火龙果•整理
uml.org.cn

中文摘要

数据文件的安全对国家机构、企业和个人都有至关重要的作用。现存的对数据文件有效的保护方法有利用硬件技术的硬盘加密技术；在驱动层的透明加解密；过滤驱动层的加解密技术；基于 API HOOK 的透明加解密技术。这些技术都能控制在硬盘上存储的数据文件以加密的形式存储，既不影响用户使用，也能在一定程度上保证了数据文件的安全。

在各种技术中，本文主要研究基于 API HOOK 的透明加解密技术，一方面是因为它是最靠近应用程序读写文件的接口，使明文数据使数据文件在内存中以明文形式出现的时间最短，只要做好进程保护，就能绝对控制数据文件的安全；另一方面，市场上此类产品因为设计实现难度较大而缺乏通用产品，具有很好的实用意义。

本文主要分析几种常用的数据文件的类型，分析软件行为，研究 WindowsNT 平台下 kernel32.dll 和 ntdll.dll 的部分 API，通过设计实现一个独立的处理系统，HOOK 部分 API 函数，实现数据文件的透明加解密。

本论文不仅提出了一套在用户 API 层实现的透明加解密的设计方法，还实现了一个能满足几种特定软件的原型系统。

关键词：数据文件　API HOOK　透明加解密　系统设计与实现

火龙果•整理
uml.org.cn

ABSTRACT

The security of data files is important. There are four main methods now that can efficiently protect data files: encrypt and decrypt data in hardware level, in file drivers, in filter driver and in user space. All of these methods can promise data files in disks are stored with encrypted format, and be transparent to customers.

Of all these methods, the objective here is aimed at transparent encryption and decryption using API HOOK. The reason can be explained in two aspects. Firstly, this method keeps the original data(plain text) nearest to the user memory. Therefore, the time plain text exists is the shortest. Secondly, the product designed and realized by this method is not easy and rare. It is of great use if it was developed.

This analyses the actions of some generally used software applications, and also studies some API functions of kernel32.dll and ntdll.dll of Windows NT platform. We achieve the objective of implementing a system that can encrypt and decrypt data file transparently by hooking some significant API function.

It does not only put forward the design methods and principles, but also develops a model of our theory which can be applied to some certain software applications.

Keywords： Data files API HOOK transparent encryption and decryption system design and implement

火龙果•整理
uml.org.cn

主要符号对照表

本文中使用的符号对照表如下：

API	应用程序编程接口（Application Programming Interface）
OS	操作系统（Operation System）
NTFS	NT 文件系统（新式技术文件系统）
EFS	加密文件系统（Encrypting File System）
FAT	文件分区表，管理文件系统（File Allocation Table）
DLL	动态链接库（Dynamic Linkable Library）
IPC	进程间通信（Inter Process Communication）

火龙果•整理
uml.org.cn

目 录

第 1 章 引 言 1
1.1 研究背景 1
1.1.1 数据文件泄漏的案例 1
1.1.2 数据文件受到威胁的分类 2
1.1.3 本文的目标和意义 3
1.2 研究现状 4
1.2.1 Windows NT 系统与文件存取相关机制简介 4
1.2.2 硬件加解密 4
1.2.3 驱动层透明加解密 6
1.2.4 过滤驱动层透明加解密 7
1.2.5 用户层 API HOOK 透明加解密 9
1.2.6 几种技术的比较 9
1.3 系统概览与论文结构 10
第 2 章 研究的相关理论和工具介绍 11
2.1 用于 API 注入的 MadCollection 开发包 11
2.1.1 MadCollection 简介 11
2.1.2 MadCollection 在本论文的中的应用 12
2.2 涉及的 API 函数的研究 13
2.2.1 Shell32.dll 中相关的 API 函数 13
2.2.2 Kernel32.dll 中相关的 API 函数 14
2.2.3 Ntdll.dll 中相关的 API 函数 17
2.3 分组密码和序列密码 20
2.3.1 分组密码及 AES 算法 20
2.3.2 序列密码及 RC4 算法 22
第 3 章 系统的分析与设计 24
3.1 系统设计的目标 24

火龙果•整理
uml.org.cn

3.1.1 系统设计的前提假设 24
3.1.2 数据文件保护的解决方案 25
3.2 软件及系统行为分析 27
3.2.1 系统的范围 27
3.2.2 分析的结果 27
3.3 系统的设计模型 32
3.3.1 系统的设计模型 32
3.3.2 钩子注入卸载模块 33
3.3.3 进程间通信模块 33
3.3.4 文件复制模块 33
3.3.5 注入的 DLL 模块 34
第 4 章 系统的实现 35
4.1 设计系统实现和运行的环境 35
4.2 DLL 文件的注入卸载模块 35
4.3 进程间通信模块 36
4.4 文件复制模块 37
4.4.1 数据结构的介绍 37
4.4.2 多线程处理多个复制的操作的实现 38
4.4.3 异步模式读写文件加解密的实现 38
4.5 注入的 DLL 模块 41
4.5.1 模块的内部结构 41
4.5.2 文件名和句柄的对应 44
4.5.3 进程和后缀的控制 45
4.5.4 文件加头的处理 45
4.5.5 加密算法的选择 46
4.6 日志系统 47
第 5 章 结论 49
5.1 取得的成果 49
5.2 未来的展望 50
插图索引 51

火龙果•整理
uml.org.cn

表格索引 52
参考文献 53
致 谢 55
声 明 56
附录 A 外文资料的书面翻译 1

火龙果•整理
uml.org.cn

第1章 引 言

1.1 研究背景

1.1.1 数据文件泄漏的案例

2007 年 10 月底，美国新英格兰地区 300 多家银行在对全球折扣零售业巨头 TJX 公司提出起诉的文件中称，由于没有采取必要的安全措施，该公司在黑客的攻击下发生了严重的数据泄露事件，导致 9400 多万用户的账户受到影响，并由此引发了至少 13 个国家发生大量信用卡和借记卡诈骗案件，仅 Visa 卡用户的损失就超过 6800 万美元。

2007 年 10 月，我国相关部门发现了境外间谍机关实施的一次大规模网络窃密行动，攻击对象全是中国政府、军队和国防科研机构、军工企业网络，受到攻击的单位遍及我国绝大部分省、自治区、直辖市，甚至还包括我国十几个驻外机构。据悉，在该案中被境外情报部门控制的电脑和网络达数百个，窃密内容涉及政治、军事、外交、经济、医疗卫生等多个领域。而此次活动的具体执行者就是一位名叫李芳荣的台湾间谍。他利用黑客技术，控制了多个服务器，又通过这些服务器将木马植入其感兴趣的电脑，然后实施网络窃密等破坏活动。

2003 年三月上旬，航天科工集团某研究所技术人员王某违反保密管理规定，私自将存有涉密信息的笔记本电脑和移动硬盘带回家。几天后，王某家中失窃，笔记本电脑和移动硬盘被盗，硬盘中存储的 1 份机密级报告、2 份秘密级资料草稿和 4 份秘密资料草稿失控。王某因泄露国家秘密被给予行政记过处分。

2003 年 1 月 11 日，某公司技术开发中心军品、民品研究所（位于公司办公楼三楼）被盗。据现场勘查，罪犯黎爬墙外排水管到达三楼，打破窗户玻璃进入研究所实施盗窃。室内 5 台涉密电脑的硬盘、主板、CPU、内存条、显示卡等部件均被盗走。被盗硬盘中存有大量军品资料，经鉴定，其中 3 份图纸属于秘密级国家秘密。

2001 年 3 月，成都某大学科技处副处长田某与副教授周某携带装有大量涉密软盘的密码箱，到北京向有关部门申报该校“十五”科研开发项目。两人乘飞机返回成都后，田某要使用密码箱中的资料时，发现密码箱丢失。箱内装有 31 张软盘，储存有 163 份项目论证报告材料。绝大多数属国防军工电子科研项目，其中

火龙果•整理
uml.org.cn

一级密级国家秘密 14 项，秘密级国家秘密 32 项。此外，箱内还有 2 份机密级文件。

随着信息技术的应用和普及，越来越多的信息通过数字化的形式存储在个人计算机和大型服务器上。信息的收集、存储和传播比以往更加的方便、快捷和有效。但在享受信息数字化的方便和快捷的同时，信息安全的问题也不能忽视。据统计，2005 年，在全国查处的泄密事件中，通过计算机泄密的占 52%；2006 年，通过计算机泄密的事件上升到 64%。计算机泄密已成为当前泄密的主要方式。如何在实际的生产生活中既能享受计算机带来的方便，又能保护好数据文件的安全是一个值得研究的课题。

信息安全的问题存在于个人和团体应用的各个方面，本论文主要是研究关键的数据文件的保护。

1.1.2 数据文件受到威胁的分类

通过对实际案例的分析，数据文件安全问题主要可以分为两类：（1）数据文件被破坏；（2）数据文件被窃取。

数据文件被破坏是指存储在计算机上的数据文件不能正常使用，可能包括的情况有数据文件被删除、数据文件被篡改等，也可能包括计算机硬盘被破坏。这种情况下有可能是因为地震、洪水等自然灾害或者战争存储数据文件的计算机硬盘被彻底报废，也可能是人为原因的故意破坏。数据文件被破坏的情况可以通过数据备份在一定程度上来解决，并且现在这种技术和产品都已经很成熟。通过把数据备份到其它存储介质、远程服务器等方式，即使数据遭到破坏，依然可以进行恢复，因此本文不对这个方面问题进行讨论。

数据文件被窃取是指存储在某个受控计算机的受控文件失控，主要可以分为硬件上直接盗窃磁盘和软件上通过复制数据文件来实现的。对于直接盗窃硬盘这种暴力的方式，主要需要加强对计算机安放环境的安全保卫。而对于软件层面的窃取行为，首先可以把它分为两类：外部入侵和内部人员泄密。外部入侵主要是指通过网络入侵到存储关键数据文件的计算机上，窃取数据文件；内部人员泄密是指能够通过合法途径访问到数据文件的人非法地把文件转移给其他没有权限访问这些文件的人。防御外部入侵也是一门很复杂的技术，这个主要不在本文中研究，详细的内容可以参见一些关于入侵检测和入侵防御的资料。除了防止外部入侵以外，可以通过把数据文件加密的方法使得即使文件被窃取，破译密码后才能使用，这也能在很大程度上对外部入侵窃取文件的危害减轻。相对于外部的黑

火龙果•整理
uml.org.cn

客入侵而言，内部人员泄密造成的后果更为严重，也更加不容易预防。因为内部人员知道哪些数据最为重要的，重要的数据在什么地方，并且在拥有足够权限的基础上可以通过合法的手段将数据窃取，而不被发觉。要防止这些潜在危险的存在，仅仅通过严格的权限控制或者人事管理是解决不了问题的，设计不出一套方案，能够在不确定每个人都是可靠的前提下，保证数据文件的绝对安全。所以，需要选择数据文件加密的方法，即使数据文件被转移到失控，也能通过密码对文件产生一定程度的保护。

通过对数据文件受到的威胁分类可知，对存储在磁盘上的数据文件进行加密，可以对防止数据文件通过外部入侵和内部泄密上有一定的作用，也是所有方法中成本较低，安全性较高的方法。

1.1.3 本文的目标和意义

本文的目标就是通过对存储在计算机硬盘上的数据文件进行加密，保证数据文件都是以密文的形式存储在硬盘上。即使文件被窃取，对加密文件的破解也能成为关键文件泄密的主要屏障。

这种加密是建立在不会消耗太多的用户体验和效率的基础上的，本文中称为透明加解密。在用户体验方面，计算机的用户完全感受不到这种加密的存在，密文文件在被加载到内存的过程中自动的解密，在从内存保存到硬盘的过程中自动被加密。这种设计让用户完全接触不到加解密的过程，一方面减轻了用户的负担，让用户不必参与到复杂的加解密的过程中；另一方面，减少用户的参与也减少了用户在参与过程中泄密的可能，让有泄密想法的用户根本插不上手。在效率方面，主要对效率影响的因素是加解密过程，不同的加解密算法将会有不同的强度，但是越强的算法通常时间、资源消耗越大，选择合适的算法进行加解密操作一方面不要牺牲太多的时间和资源，另一方面又具有较强的安全性也是本文需要考虑的问题。

通过构造论文中所提到的系统，要达到的目标是对所有离开受控计算机的文件都是加密文件。如果有人企图通过窃取计算机硬盘、通过网络窃取文件、通过可移动设备窃取文件，即使成功，得到的也是加密的文件，虽然不能保证这些加密文件不被破解，但破解密码成本相对较高，可以有效的保护文件。另外，如果企图通过打印等操作获得纸质版的文件，将也会对打印操作进行控制，这个部分的实现可以在同一个组的同学那里看到详细的情况。

火龙果•整理
uml.org.cn

本文的系统设计主要有两个部分，一个是在读写文件的透明加解密的操作，一个是在复制文件的加解密的过程。保证存在硬盘的数据均为密文，从计算机考出的文件也是被加密的文件。这么操作的目的和意义就是把物理窃取、网络入侵、内部泄密的情况最终都归结到密码破解，即使得到受控数据文件的话也需要进行复杂的密码破译的工作。并且根据不同的文件受控级别，可以设置不同的密码强度。在没有办法保证数据文件绝对安全、不被窃取的情况下，实行密码保护也许是最优的选择了。

1.2 研究现状

1.2.1 Windows NT系统与文件存取相关机制简介

由于对基于 Windows NT 文件系统进行的透明加解密的研究都是在 Windows NT 系统中不同层面上实现的，因此为了介绍并且比较这些实现的方法和机制，首先介绍一下 Windows NT 系统跟文件存取相关的某些系统机制和子系统。关于 Windows 的系统结构如图 1.1 所示。其中，跟文件存取相关的部分可以表示如图 1.2 所示。

文件存取相关的原理可以描述如下：应用程序调用 kernel32.dll 提供的 Windows API，kernel32.dll 中的 Windows API 调用 kernel32.dll 中的 Native API，或者由应用程序直接调用 ntdll.dll 提供的 Native API，发出一个请求操作。I/O 管理器接收到上层传来的 I/O 请求，通过构造输入输出请求包 IRP(I/O Request Package)来描述这个请求，然后向下传递给文件系统驱动、存储设备驱动做后续处理，低层驱动处理完毕后把结果依次向上返回，最后经过 I/O 管理器，把结果返回给 API 函数的返回值。

1.2.2 硬件加解密

现存的加密手段主要有软件加密和硬件加密两种。下面的几种透明加解密的实现都是通过软件来实现的。常用的硬件加解密方法有订制IC，FPGA，PLD，单片机等。[①]硬件实现的加密速度快、安全性高，但不灵活，算法改变和密钥改变比较困难，不容易进行升级，并且成本很高。所以选择了软件的加密算法。

[①]关于硬件加密的算法可以参见文献[13]。

火龙果•整理
uml.org.cn

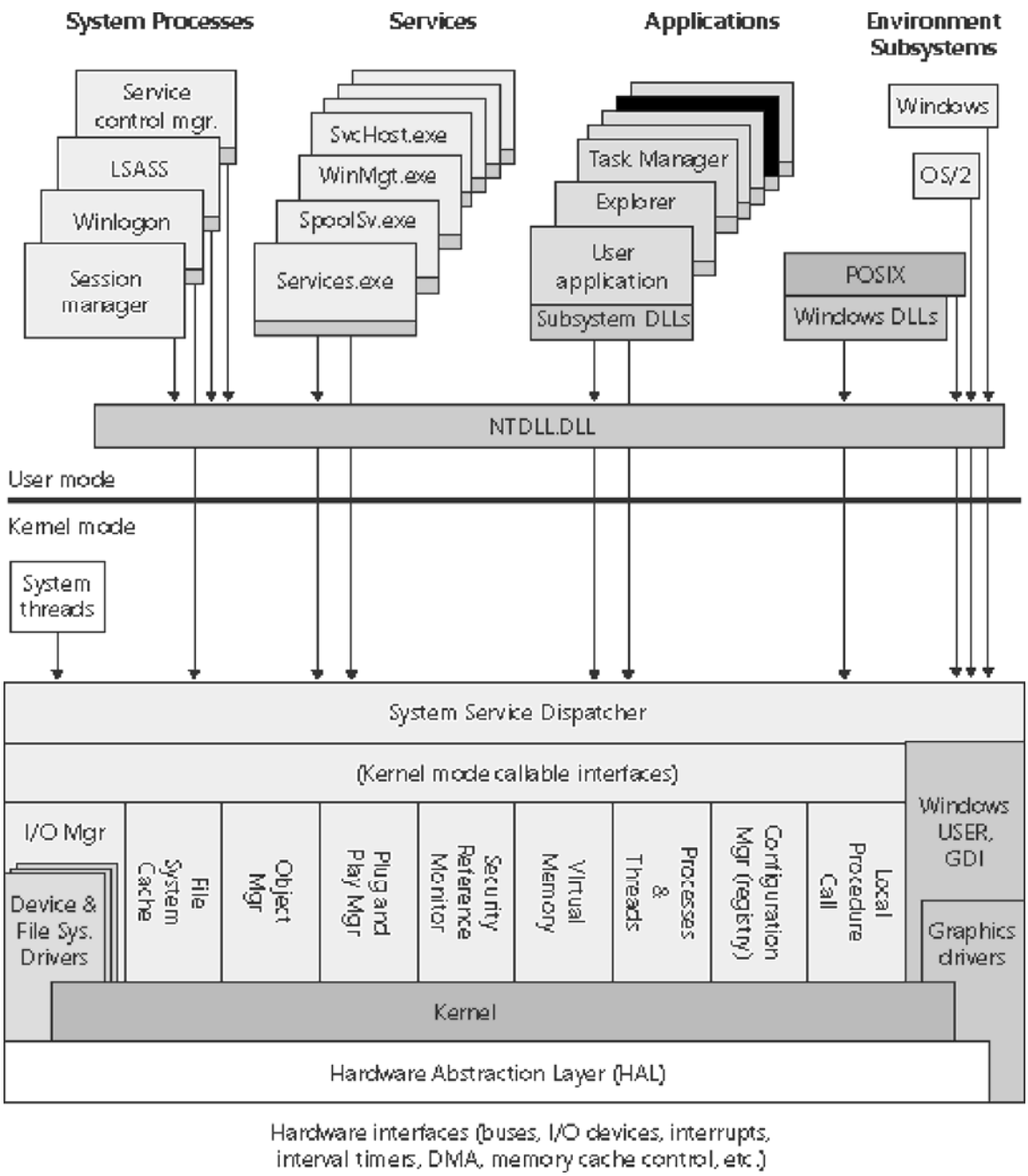

图1.1　Windows系统架构[②]

[②] 图来源于参考文献[2]，第 89 页

火龙果•整理
uml.org.cn

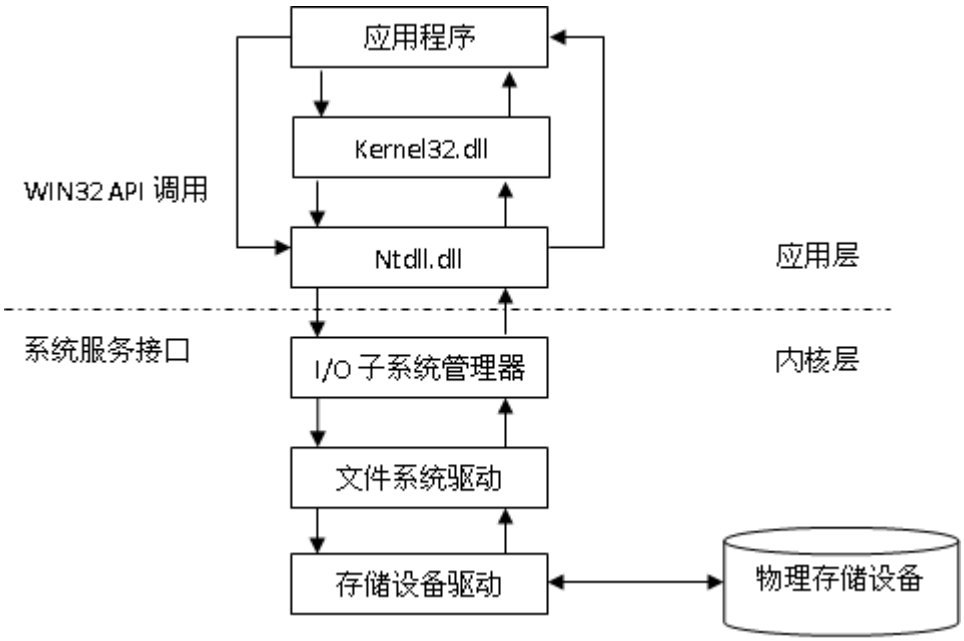

图1.2　Windows系统跟文件相关工作原理

1.2.3 驱动层透明加解密

如果在文件系统驱动层进行加密处理，就是驱动层的透明加解密技术。如图1.3 所示。

由图 1.3 可以看出，在正常的 Windows NT 系统上多出了一个加密单元，这个加密单元主要处理文件系统的一些操作，进行加解密。微软的 NTFS 的 EFS 就是一个这样的系统，因为文件系统拥有以文件或目录的粒度访问文件的能力，所以加密文件系统能够灵活的控制加密的范围，根据特定的文件类型判断是否进行加密和解密。但是微软的这个系统只能在 NTFS 文件系统基础上的，不支持 FAT 和 NTFS 的较低版本，不具有系统通用性。并且 EFS 是微软自己的产品，而对于第三方来说，在很多驱动程序没有文档化的情况下，开发这样的一个透明加解密的难度可想而知。并且驱动层程序的设计本身就很复杂，所以在本文中，并没有采用这种透明加解密的方法。

火龙果•整理
uml.org.cn

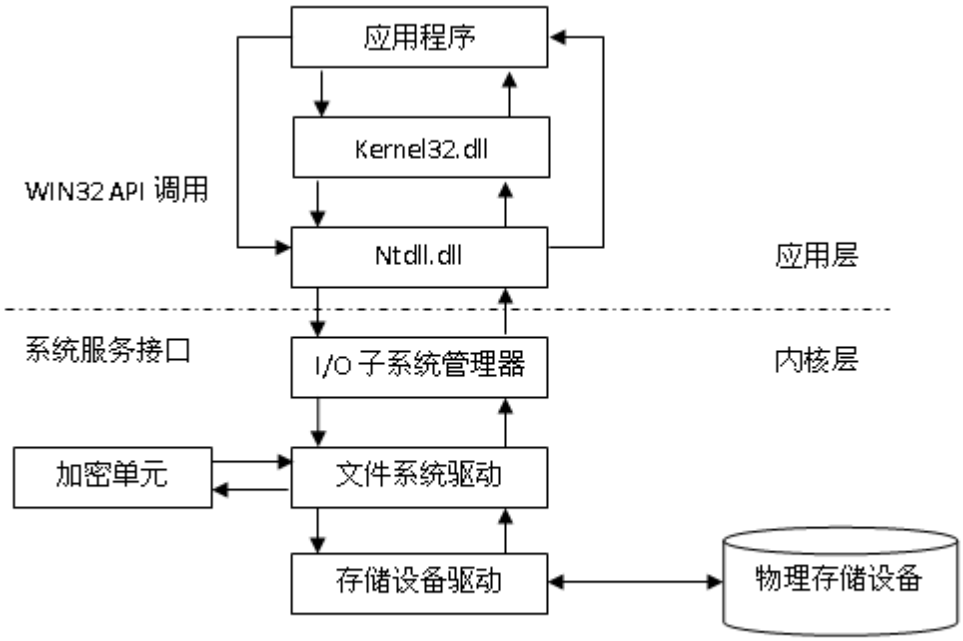

图1.3　驱动层实现透明加解密的示意图

1.2.4 过滤驱动层透明加解密③

过滤驱动层实现的透明加解密是在 I/O 管理器和文件系统驱动之间加入一个过滤驱动层。如图 1.4 所示。

文件系统过滤驱动程序是针对文件系统而言的。Windows NT 的 I/O 管理器根据用户的文件操作请求构造 IRP 发到文件系统驱动程序，文件系统驱动程序把相应于文件系统的操作转换为相应于存储设备驱动程序的操作并通过 I/O 管理器来调用存储设备驱动程序。I/O 管理器在发送操作请求到目标设备之前会检查是否有其他的设备附着于目标设备对象之上。通过构造附加的设备对象附着在文件系统或者存储设备对象之上，并为该设备对象指定专门的驱动程序，I/O 管理器就会把要发送到目标设备的请求先发到附加的过滤设备，在该设备对应的过滤驱动程序中对原始请求加入附加的处理来实现对文件系统操作的截取、监控甚至替换。

③ 以上内容引用了文献[6]。

火龙果•整理
uml.org.cn

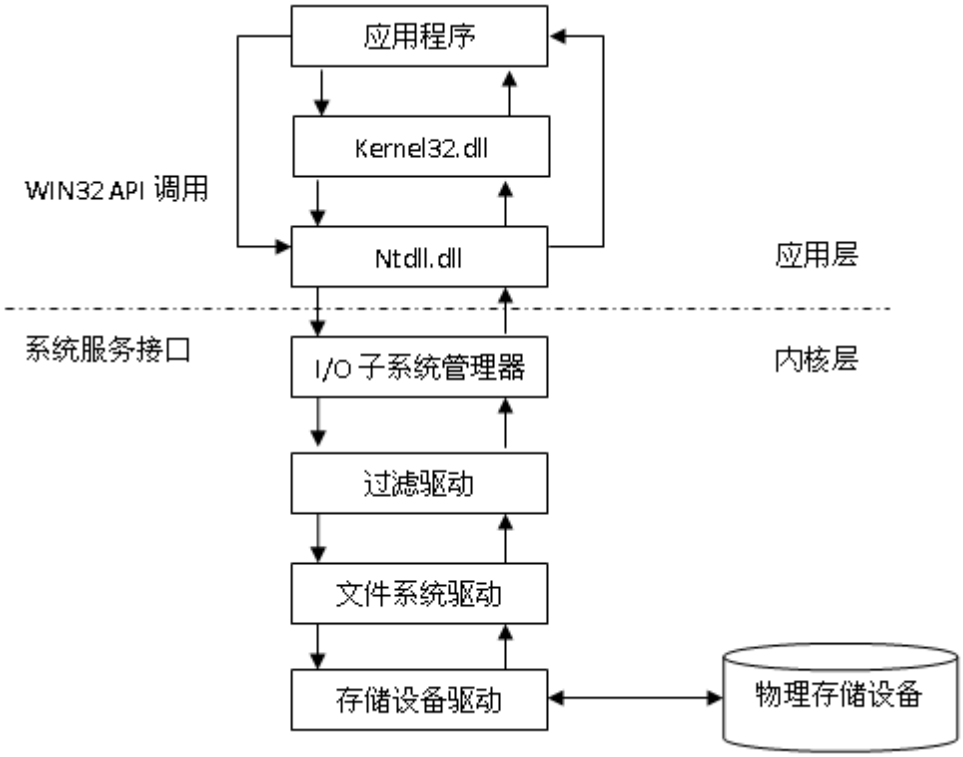

图1.4　过滤驱动层实现的透明加解密

下面给出一个在 Window NT 下进行文件加密解密的过滤器具体实现过程。用户的数据在存储到磁盘上设定的安全区域(例如安全文件夹)之前经过了加密处理，读取时经过解密返回给用户，这个过程对用户来说是透明的。对所有文件系统分区的文件实现透明的加密解密并采用特殊的加密算法，就需要通过文件系统过滤驱动程序来实现。在文件系统驱动程序和硬盘驱动程序之间加入过滤驱动程序。当文件系统驱动程序写硬盘的时候，过滤驱动程序首先对数据进行加密然后再调用硬盘驱动程序。当文件系统驱动程序读硬盘的时候，过滤驱动程序先调用硬盘驱动程序读取数据并对从硬盘返回的已加密数据进行解密，然后再返回给文件系统驱动程序。过滤器在系统中位于文件系统驱动程序和磁盘驱动程序之间。

现在过滤驱动透明加解密的技术已经很成熟，产品也已经很多。显然这是一种比较好的实现方法，但是本文也没有选择这种方法，原因一方面是这个技术已经很成熟，继续研究的意义不大很难取得新的突破；另一方面，相对于以下要介绍的方法来说，这个方法的安全性还是稍差。

火龙果•整理
uml.org.cn

1.2.5 用户层API HOOK透明加解密

用户层通过下钩子拦截 Windows 操作系统提供给用户的不同层次的 API，截获读写文件的 API，在读文件的时候把硬盘的加密数据解密到内存，在写文件的时候把内存数据加密写到硬盘上。如图 1.5 所示。

目前，对这种技术的普遍的评价是，效率低、稳定性和一致性差，不适合于大型系统的开发。因此，关于这类技术的研究和产品都十分少见。关于这种技术的产品有武汉天喻公司的产品，但这个产品只是针对一些图形软件，也并不具有普遍性。

尽管如此，本论文还是选择这种技术来研究。因为这种技术进行加解密的位置更靠近内存，能保证明文数据尽量只存在进程的内存空间中，通过保护进程的内存空间就保护了数据文件。

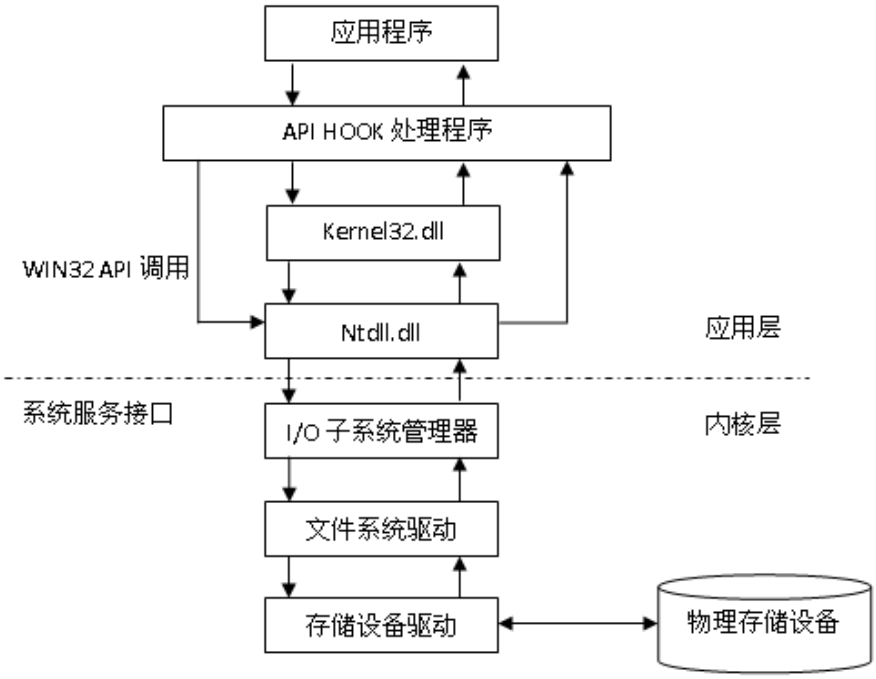

图1.5　API HOOK实现透明加解密

1.2.6 几种技术的比较

在几种透明加解密的方法中，每种方法都有自己的优势和缺陷。每种方法的优缺点在之前的部分已经都有介绍了。总的来说，从最安全的角度看，API HOOK

火龙果•整理
uml.org.cn

的方法使明文数据暴露的最少，因此如果能进行有效控制，那么将会是最安全的方法。但从效率和设计的工作量上来说，API HOOK 的方法确实最大的。

虽然 API HOOK 工作量很大，但是还是有章可循的。如果能够把应用软件应用 API 的方式分为几类，还是有可能实现一个集成的系统的。本论文也是基于这个目的进行研究和探索。

1.3 系统概览与论文结构

本论文主要实现两部分的功能，第一部分是存取文件的透明加解密，即打开解密，保存加密；第二部分是文件拷贝透明加解密，即文件考出加密，文件考入解密。透明加解密是通过下钩子来实现的，钩子是使用 madCollection 的开发包。通过钩子把信息传回给一个处理系统进行处理。这个处理系统自己维护一个链表，用于解决文件句柄和文件名的对应问题。文件是否加密是通过文件头判断的。加密文件有一个特殊的文件头作为标识。加密解密需要的密码算法，在这个实验中，传统的分组密码和序列密码都不能使用，这个会在以后的介绍中详细解释原因，因此采用何种密码是需要自己的设计和研究的。既保证效率不受太大的影响，也保证密码破译具有很大的难度。

系统的实现需要大量的研究和观察，分析不同软件的行为需要记录并且阅读大量的日志文件。记录分析也是这个实验过程中必不可少的一个部分，因此本文也要花一定的篇幅来把实现透明加解密的软件的行为进行一定的分析。

最后，由于完成这篇论文的时间有限，很多新的想法并没有能够在实验中实现。因此，提出对未来的论文中没有实现但已经详细研究过的内容的研究方向。并且论文研究的内容也比较具有商业价值，现在市场上存在这类的软件，但是只是局限在了某一种软件控制上，因此，提出了一个集成多种软件的透明加解密系统的实现的设想。

火龙果•整理
uml.org.cn

第2章 研究的相关理论和工具介绍

2.1 用于API注入的MadCollection开发包

2.1.1 MadCollection简介

MadCollection 中 madCodeHook 这个开发包提供了注入大多数 API 的功能。通常，钩子只是被注入到当前的进程中，但是 madCodeHook 依然提供了把一个特定的 dll 注入到任何一个 32 位的进程中的功能，甚至可以注入到整个系统所有的进程中，包括所有正在运行的进程或者将要运行的进程。所以，通过把自己的代码编译成一个 dll 文件然后注入到整个系统所有进程中，就可以容易并且有效的实现对整个系统所有应用进程 API 的注入。

madCodeHook 主要提供了四个方面的应用：（1）分别是 API 或代码的注入；（2）注入 DLL；（3）进程间通信（IPC）功能；（4）工具函数。下面分别对之进行一些简单的介绍。

(1) 使用“HookCode”或者“HookAPI”能够对任何一个函数和 API 下钩子。钩子的回调函数和 next hook 变量的参数必须完全一致。提供的程序代码如下：

```
function HookCode (code:pointer;callbackFunc:pointer;
         out nextHook:pointer;flags:dword=0):bool;stdcall;
function HookAPI  (module,api:pchar;callbackFunc:pointer;
         out nextHook:pointer;flags:dword=0):bool;stdcall;
```

(2) 如果要对一个其他进程或者系统所有的进程下钩子的话，需要编写一个 DLL 文件来下钩子，这个 DLL 文件需要被加载到目标进程中。把一个 DLL 注入到另外一个进程中是没有文档化的，通常会使用“SetWindowsHookEx”，但如果把这个 API 用在这个地方会有很多不便之处：首先，它会影响性能，因为这样的话其实是安装了一个真的消息钩子，这个钩子会时刻被调用；第二，它只是在进程处理消息的时候才工作，但不是所有的进程都会处理消息；第三，它只是在目标进程没有被阻塞，冻结或者崩溃的时候才能使用；第四，DLL 将会比预期晚一点被加载到目标进程中，这就导致了一些可能的对 API 调用的遗失。

火龙果•整理
uml.org.cn

而 madCodeHook 的开发包提供的这项功能是不会有这些不便之处的。

“InjectLibrary”这个函数可以把 DLL 注入到所有正在运行的 32 位的进程中。提供的接口如下：

```
function InjectLibrary (processHandle:dword;libFileName:string;
            timeOut:dword = 7000) : boolean;
function InjectLibrarySession(session:dword;systemProcesses:bool;
            libFileName:string; timeOut: dword=7000) : boolean;
```

(3) 当一个 DLL 被注入到另一个进程中的时候，如果这个 DLL 需要跟原来的进程进行通信，可以使用 IPC 相关的函数。相关的函数接口如下：

```
function CreateIpcQueueEx(ipc:pchar; callback:TIpcCallback;
        maxThreadCount:dword=16;maxQueueLen:dword=$1000):bool;stdcall;
function CreateIpcQueue(ipc:pchar;callback:TIpcCallback ):bool;stdcall;
function SendIpcMessage(ipc:pchar; messageBuf:pointer;messageLen:dword;
       answerBuf:pointer=nil;answerLen:dword=0;
       answerTimeOut:dword=INFINITE;handleMessages:bool=true):bool;stdcall;
```

(4) 由于提供的一些功能函数本论文中基本没有用到，所以这里就不再详述。如果对此感兴趣，可以参阅相关资料。[④]

2.1.2 MadCollection在本论文的中的应用

本文中大量的使用到 MadCollection 这个开发包，所有的 API 注入都是通过它来实现的，使用到了它对 API 下钩子和注入 DLL 的功能。另外，在实现复制透明加解密过程中，还用到了进程间通信的功能。在分析几种软件行为的时候还用到了这个开发包注入到几种软件中记录日志，保存几种软件调用 API 的行为，对分析软件行为起到了很大的作用。

[④] 关于MadCollection的详细资料请参见网站http://www.madshi.net。

火龙果•整理
uml.org.cn

2.2 涉及的API函数的研究

2.2.1 Shell32.dll中相关的API函数

Shell32.dll 是 Windows 壳 Shell 相关应用程序接口动态链接库文件。Shell32.dll 中共包括了 461 个导出函数。本文中主要使用了 SHFileOperation 用于拦截文件复制、移动的操作，进行复制的透明加解密操作。下面对 SHFileOperation 进行详细的描述：

SHFileOperation

```
SHFileOperation(const lpFileOp:TSHFileOpStruct :Integer; stdcall;
```

参数

TSHFileOpStruct：一个包含了需要进行操作的 `SHFILEOPSTRUCT` 结构信息的指针。

返回值

成功就返回 `0`，否则返回非 `0`。

_SHFILEOPSTRUCTA

```
  _SHFILEOPSTRUCTA = packed record
        Wnd                    : HWND;
        WFunc                  : UINT;
        PFrom                  : PAnsiChar;
        PTo                    : PAnsiChar;
        FFlags                 : FILEOP_FLAGS;
        FAnyOperationsAborted  : BOOL;
        HNameMappings          : Pointer;
        LpszProgressTitle      : PAnsiChar;
  end;
```

成员

Wnd：用显示文件操作状态的句柄

WFunc：执行的操作，可以是 FO_COPY；FO_DELETE；FO_MOVE；FO_RENAME。

PFrom：一个或多个源文件名称缓冲区地址。多个文件间空分隔。文件列表必须是双空终止。

PTo：地址的缓冲区包含目标文件或目录的名。如果 fFlags 成员指定 FOF_MULTIDESTFILES 缓冲区可包含多个目标文件名称。多名称必须是空分隔。名称列表必须是双空终止。

火龙果•整理
uml.org.cn

fFlags：文件操作控制标志。

FanyOperationsAborted：用户完成之前,终止任何文件操作，设置 TRUE 或 FALSE 值。

HnameMappings：文件名称映射对象的句柄，包含一个数组的 SHNAMEMAPPING 结构。每个结构包含每个文件被移动复制或重命名旧和新路径。只有 FFlags 成员包括 FOF_WANTMAPPINGHANDLE 标志时使用此成员。使用 SHFreeNameMappings 函数必须释放该句柄。

LpszProgressTitle：用作标题进度对话框字符串地址。FOF_SIMPLEPROGRESS 标志使用。

2.2.2 Kernel32.dll中相关的API函数⑤

Kernel32.dll 是一个重要的系统文件，它封装了 Windows 操作系统提供内用户的 API 接口，可以供用户实现所有 Windows 系统提供的功能和服务的调用。这个层次上的 API 函数很多，可供用户实现不同的功能，跟文件读写的操作也有很多。下面主要把跟本论文研究相关的 API 函数一一列举，详见表 2.1。

表2.1 Kernel32.dll中封装的部分API

名称	说明
CreateFileW	以 Unicode 方式创建或打开文件
ReadFileW	以 Unicode 形式读文件
WriteFileW	以 Unicode 形式写文件
SetFilePointer	设置文件指针的位置

对每个函数的介绍如下：

CreateFileW

```
CreateFileW(lpFileName : PWideChar; dwDesiredAccess : Integer;
    dwShareMode : Integer;lpSecurityAttributes : PSecurityAttributes;
    dwCreationDisposition : DWORD;dwFlagsAndAttributes : DWORD;
    hTemplateFile : THandle;): THandle;
```

参数

lpFileName：一个指向文件名字符串的指针，用于文件的创建或者打开。

dwDesiredAccess：指定存取文件的方式。

⑤ 要获取 kernel32.dll 中 API 的详细介绍请参见参考文献[15]。这里只对用到的做简要介绍。

火龙果•整理
uml.org.cn

dwShareMode：指定文件在不同程序之间的共享方式，如果这个值是 0，则文件不能共享，其它打开这个文件的操作将会失败。

lpSecurityAttributes：指向一个包含句柄继承和文件安全的结构，这个指针可以被设置成 nil，暗示子进程不能继承这个句柄。

dwCreationDisposition：如果一个文件已经存在或者不存在，通过这个参数来指定。

dwFlagsAndAttributes：指定文件的属性和存取方式。如果是打开文件，这个参数可以被忽略。

hTemplateFile：给要打开的文件指定一个之前只读打开文件的句柄，被创建的文件将会获得所有只读打开文件的属性。如果是打开文件，则忽略这个参数。

返回值

如果文件创建或者打开成功，返回文件的句柄；如果失败返回 INVALID_HANDLE_VALUE。可以通过 GetLastError 获得错误代码。

备注

文件使用完后，应该调用 CloseHandle 释放句柄。

ReadFileW

```
ReadFileW(hFile : THandle;var Buffer;nNumberOfBytesToRead : DWORD;
    var lpNumberOfBytesRead : DWORD;lpOverlapped : POverLapped): BOOL;
```

参数

hFile：要读得文件的句柄。这个文件必须是以可读的方式打开的。

Buffer：指向一个接受读到内容的内存空间。

nNumberOfBytesToRead：指定从文件要读得字节数。

lpNumberOfBytesRead：一个指向 DWORD 类型变量的指针，记录从文件中实际读出的字节数。在读之前这个值应该被初始化为 0，如果 lpOverlapped 参数是空，那么这个参数指针一定不能为空。

lpOverlapped：一个指向 TOverlapped 结构的指针。如果 hFile 这个句柄对应的文件是以 FILE_FLAG_OVERLAPPED 标志打开的话，这个参数不能为空，并且读的指针的位置是被这个结构里的 offset 指定的，ReadFile 可能在读操作完成之前就返回。返回值是 FALSE，GetLastError 得到的结果是 ERROR_IO_PENDING，在结构中指定的事件将会以信号的方式完成读的操作。如果没有以 FILE_FLAG_OVERLAPPED 方式打开文件但这个指针不为空，只有读文件的位置是由这个结构指定的，ReadFile 不会提前返回。如果既没有以 FILE_FLAG_OVERLAPPED 方式打开，这个标志也为空的话，读文件开始的指针指在文件开始的位置，不会提前返回。

返回值

如果读文件成功，返回 TRUE；如果失败返回 FALSE。可以通过 GetLastError 获得错误代码。

备注

这个函数在指定的文件句柄的文件中读取指定大小的数据放到指定的内存中。通常来说，这个函数的行为是跟读的文件打开的方式相关的。如果文件没有以 FILE_FLAG_OVERLAPPEDDE 的方式打开的话，那么文件的初始指针的位置是文件头，并且每读一次文件，文件指针都会随着移动相应的字节。如果是以那个标志打开的话，则需要手动的设置文件读取的指针。

WriteFileW

```
WriteFile(hFile : THandle;const Buffer;nNumberOfBytesToWrite : DWORD;
   var lpNumberOfBytesWritten:DWORD;lpOverlapped : POverlapped:BOOL;
```

这个函数参数、返回值与 ReadFile 基本一致，可参见。

TOverlapped

```
TOverlapped = record
       Internal : DWORD;
       InternalHigh : DWORD;
       Offset : DWORD;
       OffsetHigh : DWORD;
       hEvent: THandle;
     end;
```

成员

Internal：这个成员保留给操作系统内核使用。

InternalHigh：这个成员保留给操作系统内核使用。

Offset：指定文件指针双字的低位一个字。

OffsetHigh：指定文件指针双字的高位一个字。

hEvent：事件的指针，这个事件在读写的操作完成后发信号改变 WriteFile 或 ReadFile 的状态。

SetFilePointer

```
SetFilePointer(hFile : THandle;lDistanceToMove : Longint;
    lpDistanceToMoveHigh : Pointer;dwMoveMethod : DWORD): DWORD;
```

火龙果•整理
uml.org.cn

参数

hFile：文件指针要被重定位的文件句柄。这个文件需要被读或者写的方式打开。

lDistanceToMove：文件指针的双字中的低位移动的距离的字节数。正值表示向前移动，复制表示向后移动。

LpDistanceToMoveGigh：文件指针双字节中的高位移动字节数的指针。在这个函数返回后，指针指的位置也可以继续接受变化。

DwMoveMethod：要移动的起始的位置。

返回值

如果操作成功，返回移动的双字的低位的字；如果失败返回$FFFFFFFF。通过 GetLastError 获得错误代码。如果 lpDistanceToMove 不是 nil 且函数失败了，那么返回的是 NO_ERROR。

备注

这个函数在指定的文件句柄的文件中重新指定文件指针的位置。设置文件指针通过在指定的文件位置的基础上进行偏移得到的。

如果文件是以 FILE_FLAG_NO_BUFFERING 方式打开的，那么文件的指针只能以磁盘扇区的大小增加，磁盘扇区的大小可以通过调用 GetDiskFreeSpace 这个函数得到。

2.2.3 Ntdll.dll中相关的API函数

User32.dll,kernel32.dll,shell32.dll,gdi32.dll,rpcrt4.dll,comctl32.dll,advapi32.dll, version.dll 等 dll 代表了 Win32 API 的基本提供者。Win32 API 中的所有调用最终都转向了 ntdll.dll，再由它转发至 ntoskrnl.exe。ntdll.dll 是本机 API 用户模式的终端。真正的接口在 ntoskrnl.exe 里完成。事实上，内核模式的驱动大部分时间调用这个模块，如果它们请求系统服务。Ntdll.dll 的主要作用就是让内核函数的特定子集可以被用户模式下运行的程序调用。Ntdll.dll 通过软件中断 int 2Eh 进入 ntoskrnl.exe，就是通过中断门切换 CPU 特权级。比如 kernel32.dll 导出的函数 DeviceIoControl 实际上调用 ntdll.dll 中导出的 NtDeviceIoControlFile。

Ntdll.dll 是操作系统提供给用户在用户层使用的最底层的 API 集合，共有 249 个。这些 API 函数很多是未文档化的函数，再加上函数参数很多是和系统内核结合很紧密很原始的形式，因此使用起来不是很方便。

本文中使用到的几个 ntdll.dll 提供的 API 主要是因为 kernel32.dll 函数已经被下了钩子，如果还用 kernel32 所提供的 API 可能导致错误甚至是系统的不稳定。

火龙果•整理
uml.org.cn

另一方面，本文所研究的一些微软自己的软件绕过了 kernel32.dll 所提供的 API 函数，因此需要对 ntdll.dll 中的部分 API 进行拦截。

本文中使用的函数详见表 2.2。

表2.2　Ntdll.dll中封装的部分API

名称	说明
NtReadFile	读文件
NtWriteFile	写文件
NtQueryInformationFile	获取文件的信息，在本文中主要是获取文件指针位置

NtReadFile

```
NtReadFile(FileHandle : THandle;Event : THandle;
    ApcRoutine : TIoApcRoutine;ApcContext : Pointer;
    var IoStatusBlock : TIoStatusBlock;Buffer : Pointer;
    Length : ULONG;ByteOffset : PLARGE_INTEGER;
    Key : PDWORD) : NTSTATUS; stdcall;
```

参数

FileHandle：要读得文件的文件句柄。这个文件需要被 FILE_READ_DATA 的方式打开。

Event：事件的句柄，当操作完成的时候发信号。文件必须有 EVENT_MODIFY_STATE 权限。

ApcRoutine：指定一个程序，当操作完成时执行。

ApcContext：一个任意类型的指针，可以为 ApcRoutine 提供上下文信息。

IoStatusBlock：指向一个接受最后状态和被请求信息的变量。Information 是读取的字节数。

Buffer：指向一个用户自己申请的内存空间用于存放从文件中读取的内容。

Length：指定要读取的字节数。

ByteOffset：指向一个用于确定读的开始位置的变量。

Key：指向一个变量，如果这个变量的值跟文件被锁定的范围的值匹配，文件锁可以被忽略。

返回值

返回 STATUS_SUCCESS，STATUS_PENDING 或者错误的状态。

备注

这个函数是 kernel32.dll 中封装的 ReadFile 和 ReadFileEx 需要调用的函数。

火龙果•整理
uml.org.cn

NtWriteFile

```
NtWriteFile(FileHandle : THandle;Event : THandle;
    ApcRoutine : TIoApcRoutine;ApcContext:Pointer;
    var IoStatusBlock : TIoStatusBlock;Buffer : Pointer;
    Length : ULONG;ByteOffset : PLARGE_INTEGER;
    Key : PDWORD) : NTSTATUS; stdcall;
```

这个函数参数、返回值与 ReadFile 基本一致，可参见。

NtQueryInformationFile

```
NtQueryInformationFile(const FileHandle : THandle;
    var IoStatusBlock : TIoStatusBlock;FileInformation : Pointer;
    Length : ULONG;FileInformationClass : TFileInformationClass
   ) : NTSTATUS; stdcall;
```

参数

FileHandle：要查询的文件的文件句柄。

IoStatusBlock：指向一个接受最后状态和被请求信息的变量。

FileInformation：指向一个用户自己分配的空间用于存储被请求的文件的信息。

Length：FileInformation 的字节数。用户需要根据请求的文件信息的类型分配。

FileInformationClass：指定需要查询的类型和文件信息。允许的类型是一个枚举类型的子集。

返回值

返回 STATUS_SUCCESS，或者错误的状态。

备注

这个函数是 kernel32.dll 中封装的 GetFileInformationByHandle，GetFileSize，GetCompressedFileSize，GetFileTime 需要调用的函数。

TFileInformationClass[6]

```
TFileInformationClass
    = (……FilePositionInformation……);
```

备注

本文只需要使用其中的 FilePositionInformation，来获得文件指针的位置。

[6] 详情参见参考文献[14]。这里只列举本实验相关的一个。

火龙果•整理
uml.org.cn

2.3 分组密码和序列密码

2.3.1 分组密码及AES算法⑦

现代分组密码的研究始于 20 世纪 70 年代中期，至今已有 20 余年历史。分组密码的原理是把明文数据分为若干相同大小的组，组的大小是跟不同的加密算法相关的。一次对一组加密，得到密文数据，再把密文数据拼接起来就得到数据的密文。解密时也是取相同大小的组，分别解密，最后再把明文拼接起来就得到了原来的明文。

分组密码中还有更细致的分类，包括对称密码算法和非对称密码算法。分组密码算法也要考虑如果分组结果最后一组没有足够的数据使得最后一组的大小跟其他组的大小一样的话，需要进行填补的问题。

密码学是一门复杂的学问，本文重点不在研究密码算法和密码强度上，所以在这里就不再多说。本文选择了 AES 这种已经研究得很成熟的分组密码作为系统的加密工具。下面，简单的介绍下 AES 加密算法。

AES 算法属于分组密码算法，它的输入分组、输出分组以及加/解密过程中的中间分组都是 128 比特。密钥的长度 K 为 128，192 或 256 比特。用 Nk = 4，6，8 代表密钥串的字数(1 字 = 32 比特)，在本文应用的程序中可以选定，但默认为 4。用 Nr 表示对一个数据分组加密的轮数(加密轮数与密钥长度的关系见表 2.3)。

表2.3 加密轮数与密钥长度的关系

AES 类型	密钥长度	分组大小	加密轮数
AES-128	4 字	4 字	10
AES-192	6 字	6 字	12
AES-256	8 字	8 字	14

每一轮都需要一个和输入分组具有同样长度(128 比特)的扩展密钥 Ke 的参与。由于外部输入的加密密钥 K 长度有限，所以在 AES 中要用一个密钥扩展程序(Key Expansion)把外部密钥 K 扩展成更长的比特串，以生成各轮的加密密钥。

AES 的加密与解密框图如图 2.1 所示。

⑦ 此小结参见了参考文献[16]，包括本小结的图和表均来源于此篇文献。

火龙果•整理
uml.org.cn

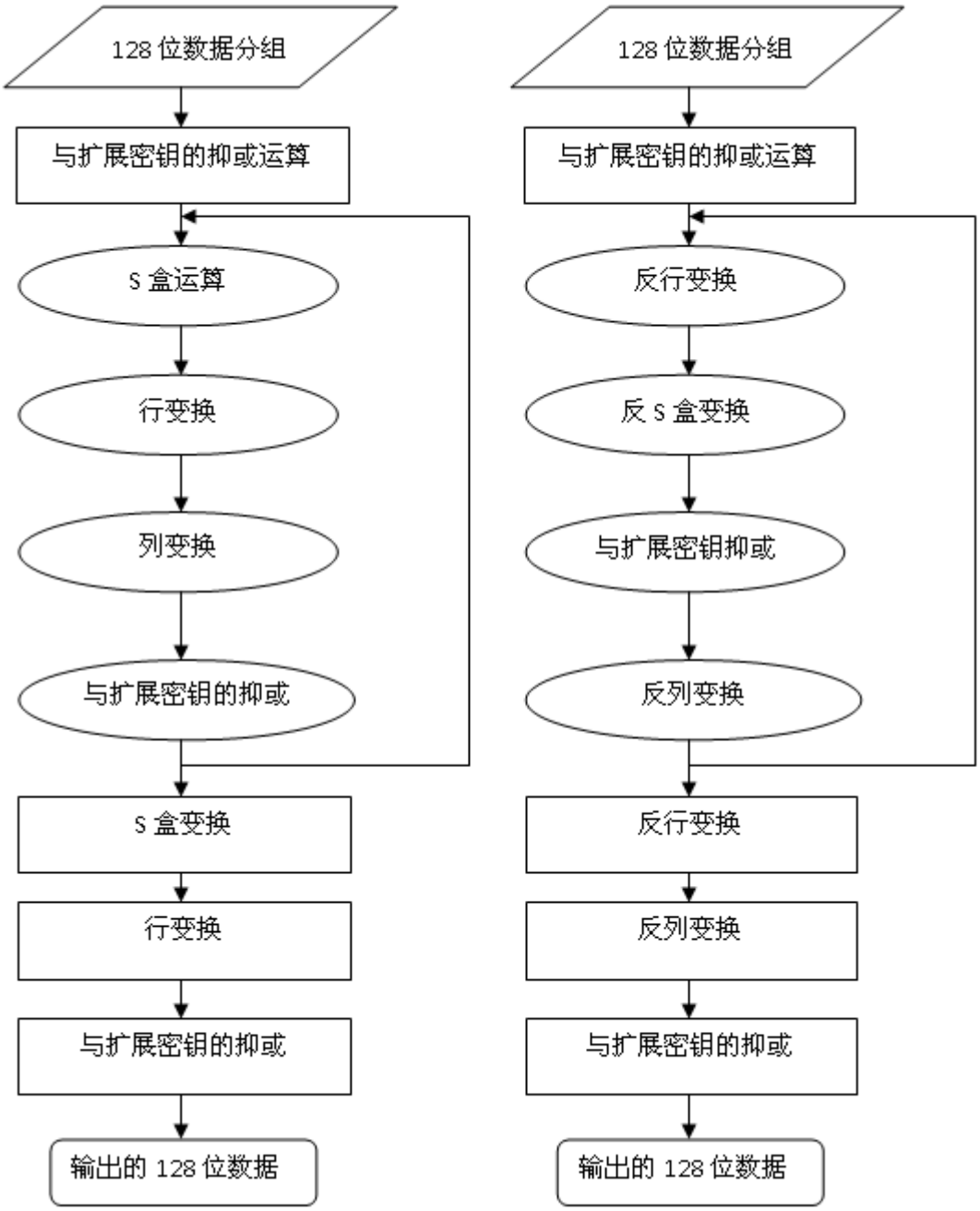

图2.1　AES的加密与解密框图

本文应用了一个线程的 AES 的加密模块作为系统的一部分，本文实现的大部分的加密算法都是应用这个算法的。

火龙果•整理
uml.org.cn

2.3.2 序列密码及RC4 算法

一个典型的流密码每次加密一个字节的明文，流密码也可被设计为每次操作一比特或者大于一个字节的单元。流密码的结构是密钥输入到一个伪随机数（比特）发生器，该伪随机数发生器产生一串随机的 8 比特数。简单地说，一个伪随机流就是在不知道输入密钥的情况下，不可预知的流。发生器的输出称为密钥流，通过与同一时刻一个字节的明文流进行异或（XOR）操作产生密文流。

序列密码的安全性跟产生的密钥有关。首先是密钥的周期要足够的长，周期就是密钥出现重复之前的长度；其次，密钥要尽量随机，即产生的 0 和 1 的个数要尽量相等，如果是字节流，则 256 个可能的字节出现频率也应尽量的相等。

本文中选择了一种典型的流加密算法 RC4 实现并应用。至于选择序列密码和 RC4 的原因将在后面部分中详细阐述。RC4 算法设计起来十分简单。描述如下：

（1） 初始化一个 256 字节的数组 S

将 S 数组中对应位置的元素赋值为下标的数字，即 S[0] = 0，S[1] = 1，……，S[255] = 255。

假设密钥为 K。建立一个临时数组 T。把 K 循环付给 T。伪码如下：

```
For I = 0 to 255 do
S[i] = i
T[i] = k[I mod keylen]
```

然后利用 T 对 S 进行置换，伪码如下：

```
J = 0
For I = 0 to 255 do
begin
   j = (j+s[i]+s[j]) mod 256
   swap(s[i], s[j])
end
```

（2） 生成密钥流

用 S[i]跟 S 中另一字节置换，当 S[255]置换完以后，从 S[0]接着开始，一直生成需要长度的密钥流。伪码如下：

火龙果•整理
uml.org.cn

```
I, J = 0
While(True)
Begin
    I = (I + 1) mod 256
    J = (j + S[i]) mod 256
    Swap(S[i], s[j])
    t = (S[i] + S[j]) mod 256
    K = S[t]
```

其中，K 则为密钥流。可以用 K 跟原文进行抑或。

火龙果•整理
uml.org.cn

第3章 系统的分析与设计

3.1 系统设计的目标

3.1.1 系统设计的前提假设

理论上，计算机系统是没有绝对的安全的概念，再严密的防护都会有被攻破的可能。但是，不是说研究人员就不要研究安全的问题，而是说可以基于一定的假设构造一个相对安全的系统。

首先，本文假设存在进程私有内存空间的数据是安全的。这个假设的依据一方面是：数据必须被加载到进程的内存空间数据才是可用的，因此数据必须保持原样的被加载到内存空间中。另外一方面是，进程间通讯本身是比较困难的。如果一个进程想访问另一个进程内存空间的内容，只有几种有限的方法。用SendMessage 发送 WM_COPYDATA 消息；文件映射（CreateFileMapping），最常用的，也是效率最高的；DDE（动态数据交换）；命名管道；socket 通讯。而这些方法都需要两个进程互相配合。对于本文研究的背景而言，可以访问文件的可信进程不会把自己的内存暴露给位置的非可信进程，因此这些方法都没有办法访问到可信进程的内存空间。但也不能说存在于一个进程空间的数据是绝对安全的，非可信进程可以把一个 DLL 注入到可信进程中，从而通过钩子获取进程的内容。而防止注入，保存进程的内存空间可以单独作为一个课题进行研究。因此，这里就假定一个可信进程的私有内存地址空间是安全的。

其次，本文假定存储在硬盘上的数据是不太安全的，需要进行保护的。但这种不安全是一种物理上的不安全，所面临的威胁就是硬盘被窃取。相比于这种可以通过其它方式排除的危险，保护的层次不需要太高。保护级别不高体现在两个方面：（1）保护的只是一些特殊的敏感文件；（2）加密的算法不需要太大的强度。

这种假设的依据是，对于普通的固定电脑及大型服务器，计算机硬盘丢失是一种特殊的情况，很少发生；对于笔记本电脑等便携式存储设备，更大的可能是一时失去控制，在一段时间内会找回，所以一般强度的密码足以保证在这段时间内数据不会遗失。另外，如果把所有的硬盘存储的文件全部加密，并且使用强度很高的密码，那么这种在用户层的加密必然会对系统的性能产生很大的影响。所以本文假设存储在硬盘上的数据只需要进行简单的加密保护即可。

火龙果•整理
uml.org.cn

最后，最需要防护的情况就是在数据文件离开本地磁盘的情况。这种情况的数据是最不安全的。但是实际中又不能限制所有的文件都不能离开本地磁盘，所以，对所有的离开本地磁盘的数据文件全部进行强度较高的加解密操作。

综上，对于系统，本文的假设如表 3.1 所示：

表3.1　数据存放位置的安全性假设

数据位置	安全性	保护级别
进程私有空间	安全	不用保护
本地磁盘	不安全	加密保护重要文件
非本地磁盘	特别不安全	加密保护所有考出的文件

3.1.2 数据文件保护的解决方案

基于以上的假设和分析，可知，可以通过加密的方式保护数据文件的安全。并且明文数据应该尽量的控制在可信进程的内存空间中，数据文件在硬盘里是密文存放的。

当打开一个数据文件是，密文数据一直被加载到进程内存空间中，然后被解密成明文在进程内存空间中被应用。当数据文件要被写到硬盘的时候，在进程的内存空间中数据被加密，然后写到硬盘中。一次过程如图 3.1 所示：

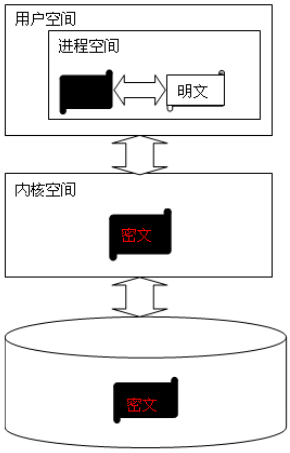

图3.1　数据文件存取透明加解密过程示意图

火龙果•整理
uml.org.cn

当要把一个数据文件复制到非可信磁盘的时候，由于跟文件读写不同，数据文件的内容不需要在复制的进程中被应用，所以在本文中只需拦截到复制进程发出的复制请求，获取复制的文件名和源、目的地址，然后就可以在自己的进程中把文件从本地的可信磁盘复制到本地非可信磁盘或者非本地磁盘。在复制完成后再给原来的复制的进程发送一个通知就可以了。如图 3.2 所示：

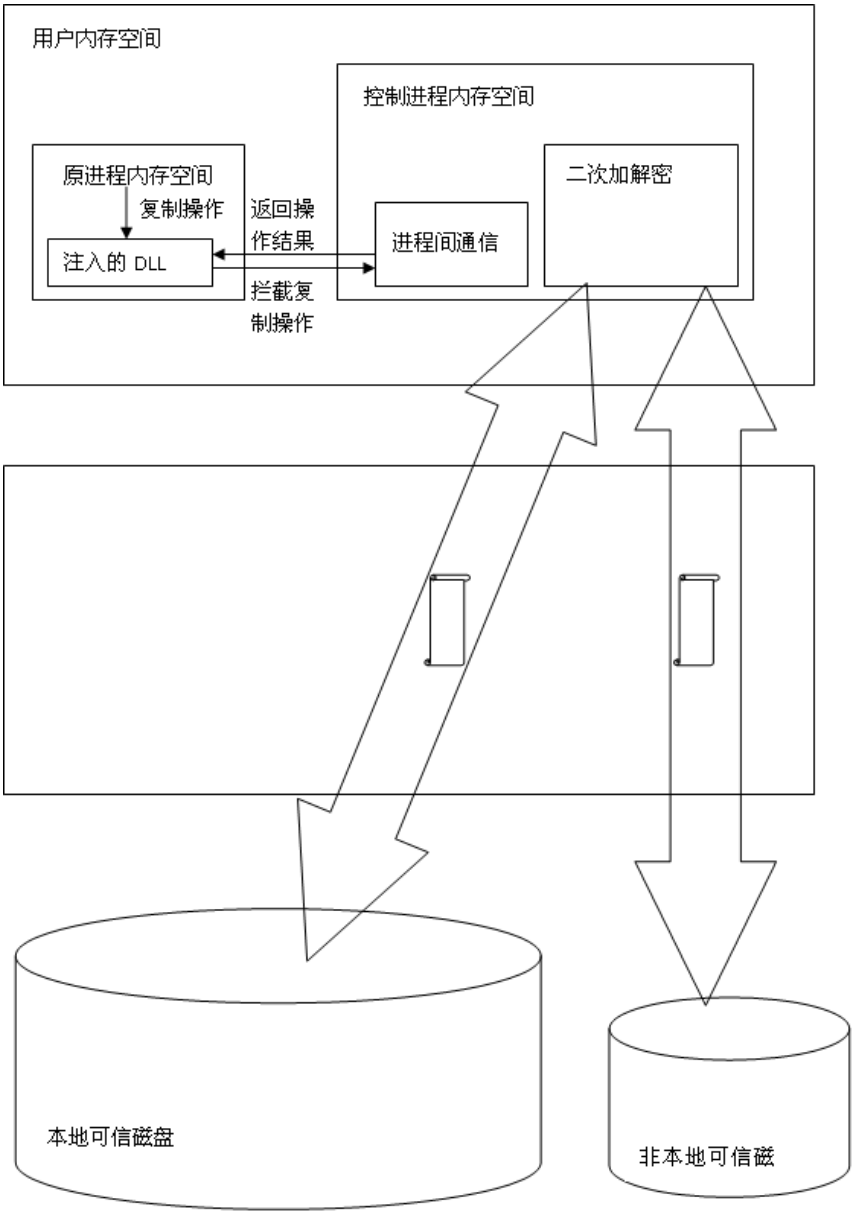

图3.2　数据文件复制透明加解密过程示意图

火龙果•整理
uml.org.cn

被复制到非本地可信磁盘的数据文件是经过二次加密的文件，一次加密是因为本身存储在磁盘的上的数据就是密文数据。然后再复制加解密的过程中又被加密一次。这种加密策略是可以被控制的，可以通过策略配置成一次加密。但在目前实现的系统中，是这样的实现的。

以上的部分简单的描述了本问所要解决的问题和所要达到的目标，在本章的后半部分，将会通过对一些细节的分析，构造出详细的理论模型。

3.2 软件及系统行为分析

3.2.1 系统的范围

要想构造系统详细的理论模型，需要明确两个问题。第一，那些存储在本地可信磁盘上的数据文件需要进行加密保护。第二，需要在哪个层面上对复制进行控制。

首先，在本论文中，需要进行保护的文件是通过进程和后缀控制的。把某些进程当成可信进程，把某些后缀的文件进行加密。选择的进程包括了 Autocad 系列软件和微软 Office 系列软件。控制的文件后缀既包括了这些软件通用的格式例如.dwg，.doc 等，还包括了这些软件生成的一些临时文件和备份文件。本论文要保护的进程及文件类型可以通过策略进行配置，进行不同级别的保护。保护的进程及后缀文件如表 3.2 所示。

其次，Windows NT 系统直接文件复制、剪切的操作都是通过 shell API 来完成的，所以，通过拦截 Shell API 中的 ShFileOperation 就可以完成对文件复制的拦截。本文所研究和实现的文件复制透明加解密也是基于这个基础上的。

3.2.2 分析的结果

在明确了本文要实现的目标和上述的这两个问题后，下面来分析下软件及系统的行为，为下面构造系统的理论模型做准备。分析软件及系统的行为主要是通过 AutoDebug，Filemon 和 ProcessMonitor 工具，以及自己编码实现的 API HOOK 工具记录日志来实现的。前三个工具都是比较普遍的工具，可以作为辅助，但是并不完全跟研究需要知道的信息是一致的。主要的分析工具是通过编码实现 API HOOK 然后记录日志。

火龙果•整理
uml.org.cn

表3.2　保护的进程及文件类型列表

进程	后缀类型	说明
Autocad	dwg	正常文件
	bak	备份文件，软件里可以设置是否有此类型文件
Word	Doc	2003 之前版本正常文件
	docx	2007 正常文件
	tmp	备份文件
Excel	xls	2003 之前版本正常文件
	xlsx	2007 正常文件
PowerPoint	ppt	2003 之前版本正常文件
	pptx	2007 正常文件

分析日志和使用上述的工具都是很复杂的工作，在分析软件行为的时候有时需要钩住所有 API 然后观察记录的日志，有时还会因为钩住太多的 API 导致系统响应时间太长，系统死机等情况的出现。

首先，来分析文件移动的操作。使用自己编写的 API HOOK 的工具，钩住 SHFileOperation 函数，在回调函数里仅仅通过写 INI 文件的方法输出日志，记录复制的源和目标地址，以及操作的代码（FO_COPY，FO_MOVE，FO_DELETE，FO_RENAME）代码如图 3.3 所示：

火龙果•整理
uml.org.cn

```
function SHFileOperationWCallBack(const lpFileOp: TSHFileOpStructW): Integer; stdcall;
var
  S, S1, S2: string;
  p : PWideChar;
begin
  case lpFileOp.wFunc of
    FO_COPY: S := 'Copy';
    FO_MOVE : S := 'Move';
    FO_DELETE: S := 'Delete';
    FO_RENAME: S := 'Rename';
  end;
  p := lpFileOp.pFrom;
  S1 := WideCharToString(p);
  S2 := WideCharToString(lpFileOp.pTo);
  SaveLog('SHFileOperationW(' + S + ')', 'Begin');
  while(S1 <> '') do
  begin
    SaveLog(S1, S2);
    p := p + Length(S1)-1;
    S1 := WideCharToString(p);
  end;
  SaveLog('SHFileOperationW(' + S + ')', 'End');
  Result := SHFileOperationWCallNext(lpFileOp);
end;
```

图3.3　记录SHFileOperation操作的代码

代码很简单很清晰，不需要多加解释。为了验证文件的复制操作和移动操作用到这个 API，设计的实验如下：

（1）　同时复制 4 个文件

（2）　剪切 1 个文件

（3）　删除 4 个文件

（4）　重命名 1 个文件

结果是：

（1）　剪切的日志

2008-6-12 15:20:55 312= #7 [explorer.exe:276]

<SHFileOperationW(Move)> Begin

2008-6-12 15:20:55 328= #8 [explorer.exe:276]

<C:\Documents and Settings\WangQian\桌面\Hook\2008-6-12-14-29-31-22.log>

C:\Documents and Settings\WangQian\桌面

2008-6-12 15:20:55 328aa= #10 [explorer.exe:276]

<SHFileOperationW(Move)> End

火龙果•整理
uml.org.cn

（2） 复制的日志

[File Operation Logs]

2008-6-12 15:20:37 0= #1 [explorer.exe:276]

<SHFileOperationW(Copy)> Begin

2008-6-12 15:20:37 46= #2 [explorer.exe:276]

<C:\Documents and Settings\WangQian\桌面\Hook\2008-6-12-14-37-56-37.log>

C:\Documents and Settings\WangQian\桌面

2008-6-12 15:20:37 46a= #3 [explorer.exe:276]

<C:\Documents and Settings\WangQian\桌面\Hook\2008-6-12-15-1-41-203.log>

C:\Documents and Settings\WangQian\桌面

2008-6-12 15:20:37 46aa= #4 [explorer.exe:276]

<C:\Documents and Settings\WangQian\桌面\Hook\2008-6-12-15-1-41-203.log.bak>

C:\Documents and Settings\WangQian\桌面

2008-6-12 15:20:37 46aaa= #5 [explorer.exe:276]

<C:\Documents and Settings\WangQian\桌面\Hook\2008-5-23-10-31-5-244.log>

C:\Documents and Settings\WangQian\桌面

2008-6-12 15:20:37 46aaaa= #6 [explorer.exe:276]

<SHFileOperationW(Copy)> End

（3） 删除的日志

2008-6-12 15:21:7 765= #11 [explorer.exe:276]

<SHFileOperationW(Delete)> Begin

2008-6-12 15:21:7 765a= #12 [explorer.exe:276]

<C:\Documents and Settings\WangQian\桌面\Hook\2008-6-12-14-37-56-37.log>

2008-6-12 15:21:7 765aa= #13 [explorer.exe:276]

<C:\Documents and Settings\WangQian\桌面\Hook\2008-6-12-15-1-41-203.log>

2008-6-12 15:21:7 765aaa= #14 [explorer.exe:276]

<C:\Documents and Settings\WangQian\桌面\Hook\2008-6-12-15-1-41-203.log.bak>

2008-6-12 15:21:7 765aaaa= #15 [explorer.exe:276]

<C:\Documents and Settings\WangQian\桌面\Hook\2008-5-23-10-31-5-244.log>

2008-6-12 15:21:7 781= #16 [explorer.exe:276]

<SHFileOperationW(Delete)> End

火龙果•整理
uml.org.cn

（4） 重命名的日志

```
2008-6-12 15:21:24 843= #17 [explorer.exe:276]
<SHFileOperationW(Rename)> Begin
2008-6-12 15:21:24 843a= #18 [explorer.exe:276]
<C:\Documents and Settings\WangQian\桌面\2007-8-27-17-12-29-224.log>
C:\Documents and Settings\WangQian\桌面\1.log
2008-6-12 15:21:24 859aa= #21 [explorer.exe:276]
<SHFileOperationW(Rename)> End
```

由日志中可以看出，只要拦截 SHFileOperation，对其进行操作就可以控制复制的过程。实现复制的透明加解密，至于用什么方法实现将在下文中提出。日志中非标出的内容跟这里的分析没有关系，这个将在下文中的日志系统中提到。

其次，对文件读写透明加解密进行研究。由于本论文中已经对 AutoCAD 的读写透明加解密技术研究的很完善，所以以 AutoCAD 为例，说明分析的结果。

分析 AutoCAD 首先要找到在 kernel32.dll 中使用的读写文件的 API 函数。这个是通过 AtuoDebug 软件把所有的 kernel32.dll 中跟文件读写的 API 全部钩上然后跟踪观察。由于这里的日志信息由好几万条，不能在这里列举，但经过详细的分析，发现 AutoCAD 读写文件只跟几个 API 函数相关，就是在上文已经详细介绍过的几个 kernel32.dll 的 API 函数。得到这个结果以后，就可以用自己的程序钩住这几个函数，获得详细的 AutoCAD 软件的运行过程。由于分析软件的行为也很繁琐，这里只以打开并写入一个以 dwg 作为后缀的文件为例分析，最后给出结论。

最后记录的日志有 1000 多行，这些日志都是描述了 AutoCAD 打开一个以 dwg 为后缀的文件，并修改保存后的行为。分析日志可知，在 AutoCAD 打开一个进程的时候，总会使用 CreateFileW 方法，并且使用 ReadFile 方法从之前打开的文件里读取数据到内存中。

在修改保存后，AutoCAD 的行为比较复杂，设计到不同位置的一些临时文件，但是这些临时文件都是作为备份的作用，跟本文要研究的内容关系不大。其中要注意两种类型的文件。一种是 dwg 文件本身，另一种是可能存在的 bak 文件。Bak 文件保存 dwg 文件修改前的内容。最后系统总会通过 WriteFile 把内存中的数据

火龙果•整理
uml.org.cn

写回到 dwg 文件中。而之间的一系列复杂的 MoveFile 和 CopyFile 的操作，由于操作的单位都是数据文件，并且都是磁盘落地的数据文件，所以不需要考虑。因此，AutoCAD 研究相对来讲比较简单。实现起来也比较容易。当不研究临时文件时，日志很清晰可以看出打开文件和读写文件的过程。

3.3 系统的设计模型

3.3.1 系统的设计模型

基于以上分析的基础上提出了本论文的理论模型。本论文主要设计和实现的系统包括两个部分，一个是作为一个进程单独存在在用户空间中的一个部分；另外一个是被第一个部分所注入到系统其他应用程序进程的部分，如图 3.4 所示：

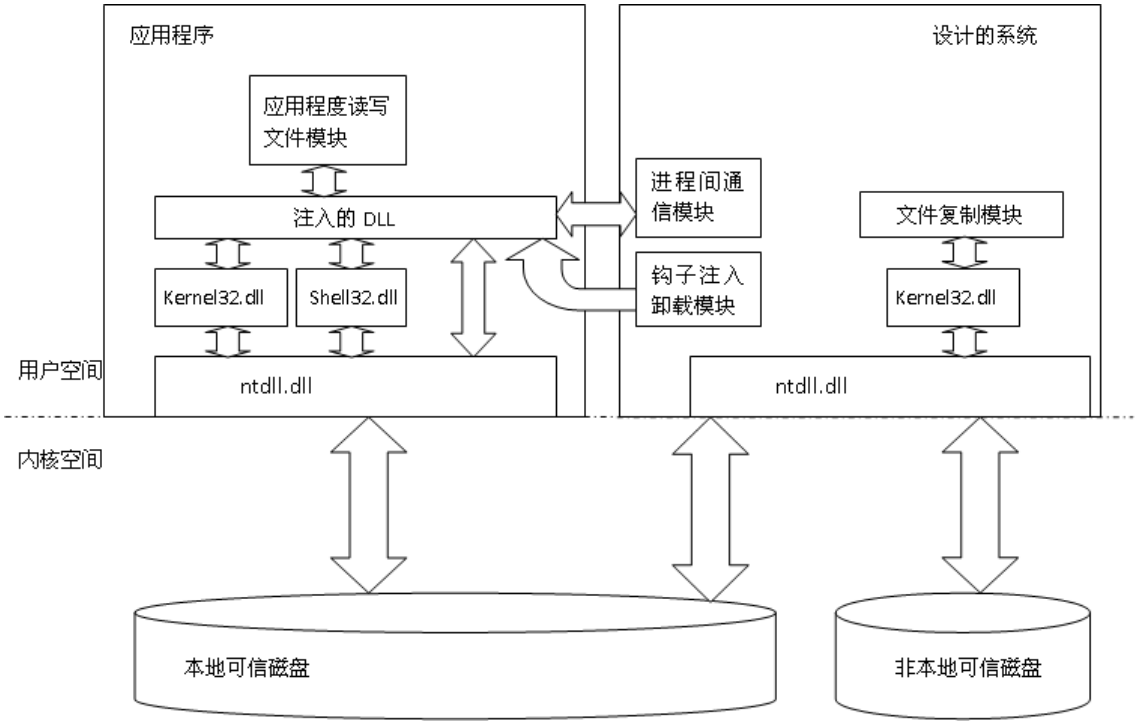

图3.4　系统的设计模型图

系统的两个部分分别是实现了论文研究的两个主要方面的内容：

火龙果•整理
uml.org.cn

被注入的 DLL 模块一个功能是实现读写文件的透明加解密，另外一个功能是拦截 shell API 的操作，跟设计的系统进程进行通信，供所设计的系统进程访问应用程序进程的内存空间。

而设计的系统主要是负责注入和取消钩子，并且负责对拦截到 Shell API 的复制和移动的操作实现复制透明加解密。下面就系统每一部分的设计思想进行简要的介绍。其中的前两个模块都有现有的研究结果可以直接应用，而后两个模块是本论文研究的重点。

这里要重点解释一下，之所以把文件读取透明加解密的过程放在被注入的进程中执行，是因为文件的内容在被注入的进程中是需要的，并且加解密的这个过程需要重复很多次（因此文件是被分成小块读取很多次的），如果通过进程间通信的方式放在设计的系统中执行，那么交互的成本太高。而另一方面，把复制的透明加解密过程放在设计的系统中进行，是因为这个过程只需要知道包括原路径和目的路径在内的很少的信息。可以通过进程间通信共享，并且可以把复制的过程变成异步的文件读写的过程，这样还能大大的提高效率。

3.3.2 钩子注入卸载模块

这个模块负责注入的 DLL 的注入和卸载。当本系统启动的时候把要注入的 DLL 注入到操作系统运行的所有的进程中。并且当有新的进程开始运行的时候，这个 DLL 也会被注入进去。

当设计的系统进程被结束的时候，自动卸载注入的 DLL。

3.3.3 进程间通信模块

进程间通信主要是用于被注入的 DLL 跟设计的系统之间进行信息交互。这种交互一方面可以用作被注入的 DLL 的一部分策略的获取，另外一方面在复制透明加解密的过程中，可以用作把复制所需要的关键信息传给设计的系统，把复制的过程在设计的系统中实现。

3.3.4 文件复制模块

文件复制模块所做的工作不是原来意义上的复制，而是先把文件读进内存，在文件被写向目的地址的过程中加密。而解密的过程正好相反，就不再赘述。这个模块需要解决很多的问题，主要是效率的问题，如果这种读写连同加密的方式比 Windows 直接复制或者剪切的方式效率低太多的话会严重影响用户的使用。这个模块也将分成几个子模块主要解决效率的问题。

火龙果•整理
uml.org.cn

3.3.5 注入的DLL模块

这个被运行在被注入的进程中的模块是最难实现的一个模块，如果稍微有一点的错误就将造成原来的应用程序不能正常使用甚至导致系统崩溃。在这个模块中，本文将实现一个简单的控制系统，在不影响原来的应用程序正常运行的基础上达到文件透明加解密的效果。这个模块的实现也是由一个控制流程加上若干子模块实现的。

火龙果•整理
uml.org.cn

第4章 系统的实现

4.1 设计系统实现和运行的环境

本论文的研究的内容是在 Windows XP 的系统下实现的。所用的开发工具是 delphi7。在实验的过程中用到 madCollection 的开发包，还使用了一个实现 AES 算法的函数接口。

因此如果要应用本系统需要的系统要求如表 4.1 所示：

表4.1 应用本系统的计算机要求

项目	要求
操作系统	Windows XP 任何版本
处理器	Pentium® III 或更高版本 500 Mhz（最低）
RAM	256 MB（最低）

4.2 DLL文件的注入卸载模块

这个模块主要是使用 madCollection 提供的函数实现的。注入的代码为：

```
InjectLibrary(ALL_SESSIONS Or SYSTEM_PROCESSES,
'inject.dll')
```

其中，InjectLibrary 这个函数在之前第 2 章中已经有过详细的描述。这段代码把第 2 个参数的文件“inject.dll”注入到了系统所有的进程中包括系统进程和系统服务。

卸载的代码为：

```
UnInjectLibrary(ALL_SESSIONS Or SYSTEM_PROCESSES,
'inject.dll')
```

这个函数的作用是把“inject.dll”从系统中所有的进程包括系统进程和系统服务中卸载掉。

火龙果•整理
uml.org.cn

4.3 进程间通信模块

这个模块也是用 madCollection 开发包实现的。首先，在被注入的 DLL 和设计的系统中都定义好一个结构 Rec，这个结构包括操作的类型（op）、源地址的长度（SucLen）、目标地址的长度（DesLen）。同时，还需要有共同的 IPC 名称。

当设计的系统把 DLL 注入到所有系统的进程的时候，同时调用 CreateIpcQueue 创建一个队列，然后自己等待返回；被注入的 DLL 把钩子挂到相应的函数上。被刮钩子的函数在原应用程序中被调用的时候，注入的 DLL 获得控制权。被注入的 DLL 首先把共享的 Rec 结构中的三个量填好，然后分配根据 Rec 中的三个量分配好一段内存空间，把 Rec。把源地址和目标地址依次填充到这段内存里，参加图 4.1。

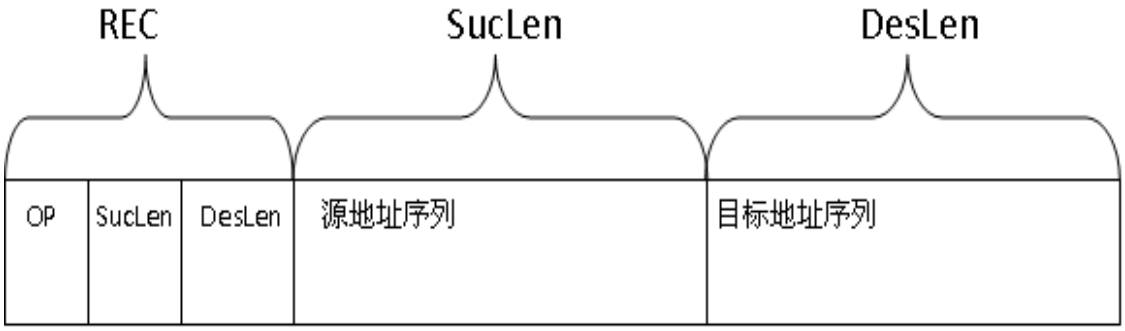

图4.1　进程间共享的内存的组织结构

然后 DLL 通过 SendIpcMessage 函数把这段内存共享给设计的系统，自己被阻塞。设计的系统通过 CreateIpcQueue 是制定的 HookBack 的函数被重新激活。获得图 4.1 所示的内存内容和其他的信息，然后就可以通过内容访问完成其他的工作。最后返回给 DLL 一个信息，结束 DLL 所在进程的阻塞状态，让其继续工作。

整个的过程比较复杂，因此用下面的表 4.2 做出一个简单的描述。

火龙果•整理
uml.org.cn

表4.2　进程间通信的整个过程

步骤	设计的系统	被注入的进程
1	CreateIpcQueue，并确定回调函数的名称，等待返回	无
2	注入 DLL	DLL 被注入，并把钩子挂到相应的 API 函数上
3	等待返回	被挂钩子的 API 被应用程序调用
4	等待返回	构造返回的内存内容
5	等待返回	调用 SendIpcMessage，把相应的内存和相关信息返回
6	回调函数捕捉到返回	阻塞等待返回
7	根据返回的内容进行相应的操作，并返回操作结果	阻塞接触，等待相关 API 再次被调用
8	操作结束，等待队列被填充	等待相关 API 再次被调用
9	如果队列被填充从 6 开始	如果 API 再次被调用，从 3 开始

4.4 文件复制模块

4.4.1 数据结构的介绍

这个模块中存在很不不同的数据结构，下面把存在的重要的数据结构列举如表 4.3 所示：

表4.3　文件复制模块中的数据结构介绍

名称	说明
TBaseItem	源文件的集合，包括文件名和是否是目录
TBaseList	TBaseItem 的列表
TWorkThreadList	工作线程的列表
TCopyThread	工作的线程类
TCopier	复制操作的类

火龙果•整理
uml.org.cn

其中，每个 TCopyThread 的实例是一个线程。多个 TCopyThread 是多线程的操作，这个在下面的小节中详细介绍；每个 TCopier 的实例是一个复制的具体操作，每个复制都是通过异步读写文件来实现的，这部分也会在下面的小节中具体介绍。

4.4.2 多线程处理多个复制的操作的实现

这里的多线程的实现利用了 delphi 中多线程的功能，当设计的系统接收到了被注入的 DLL 发回的信息后，对信息进行解析，分析出源地址和是否为目录，把这两个信息封装到 TBaseItem 的实例中然后放到 TBaseList 中。TWorkThreadList 维护了一个线程列表，目的地址一样的复制操作可以用同一个线程来实现。对目的相同的复制操作，分配一个线程 TCopyThread，用于这些文件的复制。最后把 TBaseList 类中装的原地址及操作类型以及目的地址和复制的特殊信息全部传递给 TCopyThread 类的实例中，就可以用于每个线程的复制操作。流程如图 4.2 所示。

4.4.3 异步模式读写文件加解密的实现

直接同异步模式配合事件可以有效的进行一个文件的读写。首先构造三个事件： READ_ENDING_EVENT，WRITE_ENDING_EVENT，WORK_EVENT。并且把 READ_ENDING_EVENT 赋值给 ReadFile 的最后一个变量的事件，WRITE_ENDING_FILE 赋值给 WriteFile 最后一个变量的事件。

在异步操作的过程中，成块读取、写入文件的内容，每一块的大小是 64K，并生成一个缓存空间，缓存的大小是可以配置的，在本论文中配置成 8 个文件读写块的大小及 512K。

通过标志控制读写文件都是顺序的，用一个变量 UserBuffer 来控制读写行为，当 UserBuffer 小于缓存大小时可以继续读，当 UserBuffer 大于 0 时可以写文件。

最后通过 WaitForSingleObject 通过对事件的控制来实现对读写流程的控制。每当一次读操作完成时，事件发出消息被 WaitForSingleObject 捕获，更新跟读相关的信息，当一次写操作完成时，事件发出消息被 WaitForSingleObject 捕获，更新跟写相关的信息。读写信息更新完毕后进行下一次的读或者写。简单的控制流程如图 4.3 所示。

火龙果•整理
uml.org.cn

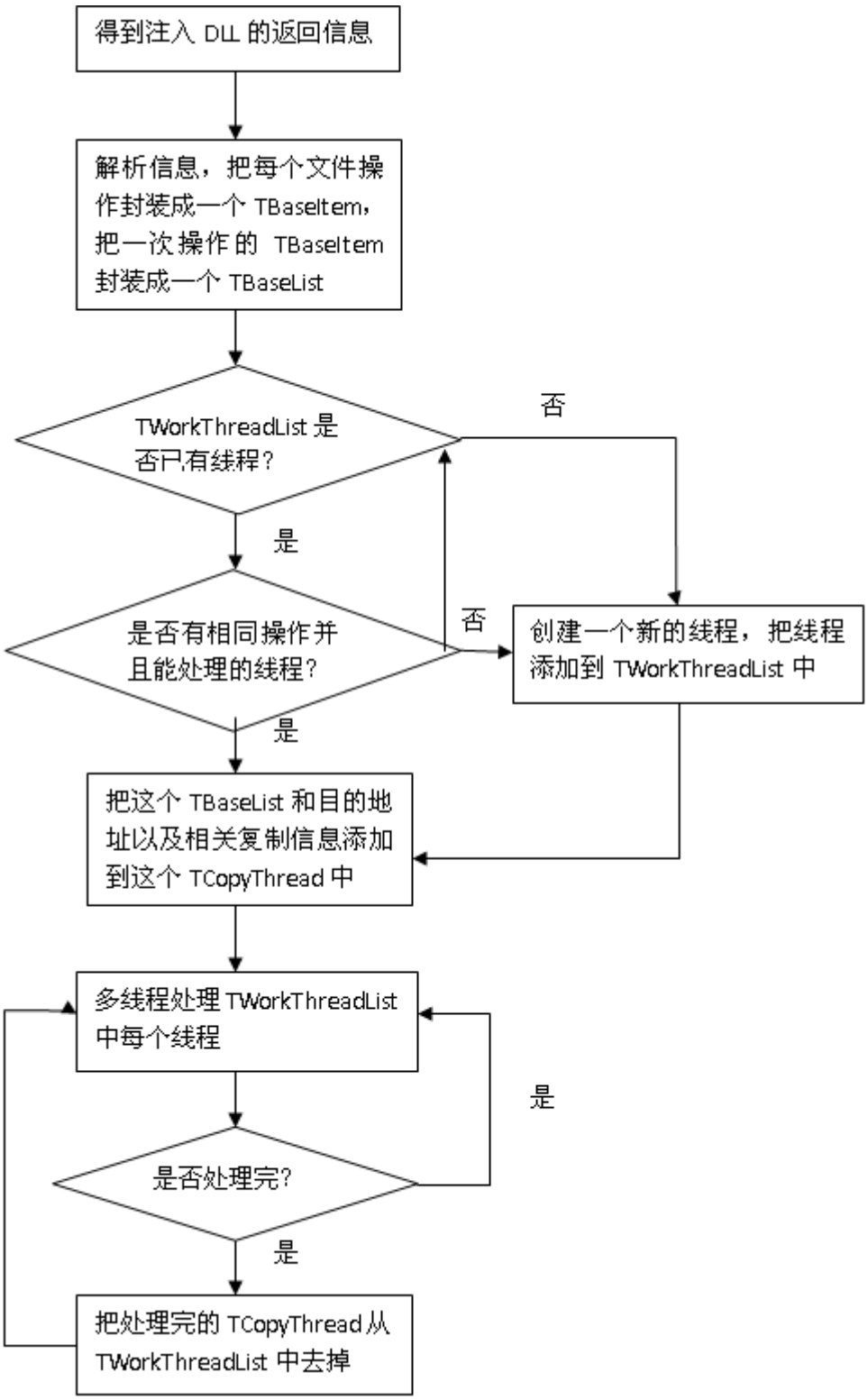

图4.2　多线程处理多个复制操作的流程图

火龙果•整理
uml.org.cn

但是如果直接加解密的话会出现一个问题，那就是没法判断文件是明文还是密文。所以必须给文件一个标志区分明文和密文。本论文中采取加文件头的办法，给密文数据加了一个 1024 字节（1K）的文件头，这 1K 的文件头目前只用了 8 个字节用于区分明文数据和密文数据，将来可以用来扩展，记录文件特定的一些信息。

文件头的加入虽然可以区分明文和密文的数据，但是也引出了新的问题。就是复制加解密以后的文件跟之前的文件大小是不同的。因此，在论文中加入了三个全局变量，一个是 CONTROL_READ，一个是 CONTROL_WRITE，还有是 CONTROL_NOTHING，这三个变量是在 TWorkThreadList 处理 TBaseList 时候复制的。CONTROL_READ 是控制读，也就是读解密；CONTROL_WRITE 是控制写，也就是写加密。另外一个变量是既不加密也不解密。在本论文中虽然没有实际的作用，但考虑到以后扩展为对文件类型区分加解密的时候，这个变量确实很有作用的。

在 CONTROL_READ 的过程中，要忽略掉已经存在的文件头，对每次读的位置都要加上一个文件头的长度。例如：如果要读第二个 64K 的快的数据，那么读的 ReadPos 不是 65536，而是 ReadPos := 65536 + headerLength。同理，在 CONTROL_WRITE 的过程中，要在写入的文件中写入一个头，文件写入的位置也要加上一个头的长度。同时在读写的过程中还有很多变量的在复制的时候需要考虑头的存在，这也是本实验实现起来比较有困难的地方。

火龙果•整理
uml.org.cn

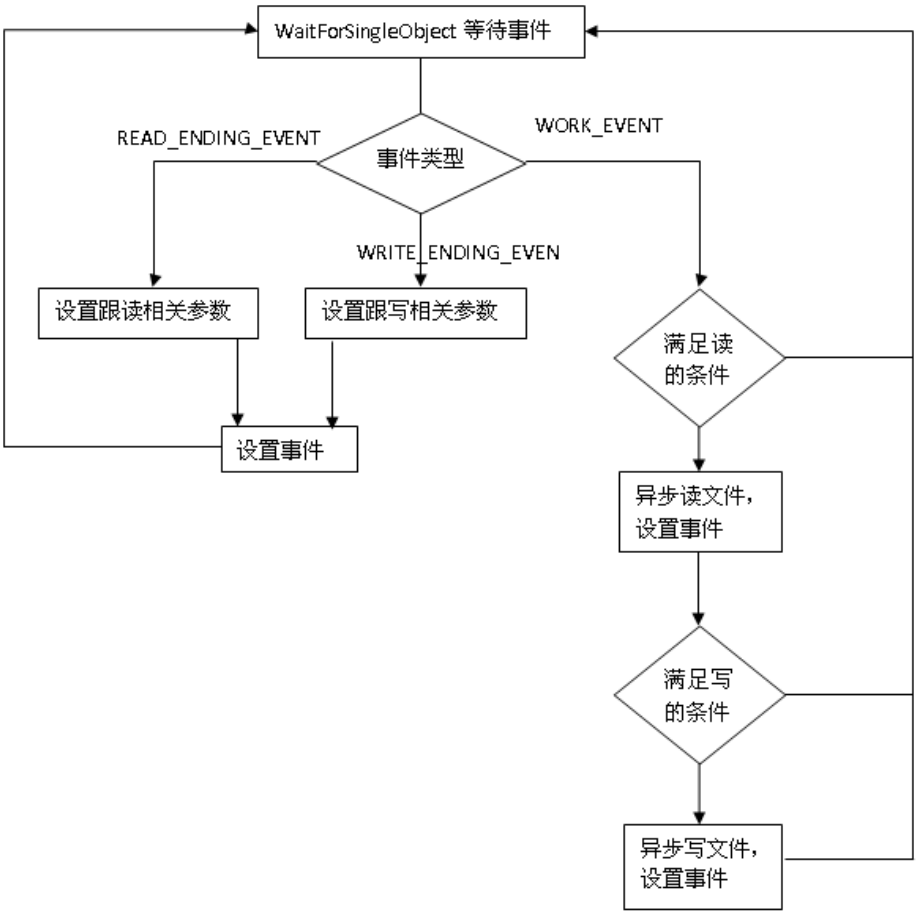

图4.3　异步读写文件的简单示意图

4.5 注入的DLL模块

4.5.1 模块的内部结构

被注入的 DLL 看起来很小，实际也是由不同的子模块组成的。因为这些子模块是在另一个未知或知之甚少的进程中运行的，所以对其其可靠性要求非常高，因此设计和实现起来难度也非常大。这个被注入的 DLL 也是整个实验非常重要的一个部分，它负责对进程 API 的注入，获得进程暂时的控制权，是其他一切操作的前提。模块的结构图如图 4.4 所示。

火龙果•整理
uml.org.cn

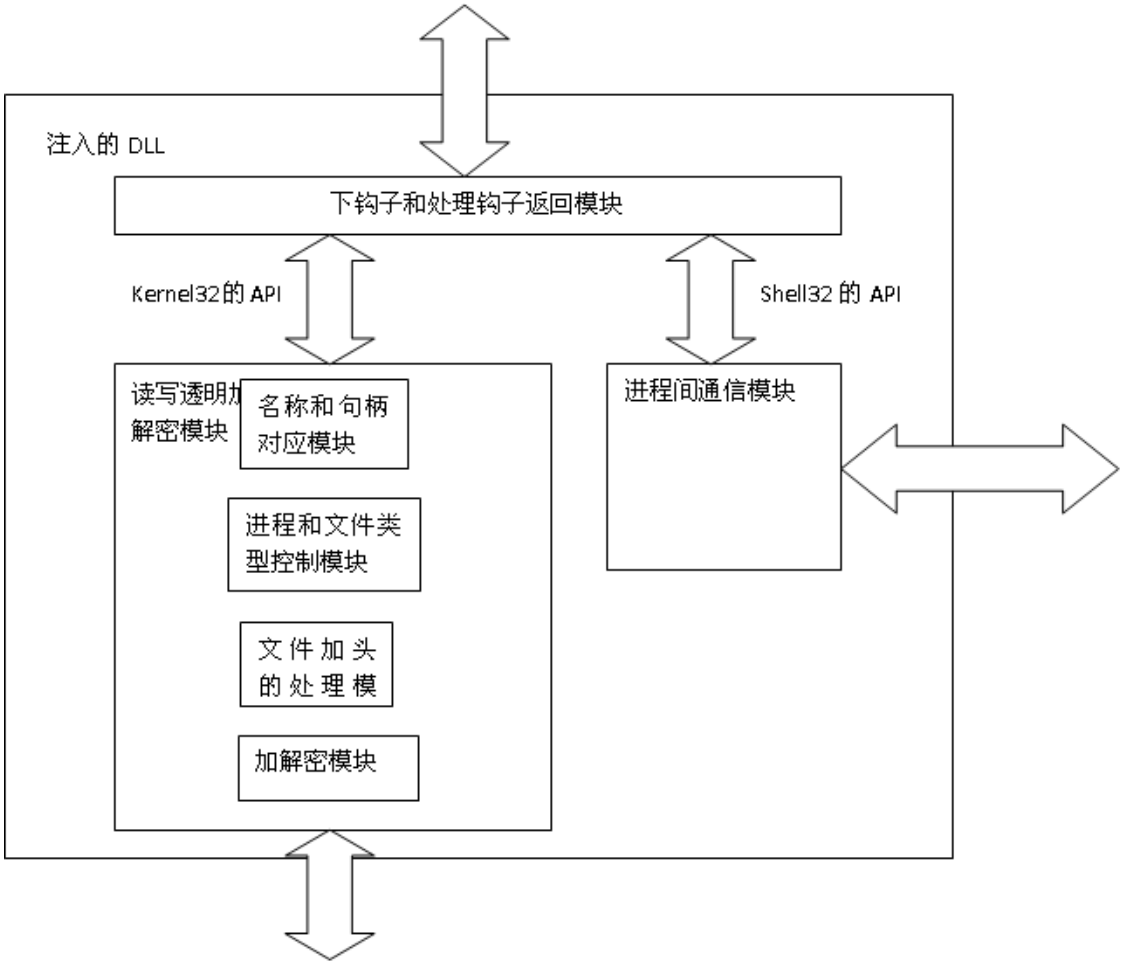

图4.4　注入DLL模块的结构图

下面简单来介绍一下图中各个模块的作用和配合工作的流程。一些复杂的模块将会在下文的章节中详细介绍。

拦截和处理拦截模块是用来下钩子，并且处理钩子的返回。这个模块主要是通过 MadCollection 中的 HooKAPI 函数实现的，消息的返回是 MadCollection 的内部机制，最后的结果就是相应的拦截 API 函数的回调函数会的进程的控制权。例如：如果被拦截的 ReadFile 函数在应用程序中被调用，那么在 ReadFile 的回调函数 ReadFileCallBack 将会被执行，用户可以在这个函数中执行自己的代码，最后调用 ReadFileCallNext 把进程的控制权交还给应用程序。

进程间通信的模块在之前 4.4 的章节中已经描述过，这里就不再赘述。进程间的通信是需要两个进程配合完成的。

火龙果•整理
uml.org.cn

Kernel32.dll 中的很多的 API 是对句柄进行操作的，而 Windows 中缺乏提供给用户的文件名和句柄进行对应的操作函数。而在本实验中不仅需要知道一个文件的句柄，还需要文件的名称甚至是路径，而通过 Windows 操作系统提供的 API 函数是没法做到的，所以本实验自己设计了一个跟实验相关的文件名和句柄的对应表，解决这个问题，具体的实现方法在后面详细介绍。

文件读取透明加解密只是对部分进程打开的部分特定后缀的文件起作用。之前的章节中也提到过如果全盘加解密的话会遇到很严重的效率问题，并且也不是十分的必要。实验中实现的是只有满足特定的条件的才会进行加解密操作，相信的部分在后面介绍。

同上节中提到的一样，区分明文密文文件是通过加了一个文件头来实现的。加头后主要的问题就是如果还是按照应用程序原来的读写过程来操作的话会出现不对应的问题。应此需要在应用程序定位文件指针的时候给文件指针进行一定的偏移，使对应用程序来说这个头和因为头造成的文件的偏移是透明的。详细的设计细节在后文中具体描述。

加解密的模块设计起来比较复杂。因为每个 API 函数中读取的内容的大小都是不定的，所以分组加密的方法不能使用，只能使用流加密的方法。而传统的流加密的算法也是有问题的，传统的流加密的算法是逐位生成密钥，跟原文内容进行抑或加解密，这样需要一块文件内容在加密和解密的时候是同一块，在本论文研究的情况下这显然是不现实的。因此，本论文设计的是生成一个密钥流，根据一个字节在文件中的位置跟对应的密钥流的位置对应加密。详细的设计细节也在后文中体现。

在本小节的最后，对整个注入 DLL 模块的控制流程进行描述，也能更清楚的说明上述的几个子模块之间是如何相互配合工作的。

首先，通过进程和后缀控制模块，保证透明加解密的过程是从 CreateFileW 的回调函数开始的。在这个回调函数中，首先判断通过进程和后缀控制模块判断要打开或者创建的文件是不是需要保护的文件。如果不是则直接返回给正常的应用程序，如果是就通过文件名和句柄对应模块把这个文件的相关内容存储到这个模块维护的一个链表中，然后如果这个文件还是明文文件，则通过加解密模块和文件加头处理模块对打开的文件进行加密和加头的操作。这个操作时在不改变文件位置和属性，只改变文件的内容的基础上实现的。

然后，当读和写 API 函数被调用时，首先通过句柄和文件名对应模块看这个句柄代表的文件是不是需要加解密，如果需要则通过加解密模块进行相应的加解

火龙果•整理
uml.org.cn

密。当设置文件指针的操作被调用时，也是首先通过句柄和文件名对应模块判断被设置指针的文件是不是需要保护的文件，是的话对文件指针进行相应的操作。文件读取透明加解密的过程如图 4.5 所示。

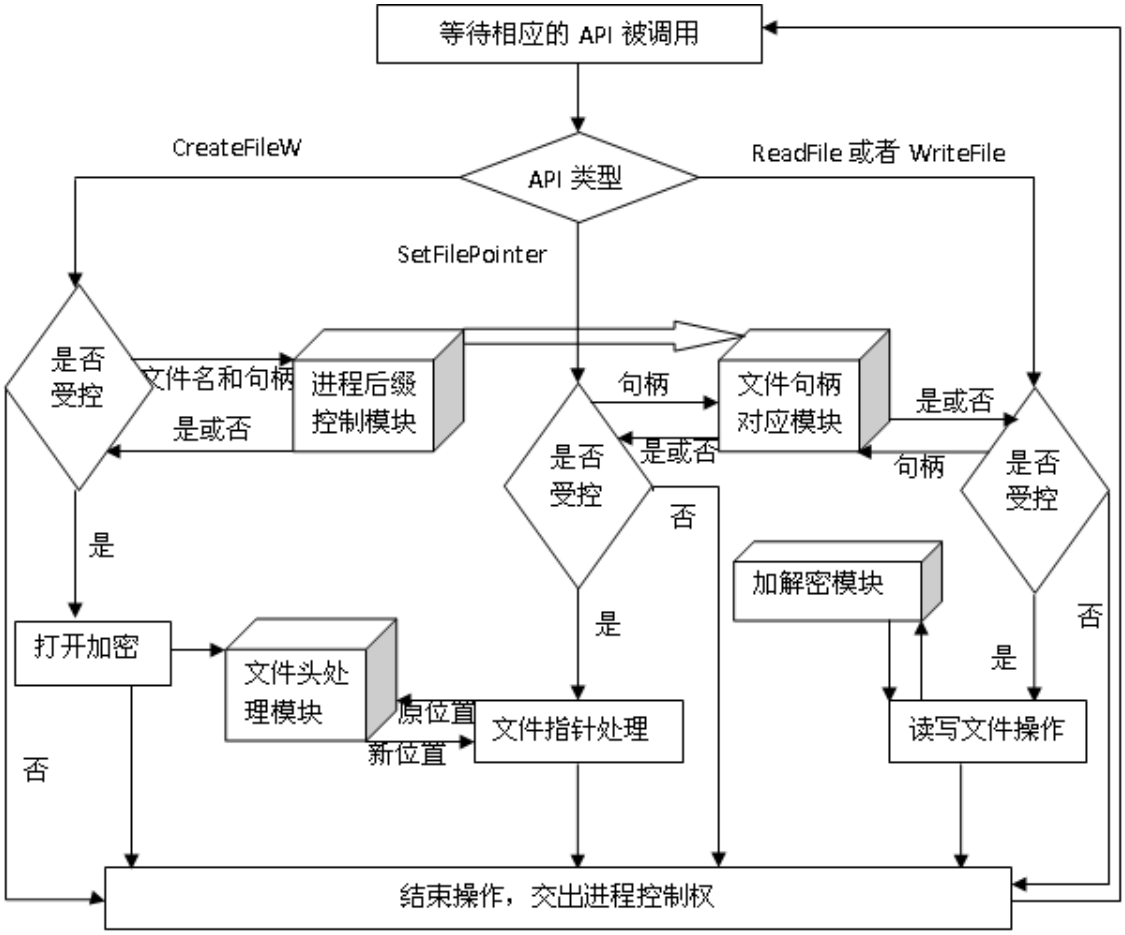

图4.5　文件读取透明加解密过程

4.5.2 文件名和句柄的对应

这个模块实在 CreateFileW 的回调函数 CreateFileWCallBack 中实现的，主要是对于每个创建或者打开的文件，将它的句柄和文件名对应，并保存这个文件是明文还是密文的信息。这三个变量是一个结构体，而这些结构体在一起组成了一个链表。这个链表是下个模块所必需的。对于需要保护的文件类型，在每次打开或者创建文件的时候判断这个文件的句柄是不是在这个链表里，如果不在，则生

火龙果•整理
uml.org.cn

成一个之前描述的结构体，然后把这个结构体加到链表里。这个结构体的定义如下：

```
type
   pnode = ^tnoderec;
   tnoderec = record
   data : THandle;
   name : string;
   Encrpy : bool;
   next : pnode;
end;
```

如果句柄在链表里存在，则用新的结构体的内容替换之前存在的结构体。本来在 CloseHandle 调用的时候应该清空这个结构体里被 Close 的句柄，但是因为 CloseHandle 被调用的太频繁，如果下钩子的话会导致系统崩溃，因此，现在使用这样一种折中的方法，并且经过实验验证基本能够达到原来的目标。

4.5.3 进程和后缀的控制

进程和后缀的控制是基于上述的文件名和句柄对应的链表实现的。ReadFile，WriteFile 和 SetFilePointer 这三个 API 都是只有进程句柄而没有文件名称的。而本实验中判断一个文件是否需要加密是需要根据文件后缀判断的。因为上述的链表存的都是需要控制的文件，所以，只要在这里访问链表，文件句柄能在链表上找到，那么就证明这个文件的这个操作需要被控制。

当然，上面也提到过，因为没有对 CloseHandle 这个 API 进行拦截，导致的可能有些文件的句柄之前是作为受控文件打开过，被存到了链表上，但后来被 Close 了，然后又被当成非受控文件的句柄生成，那么可能会出现问题。但是在不拦截 CloseHandle 的基础上，没法确定链表上的文件句柄是否一定有效。这是需要在后面研究的问题，但在这个实验中这个问题对实验结果基本没有影响。

4.5.4 文件加头的处理

文件加头的处理包括两个方面的内容。

火龙果•整理
uml.org.cn

一方面是处理 SetFilePointer 的因为加头而导致的文件指针需要偏移的问题。这个问题解决的办法就是在 SetFilePointerCallBack 回调函数中对移动的的距离再加上一个长度为文件头大小的偏移。

另一方面，是要处理在 CreateFileWCallBack 回调函数中对打开的明文文件的加密问题。如果没有多余的文件头，那么可以读出一段加密写回一段，但是因为文件头的存在，每次读出的一段加密后写会文件就会覆盖一部分（文件头长度的）明文，因此会导致错误。如 4.6 所示。其中白色是原始数据，蓝色是密文数据，红色阴影就是由于第一块数据被加密后写回到原来文件中导致原文件第二块之前那一部分被密文覆盖了。

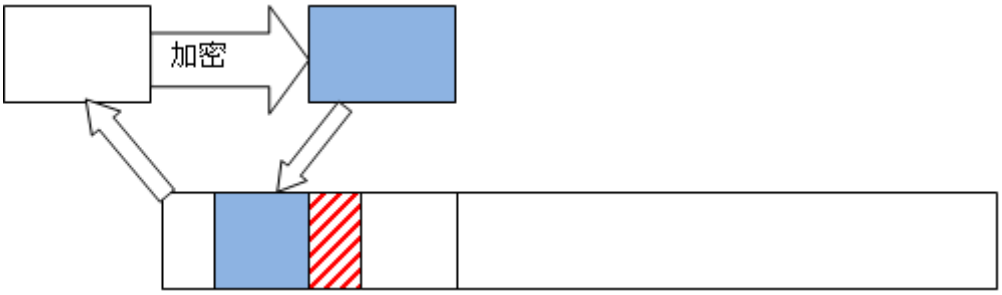

图4.6　由于文件头存在而造成文件直接加密的问题

这个问题有两个解决办法，一个是从文件尾部开始读出、加密、写回，一直写到文件头。另一种办法是运用临时文件，先把文件的内容整体写到一个临时文件里，然后再把临时文件的内容写回到原文件里。本实验就是采用了第二种方法，在 C 盘根目录下创建一个临时文件用于中转，在用完以后即时删除，即使文件被暴露存的也是密文的内容而不会泄密。文件读写使用的是 ntdll.dll 中的 NTReadFile 和 NTWriteFile 函数，为了避免使用 kernel32.dll 中的函数跟钩子产生冲突导致不可预料的风险。

4.5.5 加密算法的选择

上文提到过了在这个大的模块中，加密算法所遇到的问题。也提到过了本文的解决办法，就是生成一个密钥流，根据一个字节在文件中的位置跟对应密钥流位置的字节进行抑或加密。但在实现这个算法中还是遇到了一个可以研究的问题。

火龙果•整理
uml.org.cn

这个生成的密钥流可以有两种选择：第一，密钥流是临时生成的，在每次读或写文件的 API 的回调函数中生成。第二，密钥流在文件打开时生成，是保存在系统内存中的。

两种方法各有利弊，第一种方法有利的是，这是一种真正的流加密的方法，但缺点是要生成太多次，对系统性能会有影响，时间复杂度太高；第二种方法有利的只需要生成一次，可以根据文件大小生成相应长度的密钥，然后保存到内存中，缺点是大文件的话空间消耗太大。这个问题本质上是一个时间和空间取舍的问题。考虑到本模块是一个注入的 DLL 模块，如此消耗时间和空间都会令系统十分的不安全。太消耗时间的话，就会导致透明加解密不是那么的透明，用户会明显感觉到它的存在；太耗空间的话可能导致应用程序内存不足，引起一系列不可预知的错误。

最后本实验选择了一个折中的办法，就是在文件打开的时候用 RC4 算法生成固定长度（4K）的密钥流，考虑到这个密钥流的安全性问题，又对这个 4K 的密钥流进行了 AES 的加密。把最后加密过的 4K 的内容作为与原文抑或的密钥。

这个模块中需要解决的另外一个问题就是如何确定当前的字节在文件中的位置。因为在 kernel32.dll 上层所有对文件指针的定位的函数到了 kernel32.dll 层都是通过 SetFilePointer 来实现的，而 SetFilePointer 又被实验中的钩子拦截进行了偏移处理，所以，实验中果断的放弃了 kernel32.dll 层的 API 函数，而直接使用 ntdll.dll 层中的 NTQueryInformationFile，这个函数在前面的章节中已经详细的解释过，这里就不再介绍。

在本模块最后，介绍一下加解密的过程，以加密为例。当 ReadFileCallBack 这个回调函数得到进程控制权后，首先取得当前指针的位置作为这次的起始位置，然后读取文件，对读到内存中的内容根据起始位置计算每个字节的位置，然后跟这个位置对 4K 取余的位置的密钥元素做抑或运算。解密的过程基本类似，这里要强调一点的是，一定要先取得当前指针的位置再调用 ReadFileCallNext，因为 ReadFileCallNext 调用以后，文件指针就已经移到读完的位置而不是停留在起始的位置了。

4.6 日志系统

由于注入的 DLL 这个模块是在另外一个进程下运行的，所以没办法进行调试操作，只能通过写日志的方法。又因为普通读写文件的 ReadFile，WriteFile 都被

火龙果•整理
uml.org.cn

本实验设计的系统下钩子拦截了，因此不能使用。本实验中使用的是 Windows 提供的写 INI 文件的 API，把实验中需要的信息写到 INI 文件中，最后在钩子卸载以后再把 INI 文件复制成普通文件。写文件的操作被封装成了一个 SaveLogFile 的函数以供在系统里使用。由于这个部分比较简单和容易理解，所以就不再详述，这里提到这点的原因就是要介绍本论文自己的调试工具。

火龙果•整理
uml.org.cn

第5章 结论

5.1 取得的成果

本论文的研究历时快一年，在本人没有任何研究经历的基础上，在业界和学术界对研究的内容资料很少的情况下，在研究了几十万行各种工具的日志的基础上，本文的研究取得了很好的效果。

成果体现在以下四个方面：

（1） 数据文件保护系统的设计思路。本文设计的系统的保护是分等级的，对只存在硬盘的数据文件和需要复制或者拷贝出去的数据文件实现了不同级别的保护。这一方面能提高系统正常运行时读写文件的效率，不会因为太强的加密算法导致系统效率降低；另一方面也提高了拷贝或者剪切出去的数据安全性。不同的保护级别也可以供以后扩展，对磁盘的不同的卷进行不同级别的加密保护，或者对不同的类型的文件进行不同级别的保护。

（2） 用户层通过 API 拦截实现的透明加解密系统的理论研究。本论文从构造这样一个系统所需要的理论知识和工具（第二章内容），系统的分析与设计（第三章内容）上给出了详细的说明和系统的资料，可以供以后研究这个技术的人参考，填补了对用户层 API 拦截实现的透明加解密系统缺乏理论详细的指导的现状。

（3） 实现一个可用的系统。这也是这篇论文的最重要的部分。本文实现的系统已经可以正常的运行，对除了部分 Office 系列软件的读取透明加解密之外的其他设计功能已经全部实现。本文的系统其中一个重要的模块数据文件复制透明加解密已经相当成熟，不仅实现要求的功能，还在经过加密的基础上效率比 Windows 系统没有明显下降甚至在某些程度上还会有所提升，也完全实现了自己的复制界面。这部分内容已经被用在另外一个系统中作为核心模块，参加了清华大学挑战杯获得三等奖。系统的另外一个重要的部分数据文件读写透明加解密模块已经实现了对 AutoCAD 软件的透明加解密，实现了现在市场上为数不多的，例如武汉天喻的系统，在用户层的透明加解密的软件的核心功能。经过一定的版本测试和系统测试后，可以做成一个至少对所有的 AutoCAD 版本软件都适用的系统。

火龙果•整理
uml.org.cn

（4） 研究过程中的探索和收获。在研究过程中，深入到了用户层最底层的编程操作，对 Windows NT 系统的文件系统、内存管理等方面有了很深的了解，粗浅的研究了密码学的内容，并对其中几种加密算法进行深入的研究。在研究过程中也不断有新的想法和理念，为以后的研究工作打下了很好的基础。

5.2 未来的展望

虽然经过很长时间的研究，取得了很大的成果，但是还是受到时间的限制，有些工作没有完成，在研究中对这个系统产生了很多的新的想法也没有机会实现，但这些都为以后的研究提供了参考方向，主要有以下几个方面：

（1） 现在实现的系统还不能对 Office 系列软件进行读写透明加解密，原因是：经过分析，Word 绕过了 kernel32.dll 的读写文件的 API 函数，因此对 kernel32.dll 中函数的拦截解决不了 Word 的问题。现在分析有两种可能，一种是 Word 自己封装了一套 API，这套 API 直接调用 ntdll.dll 中的 API，甚至可能直接访问内核。另外一种可能是直接使用了 ntdll.dll 中的 API。现在分析前一种情况可能性比较大。但是如果对 ntdll.dll 进行拦截还会出现别的问题，本系统中多次使用了 ntdll.dll 中的 API 读写文件，并且记录 INI 文件当成日志的方法最终也会调用 ntdll.dll 层读写文件的 API 实现的，这些问题不是那么容易解决的，但是将来研究的方向。最理想的结果是能够发现 Word 中自己封装的 API 进行拦截。

（2） 加密算法还需要继续研究。现有的加密算法的强度还需经过理论验证，加密保护是本文保护数据文件的核心，所以加密算法对本论文研究的系统也是非常重要的。

（3） 本论文研究的内容可以同防入侵进程保护的功能模块整合到一起，实现一个在理论上更加安全的系统。解决打印、网络共享、邮件等方式造成的数据文件的泄密。

（4） 可以进行更细的权限管理。本论文保护的级别目前还是粗粒度的，在本论文实现的系统的基础上可以进行更加细致的文件访问或者修改、移动权限的控制。更大程度上降低由于加密造成的效率的损失和提高被泄露文件的密码破译的难度。

火龙果•整理
uml.org.cn

插图索引

图 1.1　Windows 系统架构 5
图 1.2　Windows 系统跟文件相关工作原理 6
图 1.3　驱动层实现透明加解密的示意图 7
图 1.4　过滤驱动层实现的透明加解密 8
图 1.5　API HOOK 实现透明加解密 9
图 2.1　AES 的加密与解密框图 21
图 3.1　数据文件存取透明加解密过程示意图 25
图 3.2　数据文件复制透明加解密过程示意图 26
图 3.3　记录 SHFileOperation 操作的代码 29
图 3.4　系统的设计模型图 32
图 4.1　进程间共享的内存的组织结构 36
图 4.2　多线程处理多个复制操作的流程图 39
图 4.3　异步读写文件的简单示意图 41
图 4.4　注入 DLL 模块的结构图 42
图 4.5　文件读取透明加解密过程 44
图 4.6　由于文件头存在而造成文件直接加密的问题 46

火龙果•整理
uml.org.cn

表格索引

表 2.1　Kernel32.dll 中封装的部分 API 14
表 2.2　Ntdll.dll 中封装的部分 API 18
表 2.3　加密轮数与密钥长度的关系 20
表 3.1　数据存放位置的安全性假设 25
表 3.2　保护的进程及文件类型列表 28
表 4.1　应用本系统的计算机要求 35
表 4.2　进程间通信的整个过程 37
表 4.3　文件复制模块中的数据结构介绍 37

火龙果•整理
uml.org.cn

参考文献

[1] Werner Vogels. File system usage in Windows NT 4.0. in: Proceedings of the seventeenth ACM symposium on Operating systems principles[C]. Charleston, South Carolina, United States: December 12-15, 1999. p.93-109.

[2] Mark E. Russinovich, David A. Solomon. Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server™ 2003, Windows XP, and Windows 2000[M]. Washington: Microsoft Press, 2005

[3] Satyam Sharma. TransCrypt: Design of a Secure and Transparent Encrypting File System[D]. Indian: Department of Computer Science & Engineering Indian Institute of Technology Kanpur, 2006.

[4] 计红梅. 杜跃进李铁军：2007，网络犯罪进入 2.0 时代[EB/PL]. [2008-1-17]. http://www.sciencenet.cn/htmlnews/2008117827242919954 6.html.

[5] 张静, Windows NT 文件系统过滤驱动的原理及应用 [J]. 信息安全与通信保密，2007(9)：114-118

[6] 李凡，刘学照，卢安，谢四江. WindowsNT 内核下文件系统过滤驱动程序开发 [J]. 华中科技大学学报：自然科学版，2003，31（1）：19-21.

[7] 宋维平，对称密码的流密码 [J]. 吉林建筑工程学院学报，2005，22（1），60-62.

[8] 林德敬，林柏钢，林德清，国内外分组密码理论与技术的研究现状及发展趋势 [J]. 天津通信技术，2002（4）：1-7

[9] 郑磊，马兆丰，顾明，基于文件系统过滤驱动的安全增强型加密系统技术研究 [J]. 小型微型计算机系统，2007，28（7）：1181-1184.

[10] 刘运毅，覃团发，倪皖荪，张淑仪，简评 ECRYPT 的候选流密码算法（上） [J]. 信息安全与通信保密，2007（7）.

[11] 刘运毅，覃团发，倪皖荪，张淑仪，简评 ECRYPT 的候选流密码算法（中） [J]. 信息安全与通信保密，2007（8）.

[12] 刘运毅，覃团发，倪皖荪，张淑仪，简评 ECRYPT 的候选流密码算法（下） [J]. 信息安全与通信保密，2007（9）.

[13] 李华，硬件加密解密技术综述 [J]. 广东自动化与信息工程，1996（3）：42-48.

[14] Gary Nebbett, Windows NT/2000 Native API Reference[M]. Thousand Oaks: New Riders Publishing, 2000.

[15] John Ayres, DELPHI WIN32 核心 API 参考(白金版)/DELPHI 开发人员系列丛书[M]. 北

火龙果•整理
uml.org.cn

京：中国电力出版社，2004

[16] 何明星，林昊. AES 算法原理及其实现 [J]. 计算机应用研究，2002，19（12）：61-63.

[17] 傅平，郑秉文. 基于 FileMapping 的进程间通信中共享内存技术的实现方法 [J]. 微计算机应用，2007，28（7）：767-770

[18] 微软公司. MSDN[EB/OL]. [2007-05-08]. http://msdn.microsoft.com/library/default.asp.

[19] Mathias Rauen. MadCollection online help[EB/OL]. http://help.madshi.net.

[20] Charles P. Wright, Jay Dave, and Erez Zado. Cryptographic File Systems Performance: What You Don’t Know Can Hurt You. In: proceedings of the 2003 IEEE Security In StorageWorkshop (SISW 2003) [C]. 31-31 Oct. 2003. P.47 – 47.

火龙果•整理
uml.org.cn

致 谢

一年来，本论文的研究项目得到了顾明老师和郭陟老师的大力指导和支持，从研究的指导思想和研究的具体细节方面都得到了两位老师的很大的帮助，在此表示最衷心的感谢。

本论文研究的内容跟其他相关内容一起得到了软件学院 SSRT 经费的支持，感谢孙家广老师、顾明老师和张慧老师及其他所有老师的支持。

最后感谢跟我一组的李欣峰、黄凌翔同学，不管从精神上还是实质上都给了我很大的支持。

火龙果•整理
uml.org.cn

附录A 外文资料的书面翻译

在 Windows NT4.0 系统中使用的文件系统（节选前半部分）

Werner Vogels
卡耐基梅隆大学计算机系

摘要

通过长时间的内核追踪，我们已经发表了关于 Windows NT 文件系统应用的研究。我们的目标是在 1985 年和 1991 年对文件系统追踪研究的基础上提出一个新的数据点，去调查 Windows NT 文件系统架构的应用细节，并且研究使用数据的统计行为。

在这篇论文中，我们通过详细的比较新旧追踪，通过可操作的特征，通过对文件系统和缓存管理器的应用分析，报告我们上述的那些观点。在对结构的深入剖析之后，我们提出一些对现在在所有文件系统应用领域普遍存在的重尾分布特性的证据。最后的一些变量在内部到达时间部分，持续次数部分，读写频率，读写缓存大小等，那些对系统工程、协调和测试很重要的部分。

1 引言

现在存在很多关于文件系统应用模式的文献，并且它帮助构造运行非常好的文件系统设计。然而，从上次在 1991 年的应用研究开始，计算的世界已经经历了主要的变化，不仅在计算和网络能力的增长超出预料，而在各方面专业生活集成计算中已经产生了新的一代新的系统和应用，这些已经不再和 80 年代后期的计算机操作相同。这些变化在计算机应用的变化可能对计算机文件系统的应用产生很大的影响。

对这个研究工作所有以及部分的数字和硬质复制版允许进行个人或者教学使用，保证不对这些应用征收费用，因为考虑到那些复制品不会产生利益和经济好处，并且那些复制品允许全文引用第一页。其他用途的复制，再次发表，在服务器上张贴或者重新安排目录，需要提前获得许可甚至付费。

在系统方面其中的一个变化就是一个新的商业操作系统的差生，微软的 Windows NT， 这个系统很快占有了市场上操作系统的很大份额。Windows NT 跟 Unix 非常的不同，对 Unix 文件系统的研究不合适应用在设计和优化 Windows NT 平台。

火龙果•整理
uml.org.cn

两个观察的结论引导我们相信我们需要关于文件系统应有的新的数据，并且它可能有其对研究 Window NT 平台的性能更有兴趣。

在这篇论文中我们发表一个文件系统的应用研究主要是通过对 1998 年 Windows NT4.0 操作系统。我们在这个研究中设计了四个目标：

1. 在先前文件系统应用研究的基础上，提供一个新的数据点，在 BSD 和 Sprite 操作系统上。
2. 详细研究 Windows NT I/O 子系统不同组件的研究，并且研究未文档化的应用，比如 FastIO 路径。
3. 研究 Windows NT 文件系统交互的复杂性，主要集中研究跟数据路径直接密切相关的那些操作。
4. 研究应用数据的总体分布。之前的研究已经暗示在分布的模型分裂问题上，但是我们相信那个问题是更加结构化的问题，并且需要详细的分析。

在这些直接的目标之后，我们要调查一个可利用的公共视野的数据集的结果，那个数据集可以用来作为文件系统模拟研究的输入，并且作为实际文件系统测试的配置信息。

Windows NT 文件系统应用的复杂性很容易被证实。当我们在记事本输入一些字符，保存成文件的时候将会出发 26 个系统调用，包括三个失败的打开尝试，1 个文件重写和 4 个文件追加打开和关闭序列。

这篇论文一下的部分是按照这样组织的：第二部分我们描述我们测量的系统，第三部分和第四部分我们描述我们收集和处理数据的方法。第五部分我们检查文件系统的布局信息，第六部分我们同 BSD 和 Sprite 追踪结果比较。第七部分包含我们收集的数据的详细的分析，第八、九、十部分包含详细的不同的 Windows NT 文件系统组件的操作。第十一部分触及到相关的工作，第十二部分摘要主要的研究的观点，大体的研究观察可以在表 1 找到。

与 Sprite 和 BSD 追踪的比较	操作特性
– 每个用户的吞吐量依旧低下，但是 Sprite（8Kb/sec）的三倍（24Kb/sec） – 读取数据的文件打开更少的时间：75%的文件仍然是开的但少于 10 毫秒跟在 Sprite 中的 75 个百分点 250 毫秒的对比 – 大多数的文件在长度上更短（80%的文件小于 26K），这个跟 Sprite 很相近 – 大多数（60%）读取文件是顺序的，但是跟 Sprite 相比也有一些随机存取的情况 – 大文件的大小按照巨大顺序增加（20%的是 4M 及以上），对这些文	– 突发的文件操作增长到一个可以打乱正常数据分析的地步 – 控制操作主宰文件系统请求：74%文件打开是为了进行控制或者目录操作 – 在 60%的文件读取请求的数据是从文件缓存中得到的 – 在 92%为读文件打开的情况下一次预读就足够把数据加载到内存中满足所有的子序列从缓存总度文件的要求 – FaetIO 路径会在 59%的读和 96%的写操作请求中被用到 – Windows NT 读写属性，例如临时

火龙果•整理
uml.org.cn

件的读取绝大多数是作为数据转移的用途 – 81%的新文件在 4 毫秒以内被重写（26%），在五秒以内被删除（55%）	文件，缓存写，只顺序读取等，可以很大程度上的提高存取性能，但是却未充分利用
追踪数据分布的特性	文件系统内容
– 又很充分的证据证明在所有的文件使用特性中存在最后的变量 – 所有的分布显示了重尾现象的明显存在，希尔评估值在 1.2 至 1.7 之间，是无限不同的证据 – 使用泊松处理和正态分布为文件系统使用建模可能会导致不正确的结果	– 可执行的，动态加载库和字体支配文件大小的分布 – 94%的文件大小的内容变化是在用户层面的树上（个人文件缓存） – 用户层面 90%的变化发生在 WWW 缓存中 – 文件中记录文件时间的属性是不可靠的

表 1：观察报告的摘要

2 研究的系统

我们研究的是一个产品的环境，在这个环境中五个有五个不同的分类和使用：

Walk-up 使用。用户使用共有的定位在中心设施的可以利用系统。这些用户的行为可以非常的不同，从可以分析到为文档准备的程序开发。

共享（Pool）使用。不同组的用户共享一些列位于他们周围的的系统。这些用户主要行为是程序开发，但是也进行一些多媒体、仿真和数据处理的操作。

个人使用。系统只提供给一个单独的用户使用，并且系统在这个用户的办公室中。绝大多数的操作是在分类和协作模式下的应用，比如电子邮件和准备文档。很少的集合的用户用系统进行程序开发。

管理员使用。所有的这些系统被用来进行一些很少的通用支持任务：数据库交互，协作应用，和一些专门的管理员工具。

科学使用。这些系统支持主要的计算任务，比如模拟，图形处理，和统计处理。这些系统专门给一小部分特殊的应用程序使用。

前 4 个类别都可以用奔腾 Pro 和奔腾 2 系统，64-128 兆的内存和 2-6G 的本地磁盘。共享使用机器通常更加搞得配置（200-450MHz，一些双重处理器），而其他的机器都可以在 200MHz 的范围。科学使用的类别包含了奔腾 2 450 Xeon 双重处理器和四重处理器，256M 的内存和 9-18G 的 SCSI Ultra-2 本地磁盘。所有的系统都是 Windows NT 4.0 最新的服务包。在追踪的时候文件系统的年龄在 2 个月到 3 年之间，平均 1.2 年。

对所有的用户讯在一个中心网络文件服务支持。只有一个有限的个人工作站集合是通过提个备份的机制所支持的，所以中心文件存储是隐含的支持的。所有的系统都是通过一个 100Mbit/ses 可选的局域网同文件服务器连接。用户被

火龙果•整理
uml.org.cn

组织成 3 种不同的 NT 域，一个是 walk-up 使用，一个是通用的使用，另一种是实验的。实验域包含一个同通用域可信的关系并且网络文件服务是可以共享的。Walk-up 域同其他的域是通过网络防火墙分离的，并且具有自己的网络我呢间服务。

从 250 个对工具可选的系统中，我们选择出一个 45 基于私人考虑和管理方便的系统。这些系统中的一小部分在 1998 年的上半年被跟踪过 3 个两周的时间，当我们在调整确定类型和数量的数据集的时候。一些变化是同我们研究是一个探索的本质，我们的数据集需要根据初始的分析进行调整的事实相关的。其他的调整是跟我们保持每次追踪数据记录最少的数据量，同时仍然能够记录足够的信息给我们的分析的要求相关的。我们不是每次都成功的，比如，记录读请求大小如果字节实际上读操作没有记录的话就产生很少的作用。这篇文章的分析报告基于一个最终的数据集合，在 1998 年的 11 月和 12 月运行了 4 周的俄时间。45 个系统在那段时间产生了接近 19G 的跟踪数据。

从那时开始，我们运行了其他的追总，选择系统了解那些在初始的追踪中没有了解的特殊的问题，比如突发的 IO 页，从压缩的大文件和目录操作中读取数据。

3 收集数据

被研究的系统报告两种类型的数据：1）本地文件系统的状态快照和 2）所有的 I/O 请求被发送到本地的和远程的文件系统。第一种类型用来提供关于文件系统在每次追踪时期开始和建立后来请求被定向的文件的基本集合的基本信息。在第二种类型数据中所有的文件系统操作都实时记录。在每种系统中，一个追踪代理为了提供追踪处理的远程控制的接入点而被安装。追踪代理负责采集周期快照，并把追踪事件流引入到收集服务器。收集服务器是三个专门的文件服务器，负责接受输入的事件流，并把它们存储成压缩文件的格式已被日后的检索。追踪服务器在启动的时候会自动运行，并且尝试连接到一个收集服务器，如果连接成功，它将会初始化本地的数据集合。如果这个追踪代理失去了跟收集服务器的链接，它将会挂起本地的操作指导链接重新建立。

3.1 文件系统快照

每天早晨 4 点，追踪代理服务启动一个线程去采集本地文件系统的快照。产生快照是通过递归遍历文件系统的树，产生一个包含每个文件属性和目录的记录序列，这样原始的树可以通过这个序列恢复。属性存储在一个动态的记录中的是文件名字和大小，创建、最后修改和最后读取的时间。对目录存储的名字，文件实例的数目和子目录的数目。名字被存储成一个简短的形式因为我们

火龙果•整理
uml.org.cn

主要对文件类型而非文件单个的名字感兴趣。在 FAT 文件系统下，创建和最后读取时间没法获得，因此可以忽略。

追踪代理把这些记录转移到收集服务器，在收集服务器他们被存储成研所的格式。对这些收集的文件的读取主要是通过 OLE/DB 接口，把每个文件表示成两个数据库的表格：一个包含目录，另一个包含文件信息。

产生一个 2G 的磁盘快照需要在 200MHzP6 上消耗 30 至 90 秒的时间。

3.2 文件系统追踪工具

为了追踪文件系统的活动，操作系统被装备，因此它可以记录所有文件存储操作。Windows NT 文件系统操作的一个重要的子集被处理可执行加载镜像和在它的内存映像文件接口不中的虚拟内存管理器激活。因此，像以前那样只追踪系统调用层是远远不够的。我们的追踪开发了 Windows NT 对设备驱动透明支持，这种支持是通过引进一个过滤驱动记录所有的发向实现文件系统的驱动的请求。追踪驱动把每个驱动的实例绑定到本地文件系统（除了那些可移除的设备）和实现那些提供通过 CIFS 协议对远程文件进行存储的网络重定向的驱动。所有的文件系统的请求都会被传给 Windows NT 操作系统的 I/O 管理组件，不管这个请求是来源于用户层的处理还是另一个内核的组件，比如虚拟内存管理器或者网络文件服务。在验证了请求后，I/O 管理器会把请求显示给驱动链上最上层的设备驱动，处理文件所在的卷信息。有两种驱动存取机制：意识一般的基于请求机制的包，在这种机制下 I/O 管理器传送一个包（一个 IRP-I/O 请求包）描述请求，顺序的到达驱动链。在处理完请求包后驱动返回给 I/O 管理器，然后管理器把它传给下一个设备。驱动习惯后处理请求，在包被他的目的驱动处理以后，修改包使之包含返回程序的地址。第二种驱动存取机制，被称为 FastIO，显示一个直接的引用机制方法，它调用下个驱动的同一个方法，一直这样持续下去。FastIO 路径会在第十部分详细介绍。

追踪驱动记录了 54 个 IRP 和 FastIO 事件，这些事件代表了所有主要的 I/O 请求操作。每个操作的详细情况被存储在内存中固定大小的记录中，这些记录定去刷新到收集服务器。记录的信息依据特定的操作，但是每个记录包含至少一个对文件对象。IRP，文件和标志头，请求过程，现在文件字节便宜和文件大小以及操作结果的状态的引用。每个记录接收到两个时间戳：一个是在操作开始的时候，另外一个是在结束的时候。这些时间戳之间有 100 纳秒时间间隔。被记录的其他信息取决于特殊的操作，比如便宜，长度和读写操作返回的字节，或者创建操作的选项和属性。曾加的追踪记录被写成每个新文件对象，映像对象文件名的标识。

火龙果•整理
uml.org.cn

追踪驱动对记录存储使用三倍缓存机制，每个存储的缓存可以记录 3000 条数据记录。一个空闲的系统将会在一个小时之内存储空间被填满；在繁重的负担下，缓存会在 3-5 秒内被填充。如果缓存在 1 秒内被填充，那么在主机和服务器之间增加的通信延迟将会导致追踪模块存储空间的溢出。追踪代理需要发现这样一个错误，但是这个不会在追踪过程中发生。

内核压型已经显示出了可接收模块的影响；在 IRP 负载中的追踪活动中在 200MHz P6 下 0.5%的负载是有它贡献的。

在一个 24 小时的周期中，文件系统追踪模块将要记录 8 万至 140 万的事件。

3.3 可执行的和页面调度的 I/O

Windows NT 为内存映像文件提供了功能性经常被很多的使用于两个系统服务：（1）可加载的执行程序和动态加载库是基于内存映像文件，并且（2）缓存管理器为在其中的每个文件建立了一个文件影响，并且利用页面不中的机制触发 VM 管理器把实际的数据加载到缓存中。文件系统的机密综合，缓存管理器和虚拟内存管理器产生了很多的问题如果我们想要去精确的计算所有我呢间系统的操作。虚拟内存管理器应用 IRPs 去请求把文件的数据加载到内存中，并且这些 IRPs 跟常规的请求具有相同的路径。文件系统能够认出虚拟内存管理器的请求，通过一个在包头的 PageingIO 位。当追踪文件系统的时候，我们可以忽略一大部分的页面调度请求，当他们表现出重复的行为时：一个请求从处理触发在虚拟内存管理器中一个页面不中，从虚拟内存管理器中触发一个页面调度请求。然而，如果我们忽略页面调度请求，我们将会失去所有跟可执行程序和动态运行库加载相关的请求，然后使用内存映像文件。我们决定去记录所欲哦的页面调度请求并且在分析处理中筛选出缓存管理器引发的重复。

我们决定赞同这个增加的复杂度，即使它几乎加配了我们的追踪，因为基于文件系统的可执行程序计算精确度的需要。在早期的追踪中，可执行的系统调用被追踪并记录着执行程序的大小，这个被用作调整所有的追踪测量。在 Windows NT 系统中，这个不太合适，因为虚拟内存管理器的优化行为：可执行的代码页面在程序结束后频繁的存在在内存中，这样可以唉程序再次执行的时候提供快速的启动。

3.4 丢失的和干扰的数据

我们相信系统在记录所有主要的文件系统 IO 事件的方面上是完备的，是足够看到 Windows NT 文件系统通用应用的。还有很多比较不重要的操作，我们没有留下记录信息（比如加锁和安全操作），但是他们都在我们研究的领域之外。在我们的分析中我们发现一个特殊的干扰源：本地文件系统可以通过网络被其他系统所共享。我们发现这个存取是很少的，因为通常它只被用来复制少量的

火龙果•整理
uml.org.cn

文件或者共享一个测试的可执行程序。考虑到这些服务操作受限制的影响，我们决定不会把他们从我们的追踪集合中去掉。

4 数据分析的过程

数据分析给我们展示了一个非常显著的问题：数据的数量太多了。在我们这篇论文中报告的收集的数据的总量接近 20G，表示为超过 1.9 亿的记录。本地磁盘静态的快照导致了 2400 万的数据。

大部分跟追踪相关研究都是集中在找到对特殊问题和猜测集合的答案，这些研究可以通过广泛的统计技术得到满意的答案，从而减少一些不易分写的数据的分析。考虑到我们实验具有探索的本质，我们需要一种能够在数据中浏览和搜索特定模式的机制，管理呈现的细节的层次。发展一个有代表性的数据就像这些操作可以被表示成很有效率的很多百万的记录一样最终将是一个非常困难的问题。

我们能够通过认识到这是一个同数据仓库支持和在线分析处理（OLAP）相似的问题而找到解决方案。我们研发了一个不规范的星型模式去追踪数据和构造相关的 SQL-sever7.0 的数据库表格。我们把在追踪运行过程中收集的信息的一系列的摘要表示成多维的表格。多维的表格被用在分析像对多维立方体对追踪数据的表示的分类斧子上。多数的多维支持多个层次的摘要，允许探索各种不同层次的细节的摘要。一个分类的例子是一个信箱的文件用.mbx 类型就是邮件文件类的一部分，同时也是应用文件的分类。我们跟传统的数据仓库模型不同的是我们应用了两张事实表格（表格记录了所有实际的信息），而不是一个表格。第一个表格（追踪）巨鹿所有追踪数据的记录，用一个关键的引用去索引多维的表格。第二章表格（实例）记录跟文件对象实例相关的信息，这是跟单个文件打开关闭序列，在文件生存时间中所有对这个对象的操摘要数据的组合相关的。虽然第二章表格可以通过 OLAP 系统产生，我们的决策时用两张事实表格通过减少所有对追踪记录接触的操作处理减少在追踪表格中需要的对视力表格索引的存储总量。应用一个产品质量数据库系统为我们提供了一个非常不同的数据存储设施。一个操作可以解除所有的追踪数据集合，比如计算基本统计的描述符（avg,stdev,min,max）的中间请求时间，在以前得到追踪数据的基础上，通过优化 C 语言过程，运行时间变成以前的 30%。更多的复杂统计处理不能用 SQL 或者 MDX 表示，可以使用 SPSS 统计处理包提供的接口直接应用在数据库中。

因为我们处理的数据使用不同的分类分组（比如，每天，每个节点，每个用户，每个过程等）我们的分析频繁的不会导致单个的值在统计描述中。在文章中，我们将会展示这些值的范围，或者相关的只显示上限和下限。