

网络安全实验报告 - 捕包软件的使用与实现

- 实验题目：捕包软件的使用与实现
- 学号：1162100526
- 姓名：蔡晨馨

1. 实验目的

- 熟练使用sniff或wireshark软件。
- 深刻理解TCP三次握手的过程。
- 熟练使用libpcap或winpcap进行编程，能够完成对数据包的捕获分析。
- 深刻TCP/IP协议，并在捕包分析中熟练使用。

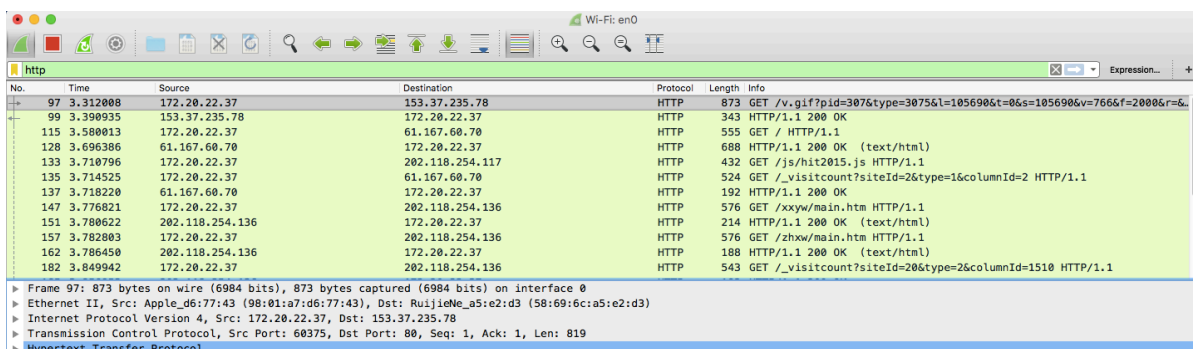
2. 实验要求及实验环境

- 实验要求：
 1. 熟练使用sniffer或wireshark软件，对协议进行还原。
能够访问网页的四元组。
 2. 利用libpcap或winpcap进行编程，能够对本机的数据包进行捕获分析。
将本机所有数据包的四元组写到指定文件。
 3. 按照自己的设想撰写需求分析和详细设计。
- 实验环境：
 1. Wireshark的使用：macOS High Sierra 10.13.3 | Wireshark 2.6.6
 2. Libpcap捕包：macOS High Sierra 10.13.3 | C++

3. 实验内容

1. Wireshark的使用

- 通过过滤仅查看与http协议有关的信息
- 登录[学校官网](#)，捕获到如下信息：



No.	Time	Source	Destination	Protocol	Length	Info
97	3.312008	172.20.22.37	153.37.235.78	HTTP	873	GET /v.gif?pid=307&type=307561=105690&t=0&s=105690&v=766&f=2000&r=6
99	3.390935	153.37.235.78	172.20.22.37	HTTP	343	HTTP/1.1 200 OK
115	3.580013	172.20.22.37	61.167.60.70	HTTP	555	GET / HTTP/1.1
128	3.696386	61.167.60.70	172.20.22.37	HTTP	688	HTTP/1.1 200 OK (text/html)
133	3.710796	172.20.22.37	202.118.254.117	HTTP	432	GET /js/hit2015.js HTTP/1.1
135	3.714525	172.20.22.37	61.167.60.70	HTTP	524	GET /_visitcount?siteId=2&type=1&columnId=2 HTTP/1.1
137	3.718220	61.167.60.70	172.20.22.37	HTTP	192	HTTP/1.1 200 OK
147	3.776821	172.20.22.37	202.118.254.136	HTTP	576	GET /xxw/main.htm HTTP/1.1
151	3.780622	202.118.254.136	172.20.22.37	HTTP	214	HTTP/1.1 200 OK (text/html)
157	3.782803	172.20.22.37	202.118.254.136	HTTP	576	GET /zhxw/main.htm HTTP/1.1
162	3.786450	202.118.254.136	172.20.22.37	HTTP	188	HTTP/1.1 200 OK (text/html)
182	3.849942	172.20.22.37	202.118.254.136	HTTP	543	GET /_visitcount?siteId=20&type=2&columnId=1510 HTTP/1.1

Frame 97: 873 bytes on wire (6984 bits), 873 bytes captured (6984 bits) on interface 0
Ethernet II, Src: Apple_d6:77:43 (98:01:a7:d6:77:43), Dst: RuijieNe_a5:e2:d3 (58:69:6c:a5:e2:d3)
Internet Protocol Version 4, Src: 172.20.22.37, Dst: 153.37.235.78
Transmission Control Protocol, Src Port: 60375, Dst Port: 80, Seq: 1, Ack: 1, Len: 819
Hypertext Transfer Protocol

- 从第一个GET报文选择Follow TCP Stream跟随，可以找到TCP三次握手的过程。

No.	Time	Source	Destination	Protocol	Length	Info
81	3.261201	172.20.22.37	153.37.235.78	TCP	78	60375 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=557063770 TSecr=0
95	3.311635	153.37.235.78	172.20.22.37	TCP	78	80 → 60375 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1452 SACK_PERM=1
96	3.311729	172.20.22.37	153.37.235.78	TCP	54	60375 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
97	3.312008	172.20.22.37	153.37.235.78	HTTP	873	GET /v.gif?pid=307&type=3075&l=105690&t=0&s=105690&v=766&f=2000&r=&u=http%

接下来分析三次握手的过程：

- 第一次握手发送SYN。Wireshark捕获解析主要如下 (未截取可选部分和timestamp，以下相同)：

<p>Frame 383: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0</p> <p>Ethernet II, Src: Apple_d6:77:43 (98:01:a7:d6:77:43), Dst: RuijieNe_a5:e2:d3 (58:69:6c:a5:e2:d3)</p> <p>Internet Protocol Version 6, Src: 2001:250:fe01:130:f8a6:fe95:f877:fd1d, Dst: 2001:da8:b800:253:dbd9:e20f</p> <p>Transmission Control Protocol, Src Port: 58918, Dst Port: 80, Seq: 0, Len: 0</p> <p>Source Port: 58918</p> <p>Destination Port: 80</p> <p>[Stream index: 6]</p> <p>[TCP Segment Len: 0]</p> <p>Sequence number: 0 (relative sequence number)</p> <p>[Next sequence number: 0 (relative sequence number)]</p> <p>Acknowledgment number: 0</p> <p>1011 = Header Length: 44 bytes (11)</p> <p>Flags: 0x0c2 (SYN, ECN, CWR)</p> <p>Window size value: 65535</p> <p>[Calculated window size: 65535]</p> <p>Checksum: 0x4e24 [unverified]</p> <p>[Checksum Status: Unverified]</p> <p>Urgent pointer: 0</p> <p>Options: (24 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), Timestamps, SACK permitted, End of Option List (EOL)</p> <p>[Timestamps]</p>
--

- 可以很容易看到Src: 172.20.22.37即源IP，Src Port: 60375即源端口，这个实例中表示本机IP和使用的端口；Dst: 153.37.235.78即目的IP，Dst Port: 80，表示服务器IP和端口，HTTP请求一般使用80端口。
- 该报文的Sequence Number(序列号)为0，Acknowledgment Number(确认号)为0。表示发送了序列号为0的数据包，并且在等待接收序列号为0的数据包。

- 第二次握手服务器端返回SYN，ACK，捕获信息如下：

<p>Frame 95: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0</p> <p>Ethernet II, Src: RuijieNe_a5:e2:d3 (58:69:6c:a5:e2:d3), Dst: Apple_d6:77:43 (98:01:a7:d6:77:43)</p> <p>Internet Protocol Version 4, Src: 153.37.235.78, Dst: 172.20.22.37</p> <p>Transmission Control Protocol, Src Port: 80, Dst Port: 60375, Seq: 0, Ack: 1, Len: 0</p> <p>Source Port: 80</p> <p>Destination Port: 60375</p> <p>[Stream index: 5]</p> <p>[TCP Segment Len: 0]</p> <p>Sequence number: 0 (relative sequence number)</p> <p>[Next sequence number: 0 (relative sequence number)]</p> <p>Acknowledgment number: 1 (relative ack number)</p> <p>1011 = Header Length: 44 bytes (11)</p> <p>Flags: 0x012 (SYN, ACK)</p> <p>Window size value: 8192</p> <p>[Calculated window size: 8192]</p> <p>Checksum: 0x2d6a [unverified]</p> <p>[Checksum Status: Unverified]</p> <p>Urgent pointer: 0</p>
--

- 同理可以得到源IP: 153.37.235.78，目的IP: 172.20.22.37，源端口80，目的端口60375。与第一次握手正好相反。
- 该报文的序列号为0，确认号为1。表示收到了客户端发送的序列号为0的数据包，正在等待序列号为1的数据包，并且发送的数据包序列号为0。

2. 第三次握手客户端(本机)返回ACK, 捕获信息如下:

```
▶ Frame 96: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
▶ Ethernet II, Src: Apple_d6:77:43 (98:01:a7:d6:77:43), Dst: RuijieNe_a5:e2:d3 (58:69:6c:a5:e2:d3)
▶ Internet Protocol Version 4, Src: 172.20.22.37, Dst: 153.37.235.78
▼ Transmission Control Protocol, Src Port: 60375, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
  Source Port: 60375
  Destination Port: 80
  [Stream index: 5]
  [TCP Segment Len: 0]
  Sequence number: 1 (relative sequence number)
  [Next sequence number: 1 (relative sequence number)]
  Acknowledgment number: 1 (relative ack number)
  0101 .... = Header Length: 20 bytes (5)
▶ Flags: 0x010 (ACK)
  Window size value: 65535
  [Calculated window size: 65535]
  [Window size scaling factor: -2 (no window scaling used)]
  Checksum: 0xc13d [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
▶ [SEQ/ACK analysis]
▶ [Timestamps]
```

- 源IP: 172.20.22.37, 目的IP: 153.37.235.78, 源端口60375, 目的端口80。

与第一次握手正好相同。

- 该报文的序列号为1, 确认号为1。

表示收到了服务器序列号为0的数据包, 正在等待序列号为1的数据包, 并且发送了序列号为1的数据包。

三次握手完成, 建立连接, 开始发送数据, 首先便是本机发送GET请求, 捕获如下:

```
▶ Frame 97: 873 bytes on wire (6984 bits), 873 bytes captured (6984 bits) on interface 0
▶ Ethernet II, Src: Apple_d6:77:43 (98:01:a7:d6:77:43), Dst: RuijieNe_a5:e2:d3 (58:69:6c:a5:e2:d3)
▶ Internet Protocol Version 4, Src: 172.20.22.37, Dst: 153.37.235.78
▶ Transmission Control Protocol, Src Port: 60375, Dst Port: 80, Seq: 1, Ack: 1, Len: 819
▶ Hypertext Transfer Protocol
```

可以看到与三次握手的TCP不同, HTTP有5层。

这是因为HTTP位于应用层, 其经过了物理层、数据链路层(以太网层)、网络层、传输层以及应用层; 而TCP位于传输层, 只经过了物理层、数据链路层、网络层和传输层。

2. Libpcap捕包

- Libpcap 运行在类UNIX系统下的网络数据包捕获函数库, 捕获网卡上的数据。
- 实验中使用到的Libpcap库函数如下:

```
/*寻找可捕获的设备*/
char *pcap_lookupdev(char *errbuf);
/*获取指定网络设备的网络号和掩码*/
int pcap_lookupnet(char *device, bpf_u_int32 *netp, bpf_u_int32 *maskp,
char *errbuf);
/*打开捕获设备, promisc参数为true时进入混杂模式, 最后返回会话处理程序*/
pcap_t *pcap_open_live(char *device, int snaplen, int promisc, int to_ms,
char *ebuf);
/*用于确保捕获的设备在以太网上*/
int pcap_datalink(pcap_t *p);
/*将表达式编译到过滤程序中, 并应用该过滤器*/
```

```

int pcap_compile(pcap_t *p, struct bpf_program *fp, char *str, int
optimize, bpf_u_int32 netmask);
int pcap_setfilter(pcap_t *p, struct bpf_program *fp);
/*对callback函数进行循环,
*其中cnt告诉pcap_loop在返回之前应该嗅探多少个数据包, 负值表示循环直到发生错误
*user为传进callback函数中的参数*/
int pcap_loop(pcap_t * p, int cnt, pcap_handler callback, u_char * user);
/*回调函数, 函数名可以自定义
*user对应pcap_loop中最后一个参数, 是用户传入的
*packet指向包含整个数据包的数据部分的第一个字节*/
void callback(u_char * user, const struct pcap_pkthdr * header, const
u_char * packet);

```

- 要对网络数据包进行捕获分析, 需要定义协议的结构体, 包括以太网头部、IP头部和TCP头部(或其他网络层头部)。
 - 以太网头部包括8位目标地址, 8位源地址, 16位以太网类型。
 - IP头部包括4位版本号, 4位首部长度, 8位服务类型, 16位数据包总长度, 16位ID, 3位Flags, 13位偏移字段, 8位生存时间, 8位协议类型, 16位首部校验和, 32位源IP和目的IP。
 - TCP头部包括16位源端口号, 16位目的端口号, 32位序号, 32位确认号, 4位首部长度, 6位保留未用, 6位标志字段, 16位接受窗口, 16位校验和, 16位紧急数据指针。

构造结构体之后, 便可以计算其头部偏移地址, 根据结构体内容直接获取需要的信息, 本次实验中仅获取源IP, 目的IP, 源端口和目的端口。

- 本次实验参考了[TCPdump](#)官网关于pcap编程的教程。
- 主要编写了callback函数, 计算IP Header位置, TCP Header位置后, 获取所需信息, 打印并存入文件中。

其中文件名为pcap_loop()传入的user指针, 此处调用即可。

```
FILE* fp = fopen((char*)user, "a");
```

- 此外还定义了命令行参数, 用户可自定义一下参数:
 - -f: 写入文件名, 默认为sniffer.log。
 - -e: 过滤表达式, 默认为tcp (其他格式数据包没有进行处理, 如果仅需获取源端口和目的端口, UDP与TCP结构相同)。
 - -n: 捕获的数据包数量, 默认值为-1, 即循环捕包直到发生错误。

4. 实验结果

Wireshark的使用实验结果以在实验内容中有所呈现。

其中包括捕获HTTP包, 根据序列号和确认号分析TCP的三次握手, 以及分析HTTP数据包与TCP数据包的不同。

Libpcap捕包

- 编译运行代码，给定参数，日志输出文件名为grap.log，捕包数量为100。

```
→ code git:(master) x gcc sniff.c -lpcap -o sniff
→ code git:(master) x ./sniff -f grap.log -n 100
Filename: grap.log
Packets_num: 100

Device: en0
172.20.0.0:255.255.128.0
10Mb以太网

Packet number 1:
  From: 121.51.36.139
  To: 172.20.22.37
Protocol: TCP
Src Port: 8080
Dst Port: 60548
Payload (183 bytes):
```

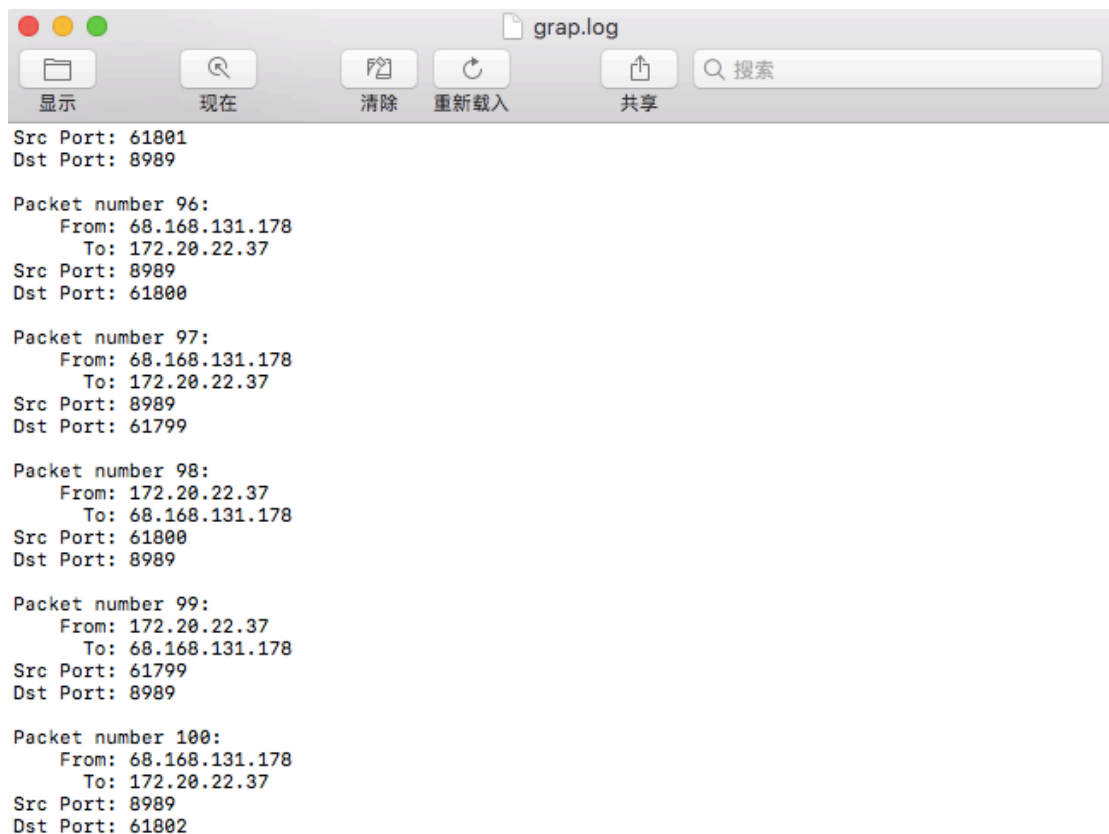
- 捕获100个数据包后，程序正常退出。

```
Packet number 99:
  From: 172.20.22.37
  To: 68.168.131.178
Protocol: TCP
Src Port: 61799
Dst Port: 8989

Packet number 100:
  From: 68.168.131.178
  To: 172.20.22.37
Protocol: TCP
Src Port: 8989
Dst Port: 61802

Capture complete.
```

- 查看输出日志grap.log



```
Src Port: 61801
Dst Port: 8989

Packet number 96:
  From: 68.168.131.178
  To: 172.20.22.37
Src Port: 8989
Dst Port: 61800

Packet number 97:
  From: 68.168.131.178
  To: 172.20.22.37
Src Port: 8989
Dst Port: 61799

Packet number 98:
  From: 172.20.22.37
  To: 68.168.131.178
Src Port: 61800
Dst Port: 8989

Packet number 99:
  From: 172.20.22.37
  To: 68.168.131.178
Src Port: 61799
Dst Port: 8989

Packet number 100:
  From: 68.168.131.178
  To: 172.20.22.37
Src Port: 8989
Dst Port: 61802
```