


## 第 8 章 Visual Basic 程序

 电子工业出版社  
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY



声明：本电子文档是《加密与解密(第三版)》的配套辅助电子教程！电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：(010) 88254396；(010) 88258888

传 真：(010) 88254397

E-mail: dbqq@phei.com.cn

通信地址：北京市万寿路 173 信箱

电子工业出版社总编办公室

邮 编：100036

说明：本篇文档是对《加密与解密》第三版“第 8 章 Visual Basic 程序”补充，为了降低书的价格，直接以电子文档提供给图书购买者。

**看雪软件安全网站**

<http://www.pediy.com>  
kanxue

2008-6-1

## 8.5 WKTVBDebugger调试工具

如果用 W32Dasm 反汇编一下光盘提供的 vbpcode.exe 程序，将会看到莫名其妙的代码，因为它不再是传统意义上的汇编代码了，只有用 P-code 反编译器才能得到正确的代码。VB5/VB6 的 P-code 反编译器现在有 Exdec，WKTVBDE，VBDE 等。SmartCheck 虽自称不支持 P-code 程序，但也可作为辅助调试工具。

WKTVBDE 是一款建立在 Exdec 基础上的动态调试工具，本节以此工具为例介绍如何调试 P-code 的程序。

### 1. 安装

执行 WKTVBDE 主文件就能完成安装。但还有几个方面要注意一下。

- 将目标软件复制到 WKTVBDebugger 安装目录里调试，即与 Loader.exe 同一目录；
- 将安装目录的 WKTVBDE.dll 文件复制到系统目录里；
- 将 MSVBVM60.DLL 替换成 2003 年以前版本。
- 如果装载还是失败，请试着关闭一些应用程序，如 VB 编译器、Word 等程序。

### 2. 界面

运行 WKTVBDE 后，单击菜单“File/Open”打开 vbpcode.exe 程序，将出现图 8.7 所示的窗口。如果提示“Can NOT find a ENGINE section for the DLL.”之类的信息，请参考安装一节找原因。

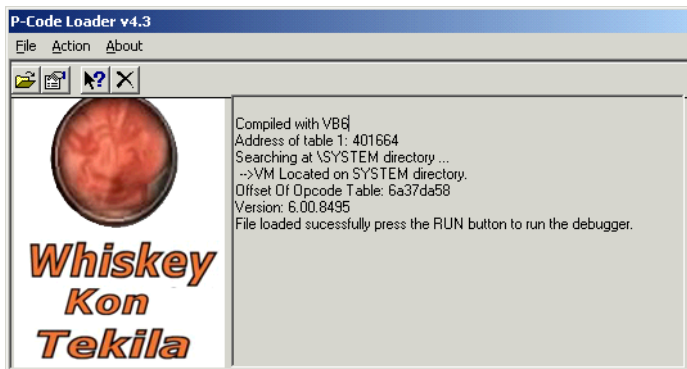


图 8.7 WKTVBDE打开程序后

单击菜单“Action/Run”装载程序。主窗口由 3 个基本部分组成：代码（Disassembly）、日志（Log）以及堆栈（Stack）窗口，如图 8.8 所示。

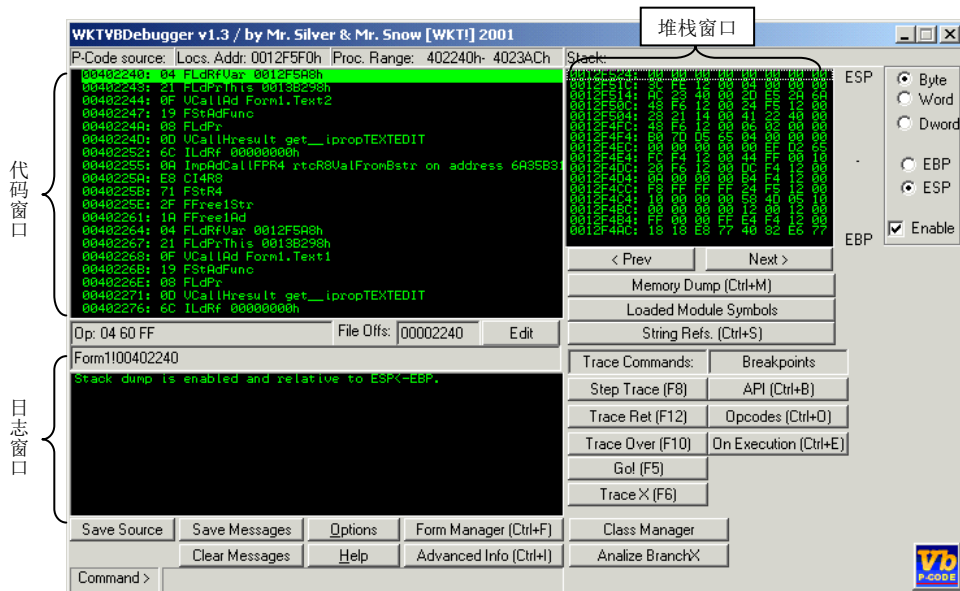


图 8.8 WKTVBDE 主窗口

### （1）代码窗口

这个窗口显示当前执行程序的反汇编伪代码。指令格式如下：

XXXXXXXX:XX 指令

其中“XXXXXXXX”是指令的内存虚拟地址，跟着的两个数字是以十六进制格式显示的机器码。注意，只有第一个机器码被显示了，后面的是 P-code 指令（会根据变量和自变量的不同而不同）。另外，在该窗口中单击鼠标右键将打开命令菜单。

### （2）日志窗口

当程序运行时，显示程序所有变量的信息和提示消息。所以当跟踪调试时，注意这个窗口是很重要的。它将提供一些有价值的信息，最重要的是，它将显示当前指令执行时所进行的操作。

### (3) 堆栈窗口

堆栈窗口有几个模式：字节、字或双字。通过选择单选框 EBP 和 ESP，以不同形式指向堆栈，两个导航按钮：Prev 和 Next 可以翻页。P-code 运行有别于传统的 CPU 结构，传统的 CPU 执行依赖于寄存器和堆栈，而 P-code 只使用堆栈，所以堆栈窗口非常重要，各种指令都通过堆栈来交换数据。

### (4) 中部状态栏

中部状态栏提供了非常有用的信息（见图 8.9）。



图 8.9 中部状态栏

- Op: 显示当前指令的完整机器码；
- File Offs: 显示当前指令的文件偏移地址；
- 最下面的一个文本框：显示的是“Form1!00402240”，表示当前指令的窗体或模块的名称；
- Edit 按钮：是内存编辑器的快捷按钮，单击该按钮将打开内存编辑器，它会从当前指令地址处显示内存数据。

### (5) 底部按钮栏

底部的按钮工具栏提供如下几个操作命令。

- Save Source: 保存代码窗口的内容；
- Save Messages: 保存日志窗口的内容；
- Clear Messages: 清除日志窗口的记录；
- Help: 打开帮助文件；
- Form Manager: 打开窗体管理器；
- Advanced Info: 显示关于 P-code 文件的高级信息、模块、类和窗体。

### (6) 侧面的按钮

侧面的按钮里都是一些与调试器有关的功能按钮。

- Memory Dump (Ctrl+M): 打开内存编辑器；
- Loaded Module Symbols: 显示符号信息对话框；
- String.Refs(Ctrl+S): 显示串式参考；
- Step Trace (F8): 相当于 SoftICE 的 T 命令，可以跟入 CALL 里；
- Trace Ret (F12): 相当于 SoftICE 的 PRET 命令；
- Trace Over (F10): 单步跟踪，相当于 SoftICE 中的 P 命令；
- Trace X (F6): 执行指定行数的指令；
- API (CTRL+B): 显示“Exported VM Functions”窗口；
- Opcode (Ctrl+O): 显示“OpCodes Control Dialog”窗口；
- On Execution (Ctrl+E): 显示“Current BPX List”窗口。

### (7) 内存编辑器

按 Ctrl+M 键打开内存编辑器（见图 8.10）。左窗口显示内存数据，右窗口显示对应的虚

拟机代码。

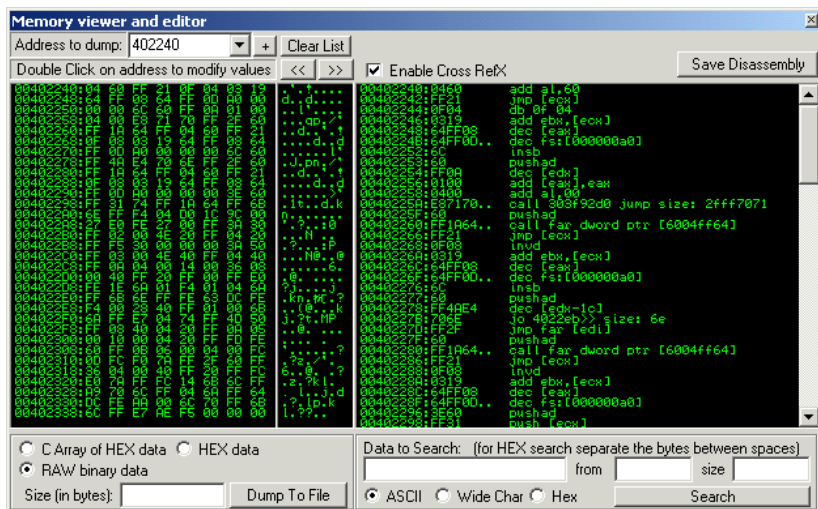


图 8.10 内存编辑器

- **Address to Jump:** 输入被显示的内存地址。按钮 **Clear List** 清除地址列表。按钮 “<<” 和 “>>” 允许在内存块中翻页（256 个字节）。
- **Dump to File:** 将指定大小的数据抓取到文件。
- **Search:** 允许在一个指定的内存范围里搜索数据字段，可以是十六进制数据、ASCII 或 WideChar 格式。

#### (8) 符号信息查看器

单击按钮 “Loaded Module Symbols” 打开符号信息查看器（见图 8.11），该对话框显示了程序调用的不同模块。

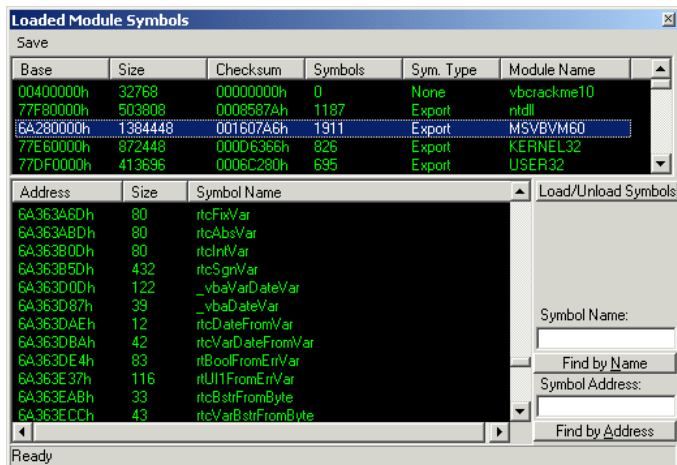



图 8.11 符号信息查看器

支持的符号调试文件有 COFF, PDB 等。典型的就是 DBG 文件，需要时将相关的符号文件复制到 WKTVBDE 的 DBG 子目录里。在任何时候单击 Load/Unload，符号就能够被加载/卸载，一旦加载了就没必要重新加载。记住，加载符号可能会是一个很漫长的过程，所以

要等到状态栏显示为 Ready 状态。

 **注意：**在光盘提供的 WKTVBDE 版本中，按钮“Find By Address”不能正确工作。

### (8) 串式参考

按 Ctrl+S 键打开串式参考窗口（见图 8.12）。该对话框显示了当前进程中的所有字符串。单击“Save”按钮可以保存全部的字符串。单击字符串，可以查看相应字符串在内存中的地址。使用内存编辑器也可以编辑字符串。

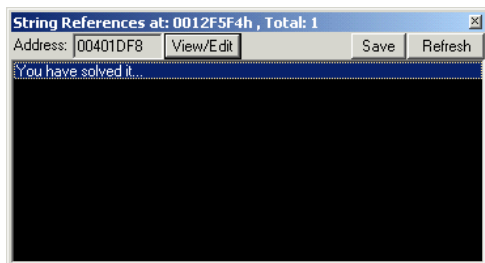


图 8.12 串式参考窗口

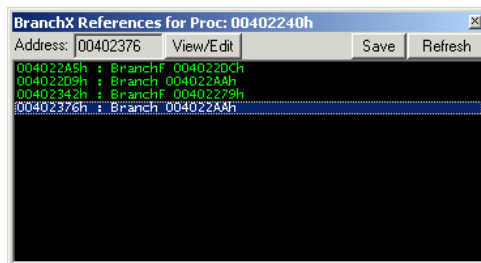


图 8.13 交叉参考窗口

### (9) 交叉参考窗口

单击“Analyze BranchX”按钮打开交叉参考窗口（见图 8.13）。该窗口显示当前进程的所有跳转的位置，可根据情况设置断点。可以在任何跳转上快速地设定断点，在要设定断点的跳转上单击鼠标右键，在弹出的菜单里选择“enable/disable the breakpoints”命令来设定或取消断点。

## 3. 设置断点

在代码窗口里可以直接双击鼠标或用鼠标右键设置断点。

### (1) 在虚拟机 API 函数呼叫上设断点

按 Ctrl+B 键打开“Breakpoints on VM API calls”窗口（见图 8.14）。

该对话框显示了所有通过虚拟机 DLL 被导出的函数列表。BP 列指出了程序地址，Ordinal 列出被导出的函数序号，Name 指出函数名。在地址旁边有个以不同颜色标出的图标，用不同颜色表示断点的状态。绿色表示有效断点，黄色表示被禁止的断点，灰色表示无效的断点。双击函数名可改变断点的状态。

### (2) 设置伪指令断点

按 Ctrl+O 键打开“Opcodes Control Dialog”窗口（见图 8.15）。该对话框显示所有助记命令列表，可以对机器码与助记命令设置断点。

- Address: 指出了控制操作指令的地址；
- Opcode: 机器码；
- Mnemoni: 助记命令，非常重要；
- Size: 指出了命令代码长度。

在地址旁边有一个以不同颜色标出的圆形图标，用于显示断点的不同状态。绿色的是被激活的，黄色的是被禁止的，灰色的是失效的。改变状态只需双击函数名。

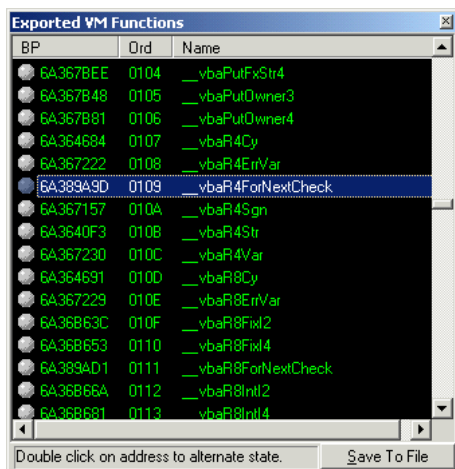


图 8.14 在虚拟机 API 函数呼叫上设断点

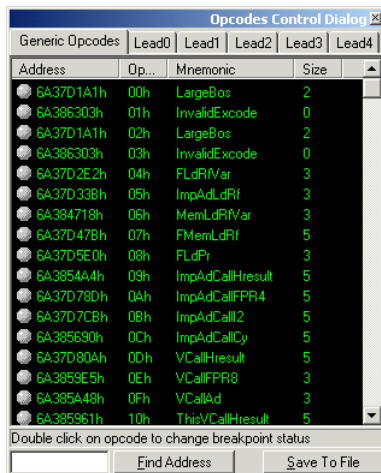


图 8.15 设置 Opcode 断点

#### 4. 管理窗体、类和模块

##### (1) 类管理

单击“Class Manager”打开 Class Manager 窗口（见图 8.16）。该对话框允许在列表中对选中的类访问。使用很简单，选择一个类，如果里面有对象，则相应的按钮将会被激活，单击按钮就可以访问，而对象属性对话框就会显示。



图 8.16 类管理

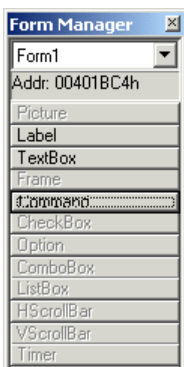


图 8.17 窗体管理



图 8.18 对象属性

##### (2) 窗体管理

按“Ctrl+F”键打开 Form Manager 窗口（见图 8.17）。操作与对象属性类似。

##### (3) 对象属性（Object Properties）

显示窗体或类的相关信息。在调试程序时，该窗口非常重要，可根据需要设置断点。图 8.18 显示的是“确定”按钮的信息，单击“BFX”按钮可以对此按钮设置断点。

- UUID: 128 位类型识别控制；
- Addr: 对象起始地址；
- Table: 对象列表起始数据地址；
- Ofcs: 列表附加数据偏移量；
- Proc: 对象控制进程地址范围；

- Index: 窗体对象索引;
- BPX: 允许直接在对象上设断点。

## 5. 机器码与助记命令

机器码与助记命令表 (Opcode and Mnemonics Table) 很重要, 具体参考其自带的帮助文件。掌握 VB P-code 的关键就在这些助记命令上。这里只列出几个常用的助记命令。

- BranchF: 机器码是 1Ch, 3 个字节。条件跳转指令, 如堆栈的值是 0 就跳转。单击 “Analyze BranchX” 按钮可以了解当前进程中的所有条件跳转指令的位置。
- BranchT: 机器码是 1Dh, 3 个字节。条件跳转指令, 如堆栈的值是 FFFFFFFh (-1) 就跳转。
- Branch: 机器码是 1Eh, 3 个字节。无条件转移。
- EqVarBool: 机器码是 33h, 1 个字节。比较指令, 比较两个变量, 根据结果将 -1 或 0 压进堆栈。可以单击 “Opcodes” 在此伪指令处设置断点。
- Lit2\_Byte: 机器码是 F4h, 2 个字节, 将数据压入堆栈。
- ConcatStr: 机器码是 2Ah, 1 字节。字符串联接指令 (相当于 C 语言中的 strcat 函数)。此指令单步跟踪 (Step Trace) 时, 会在日志窗口中留下相应字符串联接结果。所以, 可以单击 “Opcodes” 在此伪指令处设置断点, 以便了解某字符串值。
- FLdZeroAd / CVarStr: 取字符串指令, 特点同 ConcatStr。

## 6. 理解P-code语句

实例 Vbpcode.exe 程序采用 P-code 编译, 源码如下:

```
Private Sub Check_Click()  
    Dim sn As Long, x As Long  
    Dim t As Integer, i As Integer  
    sn = Val(Text2.Text)  
    t = Len(Text1.Text)  
    x = 0  
    If t < 4 Then  
        MsgBox "姓名要 4 个字符以上!", 48, "提示"  
    Else  
        For i = 1 To t  
            x = Asc(Mid(Text1.Text, i, 1)) + x  
        Next i  
        If sn - x = 0 Then  
            MsgBox "成功。", 64, "恭喜"  
        Else  
            MsgBox "不成功。", 16, "对不起"  
        End If  
    End If  
End Sub
```



End Sub

装载 vbpcode.exe 后，单击“确定”按钮就能中断在 WKTVBDE 里。也可先设置断点，按“Ctrl+F”键打开 Form Manager 窗口（见图 8.17）。在下拉菜单里选择窗体“Form1”，单击“command”按钮，打开对象属性对话框（见图 8.18）。选择“Check”，单击“BPX”设置断点，也就是对“确定”按钮事件设置了断点。

输入姓名 pedy 和序列号 1234。单击“确定”按钮将中断在 WKTVBDE 里，代码如下：

```

00402240: 04 FLdRfVar 0012F5B0h          ; 一开始中断在这，然后按 F10 键单步执行
00402243: 21 FLdPrThis 0013AF28h
00402244: 0F VCallAd Form1.Text2
.....
00402276: 6C ILdRf 00000000h
00402279: 4A FnLenStr                          ; 得到字符串长度
0040227A: E4 CI2I4
.....
0040228C: 6B FLdI2                             ; 将字符串长度入栈
0040228F: F4 LitI2_Byte: -> 4h 4              ; 将数字 4h 压进堆栈（机器码是: F4 04）
00402291: D0 LtI2 4h,5h ?                     ; 字符串长度>4
00402292: 1C BranchF 004022C9 ?               ; 跳转指令（堆栈数据是 0 就跳转），跳转时按 F8 键
.....
004022C9: F4 LitI2_Byte: -> 1h 1              ; 数据 1 入栈
004022CB: 04 FLdRfVar 0012F5B8h               ; 局部变量 0012F5B8h 入栈，i
004022CE: 6B FLdI2 FFFF0005h
004022D1: FE Lead3/ForI2:                     ; 0012F5B8h 指向的数据赋值 1，即 i=1
004022D7: 04 FLdRfVar 0012F5B0h
004022DA: 21 FLdPrThis 0013AF28h
004022DB: 0F VCallAd Form1.Text1
004022DE: 19 FStAdFunc
004022E1: 08 FLdPr
004022E4: 0D VCallHresult get__ipropTEXTEDIT
004022E9: 28 LitVarI2 1h , 1
004022EE: 6B FLdI2                             ; 将 i 取出并压入堆栈
004022F1: E7 CI4UI1
004022F2: 3E FLdZeroAd                        ; 字符 pedy 的指针入栈
004022F5: 46 CVarStr
004022F8: 04 FLdRfVar 0012F550h
004022FB: 0A ImpAdCallFPR4 rtcMidCharVar     ; Mid(Text1.Text, i, 1)
00402300: 04 FLdRfVar 0012F550h
00402303: FD Lead2/CStrVarVal                ; 将取出的字符 Name(i)入栈

```

```

00402307: 0B ImpAdCallI2 rtcAnsiValueBstr
0040230C: E7 CI4UII
0040230D: 6C ILdRf ; x 入栈
00402310: AA AddI4 ; x = Name(i) + x
00402311: 71 FStR4
00402314: 2F FFree1Str
00402317: 1A FFree1Ad
0040231A: 36 FFreeVar -> 3
00402323: 04 FLdRfVar 0012F5B8h ; 0012F5B8h 为计数器, 执行完下一句后加 1
00402326: 64 NextI2: jump to 004022D7 ; 循环, 按 F8 键进入
0040232B: 6C ILdRf 000004D2h ; 4D2h 是输入假序列号“1212”的 16 进制
0040232E: 6C ILdRf 0000021Bh ; 21Bh 是经计算的真序列号, 转成 10 进制是 539
00402331: AE SubI4 ; sn - x
00402332: F5 LitI4: -> 0h 0 ; 将数值 0 入栈
00402337: C7 EqI4 ; 将 0 与(sn - x)比较, 如不相等将 0 入栈
; 如相等将-1 入栈
00402338: 1C BranchF 0040236F ? ; 如相等就注册成功
0040233B: 27 LitVar_Missing 0012F530h
0040233E: 27 LitVar_Missing 0012F550h
00402341: 3A LitVarStr '恭喜'
00402346: 4E FStVarCopyObj 0012F570h

```

上述代码是讲述程序算法过程。如果想直接修改 P-code 程序, 该怎么办呢? 假设要修改 00402338 一句, 让其不相等就注册成功。

#### (1) 方法一

修改 BranchF 的上一条语句, 用 LitI2\_Byte 伪码, 将-1 压入堆栈, 机器码是: F4 FF。但此例中, EqI4 指令只有一个字节, 而 LitI2\_Byte -1 有两个字节, 因此不能采用。

#### (2) 方法二

将 BranchF 语句改成 BranchT (机器码是 1Dh), BranchT 跳转条件和 BranchF 相反, 如果堆栈是 0, 就不跳转。单击“EDIT”按钮打开内存编辑器, 在地址 402338 处双击, 打开数据修改框, 将 1C 改成 1D, 再单击“Patch Now”按钮修改程序。刚才只是修改了内存数据, 若想修改文件, 可以用 Hiew 来修改。光盘提供了该方法的动画教学。

P-code 程序并不难, 关键是要理解助记命令的含义。读者若感兴趣, 可以写些演示程序, 然后反编译分析。