

文章编号:1001-9081(2005)07-1529-02

基于 hook 的 Windows 防火墙驱动程序研究与设计

鲜继清¹, 谭 丹², 陈 辉²

(1. 重庆邮电学院 自动化学院, 重庆 400065; 2. 重庆邮电学院 计算机科学与技术学院, 重庆 400065)
(tdcqupt@hotmail.com)

摘 要:在分析介绍 Windows 2000/XP 平台的网络驱动程序的基础上,提出了一种使用 NDIS hook 技术实现防火墙驱动程序的方法。这种方法不同于已有的 API hook 技术,它无需重新启动操作系统就能生效,并增强了抵御网络攻击的能力。由于它工作在网络层,可以对所有进出计算机的数据包进行过滤,因此可以更方便有效地保护用户信息安全。同时提出并设计了一个基于共享内存和事件对象的驱动程序通信模型。分析证明该模型可有效提高驱动程序与应用程序通信的效率。

关键词:防火墙;NDIS 钩子;驱动程序;共享内存;事件对象

中图分类号: TP393.08 **文献标识码:** A

Research and design of Windows firewall driver based on hook

XIAN Ji-qing¹, TAN Dan², CHEN Hui²

(1. College of Automation, Chongqing University of Posts and Telecommunications, Chongqing 400065, China;
2. College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China)

Abstract: The network driver of Windows 2000/XP was analyzed, and a scheme of implementing firewall driver using NDIS (Network Driver Interface Specification) hook was presented. Different from API hook, the method could take effect without reboot and strengthen the ability of resisting intrusion. The driver worked on network layer and filtered all data packets through the computer, so it could protect users' information effectively and conveniently. A driver communication model based on share memory and event object was also provided. The analysis of this model indicates that it can greatly improve the communication efficiency between driver and application.

Key words: firewall; NDIS hook; driver; share memory; event object

0 引言

随着信息技术的飞速发展,信息安全越来越引起人们的重视。一些大中型企业通过购买防火墙、入侵检测系统等安全设备希望保障企业的信息安全。防火墙和入侵检测系统处于网络的边界,虽可以有效保护内部网络免受 Internet 的攻击,却不能对来自网络内部的攻击进行防御^[1]。因此,个人主机防火墙成为企业信息安全不可缺少的重要组成部分。实现个人防火墙的方法可分为应用态和核心态两类,分别以 LSP (Layered Service Provider) 和 TDI (Transport Driver Interface) 过滤驱动程序为代表^[2]。但这两种方法都有其自身的缺点,如 LSP 对于那些通过 TDI 直接发送数据的木马和病毒无能为力,而 TDI 无法拦截 ICMP 数据包^[3]。有的软件防火墙是利用 hook 系统核心函数的方法来实现的^[4],这种修改 PE 文件导出表实现 hook 的方法必须重新启动系统才能生效,不利于开发调试,也给病毒和木马的入侵留下了隐患。本文提出的防火墙驱动程序设计方法克服了前面的这些缺点,并针对驱动程序与应用程序的通信方法进行了改进。

1 网络体系结构及驱动程序简介

1.1 Windows 网络体系结构

微软 Windows 网络体系结构是基于 ISO (International Standards Organization) 开发的七层网络模型。OSI (Open

Systems Interconnection) 参考模型把网络描述成一系列完成特定功能的协议层。每一层向上层提供约定的服务,而隐藏服务的具体实现。在相邻的协议层之间接口定义了底层提供的服务及高层如何访问该服务^[2]。

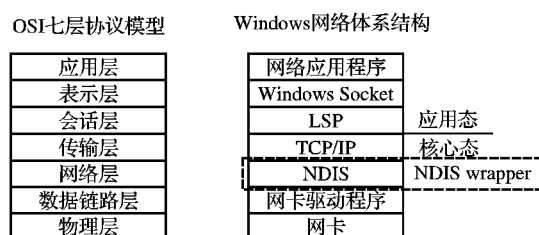


图1 Windows 网络体系结构与 OSI 的映射关系

我们提出的防火墙驱动程序位于 NDIS wrapper, 对应于 OSI 的网络层, 这是过滤网络数据包理想的层次位置。

1.2 防火墙驱动程序简介

驱动程序是操作系统的一个信任部分,运行在系统的内核模式。Windows 网络驱动程序遵循 ISO 的 OSI 标准,从上到下依次是协议驱动 (TCP 协议)、NDIS 中间层驱动、微端口驱动,各层之间的接口由网络驱动程序接口规范 (Network Driver Interface Specification, NDIS) 定义。协议驱动属于 OSI 的传输层,是 NDIS 层和更高层软件抽象层 (例如套接字和 NetBIOS) 之间的接口^[5]。

防火墙驱动程序一般属于协议驱动或 NDIS 驱动程序。

收稿日期:2005-02-07;修订日期:2005-05-08 基金项目:国家 863 计划项目(2003AA412030)

作者简介:鲜继清(1946-),男,四川西充人,副教授,主要研究方向:通信系统与网络;谭丹(1981-),男,湖南浏阳人,硕士研究生,主要研究方向:计算机网络安全;陈辉(1979-),女,新疆乌鲁木齐人,硕士研究生,主要研究方向:网络管理、IPv6 技术。

如图 1 所示,NDIS wrapper 包括网络层及网络层与上下层的接口。把驱动程序挂接到 NDIS wrapper,就可以得到系统所有的发送和接收的数据包。文中提出的防火墙驱动程序就是采用 NDIS 钩子的方式嵌入 NDIS 而实现数据包过滤的。

驱动程序使用一个入口点模型。在这个模型中,每个入口点对应一个特定的函数。在每个入口点,I/O 管理器传递相应的参数给函数,使得它可以完成所要求的功能,这个特定的函数被称为回调例程^[6]。驱动程序使用的基本入口点如下:

DriverEntry 这是驱动程序必需的入口点,当驱动程序被加载时,由内核调用 DriverEntry 例程。因此,适合在这里对设备进行初始化。

Dispatch 例程 该入口点提供给驱动程序所支持的每个主要 I/O 函数。当 I/O 管理器有要处理的请求时,它就在与请求的主要 I/O 函数代码相关的分派入口点调用驱动程序。

Unload 例程 在支持动态卸载的驱动程序中,当要求动态卸载时,I/O 管理器就会调用这个例程做设备和资源的清除工作。

2 改进的基于 NDIS hook 的驱动程序

2.1 已有 hook 技术简介

NDIS hook 就是用自定义的回调例程替换 NDIS wrapper 的相应函数,使得系统调用被截获的一种系统监视机制。在个人防火墙领域,有一种利用 hook 系统核心函数来实现软件防火墙的方法^[3]。这种技术的基本原理如下:操作系统在加载 NDIS 驱动程序时,将 NDIS 协议特征结构表中的 API 函数映射到内存中。通过在内存中定位这些 API 地址,按照 PE 格式将导出表中的函数地址替换成自定义的函数地址,在操作系统调用系统自身 API 函数前,先进行自定义函数的处理,实现对数据包的过滤。使用这种技术实现的驱动程序要求重新启动操作系统后才能生效,给开发调试带来了不便,同时给病毒和木马的入侵留下了隐患。

2.2 改进的 NDIS hook 技术的原理

针对已有的 API hook 技术的缺点,我们提出了一种注册假协议(fake protocol)的方法,它没有重启限制,不仅加快了开发调试,还可以方便防火墙的使用。

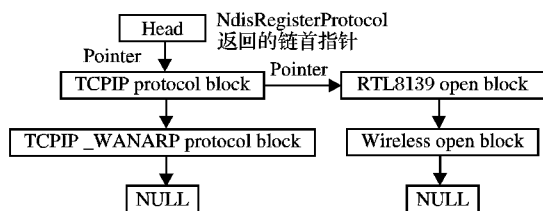


图 2 协议与适配器绑定链表结构

在 Windows NT 系统内核中,所有已注册的网络协议是通过一个单向链表来维护的。链表的节点为 NDIS_PROTOCOL_BLOCK 结构,在这个结构中保存了注册网络协议时所指定的各种信息,如支持协议即插即用的回调函数地址等。并且,每个协议驱动都对应一个 NDIS_OPEN_BLOCK 结构的单向链表来维护其所绑定的适配器信息,协议驱动发送和接收数据包的回调函数地址就保存在这个结构中,只需替换这些地址就可以控制数据包的发送和接收。协议与适配器绑定链表结构如图 2 所示。当驱动程序调用 NdisRegisterProtocol 注册新的协议时,NDIS 总是会把新注册的协议插入协议链表的表头并返回链首指针,这样只要注册一个新的协议,就可以通过这个返回的链表指针遍历系统中所有已注册的网络协议,进

而挂接数据包的发送和接收函数。

2.3 hook 的具体实现

根据上述分析可知,实现 NDIS 钩子是遍历和处理协议及其绑定适配器的过程。对协议进行挂接时,我们关心的是对适配器的 PNP,如 BindAdapter 和 UnbindAdapter 函数。而发送和接收数据包的函数是在处理适配器时挂接的。具体代码如下所示,其中省略了部分变量定义和错误处理:

```

NDIS_HANDLE hProtocol = NULL;
PNDIS_PROTOCOL_BLOCK pProtocolBlock = NULL;
PNDIS_OPEN_BLOCK pOpenBlock = NULL;
// 注册一个假协议
NdisRegisterProtocol(&Status, &hProtocol, &PROTOCHAR,
    sizeof(NDIS_PROTOCOL_CHARACTERISTICS));
// 略过注册的假协议
pProtocolBlock = ((PNDIS_PROTOCOL_BLOCK) hProtocol) ->
    NextProtocol;
while (pProtocolBlock)
{
    // 挂接协议的 PNP 相关例程
    bHookResult = HookProtocolBlock(pProtocolBlock);
    if (bHookResult)
    {
        pOpenBlock = pProtocolBlock -> OpenQueue;
        while (pOpenBlock)
        {
            // 挂接该适配器的发送和接收函数
            HookAdapterBlock(pOpenBlock);
            pOpenBlock = pOpenBlock -> ProtocolNextOpen;
        }
        pProtocolBlock = pProtocolBlock -> NextProtocol;
    }
}
// 注销开始注册的假协议
NdisDeregisterProtocol(&Status, hProtocol);
  
```

3 改进的防火墙通信模型

防火墙系统的用户界面程序可以使用 DeviceIoControl 把数据发送到底层驱动程序,但底层驱动却不能把数据直接传给上层。实际上,防火墙驱动程序往往需要向上层应用程序报告日志和告警等信息,这些操作除对实时性的要求外,还要求稳定性好。Windows 推荐的标准办法是:驱动程序为了防止数据丢失,把数据缓存到一个队列,然后用事件通知应用程序,应用程序得到事件通知后调用 DeviceIoControl 系统调用从底层队列读取数据^[2]。这种方案需要在用户态和核心态之间切换,由此增加的时延势必导致防火墙性能下降,并增加了队列维护带来的风险。

我们提出的防火墙通信模型是一个基于事件和共享内存的应用方案。其实现步骤如下:

1) 在防火墙初始时由应用程序分配一段内存,并通过 DeviceIoControl 将该内存地址及大小传递给驱动程序。

2) 驱动程序在分发例程中调用 IoAllocateMdl 把内存块映射到 MDL (Memory Descriptor List), 然后使用 MmProbeAndLockPages 将该块内存锁定为常驻内存。

3) 驱动通过 MmGetSystemAddressForMdlSafe 取得该内存块在系统空间的虚拟地址。这样,用户程序和驱动程序各自的虚拟地址其实被映射到相同的硬件地址,该内存区域成为它们的共享内存。

(下转第 1534 页)

节参数(使汇聚流 $\geq L_{R0}$), r_{arrive} 为到达速率。限流调节只在那些到达速率超过限流调节值的 r_{adj} 路由器上实施漏桶式均速限流。算法概要如下:

```

 $r_{adj} = (L_{R0} + U_{R0}) / f;$ 
while (1)
    if  $r_{arrive} > U_{R0}$  (需要进行限流)
         $r_{adj} = r_{adj} / f$ 
    elif  $r_{adj} < L_{R0}$  (限流过强)
         $r_{adj} = r_{adj} + \alpha$ 
    else
        break
    fi
end while

```

例如 $[L_{R0}, U_{R0}]$ 为 $[19, 22]$, $f = 2$, $\alpha = 1$ 。分布式限流调节与 $R(0)$ 到达速率的跟踪如表 1 所示。

表 1 分布式限流调节与 $R(0)$ 到达速率的跟踪

调节 轮次	r_{adj}	$R(4)$ 调节	$R(5)$ 保持	$R(6)$ 调节	$R(7)$ 调节	$R(3)$ 保持	$R(0)$ 到达速率
0	20.5	22.8	0.25	14.37	18.57	0.83	56.82
1	10.25	10.25	0.25	10.25	10.25	0.83	31.83
2	5.125	5.125	0.25	5.125	5.125	0.83	16.455
3	6.125	6.125	0.25	6.125	6.125	0.83	19.455

3.4 回推反馈与回推刷新

在上游那些响应回推请求的路由器,它们向下游路由器发送回推状态信息。状态信息报告相关汇聚流的总到达速率,可帮助下游的汇聚流拥塞控制路由器决定是否继续使用速率限制及回推。

上游停止限流的前提应该是保证下游汇聚流拥塞控制的路由器上汇聚流的总到达速率估计值处于该路由器接收速率的上限之下。因为上游限流的汇聚流可能会包含一些服从端对端控制的流量,终止限流可能会导致相关汇聚流的到达速率更大。

汇聚流的总到达速率估计值包括:来自上游路由器的回推状态信息,以及上游非汇聚流量份额的链路流量。如图 1 所示,链路上所标的值为回推状态信息中包含的到达速率估计值,没有收到回推请求的节点所连接链路,它所输送的汇聚流量是由汇聚

流拥塞控制路由器估计的。因此,汇聚流拥塞控制路由器是否终止上游的限流取决于总汇聚流在本地的控制结果。

由于回推采用请求/响应模式,所以在没有收到下游路由器刷新信息的情况下,上游路由器的限速工作就会停止。在更新限制速率时,下游路由器根据上游发来的状态信息来估计到达速率,利用计算汇聚流限速算法重新排序最高流量汇聚流并重新计算汇聚流的限制速率。回推状态信息里所报告的到达速率还可以为下游路由器决定上游路由器间新的带宽划分。

4 结语

根据 DDoS 攻击流的高流量汇聚流特性,结合 IP 拥塞控制技术,有区分地控制汇聚流聚积处的可疑过载流量。通过 AID 的过程构建汇聚流源反向往返树,并对高流量汇聚流向源头进行流量回推,不仅减缓了受害网络边缘的瓶颈压力,同时释放了上游的带宽资源,从而在一定程度上实现了对 DDoS 的防御,比较有效地解决了在 DDoS 的攻击下正常流仍然能够比较好地传输的问题。

参考文献:

- [1] MIRKOVIC J, REIHER P. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms [J]. SIGCOMM Computer Communication, 2004, 34(2): 39-53.
- [2] IOANNIDIS J, BELLOVIN S. Implementing Pushback: Router-Based Defense Against DDoS Attacks [A]. Proceedings of NDSS'02 [C], 2002.
- [3] MAHAJAN R, BELLOVIN S, FLOYD S, et al. Controlling high bandwidth aggregates in the network (Extended Version) [J]. ACM SIGCOMM Computer Communication, 2002, 32(3).
- [4] FLOYD S, JACOBSON V. Random early detection gateways for congestion avoidance [J]. IEEE/ACM Transactions on Networking, 1993, 1(4): 397-413.
- [5] WANG B-T. Tracing High Bandwidth Aggregates [A]. Proceedings of IASTED International Conference on Communication, Network, and Information Security (CNIS) [C], 2003. 165-170.
- [6] YAU D, LUI J, LIANG F, et al. Defending Against Distributed Denial of Service Attacks with Max-min Fair Server-centric Router Throttles [A]. Proceedings of the Tenth IEEE International Workshop on Quality of Service (IWQoS) [C], 2002. 35-44.

(上接第 1530 页)

4) 驱动部分要向上层发送数据时,只需把数据写到共享内存区,然后设置事件状态来通知应用程序。这时,应用程序从共享内存得到的就是驱动程序要传递的数据。

5) 当通信结束时,取得需调用 MmUnlockPages 解除对共享内存的锁定,并释放 MDL。

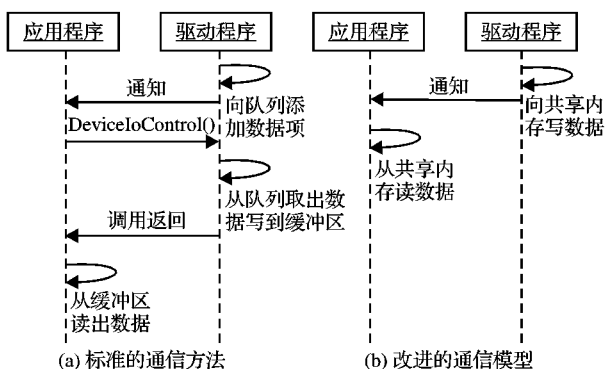


图 3 改进的通信方法与标准的通信模型^[2]的差异

从图 3 中可以看出改进的通信模型与 Windows 推荐的标准解决办法的差异。改进的模型减少了系统在用户态和核心态的切换次数,从而缩短了系统时延。同时还省略了队列的维护操作,有效提高了整体通信效率。经实践证明,基于这种方法构建的 Windows 防火墙驱动程序能稳定高效地运行。

参考文献:

- [1] 张世永. 网络安全原理与应用 [M]. 北京: 科学出版社, 2003.
- [2] Microsoft Windows 2000 DDK Help [DB/OL], 2004.
- [3] <http://www.ntkernel.com> [EB/OL], 2004.
- [4] 周剑岚, 冯珊. 运用 Hook 技术实现的软件防火墙 [J]. 华中科技大学学报 (自然科学版), 2004, 32(3): 83-85.
- [5] CANT C. Windows WDM 设备驱动程序开发指南 [M]. 孙义, 马莉波, 国雪飞, 等译. 北京: 机械工业出版社, 2000.
- [6] BAKER A, LOZANO J. Windows 2000 设备驱动程序设计指南 [M]. 施诺, 等译. 北京: 机械工业出版社, 2001.