网络攻防实验报告

姓名：蔡真真　　学号：2017280186*****　　班级：网安****班

1 实验准备

1.1 实验目的

开发一个网络嗅探器，重点对 TCP、UDP、ARP、IGMP、ICMP 等数据包进行分析，实现捕捉前过滤、数据包统计、流量统计等功能。

1.2 实验环境

集成开发环境：Microsoft Visual Studio 2012
编程语言：C++
程序框架：MFC
软件包：WinPcap V4.1.3、skin++

2 网络嗅探器

2.1 网络嗅探器总体设计

网络嗅探器的整体结构按功能分为 3 个部分,自底向上分别是数据捕获模块、协议解析模块和用户显示模块。网络嗅探器的总体结构如图 1 所示。

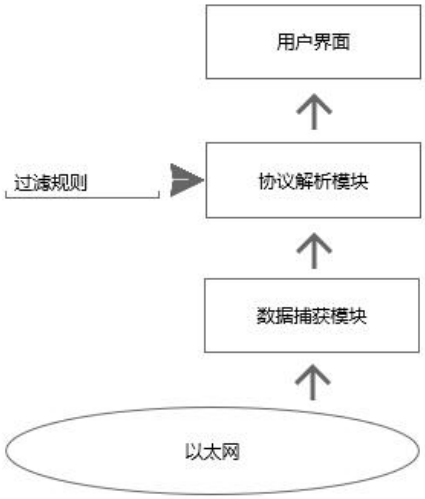

图 1 网络嗅探器的总体结构

2.2 数据捕获模块的设计与实现

数据捕获模块的主要功能是进行数据采集,这是整个系统的基础和数据来源。程序使用 Winpcap 来捕获网络中原始数据包。具体数据包捕获流程如图 2 所示。

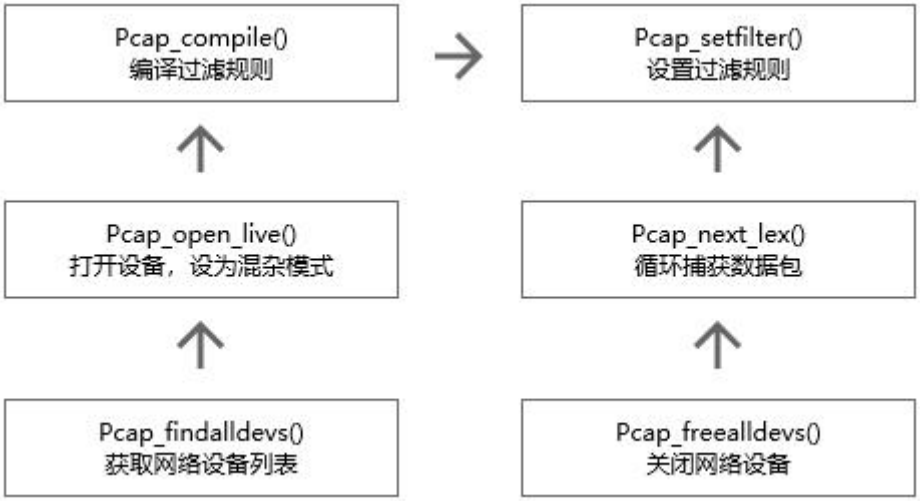

图 2 数据包捕获流程图

2.2.1 获得本地网络驱动器列表

获取一个已经绑定的网卡列表,然后 Winpcap 对捕获网络数据端口进行设定。通过 pcap 引擎找出并设定监听的网络接口。Winpcap 提供了 pcap_findalldevs_ex()函数,这个函数返回一个指向 pcap_if 结构的链表,其中的每一项都包含了一个已经绑定的网卡的全部信息。代码如图 3 所示。

```
/* 获取本地机器设备列表 */
if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL /* auth is not needed */, &alldevs, errbuf) ==

{
    fprintf(stderr,"Error in pcap_findalldevs_ex: %s\n", errbuf);
    exit(1);
}
```

图 3

2.2.2 打开网卡准备捕获数据包

获得网卡的信息后就可以按数据捕获的要求打开网卡。打开网卡的功能是通过 pcap_open_live()来实现的。snaplen 制定要捕获数据包中的哪些部分，flag 是用来指示适配器是否要被设置成混杂模式，to_ms 指定读取数据的超时时间，以毫秒计。代码如图 4 所示。

```
/* 打开设备 */
    if ( (adhandle= pcap_open(d->name,          // 设备名
                              65536,            // 65535保证能捕获到不同数据链路层上的每个数据包的全部内容
                                 PCAP_OPENFLAG_PROMISCUOUS,    // 混杂模式
                              1000,             // 读取超时时间
                              NULL,             // 远程机器验证
                              errbuf            // 错误缓冲池
                              ) ) == NULL)
    {
        fprintf(stderr,"\nUnable to open the adapter. %s is not supported by WinPcap\n", d->name);
        /* 释放设备列表 */
        pcap_freealldevs(alldevs);
        return -1;
    }
```

图 4

2.2.3 数据包的过滤设定

数据包过滤处理是嗅探技术中的难点和重点,Winpcap 提供了最强大的数据流过滤引擎。它采用了一种高效的方法来捕获网络数据流的某些数据且常常和系统的捕获机制相集成。过滤数据的函数是 pcap_compile()和 pcap_setfilter()来实现的。pcap_compile()它将一个高层的布尔过滤表达式编译成一个能够被过滤引擎所解释的低层的字节码，pcap_setfilter()将一个过滤器与内核捕获会话向关联。当 pcap_setfilter()被调用时，这个过滤器将被应用到来自网络的所有数据包，并且，所有的符合要求的数据包(即那些经过过滤器以后，布尔表达式为真的包)，将会立即复制给应用程序。代码如图 5 所示。

```
compile the filter
    if (pcap_compile(adhandle, &fcode, "ip and tcp", 1, netmask) < 0)
    {
        fprintf(stderr,"\nUnable to compile the packet filter. Check the syntax.\n");
        /* 释放设备列表 */
        pcap_freealldevs(alldevs);
        return -1;
    }

set the filter
    if (pcap_setfilter(adhandle, &fcode) < 0)
    {
        fprintf(stderr,"\nError setting the filter.\n");
        /* 释放设备列表 */
        pcap_freealldevs(alldevs);
        return -1;
    }
```

图 5

2.2.4 捕获数据包

使用 pcap_next_ex()从网络接口中读取一个数据包,该函数第一个参数是接口句柄,后两个参数由函数返回,分别为数据包的相关信息和数据包本身。函数返回 1 表示正常接收一个数据包,返回 0 表示超时,-1 表示发生错误。每捕获到一个数据包,就调用 PacketHandler()函数对数据包进行后续解析处理。

2.3 协议解析模块的设计与实现

该模块的主要功能就是对捕获的数据包按照数据链路层、网络层、传输层和应用层的层

次结构自底向上进行解析,最后将解析结果显示输出。

以解析 UDP 数据包为例,首先设置 UDP 过滤,用这种方法确保 packet_handler()只接收到基于 IPV4 的 UDP 数据。同时,定义两个数据结构来描述 IP 和 UDP 的头部信息,packet_handler()用这两个结构来定位头部的各种字段。开始捕获之前,首先要用 pcap_datalink()来检查 MAC 层,所以程序只能够工作在 Ethernetnetworks 上,再次确保 MAC 头为 14bytes。MAC 头之后是 IP 头,可以从中提取出了目的地址。IP 之后是 UDP,在确定 UDP 的位置时有点复杂,因为 IP 的长度以版本的不同而不同,所以用头长字段来定位 UDP,一旦确定了 UDP 的起始位置,就可以解析出原和目的端口。代码如图 6 所示。

```
case UDP:
    {
        m_udpCount++;
        udp_header *uh;
        const u_char *udp_data;
        udp_data = ip_data+ip_len;
        uh = (udp_header *)udp_data;
        if(ntohs( uh->dport ) == DNS || ntohs( uh->sport ) == DNS)
        {
            m_list1.SetItemText(nCount,5,_T("DNS"));
            m_dnsCount++;
        }
        else
            m_list1.SetItemText(nCount,5,_T("UDP"));
        if(ntohs( uh->dport ) == 0x1F40 || ntohs( uh->dport ) == 0x1F41 ||
           ntohs( uh->sport ) == 0x1F40 || ntohs( uh->sport ) == 0x1F41)        //QQ流量为UDP端口8000或8001
            m_qqCount++;
        if(ntohs( uh->dport ) == 0xBB9 || ntohs( uh->dport ) == 0xBBA ||
           ntohs( uh->sport ) == 0xBB9 || ntohs( uh->sport ) == 0xBBA)      //uc流量为UDP端口3001或3002
            m_ucCount++;
        break;
    }
```

图 6

2.4 用户界面

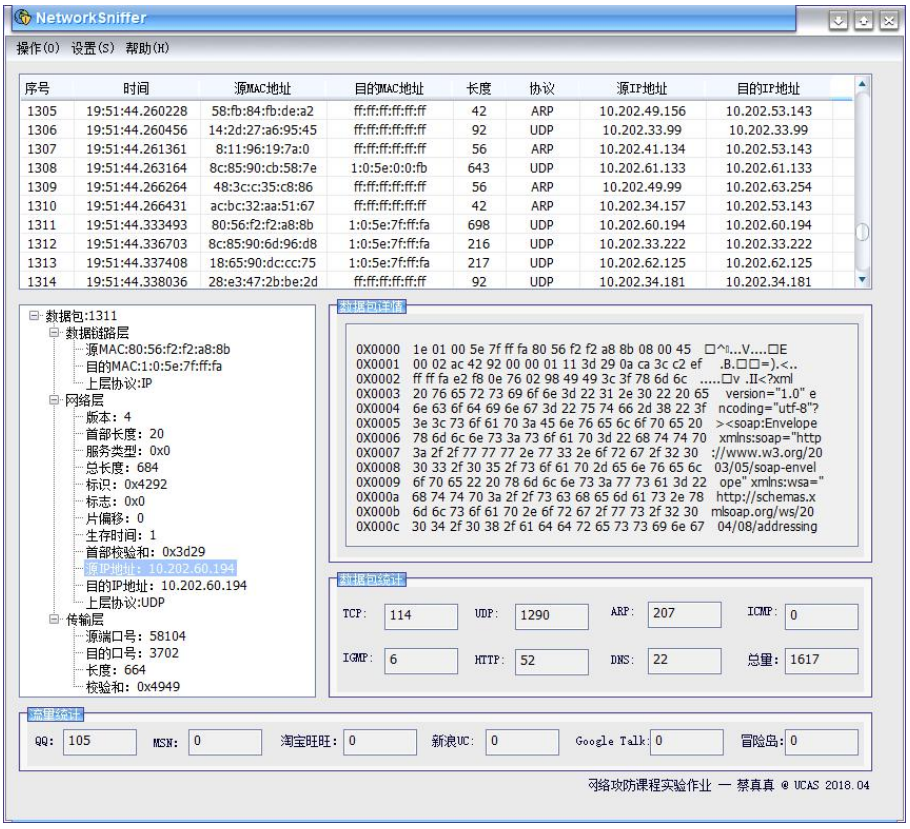

图 7 实验结果界面示意图

如图 7 所示，用户界面包括菜单栏模块、数据包总体信息栏模块、单个数据包协议栈信息栏模块、单个数据包 16 进制及 ASCII 码显示栏模块、数据包统计模块、流量统计模块等六大模块。

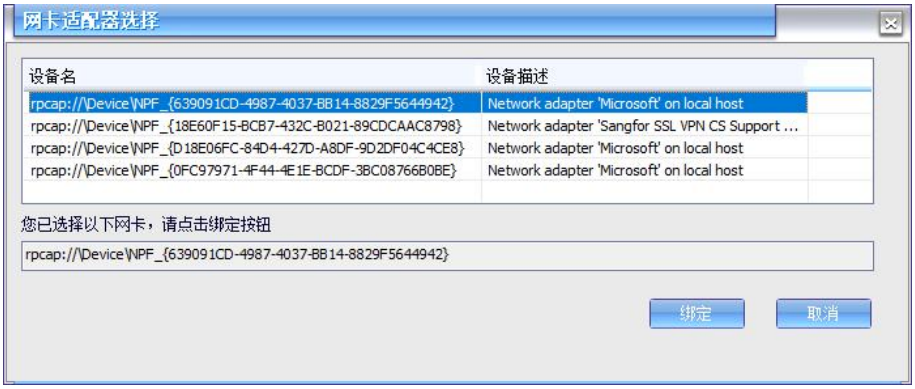

图 8 网卡适配器选择界面

如图 8 所示，点击菜单栏中的设置-网卡适配器选择，显示网卡列表，可选择其中一个网卡，默认设置为混合模式。

图 9 过滤规则设置界面

如图 9 所示，点击菜单栏中的设置-过滤器，可以选择要过滤的协议。

图 10 流量统计界面

流量统计模块，实现原理是对于某些常用的软件，都是通过固定端口进行数据传输的，比如 QQ 默认采用 UDP 通讯方式，端口 8000，8001。如果 UDP 的两个端口不通，会自动转换到 TCP80 端口或者 TCP443 端口进行通讯。QQ 同时也支持 HTTP 代理模式及 SOCK5 代理模式。阿里旺旺采用 TCP 通讯方式，默认登录端口为 16000，当 16000 端口不通时，则跳转到 443 端口进行通讯。MSN 采用 TCP 端口号 1683，googletalk 采用 TCP 端口号 5222，冒险岛游戏客户端采用 TCP 端口 8086、8484、8585，而新浪 UC 通讯器客户端采用 UDP 端口号 3001 和 3002.所以通过对固定端口的监听，可以知道哪些软件产生了流量。

3 实验中遇到的问题以及解决方法

3.1 多线程问题

编写代码时未考虑多线程问题，直接在 device_OnPacketArrival，即捕获到数据触发的事件的函数中添加操作，将捕获的数据添加入 Listview 中，这样，当网络中流过的数据包很多时就会出现异常。因此，学习并添加了多线程，同时将捕获的数据包放入一个 List 列表中，将数据的捕获和数据的显示分开放入不同的线程中，从而解决了问题。

3.2 过滤规则组装

WireShrank 的过滤采用的是过滤表达式，如“arpor（ipandicmp）”，但对于普通用户，理解并编写这样的表达式是需要一定的基础的，为了方便用户的使用，本项目采用复选框，通过勾选相应的协议实现过滤规则组装，这就需要充分理解过滤规则。

3.3 数据的 ASCII 码显示问题

将 16 进制和 ASCII 码同时显示在一个 RichTextBox 中，发现显示结果经常出错，通过检索发现，ASCII 码中有很多非显示的字符。因此，参考 Wireshark 的处理方法，在显示上将小于 32 或大于 126 的 byte 用“.”表示。处理代码如图 11 所示。

```
for (unsigned short i = 0; i < pkt_header->caplen ; i++)
{
    CString hex;
    if ( (i % 16) == 0)
    {
        hex.Format(_T("\x0d\x0a 0X%04x   "),nCount);
        nCount++;
        if( i != 0)
        {
            strHex +=_T("   ")+strText ;
            strText = _T("");
        }
        strHex += hex;
    }
    hex.Format(_T("%2.2x "),pkt_data[i-1]);
    strHex += hex;
    if(pkt_data[i-1] <= 127 && pkt_data[i-1] >= 0)
        hex.Format(_T("%c"),pkt_data[i-1]);
    else
        hex = _T(".");
    strText += hex;
}
if( strText !=_T(""))
    strHex += strText;
m_edit1.SetWindowText(strHex);
```

图 11 ASCII 码显示处理代码

3.4 捕捉后过滤实现难题

在单个数据包协议栈信息栏模块选中某个信息，单个数据包 16 进制及 ASCII 码显示栏模块会高亮显示相应的信息。在 Treeview 中也有 NodeMouseClick 事件，RichTextBox 中关于选中字体也有 selecttext.color 属性，最大的实现问题是如何逆向查找选中的字段。

3.5 字节顺序转换问题

存储完捕获的数据包，就可以利用 pkt_data 的信息来分析数据包，在分析数据包中的数据时注意，数据包的数据是按照小端模式存储的，对于大于 1 字节的内容需要用 noths 或是 nothl 进行转化。举一个例子，比如数字 0x12345678 在内存中的表示形式为：0x78|0x56|0x34|0x12，这时就需要 noths 来转化为正确的顺序。

3.6 使用 Unicode 编码问题

项目开发过程中使用的是 Unicode 编码，给项目添加 Skin++皮肤，因为编码问题而报错。查找原因，发现需要在皮肤插件的头文件中，修改变量，把 TCHAR 变量修改为 CHAR，并强制转换格式。

4 收获与体会

本次实验最大的收获就是学习使用 Winpcap 这个网络分包抓取工具，并且理解了网络嗅探器的基本工作原理。网络嗅探器工作在网络环境的底层, 拦截所有正在网络上传送的数据, 并且通过相应的解析处理,可以实时分析这些数据的内容,进而分析所处的网络状态和整体拓扑布局。本次实验使用通过 C++语言编程实现了网络嗅探器的基本功能，可以对局域网中的数据包进行捕获和分析。在开发工程中，遇见了许多难题，参考了很多相同功能的代码，特别是在流量统计模块，最好的解决方式是在应用层进行分析，例如对源 IP 和目的 IP 进行统计，但由于时间有限，IP 的查找和匹配有一定的开发难度，因此采用了端口号进行流量统计。

我觉得这个实验非常的有意义，既达到网络攻防的教学要求，让同学们对传输协议和数据包解析都能有一定的了解，又锻炼了同学的编程实践能力，是一个一举两得的好实验。在建议方面，我觉得今后的教学可以采用组队开发模式，提高实验的难度，刺激同学完成加分项的积极性，例如对流量进行可视化分析统计文件提取、密码嗅探、脚本注入等等功能。