基于 Windows 内核层的 Anti-Rootkits 研究及其实现

地理信息系统 05　肖锐

指导教师　李冬梅

摘要

Windows 操作系统是目前主流的操作系统，基于这个平台下的各种程序软件层出不穷，相应的木马病毒也在不断的进步。为了对抗杀毒软件，这些木马病毒企图霸占电脑主权，更进一步的扩展恶意行为。这些恶意程序往往具备对抗当今主流杀毒软件的能力，给用户带来了不可估量的损失。

研发探究这些病毒木马的原理及其行为，制作出相应的安全工具来对抗它们，显得十分必要。然而由于 Windows 操作系统是不开源的，这对研究其内部的原理增加了一定的难度，制作相应的 Anti-Rootkits 工具也并非易事。但正是由于这些原因，使得对抗当前流行的病毒木马变得更加具有价值和挑战性。

本文基于这个背景，设计并实现了一个检测木马病毒的工具。由于使用了比较先进的 Windows 内核技术，挖掘未被微软公开的细节，将这些知识运用于对抗病毒，因此，经过实验测试，此工具在某种程度上优于现有的某些工具，能够成功检测出病毒隐藏的文件、注册表以及它们修改内核的痕迹。

关键词：Rootkits, NTFS，注册表，内核钩子

Research and Implementation of Anti-Rootkits Based on Windows Kernel

Geographical information system 05 Xiao Rui

Supervisor Li Dong-mei

Abstract

Windows operating system is currently the mainstream operating system, many kinds of softwares were developed rapidly based on it, and the Trojan horse virus has been progress either. In order to counter the anti-virus software, the Trojan virus attempting to occupy the sovereignty of computer, further, expansion of malicious acts. These malicious programs often have the ability to fight with today's mainstream antivirus software, and then give users an immeasurable loss.

It is necessary to research the principles of these viruses, and explore their behaviors, and then try to produce the corresponding security tools to counter them. However, as the Windows operating system is not open source, it is more difficult to study the principles of its internal, and the production of corresponding anti-rootkits tools is not an easy task. It is for these reasons, the current fight against the Rootkits has become more valuable and challenging.

This paper designs a tool to detect Trojan horse virus based on this background. Because of using some more advanced technology of Windows kernel which is unpublished by Microsoft, So, after experimental testing, to a certain extent this tool is superior to most of the existing tools. It can successfully detect viruses hidden files, registry, as well as traces of their modified kernel.

Key words : Rootkits，NTFS，register table，kernel hook

目录

1 绪论

1.1 研究背景与意义

Windows 操作系统是目前主流的操作系统，基于这个平台的各种软件层出不穷，同时也出现了各种类型的计算机病毒和木马程序。而且更为严重的是 Rootkits 的诞生，给系统安全带来了极大的破坏性。

计算机病毒的产生是计算机技术和以计算机为核心的社会信息化进程发展到一定阶段的必然产物，是高技术犯罪，具有瞬时性、动态性和随机性。不易取证，风险小破坏大，从而刺激了犯罪意识和犯罪活动，是某些人恶作剧和报复心态在计算机应用领域的表现。

最初，某些程序员为了展示自己强大精湛的电脑技术，制作了恶作剧程序。这些程序不符合正常软件的常规逻辑，具有病毒木马的雏形。而后，在巨大经济利益的驱使下，逐渐演变成现今破坏重要数据，劫持网络通讯，盗取用户网络银行、游戏账号密码，窥探用户隐私攫取暴利的恶意工具。而且为了提高生存能力，企图在受害者电脑内获得一席之地，霸占电脑主权，更进一步的扩展恶意行为，这些恶意程序更是采用底层的 Windows 内核技术，来对抗当今主流的杀毒软件。比如隐藏自己的进程、文件、注册表信息，让用户很难发现它们的存在。而它们依然在后台静悄悄的窥探着用户的一切操作，盗取用户的个人重要信息。这是十分可怕的，也给用户带来了不可估量的损失。

研发探究这些病毒木马的原理及其行为，制作出相应的安全工具来对抗它们，显得十分的必要。不像 Unix 等开源系统，Windows 操作系统是不开源的，这对研究其内部的原理增加了一定的难度，制作相应的 Anti-Rootkits 工具也并非易事。但正是由于这些原因，使得对抗当前流行的病毒木马变得更加具有价值和挑战性。

这里所说的 Rootkits，是指那些病毒木马制造者们运用各种编程语言写的代码，经过编译器编译而形成的 PE 可执行文件，它们是程序，但其行为是恶意的，区别于正常的软件。这些 Rootkits 就像一层铠甲，将自身及指定的文件保护起来，使其它软件无法发现、修改或删除这些文件。

目前国外在操作系统安全领域领先国内很多，对 Windows 操作系统底层研究较深的国家比如美国、俄罗斯、德国。他们所制作的一些 Anti-Rootkis 安全工具比如 Rku、Gmer 的实用性很高，能发现并清除当前一些比较恶劣的病毒木马。但这些东西所用到的先进技术很多都是未公开的，而且它们自身也存在功能上的缺陷。在 Windows 内核领域，运用这些挖掘来的未公开技术来对抗病毒木马，效果是非常明显的。国内也有一部分大师级人物自己研制了工具，例如 IceSword(冰刃)、狙剑(SpineSword)、XueTr。无论是国内还是国外，这些工具的共同点都是运用了许多 Windows 系统底层

的未公开技术，来发现清除流行的 Rootkits。但由于国内软件研发的大环境，开源是一件很困难的事情，故很多先进技术都是没有公开探讨交流的，这就使得我们自己必须去探究挖掘 Windows 内核的知识，运用到自己的程序中去。

现在的Rootkits技术含量越来越高，随着Windows内核技术文档的逐渐被挖掘披露，给了病毒木马制造者们更多的资料来开发高级新型的恶意程序来抗衡杀毒软件和Anti-Rootkits工具。所以在反病毒反木马领域，也必然会不断的进步。技术是一把双刃剑，病毒木马和安全工具的对抗是没有尽头的，只会随着技术的不断更新发展变得更加白日化[1]。

1.2 本文的主要研究内容

本文的主要研究内容包括如下三个方面：

（1）从 NTFS 磁盘的基础知识展开，然后逐步深入的描述如何解析文件，检测隐藏的数据；

（2）从注册表 Hive 文件的格式开始分析，研究如何将复杂的数据结构转换成可视化的信息；

（3）研究 Inline Hook 的原理以及如何检测这种类型的 Hook。

1.3 本文的组织

本文组织如下：

第一章 绪论，介绍了本文的理论基础——阐述了该研究的应用场景与实际意义。

第二章 Windows 下木马病毒相关知识，介绍了木马病毒的基础理论概念及其危性。

第三章 算法的设计与实现，详述了本系统各个核心算法的设计与实现思路以及最终的分析计算结果展示。

第四章 结束语，对论文所做的工作做了小结，并指出论文进一步待完善和改进之处。

2 病毒木马等相关理论知识

2.1 Rootkits的基本概念

Rootkit 出现于二十世纪 90 年代初，是集合了系统间谍程序、病毒以及木马等特性的一种恶意程序，用来隐藏木马程序文件、进程和注册表。在 1994 年 2 月的一篇安全咨询报告中首先使用了 Rootkit 这个名词。这篇安全咨询就是 CERT-CC 的 CA-1994-01，题目是 Ongoing Network Monitoring Attacks。它利用操作系统的模块化技术，作为系统内核的一部分运行，并且可以使常规系统分析工具失效，无法捕捉到任何蛛丝马迹。

从出现至今，Rootkit 的技术发展非常迅速，应用越来越广泛，检测难度也越来越大。针对 Rootkit 使用驱动技术的特点，对 Rootkit 的检测和清除需要使用更高水准的驱动级编程技术，深入系统内核进行分析判断。对 Rootkit 的检测和清除技术是当前国际反病毒行业的前沿技术。

2.1.1 Rootkits的发展史

在 DOS 隐形病毒出现大约 10 年之后，Windows Rootkits 首次出现。其编写者 Greg Hoglund 第一个运用病毒技术，做出了在 Windows 操作系统下隐藏文件数据的工具。之后人们逐渐弄清楚了如何开发 Rootkits 技术，开始把这些技术运用到大量的恶意程序之中。然而，在最初的阶段，恶意 Rootkits 的数量及其应用的方式相对来说还是小规模的。

2000 年，俄罗斯的程序员制作了名为 He4hook 的工具，用于在内核模式下隐藏文件，虽然它不是恶意的，但其提供的功能和 Rootkits 具有很大的相关度；2002 年，Hacker Defender 被制作出来，它比 He4hook 要强大很多，能够隐藏注册表、进程、文件；2003 年，名为 Vanquish 的工具更具备了记录键盘密码的功能；这些东西的发表使得大家关注的焦点发生了改变，开始更多的研究最新的 Rootkits 技术。2003 年出现的 Haxdoor 已完全成为了 Roootkits，它隐匿于操作系统中，抹掉了自己的一切痕迹。2004 年出现的 FU Rootikit，首次采用 DKOM 的方式修改系统内核结构，将自己的进程成功隐藏。2005 年，80%的 Rootkits 是 HacDef 和 Haxdoor 的变种。随后，Rootkits 融入了多种 Windows 内核技术，让自己更难于被发现。到了 2006 年，网络蠕虫如 Bagle，间谍软件如 Goldun，邮件病毒如 Rustock 等都开始运用 Rootkits 技术进行大肆传播。

从 2005 年开始，Rootkits 技术开始被广泛应用，媒体的注意力也被吸引过来，并进一步发现这些技术不仅仅只是在恶意程序中用到，而在一些商业产品中也出现了。最典型的例子就是发生在 2006 年的 Sony DRM 丑闻事件。索尼公司在其光盘上偷偷放了一个 Rootkits 程序，为了防止其 CD

被恶意拷贝。著名的赛门特克公司也曾经在他们的产品中安装过 Rootkits 程序，用来隐藏自己的文件和目录，防止被用户看到。而且卡巴斯基杀毒软件安全厂商竟然运用 NTFS 系统的特性，将自身的数据隐藏在数据流中。用户不用一些特定的工具，根本发现不了它。这些 Rootkits 技术的使用，让普通的电脑用户感到一阵恐慌。

然而，病毒技术和反病毒技术总是形影不离的。Rootkits 的出现，给了杀毒软件等安全厂商巨大的压力，于是这些公司开始在自己的产品中添加检测 Rootkits 的功能，很多个人程序员也开始开发对抗 Rootkis 的工具。于是出现了比较强大的 Anti-Rootkits 工具，比如 Gmer 和 Rootkit Unhooker，它们能够快速全面的检测并清除系统可能存在的 Rootkits 程序。如今，网络上已经有近 20 种检测 Rootkits 的工具，他们各自运用了各种不同的方法去检测 Rootkits 的存在。然而，对抗总是存在的，最近一个叫蓝色药丸的程序，运用了先进的技术，导致所有的检测工具都无法发现它的存在。

最近，研究者们又把目光瞄向硬件虚拟化，试图在 Windows 操作系统启动之前获得对系统的控制权，这样 Rootkits 程序就不会被检测到了。到了 2008 年，一些新的恶意软件运用新的 Rootkits 技术来感染系统的引导扇区，比如 Sinowal、Mebroot、StealthMBR，而很多安全软件对其束手无策。

其中最厉害的一个 Rootkits 是在 2006 年时产生的，名为 Rustock.c。然而一年半以后，Dr. Web 厂商的研究员才发现它的存在。此程序的不可检测性和隐蔽性展现了 Rootkits 技术的可怕性。

2.1.2 Rootkits的危害性

Rootkits 由于运用了大量先进的 Windows 内核技术，所以很难被检测清除掉。于是攻击者就可以堂而皇之的做一些破坏行为。比如破坏重要的磁盘数据，盗窃用户的网络银行密码、游戏账号，进行网络欺骗攻击。2005 年，流氓软件大行其道，为了防止被杀毒软件或流氓软件卸载工具发现，采用了病毒常用的 Rootkits 技术进行自我保护，强行霸占用户电脑。比如臭名昭著的“中文上网”。由于“中文上网”取得了巨大的经济收益，促使一些其它的厂商也推出了具有同质化功能的插件程序。为了争夺市场，提供“中文上网”业务的公司之间开始“互杀”，软件安装后会试图删除竞争对手的插件，将用户的计算机变成战场，甚至一些厂商为此对簿公堂。由于插件间的恶性竞争，在此过程中往往造成用户计算机频繁死机、蓝屏。这样的事情遭殃的不仅仅是普通的电脑用户，还破坏了干净的网络环境，造成了极其恶劣的影响。

2.2 木马病毒的基本原理

木马病毒的目的即是为了留在受害者的操作系统中，不断复制自己，进行传播，保证自己的生

存不会受到威胁，盗取有用的信息。为了达到这些目的，它们会表现出异于正常软件的行为，于是围绕进程，文件，注册表展开了一些操作[2,3]。

（1）进程和线程

进程是指在系统中正在运行的一个应用程序，是具有一定独立功能的程序关于某个数据集合上的一次运行活动，是系统进行资源分配和调度的一个独立单位；线程是进程的一个实体，是 CPU 调度和分派的基本单位系统分配处理器时间资源的基本单元，或者说进程之内独立执行的一个单元。对于操作系统而言，其调度单元是线程。一个进程至少包括一个线程，通常将该线程称为主线程。一个进程从主线程的执行开始进而创建一个或多个附加线程，就是所谓基于多线程的多任务。

进程是执行程序的实例。例如，当你运行记事本程序时，你就创建了一个用来容纳组成 Notepad.exe 的代码及其所需调用动态链接库的进程。每个进程均运行在其专用且受保护的地址空间内。因此，如果你同时运行记事本的两个拷贝，该程序正在使用的数据在各自实例中是彼此独立的。在记事本的一个拷贝中将无法看到该程序的第二个实例打开的数据。打个比方，以沙箱为例进行阐述。一个进程就好比一个沙箱，线程就如同沙箱中的孩子们。孩子们在沙箱子中跑来跑去，并且可能将沙子攘到别的孩子眼中，他们会互相踢打或撕咬。但是，这些沙箱略有不同之处就在于每个沙箱完全由墙壁和顶棚封闭起来，无论箱中的孩子如何狠命地攘沙，他们也不会影响到其它沙箱中的其他孩子。因此，每个进程就象一个被保护起来的沙箱。未经许可，无人可以进出。

一般来说，木马病毒要运行，是一段 PE 可执行文件被系统加载进内存，然后得到执行权限。此 PE 文件可以是 EXE、DLL、SYS 等不同的格式。若是 EXE 程序，就具备了进程；若是 DLL 程序，会被动态加载到某进程的虚拟地址空间中，得到执行，这就是线程； 若是 SYS 程序，即是驱动程序，运行在 R0 中，拥有很大的权限。一旦木马病毒的进程、线程得到执行，便会按照程序指定的逻辑展开破坏行为。我们打开操作系统自带的任务管理器（快捷键：CTRL+SHIFT+ESC），可以在进程栏中看到系统当前的进程列表。这些就是正在运行的可执行程序。我们可以通过任务管理器提供的“结束进程”功能，来停止指定的进程。但是对付木马病毒就没那么简单了，为了保证自己的程序得到持续执行，它们会想方设法的保护住自己的进程、线程，防止被其他程序结束、暂停、挂起、注入、破坏等操作。

（2）文件

木马病毒的实体是在磁盘上的二进制文件。其要获得执行，就必须由操作系统将它们的文件加载进内存。正常情况下，用户可以选择是否有必要保留当前程序的文件，若不需要则可以放进回收站，或者直接删除掉。但是木马病毒会通过更改文件名、文件属性，修改文件图标，挂钩 API 等方式隐藏自己的文件，让普通用户找不到它们。而实际上它们的文件依然存在于物理磁盘上。即使用

户用一些高级工具能发现这些隐藏的文件，但要操作它们也不是一件容易的事。木马病毒会保护自己的文件不被删除，不被重新命名，甚至不被修改。

而且，木马病毒还会主动攻击正常程序，修改破坏它们的文件，造成非常巨大的破坏。比如一些常见的感染性病毒，一旦得到执行，便疯狂的感染硬盘上的文件，将自身添加到这些正常的文件中，下次这些被感染的文件运行时，就会再次激活病毒了。对于这种感染变异型病毒，修复它们对系统造成的破坏也绝非易事[4]。

（3）注册表

Windows 注册表是帮助 Windows 控制硬件、软件、用户环境和 Windows 界面的一套数据文件，包含在 Windows 目录下两个文件 System.dat 和 User.dat 里，通过 Windows 目录下的 Regedit.exe 程序可以存取注册表数据库。它是一个庞大的数据库，用来存储计算机软硬件的各种配置数据，针对 32 位硬件、驱动程序和应用设计的。注册表最初被设计为一个应用程序的数据文件相关参考文件，最后扩展成对于 32 位操作系统和应用程序包括了所有功能下的程序。它是一套控制操作系统外表和如何响应外来事件工作的文件。注册表因为它的目的和性质变的很复杂，它被设计为专门为 32 位应用程序工作，文件的大小被限制在大约 40MB。

一个用户准备运行一个应用程序，注册表提供应用程序信息给操作系统，这样应用程序可以被找到，正确数据文件的位置被规定，其他设置也都可以被使用。注册表保存关于缺省数据和辅助文件的位置信息、菜单、按钮条、窗口状态和其他可选项。它同样也保存了安装信息（比如说日期），安装软件的用户，软件版本号和日期，序列号等。根据安装软件的不同，它包括的信息也不同。

木马病毒要获得运行机会，往往会在注册表中做手脚，增加自启动项。这样操作系统在下次重新启动时，便会自动加载木马病毒了。正常情况下，用户对注册表有完全控制权，能够自行决定是否保留当前程序的启动项。但是木马病毒是不会让他人轻易更改自己的启动项的。它们会禁止用户打开 Windows 系统自带的注册表编辑器，隐藏自己的键值，保护其不被修改，或者不断的回写自己的键值。总之是运用各种手段保证自己在注册表中的数据，以便能更有效的获得再次执行的机会。

正常的软件也会在注册表中写入自己的数据，比如常见的杀毒软件。木马病毒为了对抗这些安全工具，会破坏它们的注册表键值，导致一些自我保护较弱的安全工具失效。从而在一定程序上削弱了杀毒软件的功能[5]。

3 系统各算法的设计与实现

3.1 系统功能概述

系统主要功能包括以下三个方面：
（1）NTFS 磁盘解析
（2）注册表解析
（3）内核钩子的检测

3.2 NTFS磁盘解析

3.2.1 磁盘的基本知识

从计算机系统的结构来看，存储器分为内存储器和外存储器两大类。内存储器与CPU直接联系，负责各种软件的运行。外存储器包括软盘、硬盘、光盘、磁带机等。硬盘和软盘很相似，它们的工作原理大致相同，不同的是软盘与软盘驱动器是分开的，而硬盘与硬盘驱动器却是装在一起。另外，在使用时，二者速度差异很大。硬盘主要由：盘片，磁头，盘片转轴及控制电机，磁头控制器，数据转换器，接口，缓存等几个部分组成[6]。

硬盘也称为温盘，其盘片及磁头均密封在金属盒中，构成一体，不可拆卸，为现代计算机系统提供了大容量的二级存储。硬盘的盘片是表面镀有磁粉的铝合金片或高强度玻璃片，一般在 1—8 片之间，甚至更多。不工作时，硬盘的磁头贴在盘片上，或停放在盘片外磁头专用停放架上，而在读写期间，硬盘的磁头不接触盘片，这种结构提高了硬盘的可靠性和耐磨性。其通过在盘片上的磁性纪录来存储信息。读-写磁头在每个盘片表面上方快速移动，磁头固定在磁臂上，这样磁臂就可以将所有磁头作为单元一起移动。盘片表面逻辑上分为圆形的磁道，磁道又可以分为扇区。在磁臂同一个位置的磁道集合构成了一个柱面。一个磁盘驱动器上可能有数以千计的柱面，而每个磁道又可能有几百个扇区。普通磁盘驱动器的容量以 G 计。磁盘在使用时驱动器马达高速旋转，大多数每秒旋转 60-200 次。磁盘速度分为两个部分，转移速度是指数据流从驱动器到达计算机内存的时间。寻址时间，也叫随机访问时间有两部分组成，寻找时间即移动磁臂到目的柱面的时间，和旋转时间，即目的扇区旋转到磁头的时间。典型的磁盘每秒能够传输几 M 数据，并且它们的寻找时间和旋转时间在几个毫秒。由于磁头在极薄的空气垫高速移动，就有接触盘面的危险。尽管盘片表面涂了很薄的保护层，有时磁头难免会损坏盘片表面。现在的磁盘以一维的逻辑块进行编址。逻辑块是传输的最小单位，一个逻辑块大小通常是 512 字节，但也有不同的，低级格式化可以选择不同的逻辑块大小，比如 1024 字节。一维的逻辑块序列顺序映射到磁盘的扇区.扇区 0 位于最外层柱面的第一个磁

道上。有两种设备：恒定线速度和恒定角速度。前者里层和外层每个磁道的位密度是一致的，为了保持磁头一致的数据传输率，磁盘在磁头由外至内时要加快旋转速度，而后者的磁道位密度从内层向外层递减。

3.2.2 NTFS的特性

NTFS是 Windows NT 操作环境和 Windows NT 高级服务器网络操作系统环的文件系统。其目标是提供可靠性，通过可恢复能力（事件跟踪）和热定位的容错特征实现；增加功能性的一个平台；对POSIX需求的支持；消除FAT和 HPFS 文件系统中的限制。NTFS 提供长文件名、数据保护和恢复，并通过目录和文件许可实现安全性。NTFS 支持大硬盘和在多个硬盘上存储文件(称为跨越分区)。提供内置安全性特征，它控制文件的隶属关系和访问。从 DOS 或其他操作系统上不能直接访问 NTFS 分区上的文件。从Win 2000 开始采用了更新版本的NTFS文件系统NTFS 5.0，它的推出使得用户不但可以像Win 9X那样方便快捷地操作和管理计算机，同时也可享受到NTFS所带来的系统安全性。NTFS的优点如下：

（1）具备错误预警的文件系统

在 NTFS 分区中，最开始的 16 个扇区是分区引导扇区，其中保存着分区引导代码，接着就是主文件表(Master File Table，以下简称 MFT)，但如果它所在的磁盘扇区恰好出现损坏，NTFS 文件系统会比较智能地将 MFT 换到硬盘的其他扇区，保证了文件系统的正常使用，也就是保证了 Windows 的正常运行。而以前的 FAT16 和 FAT32 的 FAT（文件分配表）则只能固定在分区引导扇区的后面，一旦遇到扇区损坏，那么整个文件系统就要瘫痪。

（2）文件读取速度更高效

NTFS 在文件处理速度上比 FAT32 大有提升。在 NTFS 文件系统中，一切东西都是一种属性，就连文件内容也是一种属性。这些属性的列表不是固定的，可以随时增加。 NTFS 文件系统中的文件属性可以分成两种：常驻属性和非常驻属性，常驻属性直接保存在 MFT 中，像文件名和相关时间信息（例如创建时间、修改时间等）永远属于常驻属性，非常驻属性则保存在 MFT 之外，但会使用一种复杂的索引方式来进行指示。如果文件或文件夹小于 1500 字节（其实我们的电脑中有相当多这样大小的文件或文件夹），那么它们的所有属性，包括内容都会常驻在 MFT 中，而 MFT 是 Windows 一启动就会载入到内存中的，这样当你查看这些文件或文件夹时，其实它们的内容早已在缓存中了，自然大大提高了文件和文件夹的访问速度。

（3）磁盘自我修复功能

NTFS 利用一种“自我疗伤”的系统，可以对硬盘上的逻辑错误和物理错误进行自动侦测和修复。每次读写时，它都会检查扇区正确与否。当读取时发现错误，NTFS 会报告这个错误;当向磁盘写文件时发现错误，NTFS 将会十分智能地换一个完好位置存储数据，操作不会受到任何影响。在这两种情况下，NTFS 都会在坏扇区上作标记，以防今后被使用。这种工作模式可以使磁盘错误可以较早地被发现，避免灾难性的事故发生。

（4）“防灾赈灾”的事件日志功能

在 NTFS 文件系统中，任何操作都可以被看成是一个“事件”。比如将一个文件从 C 盘复制到 D 盘，整个复制过程就是一个事件。事件日志一直监督着整个操作，当它在目标地——D 盘发现了完整文件，就会记录下一个“已完成”的标记。假如复制中途断电，事件日志中就不会记录“已完成”，NTFS 可以在来电后重新完成刚才的事件。事件日志的作用不在于它能挽回损失，而在于它监督所有事件，从而让系统永远知道完成了哪些任务，那些任务还没有完成，保证系统不会因为断电等突发事件发生紊乱，最大程度降低了破坏性。

3.2.3 解析NTFS的基础

（1）磁盘数据的物理存储

磁盘经过格式化后，系统将磁盘片划分成磁面、磁道和扇区。每个磁面是拥有相同的磁道数，每个磁道上含有相同的扇区数，每个扇区包含 512 字节的数据信息。一个或若干个扇区组织成一个“簇”。簇是文件系统进行数据读写操作的最小单位，每个簇占用的扇区数由 Windows 版本和磁盘类型决定（通常情况下一个簇等于 8 个扇区），一个文件在磁盘上分布可能不是连续的，会占用一个或多个簇。从节约硬盘空间角度讲，簇越小越好，但相反文件存取效率会降低。

硬盘的物理扇区是用“柱面、磁头、扇区”3 个参数来表示硬盘上的某一区域。逻辑扇区是物理扇区的一组连续数字的编号，由于采用了线性寻址方式（即以逻辑扇区号为单位进行寻址），硬盘在进行数据读写操作时，系统使用逻辑扇区数，这样更简单高效。

NTFS 文件系统使用逻辑簇号和虚拟簇号对卷进行管理。只有知道簇的大小、LCN 号、NTFS 卷在物理磁盘上的起始扇区，就可以读簇进行定位，找到了簇也就找到了指定的文件的数据信息。就可以对文件进行一系列的操作。

（2）硬盘的主引导扇区

硬盘的引导扇区即是硬盘的第一个扇区，由以下四部分组成：主引导记录 MBR、出错信息数据区、分区表 DPT、结束标志字。

MBR 的作用是检查分区表是否完好，查找可以引导的活到分区，将活动分区的第一个逻辑扇区

内容 DBR 载入内存。操作系统为了便于用户对磁盘的管理，加入了磁盘分区的概念，将一个磁盘逻辑划分为若干块。DPT 的大小为 64 字节，以 16 个字节为分区表项单位描述一个分区的属性，最多有 4 个基本分区，若干个扩展分区。在操作系统 62 个保留扇区之后是第一个系统（通常为 C 盘）的引导扇区，位于 0 柱面 1 磁头 1 扇区，即是 DBR，大小为 512 字节。它包含一个引导程序和一个被称为 BPB 的本分区参数记录表，如图 3.1 所示。其中 BPB 里面存储着磁盘容量和几何结构变量，如本分区的起始扇区、文件存储格式、根目录大小、分配单元大小等。 DBR 的作用是判断本活动分区根目录前两个文件是否是操作系统的引导文件。若存在且这是个 NTFS 文件系统的引导扇区，会把 NTLDR 读入内存，将控制权交给该文件。

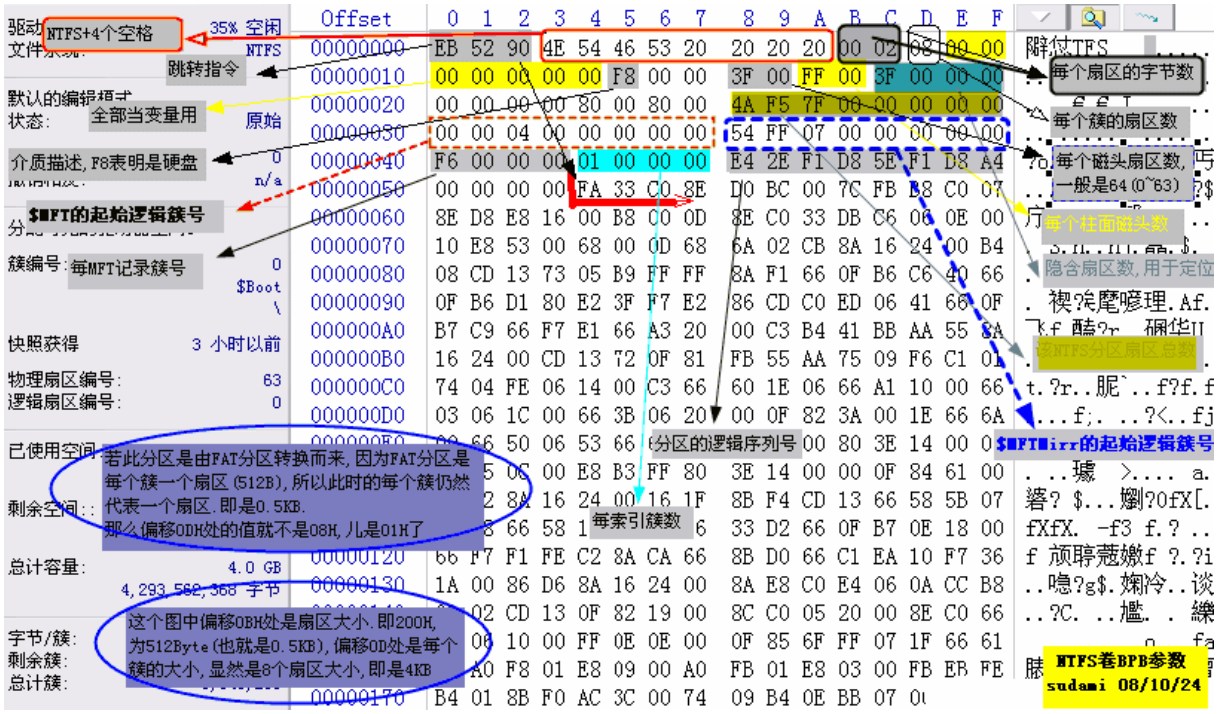

图 3.1 NTFS 分区的 BPB 参数信息

Fig.3.1 Information of BPB in NTFS partition

NTLDR 后是主控文件表区域。NTFS 系统中，一切都用属性来衡量，每个属性在 MFT 中都有记载，当一个属性太大，一个 MFT 记录不了便用多个 MFT 进行记录。当文件很小，可能该文件的所有属性都包含中 MFT 中，甚至是该文件的数据, 这样就节省了不少磁盘空间，如图 3.2 所示。

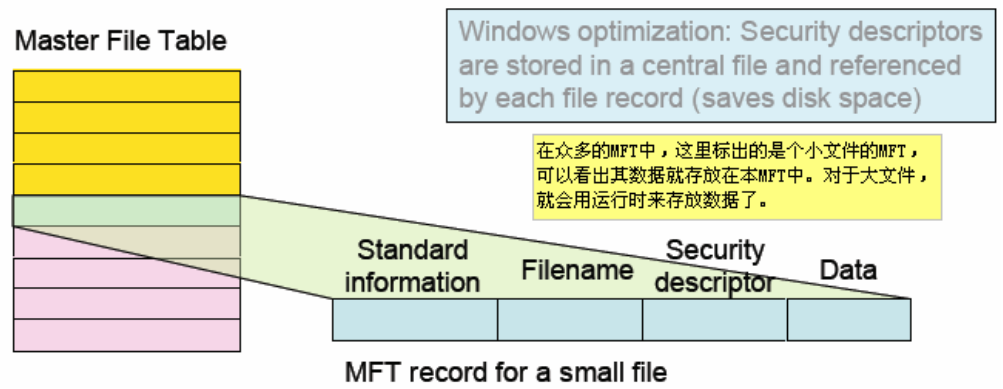

图 3.2 一个小文件的 MFT

Fig.3.2 MTF record for a small file

MFT 中还记录了一些非常重要的系统数据，即是元数据文件，它包括用于文件定位和恢复的数据结构、引导程序数据及整个卷的分配位图等信息。这些数据被系统以文件的方式进行管理，用户是不可访问的，在文件夹视图中是无法看到它们的。这样的元数据有 16 个，文件名的第一个字符都是“$”，包括 MFT 本身（$MFT）、MFT 镜像（MFTMirr）、日志文件（$LogFile）、卷文件（$Volume）、属性定义表（$AttrDef）、根目录（$\）、位图文件（$Bitmap）、引导文件（$Boot）、坏簇文件（$BadClus）、安全文件（$Secure）、大写文件（$UpCase）、扩展元数据目录（$Extended metadata directory）、重解析文件（$Extend\$Reparse）、变更日志文件（$Extend\$UsnJrnl）、配额管理文件（$Extend\$Quota）、对象 ID 文件（$Extend\$ObjID）等。在这 16 项以后就是用户建立的文件和文件夹了。

图 3.3 NTFS 分区的整体结构分布

Fig.3.3 Distribution of the overall structure in NTFS partition

每个 MFT 占据的磁盘空间一般为 1KB，即 2 个扇区的大小，NTFS 文件系统分配给 MFT 的区域占据了磁盘空间的 12%，剩余空间用来存放用户的文件，此区域被称为文件存储区，其中包含一个 MFT 前几项元数据文件的备份，如图 3.3 所示。当 NTFS 访问某个卷时，它需要“挂接”此卷，即查看引导文件，找到 MFT 的物理磁盘地址，从文件记录的数据属性中获得 VCN 到 LCN 的映射信息，并存储到内存中。这个映射信息定位了 MFT 的运行在磁盘上的位置。接着，NTFS 再打开几个元数据文件的 MFT 记录，并打开这些文件，然后用户就可以访问当前卷了。

（3）文件类型属性

在一个 NTFS 卷中文件的所有信息，包括文件数据在内，都被认为是文件的属性，构成该属性的实际数据被称为“流”。当一个文件很小时，其所有属性和属性值可存放在 MFT 的文件记录中，当属性值能直接存放在 MFT 中时，被称为“常驻属性”。只存储在运行中而不是在 MFT 文件记录中的属性被称为“非常驻属性”，其通过 VCN-LCN 之间的映射关系来记录运行。

在 MFT 记录中的每个属性都有一个属性头，总共有 4 中属性：常驻无属性名属性、常驻有属性名属性、非常驻无属性名属性、非常驻有属性名属性。我们可以根据不同类型的属性头，区别不同类型的属性，进而找到对应的文件信息分布。一个 MFT 包含的属性种类很多，从 10H 到 F0H 到 00H，每种属性类型都有着重要的意义。其中和 NTFS 解析文件相关的是 10H 属性（$STANDARD_INFORMATION）、20H 属性（$ATTRIBUTE_LIST）、30H 属性（$FILE_NAME）、80H 属性（$DATA）、90H 属性（$INDEX_ROOT）、AOH 属性（$INDEX_ALLOCATION）、B0H 属性（$BITMAP）。下面对这几个属性进行重点分析：

10H 是文件的标准信息，包含基本的文件属性，如只读、系统、隐藏等。解析 NFTS 时，可以从当前文件的此属性中获取文件的更新时间等，如图 3.4 所示。

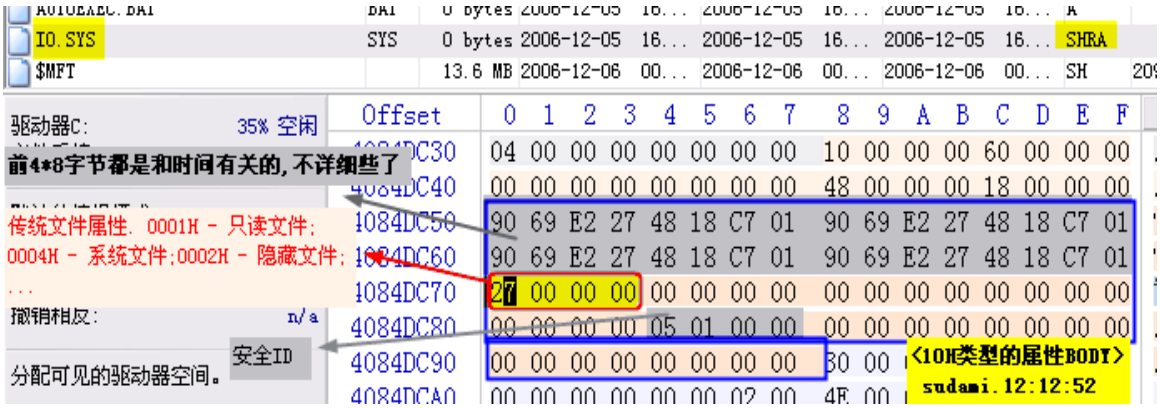

图 3.4 0x10 属性

Fig.3.4　Attribute of 0x10

20H 是属性列表属性，当一个文件需要多个 MFT 文件记录时，用它来描述文件的属性列表。只有当遇到较为特殊的文件（比如有很多硬链接，有很多碎片，很复杂的安全描述，有很多的数据流）时，才会去分析这个属性，当然它是解析 NTFS 时必须关注的属性。

30H 是属性名属性，用来存储文件名，它总是常驻属性。解析文件时，文件名就是从这个属性中获取的。删除磁盘文件时，为了将磁盘文件删的更彻底，也会清除该属性中的文件名处的内容，如图 3.5 所示。

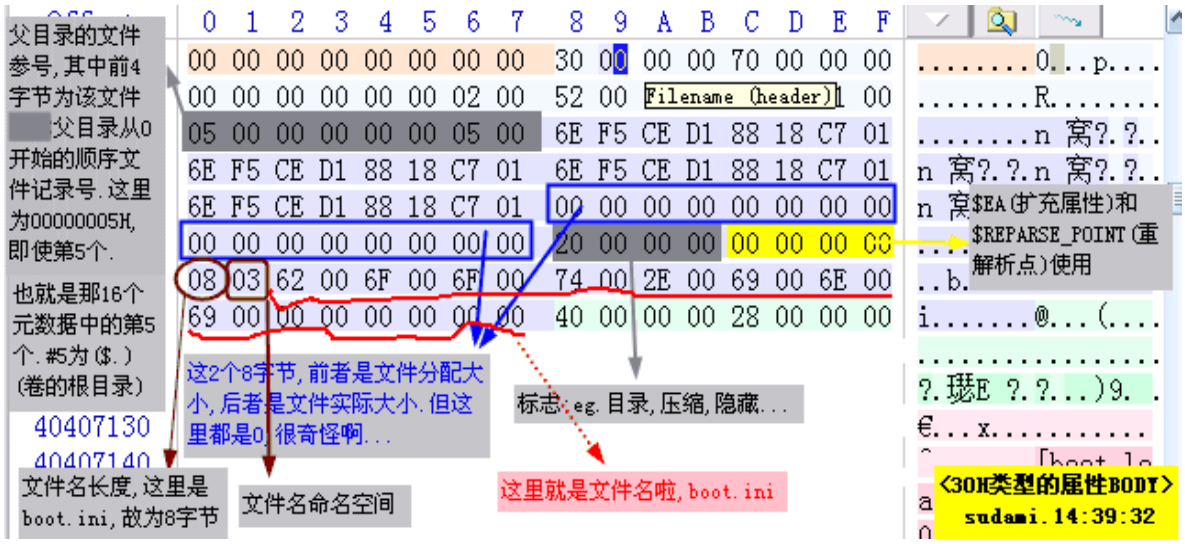

图 3.5 0x30 属性

Fig.3.5 Attribute of 0x30

80H 属性容纳了文件数据。此种属性有 2 个分类：常驻和非常驻。当文件属性能够在 MFT 中完全存储下来而不需要存储到其他运行时中时，就可以将数据存储到此属性中，一般的小文件都是这样做的。80H 属性中的命名属性就是流文件了，一些 Rootkits 会将自己的数据隐藏到流文件中，通过枚举 MFT 的 80H 属性中的命名属性，得到运行时数据，就可以揪出这些隐藏的信息了。对于 80H 的非常驻属性，这些数据存储在 MFT 以外的运行时中。

90H 为索引根属性，是实现 NTFS 的 B+树索引的根节点，其总是常驻属性。包括：标准属性头、索引根、索引头、索引项。若目录太大而不能存储在索引根中，就会出现 2 个附加属性：索引分配属性和索引位图属性。该属性在解析 NTFS 时是重点，它表示当前 MFT 是一个目录，旗下有一堆子文件、文件夹。该属性的索引根中标记有当前代表的是目录还是文件，索引项中包含了该子文件/文件夹的 MFT 号、文件时间信息、文件名及所对应的 LCN，如图 3.6 所示。

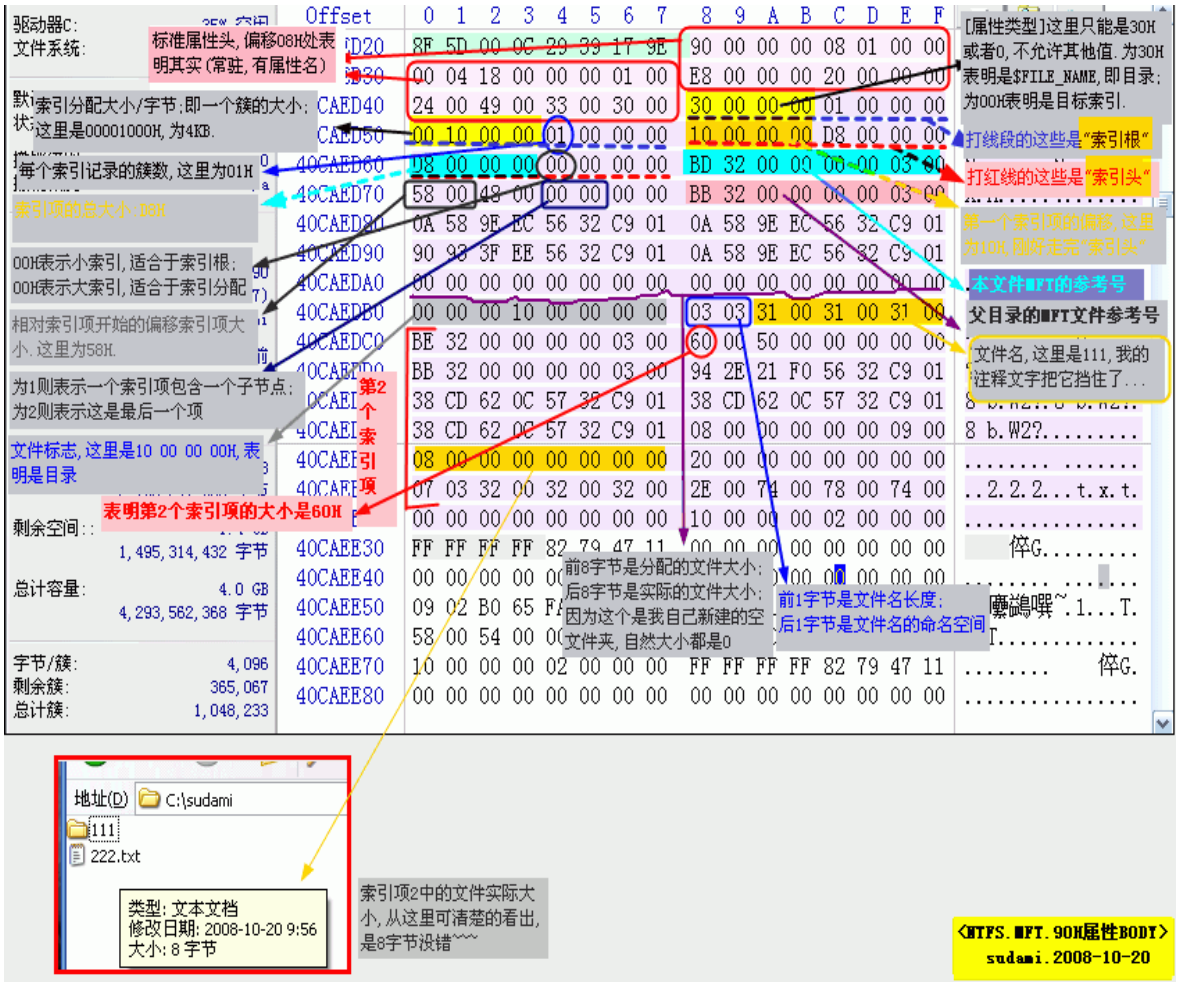

图 3.6 0x90 属性

Fig.3.6 Attribute of 0x90

A0H 是索引分配属性，存储着组成索引的 B+树目录所有子节点的定位信息，其总是非常驻属性。

B0H 是位图属性，由一系列的位构成的虚拟簇使用情况表。该属性用在 2 个地方：索引（如目录）和$MFT。

3.2.4 解析NTFS的实现

系统在内存中为 NTFS 数据流定义了一套完整的数据结构, 比如 VCB（卷控制块）、FCB（文件控制块）、LCB（链接控制块）、SCB（流控制块）、BCB（数据控制块）、VACB（虚拟地址控制块）等，如图 3.7 所示。应用层通过微软提供的标准 API 接口进行文件操作，通过一层层传递到达内核层，然后操作内存中的这些数据结构，读写数据一般会被缓存管理器接管，从缓存 Cache 中获取信息，若当前数据不在缓存中，会产生缺页中断，虚拟内存管理器会将磁盘上的数据映射进内存。比如用户打开某一个文件的操作，流程如图 3.8 所示。

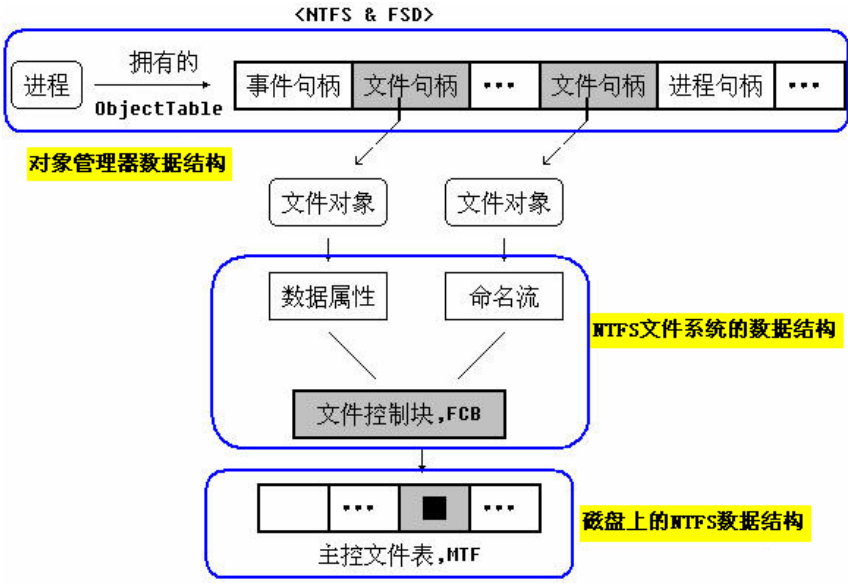

图 3.7 NTFS 和 FSD 的关系

Fig.3.7 Relationship between NTFS and FSD

图

3.8 NTFS 文件系统打开文件的流程

Fig.3.8 Procedures open a file in NTFS file system

这是在文件系统层面，IRP 往下走会经过卷设备（VolSnap.sys、ftdisk.sys、PartMgr.sys ）、端口驱动（disk.sys）、微驱动(atapi.sys)，然后调用 Hal.dll 的内部函数去 I/O 操作读写数据。从整个流程来看，木马病毒可以在其中的任意位置进行劫持，过滤干扰对其自身文件的读写操作。比如挂钩应用层的 API – FindFirstFileExW，内核层的函数 NtQueryDirectoryFile、nt!IopfCallDriver、Ntfs!FindFirstIndexEntry 等隐藏自身文件；挂钩应用层 API – DeleteFile，内核层的函数 NtOpenFile、NtQueryInformationFile、NtSetInformationFile、NtClose 等防止自身文件被删除。而一般的反木马病

毒工具，做的还是不够底层，它们查询文件信息依然经过文件系统，或者调用系统提供的 API 接口进行读写请求，这样便不能对抗顽固的木马病毒了。

无论木马病毒如何保护自己的文件，它一定在物理磁盘上，只不过在读写数据从内核层传到用户层这个复杂的过程中，它们的文件信息被过滤掉了，用户才无法察觉到它们的存在。微软提供的一套文件操作的 APi 接口就是去解析磁盘数据格式，转换成可视化的信息。我们可不用它提供的接口，自己去实现这个过程。这样在很大程度上能绕过木马病毒的过滤，使得到的磁盘数据更为真实，更好的实现检测效果。此时会涉及到两个问题：

(1) 如何解析 NTFS 复杂的数据结构；

(2) 如何读写磁盘数据

对于问题一，由于微软对NTFS的内部数据结构是不公开的，得自己挖掘。结合工具Winhex、微软提供的内核调试器Windbg、Unix下开源的Ntfs-3g，进行调试分析。对于问题二，发IRP不经过文件系统和卷设备，而是直接发送SRB到端口驱动Atapi.sys，进行读写扇区的操作[7,8,9]。

解析 NTFS 磁盘数据的思路如下：挂接相应的分区(比如主分区)，校验 NTFS 的合法性，解析 MBR & BPB ，校验$MFT 的完整性，分析系统的那几个元数据($Bitmap $UpCase $Volume $VOLUME_INFORMATION $AttrDef)，保存一些信息。这些初始化工作完成后,就可以解析指定目录的文件了。 比如在此目录对应的 MFT 中 90H 属性（索引根）、A0H 属性。分别出流文件（80H 属性中的 name 等） 和普通文件，分析 INDEX_BLOCK 下的所有 Index_entry，每个 Index_entry 对应一个文件/文件夹。根据其 MFT 号，得到对应的 MFT，再跟进去读其 90H 属性，判别偏移 10H 的地方是否为 30（即判断是否为文件夹）， 最后即可正确显示出当前目录的所有文件信息。在解析超级大目录（比如 C:\Windows\System32） 时可以这样做：当前目录的 MFT 的 B0H 属性指定了在 $Bitmap 中的位置，到这个位置查看指定的长度。对于每一个 byte 位，若为 1，表明是 In use，得到其地址，就是这一块的 index block，遍历之。一般会遍历好多块 index block，每一个 index block 里面又有很多 index_entry，每个 Index_entry 是一个文件/文件夹。

磁盘级删除文件的思路如下：清掉该文件对应 MFT 的 30H 属性, 设置偏移 16H 的地方为 0, 将 $MFT:$bitmap 中这个文件的 MFT 的标记为 0。 若有运行数据,得到其 LCN,在$bitmap 中找到对应的位置,清 0。最后在给 80H 属性或者 AOH 指向的 Run Data 进行覆盖式的填充垃圾数据。此文章做的工具必须重启后才会看到效果. 因为直接操作的是磁盘, 没有刷新 Cache。如图 3.9 所示:

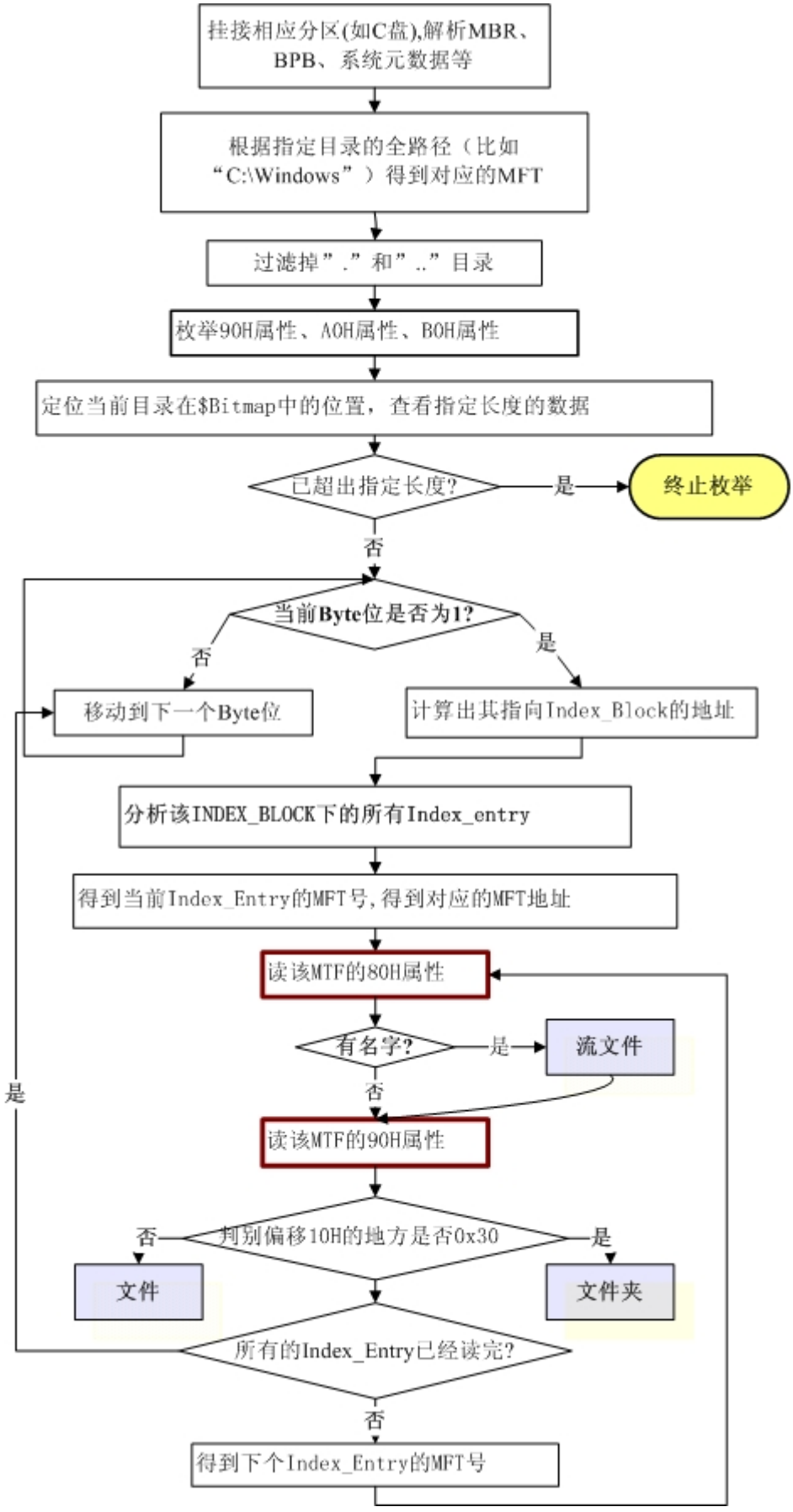
挂接相应分区(如C盘),解析MBR、BPB、系统元数据等
根据指定目录的全路径（比如“C:\Windows”）得到对应的MFT
过滤掉”.”和”..”目录
枚举90H属性、A0H属性、B0H属性
定位当前目录在$Bitmap中的位置，查看指定长度的数据
已超出指定长度?
是
终止枚举
否
当前Byte位是否为1?
否
是
移动到下一个Byte位
计算出其指向Index_Block的地址
分析该INDEX_BLOCK下的所有Index_entry
得到当前Index_Entry的MFT号,得到对应的MFT地址
读该MTF的80H属性
有名字?
是
流文件
否
是
读该MTF的90H属性
否
判别偏移10H的地方是否0x30
是
文件
文件夹
所有的Index_Entry已经读完?
否
得到下个Index_Entry的MFT号

图 3.9 组织程序流程图

Fig.3.9 Organizational process flow chart

3.2.5 实验结果和对比

针对本文提出的基于 NTFS 磁盘解析的恶意程序检测方法，已成功设计能检测出目前所有恶意程序隐藏文件行为的工具。基于本文思想，实现了一个检测软件-----NTFS Paraser，并且选择了 Windows 自身的 Explorer.exe 和流行的系统检测工具 IceSword.exe 来做对比实验。实验的是下面技术和方法都很有代表性的 Rootkit：

（1）Ak922.sys，它在 Disk.sys 级别拦截文件数据包，Patch IofCallDriver 来实现文件隐藏。检测效果如图 3.10 所示：

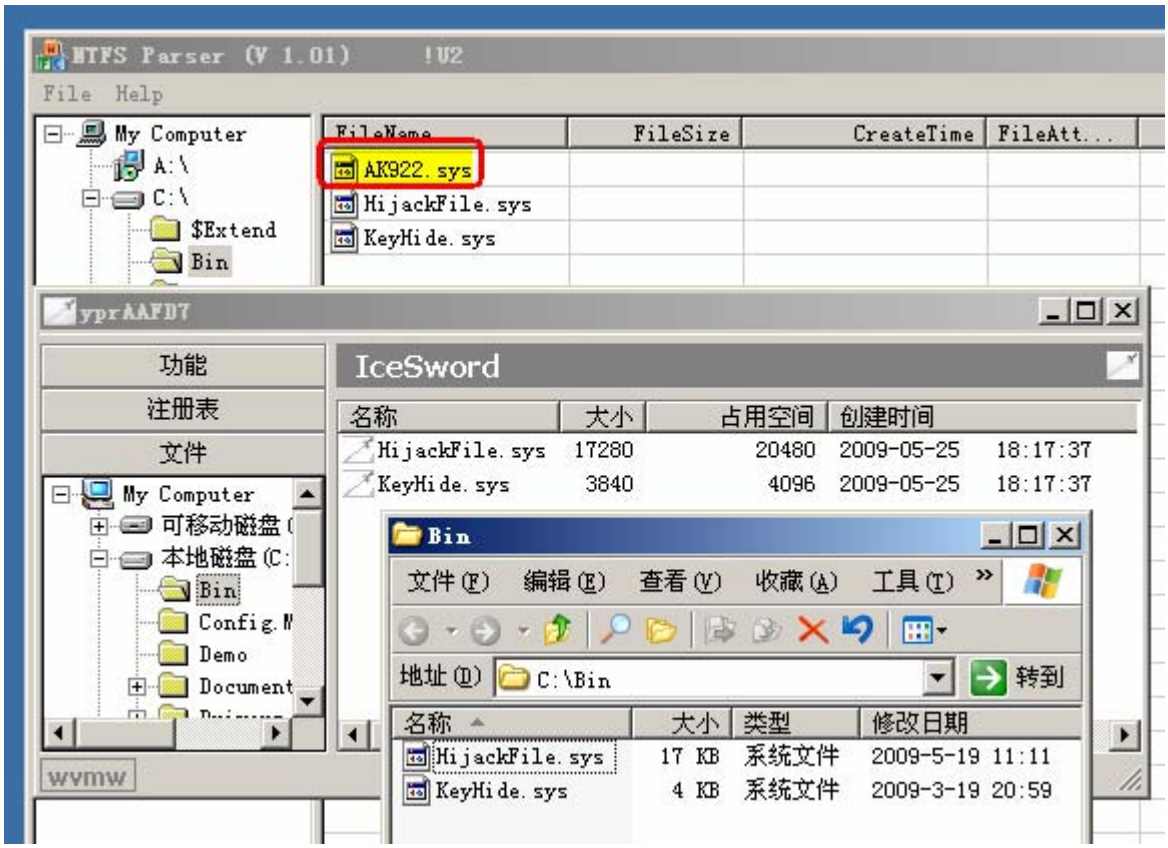

图 3.10 文件隐藏的检测效果

Fig.3.10 Effect of detecting hidden file

3.3 注册表解析

3.3.1 注册表的隐藏、检测及操作

注册表是Windows系统存储关于计算机配置信息的数据库，包括了系统运行时需要调用的运行方式的设置，是系统的核心。操作系统是用特殊的文件（Hive文件）[5]来存储注册表的内容，并提供给用户相关的编程接口来进行注册表的操作，例如RegEnumKey、RegEnumKeyEx、RegDeleteKey、NtOpenKey。操作注册表一般包括枚举、删除、重命名、新增键值等操作。比如打开某个键值，系统流程如图 3.11 所示：

```
kd> kn
 # ChildEBP RetAddr
00 f9df76c0 806251a0 nt!HvpGetCellMapped
01 f9df7890 805b420d nt!CmpParseKey+0x392
02 f9df7918 805b0b3f nt!ObpLookupObjectName+0x119
03 f9df796c 8061b117 nt!ObOpenObjectByName+0xeb
04 f9df7a40 f815dacd nt!NtOpenKey+0x1af
05 f9df7a7c 8053d808 SafeBoxKrnl+0xbacd
06 f9df7a7c 804febd1 nt!KiFastCallEntry+0xf8
07 f9df7b00 8058010b nt!ZwOpenKey+0x11
```

图 3.11 注册表中打开某个键值的流程

Fig.3.11 Procedure of opening cerain regedit key

Nt*系列的 API 是比较上层的，多数恶意程序选择在这个层面下钩子, 躲避检测。往下便是在对象管理器中找指定的对象，在此做 Hook 是比较通用的方法，能干扰不仅仅是注册表的操作，所以也有木马病毒选择这个点。继续往下走便是 Cm*级别的 API，这是相对较为底层的注册表操作函数，少部分恶意程序会 Hook 比如 CmParaseKey 函数来干扰正常程序打开注册表键值，效果是打不开其键值，或者重定向到了其他键值。在 Hvp*级别进行干扰的恶意程序数量大大减少，主要是这部分公开的资料较少，但是仍然可以实现劫持。常见的如劫持 Hive 结构体中的指针 GetCellRoutine，系统初始化时会调用 CmpInitializeHive 初始化系统 Hive 文件，进一步调用 HvInitializeHive 给此指针赋值。以后的注册表部分操作便会调用此指针指向的函数。在此做劫持的恶意程序能够很大程序上保护自己，对抗安全软件。再往底层走就是复杂的数据结构操作，目前还没有任何恶意程序在此层面做手脚。再如，枚举键值的流程如图 3.12 所示：

```
kd> kn
 # ChildEBP RetAddr
00 f9df79a0 80622e8a nt!HvpReleaseCellMapped
01 f9df79bc 8062707c nt!CmpFindSubKeyByNumber+0x68
02 f9df7a04 8061a5fc nt!CmEnumerateKey+0x44
03 f9df7a94 8053d808 nt!NtEnumerateKey+0x1ea
04 f9df7a94 804fe811 nt!KiFastCallEntry+0xf8
05 f9df7b24 8057cf7b nt!ZwEnumerateKey+0x11
```

图 3.12 注册表中枚举某个键值的流程

Fig.3.12 Procedure of enuming cerain regedit key

一些常见的安全工具运用了一定的技术来检测并清除这些恶意程序的注册表键值。比如调用应用层API获取注册表信息，保存为数据A，恢复一些必要的钩子，利用更为底层的内核函数取得注册表数据B。对比数据A和数据B，从中找出差异，就是隐藏的键值。但是这并非绝对的可靠，因为不能保证恶意程序的钩子一定被摘除了，也不能保证取得的底层函数没有受到干扰。于是通过基于解析注册表Hive的方法来获取原始的注册表数据显得十分必要[10,11,12]。

3.3.2 注册表解析的原理及实现

在磁盘上，注册表并非简单的一个大文件，而是一组称为Hive的单独文件。每个Hive文件包含了一颗注册表树。有一个键作为该树的根，子键和他们的值存储在根的下面，如图 3.13 所示。当配置管理器加载 Hive 的时候，会在HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet \Control\hivelist子键下的注册表值中记录下每个Hive的路径。

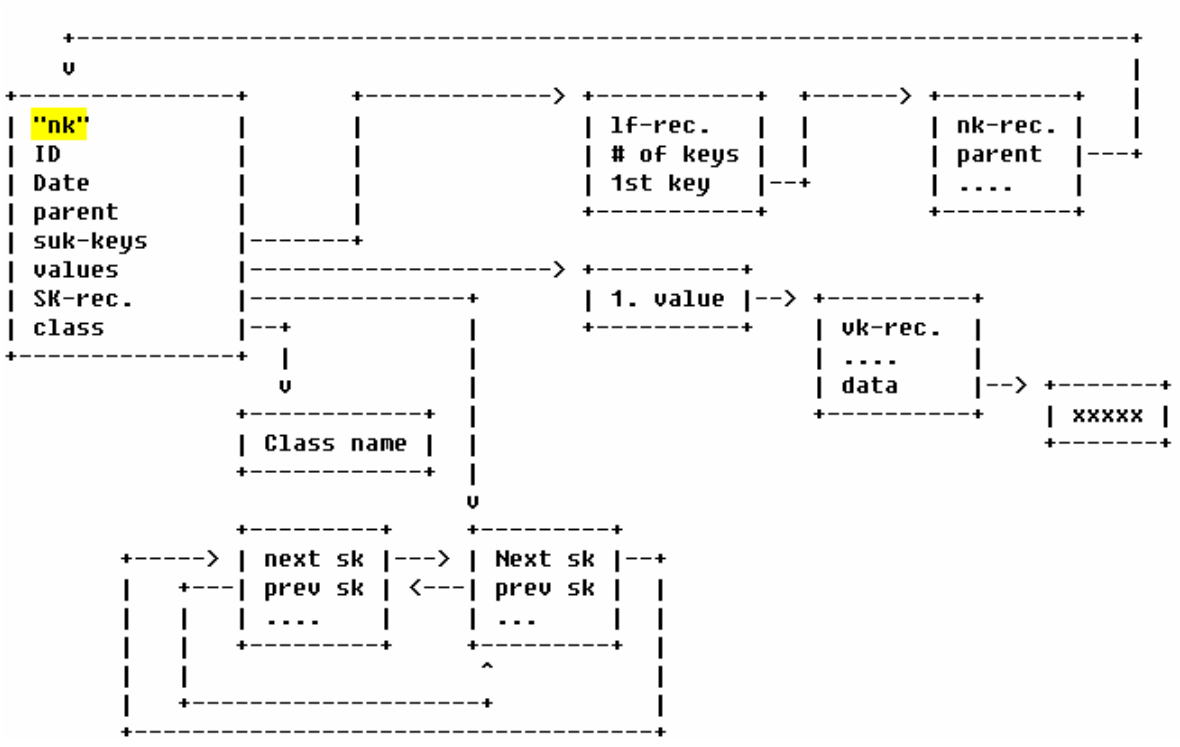

图 3.13 注册表 Hive 文件的结构

Fig.3.13 Structure of the hive registry file

结合二进制查看工具 Winhex 和内核调试器 Windbg，对 Hive 格式进行探究。系统拥有的 Hive 文件用户是无法打开的，它们被系统进程 System.exe 独占使用。那么该如何读写 Hive 文件呢？这

些 Hive 文件被 System.exe 以独占权限打开，其他程序均无法访问。但是可以解析磁盘数据结构，找到 Hive 文件对应的簇号，然后 ReadFile 将包含这些文件数据的扇区内容读出来。或者在驱动中向 Atapi.sys 设备发送控制码，读写磁盘数据。但这种方法不是很简单，而且效率不高。本文采用的方法是复制 System 进程拥有的 Hive 文件句柄，调用函数 CreateFileMapping 映射一份，然后对这份内存进行读写请求。这样比直接读磁盘文件要快的多，因为操作的是缓存中的内容，而且更安全稳定。比如要列举某个父键下的所有子键，思路如下：定位到此父键的 Hive 节点,对应的结构体为 CM_KEY_NODE，其中成员 SubKeyLists 存储着其拥有的子键信息，但是 Windows 对子键的数量做了 2 种不同的处理，对于超级大目录，会采用‘ri’结构来存储子键信息。对于一般的小型目录，采用‘lf/lh’结构存储。在‘ri/lf/lh’结构体内，包含着子键的数量、一个由所有子键的子结构组成的数组,如图 3.14 所示。遍历整个数组，从每个子结构中取出当前的子键名，进一步判断该子键是否拥有 2 级子键，然后将当前信息插入到 MFT 图形界面的树形控件中去。这样就完成了对当前父键下所有子键的枚举，然后调用应用层 API 再次枚举，将 2 份数据进行对比，有差异的部分即是恶意程序要隐藏的注册表键值了。

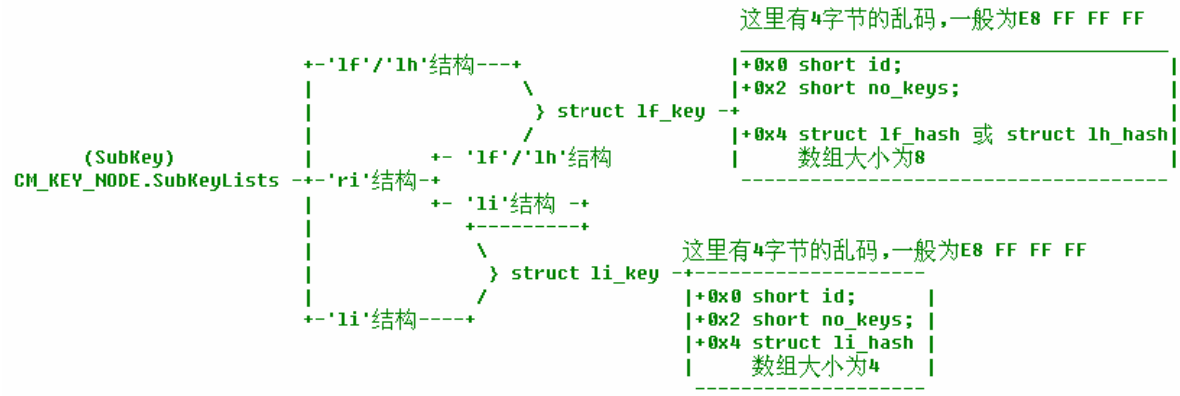

图 3.14 Hive 文件中几个结构体的对应关系

Fig.3.14 The correlation between some structure in the hive file

当然，能检测出恶意程序隐藏的注册表信息后，还需要能够删除它们，基于 Hive 解析方式的删除是非常强大的，可以删除目前任何顽固的被木马病毒所保护的注册表键值。比如要删除某个父键 A 下的一个子键 B，不管该子键 B 下是否还有子键 C，都可以一次性的删除掉。思路如下：

（1）检查该子键是否存在 2 级子键，若存在依次枚举之，递归遍历直到枚举到的节点不存在子节点，进入步骤（2）；

（2）对于当前的无子节点的子键，删除其拥有的内容（即清空其 vk_key 结构体，释放内存），删除父键中该子键（遍历父键的 SubKeyLists 数组，根据名字匹配得到数组中对应的节点，清空该节点，释放内存，更新父键对应的结构体 CM_KEY_NODE 中的部分成员变量的信息）。

3.3.3 实验结果和对比

针对本文提出的基于注册表 Hive 文件的恶意程序隐藏检测方法，可设计能检测出目前所有恶意程序隐藏注册表行为的工具软件。基于本文思想，实现了一个检测软件——RegHive，并且选择 Windows 自带的注册表查看程序 Regedit.exe 和流行的系统检测工具 IceSword.exe 来做对比实验，实验对象是下面两种技术和方法都很有代表性的 Rootkit：

(1) ObjectDisableKey.sys，它是利用 Object Hook 技术来保护自身键值，防止注册表键值 HKEY_LOCAL_MACHINE\SYSTEM\北京林业大学\BeijingLinYeDaXue 被打开；

(2) Keyhide.sys，它通过修改 Hive 结构中的函数指针来隐藏注册表键值 HKEY_LOCAL_MACHINE\SYSTEM\北京林业大学。

如图 3.15 所示，本文设计的工具 RegHive V1.02 能够成功检测出隐藏的注册表键值“HKEY_LOCAL_MACHINE\SYSTEM\北京林业大学”，而系统自带的注册表编辑器及网络上主流的 ARK 工具 IceSword 都无法检测出来。

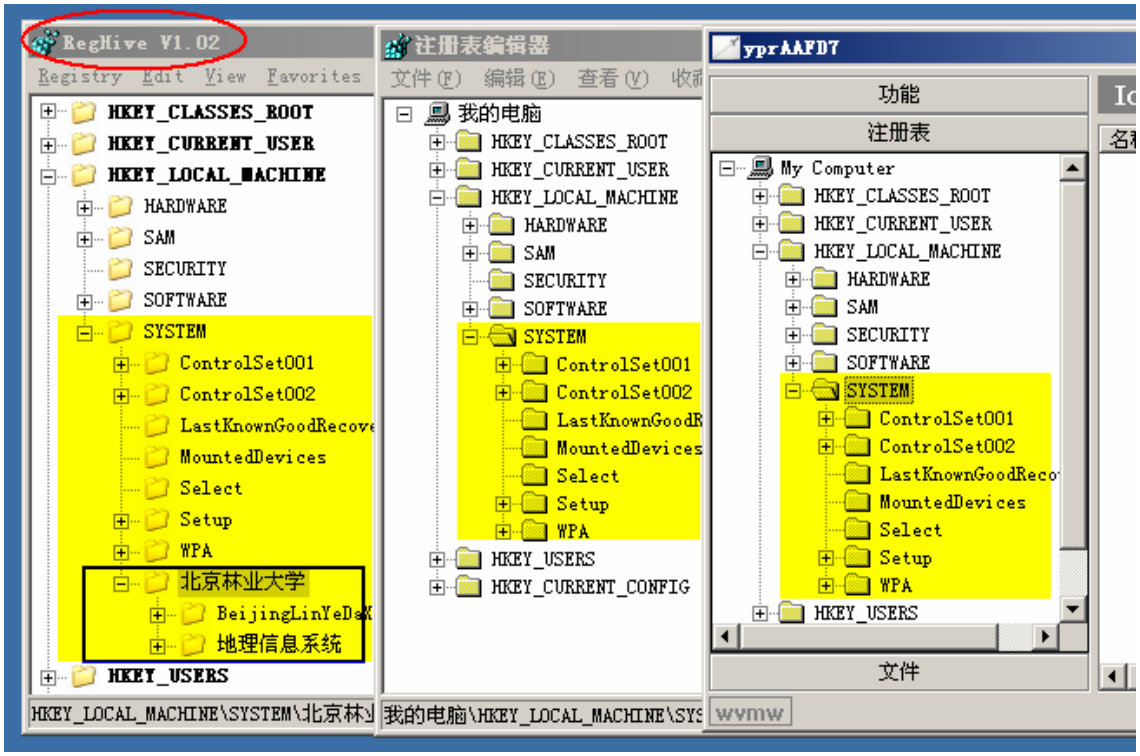

图 3.15 对隐藏注册表的检测效果

Fig.3.15 Effect of detecting hidden registry key

如图 3.16 所示，系统自带的工具和主流的 ARK – IceSword 均无法操作键值 BeijingLinYeDaXue，而本文设计的工具 RegHive V1.02 能够成功打开并且正常操作。

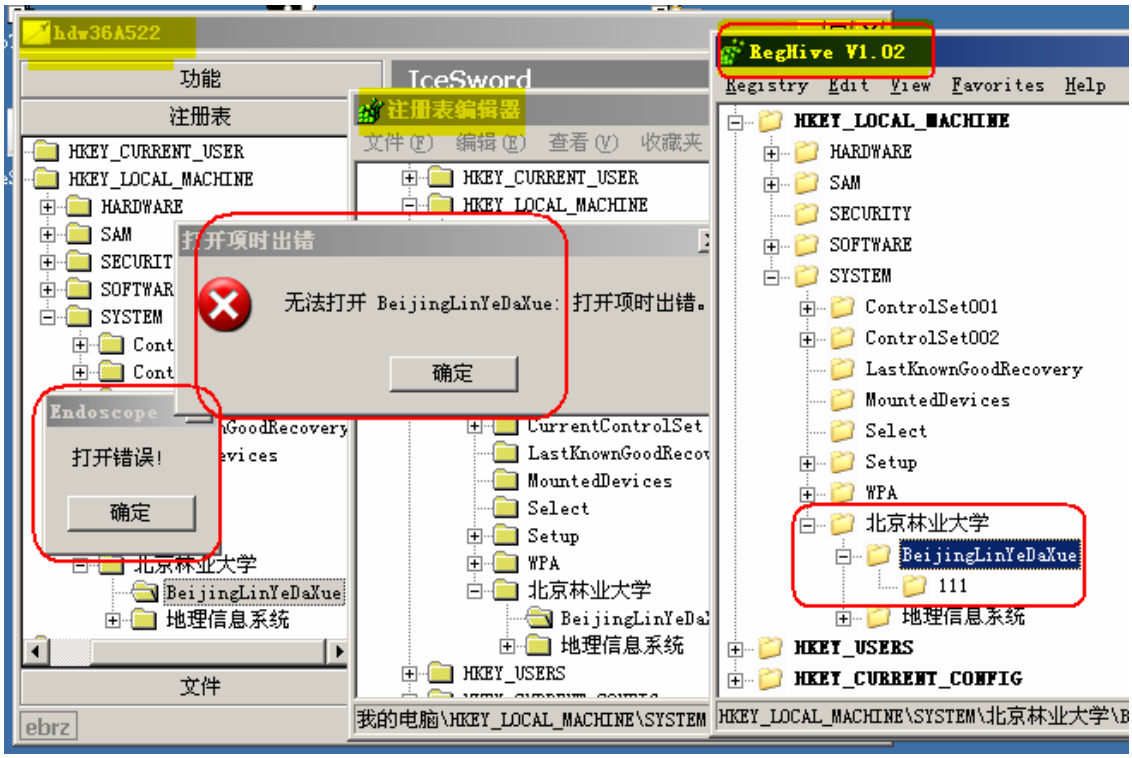

图 3.16 对隐藏注册表的检测效果

Fig.3.16 Effect of detecting hidden registry key

3.4 内核钩子的检测

3.4.1 内核钩子的基本原理

木马病毒为了生存，躲避对抗安全软件，常通过驱动程序在 Windows 内核层挂上一些钩子，以干扰系统正常 API 的流程，过滤信息，保护自己不被发现和清除。钩子的种类众多，比如 ssdt、IAT、EAT、Object、系统中断、FSD 等，各种类型的检测方法有一定的差异，在此仅阐述关于 Inline Hook 的检测方法。

目前大部分ARK，诸如GMER、RKU、IS、狙剑等在检测系统是否被恶意程序挂钩方面，基本是拿整个磁盘文件的代码段和内存做比照，分析出恶意模块，此种做法效率不高、强度不够，且不能对单个函数进行深度扫描。实际测试中, 发现网上比较出名的ARK在此方面都存在一定的不足, 诸如未导出函数、FF15 类型的Call Hook、FF25 类型的JMP Hook 、多级跳 、Call 寄存器等方式的Hook检测都存在遗漏问题[13,14,15].

3.4.2 内核钩子的检测的实现

微软有数量庞大的函数，这些函数都是一些汇编指令，每个函数对应一段汇编指令。 Inline

Hook，就是在某个函数内部，修改某条、某段汇编指令。程序在执行时,会调用某个函数，原本会按照函数内部的汇编指令一条一条的执行下去，直到该函数结束，但是木马会修改函数内部的某段汇编指令，让该函数执行的时候发生跳转，跳转到其他地方执行其他指令。一般检测这用 Hook 的方式即是扫描每个已被内核导出的函数内部指令，通过和磁盘文件的二进制进行对比，找到被修改的地方，若存在跳转指令，基本可以确定被木马病毒篡改掉了。但是这种通用的方法有很多弊端：

（1）因为微软的函数分为“已经导出”和“未导出”，前者就可以直接得到其函数地址的，而后者是无法直接得到地址，必须通过搜索指令等方式才能得到正确的函数地址。这样导致扫描的函数不完全，无法扫描检测“未导出”的函数。

（2）跳转的指令有很多种，可以具体的查看 Intel 手册。而且磁盘文件和内核数据在某些地方不一定完全相同，进行对比的时候需要判断跳转类型和跳转地址。比如 0xE8 类型 Call 指令，其调用的函数地址可以直接计算，但是 0xFF15 类型的 Call 指令需要进行二次计算；长跳转和段跳转也是有所不同的。

（3）若发现磁盘数据和内核数据不相匹配，也不能马上断定是木马造成的，还要跟进到调用地址内部去进一步判断。比如很多类型的 Inline Hook 修改函数后，跳转地址仍然在内核 Ntoskrnl.exe 内部，然后再进行多级跳转，最终跳到自己内核模块的函数中去。这样的跳转给检测工作带来了极大的难度。

综合上述问题，在此详细阐述一下基于深度递归扫描的方式检测 Inline Hook 的方法:

指定一个函数（不指定则对系统的所有 API 进行扫描），引擎负责跟进扫描其下面调用的所有函数，即扫描函数内部的汇编指令，遇到跳转就跟进去，检查其是否落在正常地址范围内，若不是，则断定为 Inline Hook，若是则继续递归跟进,在一定级数内直到发现异常的跳转地址。在判断跳转指令时，对 Call、jmp、push+ret 等方式进行区别对待，不同的指令需要用不同的方式计算跳转后的地址。通过这样的递归方式，可以检测出未导出函数的钩子，多级跳，内核代码段小跳等。流程图如图 3.17 所示：

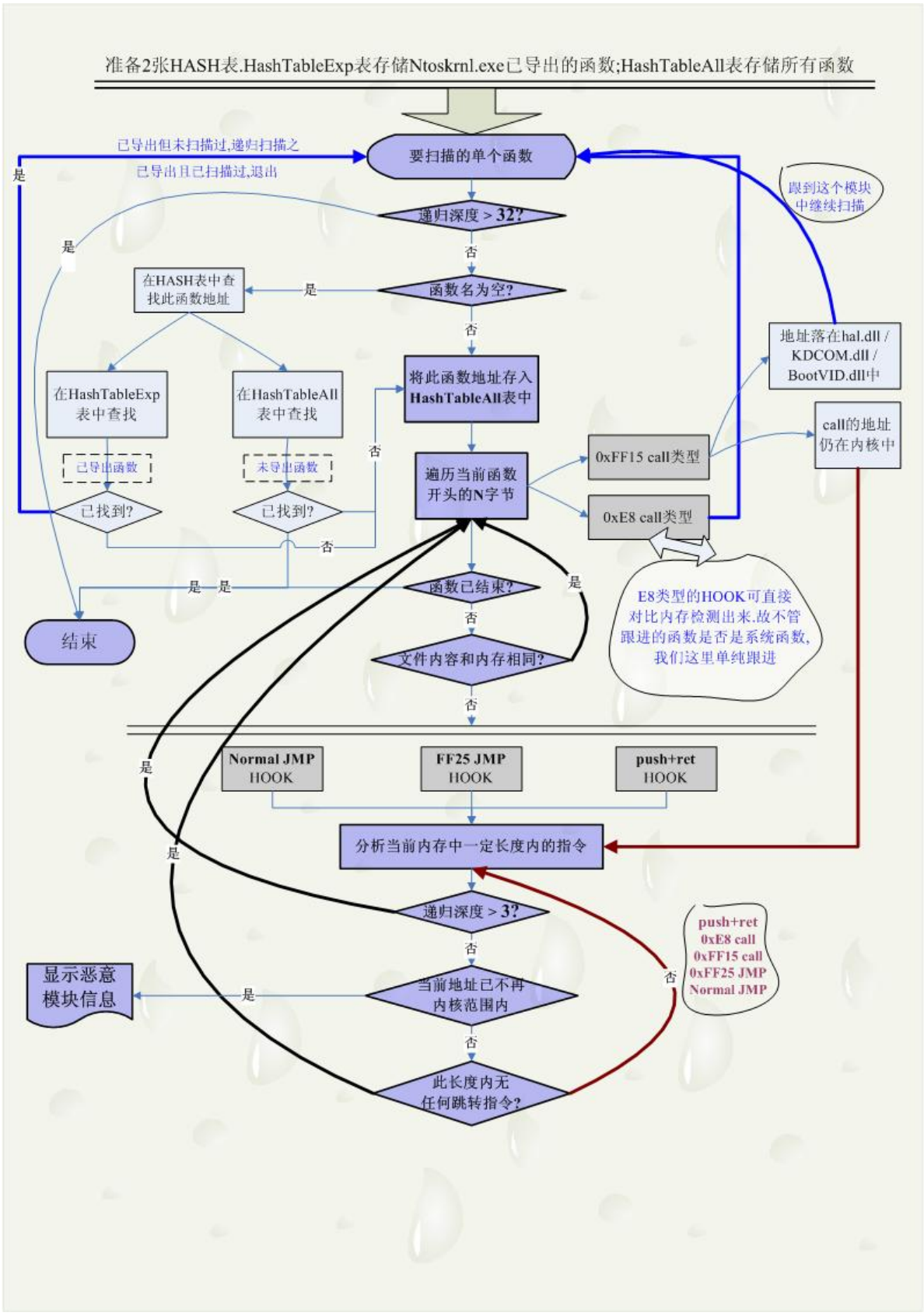

图 3.17 Inline Hook 深度递归扫描流程

Fig.3.17 Procedure of depth scan for Inline hook

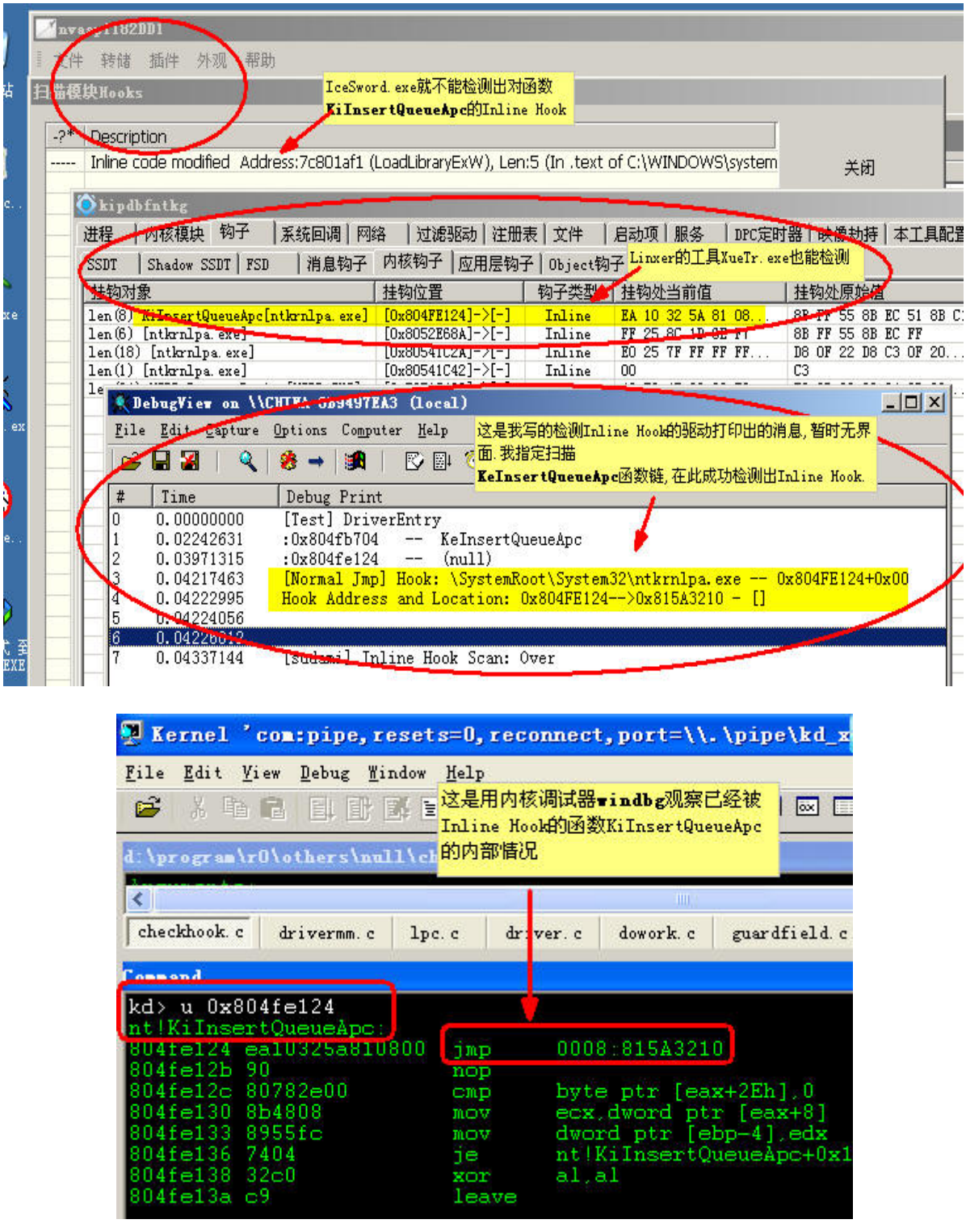

图 3.18 Inline Hook 检测效果

Fig.3.18 Effect of depth scan for Inline hook

如图 3. 18 所示，冰刃无法检测出对 KiInsertQueueApc 这个未导出函数的 Hook，而本工具可以成功的发现。

4 结束语

4.1 论文工作总结

本文首先说明了对木马病毒进行研究的意义及其应用场景，随后提出了基于 Windows 内核层的 Anti-Rootkits 的研究方法。

随后，本文对木马病毒的基本原理及其危害性进行了简要的概括与说明，同时对一些基础的概念进行了解释，主要说明了研究 Anti-Rootkits 的形式化意义，以及其在本文中的实际意义。

接下来，基于 NTFS 磁盘解析、注册表 Hive 格式解析、内核钩子检测的设计与实现是本文的核心内容。本文首先对该系统的总体构架进行了介绍，并详细说明了每个部分的分工与实现功能，紧接着，对该原形系统分析过程中所使用的各个核心算法进行了详尽的描述与说明，其中包括解析 NTFS 磁盘数据的算法，基于注册表 Hive 格式的检测效果展示，以及内核钩子检测算法的设计与实现。

4.2 今后的工作与展望

（1）基于 NTFS 磁盘解析的恶意程序检测方法是目前最为有效的方法，因为直接和磁盘数据打交道，获取的数据较为真实。但是，木马病毒依然可以在磁盘数据读取环节上下功夫，比如在 Atapi.sys 以下进行劫持过滤。虽然可以直接 I/O 端口进行数据的读写操作，但是兼容性不好，难度太大，所以如何才能更加安全的读写原始磁盘数据将成为下一步的研究重点。

（2）基于注册表 Hive 格式解析的方法检测清除恶意程序的注册表键值，是非常底层非常强大的方法，而且直接从内存中解析和非磁盘操作更加的快速高效。但是在删除键值上还不能做到绝对安全，因为删除操作会直接清空内存中的 Hive 数据，改变一些结构体内容。由于违背了正常的逻辑，Windows 并不知道这些变化，当它试图去操作这块被清空的数据时，可能会在某个阶段会发生不可预知的错误。所以如何更加稳定安全的删除木马病毒的键值，成为下一步研究的重点。

（3）内核钩子的检测在强度上采取了深度递归的方式，能够很大程序上发现隐藏的钩子。但是目前的稳定性还有待提高，检测种类也比较少。后期会继续提高程序的稳定性，增加对其他种类钩子的检测。

致谢

在本论文的编写过程中，我得到了来自李冬梅老师的悉心指导与中肯建议，在论文完成之际，我谨向李老师表达最衷心的谢意。感谢李老师在学习上曾给予我的关心与指引，李老师严谨的治学态度与平易近人的生活作风都给我留下了深刻的印象。特别是在我求学的道路上，幸好有李老师为我指点迷津，并给予我无私的帮助，才使我有机会进入到更广阔的天地去继续深造。在此衷心感谢李老师对我的帮助与关怀。

此外，我还要感谢在毕业设计过程中遇到问题时和我一同研究探讨的奇虎 360 公司的同事们。

最后，我要感谢我的家人，正是由于他们的理解支持与无私奉献才使我能够在学校专心完成自己的学业。

参考文献

[1]Greg Hoglund,Jamie Butler.Rootkits:Subverting the Windows Kernel[M].America:Addison-Wesley Professional,2005.

[2]Mark E. Russinovich, David A.Solomon.Windows Internals 4th[M].America:Microsoft Press,2005.

[3]张银奎.软件调试[M].北京:电子工业出版社,2008.

[4]Rajeev Nagar.Windows NT File System Internals[M].America:O'Reilly,1997.

[5]Abraham Silberschatz,Peter Baer Galvin,Greg Gagne.Operating Systems Principles[M].America:John Wiley & Sons,2005.

[6]戴士剑,涂彦辉.数据恢复技术(第 2 版)[M].北京:电子工业出版社,2005.

[7]FRIEDHELM SCHMIDT.The SCSI Bus and Ide Interface: Protocals, Applications and Programming (2nd Edition)[M].America:Addison-Wesley,2001.

[8]Peter M. Ridge,David Deming.The Book of Scsi[M].America:No Starch Pr,1995.

[9]于渊.自已动手写操作系统[M].北京:电子工业出版社,2005.

[10]吴伟民,严蔚敏.数据结构(C 语言版)[M].北京:清华大学出版社,2007.

[11]Jeffrey Richter,Christophe Nasarre.Windows via C/C++[M].北京:清华大学出版社,2008.

[12]段钢.加密与解密(第三版)[M].北京:电子工业出版社,2008.

[13]Mario Hewardt,Daniel Pravat.Advanced Windows Debugging[M].America:Addison-Wesley,2007.

[14]CharlesPetzold.Programming Windows (Fifth Edition)[M].北京:北京大学出版社,2004.

[15]Penny Orwick,Guy Smith.Developing Drivers with the Windows Driver Foundation[M].America:Microsoft Press,2007.