第二十六章-Visual Basic 程序的破解-Part1

接下来几章我们将介绍如何使用 OllyDbg 破解 Visual Basic 编写的程序,有人可能会说破解 VB 有更好的工具,比如 SmartCheck,但是本系列教程主要是围绕 OllyDbg 展开的,所以会尽可能的尝试少使用其他工具,如果 OllyDbg 实在解决不了的时候,我们再来使用其他工具。

首先,给大家准备了一个好用的 OllyDbg。

名字叫做 olly_parcheado_para_vb(翻译过来就是:打过补丁的 OD,专门用于定位 OEP),也就是说用其在脱壳过程中定位 OEP 比较方便。但是实际上,用它分析 VB 编写并编译成 Native code 的程序也比较方便(但并不适用于 VB 编写的并编译成为 P-CODE 的程序)。VB 编写的应用程序有两种编译方式,一种是 Native code 方式,另一种是 P-code 方式。实际上,VB 4.0 以前只采用 P-code 编译,VB 4.0 以前只采用 P-code 编译,VB Native Code 是在 VB 5.0 以后发展起来的,其目的是为了在一定程序上改善 VB 应用程序的运行速度。VB P-Code 的运行速度较慢,这是由它本身的运行机制所决定的。P-code,即 Pseudo Code(伪代码),这一概念最早出现在 Pascal 编译器中,它是为了提供跨平台可移植性而产生的,实现这一编译机制的 Pascal 编译器被称之为"Pascal P Comiler"。Sun 公司在其推出的 Java 语言上也成功实现了这种机制。Java 程序的伪编译代码由一系列代表一定意义的字节码(byte code)组成。它们同属于一套特定的指令集,这种字节码不能由不同的 CPU 直接执行,而是通过特殊的解释器翻译为 CPU 可以识别的指令才能执行,这种解释器就是我们常说的"虚拟机"。只要在不同的平台上提供虚拟机,把字节码翻译为对应的 CPU 指令集,也就实现了所谓的跨平台特性。微软推出的 VB P-code,实际上也是一组自定义的指令集,必须通过基于堆栈的虚拟机翻译为 80X86 上的指令集才能执行,担任虚拟机任务的就是 msvbvm50.dll 和 msvbvm60.dll 这两个动态链接库文件。由于在文件执行过程中多出了解释的步骤,自然要影响到其执行的速度。正如我们所看到的,VB P-code 并没有实现所谓的跨平台运行特性,这对于 Pseudo 这一词的起源就不恰当了,另一方面,采用 P-code 形式编译的 VB 应用程序的体积要小于 Native Code 形式编译的同样程序(这是由于 P-code 指令集的每条指令对应于一组 80X86 指令所完成的任务),所以 VB P-code 实际上意味着 VB Packed-code(压缩代码),用以强调 VB P-code 程序较小的代码体积。所以说这两者之间还是存在显著差别的。

关于 P-code 的具体原理我们后面章节再讨论。

那么刚刚那个打过补丁的 OllyDbg 有什么特别的地方呢?其特别之处在于:如果你设置一个内存访问断点,正常情况下应该是内存读取(ON READ)或者执行(ON EXECUTE)都断下来。但是该 OllyDbg(前面提到的 Patch 过的)只会在执行(ON EXECUTE)的时候断下来。这样更有助于我们寻找应用程序的 OEP,对于分析非 P-code 的 VB 程序也非常有帮助。

另外,我们要破解 VB 程序的话,首先应该了解一些 VB 的 API 函数,但是 VB 相关的 API 函数在 MSDN 中查不到。

并且网络上也可以找到很多专门针对 VB 的教程,你可以通过阅读 COCO 等人的 VB 系列破解教程来提高自己分析 VB 程序的功力。我这里的话直接使用这个 Patch 过的 OD 来破解 VB 程序了。

我们可以将这个 Patch 过的 OD 放到原版 OD 的根目录下,我们分析普通程序的时候使用原版的 OD,当我们需要破解 VB 程序,或者需要定位 OEP,再或者需要内存访问断点仅对内存执行断点生效的时候我们就可以使用这个 Patch 过的 OD。

但是如果我们需要读取或者执行都能生效的话,我们就可以使用硬件访问断点来替代。

现在最纠结的问题是 MSDN 中并没有提供 VB API 函数相关的解释,如果你想知道某个 VB API 函数的解释,只能 Google 了,如果你运气好的话,也许会搜索到一些对该 API 进行相应解释的页面。好了,我们介绍如何破解 VB 程序之前,先来看一看我收集的一些重要的 VB API 函数资料。

VB 的主要数据类型以及 API 函数

函数名称中缩写形式归纳:

bool - 布尔型

str - 字符串型

i2 - 双字节整型

ui2 - 无符号双字节整型

i4 - 长整型

r4 - 单精度浮点型

r8 - 双精度浮点型

cy - 货币型

var - 变体类型

fp - 浮点型

cmp - 比较

comp - 比较

下面是 VB 基本数据类型的图表:

数据类型	类型描述	所占字节数	数据范围
Integer	整型	2	-32768~32767
Long	长整型	4	-2147483648~2147483647
Single	单精度浮点型	4	-3.402823E38~-1.401298E-45 1.401298E-45~3.402823E38
Double	双精度浮点型	8	-1.79769313486232E308~-4.940656458412E324 4.940656458412E324~1.79769313486232E308
String	字符串型	每个字符占一个字节	
Byte	字节型	1	0~255
Boolean	布尔型		True,False(1,0)
Currency	货币型	8	
Date	日期型		
Object	对象型	4	
Variant	变体		

例如:

__vbaI2Str 可以将一个字符串转化 2 个字节的数值形式。

更多的定义如下:

1)数据类型转换类函数:

I)__vbaI2Str 将一个字符串转化为整型

II)__vbaI4Str 将一个字符串转化为长整型

III)__vbar4Str 将一个字符串转化为单精度浮点型

IV)__vbar8Str 将一个字符串转化为双精度浮点型

V)VarCyFromStr 将字符串转化为货币类型

VI)VarBstrFromI2 将整型数据转化为字符串

以上是几个缩写形式的 VB API 函数,我们再来看看其他的 VB API。

2)数据移动:

I)__vbaStrCopy 将一个字符串拷贝至指定内存单元中

II)__vbaVarCopy 将一个变量的值拷贝至指定内存单元中

III)__vbaVarMove 将一个变量的值移动到指定内存单元中

3)数学运算:

I)__vbavaradd 两个变量值相加

II)__vbavarsub 第一个变量减去第二个变量

III)__vbavarmul 两个变量值相乘

IV)__vbavaridiv 第一个变量除以第二个变量,得到一个整数商

V)__vbavarxor 两个变量值做异或运算

4)程序设计杂项:

I)__vbavarfornext 这里 VB 程序里的循环结构,For...Next...(Loop)

II)__vbafreestr 释放掉指定字符串所占的内存,也就是把指定内存单元中字符串抹掉

III)__vbafreeobj 释放掉 VB 的一个对象(一个窗口或者一个对话框)所占的内存,也就是把指定内存中窗口或者对话框对象抹掉

IV)__vbastrvarval 获取字符串指定的子串

V)multibytetowidechar 将多字节字符串转化为宽字节字符串

VI)rtcMsgBox 弹出一个消息框,类似于 WINDOWS API 中的 MessageBoxA/MessageBoxExA 函数

VII)__vbavarcat 将两个变量值相连,如果是两个字符串,就直接连接在一起

VIII)__vbafreevar 释放变量所占的内存空间,也就是把指定内存中的变量值抹掉

IX)__vbaobjset 给对象赋值或者实例化

X)__vbaLenBstr 获取一个字符串的长度,注意:VB 中一个汉字的长度也是 1 个字节

XI)rtcInputBox 显示一个 VB 标准的输入窗口,类似于 WINDOWS API 函数 GetWindowTextA,GetDlgItemTextA

XII)__vbaNew 显示一个对话框,类似于 WINDOWS API 函数 DialogBox

XIII)__vbaNew2 显示一个对话框,类似于 WINDOWS API 函数 DialogBoxParamA

XIV)rtcTrimBstr 将字符串左右两边的空格去掉

XV)__vbaEnd 结束进程

XVI)__vbaLenVar 获取一个变量的大小

XVII)rtcMidCharVar 从字符串中间去相应的字符,VB 中的 MID 函数,用法 MID(“字符串”,“开始的位置”,“取几个字符”)

XVIII)rtcDir 获取当前路径

XIX)__vbaFileOpen 打开文件

5)比较函数:

I)__vbastrcomp 比较两个字符串,类似于 WINDOWS API 函数 lstrcmp

II)__vbastrcmp 比较两个字符串,类似于 WINDOWS API 函数 lstrcmp

III)__vbavartsteq 比较两个变量值是否相等

IV)__vbaFpCmpCy 浮点变量值与货币变量值进行比较

V)__vbavartstNe 判断两个变量值是否不相等

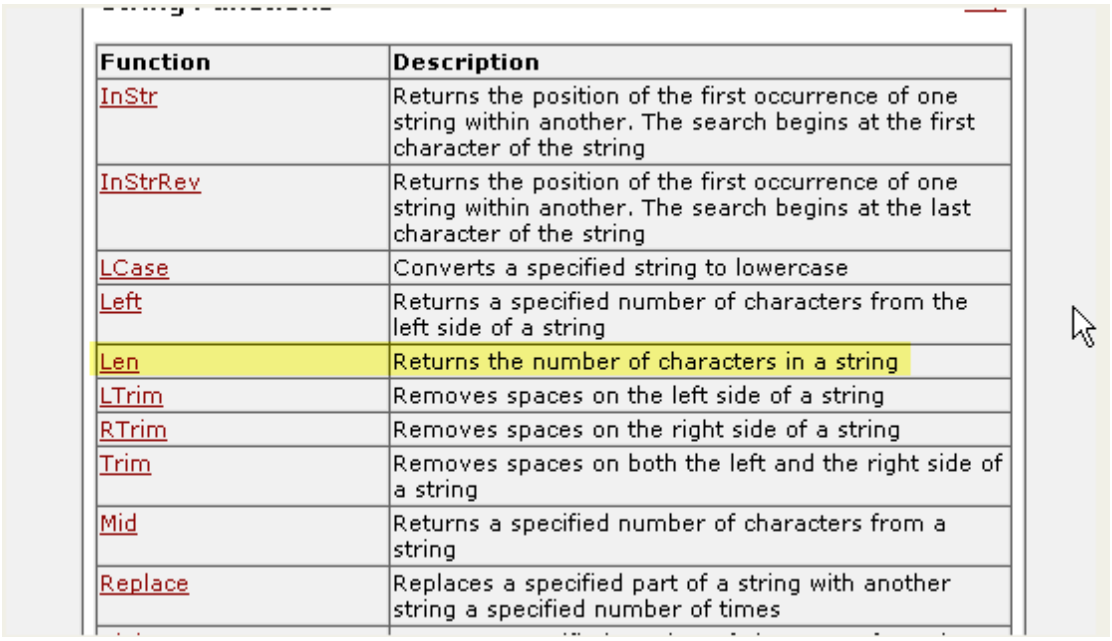
以上函数可能在以后我们分析 VB 程序的时候经常遇到,虽然并没有提供 VB API 函数的完整说明,但是你知道了这些函数,对你分析 VB 程序还是大有裨益的。

虽然我们不使用 Smart Check 来进行破解,但是其分析 VB 程序时常会显示的一些函数我们还是有必要知道的,函数列表见 W3school 中的介绍:

http://www.w3schools.com/vbscript/vbscript_ref_functions.asp

例如:Smartcheck 分析 VB 程序显示的报告中会出现 Asc 或者 Mid 这种函数,我们在 OllyBbg 也可以见到这类函数,但是这类函数并不属于 Visual Basic。以上网址中有对 Smartcheck 分析报告中出现的这类函数的详细解释。

例如:在 Smartcheck 中出现了 Len 函数,其对应的 VB API 函数是__vbaLenVar。



Function	Description
InStr	Returns the position of the first occurrence of one string within another. The search begins at the first character of the string
InStrRev	Returns the position of the first occurrence of one string within another. The search begins at the last character of the string
LCase	Converts a specified string to lowercase
Left	Returns a specified number of characters from the left side of a string
Len	Returns the number of characters in a string
LTrim	Removes spaces on the left side of a string
RTrim	Removes spaces on the right side of a string
Trim	Removes spaces on both the left and the right side of a string
Mid	Returns a specified number of characters from a string
Replace	Replaces a specified part of a string with another string a specified number of times

单击超链接我们可以看到 Len 函数的完整说明。

The Len Function

Back

The Len function returns the number of characters in a string.

Syntax

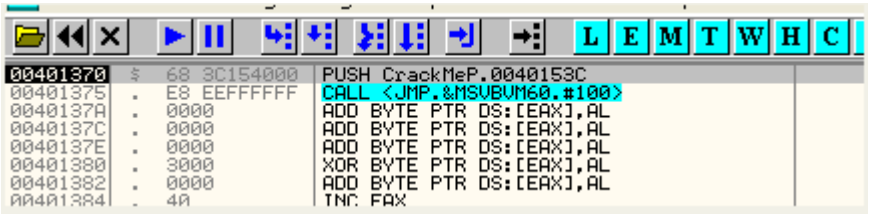
```
Len(string|varname)
```

Parameter	Description
string	A string expression
varname	A variable name

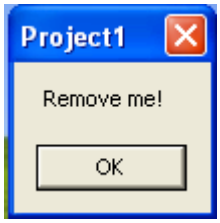
以上是大家以后使用 SmartCheck 分析 VB 程序过程中可能会遇到的问题,尽管我们并不使用它来破解 VB 程序,但是我们这里还是简单提一下,因为我暂时还没看到关于 SmartCheck 的教程中有相关函数的说明。

好了,可能会涉及到的知识点我们都知道了,现在我们来看一个 CrackMe,名字叫做 CrackMePls。

我们用本章开始介绍过的 Patch 过得 OllyDbg 来加载这个 CrackMe。



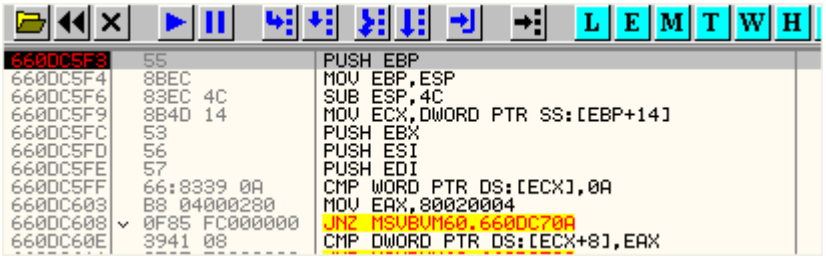
这个 CrackMe 很简单,首先会弹出一个 NAG 窗口(何谓 NAG:软件未注册或软件的试用版经常会弹出一些提示窗口要求注册,这些窗口被称为 NAG 窗口),我们首先需要将该 NAG 窗口除去掉,然后找到正确的 Password 即可,好了,现在我们运行起来看看。



(PS:如果你的系统中,文字显示的是乱码的话,可以使用英文版的操作系统来运行该程序,即可显示正常的英文字符)

弹出了一个 NAG 窗口,该窗口跟我们 Windows API 函数 MessageBoxA 弹出的消息框很相似,VB 中对应的 API 函数为 rtcMsgBox,我们给该函数设置一个断点。

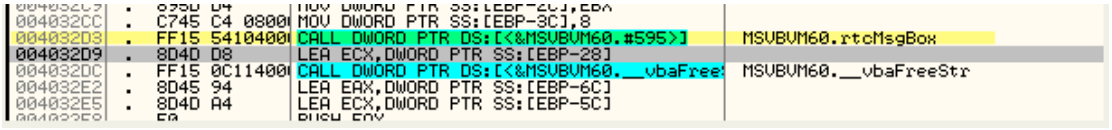
现在我们重新启动 OD 并运行起来,看看在弹出 NAG 窗口之前会不会断下来。



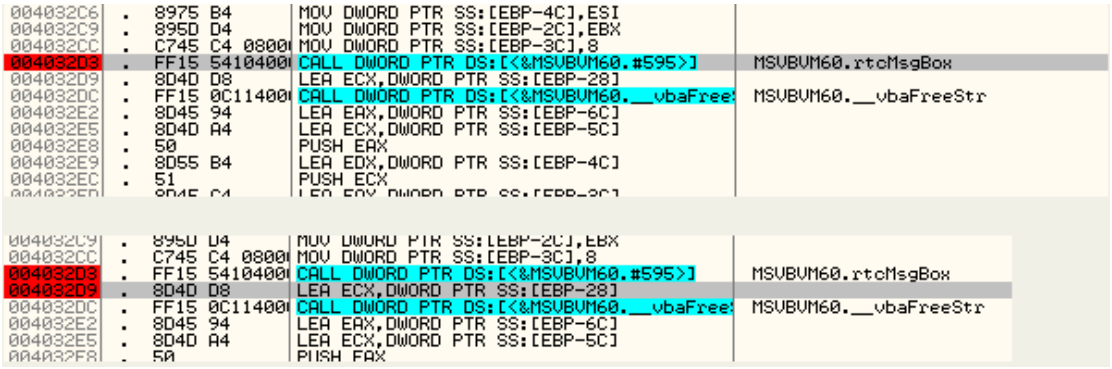
我们可以看到断了下来,但是并没有看到参数情况,也没有看到其他有用的信息,其实这无关紧要。

我们可以查看栈顶的值即函数的返回地址。

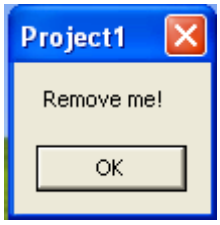
根据 OD 的提示信息我们可以看到当前断下来的这个 API 函数是 rtcMsgBox 以及在哪里调用的。



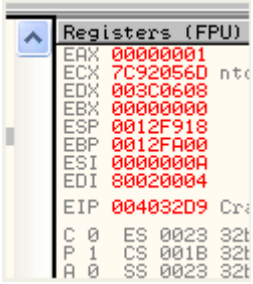
返回地址是 4032D9,其上一行就是调用 rtcMsgBox 这个 API 函数。



我们给 rtcMsgBox 调用处,以及其返回地址处都设置断点,然后运行起来。

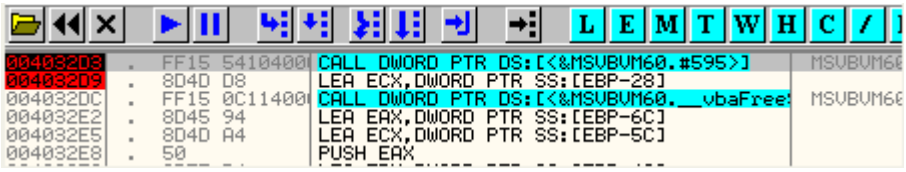


我们单击 OK,断在了返回地址处。

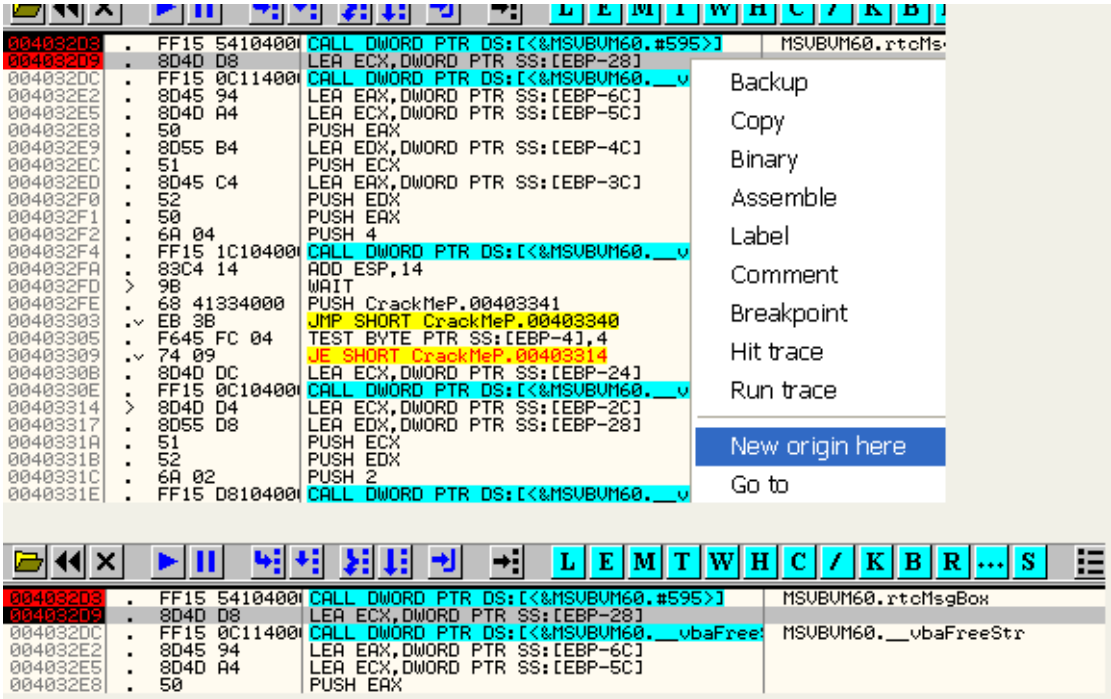


我们可以看到 EAX=1 表示该 API 函数执行成功了。现在,我们重启该 CrackMe,并尝试 NOP 掉弹出该 NAG 窗口的代码。

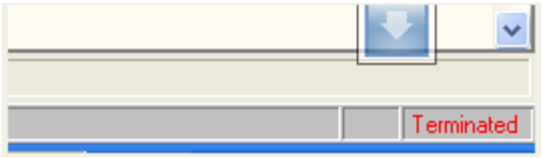
我们重启之后断在了 rtcMsgBox 这个 API 函数的调用处。



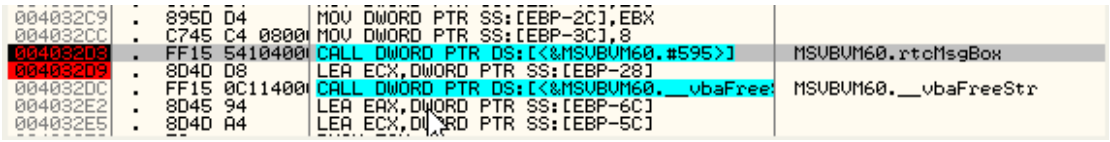
我们这里可以在下一行 4032D9 处单击鼠标右键选择-New origin here,这样就会从 4032D9 处开始执行,而上面的 rtlMsgBox 的调用代码并没有执行,这样我们就相当于模拟了 NOP 掉上一行代码的效果。



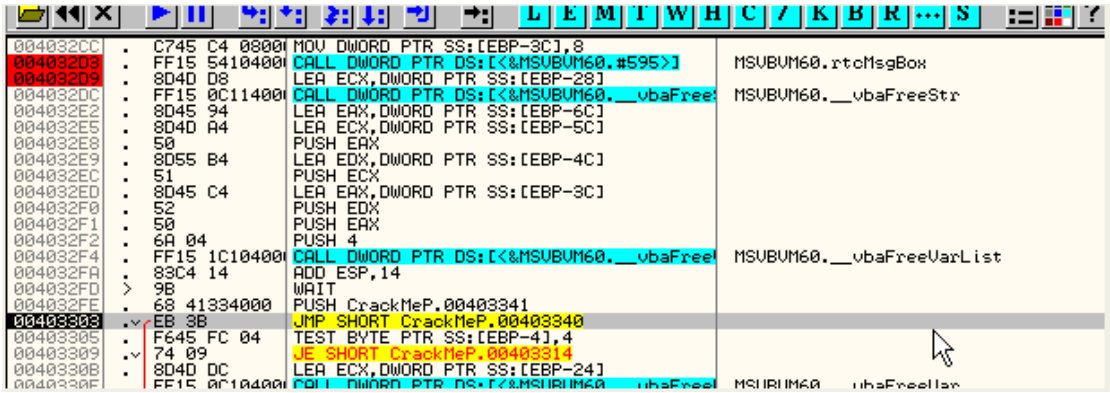
接着我们继续运行起来看看会发生什么。



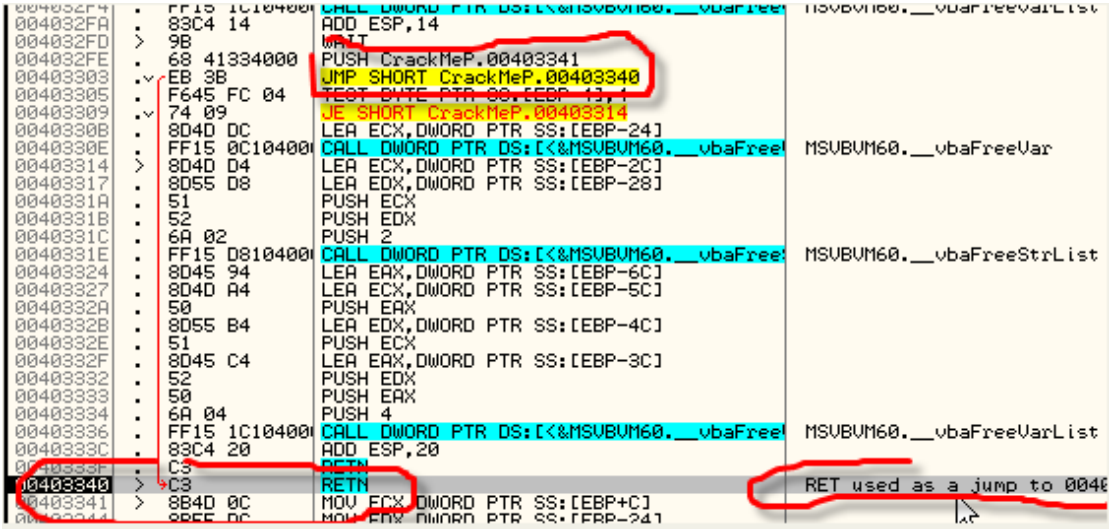
程序终止了,显示,NOP 掉 rtcMsgBox 的调用并没有出现我们想要的效果。我们再来尝试另一种方式。我们重启该 CrackMe。



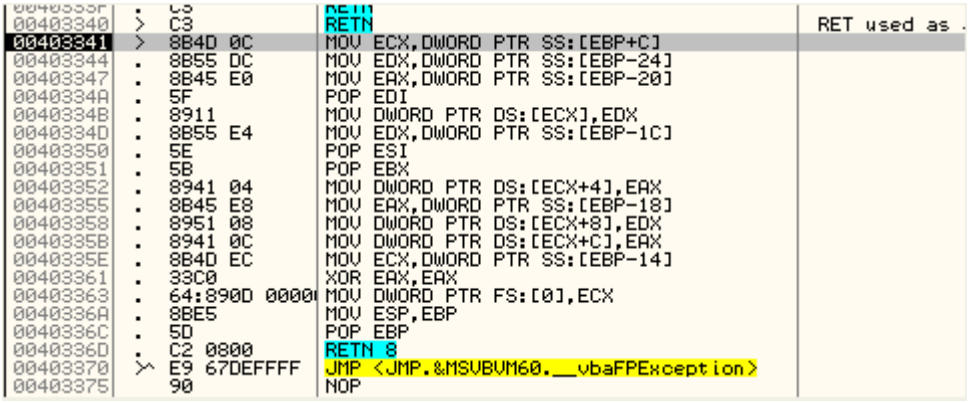
我们断在了 rtcMsgBox 的调用处,我们继续运行,弹出 NAG 窗口,我们单击 NAG 窗口中的 OK 按钮后,就返回到了 4032D9 处,我们接着继续按 F8 键单步跟踪。



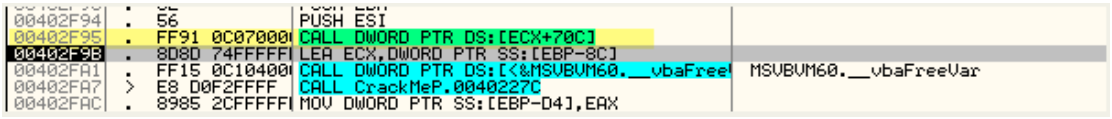
我们可以看到这里有个 JMP 指令,我们继续按 F8 键跟踪。



JMP 指令之前是 PUSH 403341,然后跳转到 RET 指令处,表示将返回到 403341 处,OllyDbg 将该处返回识别成了 RET used as a jump to 403341,这样就能够避免分析者注意到堆栈中的返回地址,干扰我们的视线,我们继续跟踪。

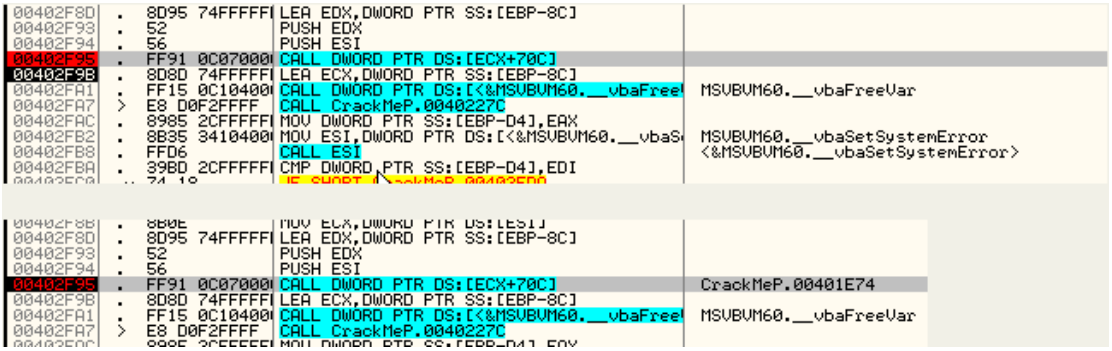


我们跟踪到 RETN 8 处。

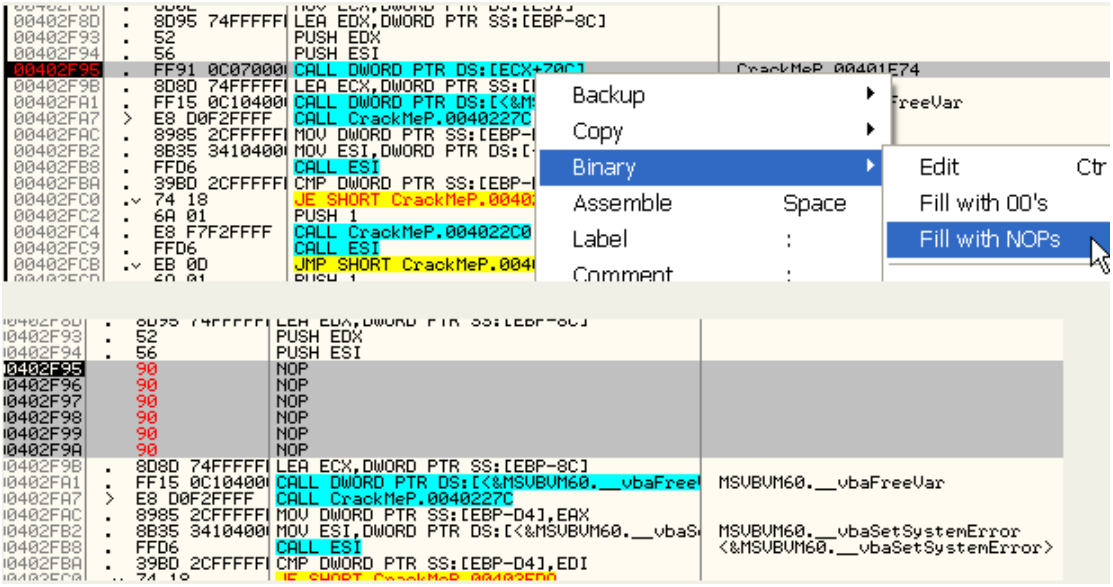


我们返回到了上层调用的返回地址 402F9B 处,但是刚刚堆栈中并没有提示调用来至于哪里以及返回地址是哪里等等信息。

现在我们给 402F95 处的 CALL 指令设置一个断点,然后重启该 CrackMe。



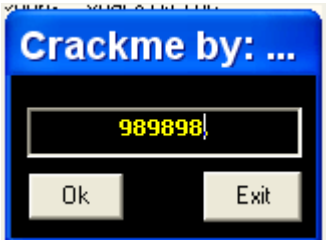
我们断在了 402F95 处的 CALL 指令处,现在我们 NOP 掉这个 CALL,然后运行起来,看看还会不会弹出烦人的 NAG 窗口。



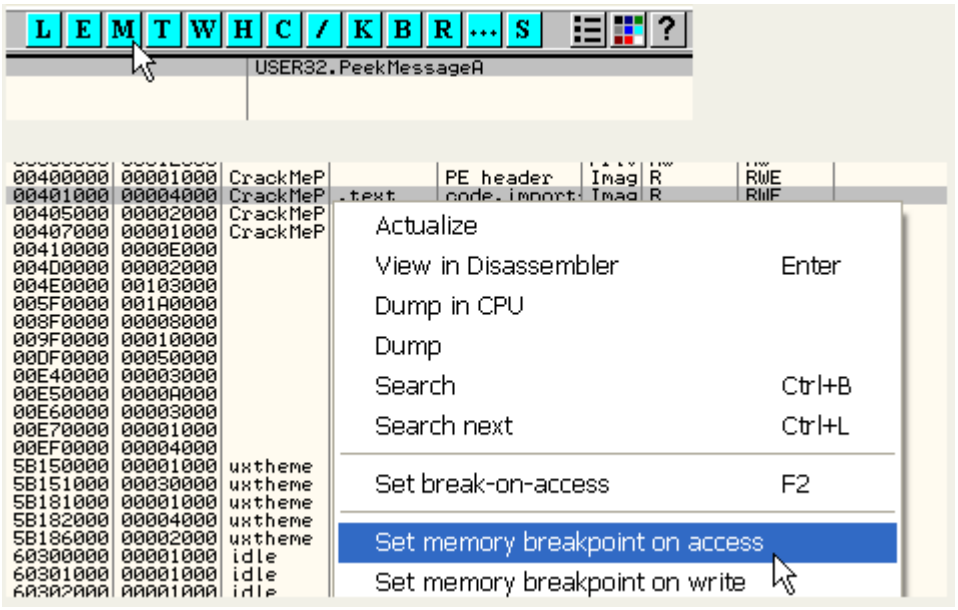
我们运行起来。

Good Bye,NAG 窗口,嘿嘿,我们以后还会看到更多去除 NAG 窗口的例子,这个例子虽然简单,但是我们也学习到了其在上面设的一些小把戏。

好了,我们接着来找正确的 Password。

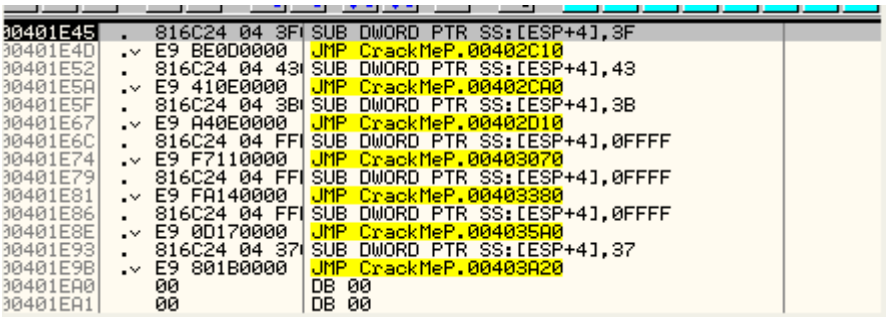


现在这个 Patch 过的 OllyDbg 就可以派上用场了。我们来试试给代码段设置内存访问断点(仅仅执行断点生效)。我们单击工具栏中的 M 按钮打开内存窗口。

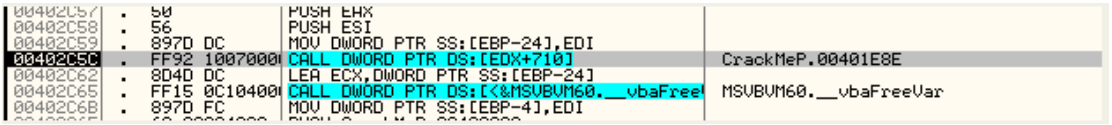


我们选中代码段所在的区段,单击鼠标右键选择-Set memory breakpoint on access。

接着单击 OK 按钮。

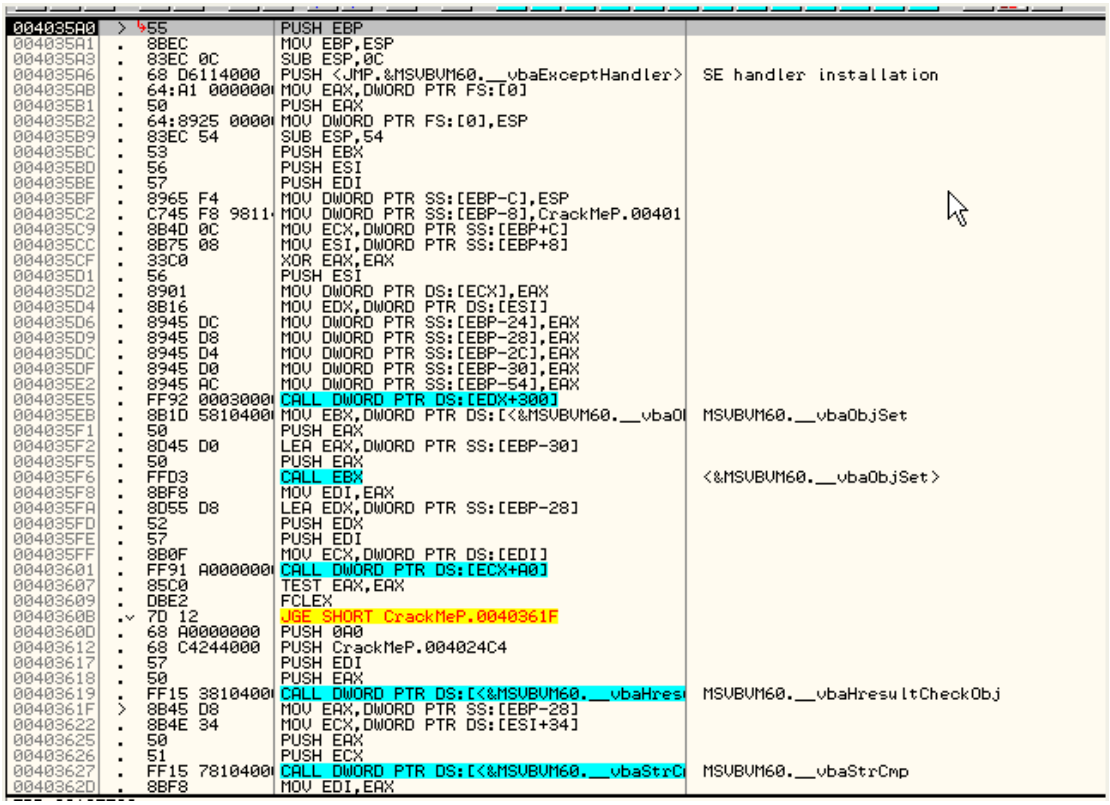


我们断在了 401E45 处,往下跟几步。

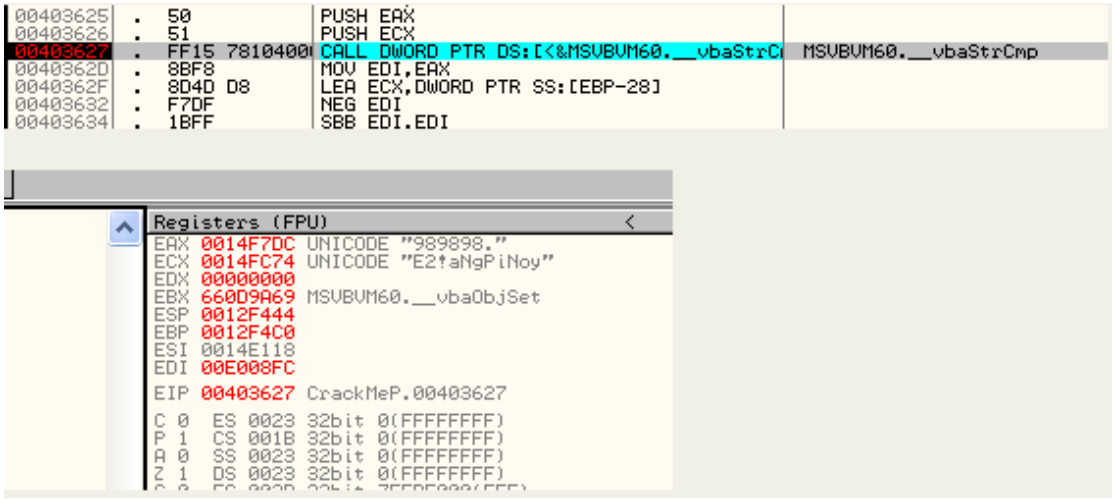


我们可以看到 402C5C 处有一个 CALL 指令,我们可以跟进这个 CALL 里面看看内部调用什么 API 函数。

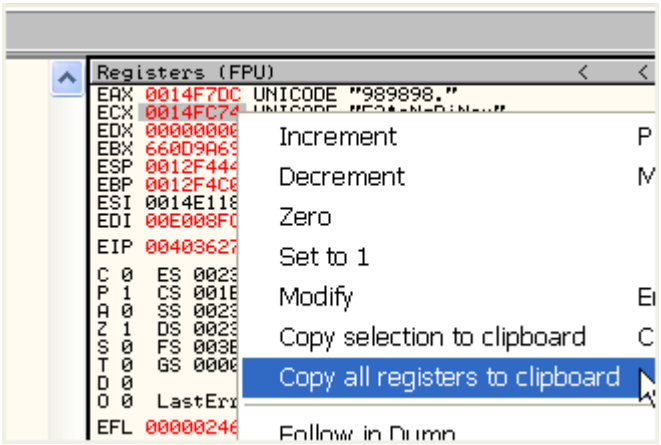
现在我们按 F7 键跟进。



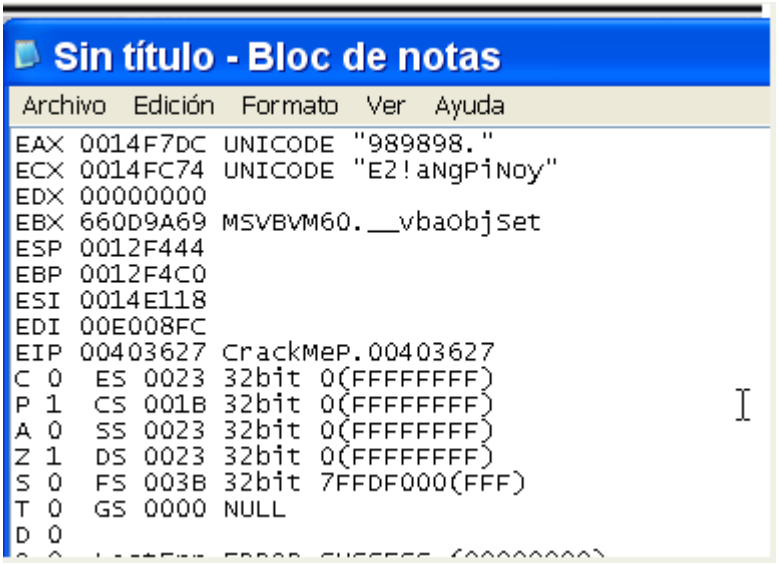
诶,我们好像达到了比较关键的地方,我们可以看到下面有一个__vbaStrCmp,这个 API 函数是用于比较两个字符串的,这里我们给__vbaStrCmp 调用处设置一个断点,然后清除掉之前设置的内存访问断点,然后运行起来。



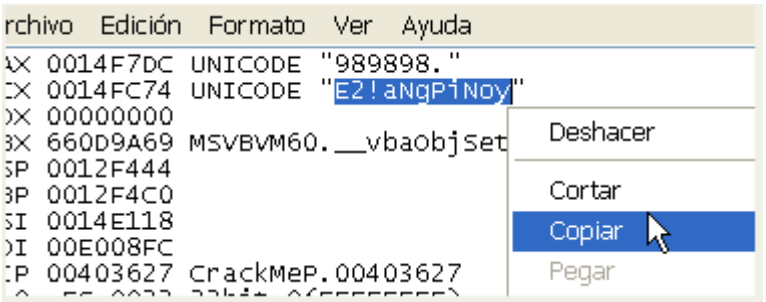
我们可以看到这里是将我们输入的字符串“989898”与正确的序列号进行比较,我们将其复制下来。



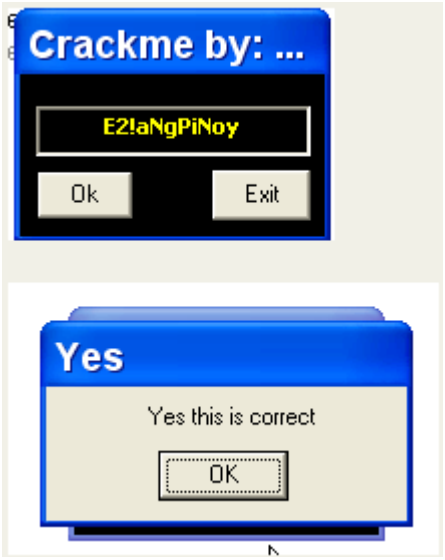
我们将剪切板上的内容粘贴到记事本中。



这样我们将可以很容易的将正确序列号复制下来了。



我们将正确的序列号粘贴到序列号的输入框中然后单击 OK。



我们可以看到提示序列号正确了,接下来我们需要将刚刚 NOP 掉的地方保存到文件中,这样 NAG 窗口就被永久的去除掉了。

下一章我们将继续介绍 VB 应用程序破解的相关话题,届时我们将接触到更加复杂的例子。