1. TLS 协议简介

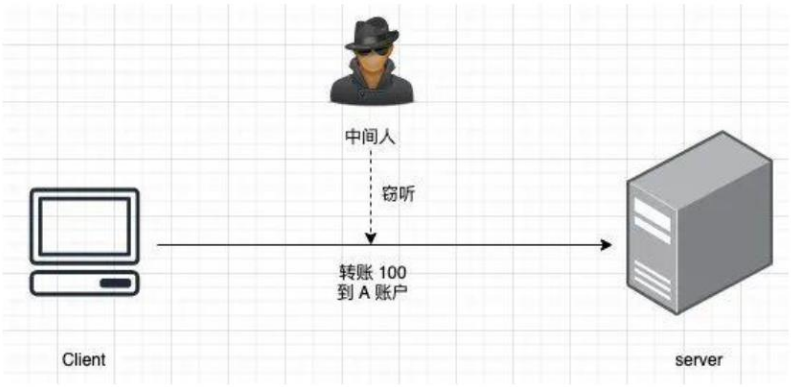
本文将会以 HTTPS 为例，来介绍 TLS 协议原理。

1.1 HTTP 为什么不安全？

在介绍 TLS 协议之前，先介绍一下 HTTP 协议存在的缺陷，以便于后续大家更好的了解 TLS 协议的设计思路。

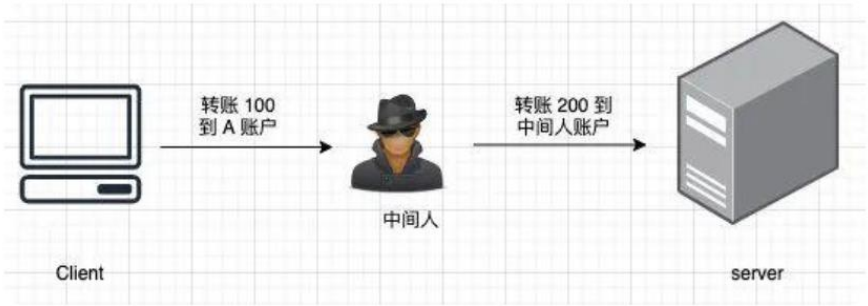
1) 窃听风险

HTTP 协议数据包在网络中以明文的形式传递，攻击者通过抓包软件捕获网络中的数据包，即可读到 HTTP 中的信息。



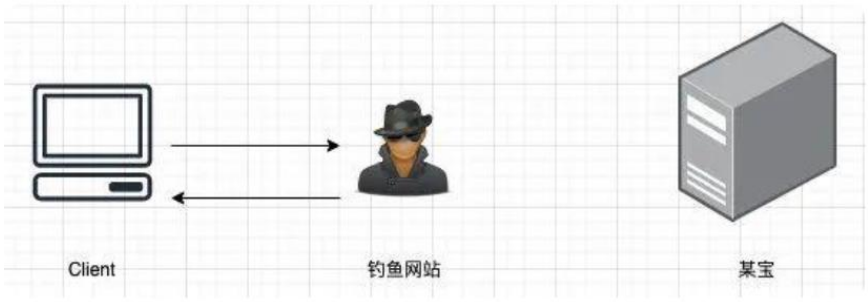
2) 篡改

攻击者（中间人）可以窃取 server 的信息，对报文内容进行篡改，然后发送给 client。



3) 冒充

client 以为在跟合法的网站在通信，实际上可能再跟一个钓鱼网站在通信。



HTTP 协议之所以会存在上述问题，主要是因为 HTTP 协议没有对应用数据进行加密，没有完整性校验和身份验证功能。

1.2 网络安全的四个原则

1) 机密性：网络中传递的数据，经过加密算法进行加密；就算中间人窃听，他也无法获取其中的内容；
2) 完整性：指数据传输过程中，没有被篡改，就算被篡改，也能检测出来。
3) 身份认证：能够确认对方的身份的合法性。
4) 不可否认：对自己操作过的网络行为，不能否认。

2.3 HTTPS 通信原理

以下例子为 HTTP+TLSv1.2 的握手和通信过程。

1) 浏览器向服务器的 443 端口发起请求，发送报文为 client hello 报文，报文中请求携带了浏览器支持的对称加密算法和哈希算法。
2) 服务器收到 client hello 报文，选择浏览器和本身都支持的加密算法和哈希算法，回应 server hello 报文。同时，服务器会将自己的证书信息发送给浏览器，这里的数字证书可以是向某个可靠机构（CA）申请的，也可以是自制的（自签名证书），使用 wireshark 抓包软件，可以捕捉到 certification、exchangeKey 和 hellodone 报文。其中 certification 携带证书信息，hellodone 表示第一阶段的握手完成（注意：服务器证书中有服务器的签名信息，签名信息怎么来的呢？服务器使用事前协商好的哈希算法对证书进行哈希运行，得到哈希值 A，然后使用密钥信息对哈希值 A 进行加密，得到密文 A，这个密文 A 就是签名）。
3) 浏览器收到服务器的证书，于是进入到认证环节。 首先浏览器会从内置的证书列表中索引，找到服务器证书的颁发机构，如果没有找到，此时就会提示用户该证书

是不是由权威机构颁发，是不可信任的。浏览器上就会显示证书不安全，由用户来选择是否信任。

4) 如果查到了对应的机构，是可信的，则取出该机构颁发的公钥。用机构的证书公钥解密得到证书的内容和证书签名，内容包括网站的网址、服务器的公钥、证书的有效期等。

5) 浏览器验证证书记录的网址是否和当前网址是一致的，不一致会提示用户。如果网址一致会检查证书有效期，证书过期了也会提示用户。这些都通过认证时，浏览器就可以安全使用证书中的网站（server）公钥了。

6) 浏览器验证证书签名的合法性，使用服务器的公钥对签名进行解密，得到哈希值 A。然后，浏览器使用事先协商好的哈希算法对证书进行哈希运行得到哈希 B。对比哈希 A 和哈希 B 是否一致。如果一致，说明证书没有被篡改，是合法的。

7) 浏览器生成一个随机数 R，并使用服务器的公钥对 R 进行加密，得到密文 RC。浏览器将密文 RC 传送给服务器。服务器收到密文 RC 后，用自己的私钥解密得到 R。

8) 后续交互应用层数据时，服务器会议 R 为密钥，使用事先协商好的对称加密算法，加密网页内容并传输给浏览器。浏览器以 R 为密钥，使用之前约定好的对称加密算法，解密网页，获取网页内容。

2. MQTT 协议的服务端和客户端如何获取证书？

2.1 服务端的配置

服务端，即 broker，代理服务器。在 broker 上的配置非常重要。

在 broker 上生成好，CA 证书，CA 密钥，broker 证书，broker 密钥，client 证书和 client 密钥。后续 client，即发布者和订阅者上，只需要直接拷贝 broker 上生成好的 CA 证书，CA 密钥，client 证书和 client 密钥即可。

2.1.1 生成 CA 证书和 CA 密钥

创建目录 certs，在 certs 目录下创建目录 ca，在 ca 目录下，使用如下命令：

```
openssl req -new -x509 -days 365 -extensions v3_ca -keyout ca.key -out ca.crt
```

```
Generating a RSA private key
.....+++++
.............................+++++
writing new private key to 'ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:FR
State or Province Name (full name) []:France
Locality Name (eg, city) [Default City]:Strasbourg
Organization Name (eg, company) [Default Company Ltd]:opeNest
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:openest.io
Email Address []:contact@openest.io
$ ls
ca.crt  ca.key
$ cd ..
```

这个地方的 Common Name 对于后续影响不大，设置为 openset.io，或者其他都可以。

2.1.2 生成 broker 密钥(即，broker.key)

在 certs 目录下创建目录 broker，在 broker 目录下，使用如下命令：

openssl genrsa -out broker.key 2048

2.1.3 使用 broker 密钥，创建一个签名请求文件(即，broker.csr)

在 broker 目录下，使用如下命令：

openssl req -out broker.csr -key broker.key –new

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:FR
State or Province Name (full name) []:France
Locality Name (eg, city) [Default City]:Strasbourg
Organization Name (eg, company) [Default Company Ltd]:opeNest
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:contact@openest.io

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
$ ls
broker.csr  broker.key
```

注意：这里的 Common Name 非常重要!! 必须填域名!!

broker 上面填主机名就可以了，例如：61 服务器的主机名为 server，那么这里就填入 server。

2.1.4 将证书签名请求(csr)文件传递给我们的验证机构

在 broker 目录下，使用如下命令：

openssl x509 -req -in broker.csr -CA ../ca/ca.crt -CAkey ../ca/ca.key -CAcreateserial -out broker.crt -days 100

```
Signature ok
subject=C = FR, ST = France, L = Strasbourg, O = opeNest, CN = localhost, emailAddress = contact@openest.io
Getting CA Private Key
Enter pass phrase for ../ca/ca.key:
$ ls
broker.crt  broker.csr  broker.key
$ rm broker.csr
$ cd ..
```

2.1.5 生成 client 密钥(即，client.key)

在 certs 目录下创建目录 client，在 client 目录下，使用如下命令：

openssl genrsa -out client.key 2048

2.1.6 根据 client 密钥，创建 client 签名请求文件(即，client.csr)

在 client 目录下，使用如下命令：

openssl req -out client.csr -key client.key –new

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:FR
State or Province Name (full name) []:France
Locality Name (eg, city) [Default City]:Strasbourg
Organization Name (eg, company) [Default Company Ltd]:opeNest
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:contact@openest.io

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

这里的 Common Name 非常重要!! 必须填域名!!

broker 上面填主机名就可以了，例如：61 服务器的主机名为 server，那么这里就填入 server。

2.1.7 将证书签名请求(csr)文件传递给我们的验证机构

在 client 目录下，使用如下命令：

openssl x509 -req -in client.csr -CA ../ca/ca.crt -CAkey ../ca/ca.key -CAcreateserial -out client.crt -days 100

到了这里单机模式下的 MQTT+TLS 就完成了，可以先测试一下单机模式能不能通。

修改配置文件：

vim /etc/mosquitto/mosquitto.conf

```
port 18883

cafile /root/cLan/mosquitto/certs/ca/ca.crt
#capath

# Path to the PEM encoded server certificate.
certfile /root/cLan/mosquitto/certs/broker/broker.crt
#certfile

# Path to the PEM encoded keyfile.
keyfile /root/cLan/mosquitto/certs/broker/broker.key
#keyfile

require_certificate true
```

重启 mosquitt 服务：

systemctl restart mosquitto.service

第一个窗口：

mosquitto_sub -p 18883 --cafile ../ca/ca.crt --cert client.crt --key client.key -h server -t ab -d

第二个窗口：

mosquitto_pub -p 18883 --cafile ../ca/ca.crt --cert client.crt --key client.key -h server -m hello -t ab –d

其中，-d 表示调试，debug，会看到明显的发布和订阅信息

2.2 发布者上的配置

a. 使用 scp 将 CA 证书，CA 密钥，client 证书和 client 密钥拷贝到发布者服务器上;

```
scp root@<broker 的 IP 地址>:/root/cLan/mosquitto/certs/ca/ca.crt ./
scp root@<broker 的 IP 地址>:/root/cLan/mosquitto/certs/ca/ca.key ./
scp root@<broker 的 IP 地址>:/root/cLan/mosquitto/certs/client/client.crt ./
scp root@<broker 的 IP 地址>:/root/cLan/mosquitto/certs/client/client.key ./
```

b. 配置 mosquitto.conf 配置文件;

```
vim /etc/mosquitto/mosquitto.conf
```

c. 重启 mosquitto 服务。

```
systemctl restart mosquitto.service
```

d. 绑定 ip 地址和域名

注意：这一步非常重要!! 因为 MQTT+TLS 访问的时候，是通过域名解析为 IP 地址，然后再根据 IP 地址去访问的!!

命令如下：

```
vim /etc/hosts
<broker 的 IP 地址>  <broker 的主机名>
```

在/etc/hosts 文件中，加入一条 ip 地址到域名的映射，即<broker 的 IP 地址> <broker 的主机名>。

2.3 订阅者上的配置

订阅者的配置和发布者的配置是一样的。

a. 使用 scp 将 CA 证书，CA 密钥，client 证书和 client 密钥拷贝到发布者服务器上;

```
scp root@<broker 的 IP 地址>:/root/cLan/mosquitto/certs/ca/ca.crt ./
scp root@<broker 的 IP 地址>:/root/cLan/mosquitto/certs/ca/ca.key ./
scp root@<broker 的 IP 地址>:/root/cLan/mosquitto/certs/client/client.crt ./
scp root@<broker 的 IP 地址>:/root/cLan/mosquitto/certs/client/client.key ./
```

b. 配置 mosquitto.conf 配置文件;

```
vim /etc/mosquitto/mosquitto.conf
```

c. 重启 mosquitto 服务;

```
systemctl restart mosquitto.service
```

d. 绑定 ip 地址和域名

注意：这一步非常重要!! 因为 MQTT+TLS 访问的时候，是通过域名解析为 IP 地址，然后再根据 IP 地址去访问的!!

命令如下：

vim /etc/hosts

<broker 的 IP 地址> <broker 的主机名>

在/etc/hosts 文件中，加入一条 ip 地址到域名的映射，即<broker 的 IP 地址> <broker 的主机名>。

2.4 实验验证

实验环境，使用 mosquitto 工具，通过命令的方式进行验证。假设选择服务器 A 作为发布者，服务器 B 作为订阅者，服务器 C 作为 broker。

发布者：

mosquitto_pub -p 18883 --cafile ../ca/ca.crt --cert client.crt --key client.key -h server -m hello -t ab –d

订阅者：

mosquitto_sub -p 18883 --cafile ../ca/ca.crt --cert client.crt --key client.key -h server -t ab -d

发布者：

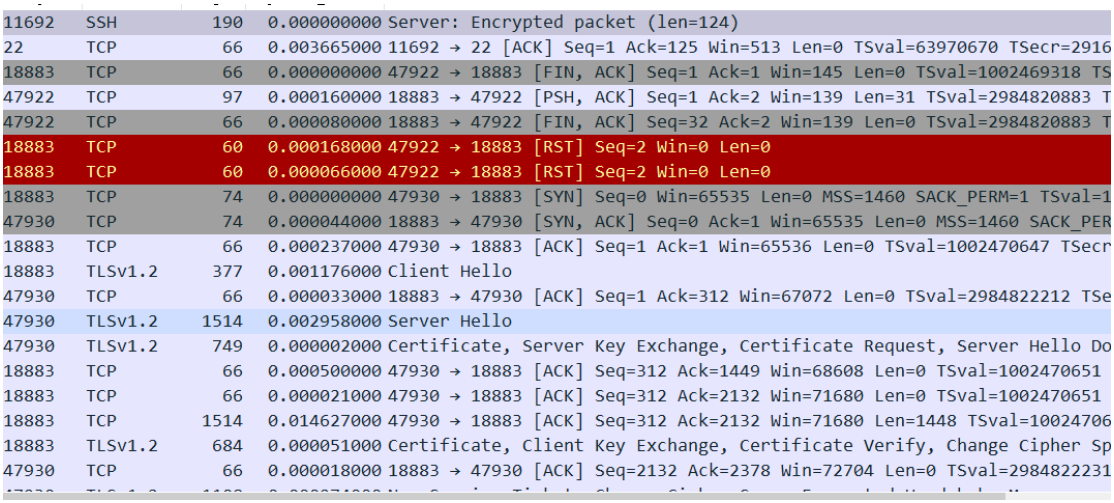
```
[root@cnware-20060 client]# mosquitto_pub -p 18883 --cafile ../ca/ca.crt --cert client.crt --key client.key -h libing-dev-61
-m hello -t ab -d
Client mosqpub|28782-cnware-20 sending CONNECT
Client mosqpub|28782-cnware-20 received CONNACK (0)
Client mosqpub|28782-cnware-20 sending PUBLISH (d0, q0, r0, m1, 'ab', ... (5 bytes))
Client mosqpub|28782-cnware-20 sending DISCONNECT
```

订阅者：

```
[root@cnware-20050 client]# mosquitto_sub -p 18883 --cafile ../ca/ca.crt --cert client.crt --key client.key -h libing-
dev-61 -t ab -d
Client mosqsub|3226-cnware-200 sending CONNECT
Client mosqsub|3226-cnware-200 received CONNACK (0)
Client mosqsub|3226-cnware-200 sending SUBSCRIBE (Mid: 1, Topic: ab, QoS: 0)
Client mosqsub|3226-cnware-200 received SUBACK
Subscribed (mid: 1): 0
Client mosqsub|3226-cnware-200 received PUBLISH (d0, q0, r0, m0, 'ab', ... (5 bytes))
hello
Client mosqsub|3226-cnware-200 received PUBLISH (d0, q0, r0, m0, 'ab', ... (5 bytes))
hello
^C
```

2.3 实验效果

在 broker 上使用 tcpdump 进行抓包，可以获取发布者发送的数据，通过 wireshark 打开，可以看到信息都通过 TLS 协议进行加密。



11692	SSH	190	0.000000000	Server: Encrypted packet (len=124)
22	TCP	66	0.003665000	11692 → 22 [ACK] Seq=1 Ack=125 Win=513 Len=0 TSval=63970670 TSecr=2916
18883	TCP	66	0.000000000	47922 → 18883 [FIN, ACK] Seq=1 Ack=1 Win=145 Len=0 TSval=1002469318 TS
47922	TCP	97	0.000160000	18883 → 47922 [PSH, ACK] Seq=1 Ack=2 Win=139 Len=31 TSval=2984820883 T
47922	TCP	66	0.000080000	18883 → 47922 [FIN, ACK] Seq=32 Ack=2 Win=139 Len=0 TSval=2984820883 T
18883	TCP	60	0.000168000	47922 → 18883 [RST] Seq=2 Win=0 Len=0
18883	TCP	60	0.000066000	47922 → 18883 [RST] Seq=2 Win=0 Len=0
18883	TCP	74	0.000000000	47930 → 18883 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1 TSval=1
47930	TCP	74	0.000044000	18883 → 47930 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 SACK_PER
18883	TCP	66	0.000237000	47930 → 18883 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1002470647 TSecr
18883	TLSv1.2	377	0.001176000	Client Hello
47930	TCP	66	0.000033000	18883 → 47930 [ACK] Seq=1 Ack=312 Win=67072 Len=0 TSval=2984822212 TSe
47930	TLSv1.2	1514	0.002958000	Server Hello
47930	TLSv1.2	749	0.000002000	Certificate, Server Key Exchange, Certificate Request, Server Hello Do
18883	TCP	66	0.000500000	47930 → 18883 [ACK] Seq=312 Ack=1449 Win=68608 Len=0 TSval=1002470651
18883	TCP	66	0.000021000	47930 → 18883 [ACK] Seq=312 Ack=2132 Win=71680 Len=0 TSval=1002470651
18883	TCP	1514	0.014627000	47930 → 18883 [ACK] Seq=312 Ack=2132 Win=71680 Len=1448 TSval=10024706
18883	TLSv1.2	684	0.000051000	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Sp
47930	TCP	66	0.000018000	18883 → 47930 [ACK] Seq=2132 Ack=2378 Win=72704 Len=0 TSval=2984822231

```
> Transmission Control Protocol, Src Port: 18883, Dst Port: 47930, Seq: 3174, Ack: 2424, Len: 33
v Transport Layer Security
  v TLSv1.2 Record Layer: Application Data Protocol: Application Data
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 28
      Encrypted Application Data: 7f9096b49b157b2c4f36455651f61b08785f16f91904f5e5fb41b607
```