

# 中山大学移动信息工程学院移动网络安全 实验报告

(2016 年春季学期)

刘爽

班级：13M2

ID：13354222

专业：移动互联网

# 1. 实验题目

## Implementation of Firewalls Based on Packet Filtering

### 1.1 实验描述

在日常生活中，防火墙的作用是帮助我们区分不同的包并防御违法的攻击。个人的防火墙控制进出个人计算机的网络流量，它根据安全协议通过或者拒绝相应的数据包。

防火墙的功能有：实现安全机制，监控安全相关行为；提供验证；VPN

防火墙不能保护：绕开防火墙的攻击；防火墙内部的后门；传输内容病毒查杀

防火墙有好几种类型，基于包过滤的防火墙、电路层防火墙、应用层防火墙、基于状态检测的防火墙。

在这次实验中，我们专注于一种非常简单的防火墙，包过滤防火墙。包过滤在网络层中进行，白名单里的过，黑名单里的直接在路由器上就丢掉，包过滤在路由器上就可以实现。其优点是简单，代价低，速度快。其缺点是功能有限，不能针对特定应用进行过滤，因为看不到上层；规则粗放，很难做到准确过滤。包过滤通过检测包来实现：如果一个包匹配包过滤原则，那么这个包就会被丢弃，或者发送一个“error responses”信息给源主机。包过滤通常是无状态的，即只根据包含在数据包中的信息进行过滤，而不需要关注数据包是否是已经存在的流量中的一部分。包过滤通常使用包的源地址和目的地址，相关协议，端口号，TCP或UDP通信，进行过滤。

### 1.2 实验目的

- 学会配置虚拟机。
- 学会使用理解防火墙的作用。

- 学会实现简单的包过滤防火墙。
- 学会看懂防火墙工程源代码。
- 理解底层应用驱动机制。

### 1.3 实验要求和实验原理

在这次课题中，我们需要实现用于 Windows XP 系统的包过滤。这个防火墙由两个部分构成：协议的配置以及包过滤。防火墙的界面基于 MFC，可使用 C++ 完成这个软件。

#### • 防火墙协议

协议配置模块允许系统使用者建立防火墙协议。个人防火墙支持多种协议，包括极简单的协议以及很复杂的协议。在这次课题中，对协议的要求如下所示，如果有额外功能可获得加分。基本要求是，防火墙可以根据下列原则阻塞和允许进入当前主机或离开当前主机的数据包：

I. Protocol: 定义包过滤策略应该应用到哪一类的协议中。这个协议可以是 TCP，UDP 或者 ICMP。

II. Source and Destination address: 将数据包与源地址和目的地址进行匹配。当有很多过滤规则都由同样的源地址和目的地址时，地址与子网掩码通常一起决定一个阻塞的地址范围。

III. Source and Destination port number：将源端口和目的端口与包进行匹配。

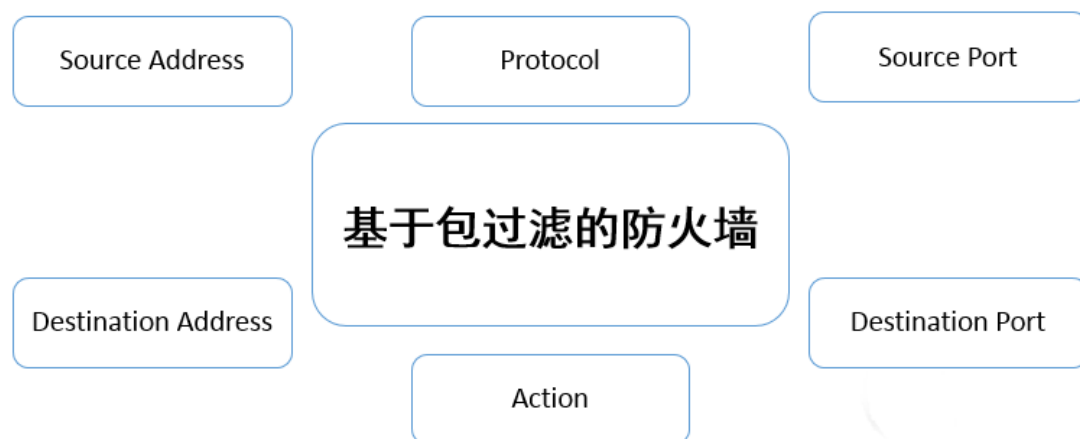
IV. Action: 当数据包匹配一条规则时，定义行为。通常的行为包括：

Deny：阻塞数据包。

Allow：当整条网络都被 Deny 的时候，允许从某些单独的地址来的数据包通

过。

具体关系如下所示：



## • 包过滤

防火墙的主要部分是过滤那一部分，它加强了经用户设置的防火墙协议。

在 NetDefender 的帮助下，我们可以轻易地过滤各种各样的数据包。

这主要通过实现许多内核中的 hooks 来达到。这些 hooks 可以被插入各种各样的地方，包括数据包进出的路径。如果我们想要控制进来的数据包，我们只需要将我们的程序与相应的 hook 连接寄来就好。一旦一个数据包到达，我们的程序就会被唤醒。我们的程序可以决定这个数据包是否要被阻塞掉。

在这次课题中，我们需要使用 NetDefender 来实现包过滤模块。这个模块将从一个数据结构中得到防火墙协议，并使用它们来决定一个数据包是否应该被阻塞。我们应该支持一个动态的备至，即用户可以动态地修改防火墙协议，而包过滤模块必须自动强化这些协议。

防火墙应当具有以下功能：

直接过滤进、出本机所有的数据包；

能够选择过滤进、出本机的某一类数据包；

可以用户自己添加过滤规则：对指定端口、指定源地址、指定目的地址的指定类型的数据包进行指定的操作；

程序的健壮性、可用性；

创新的功能。

实验任务：

根据事先的功能，设计相应的测试方法；

测试方法自己来实现，在验收时候当场演示。

## 2. 实验分析

我们首先要知道一些 project 的结构，才能够进行自己的实现。

fireView.h：基于底层文件进行过滤规则的查看与控制。

- a) 构造函数主要用于初始化变量。
- b) PreCreateWindow 指的是初始化界面时的行为，这里注册并启动了需要用到的服务。
- c) 接下来还是几个跟 NFC 有关的用于前端交互的函数。
- d) OnAddrule 指点击了 AddRule 按钮后的行为。
- e) OnStart 指点击了 Start 或者 Stop 按钮后的行为，对于 hook 进行启动和中止的控制。
- f) OnBlockping 指点击了 Block Ping 按钮后的行为，需自己补充，功能是加入一条对 ping 包完全 block 的规则。
- g) OnBlockAll 指点击了 Block All 按钮后的行为，需自己补充，功能是加入一条对所有包完全 block 的规则。
- h) OnAllowAll 指点击了 Allow All 按钮后的行为，需自己补充，功能是无效化

所有的 block 规则。

- i) ImplementRule 在 OnViewrules 函数中进行了调用,指点击了 View Rules 按钮后的行为,需自己补充,功能是显示所有的规则在前端界面上。(利用本文件中已有的跟 NFC 有关的用于前端交互的函数,比如: AddItem 函数)
- j) ParseToIp 函数,将字符串转换为 IP 地址。
- k) 后面都是一些跟 NFC 有关的用于前端交互的函数。

### 3. 实验内容与实验结果

**实现功能 1 : Block Ping , 阻塞所有的 ICMP 包。**

protocol 的值需要参考 IP 包头的标准,1 代表 ICMP,6 代表 TCP,17 代表 UDP,0 则代表所有协议类型。

IPFilter 结构体的其他变量设为 0 的时候都是代表全部的意思。

即全部的目的 IP 地址,目的端口,目的 IP 地址子网掩码,源 IP 地址,源端口,源 IP 地址子网掩码。

因为是 Block 操作,将 IPFilter 变量中的 drop 置为 true,代表碰到符合要求的数据包一律丢弃。

在补足好 IPFilter 结构体之后,需要把这个过滤规则添加进防火墙策略中。

我们需要使用到 CAddRuleDlg 的 AddFilter 函数。在 fireView.h 文件中已经有这个类型的变量 m\_Addrule,直接使用即可。

添加完过滤规则之后,我们需要改变界面的状态,将 BlockPing 按钮置灰,其余不变。界面的按钮状态有四个变量,分别是 start,block,allow,ping,代表 start 按钮,Block all 按钮,Allow all 按钮,和 Block Ping 按钮的状态。

具体的代码如下图所示。

```
//禁用所有的ICMP包
void CFireView::OnBlockping()
{
    // MessageBox("此功能需要你来实现! ");
    // Your code
    IPFilter      IPflt;

    IPflt.protocol      = 1;//1 for ICMP Protocol
    IPflt.destinationIp  = 0;//inet_addr("127.0.0.1"); // all destinations
    IPflt.destinationMask = 0;//inet_addr("255.255.255.255");// all destination masks
    IPflt.destinationPort = 0;//0 for all ports
    IPflt.sourceIp       = 0;//0 means drop all packets irrespective of source
    IPflt.sourceMask     = 0;
    IPflt.sourcePort     = 0;//0 means from any source port
    IPflt.drop           = TRUE;

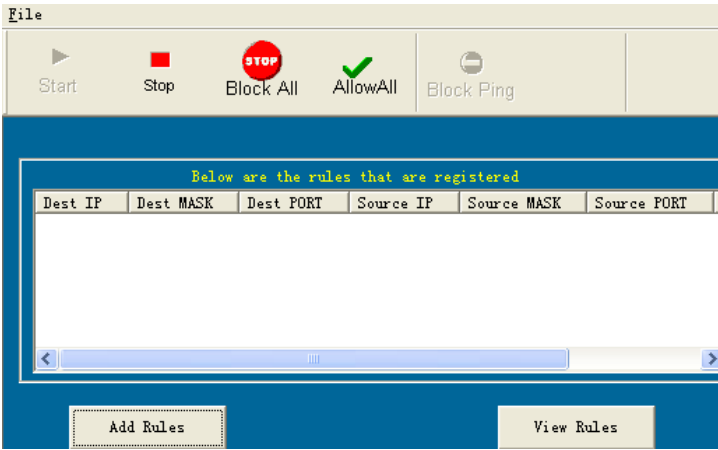
    m_Addrule.AddFilter(IPflt);
    ping = FALSE;
    allow = TRUE;
    block = TRUE;
}

```

测试方法：使用 Ping 来发送 ICMP 包。如下图所示：

```
C:\Documents and Settings\NS_Lab>ping 192.168.253.1
```

实验结果：



```
Pinging 192.168.253.1 with 32 bytes of data:
Reply from 192.168.253.1: bytes=32 time=2ms TTL=128
Reply from 192.168.253.1: bytes=32 time=2ms TTL=128
Reply from 192.168.253.1: bytes=32 time=2ms TTL=128
Reply from 192.168.253.1: bytes=32 time=2ms TTL=128

Ping statistics for 192.168.253.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 2ms, Maximum = 2ms, Average = 2ms

C:\Documents and Settings\NS_Lab>ping 192.168.253.1

Pinging 192.168.253.1 with 32 bytes of data:

Destination host unreachable.
Destination host unreachable.
Destination host unreachable.
Destination host unreachable.
```

如上图所示，在启动防火墙之前，ping 主机 192.168.253.1 收到回复，说明可以 Ping 通。在使用了防火墙之后，回复的值是 Destination host unreachable，说明数据包没有成功发出，ICMP 包被成功地阻塞。

## 实现功能 2：Block All，阻塞所有的包。

protocol 的值需要参考 IP 包头的标准，1 代表 ICMP，6 代表 TCP，17 代表 UDP，0 则代表所有协议类型。

IPFilter 结构体的其他变量设为 0 的时候都是代表全部的意思。

即全部的目的 IP 地址，目的端口，目的 IP 地址子网掩码，源 IP 地址，源端口，源 IP 地址子网掩码。

因为是 Block 操作，将 IPFilter 变量中的 drop 置为 true，代表碰到符合要求的数据包一律丢弃。

在补足好 IPFilter 结构体之后，需要把这个过滤规则添加进防火墙策略中。

我们需要使用到 CAddRuleDlg 的 AddFilter 函数。在 fireView.h 文件中已经有这个类型的变量 m\_Addrule，直接使用即可。

添加完过滤规则之后，我们需要改变界面的状态，将 BlockPing 按钮置灰，其余不变。界面的按钮状态有四个变量，分别是 start，block，allow，ping，代表 start 按钮，Block all 按钮，Allow all 按钮，和 Block Ping 按钮的状态。

具体的代码如下图所示。

```
//禁用所有包
void CFireView::OnBlockall()
{
    // MessageBox("此功能需要你来实现！");
    // Your code
    // CAddRuleDlg arule;
    IPFilter IPflt;

    IPflt.protocol = 0; //0 for all the protocols
    IPflt.destinationIp = 0; //0 for all destinations
    IPflt.destinationMask = 0; //0 for all destination masks
    IPflt.destinationPort = 0; //0 for all ports
```



```

    IPflt.sourceIp      = 0; //0 means drop all packets irrespective of source
    IPflt.sourceMask    = 0;
    IPflt.sourcePort    = 0; //0 means from any source port
    IPflt.drop          = TRUE;

    m_Addrule.AddFilter(IPflt);

    block = FALSE;
    ping  = FALSE;
    allow = TRUE;
}

```

测试方法：既然是 Block 掉所有类型的包，我们需要测试 ICMP, TCP 和 UDP 三种类型的包。

ICMP 包：在命令行使用 ping 命令即可。成功界面如下所示：

```

C:\Documents and Settings\NS_Lab>ping 127.0.0.1

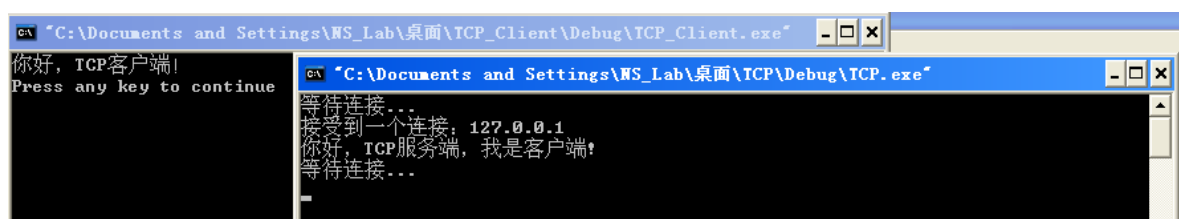
Pinging 127.0.0.1 with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time=11ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 11ms, Average = 2ms

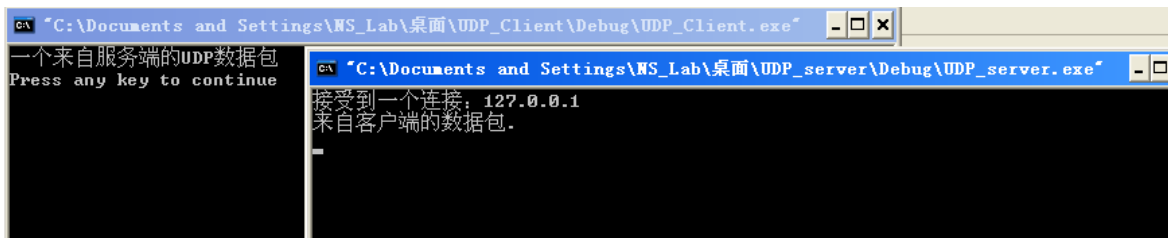
```

TCP 包：写一个发送 TCP 包的客户端的.c 文件 tcp\_client.c ,和一个接收 TCP 包的客户端的.c 文件 tcp\_server.c。成功界面如下所示：



如上图所示，TCP\_Client.exe 代表发送 TCP 包的客户端。TCP.exe 代表接收客户端发送的 TCP 包，打印内容并返回回复的服务端。客户端将服务端返回的 TCP 包的内容打印出来。

UDP 包：写一个发送 UDP 包的客户端的.c 文件 udp\_client.c ,和一个接收 UDP 包的客户端的.c 文件 udp\_server.c。成功界面如下所示：



如上图所示，UDP\_Client.exe 代表发送 UDP 包的客户端。UDP\_server.exe 代表接收客户端发送的 UDP 包，打印内容并返回回复的服务端。客户端将服务端返回的 UDP 包的内容打印出来。

开启防火墙：



实验结果：

Block ICMP:



如上图所示，在使用了防火墙之后，回复的值是 Destination host unreachable，说明数据包没有成功发出，ICMP 包被成功地阻塞。

Block TCP:



如上图所示，在开启防火墙之后，TCP 客户端和服务端无法建立连接，说明三次握手的包都无法成功发送，阻塞 TCP 包成功。

Block UDP:



如上图所示，服务器端和客户端一个无法收包，一个无法发包，因而不会有任何内容显示出来，说明 UDP 包被成功阻塞。

### 实现功能 3：Add Rules，添加自定义规则。

首先我们需要验证用户输入的 IP 地址是否正确，如果不正确需要提示用户。

验证 IP 地址的工作在 Verify 函数中做：

首先需要检测用户输入的 IP 地址中是否带有 “.”，如果没有则是错误的。

```
if(str.Find('.') == -1)
    return FALSE;
```

其次检测 IP 地址中是否含有非数字的非法字符，如果有则是错误的。

```
if(str.FindOneOf("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ") != -1)
    return FALSE;
if(str.FindOneOf("!@#$%^&*()_+|-;: '\/?><,"") != -1)
    return FALSE;
```

查看三个点的位置，如果第一个点的位置不在有效数字的后面，则是错误的。

```
int _pos = 0;
_pos = str.Find('.');//the first one, start from the 0 digit
if( (0 > _pos) || (_pos > 3))
    return FALSE;
```

如果第二个点的位置不在有效数字的后面，则是错误的。

```
int newpos = _pos;
_pos = str.Find('.', _pos+1); //the first one, start form the pos+1 digit
if((newpos+1 >= _pos) || ( _pos > newpos + 4))
    return FALSE;
```

如果第三个点的位置不在有效数字的后面，则是错误的。

```
int newpos = _pos;
_pos = str.Find('.', _pos+1); //the first one, start form the pos+1 digit
if((newpos+1 >= _pos) || ( _pos > newpos + 4))
    return FALSE;
```

继续往后找是否有 “.”，如果可以找到则说明点多于 3 个，是错误的。

```
newpos = _pos;
_pos = str.Find('.', _pos+1);
if(_pos != -1)
    return FALSE;
```

当 IP 地址的格式验证之后，需要验证 IP 地址的值是否正确，即是否所有的数值都在 0~255 的范围内。

```
for(int cnt = 0 ;cnt <= 3;cnt++)
{
    if(cnt < 3) //count the number of the value
        pos = str1.Find('.', pos+1);
    else
        pos = str.GetLength();

    str1 = str.Left(pos); //get the string including the value
    str1 = str1.Right(pos-(prevpos+1)); //get the value without a dot
    unsigned int a = atoi(LPCTSTR(str1)); //Convert the string to the number
    if((0 > a) || (a > 255)) //check whether the value is valid
    {
        return FALSE;
    }
    prevpos = pos;
}
```

在进行了对 IP 地址的验证之后，我们需要从前端界面获取用户输入的防火墙规则，并将获取到的规则进行整理后写入文件，文件名为 saved.rul，最后创建对应的防火墙过滤规则并且写入设备文件，可借助 AddFilter 函数。

这一系列的动作可在 OnAddSave 函数中实现。

获取用户选择的对特定的数据包的操作，是允许通过还是阻塞。在用户界面上，

Allow 为 0，Deny 为 1；

```

if(action == 0)
    setact = FALSE;
else
    setact = TRUE;

```

获取用户对协议的选择,并对应相应的数据包类型。在用户界面上,ICMP 为 0, TCP 为 1, UDP 为 2。

```

int proto = m_protocol.GetCurSel();
if(proto == 0)
    setproto = 1;
if(proto == 1)
    setproto = 17;
if(proto == 2)
    setproto = 6;

```

根据用户输入的规则创建新的 filter, 用 inet\_addr 将字符串转化为符合要求的 IP 地址。源子网掩码和目的子网掩码所有值全部为 255。

```

IPFilter ip;
ip.destinationIp = inet_addr((LPCTSTR)m_sdadd);
ip.destinationMask = inet_addr("255.255.255.255");
ip.destinationPort = htons(atoi((LPCTSTR)m_sdport));
ip.sourceIp = inet_addr((LPCTSTR)m_ssadd);
ip.sourceMask = inet_addr("255.255.255.255");
ip.sourcePort = htons(atoi((LPCTSTR)m_ssport));
ip.protocol = setproto;
ip.drop = setact;

```

接下来将规则存在对应文件中, 将新增规则的详细信息赋给一个字符串的变量。

```

CString _str;
char ch1[100];
wsprintf(ch1, "%s,%s,%s,%s,%s,%s,%s,%d,%d\n",
    m_sdadd,
    "255.255.255.255",
    m_sdport,
    m_ssadd,
    "255.255.255.255",
    m_ssport,
    setproto,
    setact);

```

打开文件 saved.rul, 如果不能则报错。

如果可以则调到文件最末尾, 以便添加新的规则。

```

if( NewFile() == FALSE)
    MessageBox("unable to create file"); //create a new file

GotoEnd();

```

将新增规则添加于文件末尾，并关闭文件，如果无法关闭则报错。同时，向驱动添加新增的过滤规则。

```
SaveFile(ch1);  
  
if(CloseFile() == FALSE)  
    MessageBox("Unalbe to close the file");  
  
DWORD result = AddFilter(ip);
```

**实验结果：**

对 IP 地址格式的验证



如上图所示，IP 地址长度过长会报错。



如上图所示，IP 地址长度过短会报错。

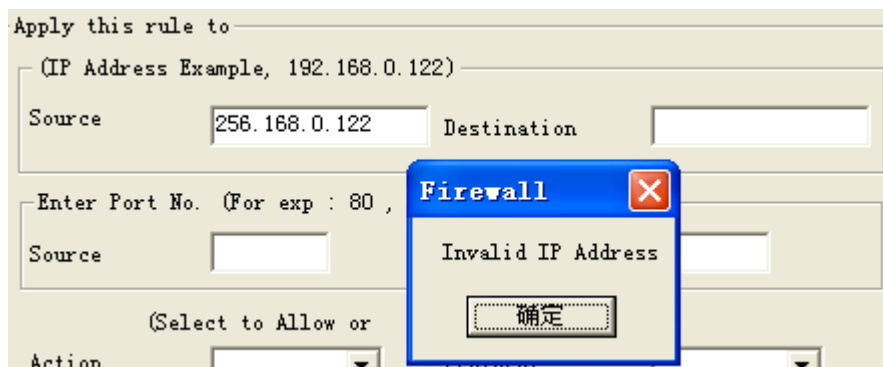


如上图所示，IP 地址形式不对会报错。

对 IP 地址数值的验证

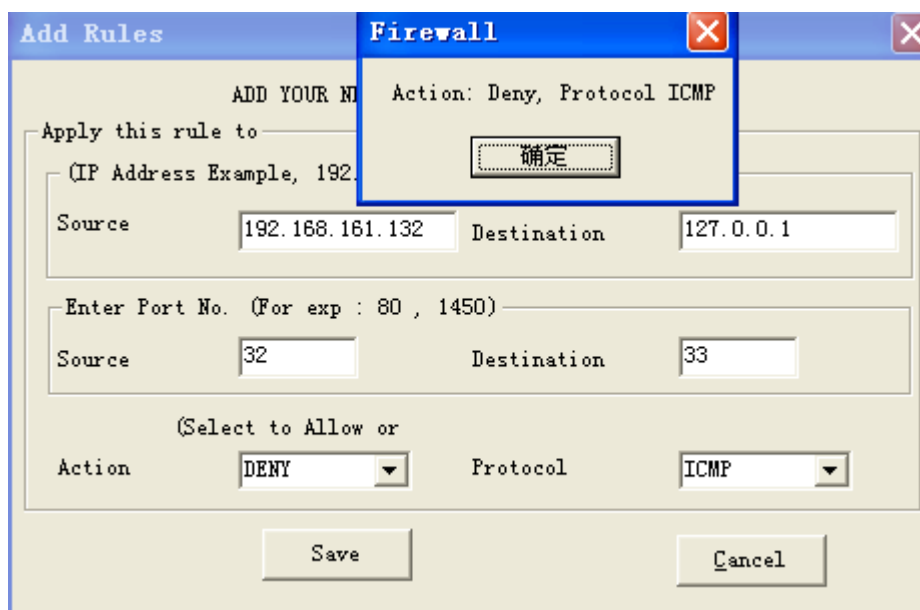


如上图所示，IP 地址数值过小会报错。

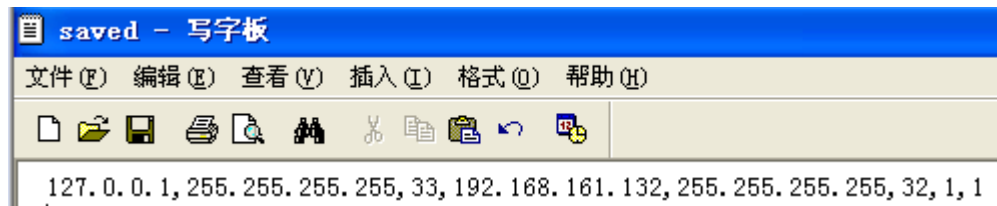


如上图所示，IP 地址数值过小会报错。

对将新增规则保存在文件 saved.rul 的验证。



如上图所示，在 IP 地址通过认证之后并按下 save 按钮之后，弹出显示对目标数据包的信息，代表添加成功。

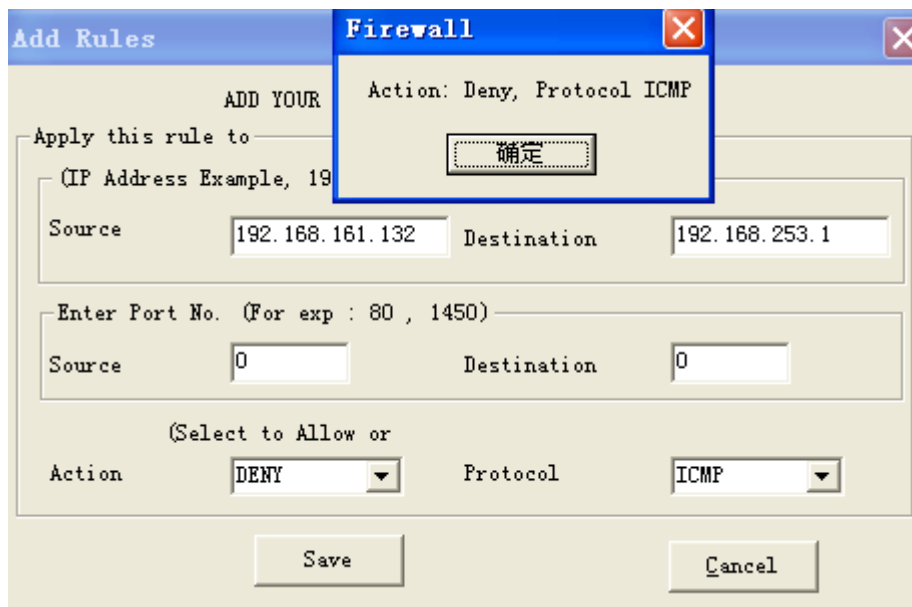


如上图所示，打开 saved.rul 文件，发现新增的规则已被添加进文件末尾，说明操作成功。规则内容为目的地址，目的子网掩码，目的端口，源地址，源子网掩码，源端口，操作类型，协议类型。

判断新增的规则是否被添加进底层驱动。代表我们需要测试满足条件 ICMP 和 TCP 包能否成功被阻塞或通过。

### 1. 验证指定 ICMP 包的阻塞

添加规则，指定源地址为 192.168.161.132（本机 IP），目的地址为 192.168.253.1 的 ICMP 包的操作时 DENY。



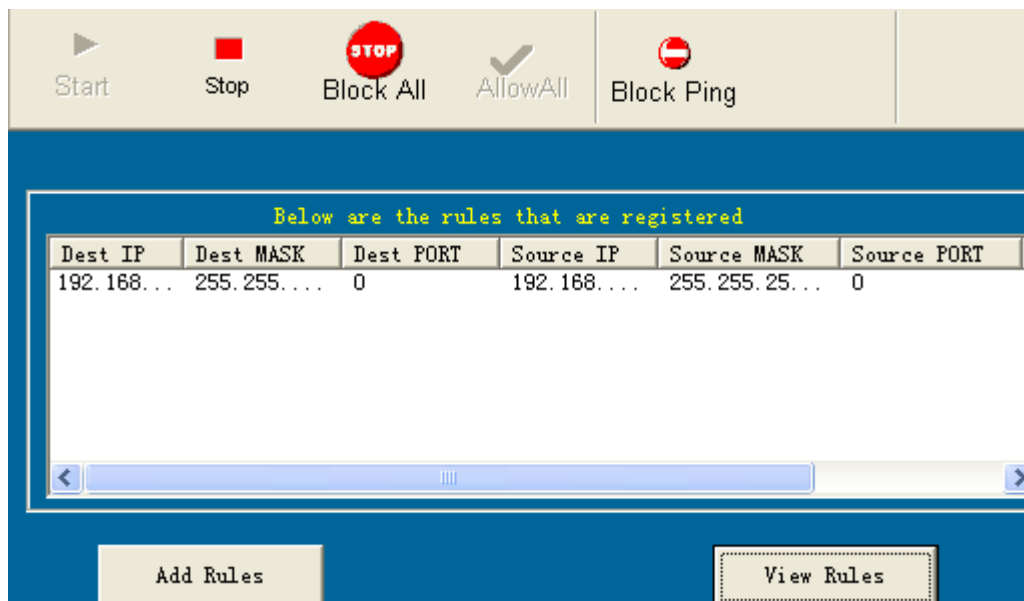
```
C:\Documents and Settings\NS_Lab>ping 192.168.253.1

Pinging 192.168.253.1 with 32 bytes of data:

Destination host unreachable.
Destination host unreachable.
Destination host unreachable.
Destination host unreachable.

Ping statistics for 192.168.253.1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```





实验结果如上图所示，在 cmd 命令行中，ping 主机 192.168.235.1，返回 Destination host unreachable，说明 ICMP 包无法成功发送，阻塞成功。

## 2. 对阻塞特定 TCP 包的检测。

如下图所示，指定目的地址和源地址都是 127.0.0.1，目的端口号为 135，行为为阻塞这个包。



如下图所示，使用 telnet 命令发送 TCP 包到本机主机的 135 端口，失败。说明此条规则阻塞成功。

```
C:\Documents and Settings\NS_Lab>telnet 127.0.0.1 135
正在连接到127.0.0.1...不能打开到主机的连接, 在端口 135: 连接失败
```

### 3. 对允许特定 ICMP 包通过的规则验证。

如下图所示，添加规则，允许源地址为 192.168.1.106(本机 IP)，目的地址为 192.168.161.132 的 ICMP 包通过。

(IP Address Example, 192.168.0.122)

Source: 192.168.1.106 Destination: 192.168.161.132

Enter Port No. (For exp : 80 , 1450)

Source: 0 Destination: 0

(Select to Allow or Deny)

Action: ALLOW Protocol: ICMP

Save Cancel

同时，为了能收到 echo 包，需要将添加另一条规则，源地址和目的地址对换。

(IP Address Example, 192.168.0.122)

Source: 192.168.161.132 Destination: 192.168.1.106

Enter Port No. (For exp : 80 , 1450)

Source: 0 Destination: 0

(Select to Allow or Deny)

Action: ALLOW Protocol: ICMP

Save Cancel

实验结果：

点击 View Rules 按钮载入自定义规则之后，再点击 Block All，再点击 Start。

Start Stop Block All AllowAll Block Ping

Below are the rules that are registered

Dest PORT	Source IP	Source MASK	Source PORT	PROTOCOL	ACTION
0	192.168.1.106	255.255.255.255	0	ICMP	ALLOW
0	192.168.161.132	255.255.255.255	0	ICMP	ALLOW
445	127.0.0.1	255.255.255.255	0	TCP	ALLOW

Add Rules View Rules

如下所示,在 Block All 的情况下, ping 主机 192.168.1.106 依然能够 ping 通,说明规则执行成功。

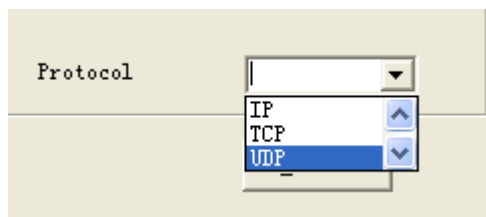
```
C:\Documents and Settings\NS_Lab>ping 192.168.1.106

Pinging 192.168.1.106 with 32 bytes of data:

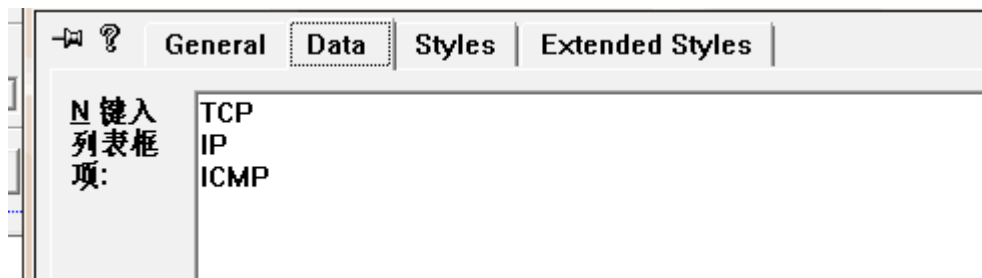
Reply from 192.168.1.106: bytes=32 time=2ms TTL=128
Reply from 192.168.1.106: bytes=32 time=2ms TTL=128
Reply from 192.168.1.106: bytes=32 time=2ms TTL=128
Reply from 192.168.1.106: bytes=32 time=2ms TTL=128

Ping statistics for 192.168.1.106:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 2ms, Maximum = 2ms, Average = 2ms
```

在 Add Rule 界面中的协议中添加一个选项,即 UDP,代表阻塞 UDP 包。



添加方法,在 VC6.0 工程的 Resource 栏点击 Add Rule 控件,再右键点击 Protocol 的图标,到数据一栏,通过 Ctrl+Enter 换行。



在 AddRuleDlg 文件中的 OnAddSave 函数中添加以下代码。与原代码的基础为多加了一个判断:判断 proto 是否为 3。如果为 3 则说明,是 UDP,则需要将 proto 设置为 17,即标准 IP 包头协议所规定的数字。如下图所示。

```

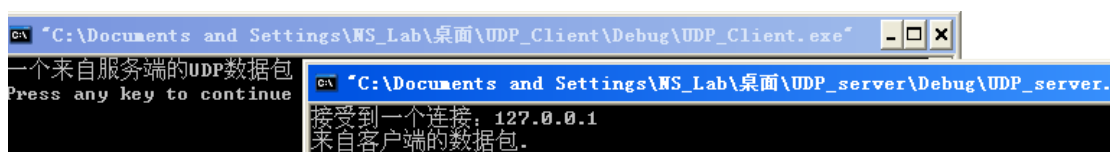
int proto = m_protocol.GetCurSel();
if(proto == 0)//ICMP
    setproto = 1;
if(proto == 1)//IP
    setproto = 0;
if(proto == 2)//TCP
    setproto = 6;
if(proto == 3)//UDP
    setproto = 17;

```

实验结果：

使用 socket 来进行验证。

添加规则前的实验现象：



如上图所示，UDP 服务器可以收到来自 UDP 客户端的包，并且 UDP 客户端也可以收到来自 UDP 服务器的包。

#### 4. 对 UDP 包的阻塞验证。

添加以下规则，环形 IP 地址，源端口为 8888。阻塞满足条件的 UDP 包。

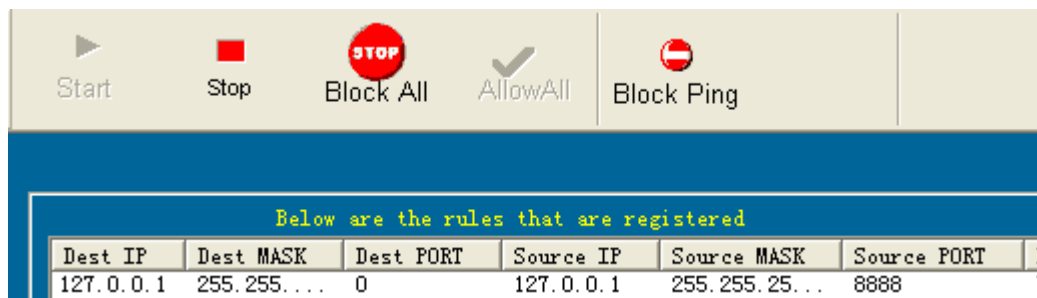
代码如下图所示，客户端向服务器端的 8888 端口发送 UDP 包，则服务器会从 8888 端口回复 UDP 包。

```

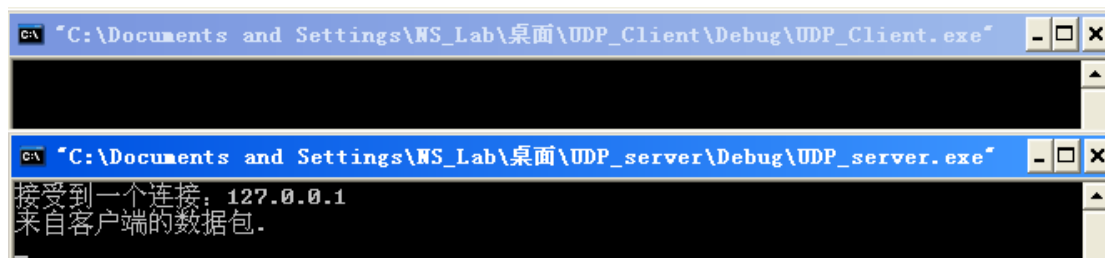
sockaddr_in sin;
sin.sin_family = AF_INET;
sin.sin_port = htons(8888);
sin.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
int len = sizeof(sin);

```

执行规则：

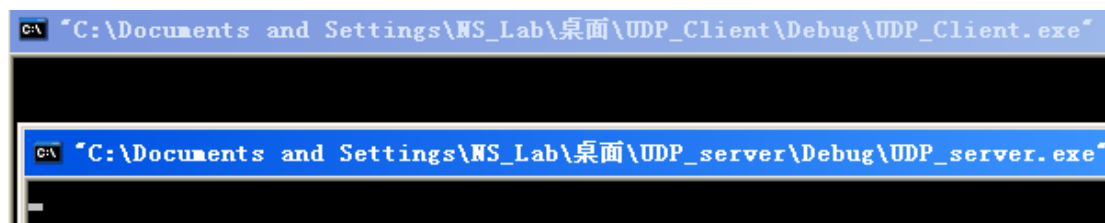


实验结果如下所示，因为客户端并不是从 8888 端口发包，因而不会被阻塞，服务器端可以收到来自客户端的包并打印内容。而服务器端从 8888 端口发包，被防火墙阻塞，客户端不能接收到来自服务器的包，因而什么都没有。规则有效。



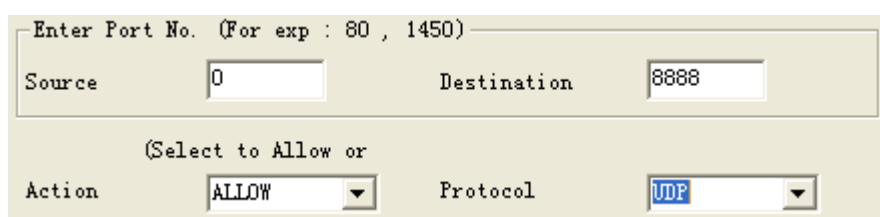
5. 对 UDP 包的允许通过验证。

在未添加允许规则前的 Block All 时 UDP 发包现象：



如上图所示，因为所有的 UDP 包都被阻塞，因而无论是客户端还是服务端都无法接收到对方的包。

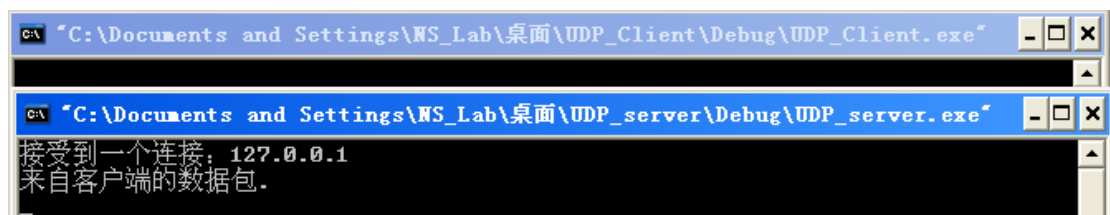
添加规则，允许从目的端口是 8888 端口的 UDP 包通过。



先点击 View Rules 载入自定义规则。再点击 Block All 按钮和 Start。



实验结果如下图所示，客户端往服务端的 8888 端口发 UDP 包，规则允许通过，因而服务器端会收到来自客户端的数据包。然而服务器端发的包不符合要求，则会被阻塞掉。因而客户端无法接收到来自服务端的数据包。说明添加的规则是有效的。



#### 实现功能 4：Allow All，允许所有包通过。

此功能在 OnAllowall 函数中实现。为了使得所有的都能通过，我们只要清除所有的过滤规则就可以。那么，我们可以通过清楚写入底层驱动的所有过滤规则来实现，使用已有的 WriteIO 函数，第一个参数 CLEAR\_FILTER，代表清除过滤规则。

使 Allow All 按钮变灰，其他按钮变亮，并重置记录规则数量的变量。

```
if(m_ipFltDrv.WriteIo(CLEAR_FILTER,NULL,0) != DRV_ERROR_IO)
{
    allow = FALSE;
    block = TRUE;
    ping = TRUE;
    _rows = 1;
}
```

实验结果：

点击 Allow All 之前的现象：

如下所示，console 中有显示自定义的规则。

Below are the rules that are registered					
Dest IP	Dest MASK	Dest PORT	Source IP	Source MASK	Source PORT
127.0.0.1	255.255.255.0	0	127.0.0.1	255.255.255.0	0
127.0.0.1	255.255.255.0	0	127.0.0.1	255.255.255.0	445
192.168.1.0	255.255.255.0	0	192.168.1.0	255.255.255.0	0
192.168.1.0	255.255.255.0	0	192.168.1.0	255.255.255.0	0
127.0.0.1	255.255.255.0	445	127.0.0.1	255.255.255.0	0

如下图所示，因为自定义规则的阻塞不能发送 TCP 包。

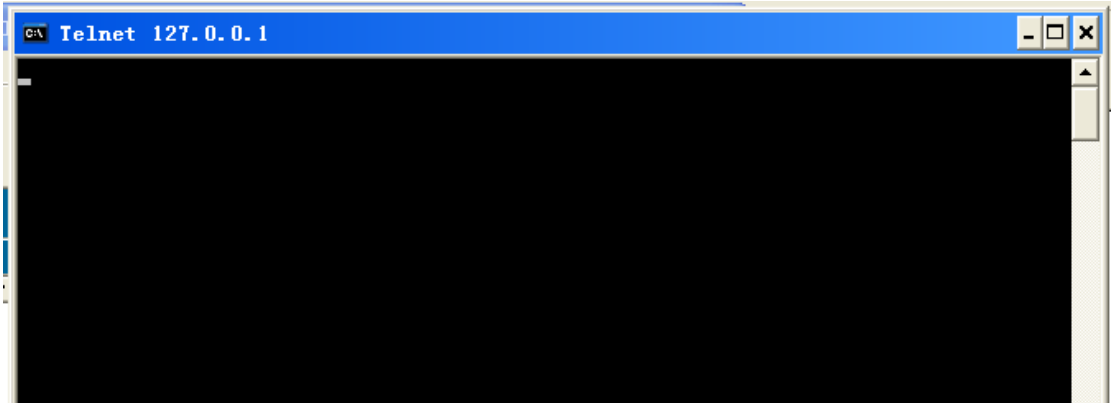
```
C:\Documents and Settings\NS_Lab>telnet 127.0.0.1 445
正在连接到127.0.0.1...不能打开到主机的连接， 在端口 445：连接失败
```

点击 Allow All 之后的现象：

如下图所示，console 中所有的规则都被清空。



如下图所示，telnet 环形主机能够成功登陆，说明阻塞规则不再有效。



## 实现功能 5：显示规则至用户界面。

显示规则的功能在 `ImplementRule` 函数中完成，`ImplementRule` 在 `OnViewrules` 函数中进行了调用，指点击了 View Rules 按钮后的行为，它的功能是显示所有的规则在前端界面上（利用已有的跟 MFC 有关的用于前端交互的函数，如 `AddItem` 函数）。

首先打开 `saved.rul` 文件以便获取规则，使用已有的 `CreateFile` 函数。

```
_hFile = CreateFile("saved.rul",           // name of the file
                   GENERIC_READ | GENERIC_WRITE, // open as readable and writeable
                   FILE_SHARE_READ | FILE_SHARE_WRITE, // shareable as read only
                   NULL,
                   OPEN_EXISTING,           // open only if it exist
                   NULL,
                   NULL);
```

成功打开文件之后则开始读取相应内容，使用已有的 `ReadFile` 函数，返回值为 `bool` 值。用 `char` 类型的变量储存每次读取的值。因为 `char` 的大小为一个 `byte`，即可以每次都只读取一个字符，会剩下一次读取多个 `byte` 的麻烦，如处理换行等。每次一次读取一个 `byte` 都添加至已有的字符串中。读取到文件末尾时，去除换行符。

得到整条规则之后，将内容显示在前端，并将规则添加进底层驱动中。功能在 `ParseToIp` 函数中完成。文件全部读取完之后，关闭文件，用 `CloseHandle` 函数。

```
BOOL bResult;

do{
    bResult = ReadFile(_hFile,&data,1,&nbytesRead,NULL);

    if((data != '\n'))//end of the line
    {
        _buff = _buff + data;
    }
    else
    if((bResult && nbytesRead) !=0)//if reading the file has no mistakes
    {
        _buff.Remove('\n');
        ParseToIp(_buff);//show the rule in the console
    }
}
```



```

        _buff = "";
    }
}while((bResult && nbytesRead) !=0);

CloseHandle(_hFile);

```

ParseToIp 函数：

因为需要显示的值有八个，分别是目的地址，目的子网掩码，目的端口，源地址，源子网掩码，源端口，操作类型，协议类型。我们用大小为八的字符数组来进行存储。

首先需要去除整条字符串中的“，”并将他们分别存储进字符数组中。

```

int    _pos,_prevpos = 0;
for(; count < 8; count++)
{
    if(count < 7)//count the value
    {
        _pos = str.Find(',',_prevpos + 1);
        if((count > 0))
        {
            _str[count] = str.Left(_pos);          //,A,B -> ,A
            _str[count].Delete(0,_prevpos + 1);//,A -> A
        }
        else{
            if(count == 0)//A,no need to delete
                _str[count] = str.Left(_pos);
        }
    }
    else//the last entry
    {
        _str[count] = str.Right(1);//1 owing to the rule
    }

    _prevpos = _pos;
}

```

将储存的值添加至前端，使用 AddItem 函数。对于协议和操作，根据数值来显示相应的英文指示。

```

AddItem(0,0,(LPCTSTR)_str[0]);
AddItem(0,1,(LPCTSTR)_str[1]);
AddItem(0,2,(LPCTSTR)_str[2]);
AddItem(0,3,(LPCTSTR)_str[3]);
AddItem(0,4,(LPCTSTR)_str[4]);
AddItem(0,5,(LPCTSTR)_str[5]);
int _proto = atoi((LPCTSTR)_str[6]);

```

```

CString    proto;
if(_proto == 0)
    proto = "ANY";
if(_proto == 1)
    proto = "ICMP";
if(_proto == 6)
    proto = "TCP";
if(_proto == 17)
    proto = "UDP";
AddItem(0,6,((LPCTSTR)proto));
int _drop = atoi((LPCTSTR)_str[7]);
if(_drop == 0)
    AddItem(0,7,"ALLOW");
if(_drop == 1)
    AddItem(0,7,"DENY");

```

添加之后，将记录现有过滤规则数量的变量\_rows 的数量加一。

```
_rows = _rows + 1;
```

将此条规则添加进底层驱动中。

```

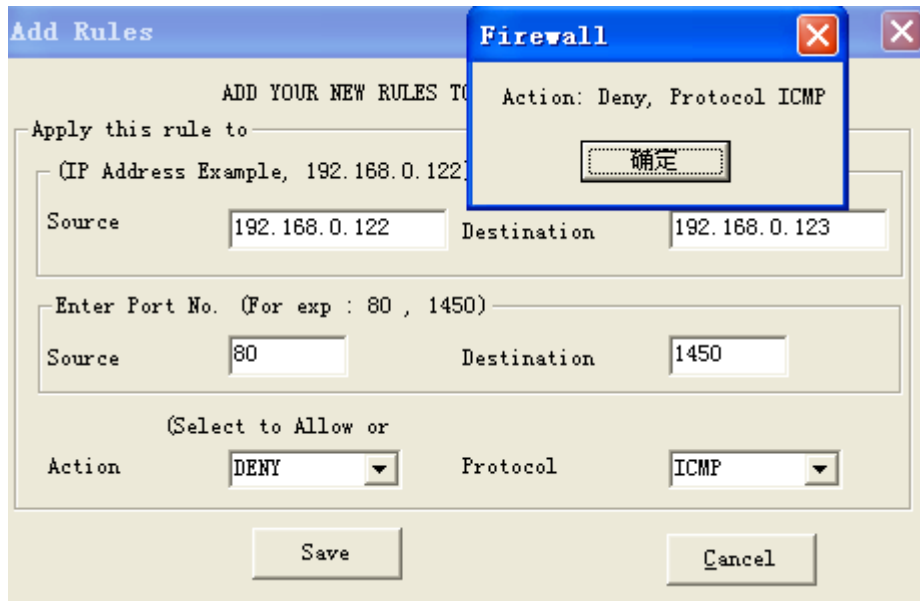
IPFilter    ip1;
ip1.destinationIp    = inet_addr((LPCTSTR)_str[0]);
ip1.destinationMask  = inet_addr((LPCTSTR)_str[1]);
ip1.sourceIp         = inet_addr((LPCTSTR)_str[3]);
ip1.sourceMask       = inet_addr((LPCTSTR)_str[4]);
ip1.sourcePort       = htons(atoi((LPCTSTR)_str[5]));
ip1.protocol         = atoi((LPCTSTR)_str[6]);

int    drop;
drop = atoi((LPCTSTR)_str[7]);
if(drop == 0)
{
    ip1.drop = FALSE;
}
if(drop == 1)
{
    ip1.drop = TRUE;
}
}
m_Addrule.AddFilter(ip1);

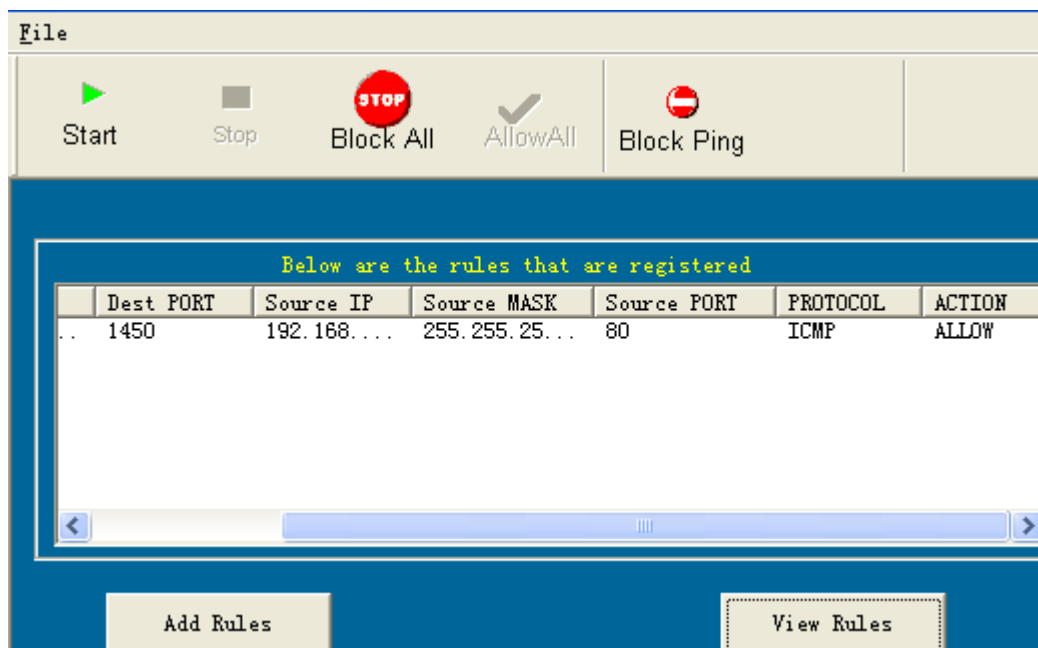
```

实验结果：

添加一条新的规则，如下图所示。



点击用户界面 View Rules 按钮，显示新增加的规则的内容，如下图所示。



**附加功能：Block UDP。**

增加一个按钮，Block UDP，作用是阻塞所有的 UDP 包。

在 Resource 栏的 FIRE\_FORM 控件中添加一个按钮，赋予 ID IDC\_BLOCKUP。



在 fireView 文件中的 BEGIN\_MESSAGE\_MAP 函数中添加如下语句，将 OnBlockUDP 函数与 Block UDP 按钮的响应连系起来。

```
BEGIN_MESSAGE_MAP(CFireView, CFormView)
    ON_BN_CLICKED(IDC_ADDRULE, OnAddrule)
    ON_BN_CLICKED(IDC_BLOCKUDP, OnBlockUDP)
    ON_BN_CLICKED(IDC_START, OnStart)
    ON_BN_CLICKED(IDC_BLOCKPING, OnBlockping)
```

在 OnBLOCKUDP 函数中，将协议号修改为 17，代表协议为 UDP 协议。其余都置为零，代表全部。如下图所示。

```
void CFireView::OnBlockUDP()
{
    IPFilter    IPflt;

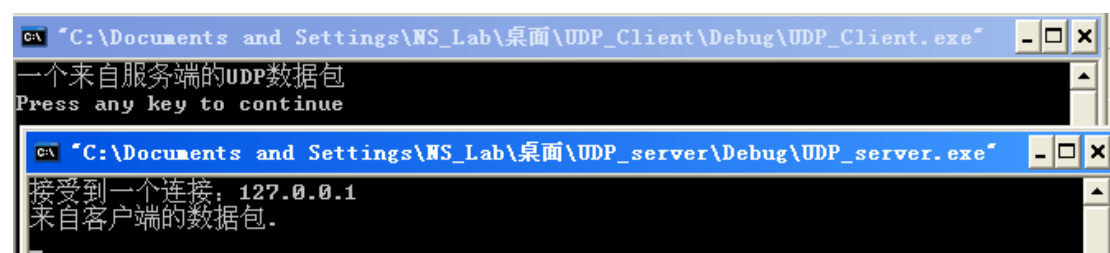
    IPflt.protocol      = 17; // 1 for UDP Protocol
    IPflt.destinationIp = 0; // inet_addr("127.0.0.1"); // all destinations
    IPflt.destinationMask = 0; // inet_addr("255.255.255.255"); // all destination masks
    IPflt.destinationPort = 0; // 0 for all ports
    IPflt.sourceIp        = 0; // 0 means drop all packets irrespective of source
    IPflt.sourceMask      = 0;
    IPflt.sourcePort      = 0; // 0 means from any source port
    IPflt.drop            = TRUE;

    m_Addrule.AddFilter(IPflt);
    ping = TRUE;
    allow = TRUE;
    block = TRUE;
}
```

实验结果：

点击 Block UDP 前的结果：

如下图所示，使用 socket 来发送 UDP 包。UDP 客户端向 UDP 服务端发送一个 UDP 包，服务端将内容打印出来。并返回客户端一个包，客户端将内容打印出来。

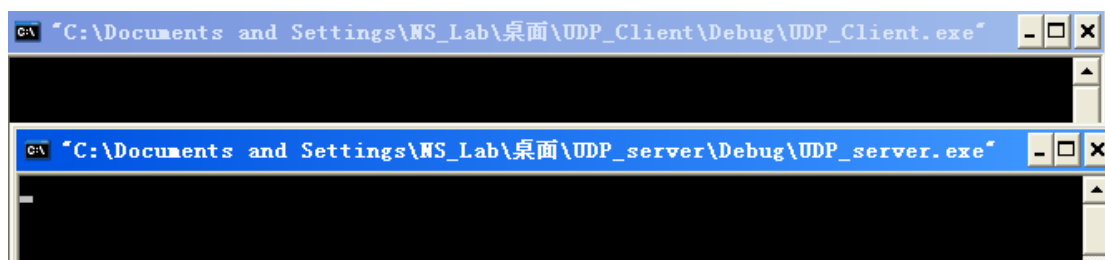


点击 Block UDP 后的结果：

如下图所示，点击 Block UDP 按钮之后，再点击 Start 按钮。



结果如下图所示。UDP 客户端无法成功向服务端发送 UDP 包，因而服务端也无法返回给客户端包。因而客户端和服务端什么都不能收到，也无法打印任何内容。说明 Block Ping 成功。



## 4. 实验心得与感想

此次 Project 内容较多，有如下感想与收获。

1. 对于自定义规则的生效需要特定的执行顺序。之前每次都先 Block All 再 View Rules，却发现每次都不能正常执行规则。后来发现需要先 View Rules 再 Block All 才能正常执行。
2. CString 的很多自带函数对字符串的处理很有帮助。因为 VC6.0 或者 MFC 框架的限制，每次我想加一个库或者采用新的变量编译就会报错。因而只能用现成的变量来进行数据的处理。

3. 在使用 socket 的时候，需要两个程序，一个用作 server，一个用作 client。  
在使用 UDP 或者 TCP 的 socket 的时候，如果两台主机系统不同，写两份不一样的代码非常麻烦的话，可以使用环形 IP 地址 127.0.0.1，可达到一样的效果。
4. telnet 发包的时候只能指定目的端口，而不能指定发包端口。原因是指定发包端口太容易被攻击者攻击。而 telnet 的默认端口一般是 23。但是有时候会碰到 23 端口没开放的情况。这时候可以用 netstat 来查看可使用的端口。
5. 如果要用 ping 来检测阻塞某个特定 ICMP 包的规则是否生效。我们需要阻塞发出去的包才会收到 Destination Host Unreachable 的回复。如果阻塞的是返回的包那么会收到 request time 超时的回复。
6. 不能够直接在 saved.rul 文件中修改规则。如果在 saved.rul 中直接修改规则会发生意想不到的错误。比如 DENY 变成 ALLOW，或者协议的类型改变等等。推测是会间接改变了规则写入驱动的操作。
7. 设计检测方法比单纯写代码更加麻烦。因为你需要定义发送特定端口的数据包。网上有很多方法可能并不适合当前的系统或者没有相应的程序及服务，能够使用之后也不一定能成功还要 Debug。所有耗时最长。
8. Allow All 被点击之后虽然 console 中的规则被清零了。但是文件 save.rul 中的内容并没有被清零。如果连续点击两次 View Rules，则会不断地在原来显示的规则的基础上添加同样的规则。可以使用 m\_cResult 结构体中的 DeleteAllItems 函数。
9. 之前一直以为做一个可以与用户的可视程序很困难。这次在做 Project 的过程中发现可以用 Windows 的 MFC 架构使得前端和后台逻辑可以轻易连接。

## **References :**

1. <http://wenda.tianya.cn/question/4187e48fda58d9c6>
2. <http://blog.csdn.net/xgx198831/article/details/8464552>
3. <http://www.cnblogs.com/Caiqinghua/archive/2009/02/16/1391190.html>
4. <http://www.codeproject.com/Articles/5602/Simple-Packet-Filter-Firewall>