**程序源代码**

#include<Winsock2.h>

#pragma comment(lib, "Ws2\_32.lib") //使用ntohs()函数,转换2B/4B的数据

#pragma comment(lib, "wpcap.lib")

#include"pcap.h" //需要另外导入，具体步骤见Winpcap使用说明

#include<fstream.h>

#include<iomanip.h> //格式化输出

#include<conio.h> //使用\_getch()

struct arppkt{

unsigned short hdtyp; //硬件地址

unsigned short protyp; //协议类型

unsigned char hdsize; //硬件地址长度

unsigned char prosize; //协议地址长度

unsigned short op; //操作值

u\_char smac[6]; //源MAC地址

u\_char sip[4]; //源IP地址

u\_char dmac[6]; //目的MAC地址

u\_char dip[4]; //目的IP地址

};

//自定义处理包函数

//pcap\_pkthdr是winpcap加入的

//pkt\_data表示MAC帧的起始位置

//out是输入流

void packet\_handler(const pcap\_pkthdr \*header,const u\_char \*pkt\_data,ostream& );

void main(int argc,char \*argv[])

{

if(argc!=2)

{

cout<<"usage:arpparse logfilename"<<endl;

cout<<"press anykey to continue."<<endl; //参数不对，进行提示

\_getch();

return;

}

pcap\_if\_t \*alldevs; //网络设备结构

pcap\_if\_t \*d;

pcap\_t \*adhandle;

char errbuf[PCAP\_ERRBUF\_SIZE]; //错误信息

u\_int netmask; //子网掩码

char packet\_filter[] = "ether proto \\arp"; //过滤,选择ARP协议

struct bpf\_program fcode;

struct pcap\_pkthdr \*header;

const u\_char \*pkt\_data;

//获取网络设备列表

if (pcap\_findalldevs(&alldevs, errbuf) == -1)

{

cout<<"Error in pcap\_findalldevs:"<<errbuf;

return;

}

//选择一个Ethernetka卡

for(d=alldevs;d;d=d->next)

{

//以混杂模式打开网卡,接受所有的帧

if((adhandle= pcap\_open\_live(d->name,1000,1,300,errbuf)) == NULL)

{

cout<<"\nUnable to open the adapter.";

pcap\_freealldevs(alldevs); //释放设备列表

return;

}

if(pcap\_datalink(adhandle) == DLT\_EN10MB && d->addresses != NULL)

break;

}

if(d==NULL)

{

cout<<"\nNO interfaces found! Make sure winpcap is installed.\n";

return;

}

//获得子网掩码

netmask=((sockaddr\_in \*) (d->addresses->netmask))->sin\_addr.S\_un.S\_addr;

//调试过滤器，只捕获ARP包

if(pcap\_compile(adhandle, &fcode,packet\_filter,1,netmask) <0)

{

cout<<"\nUnable to compile the packet filter. Check the syntax.\n";

pcap\_freealldevs(alldevs);

return;

}

//设置过滤器

if (pcap\_setfilter(adhandle, &fcode)<0)

{

cout<<"\nError setting the filter.\n";

pcap\_freealldevs(alldevs);

return;

}

cout<<"\t\tlistening on "<<d->description<<"..."<<endl<<endl;

ofstream fout(argv[1],ios::app); //日志记录文件

//加入日期记录

time\_t t;

time(&t);

fout.seekp(0,ios::end);

if(fout.tellp()!=0)

fout<<endl;

fout<<"\t\tARP request(1)/replay(2) on "<<ctime(&t);

cout<<"sour IP Addr"<<" "<<"Sour MAC Address"

<<" "<<"Des IP Addr"<<" "<<"Des MAC Address"

<<" "<<"OP"<<" "<<"Time"<<endl;

fout<<"Sour IP Addr"<<" "<<"Sour MAC Address"

<<" "<<"Des IP Addr"<<" "<<"Des MAC Address"

<<" "<<"OP"<<" "<<"Time"<<endl;

//释放设备列表

pcap\_freealldevs(alldevs);

//开始捕获MAC帧

int result; //时间到返回结果

while((result=pcap\_next\_ex(adhandle, &header, &pkt\_data)) >= 0)

{

if(result==0)

continue;

packet\_handler(header,pkt\_data,cout); //解析ARP包，输出结果

packet\_handler(header,pkt\_data,fout); //输出到文件

}

}

void packet\_handler(const pcap\_pkthdr \*header, const u\_char \*pkt\_data, ostream& out)

{

//从截获的数据帧中找到arp包头的位置

arppkt\* arph = (arppkt \*) (pkt\_data +14); //14为Ethernet帧头的长度

//输出源IP地址

for(int i=0;i<3;i++)

out<<int(arph->sip[i])<<'.';

out.setf(ios::left);

out<<setw(3)<<int(arph->sip[3])<<" ";

out.unsetf(ios::left);

//输出源MAC地址

char oldfillchar=out.fill('0');

out.setf(ios::uppercase); //MAC地址以分开'-'的大写字符表示

for(i=0;i<5;i++)

out<<hex<<setw(2)<<int(arph->smac[i])<<'\_';

out<<hex<<setw(2)<<int(arph->smac[5])<<" ";

out.fill(oldfillchar);

//输出目的的IP地址

out.unsetf(ios::hex|ios::uppercase);

for(i=0;i<3;i++)

out<<int(arph->dip[i])<<'.';

out.setf(ios::left);

out<<setw(3)<<int(arph->dip[3])<<" ";

out.unsetf(ios::left);

//输出目的MAC地址

out.fill('0');

out.setf(ios::uppercase);

for(i=0;i<5;i++)

out<<hex<<setw(2)<<int(arph->dmac[i])<<'\_';

out<<hex<<setw(2)<<int(arph->dmac[5])<<" ";

out.fill(oldfillchar);

out.unsetf(ios::hex|ios::uppercase);

out<<ntohs(arph->op)<<" "; //输出的操作类型,注意网络字节间的转换

struct tm \*ltime; //时间

ltime=localtime(&header->ts.tv\_sec);

out.fill('0');

out<<ltime->tm\_hour<<':'<<setw(2)<<ltime->tm\_min<<':'<<setw(2)<<ltime->tm\_sec;

out.fill(oldfillchar);

out<<endl;

}