这个 NAG 窗口关闭,当当前这个函数返回到 VB 的 DLL 中时,那么该 NAG 窗口将继续显示,所以我们需要定位该函数何时返回到 VB 的模块中,定位到返回指令后,我们可以将返回指令指定返回到显示主窗口程序的分支 40D090 即可,这样 NAG 窗口就会主动关闭而不需要人为的单击 Continue 按钮了。

现在我们重新启动程序。

我们来到刚刚 NOP 掉的那一行。

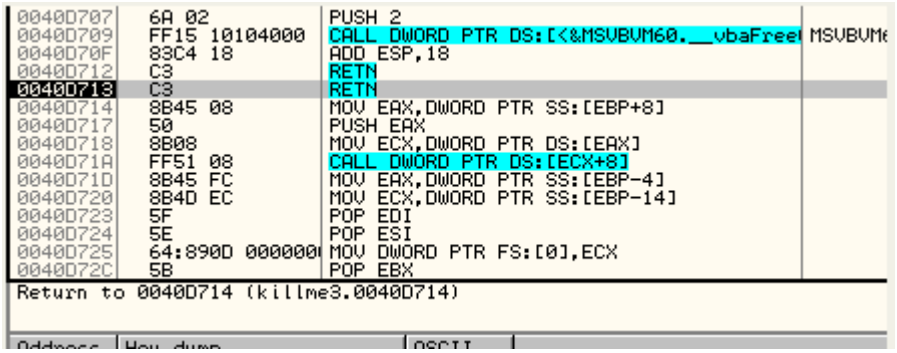
```
0040D641   DFE0            FSTSW AX
0040D643   F6C4 40         TEST AH,40
0040D646   90              NOP
0040D647   90              NOP
0040D648   B8 01000000     MOV EAX,1
0040D64D   EB 02           JMP SHORT killme3.0040D651

0040D63B   DC1D D8104000   FCOMP QWORD PTR DS:[4010D8]
0040D641   DFE0            FSTSW AX
0040D643   F6C4 40         TEST AH,40
0040D646   90              NOP
0040D647   90              NOP
0040D648   B8 01000000     MOV EAX,1
0040D64D   EB 02           JMP SHORT killme3.0040D651
```

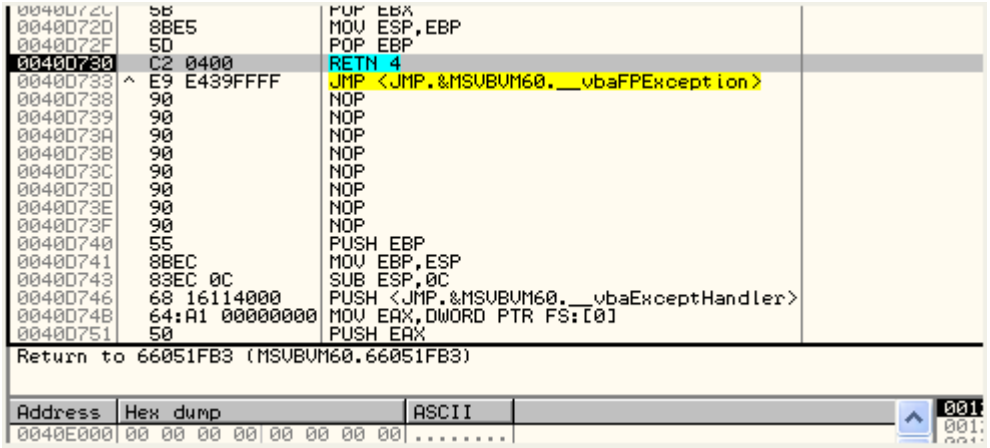
我们删掉之前设置的所有断点,接着给 TEST AH,40 这一行设置一个断点,运行起来。

```
0040D63B   DC1D D8104000   FCOMP QWORD PTR DS:[4010D8]
0040D641   DFE0            FSTSW AX
0040D643   F6C4 40         TEST AH,40
0040D646   90              NOP
0040D647   90              NOP
0040D648   B8 01000000     MOV EAX,1
```

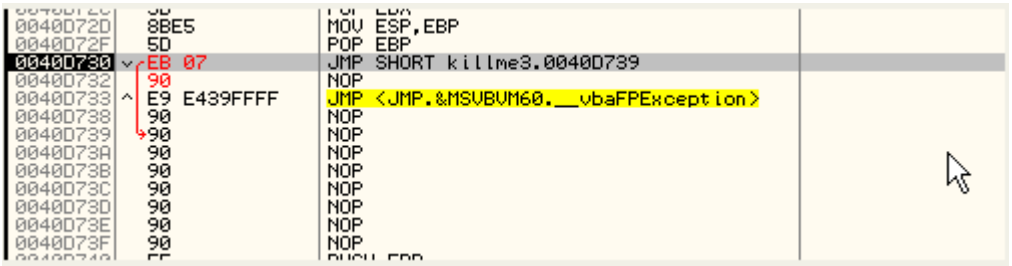
断了下来,我们按 F8 键往下跟踪,看看哪里会返回到 VB 的 DLL 中,返回到了 VB 的 DLL 中的话程序就会运行起来。这样 NAG 窗口就会显示出来并等待我们按 Continue 或者 Exit 键。



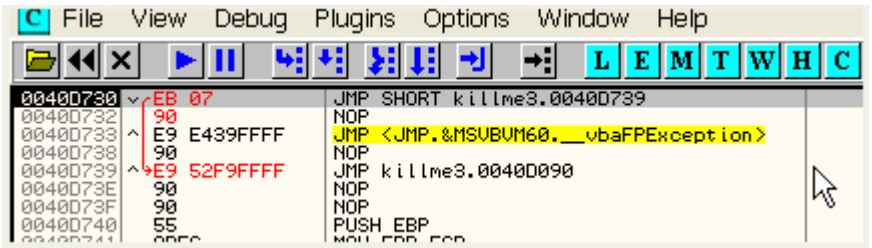

我们到达了 40D713 处的 RETN 指令处,但是我们会发现该处会返回到紧接着的下一行 40D714 处,并不是返回到 VB 的 DLL 中,所以我们继续跟踪。



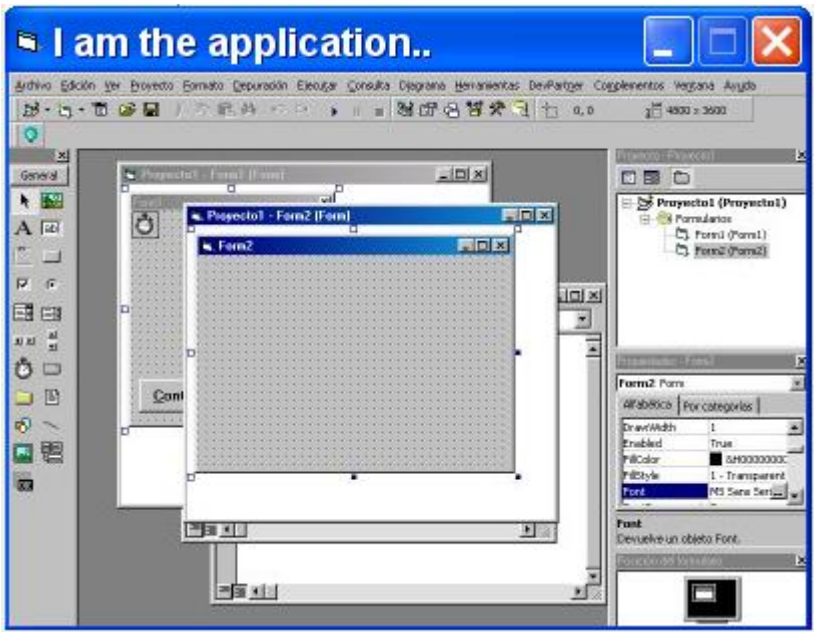

我们跟踪到了 40D730 处的 RETN 4 指令处,从解释窗口中我们可以看出来这里将返回到 MSVBVM60.DLL 中。接着程序将运行起来了。NAG 窗口将继续显示,为了剔除掉 NAG 窗口,这里我们可以尝试将该 RETN 4 修改为 JMP 指令,让其直接跳转到显示主窗口的代码处(40D090)。还有一点需要注意,这里 RETN 4 占 3 个字节,但是 JMP 40D090 占 5 个字节。我们还可以注意到 40D738 起始处有几个字节的 NOP 空间可以利用。所以我们可以先跳转到 40D739 处。



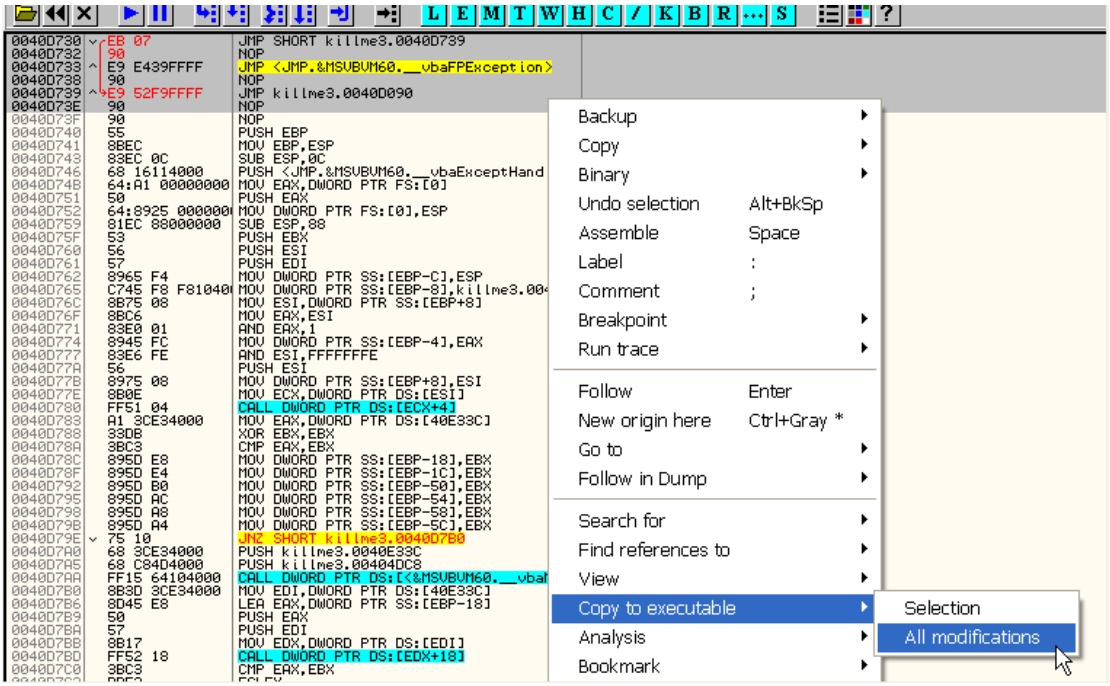

接着我们在 40D739 写入 JMP 40D090 指令即可。这样就可以直接跳转到显示主程序的代码开始执行,NVG 也被关闭了。



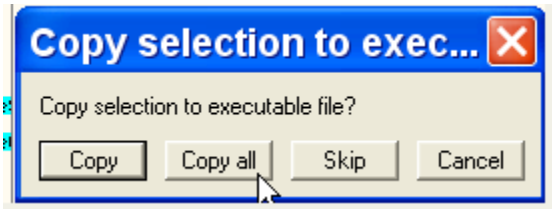

我们直接运行起来看看效果。



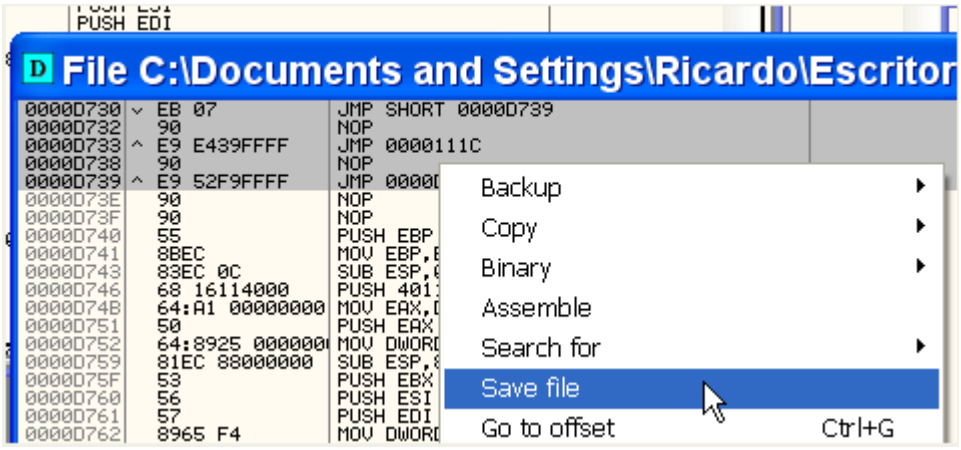

我们可以看到主窗口直接显示出来,NAG 不见了。好,现在我们将之前做的修改(NOP 掉的,两处跳转)保存到文件。



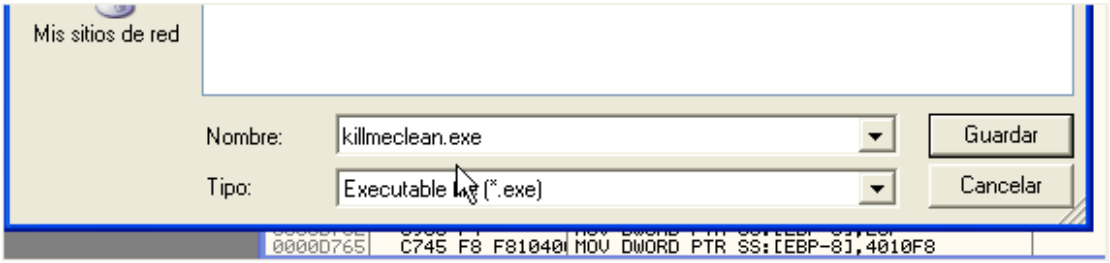

我们单击鼠标右键选择-Copy to executable-All modifications,这样就可以保存所有修改了。



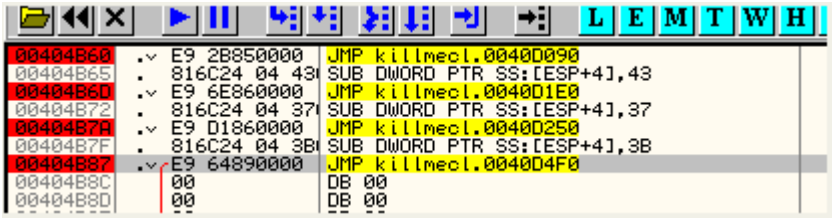

选择 Copy all。



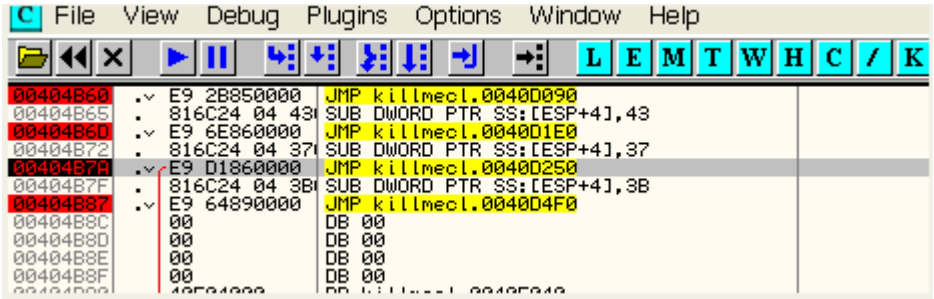

单击鼠标右键选择 Save file。



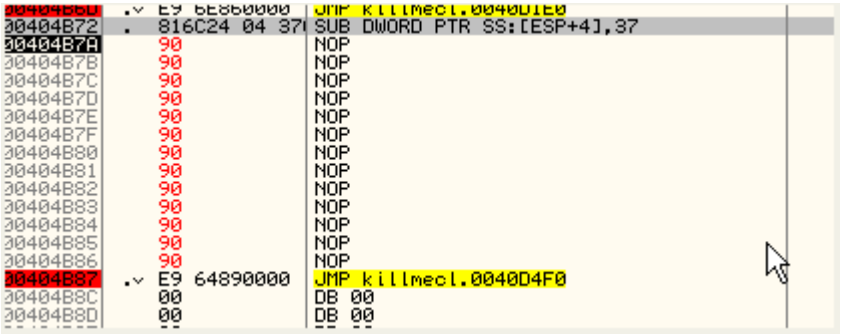

好了,保存名为 killmeclean.exe,直接双击运行起来,我们会发现 NAG 窗口还是出现了,不过几秒钟后就消失了,然后主窗口就弹出来了。就是说 NAG 窗口还没有被完全剔除掉,所以我们还是来到之前分析的多分支处。



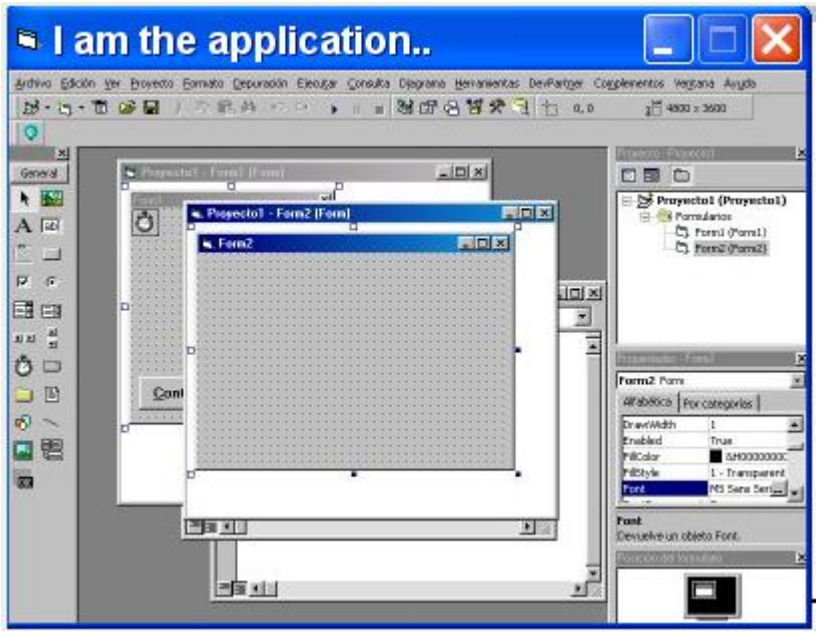

运行起来。



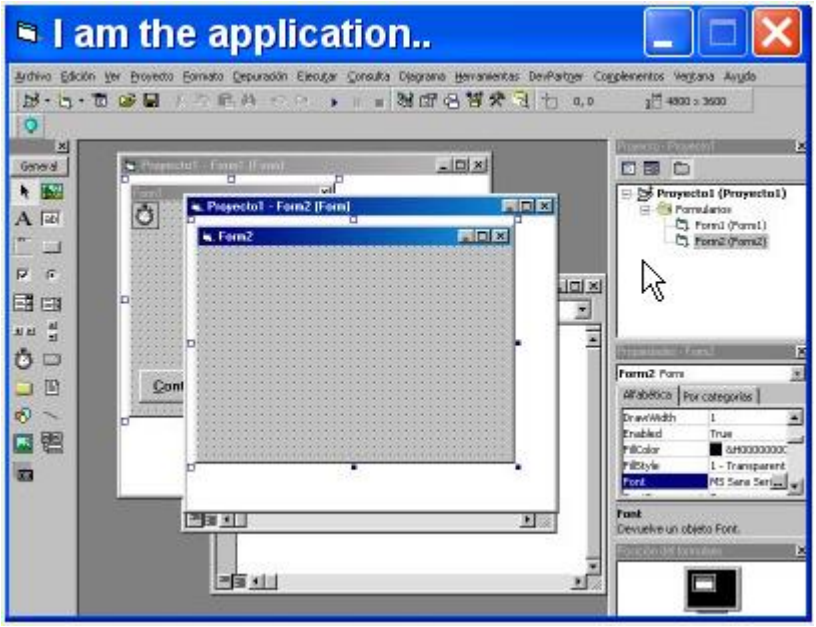

我们可以看到在 NAG 窗口创建之前断在了 404B7A 处,我们运行起来以后,几秒钟后,就断在了 404B87 处,并且这个时候 NAG 窗口被创建并显示出来了。所以 JMP 40D250 这个分支应该是创建并显示 NAG 窗口的分支。我们用 OD 加载刚刚我们 patch 过的 killmeclean 这个程序,我们将创建 NAG 窗口的跳转 NOP 掉。




NOP 掉创建 NAG 窗口的分支后将直接转入定时器计时的分支执行,我们按 F9 键运行起来看看效果。



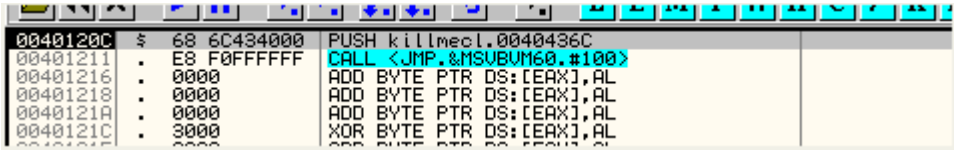

我们可以看到直接弹出了主窗口,并没有显示 NAG 窗口,我们保存所有修改到文件,然后直接双击运行看看效果。



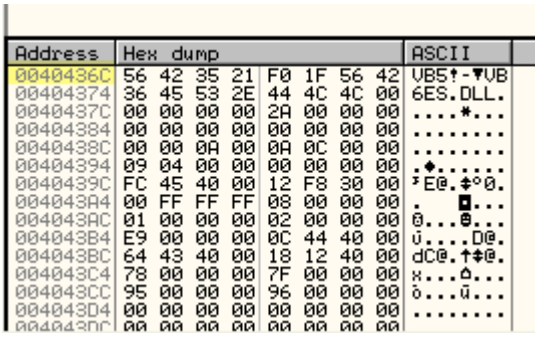

我们可以看到直接就弹出了主窗口,并没有出现 NAG 窗口,嘿嘿。这样我们将手工剔除了这个 NAG 窗口。

4C 法-这样方式更加快捷方便,该法是基于 VB 程序的特性来的。

下面我们就来介绍这种方法,我们用 OD 加载 killme 程序,停在了入口点处。




我们会注意到 VB 程序有个特点-入口点处都是一个 PUSH 指令,然后一个 CALL 指令。(如果你遇到的不是这种情况的话,那么该程序可能被加过壳) PUSH 将要压入堆栈的是 40436C,现在我们在数据窗口中定位到这个地址。

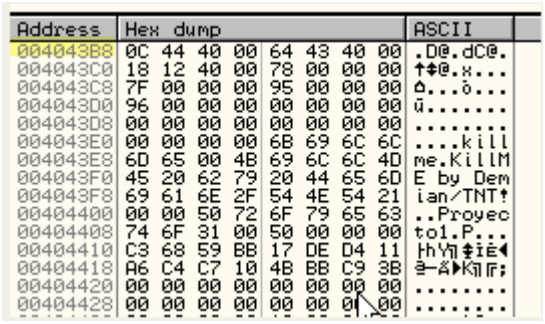



正如你说看到的,这就是 VB 程序的头部,这里我们将 40436C 加上 4C。

也就是

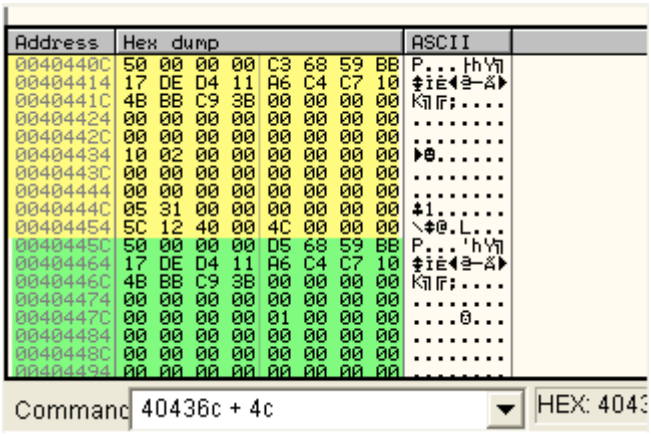
40436C + 4C


也就是 4043B8。



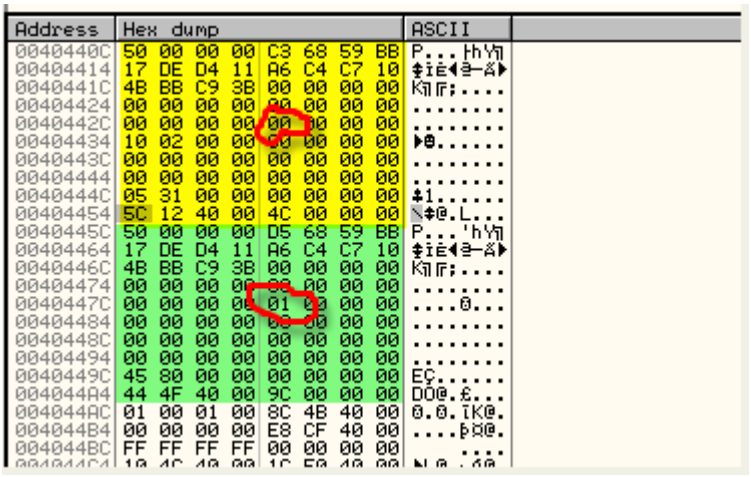

这里我们可以看到 4043B8 指向的内存单元中保存的是 40440C。(PS:这里作者写成了 4044C0,我已经更正为 40440C)

我们在数据窗口中定位到 40440C 这个地址。

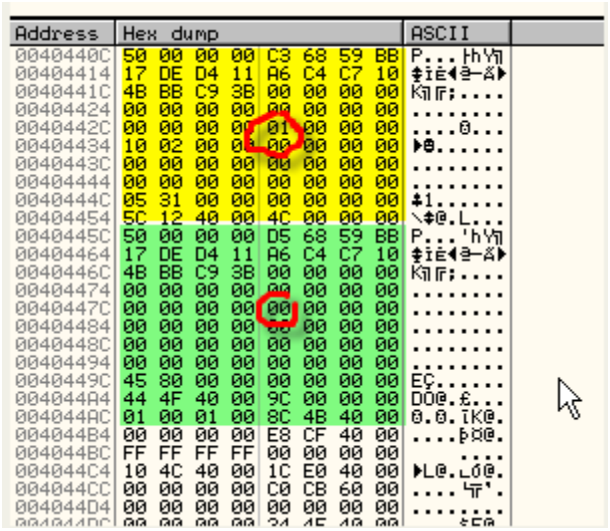



这里我们可以看到两块类似的数据,每块 50(十六进制)个字节的长度,每块数据的第 24(十六进制)个字节处都有一个标志。该标志指定了每块代码出现的顺序。我们一起来看看吧。

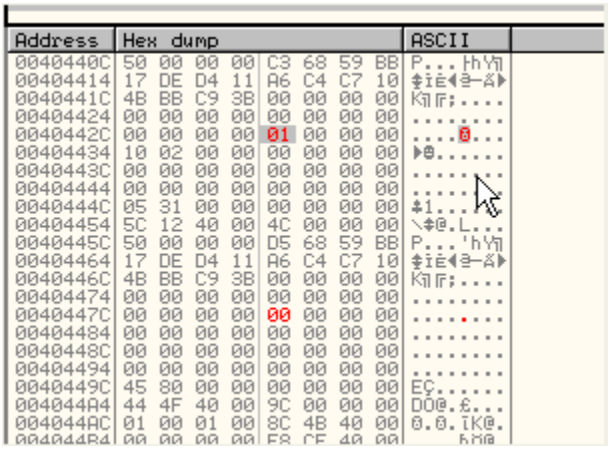
40440c +24=404430




我们可以看到第一块的中标志是 00,表示该部分代码将首先执行,而第二块中的 01 表示随后才会执行,所以这里我们将各两个标志的值颠倒一下。




这样首先弹出的就是主窗口了,嘿嘿。然后才是 NAG 窗口,其实 NAG 窗口根本不会弹出来,因为主窗口关闭后,应用程序就退出了,NAG 窗口压根就没有机会弹出来。




这样我们就用 4C法方便快捷的搞定了这个NAG程序,但是这样方式并不适用于所有的程序,所以为什么一开始我要给大家介绍分析 VB 程序的常规手法,了解原理对我们非常有帮助。

这里留两个 CrackMe 给大家练习,一易一难,名称分别为 CrackMe(这个程序的运行需要 MSVBVM50.DLL 的支持),CrackMe2。大

家尝试一下看看能不能找出序列号以及剔除 NAG 窗口,下一章我们再来讲解这两个例子。