南 开 大 学

网络空间安全学院学院

网络技术与应用课程报告

第 2 次实验报告

学号：2011428
姓名：王天行
年级：2020 级
专业：密码科学与技术

2022 年 10 月 13 日

第 1 节 实验内容说明

IP 数据报捕获与分析编程实验，要求如下：（1）了解 NPcap 的架构。（2）学习 NPcap 的设备列表获取方法、网卡设备打开方法，以及数据包捕获方法。（3）通过 NPcap 编程，实现本机的 IP 数据报捕获，显示捕获数据帧的源 MAC 地址和目的 MAC 地址，以及类型/长度字段的值。（4）捕获的数据报不要求硬盘存储，但应以简单明了的方式在屏幕上显示。必显字段包括源 MAC 地址、目的 MAC 地址和类型/长度字段的值。（5）编写的程序应结构清晰，具有较好的可读性。

第 2 节 实验准备

Npcap 了解

WinPcap 是一个基于 Win32 平台的，用于捕获网络数据包并进行分析的开源库；在 Linux 上也有对应的 LibPcap；目前 WinPcap 已经处于无人维护的状态，对于 Windows 10 有更新的且目前有人维护的开源项目 NpCap。

架构

WinPcap/NpCap 则为 Win32 应用程序提供了这样的接口：

- 捕获原始数据包；无论它是发往某台机器的，还是在其他设备（共享媒介）上进行交换的
- 在数据包发送给某应用程序前，根据指定的规则过滤数据包
- 将原始数据包通过网络发送出去
- 收集并统计网络流量信息

SDK 提供的这些功能需要借助运行在 Win32 内核中的网络设备驱动程序来实现；在安装完成驱动之后，SDK 将这些功能作为一个接口表现出来以供使用。

设备列表获取（pcap_findalldevs_ex）

NpCap 提供了 pcap_findalldevs_ex 和 pcap_findalldevs 函数来获取计算机上的网络接口设备的列表；此函数会为传入的 pcap_if_t 赋值——该类型是一个表示了设备列表的链表头；每一个这样的节点都包含了 name 和 description 域来描述设备。

除此之外，pcap_if_t 结构体还包含了一个 pcap_addr 结构体；后者包含了一个地址列表、一个掩码列表、一个广播地址列表和一个目的地址的列表；此外，pcap_findalldevs_ex 还能返回远程适配器信息和一个位于所给的本地文件夹的 pcap 文件列表。

网卡设备打开（pcap_open）

用来打开一个适配器，实际调用的是 pcap_open_live；它接受五个参数：

- name：适配器的名称（GUID）
- snaplen：制定要捕获数据包中的哪些部分。在一些操作系统中 (比如 xBSD 和

Win32)，驱动可以被配置成只捕获数据包的初始化部分：这样可以减少应用程序间复制数据的量，从而提高捕获效率；本次实验中，将值定为 65535，比能遇到的最大的 MTU 还要大，因此总能收到完整的数据包。

- flags：主要的意义是其中包含的混杂模式开关；一般情况下，适配器只接收发给它自己的数据包， 而那些在其他机器之间通讯的数据包，将会被丢弃。但混杂模式将会捕获所有的数据包——因为我们需要捕获其他适配器的数据包，所以需要打开这个开关。
- to_ms：指定读取数据的超时时间，以毫秒计；在适配器上使用其他 API 进行读取操作的时候，这些函数会在这里设定的时间内响应——即使没有数据包或者捕获失败了；在统计模式下，to_ms 还可以用来定义统计的时间间隔：设置为 0 说明没有超时——如果没有数据包到达，则永远不返回；对应的还有 -1：读操作立刻返回。
- errbuf：用于存储错误信息字符串的缓冲区

该函数返回一个 pcap_t 类型的 handle。

数据包捕获方法（pcap_loop）

API 函数 pcap_loop 和 pcap_dispatch 都用来在打开的适配器中捕获数据包；但是前者会已知捕获直到捕获到的数据包数量达到要求数量，而后者在到达了前面 API 设定的超时时间之后就会返回（尽管这得不到保证）；前者会在一小段时间内阻塞网络的应用，故一般项目都会使用后者作为读取数据包的函数；虽然在本次实验中，使用前者就够了。

这两个函数都有一个回调函数；这个回调函数会在这两个函数捕获到数据包的时候被调用，用来处理捕获到的数据包；这个回调函数需要遵顼特定的格式。但是需要注意的是我们无法发现 CRC 冗余校验码——因为帧到达适配器之后，会经过校验确认的过程；这个过程成功，则适配器会删除 CRC；否则，大多数适配器会删除整个包，因此无法被 NpCap 确认到。

```
int pcap_loop(pcap_t * p,int cnt, pcap_handler callback, uchar * user);
```

参数说明：

- p 是由 pcap_open_live()返回的所打的网卡的指针；
- cnt 用于设置所捕获数据包的个数，如果为-1 则无限循环捕获；
- pcap_handler 是 void packet_handler()使用的一个参数，即回调函数的名称；
- user 值一般为 NULL

Wireshark 使用

安装过程在写实验报告前已经完成了，报告中就忽略了具体内容。

第 3 节 实验过程

获得本机的设备列表

```
//本机接口和IP地址的获取
pcap_if_t* alldevs;                    //指向设备链表首部的指针
pcap_if_t* d;
pcap_addr_t* a;
int i = 0;
int inum = 0;
pcap_t* adhandle;
char errbuf[PCAP_ERRBUF_SIZE];  //错误信息缓冲区
//获得本机的设备列表
if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING,     //获取本机的接口设备
    NULL,                   //无需认证
    &alldevs,               //指向设备列表首部
    errbuf                  //出错信息保存缓存区
) == -1)
{
    //错误处理
    cout << "获取本机设备错误:" << errbuf << endl;
    pcap_freealldevs(alldevs);
    return 0;
}
```

打印列表

```
for (d = alldevs; d != NULL; d = d->next) //显示接口列表
{
    cout << ++i << d->name << ": "; //利用d->name获取该网络接口设备的名字
    if (d->description) { //利用d->description获取该网络接口设备的描述信息
        cout << d->description << endl;
    }
    else {
        cout << "无相关描述信息" << endl;
        return -1;
    }
}
if (i == 0)
{
    cout << "wrong!" << endl;
    return -1;
}
```

打开设备

```
cout << "请输入接口号：" << i;
cin >> inum;

//检查用户是否指定了有效的设备
if (inum < 1 || inum > i)
{
    cout << "适配器数量超出范围" << endl;

    pcap_freealldevs(alldevs);
    return -1;
}

//跳转到选定的设备
for (d = alldevs, i = 0; i < inum - 1; d = d->next, i++);


//打开网卡
if ((adhandle = pcap_open(d->name,          // 设备名
    65536,              // 要捕捉的数据包的部分
                        // 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)
{
    cout << "无法打开设备：检查是否是支持的NPcap" << endl;
    pcap_freealldevs(alldevs);
    return -1;
}
```

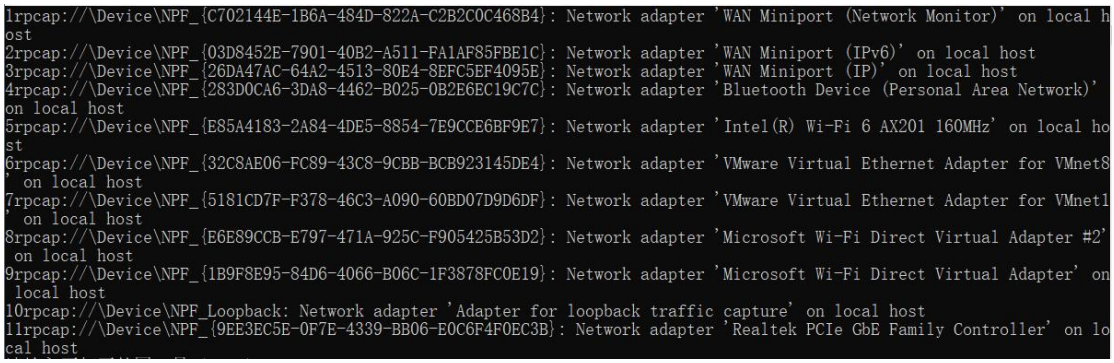
获取数据包

```
cout << "监听：" << d->description << endl;
pcap_freealldevs(alldevs);
int cnt = -1;
cout << "将要捕获数据包的个数：";
cin >> cnt;
pcap_loop(adhandle, cnt, ethernet_protocol_packet_handle, NULL);
pcap_close(adhandle);
```

实验结果

```
==============Ethernet Protocol================
Destination Mac Address:  f0:57:a6:fb:a2:d4
Source Mac Address:  de:aa:27:3b:38:8d
Ethernet type:  IP 0x0800
Version: 4
Header Length: 20 Bytes
Tos: 72
Total Length: 1400
Identification: 0x5eec
Flags: 16384
Time to live: 107
Protocol Type:  TCP(6)
Header checkSum: 0xde8c
Source: 40.90.184.73
Destination: 192.168.43.115
==============Ethernet Protocol================
Destination Mac Address:  f0:57:a6:fb:a2:d4
Source Mac Address:  de:aa:27:3b:38:8d
Ethernet type:  IP 0x0800
Version: 4
Header Length: 20 Bytes
Tos: 72
Total Length: 1400
Identification: 0x5eed
Flags: 16384
Time to live: 107
Protocol Type:  TCP(6)
Header checkSum: 0xde8b
Source: 40.90.184.73
Destination: 192.168.43.115
==============Ethernet Protocol================
Destination Mac Address:  f0:57:a6:fb:a2:d4
Source Mac Address:  de:aa:27:3b:38:8d
Ethernet type:  IP 0x0800
Version: 4
Header Length: 20 Bytes
Tos: 72
Total Length: 1400
Identification: 0x5eee
Flags: 16384
Time to live: 107
Protocol Type:  TCP(6)
Header checkSum: 0xde8a
Source: 40.90.184.73
Destination: 192.168.43.115
```

结果分析



1-3（WAN Miniport ）：wan 微型端口协议，安装了 VPN 或者 PPPOE 就会出现，使用 V-P-N 等拨号连接产生的虚拟网卡

4（Bluetooth 个人区域网 (PAN)）：Bluetooth 个人区域网 (PAN) 是一种可利用便携式计算机、移动电话和手持设备之间的无线链接创建以太网网络的技术。可以连接到以下类型的启用 Bluetooth 的设备（这些设备都使用 PAN）：个人区域网用户 (PANU) 设备、组式临时网络 (GN) 设备或网络访问点 (NAP) 设备。

下面是有关这些设备中每种设备的功能的详细信息：

- PANU 设备。 连接到启用 Bluetooth 的 PANU 设备可创建一个包含您的计算机和设备的临时网络。
- GN 设备。连接到启用 Bluetooth 的 GN 设备可创建一个包含您的计算机、GN 设备和其他任何与同一 GN 设备连接的 PANU 设备的临时网络。
- NAP 设备。 连接到启用 Bluetooth 的 NAP 设备可让您将计算机连接到更大的网络，如家庭网络、企业网络或 Internet。：

8（Microsoft virtual wifi adapter）：用来无线显示时使用的，当你的电脑和其他设备在同一个网络环境（例连接同一 WiFi）时其他设备可以将其显示内容和声音投放到你的电脑上，而不使用你原来的 WiFi 连接（用不同的网络适配器接受数据）

经实践，1-11 中，5 能快速捕获到大量的数据包；10 能捕获到数据包，但速度较慢；1-4 和 8-9 基本无法捕获到数据包；6-7 能够捕获到数据包，但速度较慢

第 4 节 特殊现象分析

问题： error C4996: 'strcpy': This function or variable may be unsafe. Consider using strcpy_s instead.

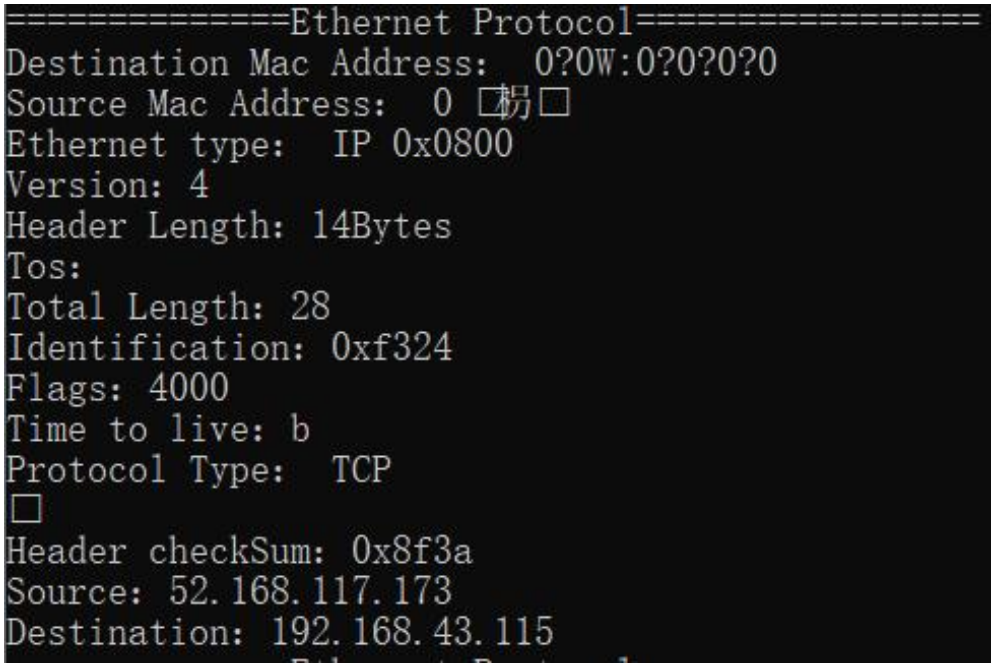
解决：找到【项目属性】，点击【C++】里的【预处理器】，对【预处理器】进行编辑，在里面加入一段代码：_CRT_SECURE_NO_WARNINGS。

问题：错误 C4996 ‘inet_ntoa‘: Use inet_ntop() or InetNtop() instead or define _WINS

解决：改使用 inet_ntop()函数　　inet_ntop(AF_INET,&source.sin_addr,str,16)

问题：error LNK2019: 无法解析的外部符号
解决：添加#pragma comment(lib,"ws2_32.lib")

问题：ip 地址打印错误



代码如下图：

```
//以太网目标地址
mac_string = ethernet_protocol->DesMAC;

cout << "Destination Mac Address:  ";
cout << hex << setw(2) << setfill('0') << *mac_string << ":";
cout << hex << setw(2) << setfill('0') << *(mac_string + 1) << ":";
cout << hex << setw(2) << setfill('0') << *(mac_string + 2) << ":";
cout << hex << setw(2) << setfill('0') << *(mac_string + 3) << ":";
cout << hex << setw(2) << setfill('0') << *(mac_string + 4) << ":";
cout << hex << setw(2) << setfill('0') << *(mac_string + 5) << endl;

//以太网源地址
mac_string = ethernet_protocol->SrcMAC;

cout << "Source Mac Address:  ";
cout << hex << setw(2) << setfill('0') << *mac_string
    << *(mac_string + 1)
    << *(mac_string + 2)
    << *(mac_string + 3)
    << *(mac_string + 4)
    << *(mac_string + 5) << endl;
```

解决：网上所给的示例使用的都是 printf（）函数，没有对应的 cout 输出示例。

```
==============Ethernet Protocol================
Destination Mac Address:  f0:57:a6:fb:a2:d4:
0?0W:0?0?0?0
```

这是 printf（）函数和 cout 输出的结果对比、

在 ASCII 中，一共定义了 128 个字符，其中 33 个无法显示，为 0～31 和 127，32 到 126 为可显示字符，当使用 cout 输出一个 char 型字符时，如果是可显示范围内，则输出相应可显示字符，其余的都输出乱码。

如果我们要使字符按十六进制输出，可以使用 hex，但是发现 cout << hex << data[0];没有输出十六进制，因为 hex 只对整数起作用，将 data[0]转换为 int，cout << hex << int(data[0]); 发现输出的结果前面带了很多 f。因为 data[0]是有符号数，最高位为 1 时，转换为 int 时，其余位都为 1，cout << hex << (unsigned int) (unsigned char)data[0];结果正确。

此时输出结果格式正确。

```
==============Ethernet Protocol================
Destination Mac Address:  f0:57:a6:fb:a2:d4
Source Mac Address:  02:1f:96:a1:0c:ac
Ethernet type:  IP 0x0800
Version: 4
Header Length: 14Bytes
Tos:
Total Length: 28
Identification: 0x84c5
Flags: 4000
Time to live: b
Protocol Type:  TCP
□
Header checkSum: 0xfd99
Source: 52.168.117.173
Destination: 192.168.43.115
```

问题：长度等数据结果不正确

```
==============Ethernet Protocol================
Destination Mac Address:  0?0W:0?0?0?0
Source Mac Address:  0 □粉□
Ethernet type:  IP 0x0800
Version: 4
Header Length: 14Bytes
Tos:
Total Length: 28
Identification: 0xf324
Flags: 4000
Time to live: b
Protocol Type:  TCP
□
Header checkSum: 0x8f3a
Source: 52.168.117.173
Destination: 192.168.43.115
```

解决：在 c++中，首先按照几进制输出在前面写出来一个 cout << hex (十六进制)，然后因为 hex 只对整形数据起作用，所以还得强制类型转换。同时部分需要按照十进制输出，而 c++

中输出有一个 hex，后面都默认为十六进制输出，所以在涉及到进制输出变化的时候需要进行改变。
修改后代码如下：

```
//开始输出
cout << dec << "Version: " << (int)(IPHeader->Ver_HLen >> 4) << endl;
cout << "Header Length: ";
cout << (int)((IPHeader->Ver_HLen & 0x0f) * 4) << " Bytes" << endl;
cout << "Tos: " << (int)IPHeader->TOS << endl;
cout << "Total Length: " << (int)ntohs(IPHeader->TotalLen) << endl;
cout << "Identification: 0x" << hex << setw(4) << setfill('0') << ntohs(IPHeader->ID) << endl;
cout << "Flags: " << dec << (int)(ntohs(IPHeader->Flag_Segment)) << endl;
cout << "Time to live: " << (int)IPHeader->TTL << endl;
cout << "Protocol Type: ";
switch (IPHeader->Protocol)
{
case 1:
    cout << "ICMP";
    break;
case 6:
    cout << "TCP";
    break;
case 17:
    cout << "UDP";
    break;
default:
    break;
}
cout << "(" << (int)IPHeader->Protocol << ")" << endl;
cout << "Header checkSum: 0x" << hex << setw(4) << setfill('0') << ntohs(IPHeader->Checksum) << endl;
cout << "Source: " << sourceIP << endl;
cout << "Destination: " << destIP << endl;
```

此时输出结果正确。

问题：某一条输出输出的协议类型和 wireshark 中的不一样

33 3.890621	40.90.184.73	192.168.43.115	TCP
34 7.989292	192.168.43.115	2.16.149.142	TCP
35 8.191361	119.3.178.178	192.168.43.115	TCP
36 8.191421	192.168.43.115	119.3.178.178	TCP
37 9.401555	192.168.43.115	40.84.185.67	TLSv1.2
38 9.727959	40.84.185.67	192.168.43.115	TCP
39 11.053847	192.168.43.115	182.61.200.7	TCP
40 11.147547	182.61.200.7	192.168.43.115	TCP

```
==============Ethernet Protocol=================
Destination Mac Address:  02:1f:96:a1:0c:ac
Source Mac Address:  f0:57:a6:fb:a2:d4
Ethernet type:  IP 0x0800
Version: 4
Header Length: 20 Bytes
Tos: 0
Total Length: 166
Identification: 0xdf91
Flags: 16384
Time to live: 64
Protocol Type:  TCP(6)
Header checkSum: 0x0000
Source: 192.168.43.115
Destination: 40.84.185.67
```

解决：TLSv1.2 是基于 TCP 协议，而代码中并没有对 TCP 数据包等进行再分析。