广西科学院学报
Journal of Guangxi Academy of Sciences
2006,22(4):293～297
Vol. 22,No. 4 November 2006

内网安全监控系统中 NDIS 中间层驱动技术的实现*

Implementation of NDIS Intermediate Driver Technique in the Security Monitoring System of Intranet

李陶深,蔡世平,严 毅,黄国石,曾 亮,黄顶源
LI Tao-shen, CAI Shi-ping, YAN Yi, HUANG Guo-shi, ZENG Liang, HUANG Ding-yuan

(广西大学计算机与电子信息学院,广西南宁 530004)
(School of Computer, Electronics and Information, Guangxi University, Nanning, Guangxi, 530004, China)

摘要:阐述在 Windows 2000/XP/2003 的核心驱动层上采用 NDIS(Network Driver Interface Specification)中间层驱动技术对数据包进行拦截和分析的关键技术,并以实例验证该技术的可行性。

关键词:中间层驱动 内网 数据 NIDS
中图法分类号:TP393.07 **文献标识码**:A **文章编号**:1002-7378(2006)04-0293-05

Abstract: The key technique of the interception and analysis of data packages using Network Driver Interface Spcification in the core driver level of Windows 2000/XP/2003 is explained. The feasibility of this technique is proved in a sample.

Key words: intermediate driver, intranet, data package, NDIS

内网安全监管系统重点解决局域网内部系统行为安全和网络行为安全。系统行为安全针对数据文件的操作,主要是对外设接口的控制,以监控数据的外流。网络行为安全针对局域网中计算机的网络操作,主要是对访问网站、访问局域网内特定主机和收发邮件等行为实施监视并加以控制,监测和防御网络的恶意访问、泄密和攻击等行为,并且对内网中的计算机资源进行有效地管理和控制。

目前,监控内网机群的数据流动实现网络行为监视和管理最先进的是采用中间层驱动技术对各种数据包进行拦截和分析[1,2]。本文阐述在 Windows 2000/XP/2003 的核心驱动层上采用 NDIS (Network Driver Interface Specification)中间层驱动技术对数据包进行拦截和分析的关键技术。

1 NDIS 中间层驱动技术原理

NDIS 是为传输层提供了标准的网络接口。在 Windows 操作系统中,所有的应用程序最终都是通过调用 NDIS 接口实现对网络的访问[1]。

中间层驱动程序是一种典型的层次结构程序,它基于一个或多个 NDIS NIC 驱动程序,其上层是一个能提供 TDI(传输驱动程序接口)支持的传输驱动程序(也可能是多层结构)。中间层驱动程序的一个示例是 LAN 仿真中间层驱动程序,其上层是一个早期传输驱动程序,下层是一个非 LAN 介质的微端口 NIC 驱动程序。另一个中间层驱动程序的例子是 ATM LANE (LAN 仿真)驱动程序,它将数据包从上层无连接的传输格式转换为下层面向连接的网卡支持的 ATM 格式。

中间层驱动程序通过调用 NDIS 打开和建立一个对低层 NIC 驱动程序或者 NDIS 中间层驱动程序的绑定。中间层驱动程序提供 MiniportSet —

收稿日期:2006-07-08
作者简介:李陶深(1957-),广西邕宁人,教授,主要从事网络安全、网络路由、分布式数据库方面的研究。
* 本文得到广西留学回国人员科学基金项目(桂科回 0342001)和广西电子信息应用项目(桂电办 2004-17 号)联合资助。

Information 和 MiniportQueryInformation 函数来处理高层驱动程序的设置和查询请求。某些情况下,可能还要将这些请求向低层 NDIS 驱动程序进行传递。如果其下边界是面向无连接的,则可通过调用 NidsRequest 实现这一功能;如果其下边界是面向连接的,则通过调用 NidsCoRequest 实现该功能。

中间层驱动程序通过调用 NDIS 提供的函数向网络低层 NDIS 驱动程序发送数据包。例如,下边界面向无连接的中间层驱动程序必须调用 NdisSend 或 NdisSendPackets 来发送数据包或者包数组,而在下边界面向连接的情况下就必须调用 NdisCoSendPackets 来发送包数组数据包。如果中间层驱动程序是基于非 NDIS NIC 驱动程序的,那么在调用中间层驱动程序的 MiniportSend 或 Miniport(Co)SendPackets 函数之后,发送接口对 NDIS 将是不透明的。

NDIS 提供了一组隐藏低层操作系统细节的 NdisXxx 函数和宏。例如,中间层驱动程序可以调用 NdisMInitializeTimer 来创建同步时钟,可以调用 NdisInitializeListHead 创建链表。中间层驱动程序使用符合 NDIS 标准的函数,来提高其在支持 Win32 接口的微软操作系统上的可移植性。

2 实现 NDIS 中间层驱动的关键技术

内网安全监管系统的 NIDS 中间层驱动是在 VC++6.0、DriverStudio2.7 和 Windows 2000 DDK 开发包的开发环境下实现的。ntDeviceName 为 NT 下的系统设备名,win32DeviceName 为 Dos 下的系统设备名,MajorFunction 为响应函数数组。

2.1 注册设备技术

把 NIDS 中间层驱动注册成为系统设备,用户应用程序就可以像操作文件一样操作 NIDS 中间层驱动。方法如下:

```
Status = NdisMRegisterDevice(*KNdisMiniDriver::DriverInstance(),&ntDeviceName,&win32DeviceName,MajorFunction,&m_pDeviceObject,&m_DeviceHandle);
//交互方式为 buffered 方式
m_pDeviceObject->Flags |= DO_BUFFERED_IO;
```

2.2 响应函数 IoDispatch

响应函数用于对用户应用程序的操作进行响应。用户应用程序每进行一种操作,都会发送一个 IRP 包即请求包,驱动程序通过 IRP 包的参数可知用户应用程序执行了何种操作,从而做出相应的响应。用户应用程序执行 CreateFile 操作时,I.MajorFunction() = IRP_MJ_CREATE。执行 WriteFile 操作时,I.MajorFunction() = IRP_MJ_WRITE,驱动程序则调用 GetRule 函数来取得用户程序传递过来的规则数据。执行 ReadFile 操作时,驱动程序则调用 ReadLogData 函数将记录的日志信息数据写入系统缓冲区,让用户应用程序取走。用户应用程序与驱动程序交互方式如图1所示。具体方法如下:

```
NTSTATUS IoDispatch(IN PDEVICE_OBJECT DeviceObject,IN PIRP Irp)
{ //响应函数
  KIrp I(Irp);
  //得到 Irp 对象栈
  PIO_STACK_LOCATION IrpStack =
      IoGetCurrentIrpStackLocation(Irp);
  switch(I.MajorFunction())  {
      case IRP_MJ_CREATE:
         // 如果有进程调用了 CreateFile
      case IRP_MJ_CLOSE:
         // 如果有进程调用了 CloseFile
      case IRP_MJ_CLEANUP:
      case IRP_MJ_READ:
      SendLog(I);
        ret = true;
        break;
      case IRP_MJ_WRITE:
          GetRule (I, IrpStack -> Parameters.Write.Length);
        ret = true;
        break;
      case IRP_MJ_DEVICE_CONTROL:
        default:
  };
};
```

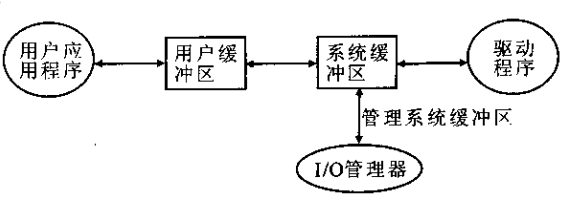

图1 用户应用程序与驱动程序交互方式

2.3 分析策略规则模块

这里,规则定义为:规则头部 + 规则体。分析

策略规则模块编写如下:

```
void SetRule(char * RuleStr) //设置规则
{
    char * RuleHeader;
    char * RuleBody;
    RuleBody=strstr(RuleStr,"$$$");
    RuleBody+=3;
    RuleHeader=RuleStr;
    int RuleLen=strlen(RuleBody);//规则体长度
    if(! strncmp("FilterIp",RuleHeader,strlen("FilterIp")))
    {
      if(FilterIp! =NULL)    delete []FilterIp;
      FilterIp=new char[RuleLen];
      strcpy(FilterIp,RuleBody);
      DbgPrint("FiterIp=%s\n",FilterIp);
    }
    else  if(! strncmp("AlarmIp",RuleHeader,
      strlen("AlarmIp")))  {
      if(AlarmIp! =NULL)  delete []AlarmIp;
      AlarmIp=new char[RuleLen];
      strcpy(AlarmIp,RuleBody);
      DbgPrint("AlarmIp=%s\n",AlarmIp);
    }
   else  if(! strncmp("FilterPort",RuleHeader,
            strlen(" FilterPort")))  {
      if(FilterPort! =NULL)  elete []FilterPort;
      FilterPort=new char[RuleLen];
      rcpy(FilterPort,RuleBody);
      DbgPrint("FilterPort=%s\n",FilterPort);
   }
   else  if(! strncmp("AlarmPort",RuleHeader,
            strlen("AlarmPort")))  {
       if (AlarmPort! = NULL)    delete [] AlarmPort;
      AlarmPort=new char[RuleLen];
      strcpy(AlarmPort,RuleBody);
      DbgPrint("AlarmPort=%s\n",AlarmPort);
   }
   else  if(! strncmp("FilterUrl",RuleHeader,
            strlen("FilterUrl")))  {
      if(FilterUrl! =NULL)  delete []FilterUrl;
      FilterUrl=new char[RuleLen];
      strcpy(FilterUrl,RuleBody);
      DbgPrint("FilterUrl=%s\n",FilterUrl);
   }
   else  if(! strncmp("AlarmUrl",RuleHeader,
            strlen("AlarmUrl")))  {
      if (AlarmUrl! = NULL)    delete [] AlarmUrl;
      AlarmUrl=new char[RuleLen];
      strcpy(AlarmUrl,RuleBody);
      DbgPrint("AlarmUrl=%s\n",AlarmUrl);
   }
   else  if(! strncmp("ServerIp",RuleHeader,
            strlen("ServerIp")))  {
      ServerIp=(ULONG)atol(RuleBody);
      DbgPrint("ServerIp=%d\n",ServerIp);
   }
   else  if(! strncmp("ClientIp",RuleHeader,
            strlen("ClientIp")))  {
      ClientIp=(ULONG)atol(RuleBody);
      DbgPrint("ClientIp=%d\n",ClientIp);
   }
}
```

2.4 数据包的拦截技术

要拦截 Windows 下的网络数据包可在两个层面进行[2]:用户态(user-mode)和核心态(kernel-mode)。如果在用户态下进行数据包拦截,其致命的缺点是拦截操作只能在 Winsock 层次上进行,而对于网络协议栈中底层协议的数据包无法进行处理。对于一些木马和病毒来说很容易避开这个层次的防火墙。在核心态下进行数据包拦截可利用网络驱动程序来实现,为此,需要开发一个过滤驱动来截获这些交互的接口,以实现网络数据包的拦截。

DriverStudio2.7 **开发包提供了** OnReceive **和** OnSend **两个十分方便的虚函数[3]。其中,物理网卡接收封包之后,** NDIS **将调用** OnReceive **虚函数;通过物理网卡发送封包之前,** NDIS **将调用** OnSend **虚函数。因此,只需在自开发的中间层驱动程序中实现这两个虚函数,加入自定义的封包处理方法,就可方便地实现网络封包截获处理功能。变量** Original **包含数据包的缓冲区首址及长度信息。方法如下:**

```
NDIS_STATUS StlFilterAdapter::OnReceive(
const KNdisPacket&. Original, KNdisPacket&. Repackaged)
{
   if (! FilterPacket(Original, "接收"))
```

```
    //数据包处理
      return NDIS_STATUS_NOT_ACCEPTED;
      //过滤
    Repackaged.CloneUp(Original);
    return NDIS_STATUS_SUCCESS;
    //放行或报警
}
NDIS_STATUS StlFilterAdapter::OnSend(const
KNdisPacket&. Original, KNdisPacket&.
Repackaged)
{
    if (! FilterPacket(Original, "发送"))
    //数据包处理
      return NDIS_STATUS_NOT_ACCEPTED;
      //过滤
    Repackaged.CloneDown(Original);
    return NDIS_STATUS_SUCCESS;
    //放行或报警
}
```

3 应用实例

利用 NDIS 中间层驱动的关键技术，我们设计实现了一个基于 Windows 系统环境的内网安全监管系统(图 2)。系统的策略设置包括 IP 访问(图 3)、URL 访问(图 4)、端口访问(图 5)、邮件访问(图 6)、网络流量等，可满足系统监控的需要。从图 2～6 可以看出，各种策略的设置非常方便用户使用。无论是一般的用户还是专业用户，都可以很容易地学会使用这些策略设置。

从系统的实际运行情况来看，采用 NDIS 中间层驱动技术来实现数据包的拦截与分析技术是可行的。

图 2 内网安全监管系统的主界面

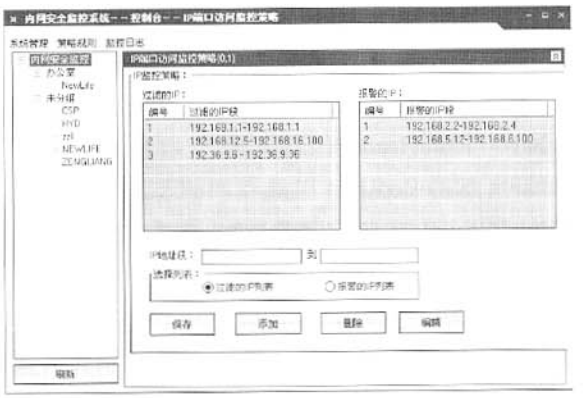

图 3 IP 访问策略

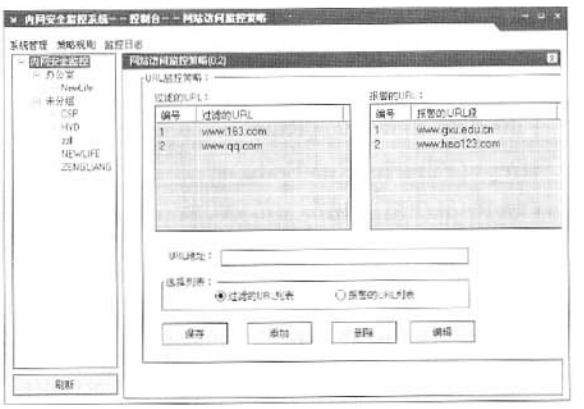

图 4 URL 访问策略

图 5 端口访问策略

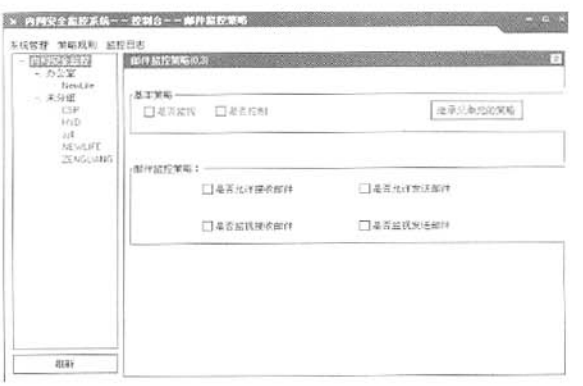

图 6 邮件访问策略

4 结束语

本文介绍的 NDIS 中间层驱动技术已经在内网安全监管系统的开发中得到应用。应用该技术开发的系统能应用于 Windows2000/XP/2003 系统中,且基本上能应用于各种以太网局域网中,能对 IP 数据包进行拦截和分析,得到其日志信息。以后还可以根据需要进行功能扩展,使其能对其它类型数据包进行拦截和分析。

参考文献:

[1] CHRIS CANT. Windows WDM 驱动程序开发指南[M]. 北京:机械出版社,2000.

[2] 谭思亮. 监听与隐藏—网络侦听揭密与数据保护技术[M]. 北京:人民邮电出版社,2002.

[3] 汪晓平,刘韬. 开发网络经典示例导航[M]. 北京:人民邮电出版社,2005.

(责任编辑:邓大玉 凌汉恩)

(上接第 288 页)

从表 3 可以看出,网络延时值较小,网络延时不到 1ms,说明网络延时很小,网络工作状态良好。表 4 结果显示,网络的日常运行正常,能够用于实际。

表 3 网络延时记录

Source Adderss	Dest Address	Summary	Len	Rel time	Delta Time
1E111.1	113.00400-565890	NCP:ROK	566	0:00:06 383	0.000.423
113.000021 D3E63	1E111.1	NCP: C Get current size of file F=DC4F0300	60	0:00:06 383	0.000.068
1E111.1	113.0050 BA72CD1	NCP:ROK	566	0:00:06 384	0.000.425

表 4 服务器运行状态记录结果

测试项目	结果
CPU 利用率	一般 10%~30%
CPU 利用率分布情况	IPX RTRNCP Work to do 0~15% Internupt 5%~15%
内存占用和空闲情况	
Current Srevice processes	<50
Dirty cache buffers	<500
Current disk requests	<51
Long term cache hits	100%
Long term cache dirty hits	100%
LRU sitting time	>1h
Cache buffers	>2000
工作站上网情况	
钱龙上网速度	正常
钱龙 81 速度	<2s
成交回报速度	<5s
与服务器、交换机相连端口错包率、冲突率	无

4 结束语

综上所述,证券公司计算机局域网络的三层结构对提高证券行情、交易的效率以及防范黑客攻击是非常好的。但是,中间件运行的操作系统 Windows NT Server 本身存在不少安全漏洞,加之针对 NT 的黑客工具较多,并不能百分之百地保证网络系统无坚不摧。所以,如何提高中间件自身乃至整个网络的安全性,仍然是我们证券行业计算机技术人员今后研究的重要课题。

参考文献:

[1] 常晓波,杨剑峰. 安全体系结构的设计部署与操作[M]. 北京:清华大学出版社,2003.

[2] 戴英侠,连一峰,王航. 系统安全与入侵检测[M]. 北京:清华大学出版社,2002.

[3] VITO AMATO. 思科网络技术学院教程:上册[M]. 北京:人民邮电出版社,2000.

[4] 方程. 操作系统—— Windows2000[M]. 北京:高等教育出版社,2003.

(责任编辑:邓大玉)