

廈門大學



信息学院软件工程系

《计算机网络》实验报告

题 目 实验三 侦听以太网和 IP 报文

组 名 第 26 组 饭友组

组 员 杨浩然

组 长 软件工程 2018 级 2 班杨浩然

实验时间 2020 年 3 月 11 日

2020 年 3 月 11 日

1 实验目的

捕获并分析以太网的帧，获取目标与源网卡的 MAC 地址

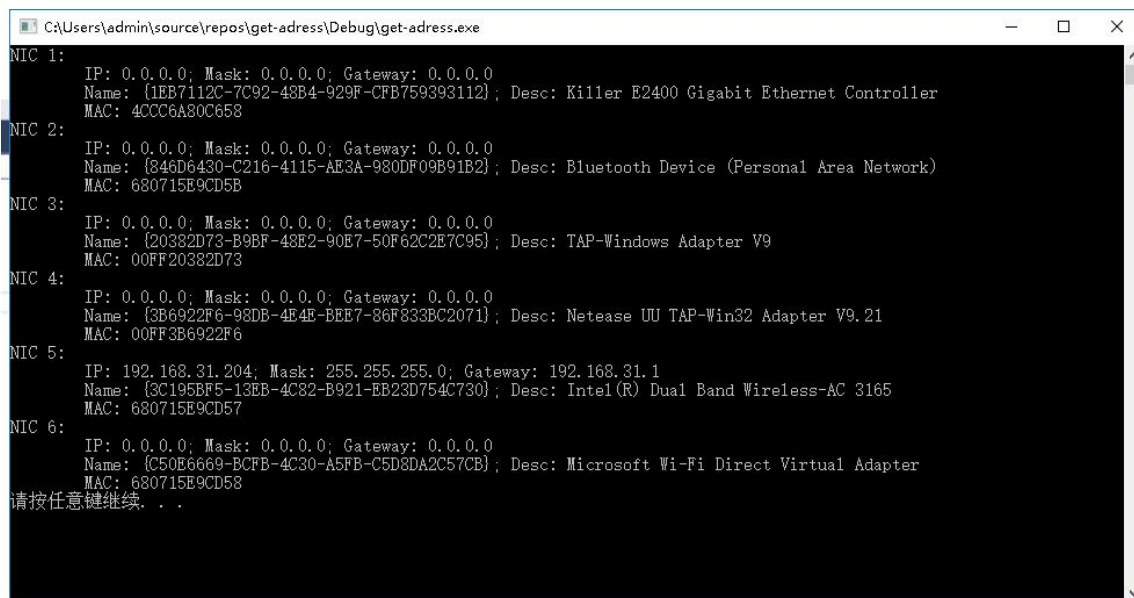
- 获取本机地址
 - IPCONFIG.EXE
 - 通过 WinSock 的 GetAddress 命令
- 获取远端 MAC 地址
 - ARP
 - WinPCAP

2 实验环境

Windows 10, visual Studio 2017, winpcap

3 实验结果

获取本机地址



```
C:\Users\admin\source\repos\get-adress\Debug\get-adress.exe
NIC 1:
  IP: 0.0.0.0; Mask: 0.0.0.0; Gateway: 0.0.0.0
  Name: {1EB7112C-7C92-48B4-929F-CFB759393112}; Desc: Killer E2400 Gigabit Ethernet Controller
  MAC: 4CCC6A80C658
NIC 2:
  IP: 0.0.0.0; Mask: 0.0.0.0; Gateway: 0.0.0.0
  Name: {846D6430-C216-4115-AE3A-980DF09B91B2}; Desc: Bluetooth Device (Personal Area Network)
  MAC: 680715E9CD5B
NIC 3:
  IP: 0.0.0.0; Mask: 0.0.0.0; Gateway: 0.0.0.0
  Name: {20382D73-B9BF-48E2-90E7-50F62C2E7C95}; Desc: TAP-Windows Adapter V9
  MAC: 00FF20382D73
NIC 4:
  IP: 0.0.0.0; Mask: 0.0.0.0; Gateway: 0.0.0.0
  Name: {3B6922F6-98DB-4E4E-BEE7-86F833BC2071}; Desc: Netease UU TAP-Win32 Adapter V9.21
  MAC: 00FF3B6922F6
NIC 5:
  IP: 192.168.31.204; Mask: 255.255.255.0; Gateway: 192.168.31.1
  Name: {3C195BF5-13EB-4C82-B921-EB23D754C730}; Desc: Intel(R) Dual Band Wireless-AC 3165
  MAC: 680715E9CD57
NIC 6:
  IP: 0.0.0.0; Mask: 0.0.0.0; Gateway: 0.0.0.0
  Name: {C50E6669-BCFB-4C30-A5FB-C5D8DA2C57CB}; Desc: Microsoft Wi-Fi Direct Virtual Adapter
  MAC: 680715E9CD58
请按任意键继续. . .
```

首先测试网卡

```

C:\Users\admin\Desktop\WpdPack_4_1_2\WpdPack\Examples-pcap\\Debug\x86\UDPdump.exe
5. \Device\NPF_{20382D73-B9BF-48E2-90E7-50F62C2E7C95} (TAP-Windows Adapter V9)
Enter the interface number (1-5):3

listening on Microsoft...
21:19:19.349873 len:86 192.168.31.204.55451 -> 61.151.164.197.9070
21:19:19.377655 len:86 61.151.164.197.9070 -> 192.168.31.204.55451
21:19:19.721886 len:80 192.168.31.204.50348 -> 192.168.31.1.53
21:19:19.734235 len:211 192.168.31.1.53 -> 192.168.31.204.50348
21:19:19.888672 len:129 182.254.110.91.8000 -> 192.168.31.204.4010
21:19:19.904913 len:86 192.168.31.204.55451 -> 113.250.20.98.9070
21:19:19.972035 len:86 113.250.20.98.9070 -> 192.168.31.204.55451
21:19:20.460895 len:86 192.168.31.204.55451 -> 123.151.190.92.9070
21:19:20.495414 len:86 123.151.190.92.9070 -> 192.168.31.204.55451
21:19:20.514261 len:129 182.254.110.91.8000 -> 192.168.31.204.4010
21:19:20.628076 len:545 182.254.110.91.8000 -> 192.168.31.204.4010
21:19:20.629035 len:97 192.168.31.204.4010 -> 182.254.110.91.8000
21:19:21.015844 len:86 192.168.31.204.55451 -> 183.3.254.56.9070
21:19:21.050139 len:86 183.3.254.56.9070 -> 192.168.31.204.55451
21:19:21.571256 len:86 192.168.31.204.55451 -> 123.151.177.75.9070
21:19:21.606274 len:86 123.151.177.75.9070 -> 192.168.31.204.55451
21:19:22.126676 len:86 192.168.31.204.55451 -> 113.250.22.95.9070
21:19:22.156142 len:86 113.250.22.95.9070 -> 192.168.31.204.55451
21:19:22.682148 len:86 192.168.31.204.55451 -> 157.255.246.65.9070
21:19:22.767627 len:86 157.255.246.65.9070 -> 192.168.31.204.55451
21:19:23.237663 len:86 192.168.31.204.55451 -> 220.194.108.140.9070
21:19:23.269754 len:86 220.194.108.140.9070 -> 192.168.31.204.55451
21:19:23.793025 len:86 192.168.31.204.55451 -> 116.128.128.58.9070
21:19:23.857248 len:81 192.168.31.204.4010 -> 182.254.110.91.8000
21:19:23.876679 len:86 116.128.128.58.9070 -> 192.168.31.204.55451

```

找到内存地址

```

186 pcap_freealldevs(alldevs);
187
188 /* start the capture */
189 pcap_loop(adhandle, 0, packet_handler, NULL);
190
191 return 0;
192
193
194 /* Callback function invoked by libpcap for every incoming packet */
195 void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char *pkt_data)
196 {
197     struct tm *ltime;
198     char timestr[16];
199     ip_header *ih;
200     udp_header *uh;
201     u_int ip_len;
202     u_short sport,dport;
203     time_t local_tv_sec;
204
205     /*
206      * unused parameter
207      */
208     (VOID)(param);
209
210     /* convert the timestamp to readable format */
211     local_tv_sec = header->ts.tv_sec;
212     ltime=localtime(&local_tv_sec);
213     strftime( timestr, sizeof timestr, "%H:%M:%S", ltime);
214
215     /* print timestamp and length of the packet */
216     printf("%s.%6d len:%d ", timestr, header->ts.tv_usec, header->len);
217
218     // pkt_data: 0x0040426c

```

08004500 就是找到的 IP 报文的特征

内存 1

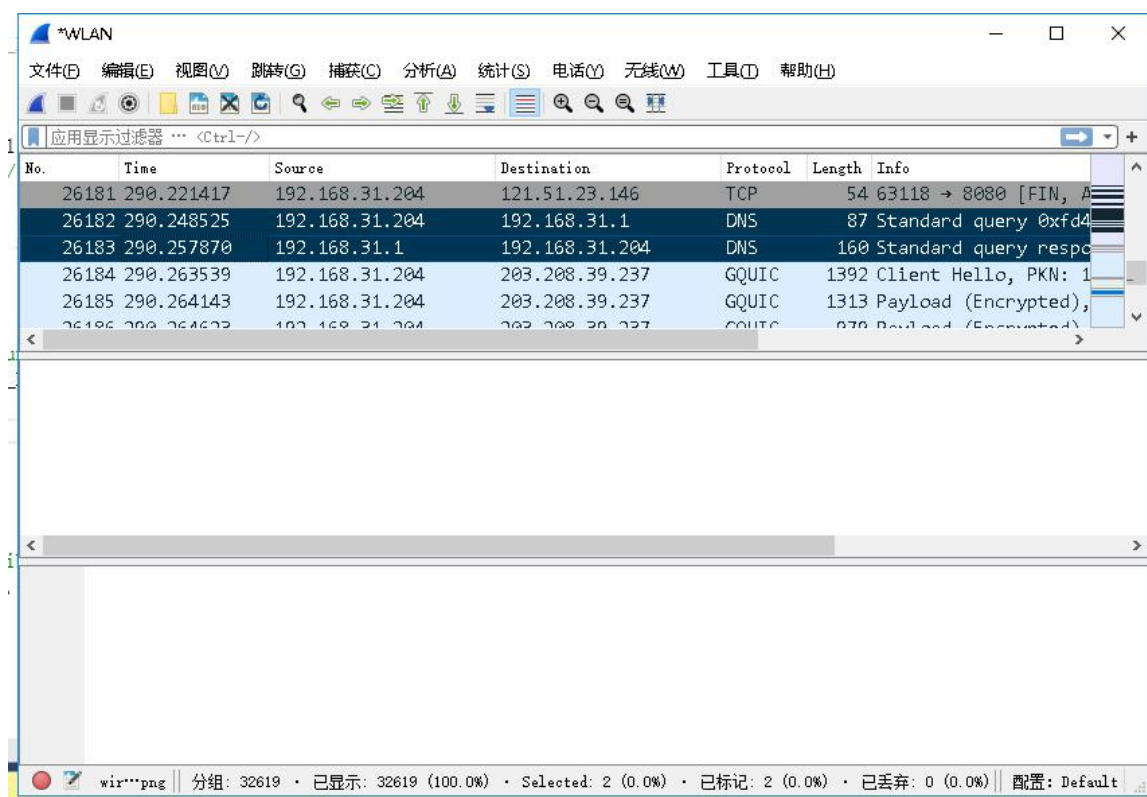
地址: 0x00D0426C

```

0x00D0426C  8c 53 c3 86 82 5c 68 07  ?S???h.
0x00D04274  15 e9 cd 57 08 00 45 00  .??W..E.
0x00D0427C  00 43 29 be 00 00 40 11  .C)?..@.
0x00D04284  4b 1e c0 a8 1f cc b6 fe  K.?? ???
0x00D0428C  6e 5b 0f aa 1f 40 00 2f  n[.?@./
0x00D04294  da df 02 38 59 00 58 54  ??8Y.XT

```

导出两则报文

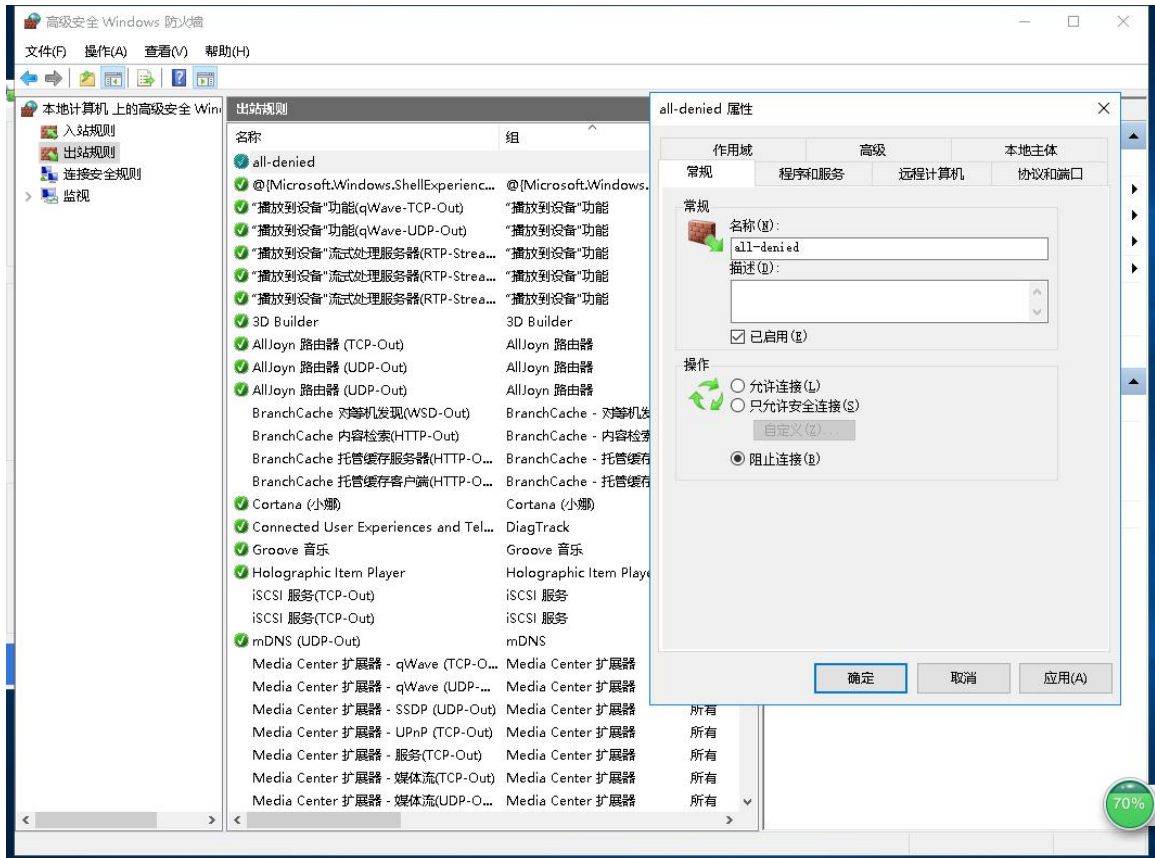


修改代码后得到的 IP 报文

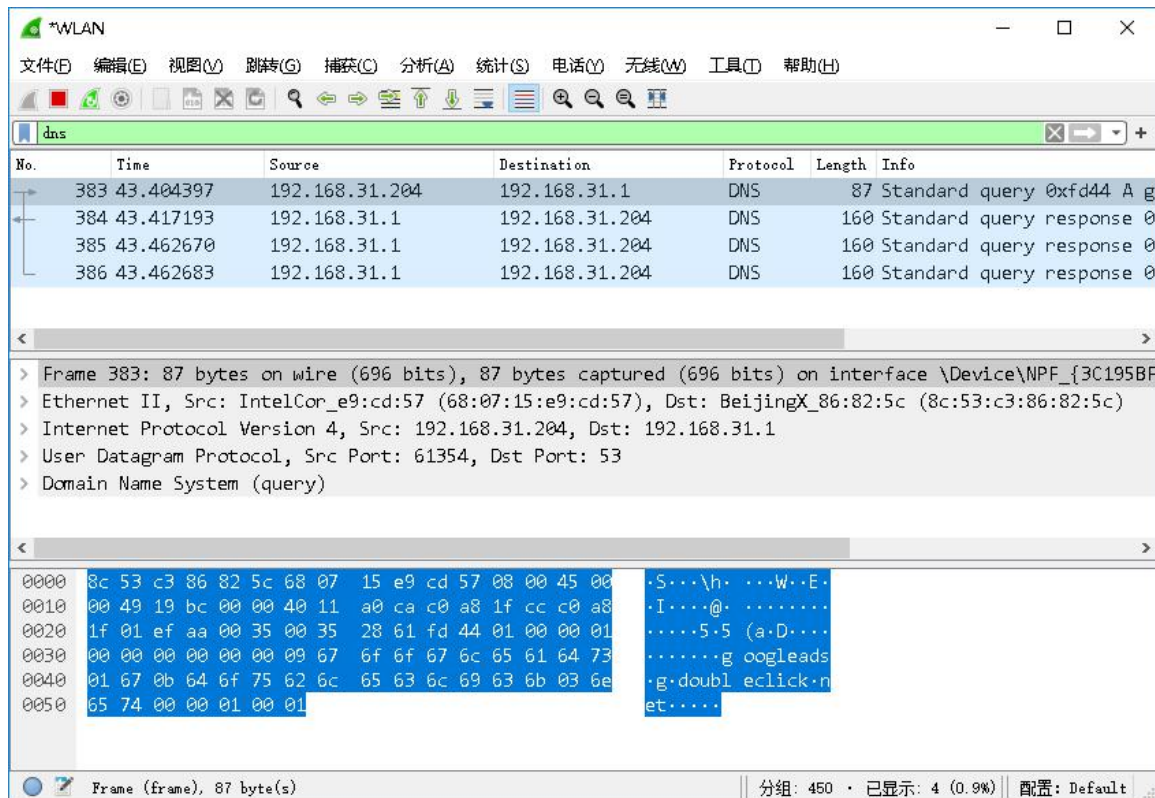


通过网卡模式验证报文

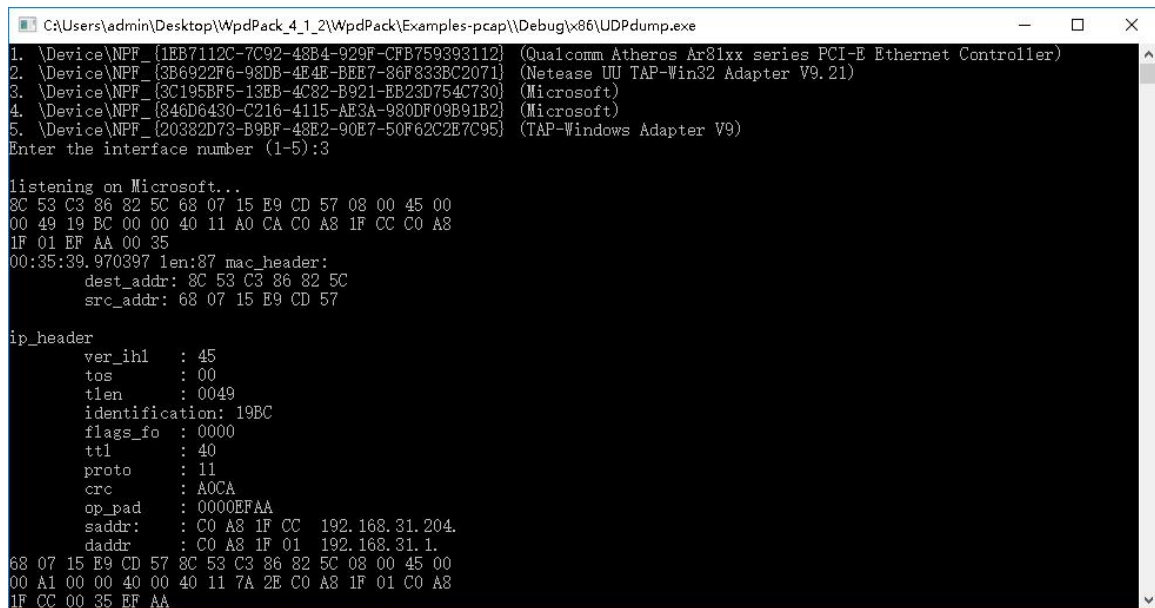
不禁用其他数据流会对程序造成干扰

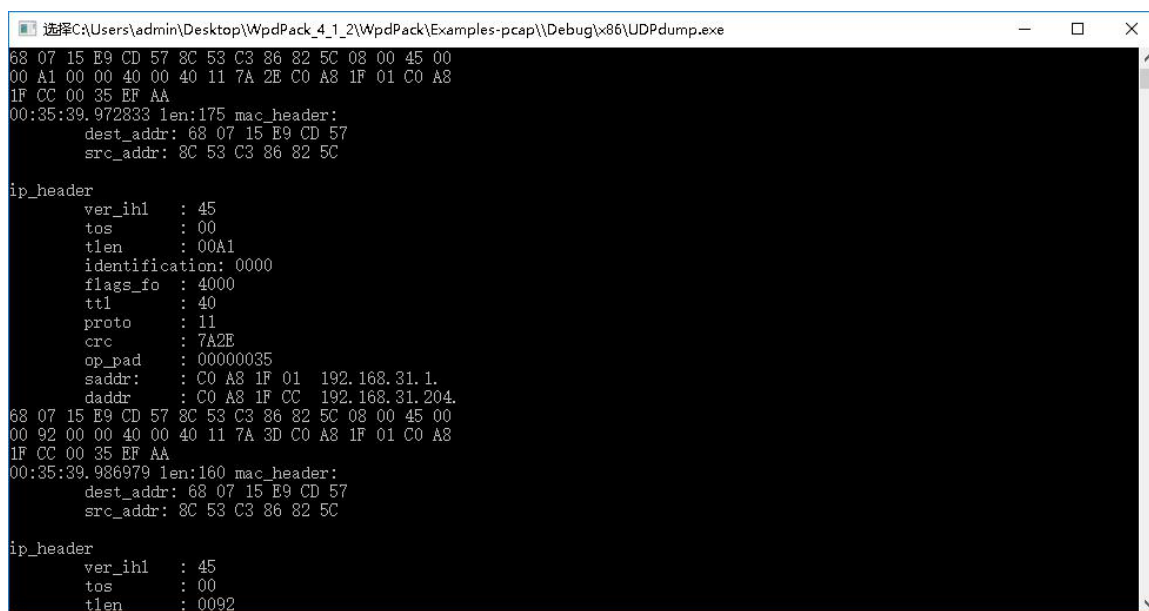


用数据包播放器发送一次得到的结果



侦听成功得到的结果





```
选择C:\Users\admin\Desktop\WpdPack_4_1_2\WpdPack\Examples-pcap\Debug\x86\UDPDump.exe
68 07 15 E9 CD 57 8C 53 C3 86 82 5C 08 00 45 00
00 A1 00 00 40 00 40 11 7A 2E C0 A8 1F 01 C0 A8
1F CC 00 35 EF AA
00:35:39.972833 len:175 mac_header:
    dest_addr: 68 07 15 E9 CD 57
    src_addr: 8C 53 C3 86 82 5C

ip_header
    ver_ihl : 45
    tos     : 00
    tlen    : 00A1
    identification: 0000
    flags_fo : 4000
    ttl     : 40
    proto   : 11
    crc     : 7A2E
    op_pad  : 00000035
    saddr:   : C0 A8 1F 01 192.168.31.1
    daddr:   : C0 A8 1F CC 192.168.31.204
68 07 15 E9 CD 57 8C 53 C3 86 82 5C 08 00 45 00
00 92 00 00 40 00 40 11 7A 3D C0 A8 1F 01 C0 A8
1F CC 00 35 EF AA
00:35:39.986979 len:160 mac_header:
    dest_addr: 68 07 15 E9 CD 57
    src_addr: 8C 53 C3 86 82 5C

ip_header
    ver_ihl : 45
    tos     : 00
    tlen    : 0092
```

4 实验总结

一般情况下只有网卡的 MAC 地址和帧头的目标地址一致主机才能接收数据帧。但是党主机处于监听模式下可以接受下流经网卡的所有数据帧，并对帧内数据内容进行分层剖析。