Windows平台个人防火墙技术的研究

孙 卫[1],荆 霞[1],王 诚[2],庄卫华[2]

(1. 南京审计学院 信息科学学院, 江苏 南京 210029;2. 河海大学 计算机及信息工程学院, 江苏 南京 210098)

摘 要:针对目前Windows下个人防火墙软件在拦截网络数据封包上所存在的不足,提出了一种新的用户态和内核态混合的网络数据封包拦截方案。该方案基于Winsock2服务提供者SPI和NDIS HOOK,能克服单一的网络数据包拦截方案的缺点,在理论上可以截获所有通过Windows的网络封包,同时接口简单、思路明确并且容易实现软件的自动安装。根据实验系统的测试表明,新的网络数据包拦截方案有很好的稳定性,系统具有良好的使用价值。

关键词:Windows平台;防火墙;包拦截;NDIS HOOK;SPI

中图分类号:TP393 **文献标识码**:A **文章编号**:1672-8750(2010)02-0084-09 **收稿日期**:2009-12-12

作者简介:孙卫(1971—),女,江苏靖江人,南京审计学院信息科学学院讲师,主要研究方向为数据挖掘、数据库应用、网络安全;荆霞(1978—),女,江苏金坛人,南京审计学院信息科学学院讲师,主要研究方向为数据库应用、网络安全;王诚(1982—),男,江苏靖江人,河海大学计算机及信息工程学院硕士生,主要研究方向为数据挖掘、网络安全;庄卫华(1950—),女,江苏如皋人,河海大学计算机及信息工程学院副教授,硕士生导师,主要研究方向为数据挖掘、数据库应用、网络安全。

因特网的发展与应用使得PC给人们的生活带来了革命性的变化,随之而来的网络安全问题正威胁着每一个网络用户。现在防火墙已成为网络中实施安全保护的核心,个人防火墙是目前最常用的网络安全防范工具。当前国内有超过90%的电脑使用Windows系统,因此研究该平台下作为网络安全工具之一的个人防火墙技术也成为各公司实际工作者和高校学者研究的对象。

一、Windows网络数据包拦截方案比较

网络数据包的拦截是防火墙的核心技术,它是整个防火墙系统高效工作的基础,为防火墙系统提供数据来源。Windows提供了多种数据包的截获方法,从操作系统的不同层次来看,大体上可以分为两种方案:用户态的截包技术和核心态的截包技术。用户态的截包技术实现起来相对容易,并且容易实现内容过滤;核心态的截包技术实现起来相对比较复杂,并且随着截包技术方案所处的层次不同,需要考虑的因素和实现的复杂性也因"层"而异。目前Windows下个人防火墙软件常见的网络数据包的拦截技术有[1-2]:利用Winsock2服务提供者接口(SPI,Service Provider Interface)的拦截技术,内核态下基于IP过滤钩子(IP Filter Hook)的拦截技术,内核态下基于传输驱动接口(Transfer Driver Interface)的拦截技术,内核态下编写中间驱动(Intermediate Driver)的拦截技术。

(一) Winsock2服务提供者接口

Winsock2服务提供者接口和Winsock 2 API接口几乎一一对应。Winsock2是围绕着Windows开放系统架构(WOSA,Windows Open System Architecture)来设计的,WOSA在Winsock和Winsock应用程序之间有一个标准API;在Winsock2和Winsock2服务提供者(比如TCP/IP)之间有一个标准的SPI。Winsock2中使用的传输服务提供者有两类:基础服务提供者和分层服务提供者。基础服务提供者执行网络

传输协议的具体细节,其中包括在网络上收发数据之类的核心网络协议功能。分层服务提供者只负责高级的自定义通信功能,然后调用下层的基础服务提供者来传输真正的数据。图 1 显示了 Winsock2 的基本组织结构[3]。

通过图 1 可以看出,在基础服务提供者之上插入自己的分层服务提供者就可以截获网络数据包,从而可以执行传输质量控制、扩展 TCP/IP 协议栈、URL 过滤及数据安全管理、带宽管理等任务。

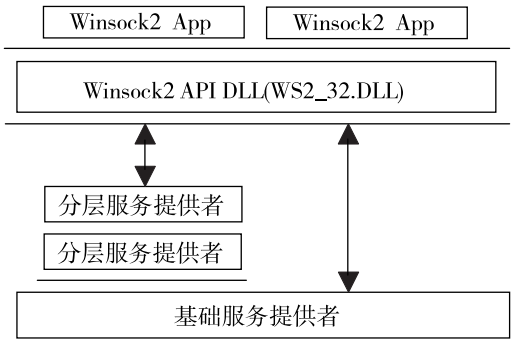

图 1　Winsock2 的基本结构

Winsock2 SPI 工作在用户模式,其优点在于:编程实现比较容易,可以获取感兴趣的信息,并且包是完整的,处理方便。其缺点在于:安全性不高,容易被绕过导致拦截失败,只能处理 SOCKET 类型的包,对 ICMP、IGMP 之类的 IP 层数据包的处理无能为力。

(二) IP 过滤钩子

IP Traffic Filter Driver 是 Windows 2000 开始给出的过滤驱动,Windows 为该过滤驱动预留了过滤接口。IP 过滤钩子截包方式是通过编写内核模式的驱动,利用系统提供的 IP Traffic Filter Driver 过滤驱动接口,通过构造指定请求码的 IRP 包设置回调函数,从而实现包的截获。

IP 过滤钩子的优点在于:它忽略了很多细节,实现起来比较容易,开发人员只要重点面向功能设计就可以了;可以截获大多数应用程序感兴趣的数据包类型(如 IP 包、TCP 包、UDP 包、ICMP 包),基本可以满足需要。其缺点在于:一个系统只允许安装一个过滤钩子,所以这种技术很难为商业软件产品所采用;采用这种方式对截获的数据包的访问有一定的限制,对接收到的数据包可以访问,对发送出去的数据包只能读取包头信息。

(三) 传输驱动程序接口客户端(Transfer Driver Interface)

传输驱动程序接口(TDI)是核心态的驱动程序,用于实现网络 API 的核心态部分。TDI 客户端根据传输驱动程序接口标准,格式化 I/O 请求包(IRP),然后再将 IRP 发送给 TDI 的传送器(也称为 NDIS 协议驱动程序或者协议驱动程序),在请求完成后,TDI 的传送器根据 TDI 客户端设置的完成例程或者事件执行 TDI 客户端指定的代码,从而完成应用层的 API 调用到核心层的实际网络传输的交互过程。图 2 表示了三者在 Windows 平台下所处的层次关系和 TDI 客户端与 TDI 传送者的交互过程。

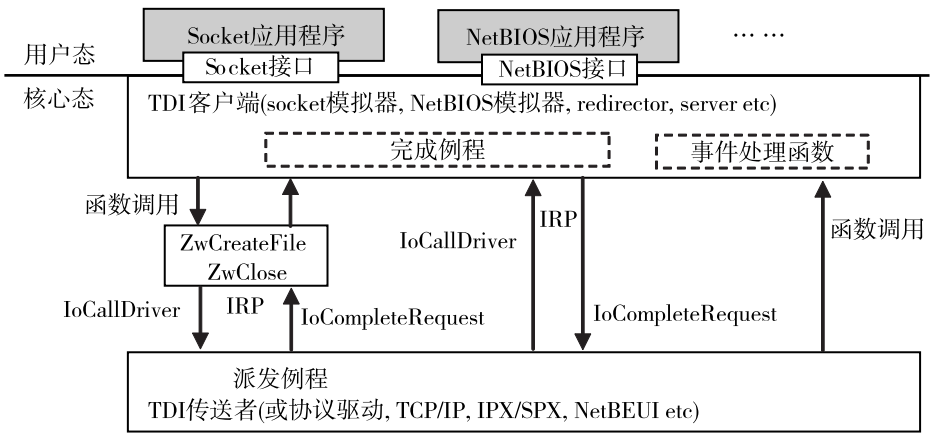

图 2　应用程序、TDI 客户端和 TDI 传送者的关系

通过上面的分析可以看出,利用 WDM[4](Windows Driver Model,Windows 驱动模型)所支持的过滤驱动程序,只要创建一个自己的过滤驱动程序对象并附加到感兴趣的实际传送对象上就可以截获到数

据包了。

采用 TDI 客户端过滤驱动程序方式拦截数据包,实现起来较为复杂,需要开发人员对 WDM 和 Windows 的网络体系结构比较熟悉,而且由下层协议驱动接收并直接处理的数据包不会传送到上层,无法过滤某些接收的数据包,典型的就是 ICMP 协议的数据包。但是,这种方法是目前大多数防火墙软件的截包方法之一,相对于中间驱动,它容易实现自动化安装,而且 TDI 层的网络数据包的拦截还可以得到进程的详细信息,这个也是更低层次的驱动所没有的。

(四) 中间驱动(Intermediate Driver)

中间驱动位于 TDI 传送者和小端口驱动之间。编写 NDIS 中间驱动来拦截网络数据包是 NDIS 中比较常用的截包手段,这种方式可以自己处理截获的数据包,比如重新封包、加密、内容过滤、网络地址转换等。

中间驱动的优点在于:能够对流经网络设备(如以太网卡)的所有输入输出报文进行处理,因此能够获得更多的操作空间。其缺点有:不能对拨号网络的报文进行处理,这是实际应用中比较严重的弊端;中间驱动的安装比较麻烦,无法实现程序的自动化安装。

二、基于 SPI 和 NDIS HOOK 的截包技术

要实现 Windows 平台个人防火墙的功能,关键是要在合适的地方拦截所有进出 Windows 的网络封包,并且根据用户规则对拦截的网络封包进行过滤。由于目前 Windows 下个人防火墙软件在拦截网络数据封包上存在着上述诸多不足,因此本文提出一种新的基于用户态和内核态相结合的网络数据封包拦截方案,该方案采用在核心态使用 NDIS HOOK 技术、在用户态插入服务提供者 SPI,实现网络封包的双重过滤,实现高效和便捷的结合。

(一) 基于 Winsock2 SPI 网络数据包的拦截

1. Winsock2 SPI 运行原理

Winsock 是为上层应用程序提供的一种标准的网络接口,它为上层应用程序提供透明的服务,上层应用程序不用关心 Winsock 实现的具体细节。Winsock2 引入的一个新功能就是打破服务提供者的透明,让开发者可以自己编写自己的服务提供者接口程序,即 SPI 程序。SPI 以动态链接库(DLL)的形式存在,它工作在应用层,为上层 API 调用提供接口函数[5]。

SPI 过滤模块有点像一个系统范围的 Winsock 钩子(HOOK),SPI HOOK 和一般的 API 函数 HOOK 概念不同,一般的 API 函数 HOOK 通常的做法是直接通过修改程序的导出表来实现,就像 HOOK NDIS.SYS 中的函数那样。但是 SPI HOOK 并不需要这么做,其原因是 SPI 的加载方式比较特殊。SPI 的信息保存在操作系统的注册表中,对于 Windows XP,这个位置在:HKLM\SYSTEM\CurrentControlSet\Service\Winsock2\Protocol_Catalog9\Catalog_Entries。操作系统会根据这张表加载 SPI 模块,所以只要修改这张表让系统加载自己的 SPI 模块,然后再利用自己的模块加载系统的相应模块。自己编写的 SPI 程序安装到系统之后,所有的 Winsock 请求都会先发送到这个程序,并由它完成网络调用。由于系统提供的 SPI 已经可以完成网络传输功能,所以自己编写的 SPI 没有必要重新编写这部分功能,一般可以直接调用系统函数完成网络传输。在自己的模块中完成过滤工作,实际的传输工作交给系统的相关模块完成。显而易见,只要不向系统转发某个请求,这个通信就被阻止了。图 3 表示了插入用户自己的 SPI 前后的结构。

2. 向 Winsock2 SPI 插入基础服务提供者

Winsock 网络应用程序调用 Win32 API WSAStartup 初始化 socket 时,Windows 将根据注册表找到传输服务提供者 SPI 的位置,然后加载该 DLL。系统对每种 SPI 的协议信息是以如下数据结构存储的:

```
Typedef struct_tagPACKEDCATALOGITEM{
    TCHAR szLSPName[MAX_PATH];
    WSAPROTOCOL_INFOW protocolInfo;
```

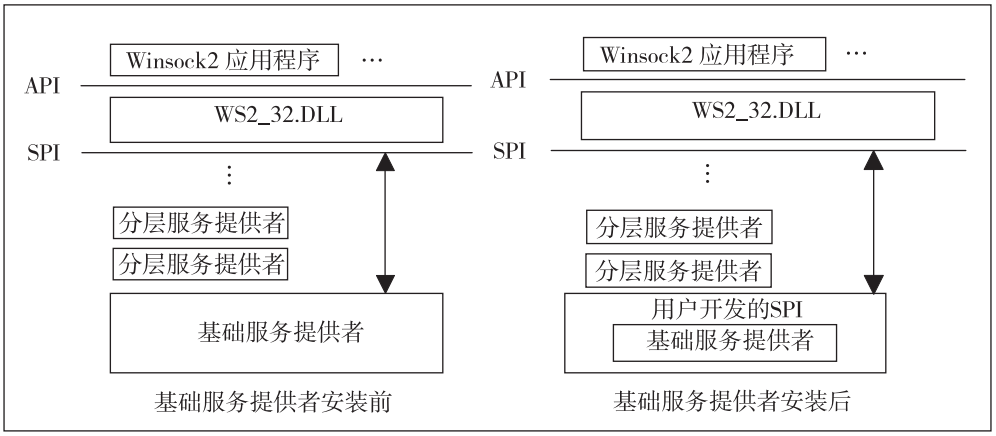

图 3　用户自己的 SPI 安装的前后结构

}PACKEDCA LOGITEM, * PPACKEDCATALOGITEM;

其中 szLSPName 表示服务提供者的完整路径或者环境变量表示的路径,比如% SystemRoot% \system32\msafd. dll。将这个值替换为用户自己开发的 SPI,使得应用程序初始化网络应用时用户自己开发的 SPI(以动态链接库的形式存在)被系统自动加载。

WSAPROTOCOL_INFOW 结构的定义在 MSDN 中有说明,本文主要关心的是它的类型为 WSAPROTOCOLCHAIN 结构的 ProtocolChain 成员,WSAPROTOCOLCHAIN 结构的定义如下:

```
typedef struct_WSAPROTOCOLCHAIN{
    int chainlen;
    DWORD chainEntries[MAX_PROTOCOL_CHAIN];
}WSAPROTOCOLCHAIN,FAR * LP WSAPROTOCOLCHAIN;
```

其中 Chainlen 的值为 0 表示分层服务提供者;为 1 表示基础服务提供者;大于 1 则是分层服务提供者在协议链中的顺序编号。

因此要向系统插入自己的基础服务提供者要进行下面的步骤:(1)读取系统注册表,枚举出系统所有的基础服务提供者,也就是 chainlen 值为 1 的键值;(2)备份系统的基础服务提供者到注册表的其他位置;(3)替换系统的基础服务提供者的文件名称为自己的基础服务提供者,也就是将自己的基础服务提供者名称写入 szLSPName,然后写回注册表。在经过这样的替换动作后,当 Winsock 应用程序请求网络功能时,Windows 将从注册表中直接读取被替换后由用户自己开发的基础服务提供者。

(二) 基于 NDIS HOOK 网络数据包的拦截

1. NDIS 框架

网络驱动接口规范(NDIS,Network Driver Interface Specification)最初是由 3COM 和 Microsoft 于 1989 年制定的 Windows 网络驱动程序的标准。NDIS 将网络硬件抽象为一套标准的接口,屏蔽了下层驱动对硬件的管理。NDIS 支持 3 种类型的驱动:小端口驱动(Miniport drivers)、中间驱动(Intermediate drivers)、协议驱动(Protocol drivers)[6]。NDIS 架构如图 4:

小端口驱动就是常说的网卡驱动,它负责网卡的管理,包括通过网卡发送和接收数据,同时为上层提供接口,小端口驱动一般由硬件开发商提供。中间驱动位于小端口驱动和协议驱动之间,是具有链路层和网络层的驱动。协议驱动位于 NDIS 体系的最高层,经常用作实现传输协议堆栈(如 TCP/IP 或 IPX/SPX 堆栈)的传输驱动的最底层驱动。

NDIS 库不但定义了上下层驱动程序之间的通信接口,而且也替驱动程序管理了很多的内部工作,比如中断的响应、各种通知消息的派发、调用相关的入口函数等等。

2. NDIS HOOK

从 NDIS 库函数入手拦截数据包的基本思想就是改写 NDIS 的收发函数,如 NdisSend、ProtocolRe-

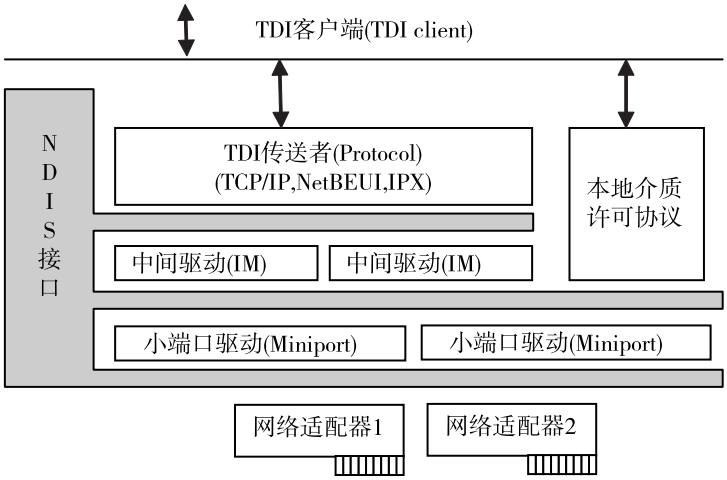

图 4　NDIS 架构

ceive 之类的函数,然而 NDIS 内部实现的收发并不是真正通过 NdisSend 之类的函数,NDIS 库也没有导出一个叫 NdisSend 的函数。

事实上,通过观察协议驱动向 NDIS 注册时填写的结构 NDIS_PROTOCOL_CHARACTERISTIC 可以发现,这个结构中填写的各种函数的地址才是协议驱动、NDIS 库以及小端口驱动三者交互的关键所在。Windows DDK 2003 中该结构的定义如下:(有删减)

```
typedef struct_NDIS_PROTOCOL_CHARACTERISTICS{
    …
    OPEN_ADAPTER_COMPLETE_HANDLER OpenAdapterCompleteHandler;
    RECEIVE_HANDLER ReceiveHandler;
    RECEIVE_COMPLETE_HANDLER ReceiveCompleteHandler;
    …
};
```

当从网卡接收数据包时,NDIS 会通过结构中的 ReceiveHandler 或 ReceivePacketHandler 通知协议驱动程序有一个该协议的数据包进入,所以这里就是拦截接收包的地方。但是这个结构中并没有发送包的派发函数入口。通过分析 NdisSend 的宏定义,可以发现发送操作的入口函数实际在 NDIS_OPEN_BLOCK 这个结构中。Windows 2003 DDK 中 NDIS_OPEN_BLOCK 的定义如下:(有删减)

```
typedef struct_NDIS_OPEN_BLOCK{
    …
    PNDIS_PROTOCOL_BLOCK ProtocolHandle;// pointer to our protocol
    PNDIS_OPEN_BLOCK ProtocolNextOpen;// pointer to next block
    SEND_HANDLER SendHandler;
    …
};
```

当协议驱动向底层小端口驱动发送数据时,NDIS 会实际调用结构中的 SendHandler 来发送数据。因此,NDIS HOOK 只要获取到这两个结构并改写其中的函数,使其指向自己定义的函数就可以了。那么,如何获取这两个结构呢? 从协议驱动和下层适配器的通信流程我们可以了解到编写协议驱动的基本步骤如下:

第一步,协议驱动调用 NdisRegisterprotocol 函数向 NDIS 注册本协议,注册成功后 NDIS 返回 NDIS_PROTOCOL_BLOCK 结构;

第二步,当有适配器可用时,NDIS 调用注册协议时提供的 NDIS_PROTOCOL_CHARACTERISTIC 结

构中的 ProtocolBindAdapter 函数将协议和适配器绑定。在绑定适配器之前,协议驱动 ProtocolBindAdapter 函数会先调用 NDIS 库函数 NdisOpenAdapter 打开适配器,打开适配器成功则返回 NDIS_OPEN_BLOCK 结构;

第三步,完成协议驱动的各个接口函数进行收发工作,直到协议驱动的注销,进行第四步;

第四步,NDIS 调用协议驱动调用提供的 ProtocolUnbindAdapter 取消绑定;

第五步,协议驱动调用 NdisDeregisterProtocol 函数取消注册。

其中,Windows 2003 DDK 对 NDIS_PROTOCOL_BLOCK 的定义如下:(有删减)

```
typedef struct_NDIS_PROTOCOL_BLOCK{
    PNDIS_OPEN_BLOCK OpenQueue; //queue of opens for this protocol
    …
    NDIS_PROTOCOL_CHARACTERISTICS ProtocolCharacteristics;//handler addresses
    Struct_NDIS_PROTOCOL_BLOCK * NextProtocol;// Link to next
    …
};
```

从上面的各个结构可以看出,NDIS_PROTOCOL_BLOCK 和 NDIS_OPEN_BLOCK 都是一个单向链表,前者指明了 NDIS 内部维护的各种协议,这些协议已经在当前系统中注册过,后者指明了该协议和适配器的绑定情况,最后形成如图 5 所示的结构。

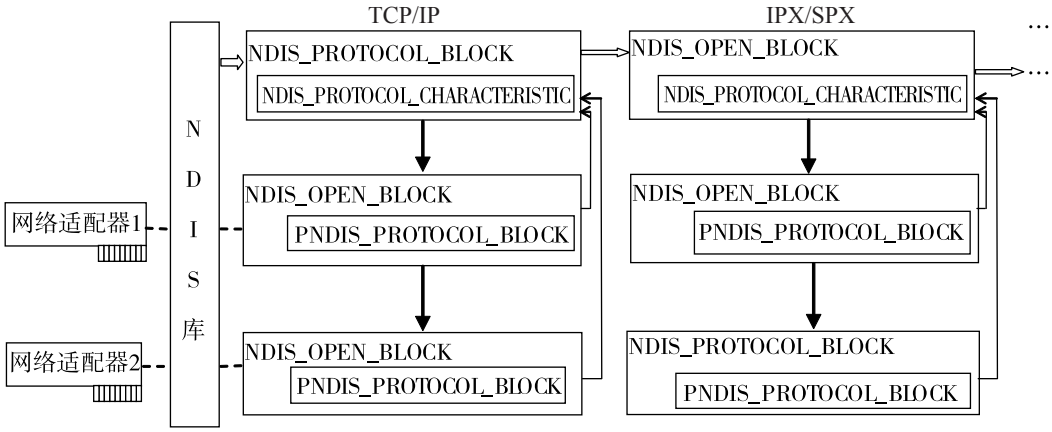

图 5　NDIS 对协议的管理

NDIS 正是通过这张链表来维护整个网络数据的收发,当网卡有数据包到达时,NDIS 获得中断通知,随后根据协议类型和适配器找到 NDIS_PROTOCOL_BLOCK 结构中注册的接收处理函数,使得协议驱动得到数据;当协议驱动发送数据时,NDIS 调用 NDIS_OPEN_BLOCK 结构中注册的发送函数发送到下层小端口驱动或者中间驱动。

综上所述,通过函数 NdisRegisterProtocol/NdisDeregisterProtocol 和 NdisOpenAdapter/NdisCloseAdapter 就可以得到每个协议驱动的 NDIS_PROTOCOL_BLOCK 结构和 NDIS_OPEN_BLOCK 结构,然后改写结构中感兴趣的函数就可以达到拦截数据包的目的了。下面给出了实现的伪码:

```
Extern "C"NTSTATUS DriverEntry(IN PDRIVER_OBJECT drvObj,IN PUNICODE_STRING regPath){
/* 先注册一个假协议获取首个 NDIS_PROTOCOL_BLOCK */
PNDIS_PROTOCOL_BLOCK protocolBlock = NULL;
NdisRegisterProtocol(&status,ProtocolBlock,…);
/*然后扫描协议链获得所有当前已经向系统注册的协议,并修改之*/
While (ProtocolBlock) {
```

```
        /* 修改 NDIS_PROTOCOL_BLOCK 表 */
        HookProtocolBlock(protocolBlock);
        PNDIS_OPEN_BLOCK openBlock = protocolBlock -> OpenQueue;
        While (openBlock){
        /* 修改 NDIS_OPEN_BLOCK 表 */
        HookOpenBlock(openBlock);
        openBlock = openBlock -> ProtocolNextOpen;
    }
    protocolBlock = protocolBlock -> NextProtocol;
    }
    /* 最后取消注册的协议 */
    NdisDeregisterProtocol(&status,protocolBlock);
    }
```

三、实验系统的设计与测试

为了验证本文提出的方案是否可行,本文根据该方案设计了一个实验系统并对该系统进行了测试。

(一) 系统总体设计

系统的整体架构如下图:

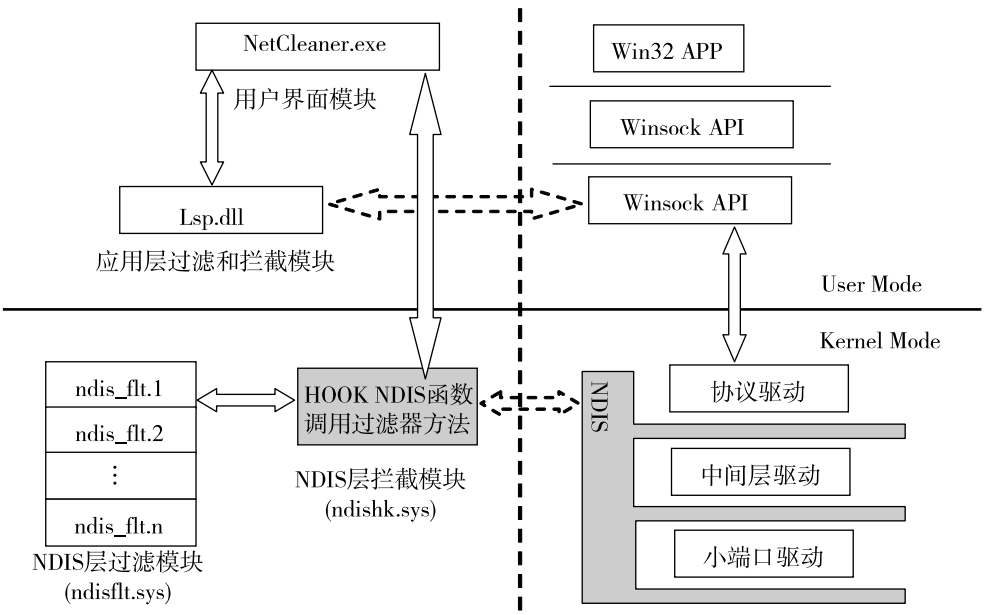

图 6　系统总体架构图

从图 6 中可以看出,本系统主要有下面几个部分:

1. 核心层中采用 NDIS HOOK 技术的拦截模块(NDISHK. SYS)和过滤模块(NDISFLT. SYS)

在内核层中,网络封包的拦截由拦截模块负责,然后交给过滤模块进行处理。

拦截模块(ndishk. sys):通过 HOOK NDIS 函数的方法首先 HOOK 了几个关键的 NDIS 函数,这样,当有协议驱动向 NDIS 注册时就可以改写协议驱动的收发函数,达到拦截数据包的目的。当拦截模块截获到网络封包时,如果当前系统没有加载过滤器,则调用自己的默认过滤器(实际上就是全部放行),否则拦截模块将调用过滤模块中的方法。同时,拦截模块还和用户界面模块有一定的交互,通过用户界

面模块可以对应用层处理过的包作出相应的动作。

过滤模块(ndisflt.sys):是一个普通的 WDM 驱动程序,它打开拦截模块设备对象,向拦截模块附加各种过滤方法。过滤模块中采用过滤器堆栈的思想,以解决内核层过滤的扩展性问题。

这里把拦截模块和过滤模块分开,主要有以下几点考虑:(1)拦截模块本身是一个完整的模块,可以独立工作。当系统中只存在拦截模块的时候,它不过滤任何数据包。(2)由于采用 HOOK NDIS 技术的特点,使得拦截模块自身依赖于各种 Windows 平台的 NDIS 版本,这样也使得拦截模块自身不应该和其他模块有过多的耦合。(3)过滤模块实际上是一系列过滤方法的集合,通过特定的接口,它可以适合于采用任何方法的拦截模块,以这种形式开发的各种接口可以灵活地适应多种需求。

2. 工作在应用层的拦截和过滤模块(LSP.DLL)

本模块以 Winsock SPI 的一个子模块存在,并以 DLL 形式存在,编程、调试方便。从系统角度来看,LSP.DLL 和 Winsock SPI 处于同一个层次,主要负责拦截应用层的各种封包,支持进程级别的各种过滤,同时能很方便地扩充支持内容过滤,做成更为强大实用的系统。

与内核层的设计不同,应用层的拦截和过滤模块被糅合到了一起,主要是由于应用层对各种平台的支持性较好,不像内核层。应用层的拦截模块不和内核层的拦截模块或者过滤模块有任何直接的关系,但是可以通过用户界面模块把经过它过滤的模块通知到内核模块。

3. 用户界面模块(NetCleaner.EXE)

用户界面模块是一个普通的 Win32 应用程序,也是整个系统的控制台。通过它,可以将整个系统融为一体。本模块负责与应用层拦截(过滤)模块交互、与内核层拦截模块交互,同时可以控制内核层过滤模块的加载、卸载、过滤器规则的更新等等操作。

(二)系统测试

1. 功能测试

测试测试环境为:机器配置是 Celeron D 2.4GHz,768M DDR400,Realtek 8139NIC,Windows XP SP2,其他软件环境略。通过交换机访问 Internet,ICMP 包使用同网段的其他主机 PING 本机。测试用例和测试结果如表 1 所示。

表 1　系统性能测试结果

测试项目	测试结果	测试用例说明
工作模式(过滤/全部通过/全部不通过)	正确	测试用例:无
基于进程的过滤	可以添加新的过滤规则,且当存在某个规则时可以自动应用规则	测试用例: 1. 向规则文件添加新进程 2. 应用已存在进程的规则
基于 IP 的过滤	正确	测试用例: 1. 针对某个 IP 的通信 2. 针对某网段的通信
ICMP 包过滤	正确	测试用例: 1. 对所有主机关闭本机 ping echo 2. 对某个主机关闭本机 ping echo

2. 性能测试

测试目的:比对安装本防火墙前和安装加载一定数量规则的防火墙后的文件传输性能。事实上,基于网络封包过滤的防火墙对所有的网络通信必然是有影响的(比如浏览网页、即时通信工具等等),由于这些动作的实际通信量很小,因此,这里的性能测试主要针对大数据量的网络 I/O 进行的。

测试用例:使用同一个基于 Winsock 的工具传输一定数量的数据,进行 10 次测试,观察其平均速度(对于过滤模式,在用户层内置 30 条进程过滤规则,内核层没有规则)。由于 Internet 环境的不稳定性,

传输在内网进行,测试主机的配置同前,内网交换机使用联想 D-Link DES1024R 10/100 Fast Ethernet Switch。

测试结果表明,本防火墙软件的安装对系统的文件传输性能影响很小,用户在使用过程中不会有网络性能受损的感觉。测试结果如表 2 所示。

表 2　系统性能测试结果

测试项目	安装防火墙前	安装后(全部通过)	安装后(过滤模式)
网络共享传输	约 9Mbps	约 9Mbps	约 9Mbps
FTP 传输	5156Kbps	5004 Kbps	4988 Kbps

系统的功能测试和性能测试结果均表明,设计基本达到要求。

四、结束语

本文研究了目前 Windows 下个人防火墙网络数据包的拦截技术及其不足,并在此基础上提出了一种新的基于应用层和核心层的双重数据包过滤方案,该方案思路简单明确,在理论上可以截获所有通过 Windows 的网络封包,避免了单一网络数据包拦截方案的缺点;同时该方案在内核态使用 HOOK NDIS 的方法来拦截网络数据封包,避免了中间驱动的缺点,并且容易实现软件的自动安装。根据提出的数据包拦截方案本文还设计了一个实验系统,经实验系统的测试表明,新的数据包拦截方案具有很好的有效性和稳定性,此方法是可行的、有效的。

参考文献:

[1]刘鹏远. Windows 下个人防火墙实现技术路线分析[J]. 计算机工程与设计,2008(21):65 - 68.

[2]陈琪,屈光. Windows 单机版防火墙包过滤多种方案比较于实现[J]. 计算机应用与软件,2005(5):114 - 116.

[3]Jones A,Ohlund J. Windows 网络编程技术[M]. 杨合庆,译. 北京:清华大学出版社,2005:102 - 105.

[4]郭艳,苗克坚. Windows 2000 下 WDM 驱动程序的研究与开发[J]. 计算机工程,2006(22):101 - 103.

[5]Jeffrey J. Windows 核心编程[M]. 5 版. 葛子昂,译. 北京:清华大学出版社,2008:563 - 585.

[6]刘慧,蔡皖东,赵熠. 基于 NDIS 的防火墙穿透通信技术研究与实现[J]. 微电子学与计算机,2007(5):34 - 37.

(**责任编辑:**黄　燕　许成安)

Research on the Personal Firewall Based on Windows

SUN Wei[1], JIN Xia[1], WANG Cheng[2], ZHUANG Wei-hua[2]

(1. School of Information, Nanjing Audit University, Nanjing 210029, China;
2. School of Computer and Information Engineering, Hohai University, Nanjing 210098, China)

Abstract: A new method based on Windows kernel mode and user mode is proposed so as to solve the problems of packet capture in personal firewall products. The method is based on Windows Socket 2 SPI and NDIS HOOK. It avoids the defect of single way to capture packet. And theoretically, it can capture all the packets. Experimental results show that experimental system based on the new method of packet capture is effective and stable.

Key words: Windows platform; firewall; packet capture; NDIS HOOK; SPI