

实验2：数据包捕获与分析

姓名：于成俊

学号：2112066

一、了解NPcap的架构

Npcap 是一种用于数据包捕获和网络分析的架构，用于 Windows 操作系统。它包括一个内核级数据包过滤器（NPF），一个低级动态链接库（packet.dll），以及高级的独立于系统的库（WPCAP.dll）。

- NPF：被实现为一个过滤驱动程序，能够执行许多不同的操作：捕获，监控，数据包注入。NPF 最重要的操作是数据包捕获。在捕获过程中，驱动程序使用网络接口侦听数据包，并将它们完整地传递给用户级别的应用程序，它还允许将原始数据包写入网络，还提供了一个内核级可编程监控模块，能够对网络流量进行简单的统计计算。
- Packet.dll：提供一个底层的API，可用来直接访问驱动程序。
 - 安装，启动和停止NPF设备驱动
 - 从NPF驱动接收数据包
 - 通过NPF驱动发送数据包
 - 获取可用的网络适配器列表
 - 获取适配器的不同信息，比如设备描述，地址列表和掩码
 - 查询并设置一个底层的适配器参数
- WPCAP.dll：提供了更强大的、更高层的捕获函数接口。

二、学习NPcap的设备列表获取方法

- pcap_if 结构体，用来保存网络接口设备基本信息类型

```
//在 pcap.h 中定义
typedef u_int bpf_u_int32
typedef struct pcap_if pcap_if_t
struct pcap_if
{
    struct pcap_if *next;           //指向下一个网络接口
    char *name;                     //网络接口的标识符，唯一标识一个网卡
    char *description;              //用来描述网络接口
    struct pcap_addr *addresses;    //pcap_addr 结构体
    bpf_u_int32 flags;              //接口标志
    //目前唯一可能的标志是PCAP_IP_LOOPBACK，如果接口是环回接口，则设置该标志
};
```

- pcap_addr 结构体，用来存储网卡的地址

```

struct pcap_addr
{
    struct pcap_addr *next;           //指向下一个 pcap_addr 结构的指针
    struct sockaddr *addr;            //存储网络接口的地址信息
    struct sockaddr *netmask;         //用于存储子网掩码（子网掩码用于确定 IP 地址
                                     的网络部分和主机部分）
    struct sockaddr *broadaddr;       //用于存储广播地址
    struct sockaddr *dstaddr;         //用于存储目标地址
};
//struct sockaddr 是一个通用的套接字地址结构，可以用来表示不同类型的地址，如 IPv4 或
IPv6。

```

- 调用 `pcap_findalldevs` 函数来获取可用的网络接口设备列表。
 - `pcap_findalldevs` 函数会遍历系统上所有的网络接口设备，包括网卡、虚拟接口等，然后获取每个接口的信息。
 - 对于每个网络接口，函数会获取接口的名称、描述、地址信息等，并将这些信息填充到 `pcap_if_t` 结构体中
 - `pcap_findalldevs` 函数将这些 `pcap_if_t` 结构体组织成一个链表。每个结构体中的 `next` 指针指向链表中的下一个元素，从而形成一个链表数据结构，表示系统上所有可用的网络接口。
 - 函数执行完成后，`alldevs` 指针将指向一个链表的头部，其中包含了系统上所有可用网络接口的信息。如果函数执行期间没有出现错误，`errbuf` 中将保持为空。

```

/*
  在pcap.h中, #define PCAP_ERRBUF_SIZE  256
*/
char errbuf[PCAP_ERRBUF_SIZE]; //用于存储错误信息

pcap_if_t* alldevs;
pcap_if_t* dev;

// 获取可用的网络设备列表
if (pcap_findalldevs(&alldevs, errbuf) == -1) {
    cerr << "Error in pcap_findalldevs: " << errbuf << endl;
    return 1;
}

// 遍历设备列表并打印设备信息
int i = 1;
for (dev = alldevs; dev; dev = dev->next) {
    cout << "Device " << i++ << ": " << dev->name << endl;
    if (dev->description) {
        cout << "    Description: " << dev->description << endl;
    }
    else {
        cout << "    Description: N/A" << endl;
    }
}
}

```

三、学习NPcap的网卡设备打开方法

- `pcap_t`结构体:

`pcap_t` 是 `npcap` 库中的一个结构体，用于表示数据包捕获的会话。它包含了与数据包捕获相关的各种设置和状态信息。通过这个结构体，你可以配置捕获参数、启动捕获会话、读取捕获的数据包等

```
//定义在pcap.h
typedef struct pcap pcap_t
```

- 使用 `pcap_open_live` 函数：

```
pcap_t* handle = NULL; //用于表示数据包捕获的会话。
// 打开网络接口以进行数据包捕获
handle = pcap_open_live(dev->name, BUFSIZ, 1, 1000, errbuf);
if (handle == NULL) {
    cerr << "Error opening network interface: " << errbuf << endl;
    return 1;
}
```

- `pcap_open_live` 函数用于打开一个网络接口以进行数据包捕获。
- `dev->name` 是选择的网络接口的名称，通常是从 `pcap_if_t` 结构体中获取的。
- `BUFSIZ` 是捕获数据包的缓冲区大小。
- `1` 表示混杂模式（promiscuous mode）的开启，允许捕获网络上的所有数据包。
- `1000` 是捕获超时时间（以毫秒为单位）。在这里，设置为1秒。
- `errbuf` 用于存储错误信息。
- 如果 `pcap_open_live` 函数成功打开网络接口，它将返回一个非空的 `pcap_t` 结构体指针，表示捕获会话。

四、学习Npcap数据包捕获方法

- `pcap_pkthdr` 结构体：用于存储捕获的数据包的元数据

```
struct pcap_pkthdr
{
    struct timeval ts; /* 捕获时间戳 */

    bpf_u_int32 caplen; /* 捕获到数据包的长度 */

    bpf_u_int32 len; /* 数据包的真正长度 */
}
```

- 使用 `pcap_next_ex` 直接捕获数据包

```
pcap_next_ex(pcap_t* p, struct pcap_pkthdr** pkt_header, const u_char**
pkt_data);
```

/*

功能：从interface或离线记录文件获取一个报文

参数： **p**：已打开的捕捉实例的描述符

pkt_header：报文头

pkt_data：报文内容

返回值：1：成功

0：获取报文超时

-1：发生错误

-2：获取到离线记录文件的最后一个报文

*/

```
// 开始捕获数据包
int result;
struct pcap_pkthdr *header;
// 用于存储捕获的数据包的内容
const u_char* packet;

while ((result = pcap_next_ex(handle, &header, &packet)) >= 0) {
    if (result == 0) {
        continue; // 没有数据包到达，继续等待
    }

    // 处理捕获的数据包
    packet_handler(header, packet);
}

if (result == -1) {
    cerr << "Error reading the packet: " << pcap_geterr(handle) << endl;
}
```

- 解析和显示捕获到的数据包信息

以太网帧结构

前导码 (7B)	帧前定界符 (1B)	目的地址 (6B)	源地址 (6B)	长度/类型 (2B)	数据 (可变长度, 46~1500B)	帧校验码 (4B)
-------------	---------------	--------------	-------------	---------------	------------------------	--------------

以太网帧头部指 目的地址+源地址+长度/类型，一共14字节

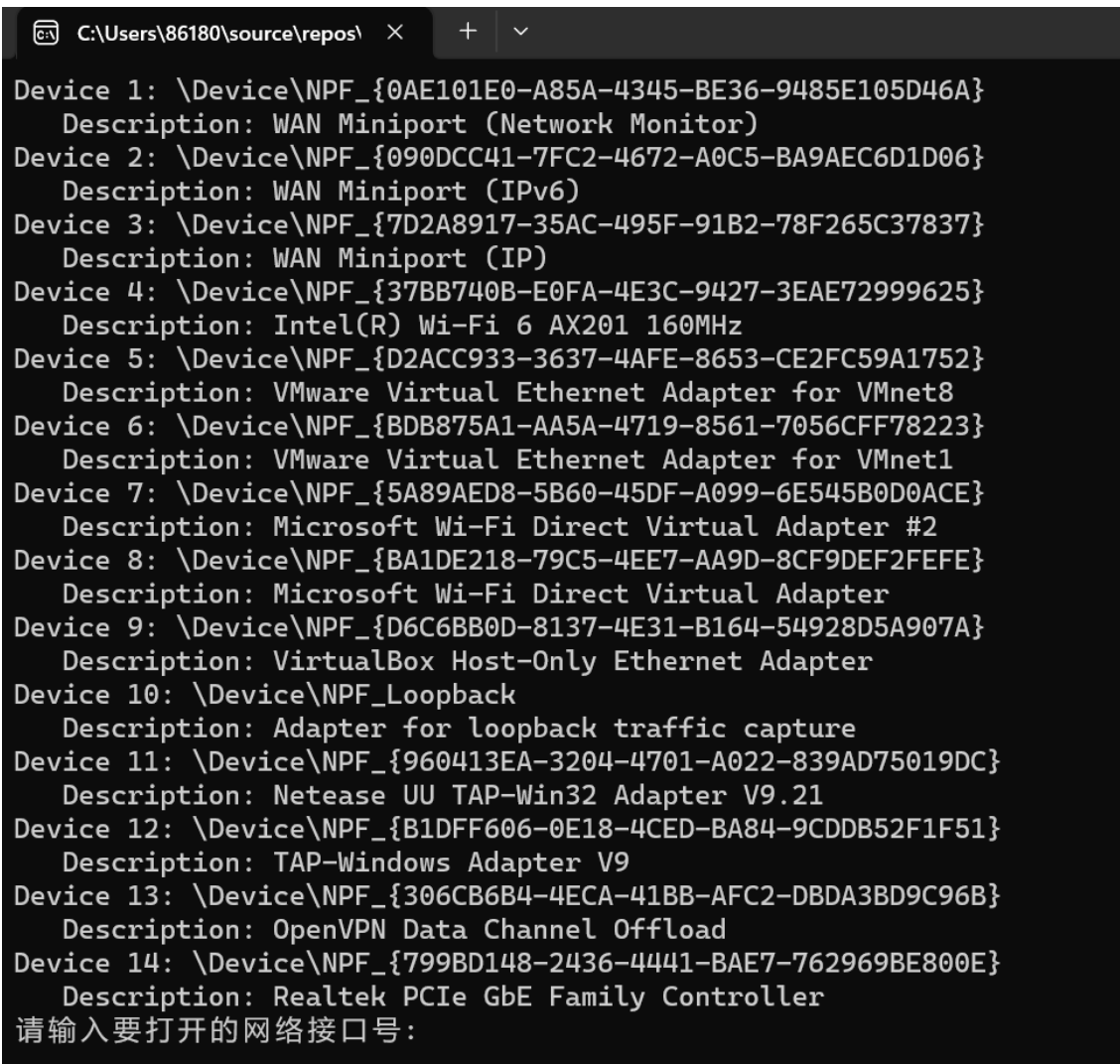
```

void packet_handler(const struct pcap_pkthdr* pkthdr, const u_char* packet)
{
    // 解析以太网头部
    const u_char* eth_header = packet;
    // 将两个字节的值合并为一个16位的整数
    unsigned short eth_type = (eth_header[12] << 8) | eth_header[13];
    printf("源 MAC 地址: %02X:%02X:%02X:%02X:%02X:%02X\n", eth_header[6],
eth_header[7], eth_header[8], eth_header[9], eth_header[10],
eth_header[11]);
    printf("目标 MAC 地址: %02X:%02X:%02X:%02X:%02X:%02X\n", eth_header[0],
eth_header[1], eth_header[2], eth_header[3], eth_header[4], eth_header[5]);
    printf("类型: 0x%04X\n", eth_type);
    printf("数据包的长度: %d\n", pkthdr->len);
    printf("\n");
}

```

五、程序演示

- 打印设备列表



```

C:\Users\86180\source\repos\ > netsh interface show interface
Device 1: \Device\NPF_{0AE101E0-A85A-4345-BE36-9485E105D46A}
Description: WAN Miniport (Network Monitor)
Device 2: \Device\NPF_{090DCC41-7FC2-4672-A0C5-BA9AEC6D1D06}
Description: WAN Miniport (IPv6)
Device 3: \Device\NPF_{7D2A8917-35AC-495F-91B2-78F265C37837}
Description: WAN Miniport (IP)
Device 4: \Device\NPF_{37BB740B-E0FA-4E3C-9427-3EAE72999625}
Description: Intel(R) Wi-Fi 6 AX201 160MHz
Device 5: \Device\NPF_{D2ACC933-3637-4AFE-8653-CE2FC59A1752}
Description: VMware Virtual Ethernet Adapter for VMnet8
Device 6: \Device\NPF_{BDB875A1-AA5A-4719-8561-7056CFF78223}
Description: VMware Virtual Ethernet Adapter for VMnet1
Device 7: \Device\NPF_{5A89AED8-5B60-45DF-A099-6E545B0D0ACE}
Description: Microsoft Wi-Fi Direct Virtual Adapter #2
Device 8: \Device\NPF_{BA1DE218-79C5-4EE7-AA9D-8CF9DEF2FEFE}
Description: Microsoft Wi-Fi Direct Virtual Adapter
Device 9: \Device\NPF_{D6C6BB0D-8137-4E31-B164-54928D5A907A}
Description: VirtualBox Host-Only Ethernet Adapter
Device 10: \Device\NPF_Loopback
Description: Adapter for loopback traffic capture
Device 11: \Device\NPF_{960413EA-3204-4701-A022-839AD75019DC}
Description: Netease UU TAP-Win32 Adapter V9.21
Device 12: \Device\NPF_{B1DFF606-0E18-4CED-BA84-9CDD852F1F51}
Description: TAP-Windows Adapter V9
Device 13: \Device\NPF_{306CB6B4-4ECA-41BB-AFC2-DBDA3BD9C96B}
Description: OpenVPN Data Channel Offload
Device 14: \Device\NPF_{799BD148-2436-4441-BAE7-762969BE800E}
Description: Realtek PCIe GbE Family Controller
请输入要打开的网络接口号:

```

- 打印捕获的数据包相关信息

其中:0x0800代表IPv4, 0x86DD代表IPv6

请输入要打开的网络接口号:

4

源 MAC 地址: 54:14:F3:02:14:5D

目标 MAC 地址: 00:00:5E:00:01:08

类型: 0x0800

数据包的长度: 55

源 MAC 地址: 00:00:5E:00:01:08

目标 MAC 地址: 54:14:F3:02:14:5D

类型: 0x0800

数据包的长度: 66

源 MAC 地址: 84:5B:12:5E:36:0B

目标 MAC 地址: 54:14:F3:02:14:5D

类型: 0x86DD

数据包的长度: 118

源 MAC 地址: 84:5B:12:5E:36:0B

目标 MAC 地址: 54:14:F3:02:14:5D

类型: 0x86DD

数据包的长度: 118

源 MAC 地址: 84:5B:12:5E:36:0B

目标 MAC 地址: 54:14:F3:02:14:5D

类型: 0x86DD

数据包的长度: 118