

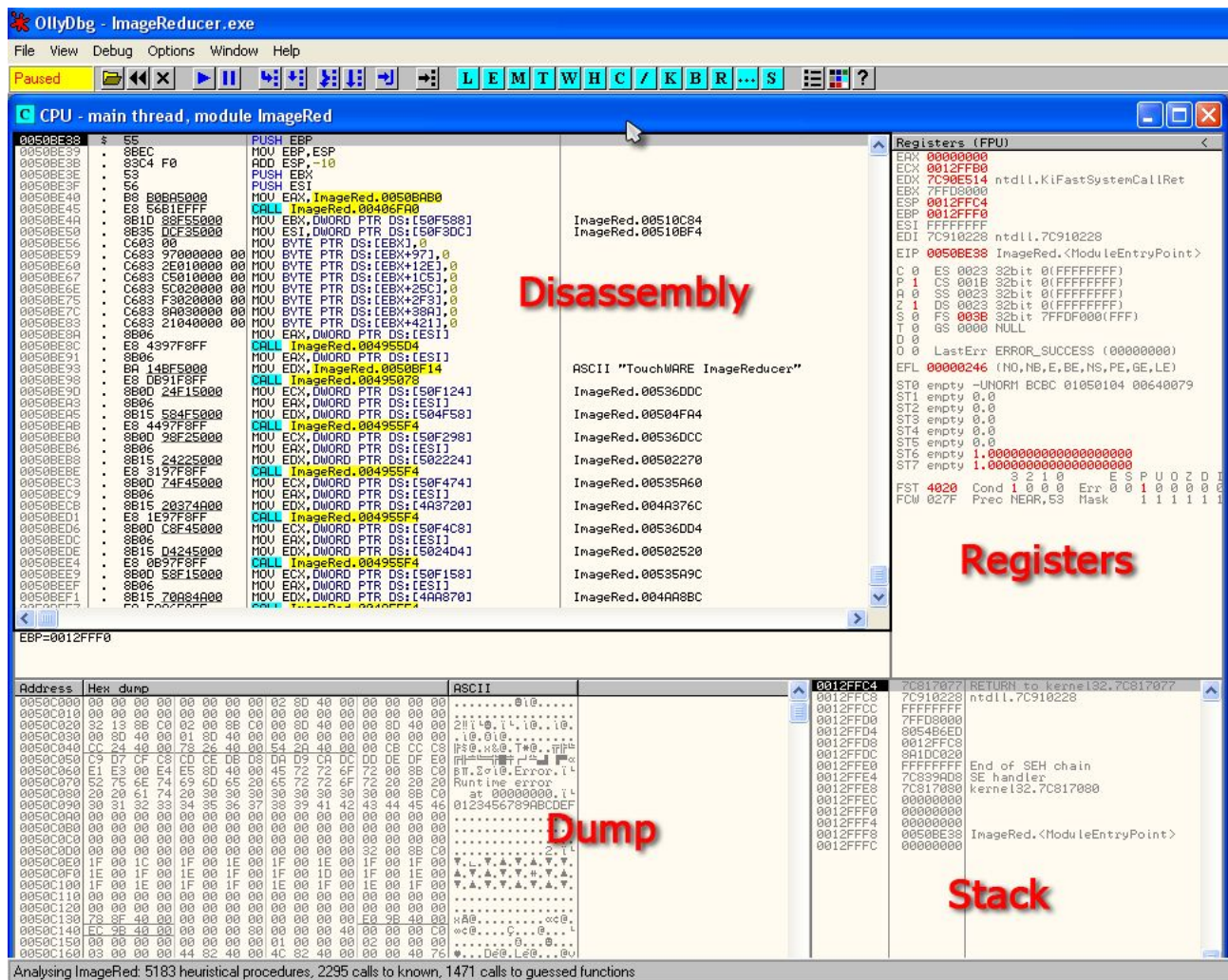
教程二、介绍 011y Debug

一、什么是 011y Debugger?

援引作者 Oleh Yuschuk 的话“011yDbg 是一个用于微软 Windows 的 32 位汇编级分析调试器”。在没有源代码的情况下，二进制代码分析非常有用。011y 也是一个动态调试器，意味着它允许用户在程序运行时修改一些东西。这在实际分析二进制文件尝试找出程序工作原理时非常的重要。011y 有许多许多很棒的特性，这就是为什么它是逆向工程领域的天字第一号调试器（至少在 Ring3 级是，我们马上就接触到了）。

二、概览

下面是 011y 的主界面图片，上面有一些说明性的标签。



打开 Olly 时有一个默认的子窗口是 CPU 窗口。这是那个“大图片”中大部分数据所在的地方，如果你什么时候把它关掉了，只需要点击工具栏中那个“C”图标就行了。窗口被分成了四个部分：反汇编区（Disassembly），寄存器区（Registers），堆栈区（Stack）以及内存数据区（Dump）。下面是对每个区的说明：

1、反汇编区

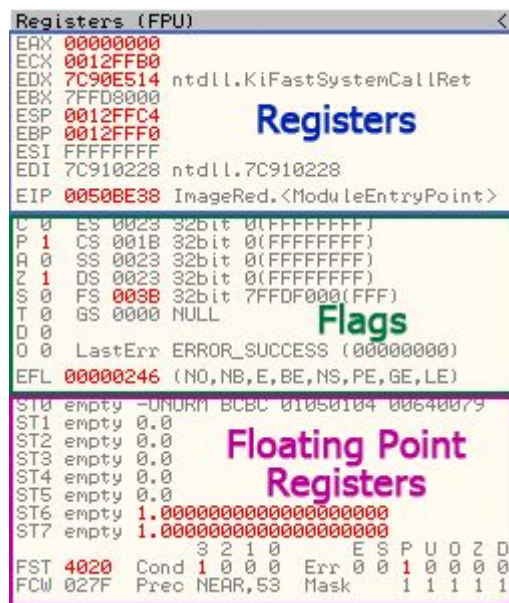
该部分主要包含了二进制文件的反汇编代码。这是 Olly 显示二进制信息的地方，包括操作码（opcode）和翻译的汇编代码。

第一列是指令的地址（内存中地址）。第二列按汇编语言叫操作码，每个指令至少对应一条代码（有很多对应多条）。这才是 CPU 真正需要并且是唯一能读懂的代码。这些操作码组成了“机器语言”，也就是计算机的语言。如果你看过二进制的原始数据（用十六进制编辑器），你除了看到这些操作码的字符串以外，就没有其他的了。Ollly 的一个主要工作是将这些“机器语言”“反汇编”成人类可读的汇编语言。第三列是汇编语言。不过退一步讲，对于不太懂汇编的人来说，汇编不比操作码好多少。不多随着学的越来越多，汇编提供了远多于代码所做的更多信息。

最后一列是 Ollly 对于该行代码的注释。有时候会包含所调用 API 的名字，比如 `CreateWindow` 和 `GetDlgItemX`。Ollly 也会尝试通过将非 API 调用命名来帮助我们理解代码，上图中的“`ImageRed.00510C84`”和“`ImageRed.00510BF4`”就是此类情况。退一步讲，这些东西不是那么有用，Ollly 也允许我们将它们修改成一个有意义的名字。你也可以在该列写自己的注释。只要双击该列中的某行，就会弹出一个对话框让你输入注释。这些注释会自动保存到下一次。

2、寄存器区

每个 CPU 都有一组寄存器。用来临时存放数值，和高级语言中的变量很像。下面是寄存器窗口的特写（有标记）：



顶部实际上是 CPU 的寄存器。如果值有变化，寄存器会从黑色变为红色（对于观察数值的变化真的非常有用）。你也可以双击任何一个寄存器来改变它的内容。这些寄存器能做很多事情，后面会讨论更多。

中间那块是标志寄存器，是 CPU 用来标记代码中一些事情的发生（两个数相等、一个数比另外一个大等等）。双击其中一个标志寄存器就可以修改它。这些玩意儿在我们的学习过程中扮演着重要的角色。

底下的部分是 FPU，或者叫浮点运算器。只要 CPU 执行任何涉及小数点的运算就会用到它们。逆向者很少用到它们，主要是在我们接触加密的时候用。

3、堆栈区

0012FFC4	7C817077	RETURN to kernel32.7C817077
0012FFC8	7C910228	ntdll.7C910228
0012FFCC	FFFFFFFF	
0012FFD0	7FFD8000	
0012FFD4	8054B6ED	
0012FFD8	0012FFC8	
0012FFDC	8A1DC020	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C839AD8	SE handler
0012FFE8	7C817080	kernel32.7C817080
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	0050BE38	ImageRed.<ModuleEntryPoint>
0012FFFC	00000000	

堆栈是内存中的一段区域，用于存储二进制数据的临时列表。这些数据包括指向内存中地址的指针，字符串，制造者（makers）及大部分重要的数据，还包括函数调用后的返回地址。当程序中的一个方法调用另一个方法时，控制权需要转移到新方法以便于它能够返回。CPU 必须知道一个新方法执行完后它是从哪被调用的，CPU 能够返回到它被调用的地方，继续执行该调用之后的代码。堆栈就是 CPU 保存返回地址的地方。

关于栈你需要知道一点，他是“先进后出”的数据结构。打个常用的比方，就像是自助餐厅里下面带有弹簧的一摞盘子一样。当你向顶部“压（PUSH）”进一个盘子，下面的所有盘子都会被往下压。当你移除（“POP”）顶部的一个盘子，下面的所有盘子都会被往上提升一级。下个教程我们会实际看看，所以这里别担心看不太懂。

图片中，第一列是每一个数据成员的地址，第二列是十六进

制的 32 位数据，如果 Olly 能够分析出来的话，那么最后一列是 Olly 关于数据项的注释。如果你注意看第一行的话，会看到“RETURN to kernel...”的注释。这里是 CPU 放在栈上的一个地址，以便于在当前的函数执行完后，CPU 知道返回到哪。

在 Olly 中，你可以右键点击堆栈区，并且选择“修改(modify)”来更改内容。

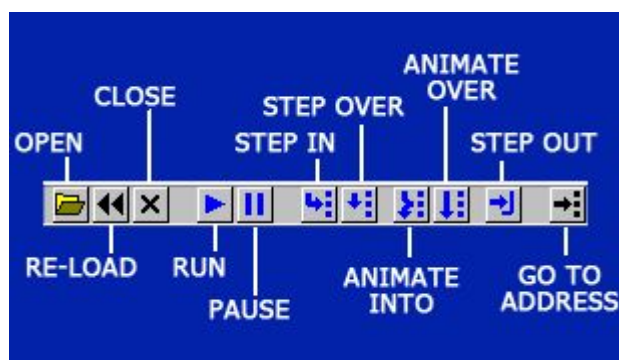
4、内存数据区

Address	Hex dump	ASCII
0050C000	00 00 00 00 00 00 00 00 02 80 40 00 00 00 00 000i@....
0050C010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050C020	32 13 88 C0 02 00 88 C0 00 80 40 00 00 80 40 00	2!!l@.l.l@..i@.
0050C030	00 80 40 00 01 80 40 00 00 00 00 00 00 00 00 00	..i@.0i@.....
0050C040	CC 24 40 00 78 26 40 00 54 2A 40 00 00 CB CC C8	ft@.x&@.T*@..ft@
0050C050	C9 D7 CF C8 CD CE D8 D8 DA D9 CA DC DD DE DF E0	ft@.x&@.T*@..ft@
0050C060	E1 E3 00 E4 E5 80 40 00 45 72 72 6F 72 00 8B C0	ft@.x&@.T*@..ft@
0050C070	52 75 6E 74 69 60 65 20 65 72 72 6F 72 20 20 20	ft@.x&@.T*@..ft@
0050C080	20 20 61 74 20 30 30 30 30 30 30 30 30 00 8B C0	ft@.x&@.T*@..ft@
0050C090	30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46	ft@.x&@.T*@..ft@
0050C0A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ft@.x&@.T*@..ft@
0050C0B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ft@.x&@.T*@..ft@
0050C0C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ft@.x&@.T*@..ft@
0050C0D0	00 00 00 00 00 00 00 00 00 00 00 00 32 00 8B C0	ft@.x&@.T*@..ft@
0050C0E0	1F 00 1C 00 1F 00 1E 00 1F 00 1E 00 1F 00 1F 00	ft@.x&@.T*@..ft@
0050C0F0	1E 00 1F 00 1E 00 1F 00 1F 00 1D 00 1F 00 1E 00	ft@.x&@.T*@..ft@
0050C100	1F 00 1E 00 1F 00 1F 00 1E 00 1F 00 1E 00 1F 00	ft@.x&@.T*@..ft@
0050C110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ft@.x&@.T*@..ft@
0050C120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ft@.x&@.T*@..ft@
0050C130	78 8F 40 00 00 00 00 00 00 00 00 00 E0 9B 40 00	ft@.x&@.T*@..ft@
0050C140	EC 9B 40 00 00 00 00 80 00 00 00 40 00 00 00 C0	ft@.x&@.T*@..ft@
0050C150	00 00 00 00 00 00 00 00 01 00 00 00 02 00 00 00	ft@.x&@.T*@..ft@
0050C160	03 00 00 00 44 82 40 00 4C 82 40 00 00 00 40 76	ft@.x&@.T*@..ft@

在教程的开始，当我们讨论 CPU 从二进制文件中读取的原生“操作码”时，我提到过你能在十六进制查看器中看到原始数据。不过，在 Olly 中你不需要这么做。因为内存数据区就是一个内置的十六进制查看器，以便于你查看原始的二进制数据，只查看内存中的而不是磁盘上的。通常对于同样的数据有两种查看方式，十六进制的和 ASCII 的。图片中右边的两列就是（第一列是数据驻留内存中的地址）。Olly 允许修改这些数据的显示方式，后面的教程就会看到。

三、工具栏

不幸的是，Ollly 的工具栏给大家留下了一点念想（尤其是当英语并不是作者的母语）。我将左边的工具栏图标进行了注释：



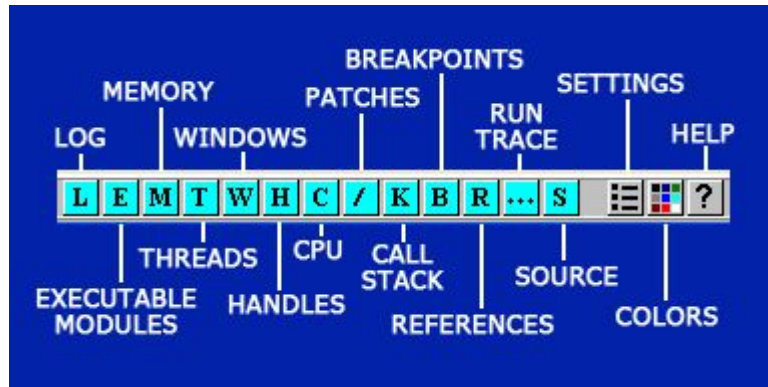
这些都是控制代码运行的主要工具。记住这些，尤其是你开始使用 Ollly 的时候，这些按钮的所有功能都可以从“调试 (Debug)”菜单的下拉菜单中访问到。如果你不知道某些东西是什么，你可以从菜单中看到。

关于一些图标我要多说几句。“Re-load”是用来重新启动应用并暂停在入口点处。所有的补丁（后面会看到）都会被删除，一些断点会失效，应用程序也不会运行任何代码。好吧，大部分情况下是这样的。“Run”和“Pause”做的就是你看到的那样。

“Step In”意思是运行一行代码然后暂停，如果有的话它会跟进函数的内部。“Step Over”做同样的事情，不过它会跳过对另一个函数的调用。“Animate”有点像 Step In 和 Step Over，不过它特别慢好让你观察。这个你用的不多，不过有时候看代码运行也挺有意思的，尤其是遇到多态二进制的时候能够观察到代

码的变化。讲的有点超前了.....

下面是各窗口的按钮图标（更加有点神秘）：



点击其中的任何一个按钮都会弹出一个窗口，有些你会经常用到，而有的却很少用。看这些字母并不是很直观，这点你可以像我学习，把它们都点一遍直到你找到你需要的那个。每一个都可以通过“View”菜单来访问，所以在第一次征程时你可以获得些许帮助。下面我会介绍最常用的窗口：

1、(M)emory——内存映射窗口

Memory map							
Address	Size	Owner	Section	Contains	Type	Access	Initial acc
00010000	00001000	showstri		stack of main thread	Priv 00021004	RW	RW
00020000	00001000				Priv 00021004	RW	RW
0012C000	00001000				Priv 00021104	RW Guarded	RW
0012D000	00003000				Priv 00021104	RW Guarded	RW
00130000	00003000				Map 00041002	R	R
00140000	00001000				Priv 00021040	RWE	RWE
00150000	00007000				Priv 00021004	RW	RW
00250000	00006000				Priv 00021004	RW	RW
00260000	00003000				Map 00041004	RW	RW
00270000	00016000				Map 00041002	R	R
00290000	00041000				Map 00041002	R	R
002E0000	00041000				Map 00041002	R	R
00330000	00006000				Map 00041002	R	R
00340000	00001000				Priv 00021004	RW	RW
00350000	00001000				Priv 00021004	RW	RW
00360000	00004000				Priv 00021004	RW	RW
00370000	00003000				Map 00041002	R	R
00380000	00002000				Map 00041002	R	R
00390000	00004000				Priv 00021004	RW	RW
003A0000	00002000				Map 00041002	R	R
003B0000	00002000				Map 00041002	R	R
003C0000	00001000				Priv 00021040	RWE	RWE
00400000	00001000	showstri		PE header	Imag 01001002	R	RWE
00401000	00005000	showstri	.text	code	Imag 01001002	R	RWE
00406000	00020000	showstri	.bss	code	Imag 01001002	R	RWE
00426000	00001000	showstri	.data	code,data	Imag 01001002	R	RWE
00427000	00001000	showstri	.idata	code,imports	Imag 01001002	R	RWE
00428000	00002000	showstri	.rsrc	code,resources	Imag 01001002	R	RWE
00430000	00003000	COMCTL32		PE header	Map 00041020	R E	R E
004F0000	00002000				Map 00041020	R E	R E
00500000	00103000				Map 00041002	R	R
00610000	00073000				Map 00041020	R E	R E
009EF000	00021000				Priv 00021104	RW Guarded	RW
5D090000	00001000				Imag 01001002	R	RWE
5D091000	00071000				Imag 01001002	R	RWE
5D102000	00003000				Imag 01001002	R	RWE
5D105000	00020000				Imag 01001002	R	RWE
5D125000	00005000				Imag 01001002	R	RWE
73D90000	00001000				Imag 01001002	R	RWE
73D91000	00010000				Imag 01001002	R	RWE
73DAE000	00006000				Imag 01001002	R	RWE
73DB4000	00001000				Imag 01001002	R	RWE
73DB5000	00002000				Imag 01001002	R	RWE
76390000	00001000				Imag 01001002	R	RWE
76391000	00015000				Imag 01001002	R	RWE
763A6000	00001000				Imag 01001002	R	RWE
763A7000	00005000				Imag 01001002	R	RWE
763AC000	00001000				Imag 01001002	R	RWE
763B0000	00001000				Imag 01001002	R	RWE
763B1000	00030000				Imag 01001002	R	RWE
763E1000	00004000				Imag 01001002	R	RWE
763E5000	00011000				Imag 01001002	R	RWE
763F6000	00003000				Imag 01001002	R	RWE
773D0000	00001000				Imag 01001002	R	RWE
773D1000	00091000				Imag 01001002	R	RWE
77462000	00001000				Imag 01001002	R	RWE
77463000	0006A000				Imag 01001002	R	RWE
774CD000	00006000				Imag 01001002	R	RWE
774E0000	00001000				Imag 01001002	R	RWE
774E1000	00120000				Imag 01001002	R	RWE
77601000	00006000				Imag 01001002	R	RWE
77607000	00007000				Imag 01001002	R	RWE
7760E000	00002000				Imag 01001002	R	RWE
77610000	0000E000				Imag 01001002	R	RWE
77C10000	00001000				Imag 01001002	R	RWE
77C11000	0004C000				Imag 01001002	R	RWE

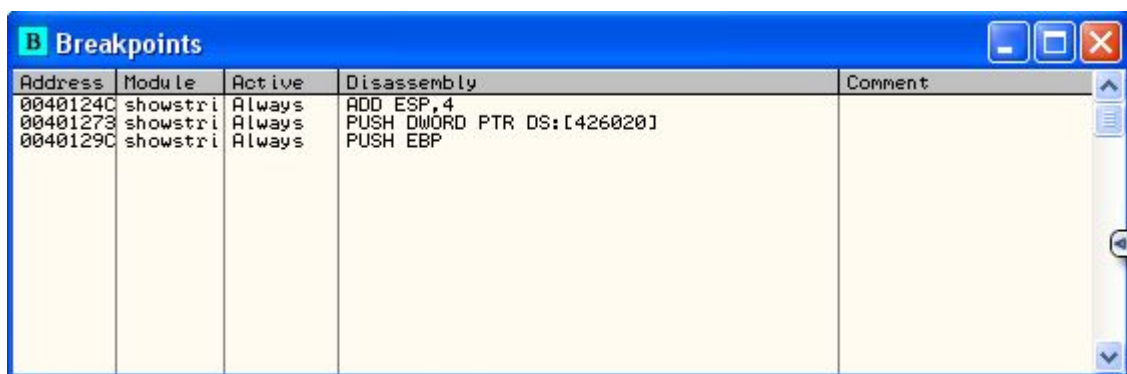
内存窗口显示程序已经分配的所有的内存块。它包括正在运行的程序的主段（本例中，是 Owner 列中的“Showstr”）。在下面你能看到很多其他的段，这些都是程序载入进内存的 DLL 的，准备将来用的。如果你双击其中的任何一行，都会打开一个显示

该段的反汇编代码（或十六进制数据）的窗口。这个窗口也显示了块的类型和访问权限、大小以及该段载入内存的地址。

2. (P)atches——补丁窗口

该窗口显示的是你做的任何“补丁”，即对原始代码的任何修改。注意那个状态（State 列）是激活的（Active）。如果你重新载入应用程序（通过点击 re-load 图标），这些补丁就会失效。为了简便的使它们重新生效（或失效），点击期望的补丁以及敲击空格键。这可以打开或关闭补丁。注意那个“Old”和“New”列，显示的是原始的指令和修改后的指令。

3. (B)reakpoints——断点窗口



Address	Module	Active	Disassembly	Comment
0040124C	showstri	Always	ADD ESP, 4	
00401273	showstri	Always	PUSH DWORD PTR DS:[426020]	
0040129C	showstri	Always	PUSH EBP	

该窗口显示了当前所有断点设置的位置。这个窗口将会是你的好朋友

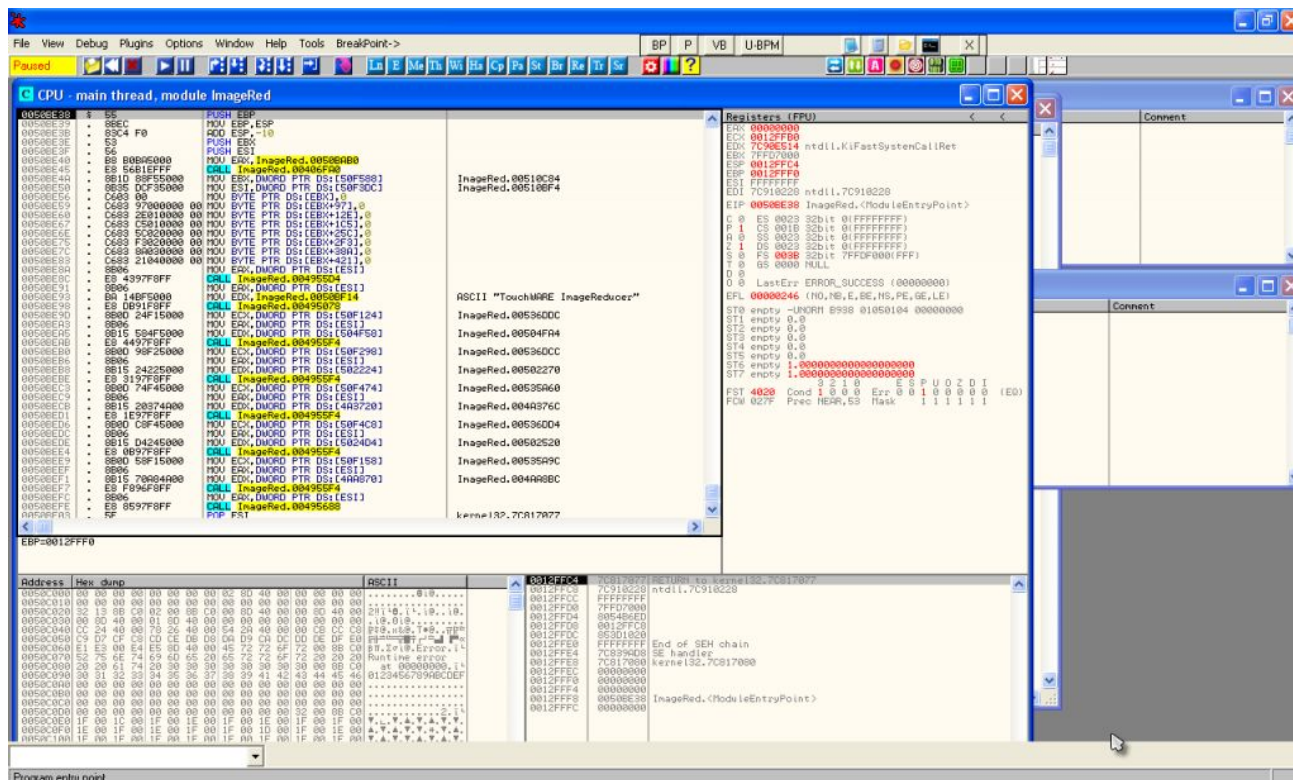
4. (K)all Stack——调用栈窗口

（哎呀，我知道为什么初学者记这些图标比较难了……）

K Call stack of main thread				
Address	Stack	Procedure / arguments	Called from	Frame
0012FEB0	7E4191BE	Includes ntdll.KiFastSystemCallRet	USER32.7E4191BC	0012FEE0
0012FEC0	7E42776B	USER32.7E4191B2	USER32.7E427766	0012FEE0
0012FEE4	00401395	<JMP.&USER32.GetMessageA>	showstri.00401390	0012FEE0
0012FEE8	0012FF30	pMsg = 0012FF30		
0012FEEC	00000000	hWnd = NULL		
0012FEF0	00000000	MsgFilterMin = 0		
0012FEF4	00000000	MsgFilterMax = 0		
0012FF50	00404EF5	showstri.0040129C	showstri.00404EF0	0012FF4C
0012FF54	00400000	Arg1 = 00400000		
0012FF58	00000000	Arg2 = 00000000		
0012FF5C	00151F35	Arg3 = 00151F35		
0012FF60	00000001	Arg4 = 00000001		
0012FF70	00401284	showstri.00404E94	showstri.0040127F	0012FF6C
0012FFC4	7C817077	Maybe showstri.0040126C	kernel32.7C817074	0012FFC0

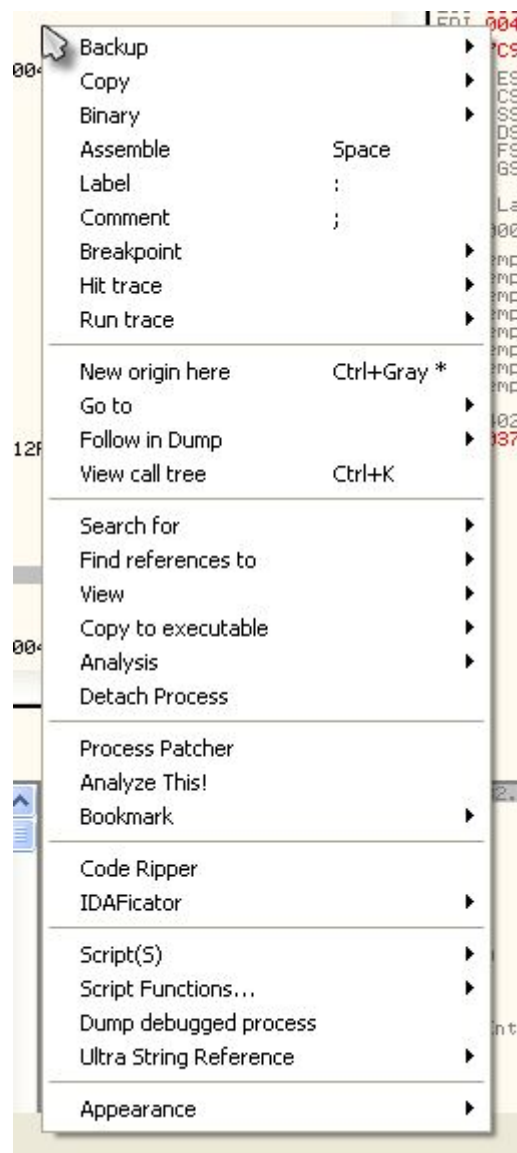
这个窗口与前面看到的“堆栈区”不一样，它显示了更多信息，有关于代码中的调用、发送给这些函数的值以及其他的东西。不久我们会了解到更多。

下一教程，我会包含我的经过“升级”的 Olly 版本，有些是你一看就明白的按钮。这里有张图片



四、上下文菜单

本教程的最后，我会快速介绍 OllyDbg 的右键菜单。它是许多操作产生的地方，所以你最少应该熟悉一下它。右键反汇编区的任何地方都会调出该菜单：



我只会介绍最常用的几项。随着经验的增多，你最终会遇到那些较少用到的选项。“Binary”菜单项允许你按字节编辑二进制数据。在这里你可以将埋在一堆二进制数据中的“未注册”几个字改成“已注册”。“Breakpoint”菜单可以设置断点。断点分好几种，下一章我会讲到。“Search for”有一个相当大的子菜单。这里你可以搜索类似字符串、函数调用等二进制数据。

“Analysis”菜单会强制 Olly 重新分析当前正在查看的代码段。有时候 Olly 会对你正在查看的是代码还是数据感到困惑(记住,它们俩都只是一些数字),这个可以强制 Olly 将你正在看的内容当做是代码,并且尝试猜测该部分看起来应该是什么样子的。

注意,我的菜单看起来和你的可能不太一样,因为我装了一些插件,这些插件在菜单中添加了一些功能。不过别担心,后面的教程中我会介绍这些菜单的。