南開大學

网络技术与应用课程实验报告

实验 **2**：数据包捕获与分析

学院：网络空间安全学院

专业：信息安全-法学

学号：2111954

姓名：许积君

目录

一、 了解 NPcap 的架构

NPcap 是 Windows 平台上的一个网络数据包捕获库，它的设计目标是提供高性能的数据包捕获能力，以便开发者可以构建网络监控、数据包分析和网络安全工具，支持 NDIS 6 技术、“只允许管理员 Administrator”访问 Npcap、与 WinPcap 兼容或并存两种模式；支持 Windows 平台的回环数据包采集和发送。

NPcap 主要分为以下几个关键部分：

1. NPF 驱动程序（Network Packet Filter）：NPF 是 NPcap 的核心组件，是一个运行在 Windows 内核中的内核模式驱动程序。它负责实际的数据包捕获和注入操作。NPF 驱动程序提供高性能的网络数据包捕获功能，允许用户模式应用程序访问网络流量。它实现了网络数据包的捕获、过滤、注入和分发。

2. 用户模式库：NPcap 包括一组用户模式库，这些库用于与 NPF 驱动程序通信并提供易于使用的 API，以便用户模式应用程序可以访问捕获功能。这些库包括 npf.sys、wpcap.dll 和 Packet.dll。通常，应用程序使用这些库来与 NPcap 进行交互，而不需要直接操作内核模式驱动程序。

3. wpcap.dll：wpcap.dll 提供与 WinPcap 兼容的 API，允许大多数使用 WinPcap 的应用程序在 NPcap 上运行而无需修改代码。这有助于应用程序迁移和支持。

4. Packet.dll：Packet.dll 是 NPcap 的另一个用户模式库，支持与 NDIS 6.x 驱动程序和 Windows Filtering Platform（WFP）的集成。这使得 NPcap 可以与 Windows 的网络过滤功能集成，为应用程序提供更多的灵活性和控制权。

5. 高级功能：NPcap 提供了一些高级功能，如支持循环捕获模式（Ring Buffer）、多核 CPU 支持、混杂模式、过滤器和数据包注入等。这些功能扩展了 NPcap 的功能，使其适用于各种网络监控、流量分析和网络安全应用。

总之，NPcap 主要分为内核模式的 NPF 驱动程序和用户模式的库，它提供了高性能的数据包捕获功能，允许应用程序访问网络流量，并支持各种高级功能，以满足不同类型的网络应用需求。这使 NPcap 成为一个强大的工具，用于网络监控、数据包分析、网络安全和其他网络相关任务。

二、 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

在使用 NPcap 对数据包进行捕获分析的时候，通常要完成“获取设备列表 → 打开网络接口 → 获取数据包并分析”几个步骤，我们将在 Visual Studio 2022 中配置使用 NPcap

（一） 获取设备列表

1. 重要函数原型和参数分析

```
PCAP_AVAILABLE_1_9
PCAP_API int    pcap_findalldevs_ex(const char *source,
            struct pcap_rmtauth *auth, pcap_if_t **alldevs, char
                *errbuf);
```

- const char *source: 用于指定捕获源。通常情况下,可以将其设置为 PCAP_SRC_IF_STRING, 表示要获取本地网络接口的列表。你也可以将其设置为其他捕获源, 例如一个网络设备、一个文件或一个远程捕获服务器的地址
- struct pcap_rmtauth *auth：一个指向 pcap_rmtauth 结构的指针，用于指定远程捕获时的认证信息。如果不需要远程捕获，可以将其设置为 NULL
- pcap_if_t **alldevs：一个指向 pcap_if_t 结构的指针的指针，用于接收捕获到的网络接口设备信息。这个参数将由函数填充，以便你可以在后续的代码中使用这些信息
- char *errbuf: 一个字符数组, 用于存储错误消息。如果在查找网络接口设备时发生错误, 错误消息将被存储在这个缓冲区中，以便你可以检查并处理错误

```
struct pcap_if {
    struct pcap_if *next;
    char *name;             /* name to hand to "pcap_open_live()" */
    char *description;      /* textual description of interface, or
        NULL */
    struct pcap_addr *addresses;
    bpf_u_int32 flags;      /* PCAP_IF_ interface flags */
};
```

- struct pcap_if *next：指向下一个网络接口结构体的指针。如果有多个网络接口，这个字段用于构建一个链表，将它们连接在一起，以便遍历所有可用的网络接口
- char *name：一个指向字符数组的指针，用于存储网络接口的名称。这个名称通常用于传递给 pcap_open_live() 函数，以打开特定的网络接口
- char *description：一个指向字符数组的指针，用于存储网络接口的文本描述信息。这个字段通常提供了有关网络接口的描述，如接口的用途、位置或其他信息。它可以是可读性更好的信息，但不一定是唯一的
- struct pcap_addr *addresses：一个指向 struct pcap_addr 结构体的指针，用于存储有关网络接口的地址信息。这个字段可能包括接口的 IPv4 或 IPv6 地址、子网掩码、广播地址等
- bpf_u_int32 flags：一个整数，表示网络接口的标志。这些标志通常是一些常量，用于表示接口的属性，如是否支持混杂模式、是否支持监听模式等。这些标志可以使用位运算来进行设置和检查

2. 功能实现结构

```
pcap_if_t* allAdapters;// 所有网卡设备保存
char errbuf[PCAP_ERRBUF_SIZE];//错误缓冲区，大小为256
int index = 1;
// 获取本地机器设备列表，并打印
//pcap.h抓包库中的函数
if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &allAdapters, errbuf) !=
    -1)
```

```
{       /* 打印网卡信息列表 */
    pcap_if_t* ptr;
    for (ptr = allAdapters; ptr != NULL; ptr = ptr->next)
    {
        if (ptr->description)
            printf("ID %d  Name: %s \n", index, ptr->description);
        index++;
    }
    index--;
}
else
{
    printf("Error in pcap_findalldevs_ex: %s\n", errbuf);
}
```

（二） 打开网卡

1. 重要函数原型和参数分析

```
PCAP_AVAILABLE_0_4
PCAP_API pcap_t *pcap_open_live(const char *, int, int, int, char *);
```

- const char *device：这是要打开的网络设备的名称或者接口的标识符。你需要提供要捕获数据包的网络接口的名称，例如，”eth0”、”en0” 或者”Wi-Fi” 等
- int snaplen：这是捕获的每个数据包的最大长度，通常以字节为单位。例如，如果你将其设置为 65536，则会捕获完整的以太网帧。如果设置得太小，数据包可能会被截断
- int promisc：这是一个标志，用于指示是否在混杂模式下进行捕获。如果将其设置为非零值，表示启用混杂模式，允许捕获所有经过网络接口的数据包，而不仅仅是目的地址与接口匹配的数据包。如果将其设置为零，表示禁用混杂模式
- int to_ms：这是设置捕获超时的参数，以毫秒为单位。如果你希望在某个时间段内捕获数据包，可以设置该值。通常，如果你希望无限期地捕获数据包，可以将其设置为零
- char *ebuf：这是用于存储错误消息的缓冲区。如果函数调用失败，错误消息将存储在该缓冲区中。你需要分配足够大的缓冲区以存储可能的错误消息
- 返回值：如果函数成功打开网络设备并返回一个 pcap_t 结构体指针，否则返回 NULL

2. 功能实现

```
//打开网卡
pcap_t* pcap_handle = pcap_open_live(ptr->name,     //设备名称
    65536,
                        //包长度最大值 65536允许整个包在所有mac电脑上被捕获
    PCAP_OPENFLAG_PROMISCUOUS,                                  /* 混杂模式*/
    1000,
                        //读超时为1秒
```

```
    errbuf);
                //错误缓冲池;//打开网络适配器，捕捉实例,是pcap_open返回的对象
```

(三) 数据包捕获

1. 重要函数原型和参数分析

```
PCAP_AVAILABLE_0_4
PCAP_API int    pcap_loop(pcap_t *, int, pcap_handler, u_char *);
```

- pcap_t *：这是一个指向 pcap_t 结构的指针，表示一个已打开的抓包会话。可以使用 pcap_open_live 或其他类似的函数打开一个网络接口，并将其返回的 pcap_t 结构传递给 pcap_loop 以捕获数据包
- int：这是一个整数参数，用于指定捕获的数据包数量。如果将其设置为负数，如 -1，则表示无限捕获数据包，直到调用者手动停止
- pcap_handler：这是一个回调函数，用于处理捕获到的数据包, 它是用户定义的函数
- u_char *：这是一个用户定义的指针，通常用于传递自定义数据给回调函数中的 user 参数
- pcap_loop 函数将在捕获数据包时循环执行回调函数，直到达到指定的捕获数量或用户手动停止。这允许应用程序捕获和处理数据包，进行网络分析和监控等任务

2. 功能实现

```
pcap_loop(pcap_handle, num, ethernet_protocol_packet_callback, NULL);
ethernet_protocol_packet_callback在上面定义，用于处理数据包
```

三、 NPcap 编程

(一) Visual Studio 2022 和 NPcap 配置

1. 点击路径” 项目 → lab2 设置”
2. 在” 配置属性 → VC++ 目录 → 包含目录” 添加”`C:\Program Files\Npcap\npcap-sdk-1.07\Include`”, 在“配置属性 → VC++ 目录 → 库目录”添加”`C:\Program Files\Npcap\npcap-sdk-1.07\Lib`”

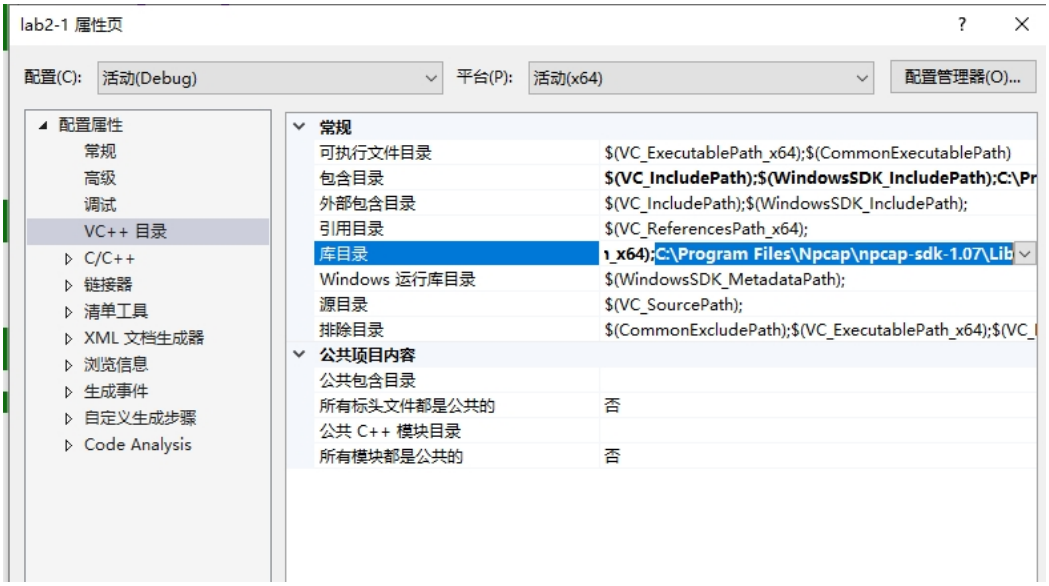

图 1: VC++ 配置

3. 在”C/C++ → 预处理器 → 预处理器定义” 添加”WIN32;WPCAP;HAVE_REMOTE”

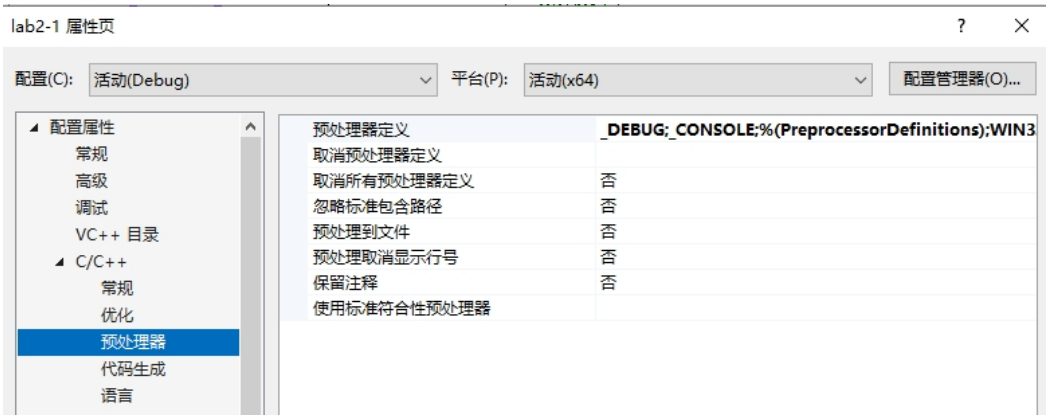

图 2: 预处理器配置

4. 在” 链接器 → 常规 → 附加库目录” 添加”`C:\Program Files\Npcap\npcap-sdk-1.07\Lib\x64`”, 在” 输入 → 附加依赖项” 添加”wpcap.lib;Packet.lib”

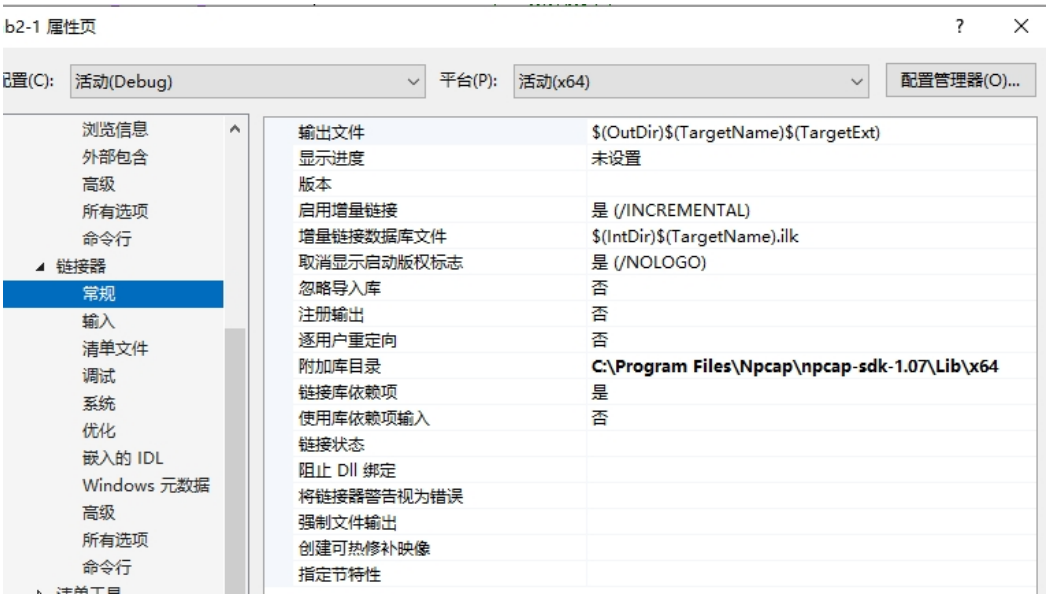

图 3: 链接器常规配置

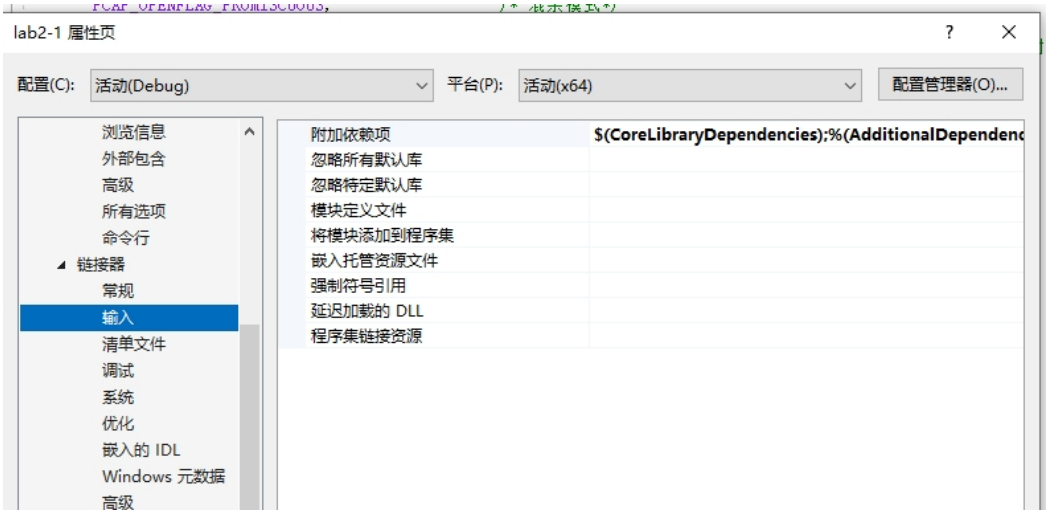

图 4: 链接器输入配置

（二） 项目源代码

```
#include <Winsock2.h>
#include <Windows.h>
#include <iostream>
#include <ws2tcpip.h>
#include "pcap.h"
#include "stdio.h"
#include <time.h>
#include <string>

#pragma comment(lib, "Packet.lib")
#pragma comment(lib,"wpcap.lib")
#pragma comment(lib,"ws2_32.lib")//表示链接的时侯找ws2_32.lib
```

```
#pragma warning( disable : 4996 )//要使用旧函数
#define _WINSOCK_DEPRECATED_NO_WARNINGS

using namespace std;

#pragma pack(1)
struct ethernet_header
{
    uint8_t DesMAC[6];  //  目的地址
    uint8_t SrcMAC[6];//    源网地址
    uint16_t FrameType;//   帧类型
};

struct ip_header
{
    uint8_t Header_Length : 4,  //首部长度
        Version : 4;                                    //版本
    uint8_t TOS;                                        //服务类型
    uint16_t Total_Length;                  //总长度
    uint16_t Id;                                        //标识,用于将分片的IP数据包重
        新组装为完整数据包
    uint8_t TTL;                                        //生存时间,指示数据包在网络中
        的最大寿命, 以避免在网络中无限循环
    uint8_t Protocol;                       //协议类型(TCP/UDP/ICMP)
    uint16_t Header_Check;                  //首部检验和,用于检测IP首部是否损坏
    struct in_addr Ip_Src_Addr; //源IP (struct表示一个32位的IPv4地址)
    struct in_addr Ip_Des_Addr; //目的IP
    uint16_t ip_offset;//片偏移

};
void ethernet_protocol_packet_callback(u_char* argument, const pcap_pkthdr*
    packet_header, const u_char* packet_content)
{
    uint16_t FrameType;
    ethernet_header* Protocol = (ethernet_header*)packet_content;
    uint8_t* SrcMAC;
    uint8_t* DesMAC;
    static int packet_number = 1;//抓包数量

    FrameType = ntohs(Protocol->FrameType);//获得以太网类型
    Protocol = (ethernet_header*)packet_content;//获得以太网协议数据内容
    SrcMAC = Protocol->SrcMAC;//Mac源地址
    DesMAC = Protocol->DesMAC;//Mac目的地址

    cout << "========================================================================" <<
        endl;
    printf("第【 %d 】个IP数据包被捕获\n", packet_number);
    printf("以太网类型为 :%04x\n", FrameType);
```

```
57
58      // 输出帧长度
59      printf("帧长度: %d bytes\n", packet_header->len);
60
61      switch (FrameType)//判断以太网类型的值
62      {
63      case 0x0800:
64          cout << "网络层使用的是IPv4协议" << endl;
65          break;
66      case 0x08DD:
67          cout << "网络层使用的是IPv6协议" << endl;
68          break;
69      case 0x0806:
70          cout << "网络层使用的是ARP协议" << endl;
71          break;
72      case 0x8100:
73          cout << "网络层使用的是VLAN 标签帧" << endl;
74          break;
75      case 0x8035:
76          cout << "网络层使用的是RARP协议" << endl;
77          break;
78      case 0x8137:
79          cout << "网络层使用的是IPX 协议" << endl;
80          break;
81      default:
82          break;
83      }
84      //获得Mac源地址
85      printf("Mac源地址:\t%02x:%02x:%02x:%02x:%02x:%02x:\n", *SrcMAC, *(SrcMAC
            + 1), *(SrcMAC + 2), *(SrcMAC + 3), *(SrcMAC + 4), *(SrcMAC + 5));
86      //获得Mac目的地址
87      printf("Mac目的地址:\t%02x:%02x:%02x:%02x:%02x:%02x:\n", *DesMAC, *(
            DesMAC + 1), *(DesMAC + 2), *(DesMAC + 3), *(DesMAC + 4), *(DesMAC +
            5));
88      packet_number++;
89  }
90  void Capture()
91  {
92      pcap_if_t* allAdapters;// 所有网卡设备保存
93      char errbuf[PCAP_ERRBUF_SIZE];//错误缓冲区, 大小为256
94      int index = 1;
95      // 获取本地机器设备列表, 并打印
96      //pcap.h抓包库中的函数
97      if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &allAdapters, errbuf)
            != -1)
98      {   /* 打印网卡信息列表 */
99          pcap_if_t* ptr;
100         for (ptr = allAdapters; ptr != NULL; ptr = ptr->next)
```

```
101         {
102             if (ptr->description)
103                 printf("ID %d  Name: %s \n", index, ptr->description);
104             index++;
105         }
106         index--;
107     }
108     else
109     {
110         printf("Error in pcap_findalldevs_ex: %s\n", errbuf);
111     }
112 
113     if (index == 0)
114     {
115         cout << "没有找到接口" << endl;
116     }
117 
118     //打开想要监控的网卡
119     cout << "请输入想要监控的网卡的ID" << endl;
120     int num;
121     cin >> num;
122 
123     while (num < 1 || num > index)
124     {
125         cout << "不存在该设备, 请重新输入合适的ID" << endl;
126         cin >> num;
127     }
128 
129     int i = 0;
130     pcap_if_t* ptr;
131     for (ptr = allAdapters, i = 0; i < num - 1; ptr = ptr->next, i++);
132 
133     //选择网卡
134 
135      //打开网卡
136     pcap_t* pcap_handle = pcap_open_live(ptr->name,     //设备名称
137         65536,
                                //包长度最大值 65536允许整个包在所有mac电脑上被捕
                获
138         PCAP_OPENFLAG_PROMISCUOUS,                                  /* 混
                杂模式*/
139         1000,
                                //读超时为1秒
140         errbuf);
                                //错误缓冲池;//打开网络适配器, 捕捉实例,是
                pcap_open返回的对象
141     if (pcap_handle == NULL)
142     {
```

```
143         cout << "无法打开该网卡接口" << endl;
144         pcap_freealldevs(allAdapters);
145         exit(0);
146     }
147 
148     cout << "正在监听" << ptr->description << endl;
149     //不再需要设备列表，释放
150     pcap_freealldevs(allAdapters);
151 
152     cout << "请输入想要捕获数据包的个数:" << endl;
153     cin >> num;
154     //-1表示无限捕获，0表示捕获所有数据包，直到读取到EOF
155 
156     pcap_loop(pcap_handle, num, ethernet_protocol_packet_callback, NULL);
157     //捕获数据包,不会响应pcap_open_live()函数设置的超时时间
158     cout << "解析ip数据包结束" << endl;
159 }
160 
161 int main()
162 {
163     Capture();
164     system("Pause");
165     return 0;
166 }
```

(三) 输出结果

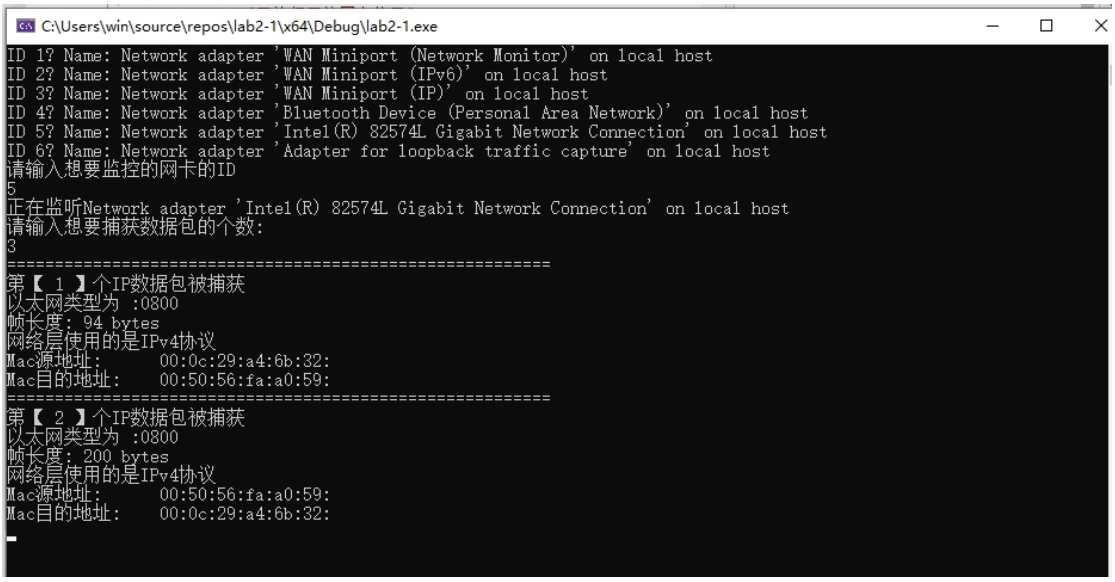

图 5: 输出结果