**江苏科技大学**

**实 验 报 告**

课 程： 入侵检测与入侵防御

课 题： 网络数据包获取

学 院： 计算机学院

学 号： 182210710119

姓 名： 陈四贵

班 级： 1822107101

指导老师： 张笑非

# 实验目的

1、掌握捕捉网络数据包的技巧，会捕捉网络数据包；

2、了解网络通信协议，会分析捕获的网络数据包；

3、明确基于网络的IDS需要分析的网络数据包的元素。

# 实验原理

1、Winpcap

Winpcap(windows packet capture)是windows平台下一个免费，公共的网络访问系统。它提供了一个强大的编程接口，它很容易地在各个操作系统之间进行移植，也很方便程序员进行开发。

Winpcap特别适用于下面这几个经典领域：网络及协议分析；网络监控；通信日志记录；traffic generators；用户级别的桥路和路由；网络入侵检测系统（NIDS）；网络扫描；安全工具。

其可用于：捕获原始数据包，包括在共享网络上各主机发送/接收的以及相互之间交换的数据包；在数据包发往应用程序之前，按照自定义的规则将某些特殊的数据包过滤掉；在网络上发送原始的数据包；收集网络通信过程中的统计信息。

2、共享网络环境下的数据捕获

共享网段的数据传输是通过广播实现的。在通常情况下，网络通信的应用程序只能响应与自己硬件地址相匹配的或是以广播形式发出的数据帧，对于其他形式的数据帧比如已到达网络接口但却不是发给此地址的数据帧，网络接口在验证投递地址并非自身地址之后将不再响应，也就是说应用程序无法收取与自己无关的数据包。

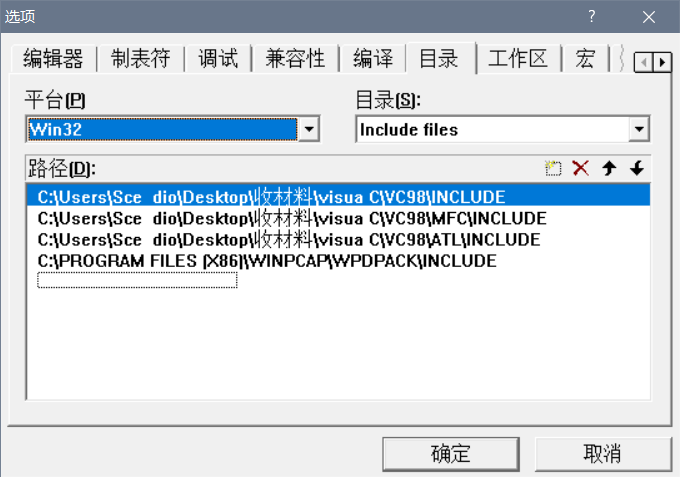
要想捕获流经网卡的但不属于自己主机的所有数据流，就必须绕开系统正常工作的处理机制，直接访问网络底层。我们可以通过设置网卡的工作模式为混杂模式，来使之接收目标地址不是自己的MAC地址的数据包，然后直接访问数据链路层，获取数据并由应用程序进行过滤处理。

# 实验环境

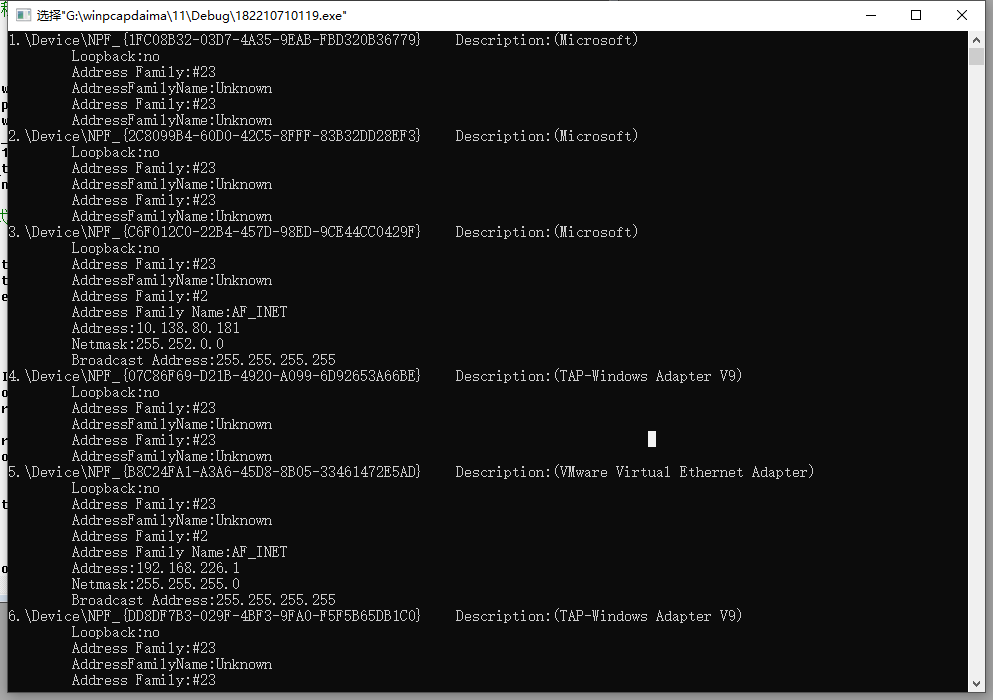
Windows 下PC机一台、Winpcap 4.1.3、Wpdpack 4.1.1

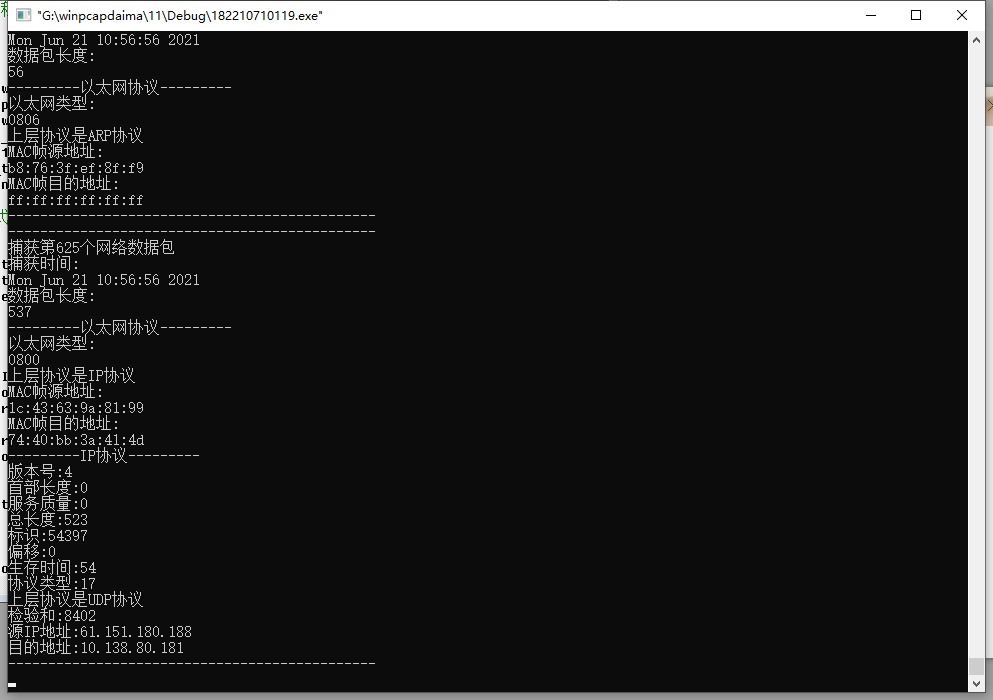
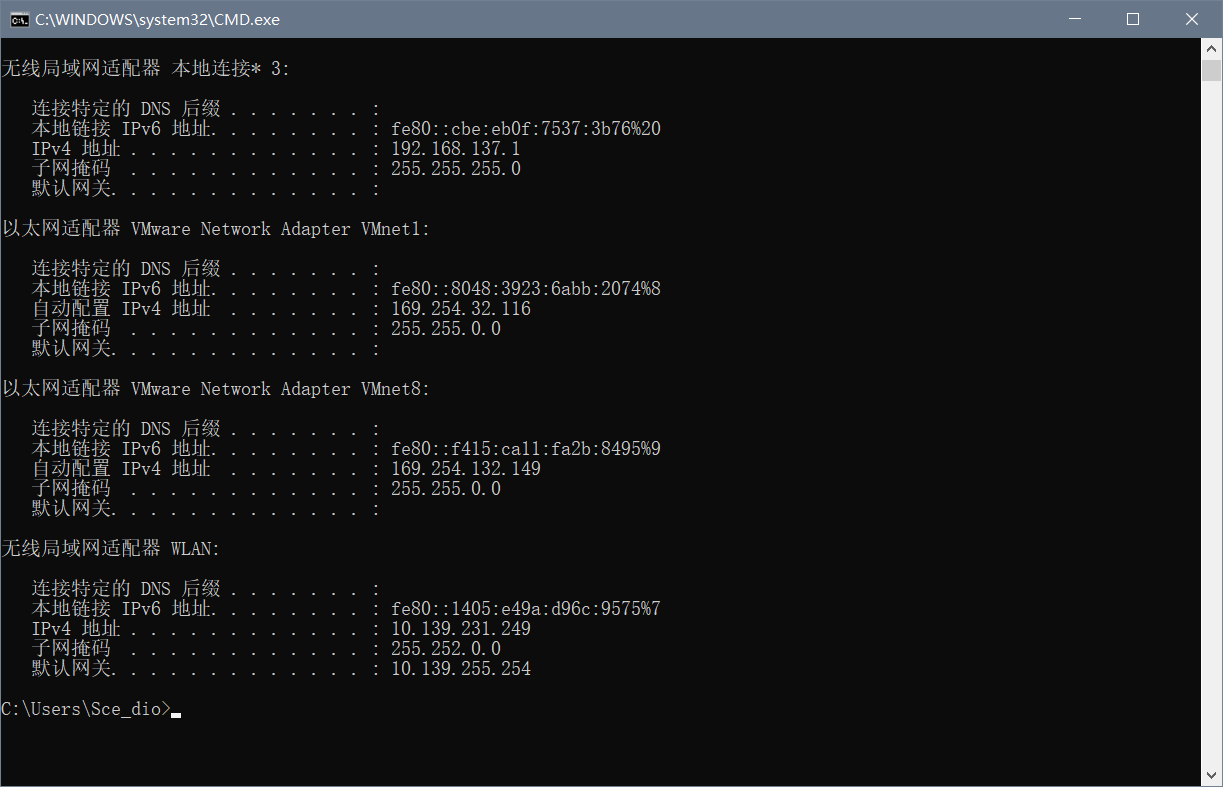
# 实验步骤与实验记录

1、下载安装winpcap 4.1.3、wpdpack 4.1.1

2、在visual C++中配置相关环境

3、运行程序，捕获网络数据包





# 实验分析

本次实验中，我们捕获了广播的数据包。我们通过ipconfig命令，得到了主机的IP地址为10.139.231.249，可以看到，捕获的数据包中，目的地址并非实验主机，而是其他机器。这可以证明我们捕获到了网络数据包。

# 心得与体会

我们可以把winpcap看做一个库而非一个软件，之后利用C++编写程序，调用该库的一些API，实现网络数据包获取的一些操作.

# 实现代码

1. //捕获网络数据包的C++程序
2. //可以获得数据包长度、通过以太网类型确定上层协议、源以太网地址和目的以太网地址！
3. #include "pcap.h"
4. #include<winsock2.h>
6. #pragma comment(lib,"wpcap.lib")
7. #pragma comment(lib,"packet.lib")
8. #pragma comment(lib,"ws2\_32.lib")
9. **void** packet\_handler(u\_char \*param,**const** **struct** pcap\_pkthdr \*header,**const** u\_char \*pcap\_data);
10. #define IPTOSBUFFERS 12
11. **void** ifprint(pcap\_if\_t \*d);
12. **char** \* iptos(u\_long in);
13. **int** i = 0;
14. /\*以下是以太网协议格式\*/
15. **struct** ether\_header
16. {
17. u\_int8\_t ether\_dhost[6]; //目的Mac地址
18. u\_int8\_t ether\_shost[6]; //源Mac地址
19. u\_int16\_t ether\_type;    //协议类型
20. };
22. **struct** ip\_header
23. {
24. #if defined(WORDS\_BIENDIAN)
25. u\_int8\_t   ip\_version:4,
26. ip\_header\_length:4;
27. #else
28. u\_int8\_t   ip\_header\_length:4,
29. ip\_version:4;
30. #endif
31. u\_int8\_t    ip\_tos;
32. u\_int16\_t   ip\_length;
33. u\_int16\_t   ip\_id;
34. u\_int16\_t   ip\_off;
35. u\_int8\_t    ip\_ttl;
36. u\_int8\_t    ip\_protocol;
37. u\_int16\_t   ip\_checksum;
38. **struct** in\_addr ip\_souce\_address;
39. **struct** in\_addr ip\_destination\_address;
40. };
42. **void** ip\_protool\_packet\_callback(u\_char \*argument,**const** **struct** pcap\_pkthdr\* packet\_header,**const** u\_char\* packet\_content)
43. {
44. **struct** ip\_header \*ip\_protocol;
45. u\_int header\_length = 0;
46. u\_int offset;
47. u\_char tos;
48. u\_int16\_t checksum;
49. //MAC首部是14位的，加上14位得到IP协议首部
50. ip\_protocol = (**struct** ip\_header \*) (packet\_content+14);
51. checksum =ntohs(ip\_protocol->ip\_checksum);
52. tos = ip\_protocol->ip\_tos;
53. offset = ntohs(ip\_protocol->ip\_off);
54. printf("---------IP协议---------\n");
55. printf("版本号:%d\n", ip\_protocol->ip\_version);
56. printf("首部长度:%d\n",header\_length);
57. printf("服务质量:%d\n",tos);
58. printf("总长度:%d\n",ntohs(ip\_protocol->ip\_length));
59. printf("标识:%d\n",ntohs(ip\_protocol->ip\_id));
60. printf("偏移:%d\n",(offset & 0x1fff) \* 8);
61. printf("生存时间:%d\n",ip\_protocol->ip\_ttl);
62. printf("协议类型:%d\n",ip\_protocol->ip\_protocol);
63. **switch** (ip\_protocol->ip\_protocol)
64. {
65. **case** 1: printf("上层协议是ICMP协议\n");**break**;
66. **case** 2: printf("上层协议是IGMP协议\n");**break**;
67. **case** 6: printf("上层协议是TCP协议\n");**break**;
68. **case** 17: printf("上层协议是UDP协议\n");**break**;
69. **default**:**break**;
70. }
71. printf("检验和:%d\n",checksum);
72. printf("源IP地址:%s\n", inet\_ntoa(ip\_protocol->ip\_souce\_address));
73. printf("目的地址:%s\n", inet\_ntoa(ip\_protocol->ip\_destination\_address));
74. }
76. **void** ethernet\_protocol\_packet\_callback(u\_char \*argument,**const** **struct** pcap\_pkthdr\* packet\_header,**const** u\_char\* packet\_content)
77. {
78. u\_short ethernet\_type;
79. **struct** ether\_header \*ethernet\_protocol;
80. u\_char \*mac\_string;
81. **static** **int** packet\_number = 1;
82. printf("----------------------------------------------\n");
83. printf("捕获第%d个网络数据包\n",packet\_number);
84. printf("捕获时间:\n");
85. printf("%s",ctime((**const** **time\_t**\*)&packet\_header->ts.tv\_sec));
86. printf("数据包长度:\n");
87. printf("%d\n",packet\_header->len);
88. printf("---------以太网协议---------\n");
89. ethernet\_protocol=(**struct** ether\_header\*)packet\_content;//获得数据包内容
90. printf("以太网类型:\n");
91. ethernet\_type=ntohs(ethernet\_protocol->ether\_type);//获得以太网类型
92. printf("%04x\n",ethernet\_type);
93. **switch** (ethernet\_type)
94. {
95. **case** 0x0800: printf("上层协议是IP协议\n");**break**;
96. **case** 0x0806: printf("上层协议是ARP协议\n");**break**;
97. **case** 0x8035: printf("上层协议是RARP协议\n");**break**;
98. **default**:**break**;
99. }
100. printf("MAC帧源地址:\n");
101. mac\_string=ethernet\_protocol->ether\_shost;
102. printf("%02x:%02x:%02x:%02x:%02x:%02x\n",\*mac\_string,\*(mac\_string+1),\*(mac\_string+2),\*(mac\_string+3),\*(mac\_string+4),\*(mac\_string+5));
103. printf("MAC帧目的地址:\n");
104. mac\_string=ethernet\_protocol->ether\_dhost;
105. printf("%02x:%02x:%02x:%02x:%02x:%02x\n",\*mac\_string,\*(mac\_string+1),\*(mac\_string+2),\*(mac\_string+3),\*(mac\_string+4),\*(mac\_string+5));
106. **if**(ethernet\_type==0x0800)//继续分析IP协议
107. {
108. ip\_protool\_packet\_callback (argument,packet\_header,packet\_content);
109. }
110. printf("----------------------------------------------\n");
111. packet\_number++;
112. }
114. **int** main()
115. // {
116. //      pcap\_t\* pcap\_handle; //winpcap句柄
117. //      char error\_content[PCAP\_ERRBUF\_SIZE]; //存储错误信息
118. //      bpf\_u\_int32 net\_mask; //掩码地址
119. //      bpf\_u\_int32 net\_ip;  //网络地址
120. //      char \*net\_interface;  //网络接口
121. //      struct bpf\_program bpf\_filter;  //BPF过滤规则
122. //      char bpf\_filter\_string[]="ip"; //过滤规则字符串，只分析IPv4的数据包
123. //      net\_interface=pcap\_lookupdev(error\_content); //获得网络接口
124. //      pcap\_lookupnet(net\_interface,&net\_ip,&net\_mask,error\_content); //获得网络地址和掩码地址
125. //      pcap\_handle=pcap\_open\_live(net\_interface,BUFSIZ,1,0,error\_content); //打开网络接口
126. //      pcap\_compile(pcap\_handle,&bpf\_filter,bpf\_filter\_string,0,net\_ip); //编译过滤规则
127. //      pcap\_setfilter(pcap\_handle,&bpf\_filter);//设置过滤规则
128. //      if (pcap\_datalink(pcap\_handle)!=DLT\_EN10MB) //DLT\_EN10MB表示以太网
129. //          return 0;
130. //      pcap\_loop(pcap\_handle,10,ethernet\_protocol\_packet\_callback,NULL); //捕获10个数据包进行分析
131. //      pcap\_close(pcap\_handle);
132. //      return 0;
133. // }
134. {
135. pcap\_if\_t \* alldevs;
136. pcap\_if\_t \* d;
137. **int** inum;
138. pcap\_t \* adhandle;
139. **char** errbuf[PCAP\_ERRBUF\_SIZE];
140. /\*取得列表\*/
141. **if**(pcap\_findalldevs(&alldevs,errbuf) == -1)
142. {
143. exit(1);
144. }
145. /\*输出列表\*/
146. **for**(d=alldevs;d != NULL;d=d->next)
147. {
148. ifprint(d);
149. }
150. **if**(i==0)
151. {
153. printf("\nNo interfaces found!Make sure WinPcap is installed.\n");
154. **char** c = getchar();
155. **return** -1;
156. }
157. printf("Enter the interface number (1-%d):",i);
158. scanf("%d",&inum);
159. **if**(inum <1 || inum >i)
160. {
161. printf("\nInterface number out of range.\n");
162. pcap\_freealldevs(alldevs);
163. **char** c = getchar();
164. **return** -1;
165. }
167. //转到选择的设备
168. **for** (d = alldevs,i=0;i <inum-1;d=d->next,i++);
169. //打开失败
170. **if** ((adhandle=pcap\_open\_live(d->name,65536,1,1000,errbuf))==NULL)
171. {
172. fprintf(stderr,"\nUnable to open the adapter.%s is not supported by WinPcap\n");
173. pcap\_freealldevs(alldevs);
174. **char** c = getchar();
175. **return** -1;
176. }
177. printf("\nlistening on %s...\n",d->description);
178. //释放列表
179. pcap\_freealldevs(alldevs);
180. //开始捕捉
181. //pcap\_loop(adhandle,0,ip\_protool\_packet\_callback,NULL);
182. pcap\_loop(adhandle,0,ethernet\_protocol\_packet\_callback,NULL);
183. **char** c = getchar();
184. **return** 0;
185. }
186. **void** ifprint(pcap\_if\_t \*d)
187. {
188. pcap\_addr\_t \*a;
189. printf("%d.%s",++i,d->name);
190. **if**(d->description)
191. {
192. printf("\tDescription:(%s)\n",d->description);
193. }**else**{
194. printf("\t(No description available)\n");
195. }
196. printf("\tLoopback:%s\n",(d->flags & PCAP\_IF\_LOOPBACK)?"yes":"no");
197. **for** (a=d->addresses;a != NULL;a=a->next)
198. {
199. printf("\tAddress Family:#%d\n",a->addr->sa\_family);
200. **switch** (a->addr->sa\_family)
201. {
202. **case** AF\_INET:
203. printf("\tAddress Family Name:AF\_INET\n");
204. **if**(a->addr)
205. {
206. printf("\tAddress:%s\n",iptos(((**struct** sockaddr\_in \*)a->addr)->sin\_addr.s\_addr));
207. }
208. **if**(a->netmask)
209. {
210. printf("\tNetmask:%s\n",iptos(((**struct** sockaddr\_in \*)a->netmask)->sin\_addr.s\_addr));
211. }
212. **if**(a->broadaddr)
213. {
214. printf("\tBroadcast Address:%s\n",iptos(((**struct** sockaddr\_in \*)a->broadaddr)->sin\_addr.s\_addr));
215. }
216. **if**(a->dstaddr)
217. {
218. printf("\tDestination Address:%s\n",iptos(((**struct** sockaddr\_in \*)a->dstaddr)->sin\_addr.s\_addr));
219. }
220. **break**;
221. **default**:
222. printf("\tAddressFamilyName:Unknown\n");
223. **break**;
224. }
225. }
226. }
227. **char** \* iptos(u\_long in)
228. {
229. **static** **char** output[IPTOSBUFFERS][3\*4+3+1];
230. **static** **short** which;
231. u\_char \*p;
232. p = (u\_char \*)∈
233. which=(which+1==IPTOSBUFFERS?0:which+1);
234. sprintf(output[which],"%d.%d.%d.%d",p[0],p[1],p[2],p[3]);
235. **return** output[which];
236. }
237. **void** packet\_handler(u\_char \*param,**const** **struct** pcap\_pkthdr \*header,**const** u\_char \*pcap\_data)
238. {
239. **struct** **tm** \* ltime;
240. **char** timestr[16];
241. ltime = localtime(&header->ts.tv\_sec);
242. strftime(timestr,**sizeof**(timestr),"%H:%M:%S",ltime);
243. printf("%s, %.6d len:%d\n",timestr,header->ts.tv\_usec,header->len);
244. }