软件逆向分析实用技术

Ruanjian Nixiang Fenxi Shiyong Jishu

宁书林　刘键林　著

北京理工大学出版社
BEIJING INSTITUTE OF TECHNOLOGY PRESS

软件逆向分析实用技术

宁书林　刘键林　著

北京理工大学出版社
BEIJING INSTITUTE OF TECHNOLOGY PRESS

图书在版编目（CIP）数据

软件逆向分析实用技术/宁书林，刘键林著. —北京：北京理工大学出版社，2013.3

ISBN 978-7-5640-7008-3

Ⅰ.①软…　Ⅱ.①宁…②刘…　Ⅲ.①软件工程-分析程序　Ⅳ.①TP311.5

中国版本图书馆 CIP 数据核字（2012）第 266858 号

出版发行／北京理工大学出版社
社　　址／北京市海淀区中关村南大街 5 号
邮　　编／100081
电　　话／(010)68914775(办公室)　68944990(批销中心)　68911084(读者服务部)
网　　址／http：// www. bitpress. com. cn
经　　销／全国各地新华书店
印　　刷／三河市天利华印刷装订有限公司
开　　本／787 毫米×1092 毫米　1/16
印　　张／7.75
字　　数／175 千字
版　　次／2013 年 3 月第 1 版　2013 年 3 月第 1 次印刷　责任校对／陈玉梅
定　　价／34.00 元　责任印制／边心超

序　言

传统的软件工程是从计算机的功能需求出发，将高层抽象的逻辑结构和设计思想通过计划和开发，生产出可实际运行的计算机软件，这个过程称为软件的“正向工程”。反之，从可运行的程序系统出发，运用解密、反汇编、系统分析以及程序理解等多种计算机技术，对软件的结构、流程、算法和代码等进行逆向拆解和分析，推导出软件产品的源代码、设计原理、结构、算法、处理过程、运行方法及相关文档等的过程，称为软件的“逆向工程”(Software Reverse Engineering)，又称软件“反向工程”。通常，人们把对软件进行逆向分析的整个过程统称为软件逆向工程，把在这个过程中所采用的技术统称为软件逆向工程技术。

传统的软件工程主要关注新品软件的分析与设计，而逆向工程则关注成品软件的拆解与剖析。

逆向工程可以让人们了解程序的结构以及程序的逻辑，因此，利用逆向工程可以深入洞察程序的运行过程。可以了解目标程序正在使用的系统函数的类型，也可以了解目标程序访问的文件，当然也可以了解目标软件使用的协议以及目标软件是如何与网络的其他部分通信的。在进行软件保护和反保护的过程中，软件逆向工程的优势是显而易见的。

目前，有许多功能强大、使用方便的软件逆向分析工具。这些软件逆向分析工具支持产生程序的高层抽象，便于分析程序结构，使维护者更容易理解程序，重用旧代码，或根据需要加入新功能，从而改变原有程序的逻辑流。

要学习软件的逆向分析技术，需要熟练掌握汇编语言的相关命令和语法，还要对软件运行的系统平台有较深入的了解。

本书旨在帮助有兴趣学习软件逆向分析技术的读者，了解软件“逆向工程”的概念，学习软件逆向分析技术的基本原理，学习软件逆向分析所涉及的汇编语言的相关命令和用法，Win32 API 调试原理及相关函数，以及常用软件逆向分析工具的使用方法等。并以实际操作案例来说明“逆向分析”的过程和方法。本书不在理论上做过多的赘述，而是注重实用技术和实际操作能力的训练。

由于作者水平有限，难免有不当和疏漏之处，敬请专家读者批评指正。

作　者

目　录

本书章节内容提要

第一章　PE 文件概述及预备知识

内容摘要：PE 是 Portable Executable File Format（可移植的执行体）的简写，它是目前 Windows 平台上的主流可执行文件格式，在这里大家需要有一个简单的了解。

第二章　软件的加密技术

内容摘要：本章主要介绍加密壳和压缩壳。需要加壳的软件按照其目的和作用，可分为两类：一是压缩壳，压缩壳主要目的是减小程序体积；二是加密壳，保护程序不被拆解还原，而达到保护自己的知识产权。

第三章　调试工具的使用

内容摘要：本章主要介绍调试分析工具的使用，包括静态分析工具和动态分析工具。

第四章　Win32 API 调试原理及相关函数

内容摘要：主要讨论 Win32 API 函数逻辑控制结构及其在汇编语言中的实现。主要给出一些在反汇编时经常会遇到的典型情况，供读者分析参考。

第五章　汇编语言基础

内容摘要：汇编和反汇编是对软件进行“逆向分析”时所运用的基本工具。这里仅对“逆向分析”技术所涉及的部分进行重点介绍。

第六章　逆向工程技术

内容摘要：简单介绍一些在学习逆向工程的分析过程中经常遇到的术语，以及逆向分析软件的使用。

第七章　抗逆向工程技术

内容摘要：创建能有效对抗逆向工程技术的程序是非常有用处的。本章从正反两个角度——开发软件的程序员和使用逆向分析技术的工程师，讨论逆向和抗逆向工程的技术特点。

第 1 章　PE 文件概述及预备知识

1.1　PE 文件格式概述

PE 文件格式，像其他的微软可执行文档格式一样，有一系列的字段，固定在一个已知的位置上，定义文件的其他部分。PE 表头内含的重要信息包括程序代码和资料区域的大小位置、适用的操作系统以及堆栈（stack）的最初大小等。

PE 的意思就是 Portable Executable，它是 Win32 环境自身所带的执行体文件格式。它的一些特性继承自 UNIX 的 COFF（Common Object File Format）文件格式。“portable executable”（可移植的执行体）意味着此文件格式是跨 Win32 平台的：即使 Windows 运行在非 Intel 的 CPU 上，任何 Win32 平台的 PE 装载器都能识别和使用该文件格式。当然，移植到不同的 CPU 上 PE 执行体必然得有一些改变。所有 Win32 执行体（除了 VxD 和 16 位的 DII）都使用 PE 文件格式，包括 NT 的内核模式驱动程序（kernel mode drivers）。因而研究 PE 文件格式是我们洞悉 Windows 结构的良机。

另外我们还应该了解 Win32 中的“相对虚拟地址”（Relative Virual Address，RVA）这个名词。PE 文件中的很多的项都是以 RVA 方式指定的，RVA 是其中的一个项相对于文件映像地址的偏移地址。例如：装载程序需要将一个 PE 文件装入到虚拟地址空间中，从 10 000 h 开始的内存中，如果 PE 中某个表在映像中的起始地址是 10 868 h，那么该表的 RVA 就是 868 h。将 RVA 装换为可用的指针，只要将 RVA 的值加上 Module 的基址即可。基地址是指装入到内存中的 EXE 或 DLL 程序的起始地址，它是 Win32 中的一个重要概念。

1. PE 文件总体层次

PE 文件结构的总体层次分布如图 1-1 所示。

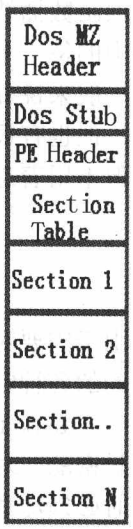


图 1-1　PE 文件结构的总体层次分布

（1）DOS MZ Header（PE 表头）。PE 文件最开始是一个简单的 DOS MZ Header，它是一个 IMage_DOS_Header 结构。有了它，如果程序在 DOS 下执行，DOS 就能识别出这是有效的执行体，然后运行紧随 DOS MZ Header 其后的 DOS Stub。

（2）DOS Stub。DOS Stub 是一个有效的 DOS 程序，和其他的微软可执行文档格式一样，PE 表头并非在文件的最开始处。文件最前面的数百字节是 DOS Stub 一个极小的 DOS 程序，用来输出像“This Program cannot run in DOS mode”这样的错误信息。如果在一个不支持 Win32 的操作系统上运行一个 Win32 程序，就会获得这个错误信息。当 Win32 加载器把一个 PE 文档映像到内存，内存映像文件（memory mapped file）的第一个字节对应到 DOS Stub 的第一个字节。我们也不必对 DOS Stub 进行很深的研究，因为大多

数情况下它是由汇编器/编译器自动生成的，我们对其了解即可。紧接着 DOS Stub 的是 PE Header。

（3）PE Header。PE Header 是 IMAGE_NT_HEADERS 结构的简称，它包含了很多 PE 文件被载入内存时所需要用到的重要域。执行体在支持 PE 文件结构的操作系统中执行，此时 PE 装载器将从 DOS MZ Header 中寻找到 PE Header 的起始偏移量，直接定位到真正的文件头 PE Header。

（4）Section Table。PE Header 接下来的即是数组结构 Section Table （节表）。PE 文件里可以有 N 个节，每节是一块拥有共同属性的数据，比如代码/数据、读/写等。我们可以把 PE 文件想象成为一个逻辑磁盘，PE Header 是磁盘的 boot 扇区，而 sections 就是各种文件，每种文件自然就有不同属性如只读、系统、隐藏或文档等。那么此 Section Table 结构数组内有几个节，就有几个成员，每个成员包含对应节的属性、文件偏移量及虚拟偏移量等。

2. PE 文件主要运行步骤

装载 PE 文件的主要运行步骤如下。

① PE 文件被执行，PE 装载器检查 DOS MZ Header 里面的 PE Header 偏移量。如果找到则跳转到 PE Header。

② PE 装载器检查 PE Header 的有效性。如果有效，就跳转到 PE Header 的尾部。

③ 紧跟 PE Header 的是节表。PE 装载器读取其中的节信息，并采用文件映射方法将这些节映射到内存，同时赋上节表里指定的节属性。

④ PE 文件映射入内存后，PE 装载器将处理 PE 文件中类似 import table（引入表）的逻辑部分。程序装载的时候需要装载很多函数和 DLL 文件，这时程序需要判定目标函数的地址并将该函数插补到该执行文件的映像中，所需要的信息都是放在 PE 文件的 Import 表中，PE 文件中的每一个输入函数都明确地列于该表中。

一般来说 Import 表是存放在程序的 idata 块中，它一般包含其他外来 DLL 的函数及数据信息。但也有很多程序的 Import 表不存在 idata 块中，给我们判断 Import 表的地址造成了困难，但不用着急，只要了解了 Import 表的结构就能迅速定位 Import 表的地址。

1.2 各种块的描述

由于需要对某些关键块进行加密处理和逆向分析，所以我们有必要了解各执行文件中经常遇到的各种常见块。

（1）text。text 是在编译或汇编结束时候产生的一种块。它的内容全是指令代码，它是必须要加密的。对它的加密可以有效地防止对原程序指令代码的静态分析和修改。它也是逆向分析时必须注意的。PE 文件是运行在 32 位方式下，所以没有必要将不同文件产生的代码再分成分离的块。连接器把所有目标文件的 text 块连接成一个大的 text 块。如果使用的是 Borland C++，其编译器将产生的代码存于名称为 CODE 的区域中，而不是 text。

（2）data。如同 text 是默认的代码块一样，data 是初始化的数据块。这些数据包括编译时被初始化的 globle 和 static 变量，也包括字符串。连接器将 OBJs 及 LIBs 文件的 data 结合成一个大的 data。local 变量存放在一个线性的堆栈中，不占 data 和 bbs 的空间。和 text 一样，

数据块是以明文的形式存放在文件中的，所以我们很有必要将其加密。

（3）idata。idata 节是输入数据，包括输入目录和输入地址名字表。idata 包含其他外来 DLL 的函数及数据信息。该块功能与文件的模块引用表类似，关键的差异在于 PE 文件中的每一个输入函数都明确地列于该块中，要找到相同的信息，必须从各个段的重定位数据中查找。

（4）rsrc。rsrc 包含模块的全部资源，如图标、菜单和位图等，该资源结构复杂。

（5）reloc。reloc 保存基地址重定位表。当装载程序不能按连接器所指定的地址装载文件时，需要对指令或已初始化的变量进行调整，基地址重定位表包含了调整所需的数据。如果装载程序能够正常装载文件，它就忽略 reloc 中的重定位数据。

（6）edata。edata 是该 PE 文件输出函数和数据的列表，以供其他模块引用。与文件中的入口表、驻留名表及非驻留名表综合功能相似。PE 格式文件没有必要输出一个函数，所以通常只是在 DLL 文件中才可以见到 edata 块。

（7）tls。tls 的意思是“thread local storage”（线程本地存储器），它与 Win32 的 TlsAlloc 系列功能有关。

（8）rdata。rdata 块通常是在 data 或 bss 中间，但程序中很少用到该块中的数据。但至少有两种情况下要用到 rdata，一是在 Microsoft 的连接器产生的 EXE 文件中，用于存放调试目录，二是用于存放说明字符串。如果程序的 DEF 文件中指定了 DESCRIPTION，则字符串就会出现在 rdata 中。

1.3　Import Table（引入表）

首先，我们先了解什么是引入函数。一个引入函数是被某模块调用但又不在调用者模块中的函数，因而命名为“import”（引入）。引入函数实际位于一个或者更多的 DLL 里。调用者模块里只保留一些函数信息，包括函数名及其驻留的 DLL 名。PE 文件的 IMPORT 在调用外部 DLL 时，CALL 指令实际上是被转化成 EXE 文件 text 块中的 Jmp dword　ptr [xxxxxxxx] 指令（如果使用的是 Borland C++，则在 icode 块中），Jmp 指令要跳转到的地方才是真正的目的地址。装载程序判定目标函数的地址并将该函数插补到执行文件的映像中，所需要的信息都是放在 PE 文件的 idata 中，也就是 Import 块。由于我们的外壳程序需要在原程序执行前进行一系列的工作，需要用到大量的 Windows 的 API 调用，所以我们必须深入地了解 Import 块的结构和装入机制。

1.4　Export Table（引出表）

当 PE 装载器执行一个程序时，它将相关 DLL 都装入该进程的地址空间，然后根据主程序的引入函数信息，查找相关 DLL 中的真实函数地址来修正主程序。PE 装载器搜寻的是 DLL 中的引出函数。DLL/EXE 要引出一个函数给其他 DLL/EXE 使用，有两种实现方法：通过函数名引出或者仅通过序数引出。比如某个 DLL 要引出名为“GetSysConfig”的函数，如果它以函数名引出，那么其他 DLL/EXE 若要调用这个函数，必须通过函数名，就是 GetSysConfig。另外一个办法就是通过序数引出。什么是序数呢？ 序数是唯一指定 DLL 中某个函数的 16 位

数字，在所指向的 DLL 里是独一无二的。例如，DLL 可以选择通过序数引出，假设是 16，那么其他 DLL/EXE 若要调用这个函数必须以该值作为 GetProcAddress 调用参数。这就是所谓的仅靠序数引出。

我们不提倡仅通过序数引出函数这种方法，这会带来 DLL 维护上的问题。一旦 DLL 升级/修改，程序员无法改变函数的序数，这样调用该 DLL 的其他程序都将无法工作。

1.5 PE 文件格式中的结构及其作用

1. 结构

在 PE 的各种结构中，涉及很多地址、偏移。有些是指在文件中的偏移，有的是指在内存中的偏移。一定要搞清楚，这个地址或者是偏移，是指在文件中，还是指在内存中。

第一种，文件中的地址。比如用 16 进制编辑器打开 PE 文件，看到的地址（偏移）就是文件中的地址，我们使用某个结构的文件地址，就可以在文件中找到该结构。

第二种，文件被整个映射到内存时，比如某些 PE 分析软件，把整个 PE 文件映射到内存中，这时是内存中的地址，如果知道某一个结构在文件中的地址的话，那么这个 PE 文件被映射到内存之后该结构在内存中的地址，用文件中的地址加上映射内存的地址就可以得到在该结构内存中的地址。

第三种，执行 PE 时，PE 文件会被载入器载入到内存，这时经常需要的是 RVA。比如知道一个结构的 RVA，那么载入点加上 RVA 就可以得到内存中该结构的实际地址。

2. 分布

PE 文件结构的总体层次分布：所有 PE 文件必须以一个简单的 DOS MZ header 开始，在偏移 0 处有 DOS 下可执行文件的“MZ 标志”，有了它，一旦程序在 DOS 下执行，DOS 就能识别出这是有效的执行体，然后运行紧随 MZ Header 之后的 DOS Stub。

DOS Stub 实际上是个有效的 EXE，在不支持 PE 文件格式的操作系统中，它将简单显示一个错误提示，类似于字符串“This program cannot run in DOS mode”或者程序员可根据自己的意图实现完整的 DOS 代码。通常 DOS Stub 由汇编器/编译器自动生成，对我们的用处不是很大，它简单调用中断 21 h 服务 9 来显示字符串“This program cannot run in DOS mode”。紧接着 DOS stub 的是 PE Header。

PE Header 是 PE 相关结构 IMAGE_NT_HEADERS 的简称，其中包含了许多 PE 装载器用到的重要域。可执行文件在支持 PE 文件结构的操作系统中执行时，PE 装载器将从 DOS MZ Header 的偏移 3CH 处找到 PE Header 的起始偏移量。因而跳过了 DOS Stub 直接定位到真正的文件头 PE Header。

PE 文件的真正内容划分成块，称之为 Sections（节）。每节是一块拥有共同属性的数据，比如“text”节等，那么，每一节的内容都是什么呢？实际上 PE 格式的文件把具有相同属性的内容放入同一个节中，而不必关心类似“text”、“data”的命名，其命名只是为了便于识别，所以，我们如果对 PE 格式的文件进行修改，理论上讲可以写入任何一个节内，并调整此节的属性就可以了。

PE Header 接下来的数组结构 Section Table（节表），每个结构包含对应节的属性、文件

偏移量和虚拟偏移量等。如果 PE 文件里有 5 个节，那么此结构数组内就有 5 个成员。

以上就是 PE 文件格式的物理分布。

1.6 小　结

可以看出，PE 文件比起传统的 DOS 的 MZ 格式的可执行文件要复杂得多，功能和性能得到了巨大的扩充和增强，这都是由新型操作系统的机制所决定的。但它又在功能增强的基础上，比 Windows 3.1 的 NE 格式文件简单了很多，这充分说明了 PE 文件格式既功能强大又较为简洁的结构和优异的装入效率及其性能。

第 2 章　软件的加密技术

2.1　壳的认识

壳（这个“壳”在本文中既指压缩壳也包括加密壳）是用来防止程序被逆向分析的。它们被商业软件合法地用于防止信息披露、篡改及盗版。可惜恶意软件也基于同样的理由在使用壳，只不过是动机不良而已。大家都会发现在自然界当中，植物用壳来保护种子，动物用壳来保护自己的某些部位等。同样，在计算机软件里也有一段专门用来负责保护软件不被非法修改或逆向分析的程序。它们一般都是先于程序运行，拿到控制权，然后完成它们保护软件的任务。就像动植物的壳一般都是在身体外面一样，由于这段程序和自然界当中的壳在功能上有很多相同的地方，因此大家就把这样的程序称为“壳”了。软件的壳和自然界中的壳相差无几，目的都是保护和隐蔽壳内的东西。如果我们从技术的角度分析的话，壳就是一段执行于原始程序前面的代码。原程序代码在加壳过程中可能被压缩、加密。当加壳后的程序需要运行时，壳的这段代码先于原始程序运行，它把压缩、加密后的代码还原成原始程序的代码，然后再把执行权交还给原始程序。软件的壳可分为加密壳和压缩壳。加密壳的主要目的就是为了隐藏程序真正的入口点，压缩壳的主要目的是减小体积、方便传播。

2.2　加密保护壳简介

为了保护好自己的软件，有必要对软件进行一些加密的保护，而目前已经有很多的成熟的专业公司在撰写加密软件可供我们选择。现在专业的加密软件有很多，这里向大家介绍几款常见的加密保护壳。

1. Themida 加密壳

如图 2–1 所示，Themida 是 Oreans 的一款商业壳，是一款强有力的加密壳软件，为保护程序免受先进的逆向工程和软件破解的需要而设计的。

2. Armadillo 加密壳

如图 2–2 所示，Armadillo 可以运用各种设置来保护程序，可以同时为软件的保护进行设置限制、时间限制、次数限制，很多商用软件均采用 Armadillo 加壳。是当今猛壳之一，壳如其名拥有厚重的装甲，其加壳方式有两种，第一种是标准方式，第二种是：CopyMem-II+Debug-Blocker。

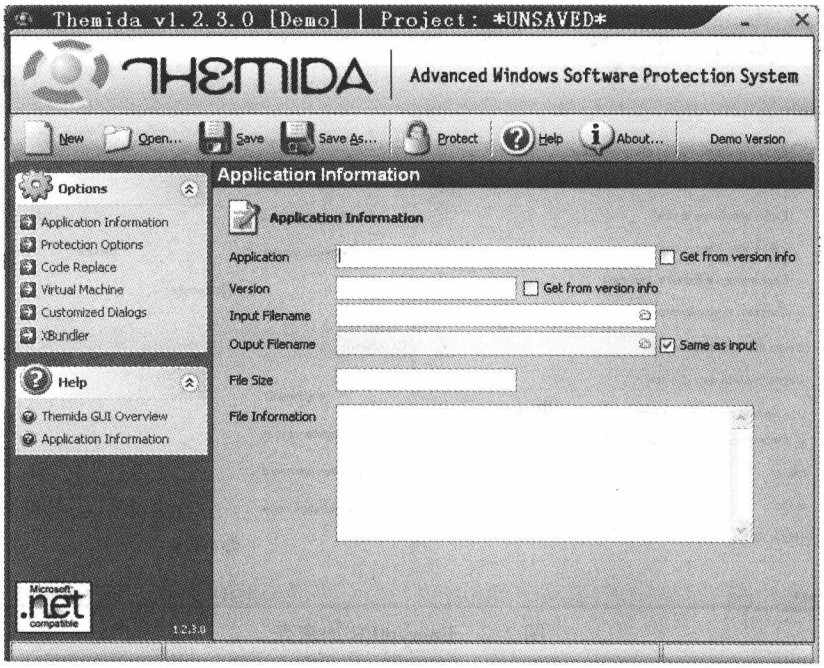


图 2–1　Themida 加密壳

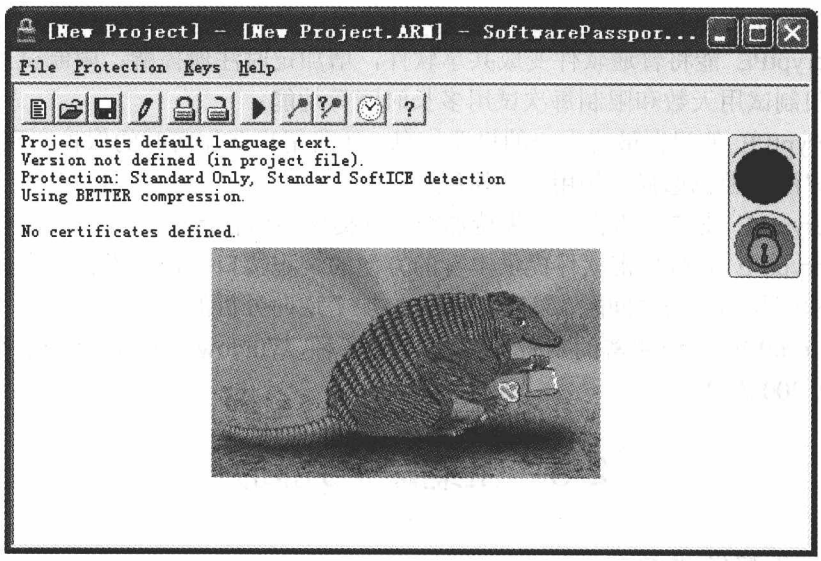


图 2–2　Armadillo 加密壳

其标准加壳方式相对来说则容易得多，双进程方式加壳的程序是 Armadillo 的最大特点。

3. EncryptPE 加密壳

EncryptPE 壳如图 2–3 所示。

（1）EncryptPE 能加密保护常规 PE 文件（EXE、DLL 和 OCX 等一般程序或 NT 服务程序），防止静态分析修改，反动态跟踪调试，有效地保护软件，防止盗版。除常规的对抗调试器（SoftIce、TRW 及 OllyDbg 等）、监视器和 DUMP 工具方法外，EncryptPE 采用的加密保

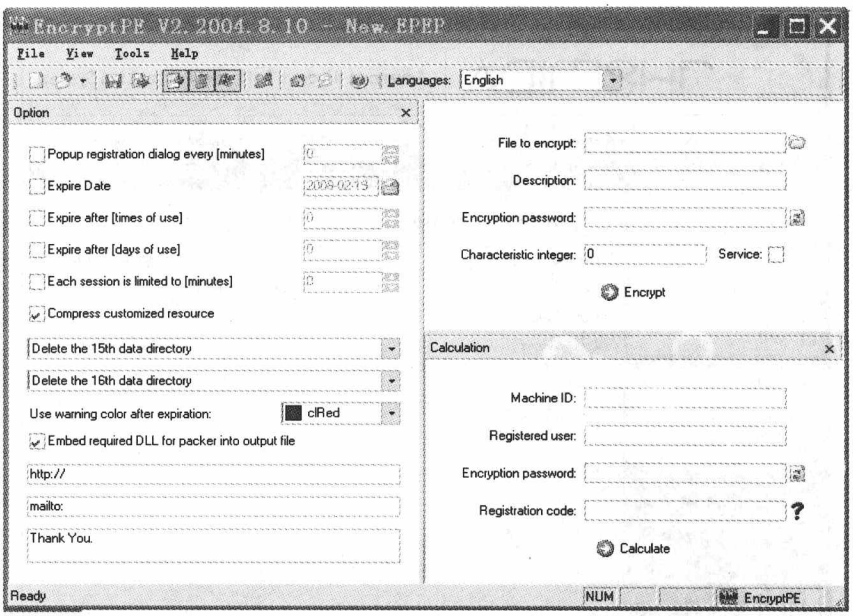


图 2–3 EncryptPE 加密壳

护的手段还有：随机加密算法、CRC 校验、变形、代码替换、进程注入、APIHOOK、多线程、调试运行及全程监控等。

（2）EncryptPE 能将普通软件变成共享软件，增加定时注册提醒、限制试用日期、限制试用次数、限制试用天数和限制每次试用多长时间等功能。

（3）EncryptPE 能根据最终用户的机器信息、注册用户及加密时的保护密码计算注册码，从诸多加密算法中随机选择一种用于注册码的计算。

（4）EncryptPE 支持多语言，并为待加密软件提供多语言接口。

（5）EncryptPE 向待加密软件提供丰富的方便的编程接口，便于设计个性注册方式，同时使被加密程序与加密壳之间融为一个整体，增加了逆向分析的难度。

（6）EncryptPE 及经加密的软件可以运行于多种 Windows 平台，包括 Windows 9X/ME/NT/2000/ 2003/ XP。

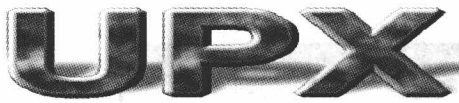
2.3 压缩保护壳简介

1. UPX 压缩保护壳

如图 2–4 所示是 UPX 压缩保护壳运行在 DOS 下的帮助文件。

（1）UPX 是一款先进的可执行程序文件压缩器，其压缩过的可执行文件体积可缩小 50%～70%，这样减少了磁盘占用空间、网络上传下载的时间和其他分布以及存储费用。

（2）通过 UPX 压缩过的程序和程序库完全没有功能损失，和压缩之前一样可正常地运行，对于支持的大多数格式文件也没有不利的影响。UPX 支持许多不同的可执行文件格式，包括 Windows 95/98/ME/NT/2000/XP/ Vista 程序和动态链接库、DOS 程序以及 Linux 可执行文件和核心。



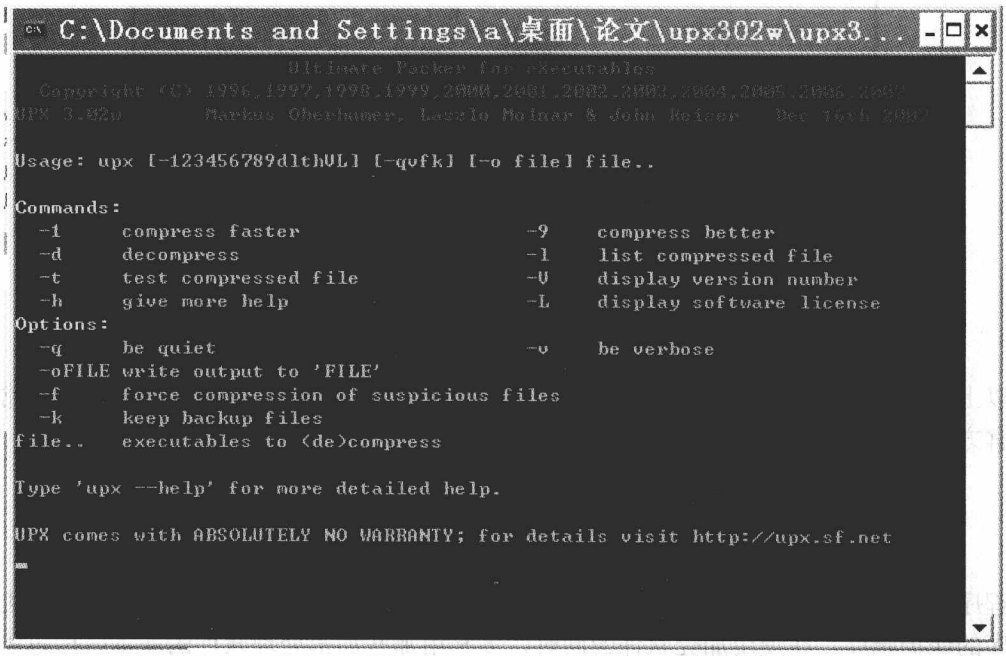


图 2–4　UPX 压缩保护壳帮助文件

2. ASPack 压缩保护壳

如图 2–5 所示是 ASPack，它是由俄国作者 Alexey Solodovnikov 写的一款非常好的 Win32 格式可执行文件压缩件，使用非常方便，而且操作很快捷。以往的压缩工具，通常是将计算机中的资料或文档进行压缩，用来缩小储存空间，但是压缩后就不能再运行了，如果想运行必须解压缩。另外当你的系统中无压缩软件时，压缩包即无法解开。而 ASPack 的独特就在这里，ASPack 是专门对 Win32 可执行程序进行压缩的工具，压缩后程序能正常运行，丝毫不会受到任何影响。而且即使你已经将 ASPack 从系统中删除，但曾经压缩过的文件仍可正常使用，文件压缩比率高达 50%～70%。

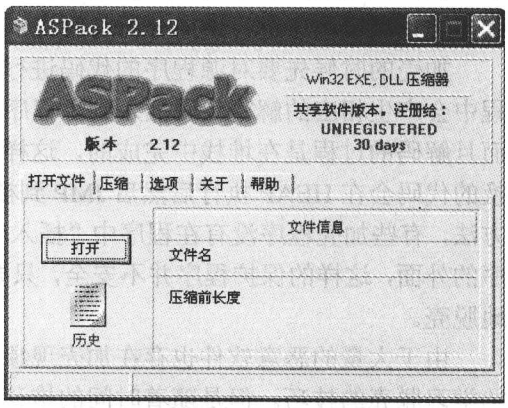


图 2–5　ASPack 压缩保护壳

3. Petite 2.3 –GUI

如图 2–6 所示，Petite 是一个 Win 32 （Windows 9x/NT 等）可执行文件压缩器。它允许压缩整个可执行文件、代码或数据，能压缩 PE 文件的 code 及 data 等资源。Petite 可以自动决定压缩可执行文件的哪些部分，以及哪些部分需要保留。压缩过的文件运行起来就像原

始的未压缩版本一样。

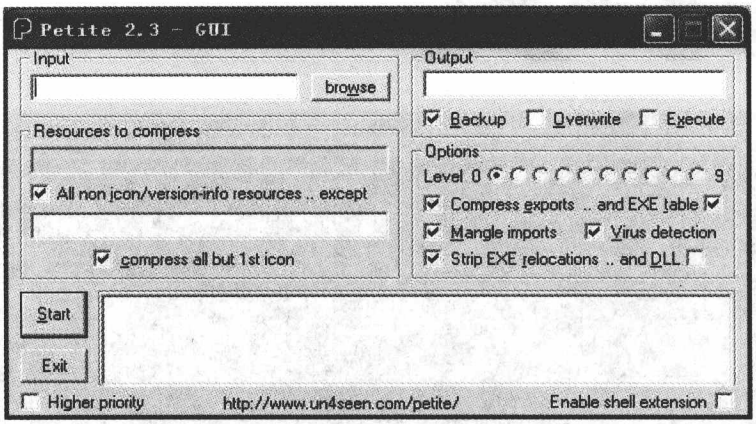


图 2–6 Petite

以上只是简单地介绍了很少的一部分加密壳和压缩壳。如果大家有兴趣，可以到常见加密壳官方站点购买。

2.4 小 结

程序员在充分利用现有加密软件的同时，可以将更多的精力放到程序设计编写中去，提高程序的保护强度。在使用加壳软件的同时，多利用那些加密处理软件的基本分析思路，加入反逆向分析的关键代码以及反调试器的语句，关键是使用得要好，要将程序关键代码进行处理。

当然，在选择加壳软件时要选一个没有脱壳机的加壳软件，或者不大可能写出脱壳机的加壳软件。这样的壳对程序有较好的保护措施，例如：对 IAT 进行加密处理，加入 Anti-Debug 代码防止调试程序进行逆向分析与脱壳。

加壳的时候先要对源程序的代码进行分析，然后进行关键代码的替换与加密，在这个过程中会产生相应的解密代码插入到源程序中。加壳后的程序在执行的时候是分段解密执行的，而且解码的过程是在堆栈中完成的，这样做是为了增加代码还原的难度，原程序的一些被替换的代码会在 HEAP 执行后然后 JMP 到被解码的程序处再执行。现在很多壳都使用了这样的方法，有些加壳程序没有在程序中“插入/替换”加/解密代码，只是将自身的加密壳包在原程序的外面，这样的保护程序并不安全，只要找到了关键的跳转和 OEP 入口点就可以非常轻松地脱壳。

由于大量的恶意软件也存在加壳现象，研究人员和恶意代码分析人员为了分析代码，开始学习脱壳的技巧。但是随着时间的推移，为防止逆向分析人员分析受保护的程序并成功脱壳，新的反逆向分析技术也被不断地添加到壳中。现在互相推进还在继续，新的反逆向技术被开发的同时逆向分析人员也在针锋相对地发掘技巧、研究技术并开发工具来对付它们。

无论从哪个方面讲，学习、了解加密与逆向分析技术对我们今后的程序开发与保护将起到很大的推进作用。