文章编号：2096-1472(2016)-11-41-04

基于NDIS过滤驱动防火墙的设计和实现

王亚伟
（天津工业大学计算机科学与软件学院，天津 300387）

摘　要：Windows作为时下最流行的个人操作系统，研究Windows环境下的防火墙有一定的现实意义。主机防火墙的数据包过滤算法是线性查找，其性能会随着过滤规则的增加而变得越来越差，为了提高防火墙的运行速度，对非法数据包的拦截能力，提出一种基于HASH双链表的过滤算法，并且通过分析、比较Windows环境下网络数据包的拦截技术，开发了一款基于NDIS6.0过滤驱防火墙软件，经过测试，它对非法数据包具有强大的拦截能力。

关键词：NDIS过滤驱动；防火墙；包过滤；HASH双链表；Duilib
中图分类号：TP391.9　　**文献标识码**：A

Design and Implementation of the Firewall Based on the NDIS Filter Driver

WANG Yawei
(*School of Computer Science and Software Technology*,*Tianjin Polytechnic University*,*Tianjin* 300387,*China*)

Abstract: Windows as the most popular individual operating system.The research of firewall in the windows environment has certain practical significance.Host firewall packet filtering algorithm is linear search,its performance will only get worse and worse with the increase of filtering rules,in order to improve the running speed of the firewall,the illegal packet interception ability,in this paper,a filtering algorithm based on HASH double linked list,and by analyzing and comparing Windows environment network packet interception technology,developed a firewall software,based on NDIS6.0 filtering flooding,after test,it has strong ability to intercept illegal packet.

Keywords:NDIS filter driver;firewall;packet filtering;HASH double linked list;duilib

1 引言(Introduction)

防火墙在网络安全防护中具有重要作用，从应用的角度来看，防火墙可以分为企业级防火墙和个人防火墙[1]。前者主要是部署在内外部网络的边界，对内外部网络实施隔离，从而保护内部网络的安全，而后者则通过软件的方式来实现，防火墙包过滤按顺序执行从第一条规则开始，直到找到一个匹配的规则。如果没有找到匹配的规则，则数据包被默认规则处理，叫做规则库的线性搜索算法。因此，过滤过程的计算复杂度大大取决于每个过滤规则的长度(规则数量字段)，以及找到一个匹配的深度过滤规则的安全政策(过滤规则的数量)[2]。为了提高防火墙的运行速度，将防火墙的过滤规则组织成HASH双链表，实验表明在防火墙规则足够多时候，HASH双链表极大的提高了运行速度。

本文深入研究基于NDIS过滤驱动的数据包拦截方法和Windows内核态驱动程序与用户态应用程序间的通信机制，设计并实现了一种基于Windows内核态的个人防火墙系统。

2 系统的设计目标(System design object)

本防火墙对所有的网络报文进行监控，从而对非法报文进行拦截，隔断非法入侵。为了保护系统安全目的，本防火墙主要完成以下功能。

(1)驱动程序

可以根据需要设置过滤规则，规则设置是防火墙的核心功能之一，基于五元组[3]的数据包过滤规则可以管理应用联网进程的黑白名单、端口过滤、IP过滤、异常端口连接等。用户可以通过设置规则来自由的对数据进行过滤。降低计算机被入侵的风险对所有的网络数据包拆包分析，根据过滤规则匹配，从而决定包过滤或放行。对网络报文的统计，实时统计所有数据包，对错误的数据报文记录日志。日志文件是用于记录防火墙操作事件的记录文件或文件集合，为网络维护者判断故障提供依据。

(2)主程序

防火墙的过滤规则的管理需要一个友好的界面供用户操作，比如添加删除规则和启动关闭防火墙等。

(3)规则管理

规则设置是个人防火墙的核心功能之一，通过规则设置可以管理应用联网进程的黑白名单、端口过滤、域名过滤、IP过滤、异常端口连接等。用户可以通过设置规则来自由的对数据进行过滤。降低计算机被入侵的风险。基于五元组的

数据包过滤。

3 关键技术(Key technology)

3.1 驱动程序的设计

NDIS5.0和之前的版本已经被官方抑制，换句话说就是Windows Vista和之后的系统不在支持NDIS5.0，NDIS6.0支持Windows Vista和之后的系统。NDIS6.0内核网络驱动程序可分为微端口驱动程序、协议驱动程序、过滤驱动程序、中间驱动程序。其中过滤驱动程序是NDIS6.0新引入的驱动程序模型。过滤驱动程序提供微端口驱动程序过滤服务，它可以实现数据过滤应用程序安全或其他目的，监控和收集网络数据统计信息的应用程序。过滤驱动可以监视和修改协议驱动程序和微端口的驱动程序之间的交互，过滤驱动程序更容易实现，处理开销低于NDIS中间驱动程序。本防火墙驱动程序部分就是在该框架的基础上实现的。

在NDIS 6.0和以后的版本，NET_BUFFER是网络数据的基本单元。每个NET_BUFFER结构有一个MDL链表，一个MDL链表组成一个完整的数据包。多个NET_BUFFER结构可以被附加到一个NET_BUFFER_LIST结构，NET_BUFFER结构组织是一个以null结尾单链表。下面给出微软对这三个数据结构的定义：

```
typedef struct _NET_BUFFER {
  NET_BUFFER_HEADER      NetBufferHeader;
  USHORT                 ChecksumBias;
  USHORT                 Reserved;
  NDIS_HANDLE            NdisPoolHandle;
  ...
} NET_BUFFER,*PNET_BUFFER;
typedef union _NET_BUFFER_HEADER {
  NET_BUFFER_DATA NetBufferData;
  SLIST_HEADER    Link;
} NET_BUFFER_HEADER,*PNET_BUFFER_HEADER;
typedef struct _NET_BUFFER_DATA {
  PNET_BUFFER            Next;
  PMDL                   CurrentMdl;
  ULONG                  CurrentMdlOffset;
  NET_BUFFER_DATA_LENGTH NbDataLength;
  PMDL                   MdlChain;
  ULONG                  DataOffset;
} NET_BUFFER_DATA,*PNET_BUFFER_DATA;
typedef struct _NET_BUFFER_LIST {
  NET_BUFFER_LIST_HEADER NetBufferListHeader;
  PNET_BUFFER_LIST_CONTEXT Context;
  PNET_BUFFER_LIST         ParentNetBufferList;
  NDIS_HANDLE              NdisPoolHandle;
  ...
} NET_BUFFER_LIST,*PNET_BUFFER_LIST;
typedef union _NET_BUFFER_LIST_HEADER {
  NET_BUFFER_LIST_DATA NetBufferListData;
  SLIST_HEADER         Link;
} NET_BUFFER_LIST_HEADER,*PNET_BUFFER_LIST_HEADER;
typedef struct _NET_BUFFER_LIST_DATA {
  PNET_BUFFER_LIST Next;
  PNET_BUFFER      FirstNetBuffer;
} NET_BUFFER_LIST_DATA,*PNET_BUFFER_LIST_DATA;
```

MDL(内存描述符列表)是一个系统定义的结构，通过一系列物理地址描述缓冲区。执行直接I/O的驱动程序从I/O管理器接收一个MDL的指针，并通过MDL读写数据。

NET_BUFFER_DATA结构包含MDL用于管理数据缓冲区的信息附加到NET_BUFFER结构，并确定下一个NET_BUFFER结构NET_BUFFER列表结构。而每个NET_BUFFER结构均包含一个MDL链表，MDL是存放网络数据的仓库。NET_BUFFER链表附加在NET_BUFFER_LIST结构中。附加在同一个NET_BUFFER_LIST上的数据拥有相同的以太网类型，IP协议，源MAC地址和目标MAC地址。MDL(内存描述链表)、NET_BUFFER、NET_BUFFER_LIST_CONTEXT和NET_BUFFER_LIST之间的关系如图1所示。

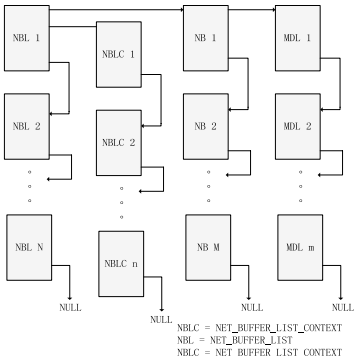

图1 四种结构体之间的关系

Fig.1 The relationship between the four types of structures

NDIS过滤驱动程序框架的数据包发送和接收回调函数分别是FilterReceiveNetBufferLists、FilterSendNetBufferLists、防火墙的过滤规则在这两个回调函数逐一匹配，更加过滤规则的行为拒绝，发送或者接收。对发送回调函数介绍如下：

```
VOID FilterSendNetBufferLists(
```

```
    NDIS_HANDLE FilterModuleContext,
    PNET_BUFFER_LIST NetBufferLists,
    NDIS_PORT_NUMBER PortNumber,
    ULONG  SendFlags )
    {  ...
      QUEUE_HEADER NormalHead,DropHead;
        SendHandler(pFilter,NetBufferLists,&NormalHead,&
DropHead);
        if (!IsQueueEmpty(&NormalHead)) {
          NdisFSendNetBufferLists(pFilter->
FilterHandle,ENTRY_TO_NET_BUFFER_
LIST(NormalHead.Head),PortNumber,SendFlags);
        }
        if (!IsQueueEmpty(&DropHead)) {
          NdisFSendNetBufferListsComplete(pFilter->
FilterHandle,ENTRY_TO_NET_BUFFER_
LIST(DropHead.Head),DispatchLevel
          NDIS_SEND_COMPLETE_FLAGS_DISPATCH_
LEVEL:0);
        }
    }
```

对将要发送的数据包交由SendHanler回调函数处理，返回NormalHead链表和DropHead链表，对NormalHead链表的数据包全部正常发送，对DropHead链表里的所有数据包做丢包处理。

```
    ULONG SendHandler(
        __in   PMS_FILTER
pFilter,
        __in  PNET_BUFFER_LIST
pSrcNbls,//将原来的链表分开
        __out  PQUEUE_HEADER
pNormalHead,//需要接受的Nbl
        __out  PQUEUE_HEADER
pDropHead//需要丢弃的数据链表
        ){...
        PKTACTION pktAction;
        PNET_BUFFER pNetBuffer=NULL;
        while (pSrcNbls) {
                pNextNbl=NET_BUFFER_LIST_NEXT_
NBL(pSrcNbls);
                NET_BUFFER_LIST_NEXT_
NBL(pSrcNbls)=NULL;
        pNetBuffer=NET_BUFFER_LIST_FIRST_
NB(pSrcNbls);
        uLength=NET_BUFFER_DATA_
LENGTH(pNetBuffer);
        pBuf=MyReadNetBufferData(pFilter->
FilterHandle,pNetBuffer);
        if (pBuf==NULL) {
        pNumsOfNormalNbl++;
        InsertTailQueue(pNormalHead,pSrcNbls);
        goto DoNextNBL;//直接跳到下一步
        }
        pktAction=DoSendPacket(pBuf,uLength);//数据
包的过滤函数
        if (pktAction==PKT_DROP) {//收到bad包直接插
入丢包NBL跳到下一步循环
        FILTER_FREE_MEM(pBuf);
        InsertTailQueue(pDropHead,pSrcNbls);//插入丢
包NBL
        goto DoNextNBL;//直接跳到下一步
                }
                FILTER_FREE_MEM(pBuf);
                pNumsOfNormalNbl++;
                InsertTailQueue(pNormalHead,pSrcNbls);
        DoNextNBL:
                pSrcNbls=pNextNbl;
        }
        return pNumsOfNormalNbl;
    }
```

SendHandler回调函数对将要发送的数据链表拆包对于每个数据包的内容交付给DoSendPacket回调函数。DoSendPacket(数据包、数据包的长度)，函数定义如下：

```
    PKTACTION DoSendPacket(PUCHAR pData,ULONG
uLength) {
    ...
        if (错误的报文)
    return PKT_DROP;
    PLIST_ENTRY pRuleHead;
        PLIST_ENTRY pRuleEntry;
        BOOLEAN bFound=FALSE;
        PIPFILTER p=NULL;
        int id=FilterHashValue(pIP->srcIp,pIP->
dstIp);//查找hash链表
        FILTER_ACQUIRE_LOCK(&g_IPFilterManage.
Locks[id],FALSE);
```

```
pRuleHead=&g_IPFilterManage.Lists[id];
for (pRuleEntry=pRuleHead->Flink; pRuleEntry!=pRuleHead; pRuleEntry=pRuleEntry->Flink) {
    p=CONTAINING_RECORD(pRuleEntry, IPFILTER,ListEntry);
    if (p->srcIp==pIP->srcIp&&p->dstIp==pIP->dstIp&&p->srcPort==sPort&&p->dstPort==dPort) {
        bFound=TRUE;
        break;
    }
}
FILTER_RELEASE_LOCK(&g_IPFilterManage.Locks[id],FALSE);
if (bFound) return p->action;
return PKT_SEND;
}
```

DoSendPacket函数是防火墙发送数据包匹配防火墙的关键，通过查找hash(拉链法)双链表，如果找到相应的规则就按照规则定义的行为返回，否则直接发送数据包不做处理。

3.2 应用程序的设计

应用程序是防火墙的GUI部分，负责向用户报告结果和提示操作的界面。应用程序对数据包监控分析及过滤规则的设置，用户通过防火墙安全规则的设定，向数据包拦截模块传递规则。基于MFC和WTL框架的程序界面的不够友好，如果想做出漂亮的个性化界面，需要重新绘制各种控件比如按钮、滚动条、编辑框、列表、下来菜单、容器等。而duilib是一款基于win32开源的界面库，已经绘制好各种常用的控件，编写漂亮的界面只需对xml进行编辑即可，这点跟html原理有点像，duilib遵循BSD协议，可以免费用于商业项目。

现在大多数有关界面的产品，都是基于DirectUI的设计思想，界面和逻辑分离。基于以上原因，决定使用Windows比较流行的duilib库进行应用程序的开发，其实也有其他的一些库，但是比较而言duilib更具优势。duilib是一款强大的界面开发工具，可以将用户界面和处理逻辑彻底分离，提高开发效率[4]。

3.3 应用程序和驱动程序的通信

个人防火墙不仅要实现对网络数据包的截获，还要在分析数据包后，根据安全规则对数据包进行处理，并且将这些事件记入日志。由于安全规则设置和日志的记录一般是由一个具有人机界面的应用程序来完成，因此，要实现个人防火墙还必须实现设备驱动程序和应用程序的信息交互[5]。设备驱动程序和应用程序之间的通信包括应用程序传送数据给设备驱动程序和设备驱动程序给应用程序发送消息两个方面。前者实现较为容易，后者实现远比前者复杂。驱动程序和应用程序之间通过共享内存的方式实现通信，两者间的内存共享有两种实现方法，一种是通过共享内存对象方法来实现，另一种是通过设备输入和输出控制(IOCTL)方法来实现[6]。

内存映射方法：首先由在驱动程序级分配缓冲区，再将这个缓冲区映射到特定的用户程序的地址空间，并将映射地址返回给用户程序，供内核线程与用户线程共享缓冲区。这种方式是内核来分配内存空间，但是使用MmAllocatePagesForMDL从主内存池中分配，返回得到一个MDL，对于驱动如何使用该共享内存，采用MmGetSystemAddressForMdlSafe得到其内核地址。对于应用层使用该共享内存，采用MmMapLockedPagesSpecifyCache映射到应用层进程地址空间中，返回用户层地址空间的起始地址，将其放在IOCTL中返回给用户应用程序。设定AccessMode参数为UserMode。对MmMapLockedPagesSpecifyCache函数调用返回值是MDL描述内存页映射的用户虚拟地址空间地址。驱动可以将其放在对应IOCTL的缓存中给用户应用程。在不需要时将分配的内存清除掉。你需要调用MmFreePageFromMdl来释放内存页。并且调用IoFreeMdl来释放由MmAllocatePageForMdl(Ex)创建的MDL[7]。

设备输入和输出控制(IOCTL)方法：在内核线程和用户线程之间共享缓冲区最简单的方法是使用DeviceIoControl函数。用户程序发送一个IO控制码(IOCTL)给驱动程序，同时还传递一个指向要共享的缓冲区的指针，内核程序与用户程序根据该指针实现对缓冲区的访问。操作过程是首先由用户线程分配要共享的缓冲区。再调用DeviceIoControl，将被共享的缓冲区的地址和长度放在参数OutBuffer中，同时设置共享模式，驱动程序将根据共享模式来决定如何访问缓冲区。共享模式包括缓冲IO方式与直接IO方式两种缓冲IO方式[8]。

4 驱动的安装和调试(The installation and commissioning of driver)

目标平台：Win7、Win8、Win10。

开发环境：Visual Studio 2015 Common和Windows WDK10.0.26639、Windows SDK10.0.26639。

调试环境：VM 虚拟机。

NDIS驱动程序使用VS2015编译完毕，在Release目录生成的文件firewallex.inf、firewallex.sys、firewallex.cer。.inf是驱动程序的安装信息，.sys就是我们驱动程序，.cer是

(下转第40页)