第29卷 第1期 昆明理工大学学报(理工版) Vol.29 No.1
2004年2月 Journal of Kunming University of Science and Technology (Science and Technology) Feb. 2004

基于NDIS网络数据包过滤技术研究

严 馨
(昆明理工大学 信息工程与自动化学院,云南 昆明 650051)

摘要:介绍了Windows操作系统的设备驱动程序的结构,着重介绍网络NDIS中间层接口的原理,论述了如何通过NDIS实现TCP/IP网络数据包的截获技术,通过对截获的数据进行分析,可以非常方便地在此基础上做加密、过滤等附加操作.

关键词:网络安全; TCP/IP; NDIS; DDK;数据包过滤;设备驱动程序
中图分类号:TP393.09 **文献标识码**:A **文章编号**:1007-855X(2004)01-0036-05

Research on the Technology of Netware Data Package Filter Based on NDIS

YAN Xin
(Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650051, China)

Abstract: The device driver program structure of Windows and the mesosphere interface principle of netware NDIS are introduced and the TCP/IP netware data package interception technology by NDIS is discussed. Furthermore, the encryption and filter can be realized conveniently by analysing the intercepted data.

Key words: network safety; TCP/IP; NDIS; DDK; packet filtering; device driver

0 引 言

随着计算机应用的发展和深入,计算机在人们的生活和工作中占有越来越重要的地位,促进了计算机网络快速发展,特别是Internet在全球的蓬勃发展,网络安全问题日益严重,重要的信息和数据被窃取,计算机资源被破坏等,一旦网络安全问题发生,可能会带来非常严重的后果.

为了有效地防范网络攻击,我们必须采取行之有效的措施,保障系统安全. Windows下的安全产品基本上是基于对数据报的拦截技术实现的.当然在具体的实现方式上它们却有很大的不同.总的来说可分为用户级和内核级数据报拦截两类.其中内核级主要是TDI过滤驱动程序,NDIS中间层过滤驱动程序,NDIS过滤钩子驱动程序等,它们都是利用网络驱动来实现的;而用户级的过滤包括SPI接口,Winsok API接口的HOOK.

本文主要论述基于NDIS中间层的数据包过滤实现.

1 过滤驱动程序的特点

开发设备驱动程序需要开发工具DDK(Device Driver Kit),设备驱动程序工作在Windows的核心层,它可以由CPU完全控制,并可以突破Windows应用程序保护机制,可以完成非常底层的操作.

驱动程序有硬件和软件驱动程序之分,软件驱动程序位于硬件驱动程序之上,可以提供分层服务,不同版本系统不能兼容.

过滤驱动程序通过创建一个或多个设备对象(DeviceObject)直接挂接到一个现有的驱动程序之上.当有应用程序或其他驱动程序调用这个对象时,会首先映射到过滤驱动程序上,然后由过滤驱动程序再传递给原来的设备对象.由于过滤驱动程序挂接的透明性,所有上层的应用程序或驱动程序并看不到过滤驱动程序,实际上已经被过滤了.像DLL一样,驱动程序必须有入口函数DriverEntry.

收稿日期:2003-05-19.
作者简介:严馨(1969.2~),女,硕士,讲师.主要研究方向:网络与数据库技术.

Windows 98 DDK 提供了一个 Hook _ Device _ Service 的服务,利用这个服务可以轻松地用自己的函数地址替换原来的函数地址,并返回原来的函数地址.但是这需要汇编代码,VToolsD 为我们提供了一个更简单的函数 Hook _ Device _ Service _ C,这个函数为 C 语言原型,可以使用 C 语言直接调用,而且使用简单,函数原型如下:

PVOID Hook _ Device _ Service _ C(DWORD Service, PVOID pHandler, HDSC _ Thunk * pThunk);

参数:

Service: 要 Hook 的虚拟设备或者 VMM 服务,即函数编号,比如:__PELDR _ AddExportTable.

pHandler: 我们自己的用来 Hook 的函数句柄,这个函数的参数必须与要 Hook 的系统函数参数相同.

pThunk: 指向一个锁定的内存块,用来建立呼叫 Hook _ Device _ Service 的堆栈代码数据.这个参数一定不能为空,需要为每一个 Hook 的函数建立一个 HDSC _ Thunk 类型的全局变量,不能多个函数共用一个 HDSC _ Thunk.

返回值:

如果 HOOK 成功,返回 HOOK 的系统服务函数的地址;如果失败,返回 0.

说明:

如果需要卸载钩子函数,可以使用 Unhook _ Device _ Service _ C 函数.

BOOL Unhook _ Device _ Service _ C(DWORD Service, HDSC _ Thunk * pThunk);

参数:

Service: 要解除 Hook 的虚拟设备或者 VMM 服务,即函数编号,比如:__PELDR _ AddExportTable.

pThunk:指向通过 Hook _ Device _ Service _ C Hook 函数时使用的 HDSC _ Thunk 指针.

返回值:

TRUE: 成功;FALSE: 失败

Windows NT/2000 没有像 Hook _ Device _ Service 这样的 Hook 函数,那如何对系统函数进行 Hook 呢? 其实无论是代码也好,程序也好,被操作系统管理起来都是一堆数据,关键问题是如何从这么一大堆数据中找到我们所需要的,并明白这些数据的作用和调用的原理,就可以按需求对其进行修改. 微软总是有所保留,对系统核心看得很紧.不过可以用没有公开的函数和数据结构,这些知识对于我们做系统钩子就显得非常重要.接下来举例说明如何 Hook 系统核心函数.这里所说的核心函数,指的是在驱动程序中提供的服务函数,许多人称它为核心层的 API.在应用层,API 的提供者往往是 DLL,在核心层就是 SYS.SYS 提供两种接口方式,一种是通过 IRP,这是微软推崇的一种方式;另外一种是通过导出表直接导出函数,供其它程序调用,这种方式与 DLL 没什么两样.SYS 遵循标准的 PE 格式,而对于 PE 格式论述的文章和图书很容易找到,这里不再论述.操作系统在加载 SYS 程序时,会将整个 SYS 文件映射到内存的一个虚拟地址空间,所以只要我们能在内存中定位到这个地址,就可以按照 PE 格式找到它的导出表,然后将导出表中的函数地址换成我们自己的函数地址,就完成了 HOOK.需要注意的是,导出表里保存的是函数的偏移地址,而不是绝对地址,所以替换时一定要用自己的函数地址减掉模块的基地址,这样得到的就是偏移地址,后面我们会举例说明如何进行偏移地址的转换.下面我们首先来实现第一步:如何得到 SYS 在内存中的地址? Native Api 提供了一个非常重要的函数 ZwQuerySystemInformation,这个函数可以轻而易举的得到许多系统核心数据.它在 NtDll.dll 中,可以通过 DumpBin 工具查看.DDK 中没有提供关于这个函数的文档,但是 DDK 的 Lib 库中却包含这个函数,所以,只要我们按照正确的函数定义声明它,就可以编译通过并正常使用.ZwQuerySystemInformation 的定义如下:

NTSYSAPI NTSTATUS NTAPI ZwQuerySystemInformation(ULONG sic,PVOID pData, ULONG dSize, PULONG pdSize);

参数:

sic:标识向查询的系统信息的类型,比如:11 就是得到模块信息的标识,详细介绍请参阅 Native Api 的书籍.

pData:用来返回数据的缓冲区指针　dSize:当用来返回系统信息数据时,pData 缓冲区的大小.

pdSize:用来返回系统信息实际需要的长度.

返回值:

STATUS _ SUCCESS:成功

其它一些错误代码:失败

利用代码 11 我们就可以得到系统加载的模块信息,正是我们需要的 SYS 的地址信息.得到的模块信息结构如下:

```
typedef struct _SYSTEM _ MODULE _ INFORMATION
{
  ULONG Reserved[2];
  PVOID BaseAddress;
  ULONG Size;
  ULONG Flags;
  USHORT Index;
  USHORT Rank;
  USHORT LoadCount;
  USHORT NameOffset;
  CHAR Name [256];
} SYSTEM _ MODULE _ INFORMATION, * PSYSTEM _ MODULE _ INFORMATION;
```

关于这个结构详细的介绍,请参阅 Native Api.我们这里只关心两个字段,一个是 Name,也就是模块的名称,比如:Ndis.sys,可以利用这个字段确定出是否是我们感兴趣的模块.另外一个就是 BaseAddress,这个字段保存的地址就是模块的基地址,正是我们需要的.这个地址是位于系统地址空间的虚拟地址,必须使用驱动程序才能操作这个地址空间,所以完成系统核心函数 Hook 的任务一般使用驱动程序.得到了 NDIS.SYS 的基地址,接下来是如何分析 PE 格式定位到在导出表中我们需要 HOOK 的函数的地址.PE 格式的数据结构定义在 Winnt.h 中.

2 NDIS 中间驱动程序描述

2.1 NDIS 介绍

NDIS(Network Driver Interface Specification)即网络驱动程序接口规范,NDIS 为传输层提供标准网络接口,所有的传输层驱动程序都需要调用 NDIS 接口来访问网络,NDIS 支持三种类型驱动程序:

·Miniport drivers ,通过 NDIS 接口完成对网卡的操作,同时开放 Miniport 接口供上层驱动调用.

·Intermediate drivers ,位于 Miniport 和 Protocol 驱动之间,同时具有 Miniport 和 Protocol 两种驱动程序接口.其实中间驱动程序在自己的上下两端分别开放出一个 Miniport 接口和一个 Protocol 接口,其中上面的 Miniport 接口同上层驱动程序的 Protocol 接口进行对接;下面的 Protocol 接口同底层驱动程序 Miniport 接口进行对接.这样中间驱动程序在两个驱动之间插入一层.利用 NDIS 中间驱动程序可以在网卡驱动程序与传输驱动程序之间插入一层自己的处理,从而可以用来截获网络封包并从新进行封包、加密、网络地址转换及过滤等操作.

·Protocol drivers ,开放 Protocol 接口供底层驱动调用,实现协议驱动.

2.2 NDIS 封包结构

NDIS 接口中,封包结构通常用 NDIS _ PACKET 来表示.而对于不同的 NDIS 版本,NDIS _ PACKET 的结构内容又略有不同,所以将 NDIS _ PACKET 分为 95/98/me 和 2000 两种情况阐述. NDIS _ PACKET 在 DDK 中的定义:

```
typedef struct _NDIS _ PACKET {
  NDIS _ PACKET _ PRIVATE Private;
  union {
```

```
    struct {
      UCHAR WidgetReserved[8];
      UCHAR WrapperReserved[8];
    };
    struct {
      UCHAR MacReserved[16];
    };
  };
  UINT Signature;
  UCHAR ProtocolReserved[1];
} NDIS_PACKET, * PNDIS_PACKET;
```

NDIS_PACKET_PRIVATE 定义:

```
typedef struct _NDIS_PACKET_PRIVATE {
  UINT PhysicalCount; // number of physical pages in packet.
  UINT TotalLength; // Total amount of data in the packet.
  PNDIS_BUFFER Head; // first buffer in the chain
  PNDIS_BUFFER Tail; // last buffer in the chain
  PNDIS_PACKET_POOL Pool; // so we know where to free it back to
  UINT Count;
  ULONG Flags;
  UCHAR Reserved[8]; // for future expansion
} NDIS_PACKET_PRIVATE, * PNDIS_PACKET_PRIVATE;
```

.NDIS_BUFFER 定义:

```
typedef struct _NDIS_BUFFER {
  struct _NDIS_BUFFER * Next;
  PVOID VirtualAddress;
  PNDIS_BUFFER_POOL Pool;
  UINT Length;
  UINT Signature;
} NDIS_BUFFER, * PNDIS_BUFFER;
```

通过上面结构的定义可以看出 NDIS_PACKET 中保存这个一个 NDIS_BUFFER 结构的链表.其实,封包数据就是用 NDIS_BUFFER 来表示,VirtualAddress 是封包的缓冲区指针,Length 是 VirtualAddress 的长度,Next 是下一个封包缓冲区.这些封包缓冲区的数据联合起来就是一个完整的网络封包.NDIS 在分包的时候通常有一定的规律,每一个 NDIS_BUFFER 里通常保证是一个或几个完整的协议头部,比如:Windows 95/98/me/2000 server 下 NDIS_PACKET 的第一个 NDIS_BUFFER 保存的是 Ethernet Header,第二个 NDIS_BUFFER 则是 IP Header + Tcp/Udp/Icmp Header,接着是封包数据.而 2000 Professional 的第一个 NDIS_BUFFER 则保存着 Ethernet Header + IP Header + Tcp/Udp/Icmp Header,接着就是封包数据.这个例子是通过测试得出的结论,所以不能作为通用的操作,还要具体情况具体分析.Windows 95/98/ME 下,可以直接通过结构的字段引用感兴趣的字段.Windows 2000 下,一般不能直接引用结构的字段,而是使用 2000 定义的一组类似 NdisQueryBuffer、NdisGetNextBuffer 的函数来得到感兴趣的字段.

2.3 NDIS 中间层驱动的实现

过滤程序通过入口函数 DwiverEntry 向 NDIS 注册一个 Miniport 接口和一个 Protocol 接口.通常必须实现以下功能:

调用 NdisMInitializeWrapper 函数注册 miniport 设备,得到设备句柄.

利用得到的句柄调用 NdisMRegisterLayeredMiniport 函数,注册 Miniport 功能函数以开放出为上层协议驱动提供的接口.其 Miniport 功能函数主要包含 Minitport 设备初始化并加入全局列表、发送封包函数、状态相关函数以及 PNP 相关函数.这些功能的实现请参考 DDK 文档资料.

调用 NdisRegisterProtocol 函数,注册 Protocol 功能函数开放出为底层 Miniport 驱动提供的接口,并将本身绑定到底层驱动程序上.Protocol 功能函数主要有绑定到 Miniport 设备函数、接收封包函数、状态函数以及 PNP 函数.这些功能的实现请参考 DDK 文档资料.

如果既注册了 Miniport 功能函数和 Protocol 功能函数,则会调用 NdisMAssociateMiniport 函数并通知 NDIS 库.

3 小 结

本方法在系统核心层实现,可以过滤所有的网络访问,包括网络邻居、WWW、FTP、ICMP、TCP、UDP、SMTP、POP 等,是最安全的实现方法,但由于要在核心设备层实现,实现难度最大、兼容性很差、程序质量要求很高、必须对每个版本实现一个程序.必须具有 Windows 设备程序开发能力和 Windows DDK 开发软件包.

参考文献:

[1] 张友生.远程控制编程技术[M].北京:电子工业出版社,2002. 9~18.
[2] 朱雁辉.Windows 防火墙与网络封包截获技术[M].北京:电子工业出版社,2002.39~72.
[3] 贾佳,郝洪明.网上警戒——防范黑客技术[M].北京:电子工业出版社,2002.391~393.

(上接第 28 页)

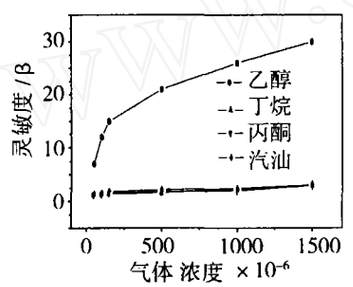

图 3 元件的灵敏度随气体浓度的变化

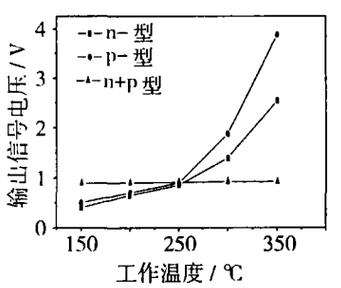

图 4 输出信号电压随元件温度的变化

3 结 语

上述理论分析和实验结果表明,半导体 n + p 结构气敏元件克服了现行的半导体电阻式气敏元件选择性、灵敏度、热稳定性等不佳的缺点.当构成整体气敏元件的两种敏感体满足上述理论分析所述条件时,由于互补反馈原理,整体气敏元件的选择性可大大提高,同时还具有好的热稳定性.使用该结构的气敏元件时,无需外接负载电阻而具有使用方便的优点.

参考文献:

[1] 1 吴兴惠.敏感元器件及材料[M].北京:电子工业出版社,1992.15~21.
[2] 吴兴惠,李艳峰,周桢来,等.实现气敏元件高选择性的一种方法[J].半导体学报,1993,14(7):439~444.
[3] 吴兴惠,李艳峰,周桢来,等.实现气敏元件灵敏度倍增的一种方法[J].半导体学报,1994,15(9):643~649.
[4] 吴兴惠,李艳峰,周桢来,等.N-P 型半导体气敏元件的理论分析[J].云南大学学报(自然科学版),1993,15(1):74~77.