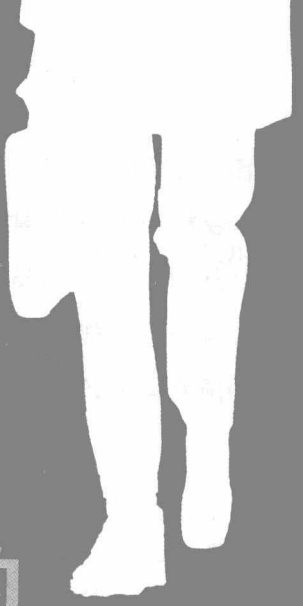
揭秘数据解密的关键技术

著

学数据解密，
从这里开始……

人民邮电出版社
POSTS & TELECOM PRESS



揭秘
数据解密的关键技术

刘颖东　编著

人民邮电出版社
北京

图书在版编目（CIP）数据

揭秘数据解密的关键技术 / 刘颖东编著. —北京：人民邮电出版社，2009.4
ISBN 978-7-115-19670-5

Ⅰ. 揭… Ⅱ. 刘… Ⅲ. 电子计算机—密码术 Ⅳ. TP309.7

中国版本图书馆CIP数据核字（2009）第009241号

内 容 提 要

本书是一本以游戏资源文件格式为研究对象的数据逆向工程的技术书籍，主要讲解如何分析和研究自定义文件格式的数据结构。本书内容包含反汇编的阅读和理解，数据在计算机中的存储原理，常用媒体格式的解析，加密和解密的识别和分析，数据压缩的特征识别，打包文件格式的识别和游戏窗口化的方法。本书对每一个问题都给出了详细和完整的分析过程，力求用最通俗和简单的方法让读者学会分析和研究自定义文件格式。

本书适合对数据解密、游戏资源提取、软件逆向工程感兴趣的读者以及广大编程爱好者阅读。

揭秘数据解密的关键技术

◆ 编　　著　刘颖东
责任编辑　屈艳莲
执行编辑　黄　焱

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京顺义振华印刷厂印刷

◆ 开本：800×1000　1/16
印张：25.25
字数：500 千字　　2009 年 4 月第 1 版
印数：1－3 500 册　　2009 年 4 月北京第 1 次印刷

ISBN 978-7-115-19670-5/TP

定价：49.00 元

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

感谢

本书献给国内所有热爱修改游戏，以及不求回报无私奉献自己的时间和精力奋战在游戏汉化第一线的朋友们，非常感谢出版社黄焱为本书所做的编辑工作，同时还要感谢我的同学张介溆和孙庚教授，多谢你们审校本书，指出书中的错误并提出宝贵的修改意见，没有你们的工作和帮助本书是不可能完成的。

写作背景

在国内，数据解密的普及程度明显相当不足，在高校里也不太重视这个领域的知识，甚至在市面上的技术书籍也未曾见过一本专门讨论数据解密的书籍，而最接近这个领域的书籍在国内有“看雪论坛”的《加密与解密》，但此书重点是研究代码的逆向分析而非数据的逆向分析。

数据解密在日常的应用非常广泛，包括游戏资源提取、游戏外挂和游戏脚本修改器这些软件的开发都与数据解密形影不离。本书以研究和分析游戏中的资源文件数据结构作为学习数据解密的例子，通过学习这些例子读者能将此技术应用到其他软件领域。

技术是一把双刃剑，本书的写作目的并不是鼓励读者通过学习本书的内容后去搞破坏和谋取利益，本书的写作目的很简单，那就是尽笔者最大的努力和能力告诉读者数据解密是怎么一回事罢了。

本书包含了笔者早年学习数据解密技术的资料、经验和研究成果，读者可以将此书作为一本数据解密的入门书籍来学习。本书并没有什么高深莫测的知识，研究和分析非公开数据结构只需读者具备耐性、时间和精力，如果读者已经具备了这三个条件，那么相信本书能很好地引导您入门。

内容简介

本书一共分为 9 章，每章节的内容组织如下。

第 1 章　走进数据解密

介绍了数据解密常用的方法和汇编语言的入门知识。如果读者已经具备了基本

的汇编语言知识可以略过。数据解密并不一定需要掌握汇编知识，但有时候汇编是找到答案的惟一途径。

第 2 章　识别汇编代码的高级模式

代码逆向分析属于代码逆向工程的范畴。使用代码逆向有时或许并不一定是最好的方法但却是万能的方法，代码逆向分析成功与否很大程度上依赖于反汇编代码分析水平和经验，代码逆向分析主要是要找到反汇编代码和高级语言之间的对应关系，通过观察分析反汇编代码来识别出代码原来的高级模式是代码逆向工程的一项重要技能。

第 3 章　资源文件简介

简要介绍了游戏资源文件的定义和几个游戏资源提取工具，并分析了数据解密在不同领域上的应用。

第 4 章　揭秘文件数据基础——0 和 1

数据解密的基础是深入了解和明白数据在计算机上的存储形式，很多一直使用高级语言开发的人都不清楚自己在程序中定义的数据在计算机上是如何存储的，不知道数据如何存储就不知道如何解密数据，通过本章的学习有助于读者更深刻地理解数据的存储。

第 5 章　媒体数据格式解析

研究和分析游戏的资源文件主要是游戏中的图像文件和模型文件，所以有必要熟悉几种常用的多媒体文件存储格式，掌握图像文件和模型文件中大致要存储哪些数据对分析游戏使用的非公开数据结构的图像和模型格式非常有帮助。

第 6 章　数据加密 VS 数据解密

现在很多游戏都使用了加密技术保护资源文件不被轻易提取，知己知彼才能在数据解密中少走弯路并找到解密的捷径和方法。本章主要讲解了一些游戏中常使用的公开加密算法识别方法和解密自定义加密算法的方法。

第 7 章　神奇的数据压缩算法

游戏几乎不可能不使用压缩技术处理资源文件，压缩过的资源文件对解密者又是一层障碍。本章介绍了压缩资源文件常用的压缩编码并总结了一些压缩编码的识别方法。

第 8 章　分析打包数据存储结构的模式

游戏的资源文件也叫归档、打包文件和封包文件。虽然不同的游戏的资源文件格式也不尽相同，但总的来说目前的打包文件格式可以分为 6 大类。本章主要讲解了 6 种不同类型的打包文件格式，详细分析了 5 个打包文件的过程和方法。

第 9 章　将游戏窗口化

在单机状态下使用 RING3 级的调试器调试全屏游戏首要解决的问题是如何不让游戏全屏显示，因为使用 RING3 级调试器调试全屏游戏时游戏可能会挡住了调试器

的界面或只能以低分辨率显示调试器界面，这样对调试程序很不方便，本章通过实例讲解如何将游戏窗口化。

本书适合哪些读者

本书是针对有基本编程能力的读者，如果你不会基本的编程，那么很遗憾，本书不适合你。本书没有限制读者需要掌握什么编程语言，编程语言都是相通的，由于本书采用了 C 语言和 Intel 80386 汇编语言进行写作，所以读者阅读本书时至少需要能看懂 C 语言语法。对于汇编语言，读者不一定要掌握，但如果不熟悉汇编语言会影响本书中某些章节的阅读。

除此以外：

- 如果你对 C 语言在汇编中的实现机制和汇编语言还原成 C 语言感兴趣；
- 如果你对数据在计算机中的存储形式感兴趣；
- 如果你对加密和压缩算法的识别感兴趣；
- 如果你对游戏的资源文件格式研究和分析感兴趣；
- 如果你对游戏资源的提取感兴趣；
- 如果你对 MOD 游戏感兴趣。

那么恭喜你，本书正适合你。

联系我们

本书由刘颖东编著，同时，参与本书编写工作的还有刘燕祎、周晶、周丰、梅乐夫、房明浩、王亮、门店宏、吴洋、石峰、张圣亮、邱文勋、刘鲲、林远长、董前程、朱飞、汤嘉立、刘变红、刘会灯、张高煜、赵红波、张宪栋、邓志宝、刘坤、刘明辉、李鹏、白学明、步士建等。在此，对以上人员致以诚挚的谢意。

由于笔者水平有限，编写时间仓促，书中难免存在疏漏和不足之处，恳请广大读者提出宝贵意见。本书责任编辑的联系方式是 huangyan@ptpress.com.cn，欢迎来信交流。本书提供了一个专门与作者联系的邮箱 Crazypal3@126.com，读者可以随时与我们联系。

编　者

2009.1

目 录

第 1 章　走进数据解密

本章主要介绍什么是数据解密。汇编语言基础扎实的读者可以跳过这章，直接阅读第 2 章，想详细了解如何将汇编语言翻译成高级语言的读者可以直接阅读 1.6 节，没有汇编基础的读者建议从头读起。

通过本章的学习读者可以掌握以下内容：

- 数据解密是什么；
- 数据解密的方法；
- 万能的汇编语言；
- 通用寄存器；
- 变址寄存器；
- 指针寄存器；
- 标志寄存器；
- 灵活的寻址方式；
- 80386 指令；
- 函数调用约定；
- 字节码。

1.1 数据解密是什么

数据解密没有一个精确的定义，是一个抽象的名词，数据解密不是计算机专有的，凡是将秘密揭开、解开、破解都可以叫做数据解密。

小时候我们很多人都玩过猜谜语的游戏，没错，猜谜语也是数据解密。各行各业都有相对应的数据解密，但总体上数据解密可以分成两部分：科学研究和人工研究。

人工研究是研究人类自己发明或制造的事物。从工业革命开始到现在，人们制造了很多有价值的东西，例如飞机、汽车、炮弹、火箭、计算机等。人们通过数据解密知识和方法揭示各种的设计秘密和技术，从而改造或复制出同样的产品，有的为了纯粹的研究，有的为了商业利益，有的为了国家安全。

1.1.1 代码逆向工程和数据逆向工程

本书讨论的数据解密是针对计算机软件的数据解密，数据解密有一个代名词叫数据

逆向工程，在软件技术里很多人将逆向工程这个名词看作是代码逆向工程，一般来说“软件 = 代码+数据”，代码逆向工程主要是利用反汇编分析出程序代码的执行流程，从而了解和掌握程序里的秘密，而数据逆向工程研究分析的主题更侧重于数据而不是代码。

当然代码逆向工程和数据逆向工程都有互相交叉的部分，代码和数据的逆向分析不是固定不变的，代码逆向工程有时需要利用数据逆向工程的技术，同样数据逆向工程有时也要利用代码逆向工程的技术，只是双方的侧重点不同罢了。

1.2 数据解密的方法

在计算机软件中，数据解密的方法一般分为两种，一种叫黑盒分析法，另一种叫白盒分析法。下面将简要叙述一下黑盒分析法和白盒分析法。

1.2.1 黑盒分析法

黑盒一词源自软件测试中的黑盒，意思是通过观察和分析软件的功能来获取相关的信息。在本书中黑盒分析法的意思是通过观察计算机文件中的二进制码数据来分析获取相关信息的方法。

一般来说如果文件数据没有加密，使用黑盒分析法是最直接、快速获得相关信息的方法，使用黑盒分析法获取文件结构的信息通常需要积累一定的文件分析经验，这些经验来自不断自行研究和分析不同的文件格式，图 1-1 所示为使用黑盒分析法在 winhex 中分析文件的二进制数据。

character.dat

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	50	41	43	4B	B0	06	00	00	20	98	39	00	48	50	00	00	PACK?.. ?.HP..
00000010	69	27	04	00	BA	22	66	00	D9	05	03	00	7B	E1	01	00	i'..?f.?..{?.
00000020	C0	B5	76	00	AA	6E	04	00	2E	8E	00	00	B2	46	80	00	赖v.獰...?.睩e.
00000030	F9	88	04	00	03	97	00	00	CF	4C	AF	00	98	9B	04	00	鵙...?.鐜?楣..
00000040	0C	56	02	00	57	C7	BE	00	FD	AB	05	00	78	AA	00	00	.V..W蕾. ..x?.
00000050	4C	A7	C8	00	73	B8	05	00	0F	E1	00	00	A0	E6	F5	00	L .s?..?.狗?
00000060	D3	EB	05	00	F4	01	02	00	C4	2C	39	01	43	00	07	00	与..?..??9.C...
00000070	F1	8F	03	00	CC	E5	8C	01	C9	A9	09	00	5D	90	00	00	駒..体?嫂..]?.
00000080	DC	B9	E5	01	C5	C4	09	00	5B	31	04	00	FB	81	F1	01	芄?拍..[1..麋?
00000090	14	AD	0B	00	0D	D6	01	00	2F	85	F7	01	76	F8	0C	00	.?..?./珮.v?.
000000A0	E2	FF	00	00	E9	68	36	02	7E	09	0D	00	E8	CE	02	00	?..閎6.~...柿..
000000B0	C3	32	68	02	1A	00	0E	00	58	07	02	00	2D	CC	69	02	?h.....X...-藤.
000000C0	87	73	0F	00	05	BE	02	00	B5	46	79	02	DD	7C	10	00	嘩...?.磲y.韬..

图 1-1　黑盒分析法

1.2.2 白盒分析法

白盒一词源自软件测试中的白盒分析法，意思是分析软件源代码找出相关信息。

在本书中白盒分析法的意思是通过分析程序的指令代码，即反汇编来获取相关信息的方法。

1.2.3 黑盒分析法与白盒分析法的比较

通常如果使用黑盒分析法能获取文件的数据结构就不会使用白盒分析法，因为一般来说使用白盒分析法需要破解人员具备一定的汇编知识和经验，对于新手来说比较困难，但对于有经验的人员来说白盒分析法往往是解决问题的万能方法。很多时候通过黑盒分析法无法得到有用的信息时，惟一的办法就是进行白盒分析了。

白盒分析也就是俗称的反汇编，一般不会对数据进行反汇编，因为对数据进行反汇编是没有意义的，反汇编针对的是使用文件数据的程序。解密者通过分析、跟踪程序的反汇编代码，定位到程序读取或写入数据的代码段位置，通过分析反汇编代码研究文件的数据结构。

现在的程序大多数都使用了强悍的加密壳保护自己，直接反汇编加了壳的程序是没意义的，解密者通常需要自行将加了壳的程序脱壳并修复后才能正常反汇编分析程序，而脱壳这一环节无形中又增加了解密文件数据结构的难度。

如果为了锻炼自身的计算机技能水平，想要深入学习计算机编程，那么学习汇编、看懂反汇编是必经之路，如图 1-2 所示为使用白盒分析法在 OllyICE 中通过反汇编分析程序的指令。

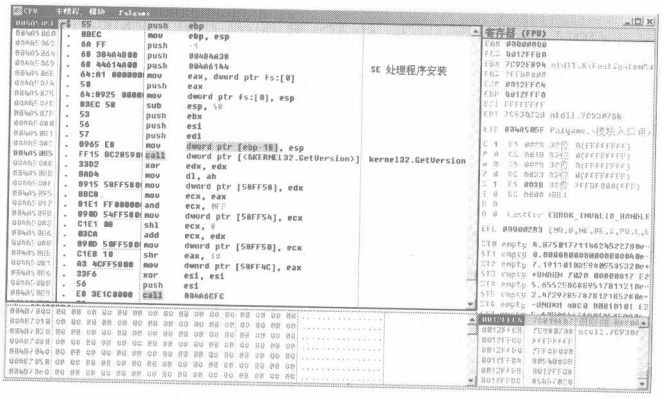


图 1-2 白盒分析法

1.3 万能的汇编语言

很多人都知道汇编语言是一种古老的编程语言，在汇编语言的时代，编程人员

只有汇编语言可选择。

如今可选择的编程语言多不胜数，如命令式语言 Basic、C 和 Pascal，面向对象语言 C++、C#和 Java，脚本语言 Perl、ASP 和 PHP 等。

编程人员可选择的语言余地很大，加上无论从易用性和开发效率上考虑，这些高级编程语言的优势很大。

那么汇编语言是不是可以退出历史的舞台呢?

1.3.1 为什么选择汇编语言

虽然到目前为止已经存在有多种计算机编程语言，但总体上可以分为两大部分，分别是低级语言和高级语言。很多人都喜欢高级语言，而不喜欢低级语言，原因是高级语言提供了丰富的函数和高级的指令，短短几十行的代码就可以创建出一个具有一定功能且能正常运行的程序，相反，低级语言不具备丰富的函数库，也不提供高级的指令，要编写什么程序都要从基层做起，十分麻烦不方便。

即使选择高级语言有千万个理由，但在解密者前面只有一条路可行，那就是选择低级语言——万能的汇编语言。

为什么呢? 答案很明显，因为高级语言只能存在于程序编译之前，程序编译成目标代码后，所有高级语言的语句、概念等一概不存在。一个可执行程序在计算机看来只有 0 和 1 的不同组合，计算机的 CPU 不知道什么是类、什么是对象、什么是模板、什么是组件。一切高级特性在编译成可执行程序后被消灭了（本书讨论的是 IA-32 架构的 Intel CPU，同样适用于 AMD 系列的 CPU。.NET 和 Java 虚拟机不在本书讨论范围内）。

幸运的是读者不需要一定会编写汇编语言，只需看懂即可，这点读者不需要太担心。

1.3.2 16 位和 32 位的 80x86 汇编语言

需要注意的是汇编不是一种语言，不同平台有不同的汇编语言相对应，因为汇编和系统平台相关，所以汇编语言没有移植性。对于 IA-32 架构平台而言，选用的是 32 位的 80386 汇编语言，也就是说本书讨论的操作系统平台是 32 位的，可执行文件的格式也是 32 位而不是 64 位或 16 位的。

实际分析中读者要知道研究的程序是运行在什么平台上，以选择相应的汇编语言，对于 IA-32 架构而言，IA-16 架构的汇编语言原理其实和 IA-32 的汇编语言一样，学习过 16 位的 80x86 汇编语言的读者只需花一点时间就可以转到 32 位的 80386 汇编语言上。

16 位操作系统和 32 位的操作系统的 80x86 汇编语言主要区别如下。

（1）16 位操作系统中的中断调用相当于 32 位操作系统中的 API 调用。16 位操作系统中的段地址和偏移地址在 32 位中消失了，在 32 位操作系统中统一采用平坦的内存地址模式寻址。

（2）16 位操作系统中的程序运行在 RING0 级，也就是说普通程序和操作系统程序运行在同一级别并拥有最高权限，而 32 位操作系统中的程序一般只拥有 RING3 级运行权限，程序的所有操作都受到操作系统控制，若程序要获得 RING0 操作特权只能通过驱动程序实现。

（3）16 位操作系统的可执行文件格式和 32 位操作系统的可执行文件格式不同，在 32 位的 Windows 操作系统中，可执行文件的格式叫 PE 格式，32 位的 Windows 操作系统运行在 CPU 的保护模式之上，而 16 位的系统则运行在 CPU 的实模式上。

1.4 通用寄存器

寄存器是 CPU 中的一个存储单元，寄存器容量一般很小，通常只有 32 位和 64 位，所以寄存器通常用来保存临时的 CPU 数据。

1.4.1 EAX、EBX、ECX 和 EDX 寄存器

IA-32 架构中一共有 4 个 32 位的通用寄存器，用于保存临时数据，它们分别是 EAX、EBX、ECX 和 EDX。

这 4 个 32 位的通用寄存器名字前面都有一个“E”字母，含义是“expand”扩展，这是由于在 16 位的时代，这 4 个通用寄存器的名字是 AX、BX、CX 和 DX，到了 32 位后就在它们的名字前加个“E”来区别是 32 位还是 16 位。

这 4 个 32 位的通用寄存器可以当作 16 位使用，也可以当作 8 位使用。当作 8 位使用时，就将 AX 拆开为 AH 和 AL，AH 中的“H”代表“high”，意思是高位的意思，AL 中的“L”代表“low”，意思是低位的意思。同理，BX、CX 和 DX 可拆开为 BH、BL、CH、CL、DH、DL 来使用。

1.4.2 EAX、EBX、ECX 和 EDX 寄存器的用途

（1）EAX 寄存器：EAX 称为累加器，常用于算数运算、布尔操作、逻辑操作、返回函数结果等。

（2）EBX 寄存器：EBX 称为基址寄存器，常用于存档内存地址。

（3）ECX 寄存器：ECX 称为计数寄存器，常用于存放循环语句的循环次数，字符串操作中也常用。

（4）EDX 寄存器：EDX 称为数据寄存器，常常和 EAX 一起使用。

> 注意：上面所述的 4 个通用寄存器的专门用途不是一成不变的，编译器在编译程序的时候会根据很多因素，例如编译器、编译条件、操作系统等作出相应的改变，读者要知道着手研究的程序是用什么编译器编译，然后针对具体的编译器参考该编译器的说明。

1.5 变址寄存器

顾名思义，变址的含义是内存地址会变动的，也就是说变址寄存器中存放会变动的内存地址。80386 架构中有两个变址寄存器，分别是 ESI 和 EDI。

1.5.1 ESI 和 EDI 寄存器

（1）ESI：ESI 称为源变址寄存器，通常存放要处理的数据的内存地址。

（2）EDI：EDI 称为目的变址寄存器，通常存放处理后的数据的内存地址。

1.5.2 ESI 和 EDI 寄存器的用途

ESI 和 EDI 常用来配合使用完成数据的赋值操作，下面是一个 ESI 和 EDI 配合使用的例子。

```
rep   movs   dword   ptr   [edi],dword   ptr   [esi]
```

上面的指令把 esi 所指向的内存中的内容复制到 edi 所指向的内存中，数据的长度在 ecx 寄存器中指定。

1.6 指针寄存器

80386 的指针寄存器有基址寄存器 EBP、堆栈指针寄存器 ESP 和指令指针寄存器 EIP。读者只需了解基址寄存器 EBP 和堆栈指针寄存器 ESP 即可，指令指针寄存器 EIP 总是指向下一条要执行的指令的地址，一般情况下无需修改 EIP。

1.6.1 EBP 和 ESP 寄存器

（1）EBP：EBP 称为基址寄存器，可作为通用寄存器用于存放操作数，常用来代替堆栈指针访问堆栈中的数据。

（2）ESP：ESP 称为堆栈指针寄存器，不可作为通用寄存器使用，ESP 存放当前堆栈栈顶的地址，一般情况下，ESP 和 EBP 联合使用来访问函数中的参数和局部变量。

1.6.2 EBP 和 ESP 寄存器的用途

EBP 和 ESP 常配合使用完成堆栈的访问，下面是一段常见的堆栈访问的指令。

```
push    ebp
mov     ebp, esp
sub     esp, 78
push    esi
push    edi
cmp     dword ptr [ebp+8], 0
```

1.7 标志寄存器

在 80386 中有一个 32 位的标志寄存器 EFLAGS。在汇编语言中，标志寄存器 EFLAGS 是实现条件判断和逻辑判断的一种机制。许多指令执行后都会更新标志寄存器 EFLAGS，然后接下来的指令就可以根据当前标志寄存器 EFLAGS 而作出相对应的操作。

1.7.1 EFLAGS 寄存器

标志寄存器 EFLAGS 一共有 32 位，在这 32 位中大部分是保留和给编写操作系统的人用的，一般情况下只需知道 32 位的低 16 位中的 8 位即可，图 1-3 列出了标志寄存器 EFLAGS 中需要了解的 8 个位的位置。

- OF（Overflow Flag）：溢出标志，溢出时为 1，否则置 0。
- DF（Direction Flag）：方向标志，在串处理指令中控制信息的方向。
- IF（Interrupt Flag）：中断标志。
- AF（Auxiliary carry Flag）：辅助进位标志，有进位时置 1，否则置 0。
- ZF（Zero Flag）：零标志，运算结果为 0 时 ZF 位置 1，否则置 0。