南开大学

Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

计算机网络实验报告

---

wireshark TCP 通信过程分析

于文明

年级：2020 级

专业：信息安全

指导教师：张建忠，徐敬东

2022 年 10 月 29 日

摘要

关键字：socket,TCP,Protocol Design

目录

一、 实验要求

（1）搭建 Web 服务器（自由选择系统），并制作简单的 Web 页面，包含简单文本信息（至少包含专业、学号、姓名）和自己的 LOGO。

（2）通过浏览器获取自己编写的 Web 页面，使用 Wireshark 捕获浏览器与 Web 服务器的交互过程，并进行简单的分析说明。

（3）提交实验报告。

二、 Web 服务器端搭建

Web 服务器端代码采用了 html 语言进行编写，根据实验要求包含了基本的文本信息和个人 LOGO，服务器基于 apache 进行搭建，实现了局域网访问配置，经过实验，可以通过 ip 进行局域网访问。并在后续拓展学习中实现了基于 SSL 协议的 https 升级，并使用 wireshark 对数据包进行了分析.

(一) 网页 php 代码

网页组成比较简单，主要包含了学号、姓名、专业，还有一张图片作为自己的 LOGO

网页 php 代码

```
<html>
<body>
<center><h3>NETWORK LAB2</h3></center>
<center>
姓名：于文明</br>
学号：2011762</br>
专业:信息安全</br>
</center>
<center><img src="./a1.jpg" /></center>
</body>
</html>
```

(二) 网页效果

网页效果如下

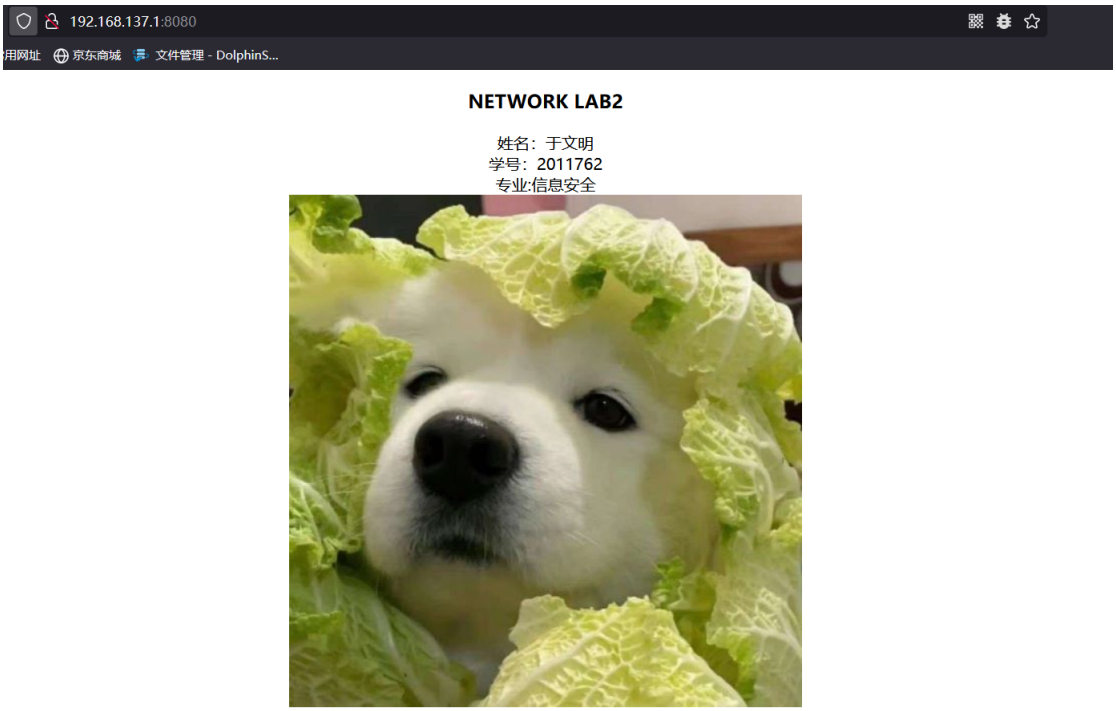

图 1: 网页效果图

三、 交互过程

我们使用 wireshark 对网页访问的交互过程进行数据抓包，然后通过对抓到的数据包分析来解释 TCP、HTTP 协议的通信过程

(一) wireshark 抓包

wireshark 是网络流量的专用抓包工具，因为我们的客户端和服务器都是在本地搭建，所以需要使用 wireshark 对本地流量进行抓包。打开 wireshark，因为是对本地流量进行抓包，选择 Adapter for loopback traffic capture, 开始抓包 (start capture)。之后通过本地域名和端口访问网页。

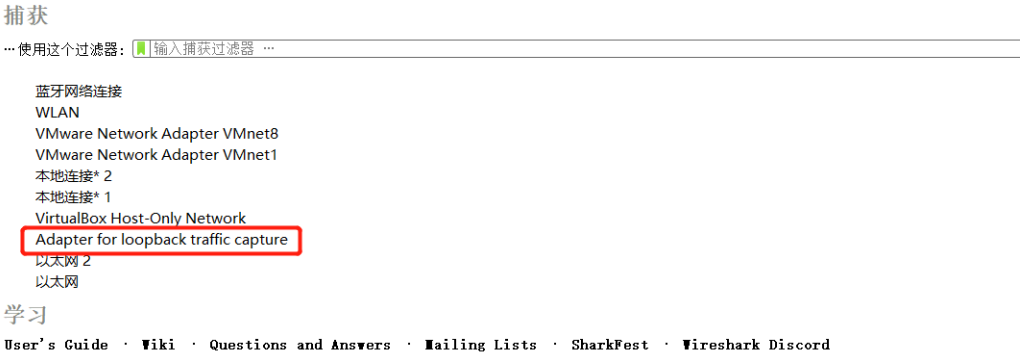

图 2: wiershark 界面

在 wireshark 抓到数据包后，我们需要设置过滤器对冗余的数据包进行过滤，这里我们通过

设置端口条件 (tcp.port=8080) 进行过滤。

tcp.port==8080

No.	Time	Source	Destination	Protocol	Length	Info
21	1.210871	192.168.137.1	192.168.137.1	TCP	56	55227 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_P
22	1.210952	192.168.137.1	192.168.137.1	TCP	56	8080 → 55227 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS
23	1.210969	192.168.137.1	192.168.137.1	TCP	44	55227 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
24	1.211140	192.168.137.1	192.168.137.1	HTTP	441	GET / HTTP/1.1
25	1.211148	192.168.137.1	192.168.137.1	TCP	44	8080 → 55227 [ACK] Seq=1 Ack=398 Win=2619648 Len=0
30	1.212524	192.168.137.1	192.168.137.1	HTTP	551	HTTP/1.1 200 OK (text/html)
31	1.212540	192.168.137.1	192.168.137.1	TCP	44	55227 → 8080 [ACK] Seq=398 Ack=508 Win=2619136 Len=0
76	1.247240	192.168.137.1	192.168.137.1	HTTP	394	GET /a1.jpg HTTP/1.1
77	1.247252	192.168.137.1	192.168.137.1	TCP	44	8080 → 55227 [ACK] Seq=508 Ack=748 Win=2619392 Len=0
78	1.247691	192.168.137.1	192.168.137.1	HTTP	37495	HTTP/1.1 200 OK (JPEG JFIF image)
79	1.247735	192.168.137.1	192.168.137.1	TCP	44	55227 → 8080 [ACK] Seq=748 Ack=37959 Win=2581760 Len=0
112	1.253007	192.168.137.1	192.168.137.1	HTTP	399	GET /favicon.ico HTTP/1.1
113	1.253023	192.168.137.1	192.168.137.1	TCP	44	8080 → 55227 [ACK] Seq=37959 Ack=1103 Win=2618880 Len=0
122	1.253881	192.168.137.1	192.168.137.1	HTTP	639	HTTP/1.1 404 Not Found (text/html)
123	1.253898	192.168.137.1	192.168.137.1	TCP	44	55227 → 8080 [ACK] Seq=1103 Ack=38554 Win=2580992 Len=0
172	6.268873	192.168.137.1	192.168.137.1	TCP	44	8080 → 55227 [FIN, ACK] Seq=38554 Ack=1103 Win=2618880 Len=0
173	6.268941	192.168.137.1	192.168.137.1	TCP	44	55227 → 8080 [ACK] Seq=1103 Ack=38555 Win=2580992 Len=0
174	6.269152	192.168.137.1	192.168.137.1	TCP	44	55227 → 8080 [FIN, ACK] Seq=1103 Ack=38555 Win=2580992 Len=0

图 3: wireshark 过滤数据包

(二) 交互过程分析

我们知道，HTTP1.1 访问网页的过程大致要经历三个过程:

- 三次握手, 建立连接 (TCP 协议)
- 请求响应 (HTTP)
- 四次挥手，断开连接 (TCP)

1. HTTP 相关数据包

http 遵循请求应答机制，客户端向 Web 服务器发送 HTTP 请求，Web 服务器处理请求并返回适当的应答，并向客户端发送数据，数据按照 HTTP 报文格式进行传递。

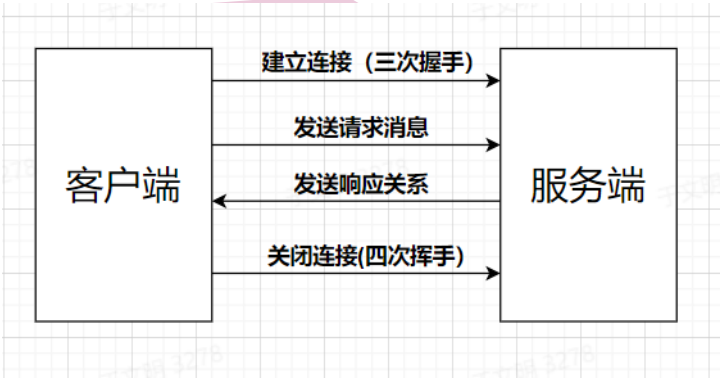

图 4: 交互过程图

对图中过程分析，首先，客户端通过 TCP 三次握手建立连接；建立连接成功之后，向服务器发送 HTTP 请求 (实验中使用 GET 的方式请求); 服务端在接收到请求之后，处理请求，返回适当的应答 (TCP), 并向客户端发送数据 (HTTP)。之后客户端通过四次挥手断开连接。

HTTP 由请求和响应组成，所以也对应两种报文格式, 下面将分别做简单介绍

HTTP 请求报文:

图 5: 请求报文格式

请求报文由以下部分组成:

- 请求行：由 3 部分组成，分别为：请求方法、URL（见备注 1）以及协议版本，之间由空格分隔，请求方法包括 GET、POST 等。协议版本的格式为：HTTP/主版本号. 次版本号，比如本实验中使用的就是 HTTP1.1 版本
- 请求头部包含很多客户端环境以及请求正文的有用信息。请求头部由“关键字：值”对组成，每行一堆，关键字和值之间使用英文“:”分隔。
- 空行，这一行非常重要，必不可少。表示请求头部结束，下面就是请求正文。
- 请求正文：可选部分，比如 GET 请求就没有请求正文；POST 比如以提交表单数据方式为请求正文。

下面我们结合具体的实验数据包分析

```
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /
    Request Version: HTTP/1.1
  Host: 192.168.137.1:8080\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101 Firefox/
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*
  Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  \r\n
```

图 6: GET 请求报文

可以看出，客户端使用 HTTP1.1 版本, 向服务器发送 GET 请求，由于访问的是部署在本地的服务器,Host 为主机 ip，端口号为设置好的 8080, 其他的信息还包括了用户代理 (User-Agent), 接收的消息格式 (Accept), 接收语言 (Accept-Language), 接收的编码方式 (Accept-Encoding), 连接方式 (Connection)

需要注意的是，在客户端发送请求后，服务端会向客户端发送一个确认收到请求 (TCP), 做出应答, 其中 ack 会设置为 1，序列号设置为 1，表示确认收到请求

```
Transmission Control Protocol, Src Port: 8080, Dst Port: 55227, Seq: 1, Ack: 398, Len: 0
    Source Port: 8080
    Destination Port: 55227
    [Stream index: 1]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 3235306934
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 398    (relative ack number)
    Acknowledgment number (raw): 1116255026
    0101 .... = Header Length: 20 bytes (5)
  Flags: 0x010 (ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Accurate ECN: Not set
      .... 0... .... = Congestion Window Reduced: Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·····A····]
```

图 7: 请求应答

HTTP 响应报文

图 8: 响应报文格式

响应报文由以下部分组成:

- 状态行由 3 部分组成，分别为：协议版本，状态码，状态码描述，之间由空格分隔。状态代码为 3 位数字，200 299 的状态码表示成功，300 399 的状态码指资源重定向，400 499 的状态码指客户端请求出错，500 599 的状态码指服务端出错
- HTTP 的响应头和请求头组成类似
- 空行，这一行非常重要，必不可少。表示响应头部结束
- 响应正文，服务器返回的文档，最常见的为 HTML 网页。

同样，我们将结合具体数据包进行分析

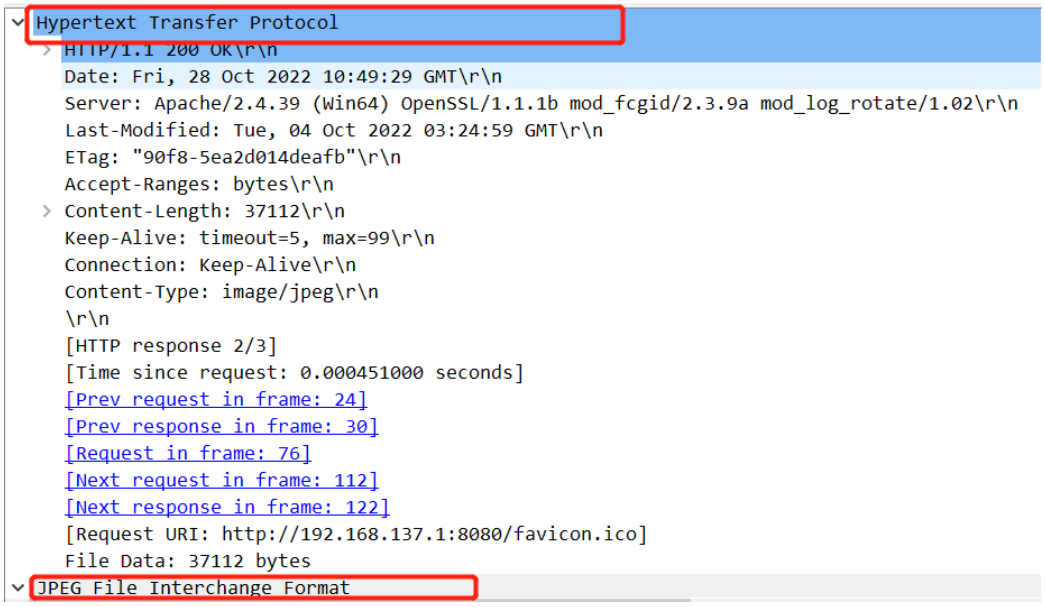

图 9: 响应报文格式

可以看到，服务器用 HTTP1.1 200 OK 响应了客户端的请求，因为发送数据是图片，还多了一个 JPEG File interchage format 段

2. TCP 相关数据包

TCP 是一种面向连接的、可靠的、基于字节流的传输层通信协议,HTTP 的连接和断开都基于 TCP 完成，具体来说，建立连接需要三次握手，断开连接需要四次挥手

1. 三次握手

TCP 的三次握手过程具体如下

- 第一次握手，客户端发送连接请求包到服务器，标志位 SYN(同步序号) 置为 1，序列号为 0，等待服务器确认
- 第二次握手，服务器收到客户端发过来的报文，由 SYN=1 知道客户端要求建立连接。向客户端发送一个 SYN 为 1 的 TCP 报文，设置初始序列号为 0，将 Acknowledgement Number 设置为客户端的序列号加 1
- 第三次握手，客户端收到服务端发来的包后检查确认序号是否正确，以及 ACK 是否为 1. 如果正确，客户端再次发送确认包，ACK 置为 1，SYN 置为 0.Acknowledge Number 设为服务端的序列号加 1，发送的序列号加 1。然后服务器端在收到后检查 Acknowledge Number 和 Ack=1 则建立连接成功

在握手过程中传送的 TCP 包中不包含数据，三次握手完毕后，客户端与服务端才开始正式传送数据。在理想状态下，TCP 连接一旦建立，在通信双方中的任何一方关闭连接之前，TCP 连接状态将被一直保留下去.

同样，我们还是结合具体数据包进行分析

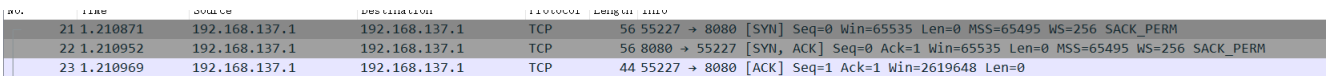

图 10: 三次握手

首先是客户端 (端口 55227) 向服务端 (端口 8080) 发送了一条 syn 包，并林 seq=x=0, 报文长度为 0，滑动窗口为 65535，最大窗口长度为 65495，窗口扩大影子为 256, 第一次握手完成; 然后服务端 8080 端口在收到浏览器发送的 TCP 包后，确认客户的 syn，使 ack=x+1=1, seq=y=0，对序列号为 1 之前的报文进行确认，同时向客户发送一个 SYN 为 1 的 TCP 包，第二次握手完成；客户端在接收到服务端发来的包后进行检查，检查无误，向服务端发送确认包，其中 ack=y+1=1，第三次握手完成，至此三次握手完成，成功建立了客户端和服务端的连接.

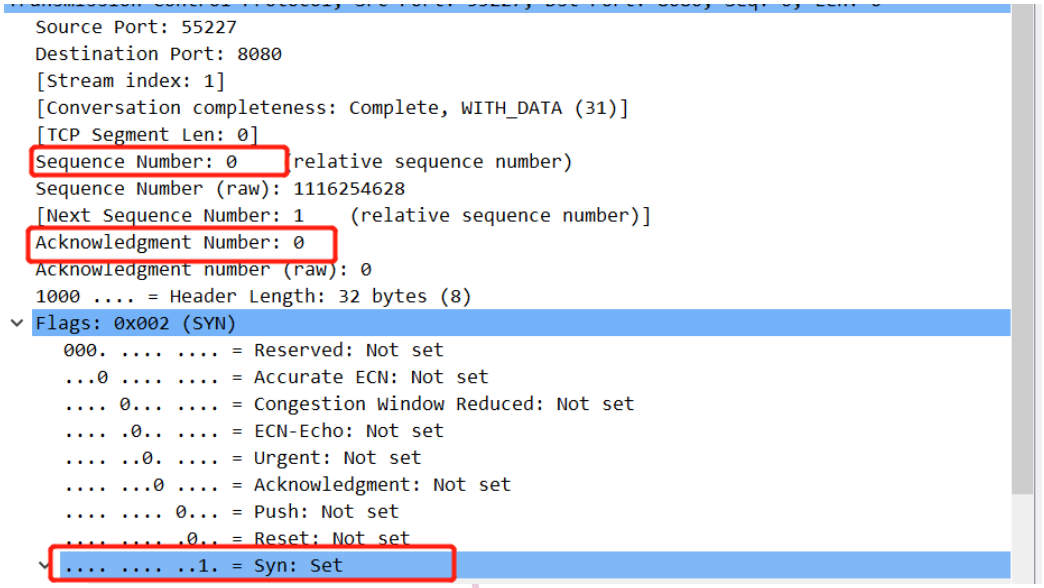

图 11: 第一次握手

2. 四次挥手

TCP 断开连接时，由四次挥手过程，一般由客户端先发起断连请求，但是由于我们的客户端和服务端都部署在本地，因此由服务端先发起请求. 四次挥手示意图如下

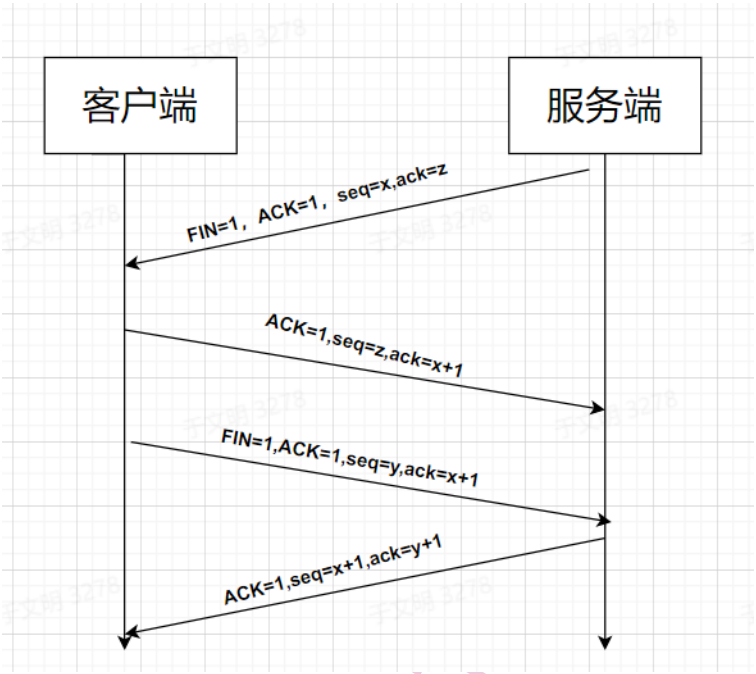

图 12: 四次挥手示意图

TCP 四次挥手的过程如下:

- 第一次挥手:Server 发送一个 FIN，用来关闭 Cilent 到 Server 的连接
- 第二次挥手:Cilent 在收到 FIN 后，发送一个 ACK 给 Cilent，确认序号为收到序列号 +1
- 第三次挥手：Cilent 发送一个 FIN，用来确认关闭数据连接
- 第四次挥手：Server 在收到 FIN 后，发送一个 ACK 给 Cilent，确认序号为收到的序列号 +1，Cilent 接收到后关闭连接, 四次挥手结束

172	6.268873	192.168.137.1	192.168.137.1	TCP	44 8080 → 55227 [FIN, ACK] Seq=38554 Ack=1103 Win=2618880 Len=0
173	6.268941	192.168.137.1	192.168.137.1	TCP	44 55227 → 8080 [ACK] Seq=1103 Ack=38555 Win=2580992 Len=0
174	6.269152	192.168.137.1	192.168.137.1	TCP	44 55227 → 8080 [FIN, ACK] Seq=1103 Ack=38555 Win=2580992 Len=0
175	6.269225	192.168.137.1	192.168.137.1	TCP	44 8080 → 55227 [ACK] Seq=38555 Ack=1104 Win=2618880 Len=0

图 13: 四次挥手数据抓包

第一次挥手，服务端给客户端发送 TCP 包，用来关闭客户端到服务器的数据传送。将标志位 FIN 和 ACK 置为 1，序号为 X=1，确认序号为 Z=1。第二次挥手, 客户端收到 FIN 后，发回一个 ACK(标志位 ACK=1), 确认序号为收到的序号加 1，即 X=X+1=2。序号为收到的确认序号 =Z。第三次挥手，客户端关闭与服务端的连接，发送一个 FIN。标志位 FIN 和 ACK 置为 1，序号为 Y=1，确认序号为 X=2。第四次挥手，服务端收到客户端发送的 FIN 之后，发回 ACK 确认 (标志位 ACK=1), 确认序号为收到的序号加 1，即 Y+1=2。序号为收到的确认序号 X=2。

四、 拓展学习 (https 抓包分析)

(一) 配置 https

HTTPS 有两部分组成：HTTP + SSL / TLS，也就是在 HTTP 上又加了一层处理加密信息的模块。服务端和客户端的信息传输都会通过 TLS 进行加密，所以传输的数据都是加密后的

数据。首先通过命令生成证书文件

网页 php 代码

```
openssl genrsa -des3 -out root.key
openssl req -new -key root.key -out root.csr
```

之后将证书内容输入 phpstudy_pro 的 https 的相关部分，工具就会自动帮助我们完成配置

图 14: 配置 https

(二) 数据抓包分析

抓到的包分为两个部分，一部分是正常的 TCP 协议 (包括三次握手等)，之后是 SSL 的握手阶段，具体过程如下

- 初始化阶段。客户端创建随机数，发送 ClientHello 将随机数连同自己支持的协议版本、加密算法和压缩算法发送给服务器。服务器回复 ServerHello 将自己生成的随机数连同选择的协议版本、加密算法和压缩算法给客户端。
- 认证阶段。服务器发送 ServerHello 的同时可能将包含自己公钥的证书发送给客户端，并请求客户端的证书
- 密钥协商阶段。客户端验证证书，如果收到 Certificate Request 则发送包含自己公钥的证书，同时对此前所有握手消息进行散列运算，并使用加密算法进行加密发送给服务器。同时，创建随机数 pre-master-secret 并使用服务器公钥进行加密发送。服务器收到这个 ClientKeyExchange 之后解密得到 pre-master-secret。服务器和客户端利用 1 阶段的随机数，能够计算得出 master-secret。
- 握手终止。服务器和客户端分别通过 ChangeCipherSpec 消息告知伺候使用 master-secret 对连接进行加密和解密，并向对方发送终止消息

```
Transport Layer Security
  TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 512
    > Handshake Protocol: Client Hello
```

图 15: Cilent hello

首先发送 ClientHello 将随机数连同自己支持的协议版本、加密算法和压缩算法发送给服务器, 流量包里也能看到客户端发送支持的加密算法

```
Transport Layer Security
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 80
    > Handshake Protocol: Server Hello
  TLSv1.2 Record Layer: Handshake Protocol: Certificate
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 674
    > Handshake Protocol: Certificate
```

图 16: Cilent hello

server hello 包里能看到服务端选择的加密算法;

```
Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 76
    Version: TLS 1.2 (0x0303)
  > Random: de88af47059ef855f6ed5e7dd98e99666e0f071bc599ea0330e6265edb40b012
    Session ID Length: 0
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Compression Method: null (0)
    Extensions Length: 36
  > Extension: renegotiation_info (len=1)
  > Extension: ec_point_formats (len=4)
  > Extension: session_ticket (len=0)
  > Extension: application_layer_protocol_negotiation (len=11)
  > Extension: extended_master_secret (len=0)
    [JA3S Fullstring: 771,49199,65281-11-35-16-23]
    [JA3S: 4ef1b297bb817d8212165a86308bac5f]
```

图 17: Cilent hello

服务器发送 ServerHello 的同时可能将包含自己公钥的证书发送给客户端

```
TCP payload (7 bytes)
Transport Layer Security
  TLSv1.2 Record Layer: Alert (Level: Fatal, Description: Certificate Unknown)
      Content Type: Alert (21)
      Version: TLS 1.2 (0x0303)
      Length: 2
    Alert Message
        Level: Fatal (2)
        Description: Certificate Unknown (46)
```

图 18: Cilent hello

客户端验证证书，如果收到 Certificate Request 则发送包含自己公钥的证书，同时对此前所有握手消息进行散列运算，并使用加密算法进行加密发送给服务器；最后断开连接仍然是 TCP 的四次挥手

172	6.268873	192.168.137.1	192.168.137.1	TCP	44 8080 → 55227 [FIN, ACK] Seq=38554 Ack=1103 Win=2618880 Len=0
173	6.268941	192.168.137.1	192.168.137.1	TCP	44 55227 → 8080 [ACK] Seq=1103 Ack=38555 Win=2580992 Len=0
174	6.269152	192.168.137.1	192.168.137.1	TCP	44 55227 → 8080 [FIN, ACK] Seq=1103 Ack=38555 Win=2580992 Len=0
175	6.269225	192.168.137.1	192.168.137.1	TCP	44 8080 → 55227 [ACK] Seq=38555 Ack=1104 Win=2618880 Len=0

图 19: 四次挥手数据抓包

五、 附录

我们已经将源代码,实验结果,实验报告等上传至 github:https://github.com/Metetor/network

参考文献