南开大学

Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

网络技术与应用实验报告

数据包捕获与分析

李潇逸 2111454

年级：2021 级

专业：信息安全、法学

指导教师：张建忠

2023 年 10 月 21 日

目录

一、 Npcap 概述

(一) 简介

大多数网络应用程序通过广泛使用的操作系统原语（如套接字）访问网络。使用这种方法很容易访问网络上的数据，因为操作系统处理低级细节（协议处理、数据包重组等），并提供类似于用于读取和写入文件的熟悉界面。然而，有时“简单的方法”无法胜任这项任务，因为某些应用程序需要直接访问网络上的数据包。也就是说，他们需要访问网络上的“原始”数据，而无需操作系统干预协议处理。

Npcap 是用于 Windows 操作系统的数据包捕获和网络分析的体系结构，由软件库和网络驱动程序组成。其目的是提供对 Windows 应用程序的这种访问。它提供以下设施:

- 捕获原始数据包，包括发往运行该计算机的原始数据包和其他主机交换的数据包（在共享媒体上）
- 在将数据包分派到应用程序之前，根据用户指定的规则过滤数据包
- 将原始数据包传输到网络
- 收集有关网络流量的统计信息

简而言之，通过 Npcap 我们可以得到原始（raw）网络数据，即未经过 TCP/IP 协议栈的数据，也就是网卡收到的数据。同时也可以通过 Npcap 设置接收过滤器，这样收到的数据就是我们感兴趣的数据，比如某个端口的数据。而且，Npcap 还提供发送原始（raw）网络数据的功能。

(二) 原理

Npcap 实现了 Win10 驱动程序，叫做 NPF（Netgroup Packet Filter），该驱动从 Win10 miniport 驱动获取网卡数据实现监控网络数据包的功能（Win10 使用 miniport 驱动控制网卡）。

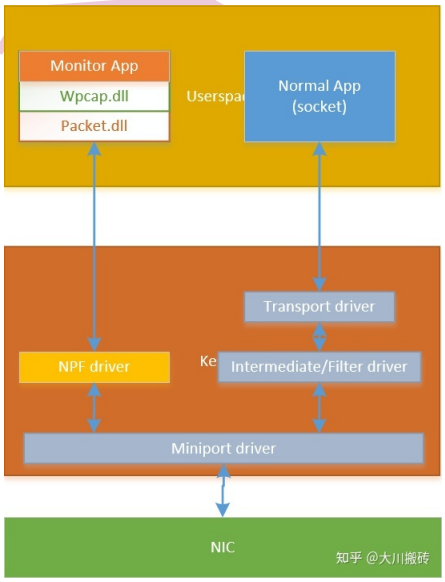

图 1: Caption

(三) 使用函数

1. 获取设备列表

获取设备列表

```
if (pcap_findalldevs_ex((char*)PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) ==
    -1)
{
    fprintf(stderr, "Error in pcap_findalldevs_ex:%s\n", errbuf);
    exit(1);
}
```

使用 pcap_findalldevs_ex() 函数进行设备列表的获取，将指向设备链表首部的指针存在 alldevs 中。

2. 网卡设备打开方式

网卡设备打开方式

```
for (d = alldevs; num < (n);num++)
{
    d = d->next;
}

pcap_t * adhandle = pcap_open(
    d->name, // name of the device
    65536, // portion of the packet to capture
    PCAP_OPENFLAG_PROMISCUOUS | // promiscuous mode
    PCAP_OPENFLAG_NOCAPTURE_LOCAL | PCAP_OPENFLAG_MAX_RESPONSIVENESS,
    1000, // read timeout
    NULL,
    errbuf); // error buffer
if (adhandle == NULL)
{
    cout << "产生错误，无法打开设备" << endl;
    pcap_freealldevs(alldevs);
    return 0;
}
else
{
    cout << "监听: " << d->description << endl;
    pcap_freealldevs(alldevs);

}
```

首先循环查找应当打开哪一个设备，找到之后使用 pcap_open() 进行打开。其中的参数含义为:

- d->name：设备名称
- 65536：要抓的包

- PCAP_OPENFLAG_PROMISCUOUS：混合模式
- 1000：延迟时间
- errbuf：错误缓冲

3. 数据包捕获方法

使用 pcap_loop(adhandle, read_count, (pcap_handler)pcap_callback, NULL) 函数。其中 pcap_callback 定义如下：

pcap_callback

```
void pcap_callback(u_char* argument, const struct pcap_pkthdr* packet_header,
    const u_char* packet_content) //packet_content表示的捕获到的数据包的内容
{
    const u_char* ethernet_protocol;            //以太网协议
    u_short ethernet_type;                  //以太网类型
    //获取以太网数据内容
    ethernet_protocol = packet_content;
    ethernet_type = (ethernet_protocol[12] << 8) | ethernet_protocol[13];
    printf("Mac源地址: \n");
    printf("%02x:%02x:%02x:%02x:%02x:%02x:\n",
        (ethernet_protocol[6]),
        (ethernet_protocol[7]),
        (ethernet_protocol[8]),
        (ethernet_protocol[9]),
        (ethernet_protocol[10]),
        (ethernet_protocol[11])
    );
    printf("Mac目的地址: \n");
    printf("%02x:%02x:%02x:%02x:%02x:%02x:\n",
        (ethernet_protocol[0]),
        (ethernet_protocol[1]),
        (ethernet_protocol[2]),
        (ethernet_protocol[3]),
        (ethernet_protocol[4]),
        (ethernet_protocol[5])
    );
    printf("以太网类型为␣:\t");
    printf("%04x\n", ethernet_type);
    switch (ethernet_type)
    {
    case 0x0800:
        printf("网络层是: IPv4协议\n");
        break;
    case 0x0806:
        printf("网络层是: ARP协议\n");
        break;
    case 0x8035:
```

```
            printf("网络层是: RARP协议\n");
            break;
        default:
            printf("网络层协议未知\n");
            break;
        }
        if (ethernet_type == 0x0800)
        {
            IP_Packet_Handle(packet_header, packet_content);
        }
}
```

此外我们还定义了一个结构体如下以方便获取 IP 地址:

IPHeader$_t$

```
typedef struct IPHeader_t {                    //IP首部
        BYTE Ver_HLen;//IP协议版本和IP首部长度: 高4位为版本, 低4位为首部的长
            度
        BYTE TOS;//服务类型
        WORD TotalLen;//总长度
        WORD ID;//标识
        WORD Flag_Segment;//标志 片偏移
        BYTE TTL;//生存周期
        BYTE Protocol;//协议
        WORD Checksum;//头部校验和
        u_int SrcIP;//源IP
        u_int DstIP;//目的IP
}IPHeader_t;
```


二、 数据包结构

为了捕获数据包, 我们首先需要了解数据包的结构。具体如图所示。

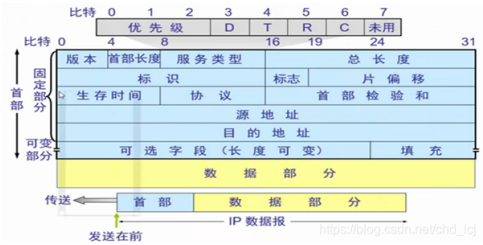

图 2: Caption

三、 结果展示

运行代码, 首先将直接显示电脑上的所有设备运行列表

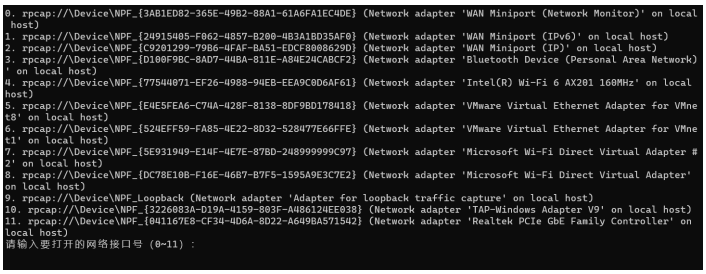

图 3: Caption

之后根据所显示的设备序号选择所要监听的设备并进行监听

图 4: Caption

在这里我们选择监听 4，这是由于 WiFi 设备必定有数据包的传输，其余设备则不一定
在此之后我们输入想要捕获的包的数量，最后开始捕获数据包

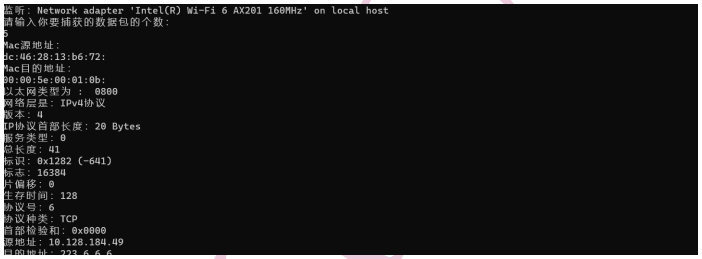

图 5: Caption

在这里截取了所捕获数据包的第一个进行展示。可以发现正确显示了捕获数据帧的源 MAC 地址和目的 MAC 地址，以及类型/长度字段的值，程序运行成功。

四、 完整代码

code

```
#include<Winsock2.h>
#include<iostream>
#include<pcap.h>
#include<stdio.h>
#include<time.h>
#include<string>
#pragma comment(lib,"ws2_32.lib")
#pragma comment(lib, "packet.lib")
#pragma comment(lib,"wpcap.lib")
#pragma comment(lib,"wsock32.lib")
#pragma warning(disable : 4996)

```

```
13 using namespace std;
14
15 typedef struct IPHeader_t {                    //IP首部
16         BYTE Ver_HLen;//IP协议版本和IP首部长度: 高4位为版本, 低4位为首部的长
               度
17         BYTE TOS;//服务类型
18         WORD TotalLen;//总长度
19         WORD ID;//标识
20         WORD Flag_Segment;//标志 片偏移
21         BYTE TTL;//生存周期
22         BYTE Protocol;//协议
23         WORD Checksum;//头部校验和
24         u_int SrcIP;//源IP
25         u_int DstIP;//目的IP
26 }IPHeader_t;
27
28 void pcap_callback(u_char* argument, const struct pcap_pkthdr* packet_header,
       const u_char* packet_content);
29 void IP_Packet_Handle(const struct pcap_pkthdr* packet_header, const u_char*
      packet_content);
30
31 int main()
32 {
33     pcap_if_t* alldevs;//指向设备链表首部的指针
34     pcap_if_t* d;
35     int num = 0;//接口数量
36     char errbuf[PCAP_ERRBUF_SIZE];
37
38     /*获取本地机器设备列表*/
39     if (pcap_findalldevs_ex((char*)PCAP_SRC_IF_STRING, NULL/*auth is not
           needed*/, &alldevs, errbuf) == -1)
40     {
41         fprintf(stderr, "Error in pcap_findalldevs_ex:%s\n", errbuf);
42         exit(1);
43     }
44     /*打印列表*/
45     for (d = alldevs; d != NULL; d = d->next)
46     {
47         printf("%d. %s", num++, d->name);
48         if (d->description)
49             printf(" (%s)\n", d->description);
50         else
51             printf(" (No description available)\n");
52     }
53
54     if (num == 0)
55     {
56         printf("\n No interfaces found!\n");
```

```
57          exit(1);
58      }
59
60      int n;
61      cout << "请输入要打开的网络接口号" << " (0~" << num-1 << ") : " << endl;
62      cin >> n;
63      num = 0;
64
65      for (d = alldevs; num < (n);num++)
66      {
67          d = d->next;
68      }
69
70      pcap_t * adhandle = pcap_open(
71          d->name, // name of the device
72          65536, // portion of the packet to capture
73          // 65536 guarantees that the whole packet
74          // will be captured on all the link layers
75          PCAP_OPENFLAG_PROMISCUOUS | // promiscuous mode
76          PCAP_OPENFLAG_NOCAPTURE_LOCAL | PCAP_OPENFLAG_MAX_RESPONSIVENESS,
77          1000, // read timeout
78          NULL,
79          errbuf); // error buffer
80      if (adhandle == NULL)
81      {
82          cout << "产生错误, 无法打开设备" << endl;
83          pcap_freealldevs(alldevs);
84          return 0;
85      }
86      else
87      {
88          cout << "监听: " << d->description << endl;
89          pcap_freealldevs(alldevs);
90
91      }
92
93      int read_count;
94      cout << "请输入你要捕获的数据包的个数: " << endl;
95      cin >> read_count;
96      pcap_loop(adhandle, read_count, (pcap_handler)pcap_callback, NULL);
97      pcap_close(adhandle);
98
99      return 0;
100 }
101
102 void pcap_callback(u_char* argument, const struct pcap_pkthdr* packet_header,
        const u_char* packet_content) //packet_content表示的捕获到的数据包的内容
103 {
```

```
104     const u_char* ethernet_protocol;            //以太网协议
105     u_short ethernet_type;                  //以太网类型
106     //获取以太网数据内容
107     ethernet_protocol = packet_content;
108     ethernet_type = (ethernet_protocol[12] << 8) | ethernet_protocol[13];
109     printf("Mac源地址: \n");
110     printf("%02x:%02x:%02x:%02x:%02x:%02x:\n",
111         (ethernet_protocol[6]),
112         (ethernet_protocol[7]),
113         (ethernet_protocol[8]),
114         (ethernet_protocol[9]),
115         (ethernet_protocol[10]),
116         (ethernet_protocol[11])
117     );
118     printf("Mac目的地址: \n");
119     printf("%02x:%02x:%02x:%02x:%02x:%02x:\n",
120         (ethernet_protocol[0]),
121         (ethernet_protocol[1]),
122         (ethernet_protocol[2]),
123         (ethernet_protocol[3]),
124         (ethernet_protocol[4]),
125         (ethernet_protocol[5])
126     );
127     printf("以太网类型为␣:\t");
128     printf("%04x\n", ethernet_type);
129     switch (ethernet_type)
130     {
131     case 0x0800:
132         printf("网络层是: IPv4协议\n");
133         break;
134     case 0x0806:
135         printf("网络层是: ARP协议\n");
136         break;
137     case 0x8035:
138         printf("网络层是: RARP协议\n");
139         break;
140     default:
141         printf("网络层协议未知\n");
142         break;
143     }
144     if (ethernet_type == 0x0800)
145     {
146         IP_Packet_Handle(packet_header, packet_content);
147     }
148 }
149
150 void IP_Packet_Handle(const struct pcap_pkthdr* packet_header, const u_char*
        packet_content)
```

```
151 {
152     IPHeader_t* IPHeader;
153     IPHeader = (IPHeader_t*)(packet_content + 14);//IP包的内容在原有物理帧后
            14字节开始
154     sockaddr_in source, dest;
155     char sourceIP[16], destIP[16];
156     source.sin_addr.s_addr = IPHeader->SrcIP;
157     dest.sin_addr.s_addr = IPHeader->DstIP;
158     strncpy(sourceIP, inet_ntoa(source.sin_addr), 16);
159     strncpy(destIP, inet_ntoa(dest.sin_addr), 16);
160     printf("版本: %d\n", IPHeader->Ver_HLen >> 4);
161     printf("IP协议首部长度: %d␣Bytes\n", (IPHeader->Ver_HLen & 0x0f) * 4);
162     printf("服务类型: %d\n", IPHeader->TOS);
163     printf("总长度: %d\n", ntohs(IPHeader->TotalLen));
164     printf("标识: 0x%.4x␣(%i)\n", ntohs(IPHeader->ID));
165     printf("标志: %d\n", ntohs(IPHeader->Flag_Segment));
166     printf("片偏移: %d\n", (IPHeader->Flag_Segment) & 0x8000 >> 15);
167     printf("生存时间: %d\n", IPHeader->TTL);
168     printf("协议号: %d\n", IPHeader->Protocol);
169     printf("协议种类: ");
170     switch (IPHeader->Protocol)
171     {
172     case 1:
173         printf("ICMP\n");
174         break;
175     case 2:
176         printf("IGMP\n");
177         break;
178     case 6:
179         printf("TCP\n");
180         break;
181     case 17:
182         printf("UDP\n");
183         break;
184     default:
185         break;
186     }
187     printf("首部检验和: 0x%.4x\n", ntohs(IPHeader->Checksum));
188     printf("源地址: %s\n", sourceIP);
189     printf("目的地址: %s\n", destIP);
190     cout << "
            --------------------------------------------------------------------------------
            " << endl;
191 }
```

五、 总结

本次实验编写了抓包程序，深入理解了相应原理，对网络技术有了更深的了解。