第3章

软件调试基础

学习要求：理解 PE 文件格式，了解软件加壳与脱壳的思想；理解虚拟内存的概念；掌握 PE 导入表和 IAT 的概念；学会使用 OllyDbg 和 IDA Pro 等工具，能掌握其基本用法；了解 PE 文件代码注入实验的原理。

课时：2 课时。

分布：[二进制文件][调试分析工具—演示示例]。

3.1 二进制文件

3.1.1 PE 文件格式

源代码通过编译和连接后形成可执行文件。可执行文件之所以可以被操作系统加载且运行，是因为它们遵循相同的规范。

PE(Portable Executable)是 Win32 平台下可执行文件遵守的数据格式。常见的可执行文件(如 *.exe 文件和 *.dll 文件)都是典型的 PE 文件。

一个可执行文件不光包含了二进制的机器码，还会自带许多其他信息，如字符串、菜单、图标、位图、字体等。PE 文件格式规定了所有的这些信息在可执行文件中如何组织。在程序被执行时，操作系统会按照 PE 文件格式的约定去相应的地方准确地定位各种类型的资源，并分别装入内存的不同区域。如果没有这种通用的文件格式约定，可执行文件装入内存将会变成一件非常困难的事情。

PE 文件格式把可执行文件分成若干数据节(Section)，不同的资源被存放在不同的节中。一个典型的 PE 文件中包含的节如下。

(1) .text：由编译器产生，存放着二进制的机器码，也是反汇编和调试的对象。

(2) .data：初始化的数据块，如宏定义、全局变量、静态变量等。

(3) .idata：可执行文件所使用的动态链接库等外来函数与文件的信息，即输入表。

(4) .rsrc：存放程序的资源，如图标、菜单等。

除此以外，还可能出现的节包括.reloc、.edata、.tls、.rdata 等。

题外话：如果是正常编译出的标准 PE 文件，其节信息往往是大致相同的。但这些节的名字只是为了方便人的记忆与使用，使用 Microsoft Visual C++ 中的编译指示符 #pragma data_seg()可以把代码中的任意部分编译到 PE 的任意节中，节名也可以自己定义，如果可执行文件经过了加壳处理，PE 的节信息就会变得非常“古怪”。在 Crack 和反病毒分析中需要经常处理这类古怪的 PE 文件。

3.1.2 软件加壳

加壳的全称应该是可执行程序资源压缩，是保护文件的常用手段。加壳过的程序可以直接运行，但是不能查看源代码。要经过脱壳才可以查看源代码。

1. 基本原理

加壳其实是利用特殊的算法，对 EXE、DLL 文件里的代码、资源等进行压缩、加密。类似 WinZip 的效果，只不过这个压缩之后的文件，可以独立运行，解压过程完全隐蔽，都在内存中完成。它们附加在原始程序上通过 Windows 加载器载入内存后，先于原始程序执行，得到控制权，执行过程中对原始程序进行解密、还原，还原完成后再把控制权交还给原始程序，执行原来的代码部分。加上外壳后，原始程序代码在磁盘文件中一般是以加密后的形式存在的，只在执行时在内存中还原，这样就可以比较有效地防止破解者对程序文件的非法修改，同时也可以防止程序被静态反编译。

加壳工具在文件头里加了一段指令，告诉 CPU 怎么才能解压自己。现在的 CPU 运行速度都很快，所以这个解压过程看不出差别。软件一下子就打开了，只有机器配置非常差，才会感觉到不加壳和加壳后的软件运行速度的差别。当加壳时，其实就是给可执行的文件加上个外衣。用户执行的只是这个外壳程序。当执行这个程序的时候这个外壳就会把原来的程序在内存中解开，解开以后就交给真正的程序。所以，这些工作只是在内存中运行，具体怎样在内存中运行并不可知。通常说的对外壳加密，都是指很多网上免费或者非免费的软件，被一些专门的加壳程序加壳，基本上是对程序的压缩或者不压缩。因为有的时候程序会过大，需要压缩。但是大部分的程序是因为防止反跟踪，防止程序被人跟踪调试，防止算法程序不想被别人静态分析。加密代码和数据，保护程序数据的完整性。不被修改或者窥视程序的内幕。

加壳虽然增加了 CPU 负担，但是减少了硬盘读写时间，实际应用时加壳以后程序的运行速度更快（当然有的加壳以后会变慢，那是选择的加壳工具问题）。如果程序员给 EXE 程序加壳，那么这个加壳的 EXE 程序就不容易被修改，如果想修改就必须先脱壳。

2. 加壳分类

加壳工具通常分为压缩壳和加密壳两类。

（1）压缩壳的特点是减小软件体积大小，加密保护不是重点。

（2）加密壳种类比较多，不同的壳侧重点不同。一些壳单纯保护程序；另一些壳提供额外的功能。例如，提供注册机制、使用次数、时间限制等。

3.1.3　虚拟内存

为了防止用户程序访问并篡改操作系统的关键部分，Windows 使用了两种处理器存取模式：用户模式和内核模式。用户程序运行在用户模式，而操作系统代码（如系统服务和设备驱动程序）则运行在内核模式。在内核模式下程序可以访问所有的内存和硬件，并使用所有的处理器指令。操作系统程序比用户程序有更高的权限，使得系统设计者可以确保用户程序不会破坏系统的稳定性。

Windows 的内存可以被分为两个层面：物理内存和虚拟内存。其中，物理内存非常复杂，需要进入 Windows 内核级别才能看到。通常，在用户模式下，用调试器看到的内存地址都是虚拟内存。用户编制和调试程序时使用的地址称为虚拟地址（Virtual Address）或逻辑地址（Logical Address），其对应的存储空间称为虚拟内存或逻辑地址空间；而计算机物理内存的访问地址则称为实地址或物理地址，其对应的存储空间称为物理存储空间或主存空间。程序进行虚拟地址到物理地址转换的过程称为程序的再定位。

在 Windows 系统中，在运行 PE 文件时，操作系统会自动加载该文件到内存，并为其映射出 4GB 的虚拟存储空间，然后继续运行，这就形成了进程空间。用户的 PE 文件被操作系统加载进内存后，PE 对应的进程支配了自己独立的 4GB 虚拟空间。在这个空间中定位的地址称为虚拟内存地址。

目前，系统运行在 x64 架构的硬件上，可访问的内存也突破了以前 4GB 的限制，但是独立的进程拥有独立的虚拟地址空间的内存管理机制还在沿用。Windows 装载器在装载的时候仅仅建立好虚拟地址和 PE 文件之间的映射关系，只有真正执行到某个内存页中的指令或访问某页中的数据时，该页才会从磁盘被提交到物理内存。但因为装载可执行文件时，有些数据在装入前会被预先处理（如需要重定位的代码），装入以后，数据之间的相对位置也可能发生改变。因此，一个节的偏移和大小在装入内存前后可能是完全不同的。

注意：操作系统原理中也有虚拟内存的概念，那是指当实际的物理内存不够时，有时操作系统会把部分硬盘空间当作内存使用，从而使程序得到装载运行的现象。请不要将用硬盘充当内存的虚拟内存与本书介绍的虚拟内存相混淆。此外，本书中所述的内存均指 Windows 用户态内存映射机制下的虚拟内存。

3.1.4　PE 文件与虚拟内存的映射

在调试漏洞时，可能经常需要做以下两种操作。

（1）静态反汇编工具看到的 PE 文件中某条指令的位置是相对于磁盘文件而言的，即文件偏移，我们可能还需要知道这条指令在内存中所处的位置，即虚拟内存地址。

（2）反之，在调试时看到的某条指令的地址是虚拟内存地址，我们也经常需要回到 PE 文件中找到这条指令对应的机器码。

为此，需要弄清楚 PE 文件地址和虚拟内存地址之间的映射关系，首先看下面 4 个重要的概念。

- 文件偏移地址（File Offset）

数据在 PE 文件中的地址称为文件偏移地址，是文件在磁盘上存放时相对文件开头的偏移。

• 装载基址(Image Base)

PE 文件装入内存时的基址。在默认情况下，EXE 文件在内存中的基址是 0x00400000，DLL 文件是 0x10000000。这些位置可以通过修改编译选项更改。

• 虚拟内存地址(Virtual Address，VA)

PE 文件中的指令被装入内存后的地址。

• 相对虚拟地址(Relative Virtual Address，RVA)

相对虚拟地址是虚拟内存地址相对于装载基址的偏移量。

虚拟内存地址、装载基址、相对虚拟内存地址三者之间有如下关系：

$$VA = Image\ Base + RVA$$

如图 3-1 所示，在默认情况下，一般 PE 文件的 0 字节将映射到虚拟内存的 0x00400000 位置，这个地址就是**装载基址**。

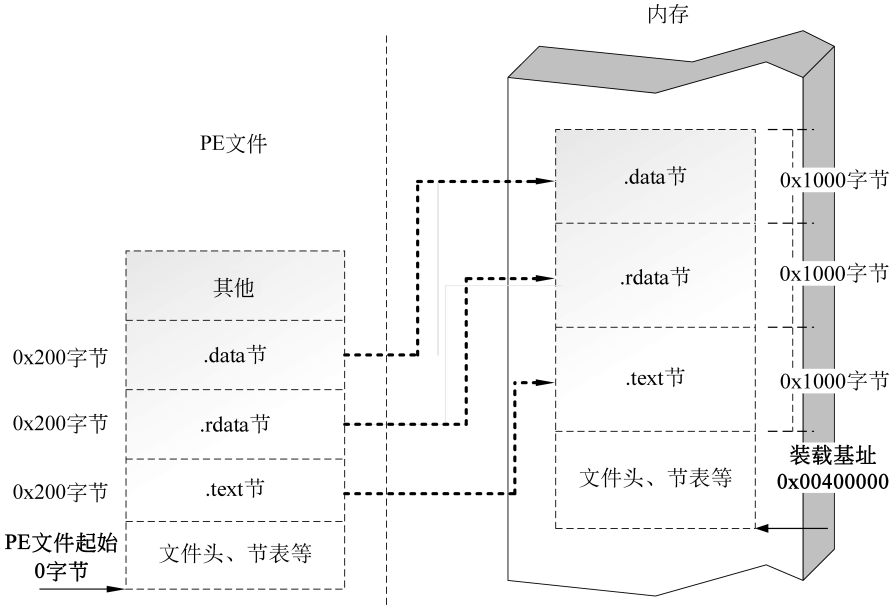


图 3-1 PE 文件与内存的映射关系

文件偏移是相对于文件开始处 0 字节的偏移，相对虚拟地址则是相对于装载基址 0x00400000 处的偏移。由于操作系统在进行装载时基本上保持 PE 中的各种数据结构，所以文件偏移地址和相对虚拟地址基本一致。

之所以说基本一致是因为还有一些细微的差异。这些差异是由于文件数据的存放单位与内存数据存放单位不同而造成的。

(1) PE 文件中的数据按照磁盘数据标准存放，以 0x200 字节为基本单位进行组织。当一个数据节不足 0x200 字节时，不足的地方将被 0x00 填充；当一个数据节超过 0x200 字节时，下一个 0x200 块将分配给这个节使用。因此，PE 数据节的大小永远是 0x200 的

整数倍。

（2）当代码装入内存后，将按照内存数据标准存放，并以 0x1000 字节为基本单位进行组织。类似地，不足将被补全，若超出将分配下一个 0x1000 块为其所用。因此，内存中的节总是 0x1000 的整数倍。

1. 使用 LordPE 可以查看节信息

LordPE 是一款功能强大的 PE 文件分析、修改、脱壳软件。LordPE 是查看 PE 格式文件信息的首选工具，并且可以修改相关信息，其界面如图 3-2 所示。

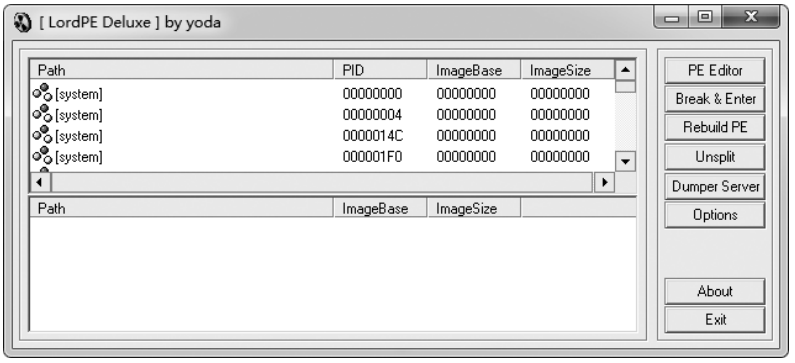


图 3-2　LordPE 界面

单击 PE Editor 按钮，选择需要查看的 PE 文件，如图 3-3 所示。

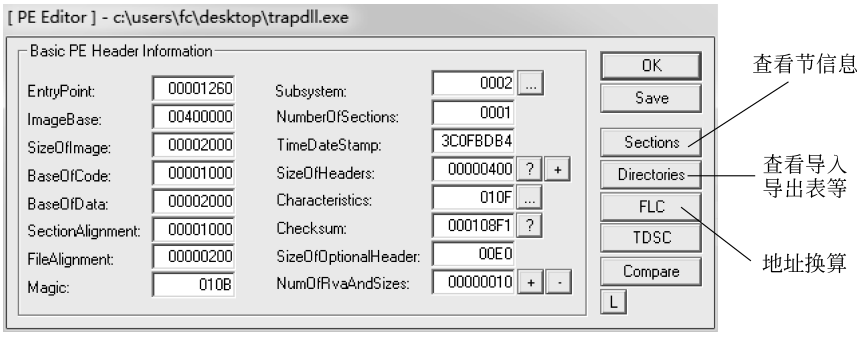


图 3-3　LordPE 查看 PE 文件界面

单击 Sections 按钮，可以查看节信息，如图 3-4 所示。

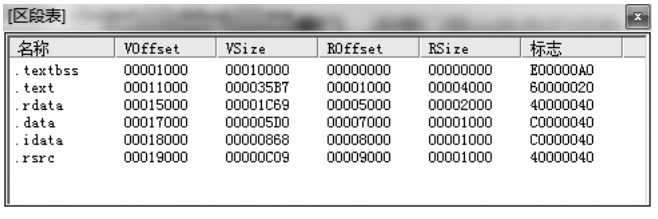


图 3-4　LordPE 查看节信息界面

在图 3-4 中，VOffset 是 RVA，ROffset 是文件偏移地址。也就是说，在系统进程中，代码(.text 节)将被加载到 0x400000＋0x11000＝0x411000 的虚拟地址中(装载基址＋RVA)。而在文件中，可以使用二进制文件打开，看到对应的代码在 0x1000 位置处。

通过文件位置计算器，如图 3-5 所示，也可以看出上述装载基址、RVA、VA 和文件偏移地址的关系。

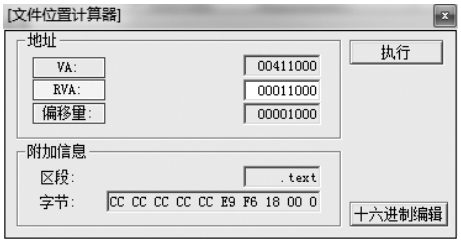


图 3-5　LordPE 文件位置计算器界面

2. 使用 LordPE 查看导入表信息

使用 LordPE 可以查看相关节信息，打开另一个文件，查看区段表信息如图 3-6 所示。

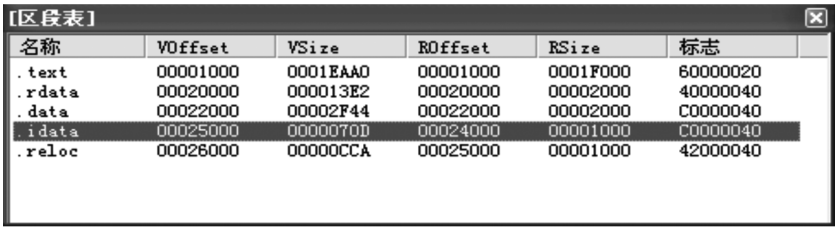


图 3-6　LordPE 查看区段表信息界面

导入表在文件里的偏移地址 ROffset 为 0x24000，RVA 是 0x25000。

如图 3-7 所示，打开目录表可以看到输入表的 RVA 确实是 0x25000。单击 L 按钮可以查看具体输入表里的内容，如图 3-8 所示。可以看到 API 的字符串名字(在按钮 H 打开 PE 文件相关内容后，向后翻阅，可以看到 PE 文件中的相关 API 字符串的内容)。

同时，在目录表里，也可以看到 IAT(Import Address Table，导入地址表)的 RVA 是 0x2513C，如图 3-7 所示。

认识 IAT。每个 API 函数在对应的进程空间中都有其相应的入口地址。众所周知，操作系统动态库版本的更新，其包含的 API 函数入口地址通常也会改变。由于入口地址的不确定性，程序在不同的计算机上很有可能会出错，为了解决程序的兼容问题，操作系统就必须提供一些措施来确保程序可以在其他版本的 Windows 操作系统，以及 DLL 版本下也能正常运行。这时 IAT 就应运而生了。

单击 IAT 右侧的 H 按钮，就可以打开 IAT 信息在 PE 文件中的内容，如图 3-9 所示。

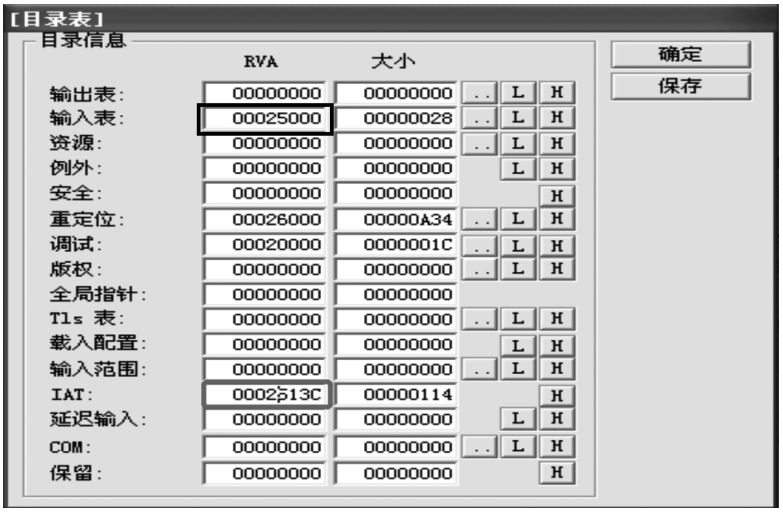


图 3-7　LordPE 查看目录表界面

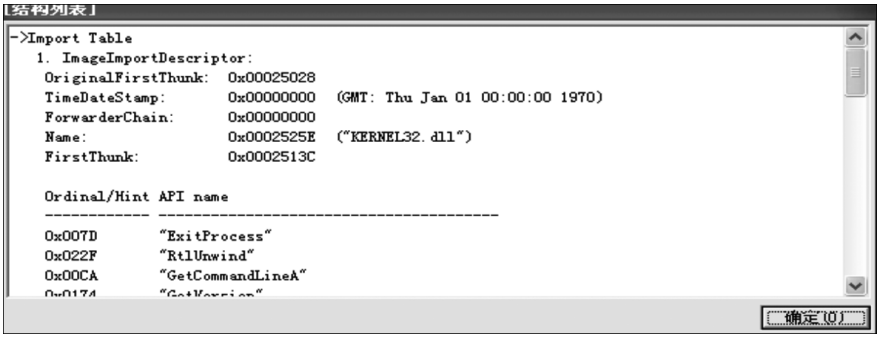


图 3-8　LordPE 查看输入表内容界面

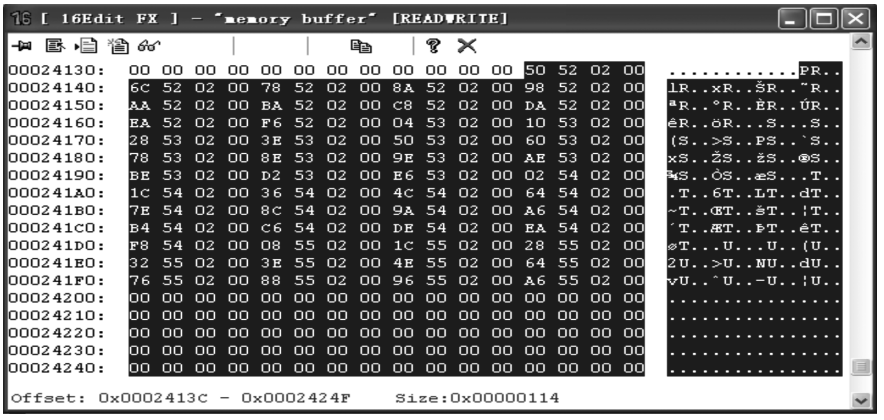


图 3-9　查看 IAT 信息在 PE 文件中的内容

注意：有很多工具可以更加直观地查看 PE 文件格式，如 PEview(见图 3-10)等，可以自行下载查阅 PE 文件，加深对 PE 文件格式的认识和理解。

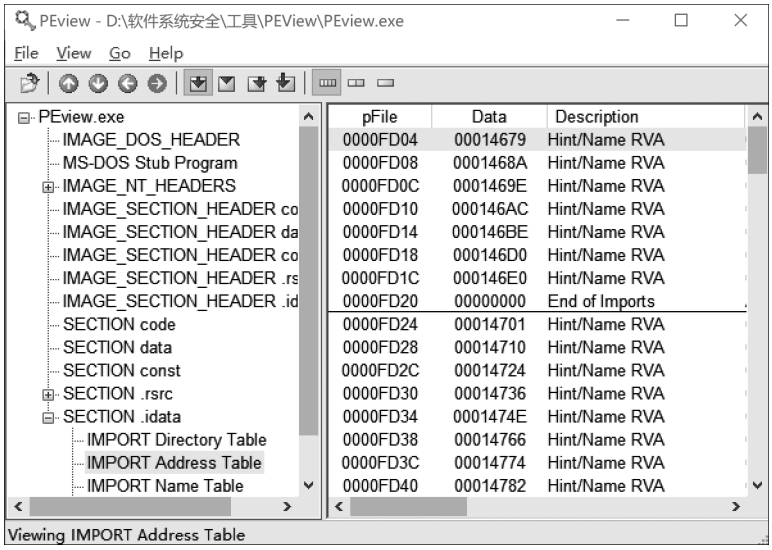


图 3-10　PEview 查看 PE 文件格式

3.2　调试分析工具

在软件调试分析过程中，出现了很多分析工具，被广泛用在逆向分析、调试等领域，包括 OllyDbg、IDA Pro、WinDbg、SoftiCE 等。本书主要介绍前两个工具，并基于这两个工具演示相关的漏洞案例。

3.2.1　OllyDbg

OllyDbg 是一种具有可视化界面的 32 位汇编-分析调试器，适合动态调试。

1. 安装

OllyDbg 版的发布版本是个 Zip 压缩包，解压就可以使用了，如图 3-11 所示。

反汇编窗口：显示被调试程序的反汇编代码。

寄存器窗口：显示当前所选线程的 CPU 寄存器内容。

信息窗口：显示反汇编窗口中选中的第一个命令的参数及一些跳转目的地址、字符串等。

数据窗口：显示内存或文件的内容。

堆栈窗口：显示当前线程的堆栈。

如果要调整上述各窗口的大小，只需按住左键拖动边框，等调整好了，重新启动 OllyDbg 即可。

图 3-11　OllyDbg 界面功能区域说明

2. 基本调试方法

OllyDbg 有两种方式来载入程序进行调试：一种是选择菜单“文件”→“打开”命令(快捷键是 F3)打开可执行文件进行调试；另一种是选择菜单“文件”→“附加”命令附加到一个已运行的进程上进行调试，要附加的程序必须已运行。通常采用第一种。

例如，选择一个 First.exe 来调试，通过选择菜单“文件”→“打开”命令载入这个程序，OllyDbg 中显示的内容如图 3-12 所示。

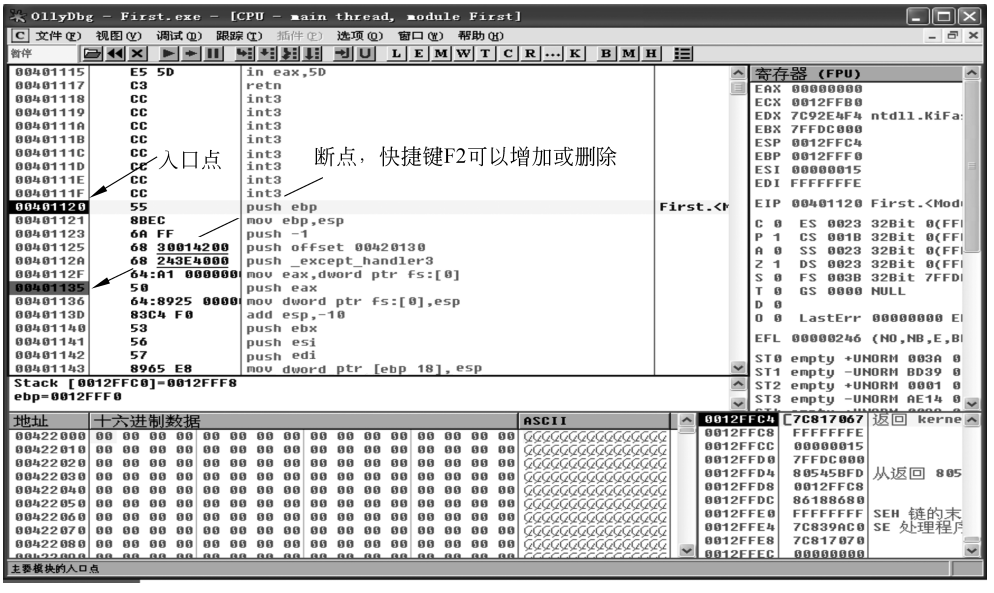


图 3-12　OllyDbg 载入程序

入口点是程序载入后暂停的位置，也可以通过 LordPE 查看入口点的偏移地址。断点可以通过快捷键 F2 来增加或者删除，断点的作用是只要程序运行到这里就会暂停。

调试中经常要用到的快捷键如下。

（1）F2：设置断点，只要在光标定位的位置（图 3-12 中灰色条）按 F2 键即可，再按一次 F2 键则会删除断点。

（2）F8：单步步过。每按一次 F8 键执行反汇编窗口中的一条指令，遇到 CALL 等子程序不进入其代码。

（3）F7：单步步入。功能同单步步过类似，区别是遇到 CALL 等子程序时会进入其中，进入后首先会停留在子程序的第一条指令上。

（4）F4：运行到选定位置。作用就是直接运行到光标所在位置处暂停。

（5）F9：运行。按 F9 键如果没有设置相应断点，被调试的程序将直接开始运行。

（6）Ctrl＋F9：执行到返回。此命令在执行到一个 ret（返回）指令时暂停，常用于从系统返回到调试的程序中。

（7）Alt＋F9：执行到用户代码。可用于从系统快速返回到调试的程序中。

3. 跟踪

使用调试功能时通常会碰到在断点处无法定位入口的情况，即无法确定前序执行指令，通过跟踪（Trace）功能可以记录调试过程中执行的指令，用于分析前序执行指令。Trace 记录可选择是否记录寄存器的值，如图 3-13 所示。

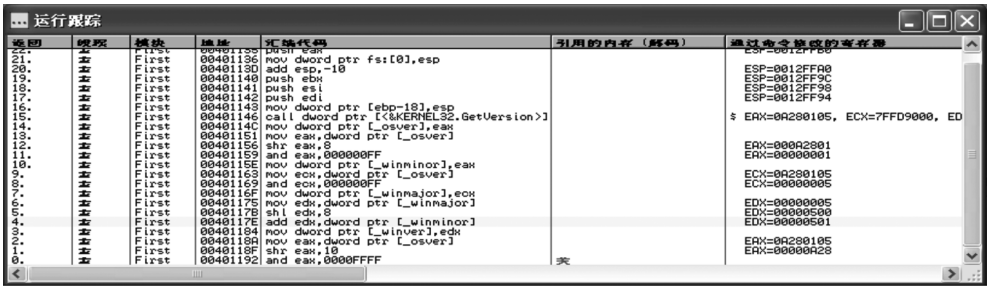


图 3-13　OllyDbg 的 Trace 功能

3.2.2　IDA Pro

IDA Pro 简称 IDA（Interactive Disassembler），是一个世界顶级的交互式反汇编工具。其有两种可用版本：标准版（Standard）支持 20 多种处理器，高级版（Advanced）支持 50 多种处理器。IDA 是逆向分析的主流工具。

打开 IDA，主界面如图 3-14 所示。

IDA 使用 File 菜单中的 Open 命令，可以打开一个计划逆向分析的可执行文件，打开的过程是需要耗费一些时间的。IDA 会对可执行文件进行分析。一旦成功打开，会提示你是否进入 Proximity view。通常都会单击 Yes 按钮，按默认选项进入。如图 3-14 的树状结构的示意图。

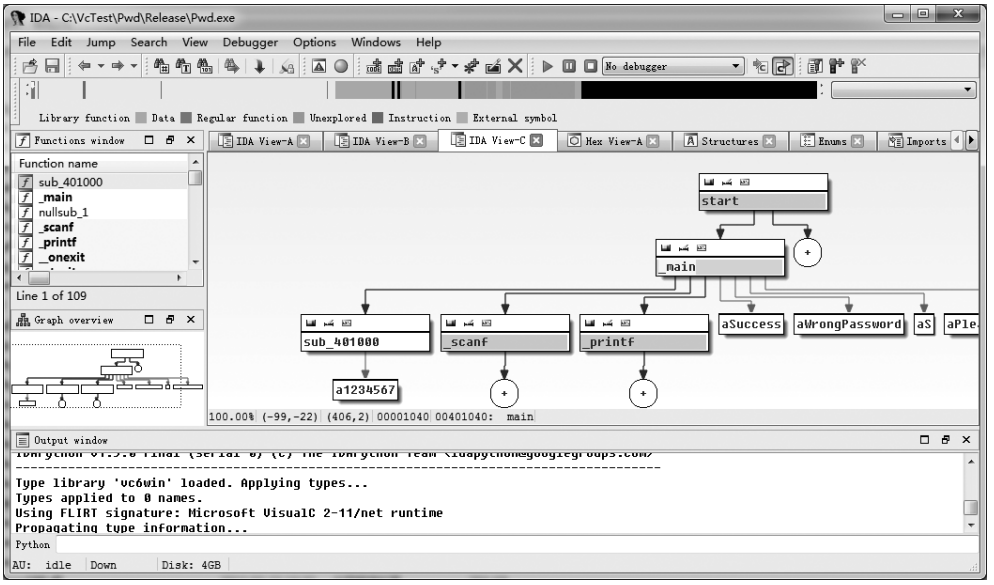


图 3-14　IDA Pro 主界面

1. 主要的数据窗口

在默认配置下，IDA 打开后，3 个立即可见的窗口分别为 IDA View 窗口、Names 窗口和消息输出窗口。在 IDA 中，ESC 键是一个非常有用的热键，在反汇编窗口中，ESC 键的作用与 Web 浏览器中的“后退”按钮类似，但是在其他打开的窗口中，ESC 键用于关闭窗口。

1）反汇编窗口

反汇编窗口也称 IDA View 窗口，是操作和分析二进制文件的主要工具。以前的反汇编窗口有两种显示格式：图形视图(Graph view)和文本视图(Text view)。在默认情况下，会以图形视图显示，在新的版本(IDA 6)里，启动时会提示是否进入 Proximity view，该视图将显示函数及其调用关系。

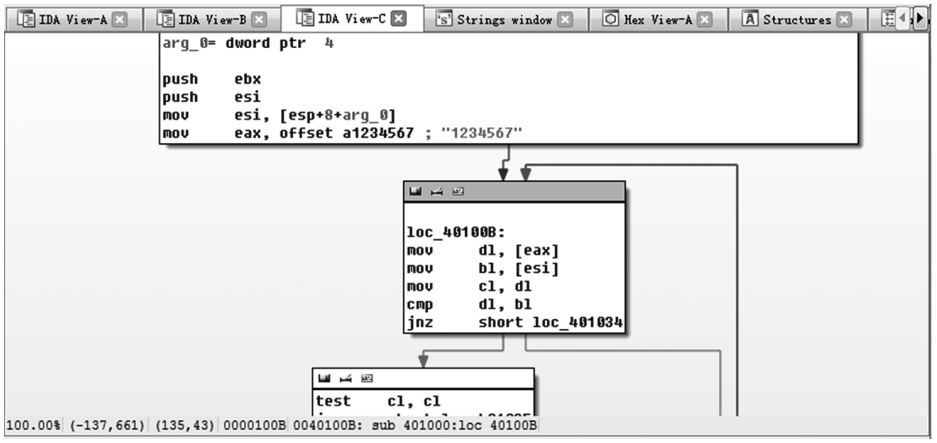
在图 3-14 的 Proximity view 视图中，选择一个块，如_main 函数块，在其上右击，可以看到 Text view 和 Graph view 等选项。通过右键可以实现不同视图的切换。

图形视图：将一个函数分解为许多基本块，类似程序流程图，显示该函数由一个块到另一个块的控制流程。

图 3-15 为_main 函数的图形视图。

在屏幕上可以发现，IDA 使用不同的彩色箭头区分函数块之间各种类型的流：Yes 边的箭头默认为绿色，No 边的箭头默认为红色。蓝色箭头表示指向下一个即将执行的块。

在图形视图下，IDA 一次显示一个函数，使用滑轮鼠标的用户，可以使用“Ctrl 键＋鼠标滑轮”来调整图形大小。键盘缩放控制需要使用“Ctrl＋加号键”来放大，或者“Ctrl＋减号键”来缩小。如果图形太大太乱，不能通过一个视图就完整阅读，则需要结合左侧的图形概况视图(Graph overview)来定位需要阅读的区域。



扫码见彩图

图 3-15　IDA Pro 中_main 函数的图形视图

文本视图：文本视图则呈现一个程序的完整反汇编代码清单(在图形视图下一次只能显示一个函数)，用户只有通过这个窗口才能查看一个二进制文件的数据部分。

如图 3-16 所示的文本视图。

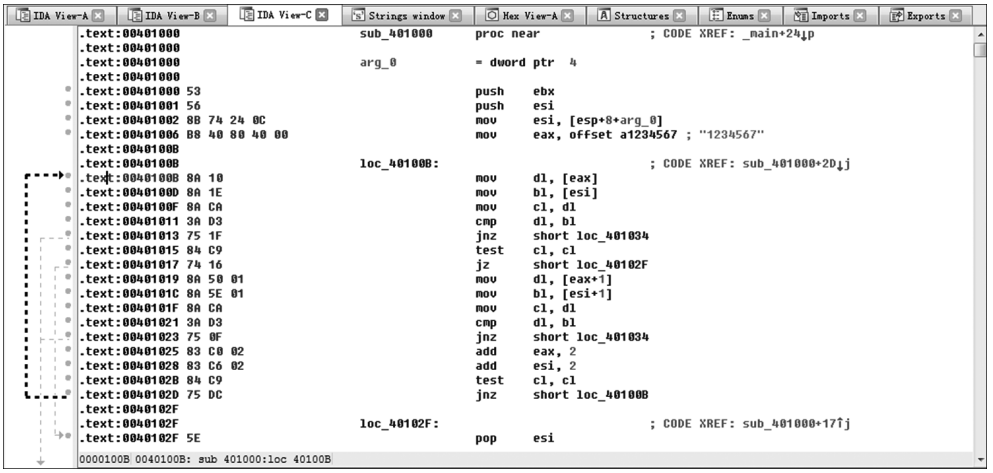


图 3-16　IDA Pro 切换到文本视图

在图 3-16 显示的文本视图中，窗口的反汇编代码分行显示，虚拟地址则默认显示。通常虚拟地址以[区域名称]：[虚拟地址]这种格式显示，如.text：0040110C0。

显示窗口的左边部分称为箭头窗口，用于描述函数中的非线性流程。实线箭头表示非条件跳转，虚线箭头则表示条件跳转。如果一个跳转将控制权交给程序中的某个地址，这时会使用粗线，出现这类逆向流程，通常表示程序中存在循环。

通过选择 Views→Open subviews 命令可以打开更多窗口。

2) Names 窗口

简要列举了一个二进制文件的所有全局名称。名称是指对一个程序虚拟地址的符号

描述。在最初加载文件的过程中，IDA 会根据符号表和签名分析派生出名称列表。用户可以通过 Names 窗口迅速导航到程序列表中的已知位置。双击 Names 窗口中的名称，立即跳转到显示该名称的反汇编窗口。

Names 窗口显示的名称采用了颜色和字母编码，其编码方案总结如下。

（1）F：常规函数。

（2）L：库函数。

（3）导入的名称，通常为共享库导入的函数名。

（4）D：数据。已命名数据的位置通常表示全局变量。

（5）字符串数据。

如图 3-17 所示的 Names 窗口。

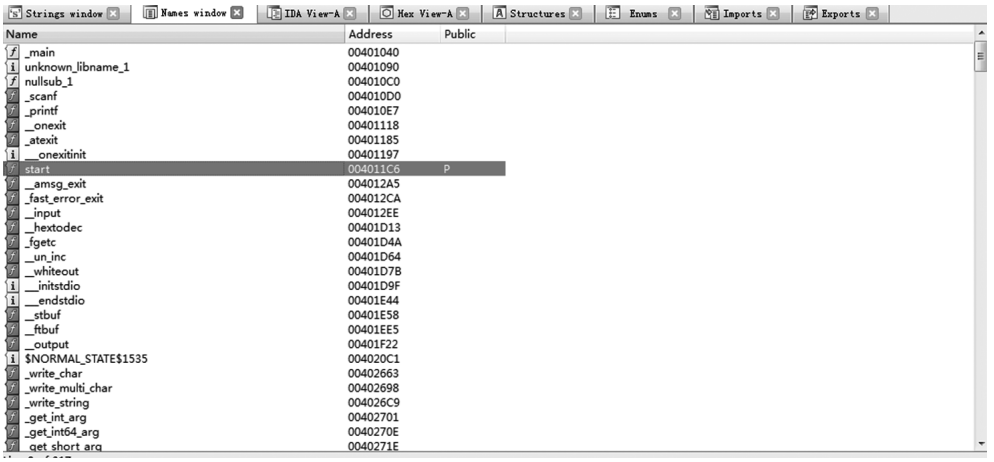


图 3-17　IDA Pro 切换到 Names 窗口

3）Strings 窗口

Strings 窗口功能在 IDA 5 及以前的版本是默认打开的窗口。新版本已经不再默认打开，但是可以通过选择 Views→Open subviews→Strings 命令打开。

Strings 窗口中显示的是从二进制文件中提取出的字符串，以及每个字符串所在的地址。与双击 Names 窗口中的名称得到的结果类似，双击 Strings 窗口中的任何字符串，反汇编窗口将跳转到该字符串所在的地址。将 Strings 窗口与交叉引用结合，可以迅速定义感兴趣的字符串，并追踪到程序中任何引用该字符串的位置。

4）Functions 窗口

Functions 窗口显示所有的函数。单击函数名，可以快速导航到反汇编窗口中的该函数区域。

Functions 窗口中的条目如图 3-18 所示。

这一行信息指出：用户可以在二进制文件中虚拟地址为 00401040 的.text 部分找到 _main 函数，该函数长度为 0x50 字节。

5）Function calls 窗口

Function calls（函数调用）窗口将显示所有函数的调用关系，如图 3-19 所示。

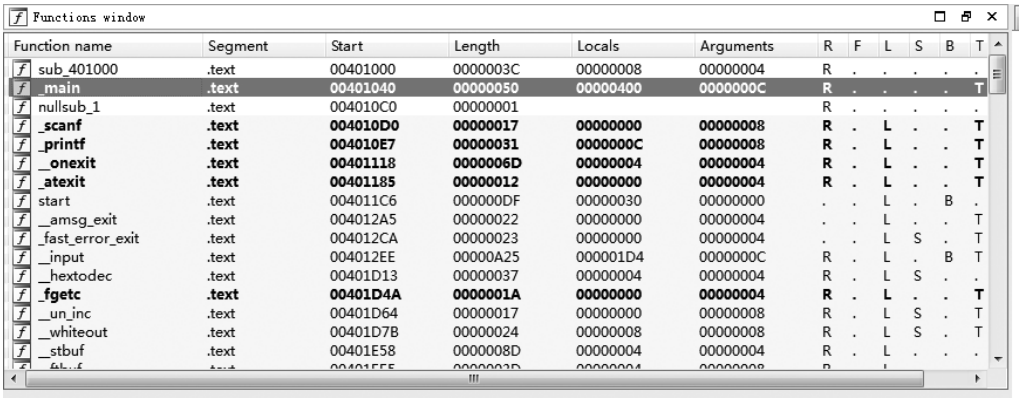


图 3-18　IDA Pro 的 Functions 窗口

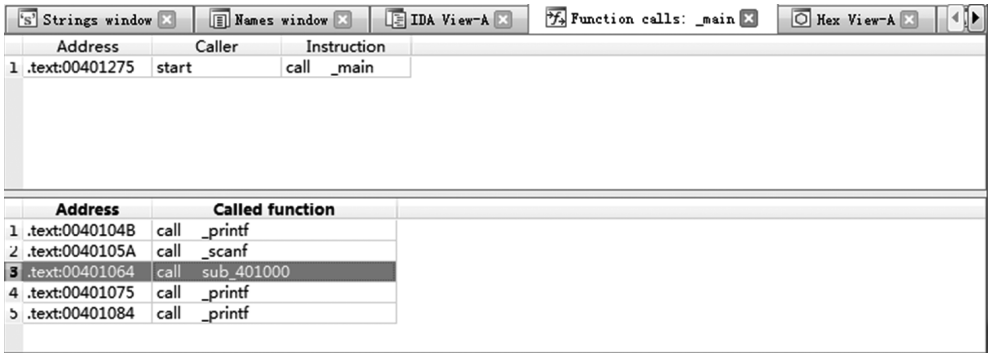


图 3-19　IDA Pro 的 Function calls 窗口

2. 反编译功能

新版本的 IDA 增加了反编译功能，加强了分析能力。

在 IDA View 窗口下制定汇编代码，按 F5 快捷键，IDA 会将当前所在位置的汇编代码编译成 C/C++ 形式的代码，并在 Pseudocode 窗口中显示，如图 3-20 所示。

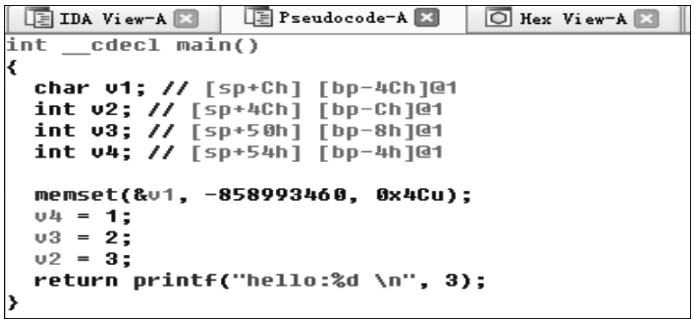


图 3-20　IDA Pro 的反编译窗口

3. 脚本和插件功能

IDA 新版本支持脚本和插件功能，在 IDA 安装目录下 plugins 文件夹里可以存放开发的插件。已经加载的插件可以通过选择 Edit→Plugins 命令查看。

IDA 也支持自己开发的脚本，对于一些经验性操作，可以通过脚本来编程实现。主要支持两类语言的脚本：Python 和 IDC。其中，IDC 是 IDA 自己的脚本语言。在第 6 章中，将演示一个基于 IDA 脚本的漏洞挖掘示例。

3.3　演 示 示 例

3.3.1　PE 文件代码注入

本节将演示利用 PE 文件输入表 API 实现代码注入：让目标程序运行之前，先运行注入的代码，注入的代码将运行 PE 文件输入表里包含的 API。

【实验 3-1】　对 Windows XP 下扫雷程序，使用 OllyDbg 进行代码注入。

目标 PE 文件为 Windows XP 下的扫雷程序，使用的工具包括 OllyDbg 和 LordPE。在 Windows 下找到附件里的扫雷程序右击，在弹出的快捷菜单中选择“属性”命令可以看到具体文件的位置，即 C:\WINDOWS\system32\ winmine.exe。为了方便使用，可以将其复制到桌面上。

首先，用 OllyDbg 打开桌面上的扫雷程序，如图 3-21 所示。

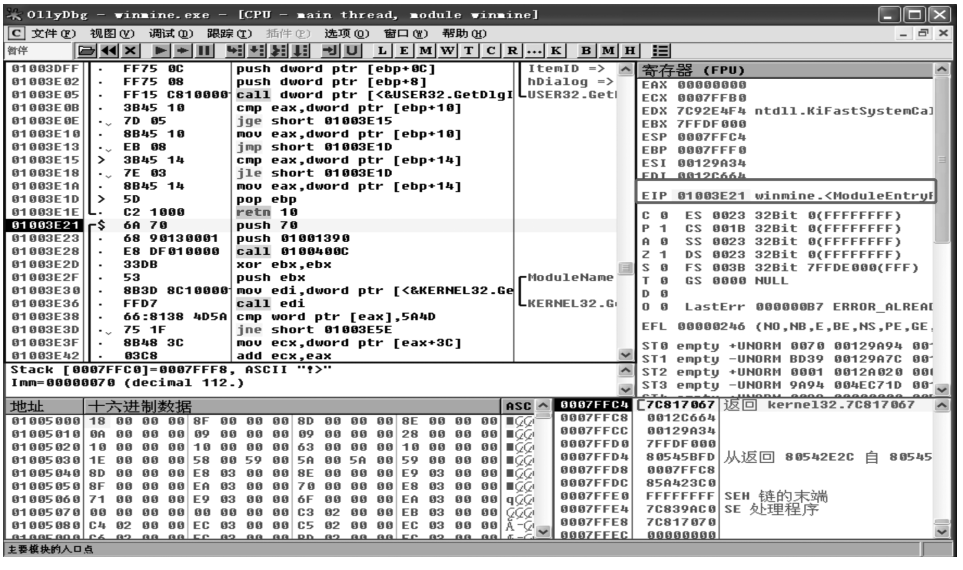


图 3-21　OllyDbg 打开扫雷程序

可以看到，程序会停下来，自动停下来的这一行代码位置就是程序入口点。可以通过 LordPE 文件来查看，得知程序入口点的 RVA 是 0x00003E21，同时也可以看到装载基址是 0x01000000（扫雷程序是 C ++ 语言编写）；也可以通过右侧寄存器 EIP 的值

0x01003E21 观察到注释信息里，提示是 ModuleEntryPoint。

在反汇编区域往下翻页，可以看到相关的导入表动态链接库及其相关函数的信息，如图 3-22 所示。

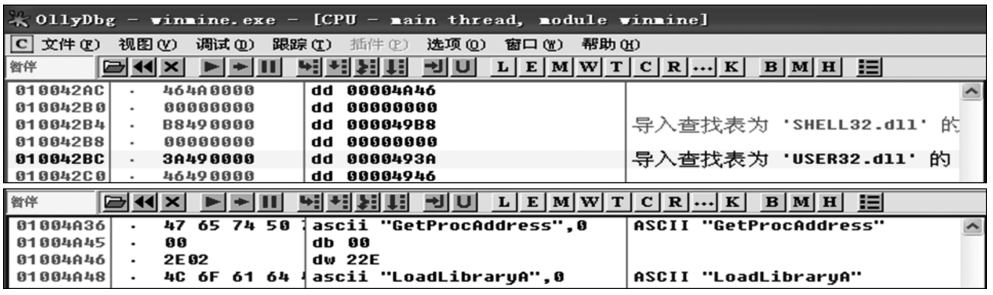


图 3-22 导入表动态链接库及其相关函数的信息

再往下翻页，可以找到大量的空白代码区域，因为这段区域也是在代码区，因此，如果往这里植入代码，直接修改 PE 文件相关跳转地址，就可以执行相关的植入代码。例如，本实验就演示让扫雷程序运行之前，先运行注入的代码，注入的代码将调用 PE 文件输入表里包含的 MessageBox 函数，弹出对话框，显示相关信息。

MSDN 对 MessageBox 函数的解释如下：

```
int MessageBox(
    HWND hWnd,              //handle to owner window
    LPCTSTR lpText,         //text in message box
    LPCTSTR lpCaption,      //message box title
    UINT uType              //message box style
);
```

- hWnd：消息框所属窗口的句柄，如果为 NULL，消息框则不属于任何窗口。
- lpText：字符串指针，所指字符串会在消息框中显示。
- lpCaption：字符串指针，所指字符串将成为消息框的标题。
- uType：消息框的风格（单个按钮、多个按钮等），NULL 代表默认风格。

注意：熟悉 MFC 的程序员一定知道，其实系统中并不存在真正的 MessageBox 函数，对 MessageBox 这类 API 的调用最终都将由系统按照参数中的字符串的类型选择 A 类函数（ASCII）或者 W 类函数（UNICODE）调用。因此，在汇编语言中调用的函数应该是 MessageBoxA。

1. 编辑及注入代码

首先，演示如何直接在 PE 文件里注入代码，计划注入的代码的功能：弹出对话框，显示“You are Injected!”。要达到这个目的，首先要构造相关的字符串，然后构造函数调用的相关汇编代码。

在代码空白区域右击，在弹出的快捷菜单中选择“编辑”→“二进制编辑”命令（不同的

OllyDbg 版本可能稍有差异，有的右击后，直接在快捷菜单里可以看到 ASCII、汇编等功能），如图 3-23 所示。

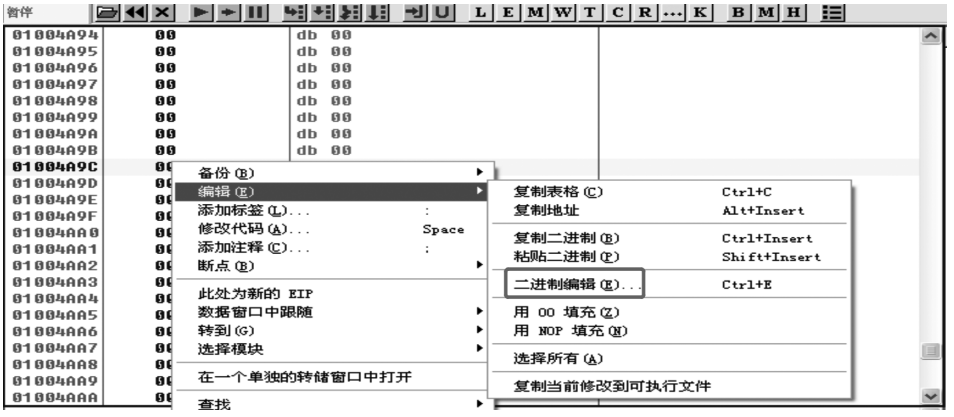


图 3-23　选择“二进制编辑”命令

如图 3-24 所示，在弹出的“编辑地址处的数据”界面里，输入 ASCII 为 PE Inject，将取消勾选“保持代码空间大小”复选框，单击“确定”按钮后，状态如图 3-25 所示。

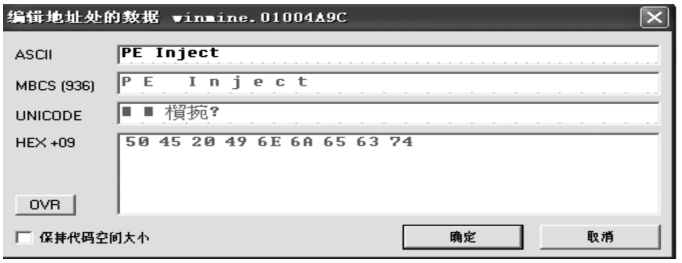


图 3-24　“编辑地址处的数据”界面

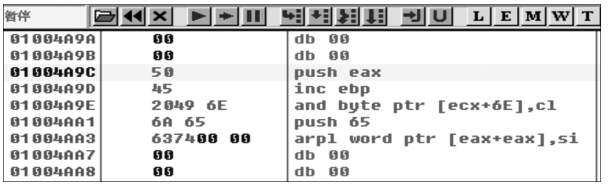


图 3-25　编辑后的代码状态

按 Ctrl＋A 快捷键（分析），将根据具体内容，显示为 ASCII。依这个方式，再加入另一条语句“You are Injected!”，如图 3-26 所示。

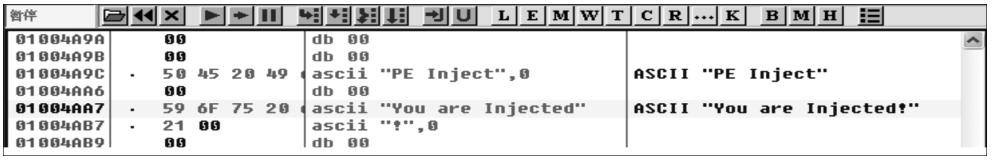


图 3-26　再加入另一条语句“You are Injected!”

注意：上面的每条语句后面都留了一行 00。因为，字符串后面是需要结束符 0x00 的。

接下来，构造函数调用的代码：

```
push 0(默认风格)
push 0x01004AA7(标题字符串地址)
push 0x01004A9C(内容字符串地址)
push 0(窗口归属)
call MessageBoxA
```

注意：直接双击要修改的当前行，就进入修改当前汇编代码的状态，如图 3-27 所示。

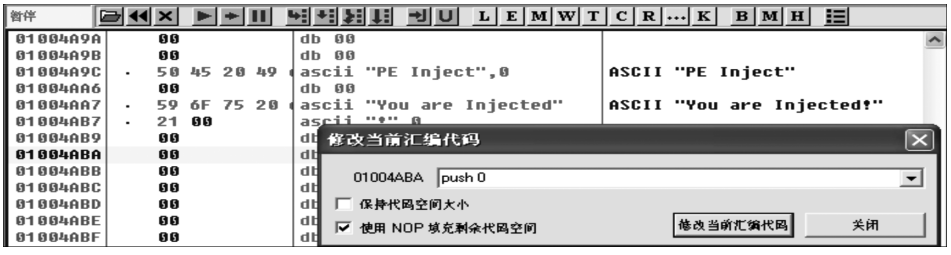


图 3-27　双击进入修改当前汇编代码状态

注意：取消勾选“保持代码空间大小”复选框，直接写汇编代码即可。

我们输入的汇编指令 call MessageBoxA 之所以后面能成功运行，也是因为 PE 文件的输入表里已经有这个函数的入口地址了。以上代码完成输入后，结果如图 3-28 所示。

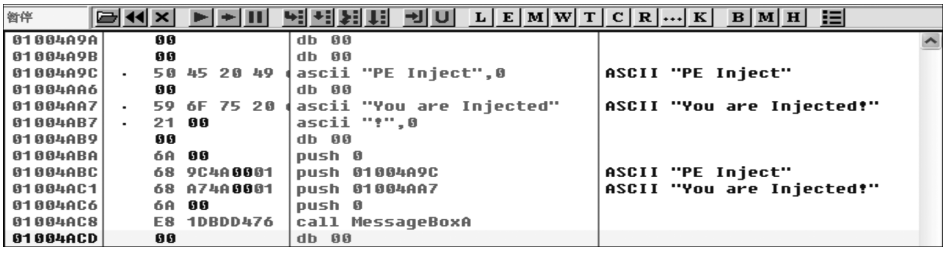


图 3-28　修改后的代码状态

2. 挂接代码及完成跳转

首先继续输入一条指令 jmp 0x01003E21。该条指令的意思是运行完注入的弹出对话框后，会跳转到原来的 PE 文件的入口点继续运行。

结果如图 3-29 所示。

需要注意的是，上述修改是在原始文件副本里修改的，如果要保存修改，需要做到以下两点：①在代码空白区域右击，在弹出的快捷菜单中选择“编辑”→“复制所有修改到可执行文件”命令，会弹出一个对话框，包含所有修改后的代码；②在这个对话框空白处继续右击，在弹出的快捷菜单中选择“编辑”→“保存文件”命令，弹出保存文件的界面，在这里选择保存类型为“可执行文件”或 DLL，输入新的文件名，如 winmine1.exe，单击“保存”

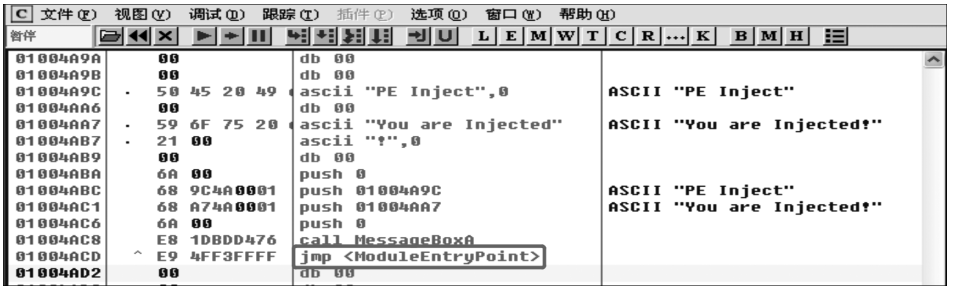


图 3-29　程序跳入原来的 PE 文件入口点

按钮。

到此，文件修改完毕，但是如果直接运行这个扫雷程序，并没有发生任何变化。

因为，我们只是编辑了一段代码，只有这些代码被运行了才算真正被注入。

利用 LordPE 文件，更改一下程序入口点，为程序的起始位置，即编辑的代码段的第一个 push 0 的位置，地址为 0x01004ABA，因为只需要更改 RVA，就修改为 0x00004ABA 即可，如图 3-30 所示。

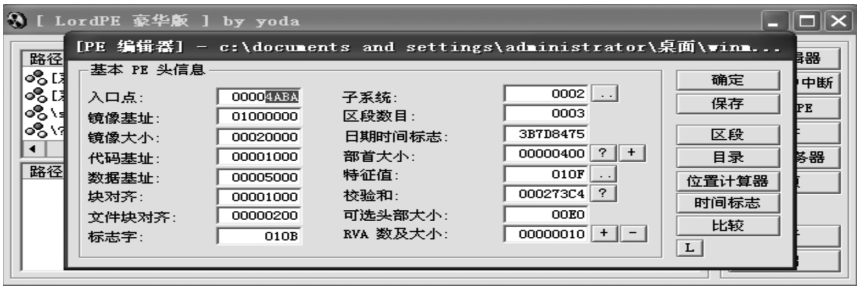


图 3-30　修改程序入口点

单击“保存”按钮，运行程序，先弹出如图 3-31 所示对话框，单击“确定”按钮后，才会出现扫雷程序。

图 3-31　运行程序显示被注入

思考：如果运行程序前注入的不是弹出对话框代码呢？会给我们带来哪些危害？

3.3.2　软件破解示例

本节将对一个简单的密码验证程序，演示如何使用 OllyDbg 进行破解。

【实验 3-2】 对示例 3-1 源代码生成的 Debug 模式的可执行文件，使用 OllyDbg 进行破解。

【示例 3-1】

```
#include<iostream>
using namespace std;
#define password "12345678"
bool verifyPwd(char * pwd)
{
    int flag;
    flag=strcmp(password, pwd);
    return flag==0;
}
void main()
{
    bool bFlag;
    char pwd[1024];
    printf("please input your password:\n");
    while (1)
    {
        scanf("%s",pwd);
        bFlag=verifyPwd(pwd);
        if (bFlag)
        {
            printf("passed\n");
            break;
        }else{
            printf("wrong password, please input again:\n");
        }
    }
}
```

破解对象是该程序生成的 Debug 模式的 exe 程序。

注意：Debug 模式和 Release 模式生成的可执行文件是不同的，采用了不同的编译和连接过程。Release 模式生成的可执行文件不包含调试信息，代码更加精简、干练。

对得到的 exe 程序（假定不知道上面的源代码）有多种方式实现破解。例如，一种方式是使用 OllyDbg，通过运行程序，观察关键信息，通过对关键信息定位得到关键分支语句，通过对该分支语句进行修改，达到破解的目的；另一种方式可以通过 IDA Pro 观察代码结构，确定函数入口地址，对函数体返回值进行更改。

运行程序，输入一个密码，发现运行结果如图 3-32 所示。

在 OllyDbg 中，为了尽快定位到分支语句处，在反汇编窗口右击，在弹出的快捷菜单中选择“查找”→“所有引用的字符串”命令，如图 3-33 所示。