



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

<b>一、 Npcap 概述</b>	<b>1</b>
(一) 简介 . . . . .	1
(二) 原理 . . . . .	1
(三) 使用函数 . . . . .	2
1. 获取设备列表 . . . . .	2
2. 网卡设备打开方式 . . . . .	2
3. 数据包捕获方法 . . . . .	3
<b>二、 数据包结构</b>	<b>4</b>
<b>三、 结果展示</b>	<b>4</b>
<b>四、 完整代码</b>	<b>5</b>
<b>五、 总结</b>	<b>10</b>

## 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）。

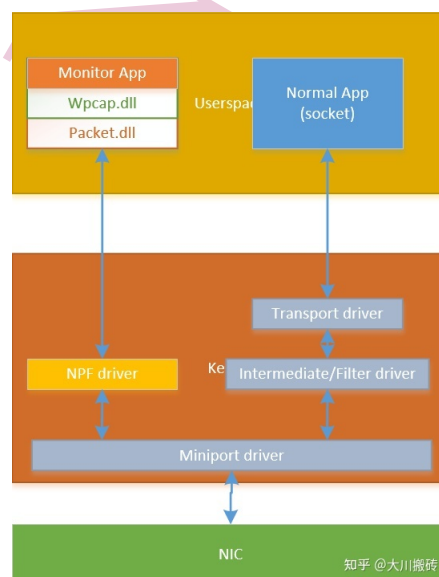


图 1: Caption

### (三) 使用函数

#### 1. 获取设备列表

获取设备列表

```
1 if (pcap_findalldevs_ex((char*)PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) ==  
    -1)  
2 {  
3     fprintf(stderr, "Error in pcap_findalldevs_ex:%s\n", errbuf);  
4     exit(1);  
5 }
```

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

```
1 for (d = alldevs; num < (n); num++)  
2 {  
3     d = d->next;  
4 }  
5  
6 pcap_t * adhandle = pcap_open(  
7     d->name, // name of the device  
8     65536, // portion of the packet to capture  
9     PCAP_OPENFLAG_PROMISCUOUS | // promiscuous mode  
10    PCAP_OPENFLAG_NOCAPTURE_LOCAL | PCAP_OPENFLAG_MAX_RESPONSIVENESS,  
11    1000, // read timeout  
12    NULL,  
13    errbuf); // error buffer  
14 if (adhandle == NULL)  
15 {  
16     cout << "产生错误，无法打开设备" << endl;  
17     pcap_freealldevs(alldevs);  
18     return 0;  
19 }  
20 else  
21 {  
22     cout << "监听：" << d->description << endl;  
23     pcap_freealldevs(alldevs);  
24 }  
25 }
```

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
1 void pcap_callback(u_char* argument, const struct pcap_pkthdr* packet_header,
2   const u_char* packet_content) //packet_content表示的捕获到的数据包的内容
3 {
4   const u_char* ethernet_protocol; //以太网协议
5   u_short ethernet_type; //以太网类型
6   //获取以太网数据内容
7   ethernet_protocol = packet_content;
8   ethernet_type = (ethernet_protocol[12] << 8) | ethernet_protocol[13];
9   printf("Mac 源地址: \n");
10  printf("%02x:%02x:%02x:%02x:%02x:%02x:\n",
11    ethernet_protocol[6],
12    ethernet_protocol[7],
13    ethernet_protocol[8],
14    ethernet_protocol[9],
15    ethernet_protocol[10],
16    ethernet_protocol[11]);
17  printf("Mac 目的地址: \n");
18  printf("%02x:%02x:%02x:%02x:%02x:%02x:\n",
19    ethernet_protocol[0],
20    ethernet_protocol[1],
21    ethernet_protocol[2],
22    ethernet_protocol[3],
23    ethernet_protocol[4],
24    ethernet_protocol[5]);
25  );
26  printf("以太网类型为: \t");
27  printf("%04x\n", ethernet_type);
28  switch (ethernet_type)
29  {
30  case 0x0800:
31    printf("网络层是: IPv4 协议\n");
32    break;
33  case 0x0806:
34    printf("网络层是: ARP 协议\n");
35    break;
36  case 0x8035:
```

```
37     printf("网络层是：RARP协议\n");
38     break;
39 default:
40     printf("网络层协议未知\n");
41     break;
42 }
43 if (ethernet_type == 0x0800)
44 {
45     IP_Packet_Handle(packet_header, packet_content);
46 }
47 }
```

此外我们还定义了一个结构体如下以方便获取 IP 地址：

IPHeader<sub>t</sub>

```
1 typedef struct IPHeader_t { //IP首部
2     BYTE Ver_HLen; //IP协议版本和IP首部长度的高4位为版本，低4位为首部的长度
3     BYTE TOS; //服务类型
4     WORD TotalLen; //总长度
5     WORD ID; //标识
6     WORD Flag_Segment; //标志 片偏移
7     BYTE TTL; //生存周期
8     BYTE Protocol; //协议
9     WORD Checksum; //头部校验和
10    u_int SrcIP; //源IP
11    u_int DstIP; //目的IP
12 }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。

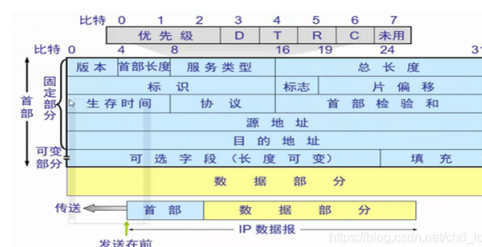


图 2: Caption

## 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

```
0. rpcap://Device\NPF_{3AB1ED82-365E-4982-88A1-61A6FA1EC4DE} (Network adapter 'WAN Miniport (Network Monitor)' on local host)
1. rpcap://Device\NPF_{24915405-F862-4857-B280-4B3A1B35AF0} (Network adapter 'WAN Miniport (IPv6)' on local host)
2. rpcap://Device\NPF_{C9201299-79B6-4FAF-BA51-EDCF8088629D} (Network adapter 'WAN Miniport (IP)' on local host)
3. rpcap://Device\NPF_{D108F9BC-8AD7-44BA-811E-AB4E24CABCF2} (Network adapter 'Bluetooth Device (Personal Area Network)' on local host)
4. rpcap://Device\NPF_{77544871-EF26-4988-94E8-EEA9C0D6AF61} (Network adapter 'Intel(R) Wi-Fi 6 AX201 160MHz' on local host)
5. rpcap://Device\NPF_{E9E5FEA6-C70A-428F-8138-80F9B0178418} (Network adapter 'VMware Virtual Ethernet Adapter for VMnet8' on local host)
6. rpcap://Device\NPF_{524EFF59-FA85-4E22-8D32-528477E66FFE} (Network adapter 'VMware Virtual Ethernet Adapter for VMnet1' on local host)
7. rpcap://Device\NPF_{5E931949-E14F-4E7E-87BD-248999999C97} (Network adapter 'Microsoft Wi-Fi Direct Virtual Adapter #2' on local host)
8. rpcap://Device\NPF_{DC78E10B-F16E-46B7-87F5-1595A9E3C7E2} (Network adapter 'Microsoft Wi-Fi Direct Virtual Adapter' on local host)
9. rpcap://Device\NPF_{Loopback} (Network adapter 'Adapter for loopback traffic capture' on local host)
10. rpcap://Device\NPF_{3226803A-D19A-4159-803F-A486124EE038} (Network adapter 'TAP-Windows Adapter V9' on local host)
11. rpcap://Device\NPF_{041167E8-CF34-4D6A-8D22-A649BA571542} (Network adapter 'Realtek PCIe GbE Family Controller' on local host)
请输入要打开的网络接口号 (0-11) :
```

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

```
请输入要打开的网络接口号 (0-11) :
4
监听: Network adapter 'Intel(R) Wi-Fi 6 AX201 160MHz' on local host
请输入你要捕获的数据包的个数:
```

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

```
监听: Network adapter 'Intel(R) Wi-Fi 6 AX201 160MHz' on local host
请输入你要捕获的数据包的个数:
1
源地址:
目的地址:
以太网类型为: 0800
网络层: IPv4协议
版本: 4
IP协议首部长度: 20 Bytes
总长度: 41
标识: 0x1282 (~641)
标志: 16384
片偏移: 0
生存时间: 128
协议号: 6
协议种类: TCP
首部校验和: 0x0000
源地址: 10.128.184.49
目的地址: 223.6.6.6
```

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

```
1 #include<Winsock2.h>
2 #include<iostream>
3 #include<pcap.h>
4 #include<stdio.h>
5 #include<time.h>
6 #include<string>
7 #pragma comment(lib, "ws2_32.lib")
8 #pragma comment(lib, "packet.lib")
9 #pragma comment(lib, "wpcap.lib")
10 #pragma comment(lib, "wsock32.lib")
11 #pragma warning(disable : 4996)
12
```

```

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("\nNo 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,
103     const u_char* packet_content) //packet_content表示的捕获到的数据包的内容
104 {
```

```
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\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。

NIKU