南開大學

网络技术与应用课程实验报告

实验二：利用 NPcap 编程捕获数据包

学　　院　密码与网络空间安全学院

专　　业　信息安全、法学双学位班

学　　号　2313815

姓　　名　段俊宇

班　　级　信息安全、法学双学位班

一、实验目的

1. 配置和安装 NPcap,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，计算捕获 IP 数据报的校验和，并与数据报的校验和字段比较。

二、实验原理

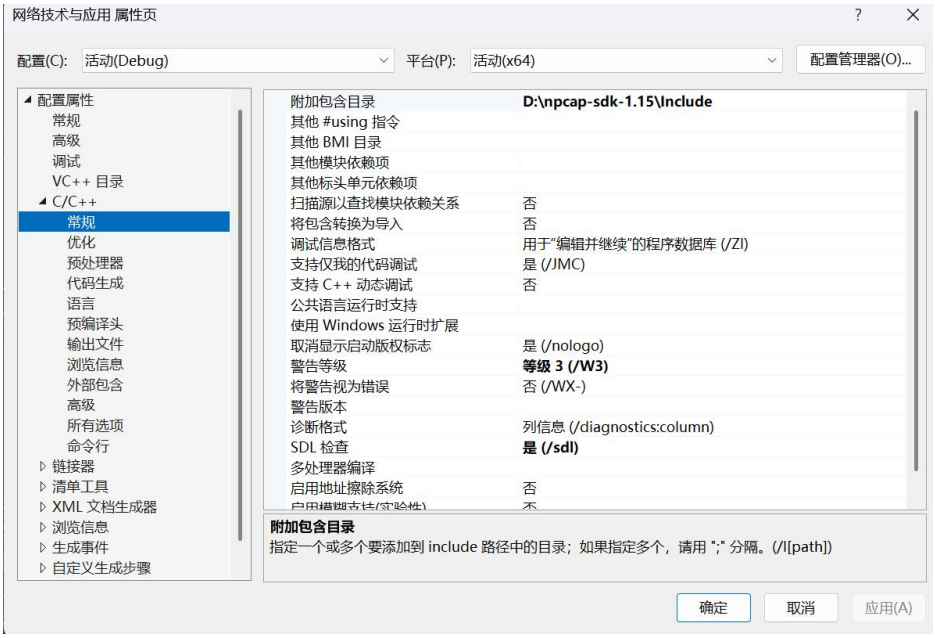
1. NPcap 基本介绍

NPcap 是基于 Windows 平台的现代数据包捕获架构,它使用自定义的 Windows 内核驱动程序和 libpcap 库来完成 PcapAPI 的构建。

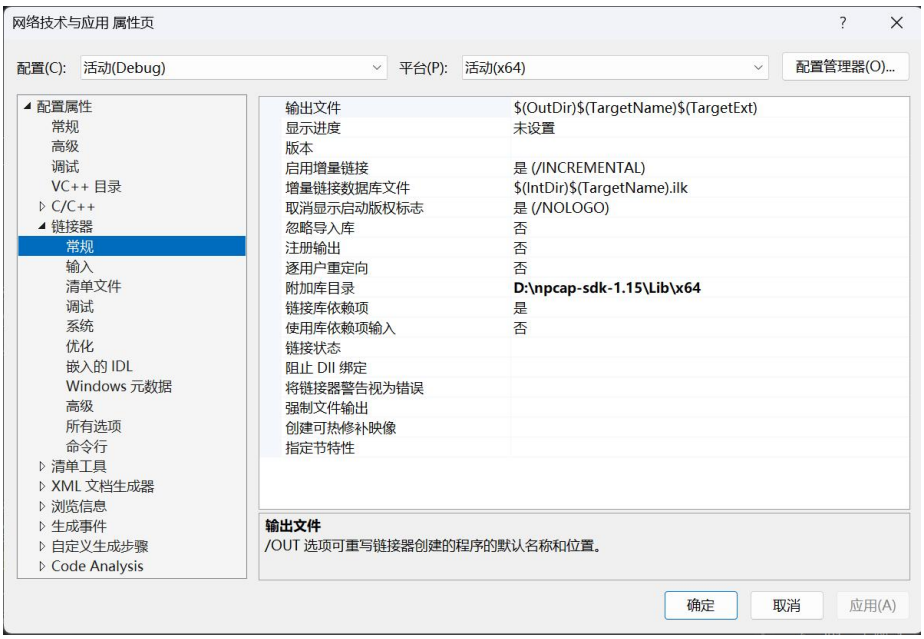
架构上，NPcap 采用分层设计：在用户态提供与 libpcap 兼容的 API 接口，确保现有网络工具的无缝移植；在内核态通过 NDIS 协议驱动直接与网络栈交互，实现高性能的数据包过滤和转发。这样在传输原始数据包的同时，也可以防止普通用户滥用网络嗅探功能。

NPcap 内置了 BPF 过滤引擎，允许用户设置灵活的过滤规则，提升了捕获效率。同时通过绕过操作系统协议栈的"原始数据包注入"技术，为网络扫描、流量生成等应用提供底层支持。相较于前代 WinPcap，NPcap 还新增了环回数据包捕获、802.11 帧处理等增强功能，并通过服务形式安装避免了 DLL 依赖问题，形成了更稳定可靠的数据包捕获系统。

在使用 NPcap 时,需要下载安装程序和 SDK,并在 Visual Studio2022 中配置相关链接库。如下图所示，项目->属性->C/C++->常规的附加包含目录中填写 SDK 中 include 文件夹的路径



之后还需要在链接器->常规的附加库目录中填写 Lib 文件夹的路径，如果使用的是 x64 架构，需要填写 Lib 文件夹中 x64 文件夹的路径，如下图所示



最后还需要在链接器->输入的附加依赖项中添加 wpcap.lib 和 Packet.lib 以及 Ws2_32.lib，这样就配置好了基本环境。

2. 设备列表获取方法

在 NPcap 中提供了 pcap_if_t 类型用于定义网卡设备的链表节点，pcap_findalldevs_ex()函数能够获取本机的接口设备以及远程连接的设备，函数参数如下所示

```
PCAP_API int pcap_findalldevs_ex(const char *source,
        struct pcap_rmtauth *auth, pcap_if_t **alldevs, char *errbuf);
```

在使用时，第一个参数为 PCAP_SRC_IF_STRING，这是一个宏定义，内容是 rpcap://，添加这个参数用于获取本机的接口设备；第二个参数为 NULL，表示无需认证；第三个参数是自定义的设备链表；第四个参数是错误缓冲区，用于写入错误信息。

3. 网卡设备打开方法

在 NPcap 中提供了 pcap_t 类型用于定义活动的数据包捕获会话的句柄或上下文，pcap_open()函数能够打开指定的网卡设备并获取句柄，函数参数如下所示

```
PCAP_API pcap_t   *pcap_open(const char *source, int snaplen, int flags,
        int read_timeout, struct pcap_rmtauth *auth, char *errbuf);
```

在使用时，第一个参数是你要打开的设备的名字；第二个参数是捕获长度，也就是能够捕获多少字节的数据，一般设置为 65536 足以捕获大多数完整数据帧的数据；第三个参数是标志位，可以设置各种模式，一般设置为 PCAP_OPENFLAG_PROMISCUOUS，代表混杂模式，也就是捕获所有数据包；第四个参数为读取超时时间，我这里设置的是 100，单位为毫秒；第五个参数是认证信息，这里填写 NULL 即可；第六个参数是错误缓冲区，用于写入错误信息。

4. 数据包捕获方法

在 NPcap 中提供了 pcap_next_ex()、pcap_loop()以及 pcap_dispatch()函数，这些函数都可以用于捕获数据包，但是存在细微的不同。下面是函数参数

```
PCAP_API int pcap_loop(pcap_t *, int, pcap_handler, u_char *);
PCAP_API int pcap_next_ex(pcap_t *, struct pcap_pkthdr **, const u_char **);
PCAP_API int pcap_dispatch(pcap_t *, int, pcap_handler, u_char *);
```

pcap_dispatch()在读取超时时间到达后就返回；pcap_loop()在捕获到 cnt 个数据包后返回，cnt 是第二个参数，代表数据包的数量；pcap_next_ex()是直接捕获数据包，上面两个都是利用回调函数捕获，在读取超时时间到达后返回。这三个函数的第一个参数都是打开网卡设备后获取的句柄；pcap_loop 和 pcap_dispatch 的第三个参数都是回调函数，这个需要自定义；pcap_next_ex 的第二个参数是数据包头二级指针，pcap_pkthdr 是数据包头部结构体；最后一个参数是可选的用户自定义指针，pcap_next_ex 的最后一个参数是二级指针，函数会让这个指针变量指向数据包的原始字节流并返回。本次实验中我使用的是 pcap_next_ex()函数。

三、实验过程

在编程的过程中我定义了以太网帧结构体、IP 头部结构体和一些函数，用来获取设备列表、打开网卡设备以及捕获数据包并进行处理等。接下来我将使用注释的形式进行解释。

1. 以太网帧结构体

```
// 以太网帧头部
struct EthernetHeader
{
    byte dstMAC[6];       // 目标 MAC 地址，大小为 6 字节
    byte srcMAC[6];       // 源 MAC 地址
    WORD ethertype;       // 协议类型
};
```

2. IP 头部结构体

```
// IP 报文头部
struct IPHeader
{
    byte ver_ihl;         // 版本类型+头部长度
    byte TOS;             // 服务类型
    WORD Total_len;       // 总长度
    WORD ID;              // 标识
    WORD Flag_fragment;   // 标志+片偏移
    byte TTL;             // 生存时间
    byte protocol;        // 协议类型
```

```
    WORD Check_sum;        // 头部校验和
    in_addr srcIP;         // 源 IP 地址
    in_addr dstIP;         // 目标 IP 地址
};
```

3. MAC 地址转字符串

```
// MAC 地址转字符串
string MACtostring(const byte* MAC)
{
    stringstream ss;                    // 创建一个字符串流对象
    ss << hex << setfill('0');          // 设置流格式为输出 16 进制
    for (int i = 0; i < 6; i++)         // 循环处理每个字节
    {
        ss << setw(2) << (int)MAC[i];
        if (i < 5) ss << ":";           // 只需要 5 个分隔符
    }
    return ss.str();
}
```

这里需要添加头文件如下

```
#include <iomanip>
#include <sstream>
```

4. 计算头部校验和

计算头部校验和时，需要将头部除校验和的部分之外划分为若干个长度为双字节的部分，然后将其累加，如果发生进位，则需要将进位的部分和其他部分累加，最后保证结果长度为 16bit，之后取反就可以得到校验和，实现方法如下

```
// 计算头部校验和，将 IP 头部除校验和之外全部划分为多个 16 位的部分并累加，之后取反得到校验和
WORD calculate_check_num(const IPHeader* header)
{
    DWORD sum = 0;                                  // 设置校验和为 0
    WORD* ptr = (WORD*)header;                      // 将 header 指针转成 16 位的
    sum += ((int)header->ver_ihl << 8) | (int)header->TOS;    // 前 8 位和后 8 位通过或运算拼接成 16 位，版本+头部长度+服务类型
    sum += ntohs(header->Total_len);                // 16 位的总长度，使用 ntohs 将网络序转换成主机序
    sum += ntohs(header->ID);                       // 16 位的标识
    sum += ntohs(header->Flag_fragment);            // 16 位的标志+片偏移
```

```
    sum += ((int)header->TTL << 8) | (int)header->protocol;  // 将 8 位的
生存时间和 8 位的协议类型拼接
    sum += ntohs(header->srcIP.S_un.S_un_w.s_w1);    // 源 IP 前两个字节
    sum += ntohs(header->srcIP.S_un.S_un_w.s_w2);    // 源 IP 后两个字节
    sum += ntohs(header->dstIP.S_un.S_un_w.s_w1);    // 目标 IP 前两个字节
    sum += ntohs(header->dstIP.S_un.S_un_w.s_w2);    // 目标 IP 后两个字节
    while ((sum >> 16)!=0)                           // 若发生进位，则将进
位部分与低 16 位累加，直至没有进位
    {
        sum = (sum & 0xffff) + (sum >> 16);
    }
    sum = ~sum;                                      // 取反得到校验码
    return sum;
}
```

在这里需要注意，sum 应当定义为 DWORD 类型，防止进位发生溢出。最初我定义的是 WORD 类型，但是发生进位后数据会不全，进位部分遗失，最后导致计算出的校验和与原始校验和不一样，后来经过排查修改了类型，才输出了正确的校验和。

5. 打印设备信息

```
// 打印设备信息
int PrintDeviceInfo(pcap_if_t* device_list)
{
    pcap_if_t* d;              // 遍历指针
    int count = 0;             // 可用设备数量
    for (d = device_list;d != nullptr;d = d->next)           // 开始遍历
    {
        cout << (count+1) << ".Device:" << d->name << endl;  // 设备序号
        if (d->description)
            cout << "Description:" << d->description << endl;// 设备描述
        else
            cout << "No description available" << endl;
        count++;
    }
    cout << "Total devices:" << count << endl;
    return count;
}
```

6. 让用户选择需要打开的网卡设备

```
// 让用户选择需要打开的网卡设备
pcap_if_t* select_device(pcap_if_t* alldevices)
{
    int count;        // 可用设备数量
    pcap_if_t* d;// 遍历指针
    count = PrintDeviceInfo(alldevices);    // 调用函数打印设备信息并返回设备总数
LOOP:
    // 用户选择要打开的网卡设备
    int choice;
    cout << "Enter the number of the Network interface Device you want to open:";
    cin >> choice;
    if (choice<1 || choice>count)           // 输入的数字无效
    {
        cout << "Invalid number!Out of range!" << endl;
        cout << "continue or quit ([c/q]):";// 选择继续或者退出
        char op;
        cin >> op;
        if (op == 'c' || op == 'C')
            goto LOOP;
        else
        {
            pcap_freealldevs(alldevices);  // 释放设备列表
            exit(0);
        }
    }
    // 寻找指定的设备
    int i;
    for (d = alldevices, i = 0;i < choice - 1 && d != nullptr;d = d->next, i++);

    return d;
}
```

在寻找指定设备时，需要将上限设置为 choice-1，否则会寻找到指定设备的下一台设备，导致输出错误。

7. 处理数据包

```
// 处理数据包
void PackageHandler(const pcap_pkthdr* header, const u_char* packet)
{
    // 数据包过小，无法解析
    if (header->caplen < sizeof(EthernetHeader) + sizeof(IPHeader))
    {
        cout << "The length of the data packet is too short, parsing failed!"
<< endl;
        return;
    }
    // 解析以太网帧头部
    const EthernetHeader* ethheader = (const EthernetHeader*)packet;

    cout << "source MAC address:" << MACtostring(ethheader->srcMAC) << endl;
    cout << "destination MAC address:" << MACtostring(ethheader->dstMAC) <<
endl;

    // 解析 IP 数据包头部，以太网帧占 14 字节
    const IPHeader* IPheader = (const IPHeader*)(packet + 14);
    cout << "source IP address:" << inet_ntoa(IPheader->srcIP) << endl;
    cout << "destination IP address:" << inet_ntoa(IPheader->dstIP) << endl;
    if (ntohs(IPheader->Check_sum) == 0)
        cout << "original checksum:0 (checksum offloading)" << endl;
    else
    cout << "original checksum:" << ntohs(IPheader->Check_sum) << endl;
    cout << "calculated checksum:" << calculate_check_num(IPheader) << endl;
}
```

Inet_ntoa()函数是 in_addr 结构体自带的函数，能够将网络序转换成主机序，避免显示或者计算的时候出现错误。但是这一函数已经弃用，因此需要在#include<iostream>上面加上忽视警告的宏定义#define _WINSOCK_DEPRECATED_NO_WARNINGS，或者更换其他函数。

8. 捕获数据包

```
// 捕获数据包
void CaptureDataPackage(pcap_t* handle)
{
    pcap_pkthdr* header;    // 数据包头
    const u_char* packet;   // 数据包本身
    int result;
    cout << "Start capturing data packets ..." << endl;
```

```
    int packetnum = 0;
    while(true)                 // 无限循环捕获
    {
        result = pcap_next_ex(handle, &header, &packet);
        // 超时，继续等待
        if (result == 0)
        {
            continue;
        }
        packetnum++;
        cout << "the number of data packet: " << packetnum << endl;
        cout << "the length is:" << header->len << "bytes" << endl;
        cout << "Start analysis ..." << endl;
        PackageHandler(header, packet);     // 对数据包进行分析
        if (packetnum == 20)                // 当捕获 20 个数据包时停止
            break;
    }
    pcap_close(handle);                     // 关闭句柄并释放相关资源
}
```

9. main 函数

```
int main()
{
    pcap_if_t* alldevices;          // 设备列表的存储链表
    char errbuf[PCAP_ERRBUF_SIZE]; // 存储错误信息，256 字节

    cout << "================================Devices
List================================" << endl;

    // 从本地设备中获取可用设备列表
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevices, errbuf)
== -1)
    {
        // 查找失败，则输出错误信息
        cerr << "ERROR:" << errbuf << endl;
        return 1;
    }
    // 查找成功的话，设备列表已经存储在 alldevices 里面
    pcap_if_t* device = select_device(alldevices);
    if (!device)
        return 1;
```

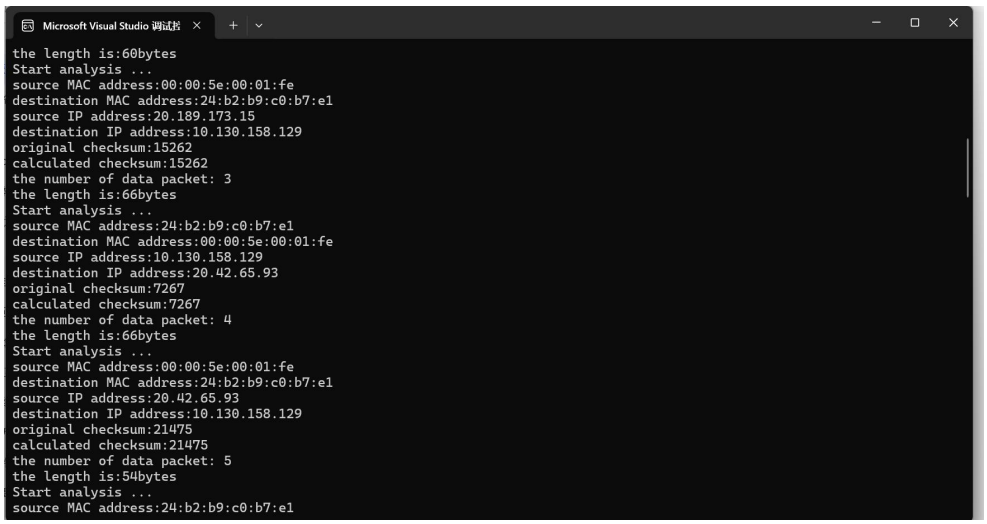
```
    pcap_t* handle;
    // 打开设备
    handle = pcap_open(device->name, 65536, PCAP_OPENFLAG_PROMISCUOUS,
100, NULL, errbuf);
    if (!handle)
    {
        cerr << "ERROR:" << errbuf << endl;
        return 1;
    }
    cout<<"Successfully open network device:"<< device->name << endl;
    pcap_freealldevs(alldevices);
    CaptureDataPackage(handle);

    return 0;
}
```

在 main 函数中，首先从本地设备中获取设备列表，然后让用户选择想要打开的设备，之后打开该设备并捕获数据包、分析数据包，最后退出程序。在我的本地设备列表中有如下设备

设备名称	设备用途
WAN Miniport (Network Monitor)	虚拟驱动程序，用于处理出站连接，捕获原始数据包。
WAN Miniport (IPv6)	虚拟驱动程序，用于处理出站连接，提供 IPv6 协议支持
WAN Miniport (IP)	虚拟驱动程序，用于处理出站连接，提供 IPv4 协议支持
Hyper-V Virtual Ethernet Adapter	微软 Hyper-V 虚拟交换机，让虚拟机和宿主机之间可以通信，也可以让虚拟机通过主机的网络连接上网。
Bluetooth Device (Personal Area Network)	蓝牙网络接口，当电脑通过蓝牙与其他设备配对并创建了“个人区域网络”时，这个适配器会出现并通过蓝牙进行 TCP/IP 网络通信

Realtek 8922AE WiFi 7 PCI-E NIC	无线网卡，用于连接 Wi-Fi 无线网络
VMware Virtual Ethernet Adapter for VMnet8	VMware 虚拟网络，创建一个网络地址转换模式的网络，虚拟机可以连接到外部互联网（通过主机的 IP），但是外部互联网不能直接发起和虚拟机的连接
VMware Virtual Ethernet Adapter for VMnet1	VMware 虚拟网络，创建一个仅主机模式的网络，虚拟机无法访问外部互联网
Realtek 8922AE WiFi 7 PCI-E NIC #5	无线网卡，进行无线网络分析、嗅探、安全审计
Realtek 8922AE WiFi 7 PCI-E NIC #3	无线网卡，用于无线渗透测试、性能测试
Adapter for loopback traffic capture	NPcap 创建的环回适配器，捕获发往本机（localhost, 127.0.0.1）的网络流量
Realtek PCIe GbE Family Controller	有线以太网卡，用于通过网线连接局域网或互联网

其中我选择的是 Realtek 8922AE WiFi 7 PCI-E NIC 网卡设备并捕获流量。在实验过程中我发现有一些数据包的原始校验和为 0，和我计算得到的校验和并不一致，经过搜集资料发现这是校验和卸载现象，是网卡设备直接计算校验和，省去 CPU 计算，减少系统资源的使用。修改代码后，除去校验和为 0 的情况，得到了如下的数据包以及其中的信息



非 0 的校验和都一致，说明该程序运行成功，代码没有问题！

四、实验结论及心得体会

本次实验我深刻理解了网络协议栈的底层实现机制以及 NPcap 的架构，学习了 NPcap 获取设备列表、打开网卡设备、捕获数据包的基本方法以及校验和的计算方法等。在调试校验和验证的过程中，我遇到了因数据类型溢出导致的微小计算偏差，这一问题让我感受到网络编程中对字节序处理和数值运算精确性的严苛要求。这次实验让我对以太网帧、IP 头部有了详细了解，为后续深入学习网络安全和协议分析奠定了扎实基础。