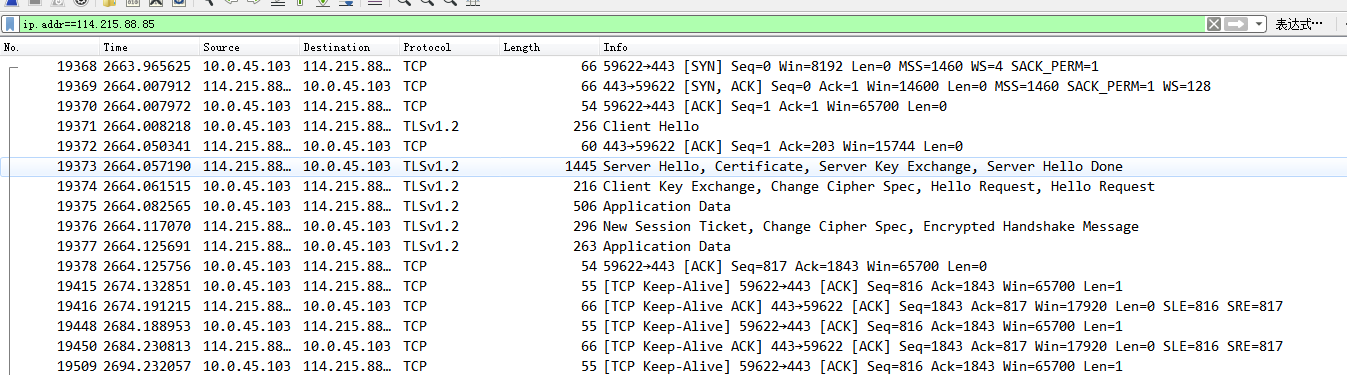
**使用wireshark分析HTTPS流程的建立**

摘要: https流程

**一、概要**

为了网站以及用户的安全性，现在很多的网站都是https，大家都知道tcp通过三次握手建立连接，并且还有很多的同学对https连接建立的流程不太明白，包括我自己，通过借助于wireshark这种抓包工具，我们可以尝试着了解一下大概的流程。

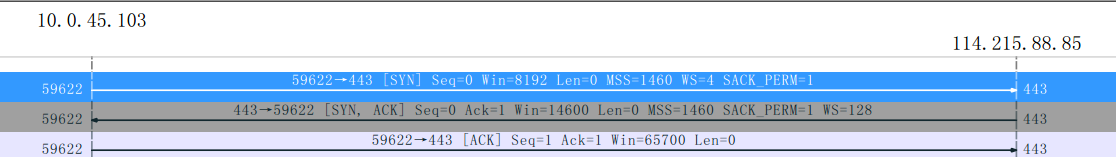


（图1）

本图是客户端（10.0.45.103）访问服务端（114.215.88.85）通过wireshark抓包显示出来的双方交互数据，访问是通过https访问服务器上的一台nginx服务器服务。请关注第一列的No。下文中很多时候会用no表示一次交互。

图中可以很明显的看出tcp的三次握手以及之后的TLS加密流程的建立。

**二、tcp的三次握手**



通过流程图可以看出（也可以看图1 的19368到19370这三个编号），首先客户端向服务端发起一个SYN的请求，序号(Seq)为0，确认号（ACK）也为0，这代表是本次是由客户端发起的首次请求。本次请求的数据长度为0

然后服务端给客户端响应，此时服务端的Seq也是0，值得是服务端的第一回应，并且给客户端说，你的请求我已经收到了（ACK=1），

最后返还给客户端以后，客户端的序号+1，并且对服务端的响应做出确认，最后回给服务端，此时ACK=1，Seq=1

tcp的握手过程建立成功。

**三、ssl连接的建立**

通过以上可以看出，SSL也是建立在TCP的基础上的，经过tcp的三次握手，接下来才是加密信道的建立。

客户端和服务端建立安全连接大致经过以下几个步骤

1. 客户端：ClientHello
2. 服务端：Server Hello，Server certificate，Server Exchange，Server Hello Done
3. 客户端：client Exchange
4. 客户端：Application Data
5. 服务端：New Session
6. 服务端：Application Data

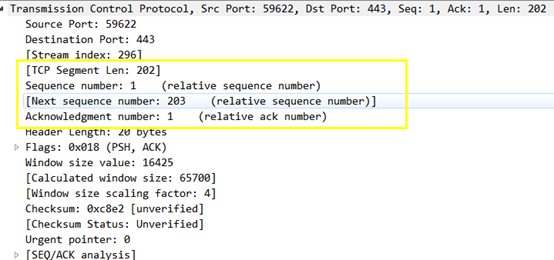
接下来看这几个步骤中具体的每一个步骤传输的内容

**ClientHello**

https://static.oschina.net/uploads/space/2016/1125/134547_ojw2_2457218.png

client首先给服务端打招呼，对服务端说hello

应用层没什么特别的



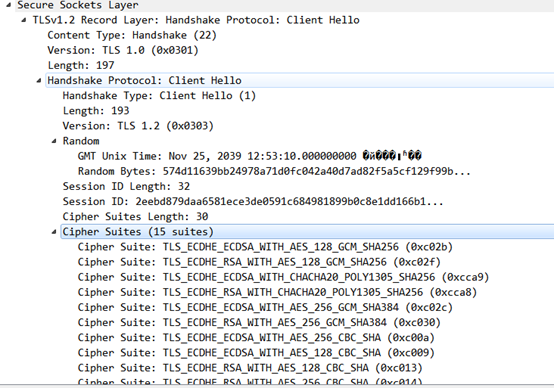
客户端向服务端发送202个字节的数据，并且客户端此时的 seq num 为1 ，tcp三次握手已经通过了，所以客户端ack num 确认的是服务端的tcp的最后一次信息。由于这次发送的长度是202个字节，所以给服务端说，下一个字节序列号是从203开始的。

标志位为ACK和PSH

但是这次多了一个 **Secure Socket layer**层。协议使用的时候，用的是Handshake Protocol，给服务端发消息ClientHello

https://static.oschina.net/uploads/space/2016/1125/134600_evqf_2457218.png

详细的信息如下：



Secure Socekts layer层使用的是版本是TLS  1.0

HandShake Type的类型，是客户端打招呼 client hello

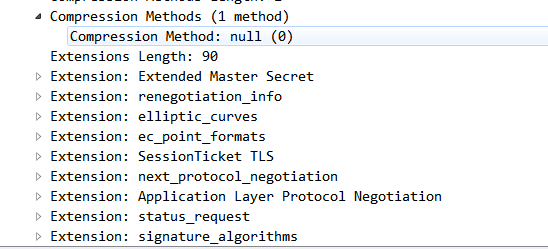
HandShake protocol 协议使用的是TLS 1.2

发送的信息还有客户端在本地生成的随机码（**Random**）

然后客户端声明自己所支持的加密套件（**Cipher Suites**）这个客户端支持15种加密套件

加密套件中表明了使用的对称加密算法，非对称加密算法，摘要算法以及使用的是TLS或者是SSL

还有一些其他的信息



第一行指明是否进行了压缩以及使用的压缩算法，第二行null表明未进行压缩，以及选用相关的压缩算法或者压缩工具

剩下的就是一些扩展的字段了

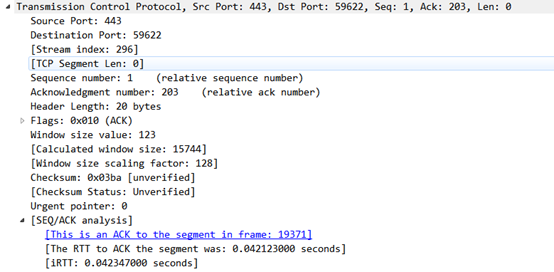
总结下来，客户端向服务端第一次打招呼（client Hello）的时候实际上主要发送了以下主要的信息

**客户端的随机数**

**客户端所支持的加密套件**

**以及客户端和服务器之间的sessionId**

接下来就是服务端对客户端Hello的第一次回应，也就是编号19372



可以看到 服务端使用的是443端口，序列号（Sequence number）也是1 ，并且回应客户端说我已经确认收到你的202个自己的数据（203-1），Flags表明本次是服务端反馈给客户端请求的应答（蓝色的文字也写出来了，这是一个队19371编号的应答）

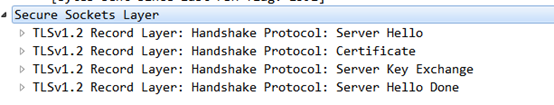
从图1可以看出这是一个TCP连接

**Server Hello**

接下来不等客户端反应，服务端又给客户端发送了19373的数据，而这个数据就是使用了TLS1.2协议了。

摘要信息中说明了这是服务端的的hello,Server hello,服务端秘钥交换，服务端hello done。接下来看服务端发过来的具体都有什么。

传输控制层（**transmission control protocol**）和上面一样，主要是一些Flags，端口以及数据长度的确认等等，主要看一下安全套接字层的东西（**Secure Sockets Layer**）中的东西。

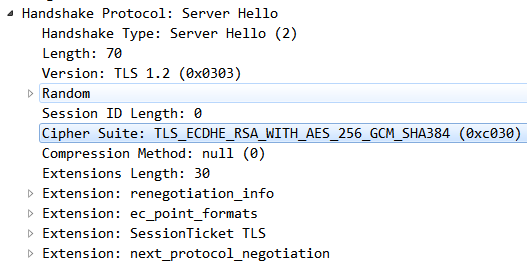


通过上图，可以看出服务端主要返回四种内容。

* Server Hello 服务端的回应客户端的hello信息
* Certificate  服务端证书
* Server key Exchange 服务器秘钥交换
* Server Hello Done 服务器信息发送完毕

详细看一下各个Record layer中的具体内容

**Server Hello**



在Server Hello中，服务器返回的服务端的随机数，所选用的TLS 版本，以及服务器最终选用的客户端和服务端交互的加密套件（**TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384** (0xc030)），**最终选用的加密套件是RSA非对称加密算法以及AES对称加密算法，用的是SHA384做摘要**，注意，这个值必须是客户端发给服务端的列表中选出来的。实际上如果客户端只能支持版本和安全性比较低的加密套件，这样服务端选择和客户端交互的时候也被迫只能使用低版本的加密套件，其安全性就会降低。除了以上这些，服务端Server Hello时发送回来的还有是否使用了压缩以及一些其他的扩展字段

**Certificate**

Server Hello以后，服务端会发送公钥证书给客户端

Certificates (953 bytes)

Certificate Length: 950

Certificate: 308203b23082029aa003020102020101300d06092a864886... (id-at-commonName=www.wtf.com,id-at-organizationName=JD,id-at-stateOrProvinceName=BJ,id-at-countryName=CN)

signedCertificate

version: v3 (2)

serialNumber: 1

signature (sha256WithRSAEncryption)

Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)

issuer: rdnSequence (0)

rdnSequence: 7 items (pkcs-9-at-emailAddress=iloveme313@163.com,id-at-commonName=wangtengfei,id-at-organizationalUnitName=section,id-at-organizationName=JD,id-at-localityName=BJ,id-at-stateOrProvinceName=BJ,id-at-countryName=CN)

RDNSequence item: 1 item (id-at-countryName=CN)

RDNSequence item: 1 item (id-at-stateOrProvinceName=BJ)

RDNSequence item: 1 item (id-at-localityName=BJ)

RDNSequence item: 1 item (id-at-organizationName=JD)

RDNSequence item: 1 item (id-at-organizationalUnitName=section)

RDNSequence item: 1 item (id-at-commonName=wangtengfei)

RDNSequence item: 1 item (pkcs-9-at-emailAddress=iloveme313@163.com)

validity

notBefore: utcTime (0)

utcTime: 16-11-22 06:38:18 (UTC)

notAfter: utcTime (0)

utcTime: 17-11-22 06:38:18 (UTC)

subject: rdnSequence (0)

rdnSequence: 4 items (id-at-commonName=www.wtf.com,id-at-organizationName=JD,id-at-stateOrProvinceName=BJ,id-at-countryName=CN)

RDNSequence item: 1 item (id-at-countryName=CN)

RDNSequence item: 1 item (id-at-stateOrProvinceName=BJ)

RDNSequence item: 1 item (id-at-organizationName=JD)

RDNSequence item: 1 item (id-at-commonName=www.wtf.com)

subjectPublicKeyInfo

algorithm (rsaEncryption)

Algorithm Id: 1.2.840.113549.1.1.1 (rsaEncryption)

subjectPublicKey: 3082010a0282010100be56d1a2b725cf5d6fa1997c83b221...

modulus: 0x00be56d1a2b725cf5d6fa1997c83b221de8452658b1e7c86...

publicExponent: 65537

extensions: 4 items

algorithmIdentifier (sha256WithRSAEncryption)

Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)

Padding: 0

encrypted: 41bfd96f86c44a731d6ff7af7e9e666703c744aa8c691d38...

1中有证书的指纹，以及证书的签名信息

Certificate:308203b23082029aa003020102020101300d06092a864886...(id-at-commonName=www.wtf.com,id-at-organizationName=JD,id-at-stateOrProvinceName=BJ,id-at-countryName=CN)

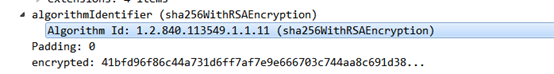
看以看出，这个证书是针对[www.wtf.com](http://www.wtf.com)这个域名签发的，然后就是一些签发机构信息等。

2 证书的过期时间（validity），可以看出证书的过期时间为1年，从16年11月22日 6点38分18秒(**not Before**)到从17年11月22日 6点38分18秒**(not After)**

3 证书签发的主体(subject)

4 公钥证书的key.

当然这证书下放以后，服务端会把签发的证书的hash也发送过来，以证明此证书在传递过程中没有被修改。

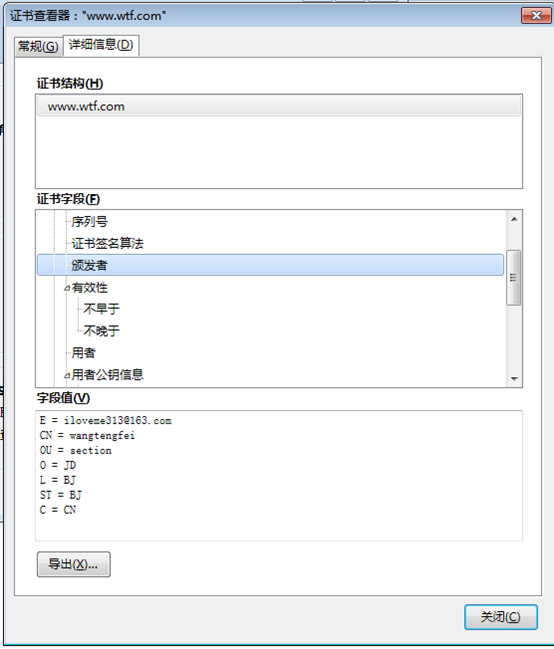


可以看出使用的sha256算法做的摘要。

证书我们也能在浏览器中用可视化的方法查看。



我会把证书导出，可以自行查看



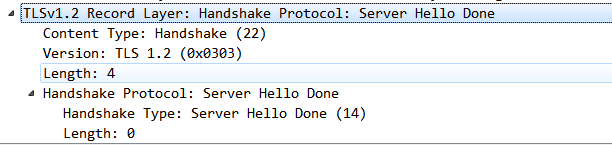
接下来就是秘钥交换了

**Server key Exchange**



这个就是服务端公钥的key

**HelloDone**



最后就是服务端发送一个HelloDone标示本次内容全部发放完毕。

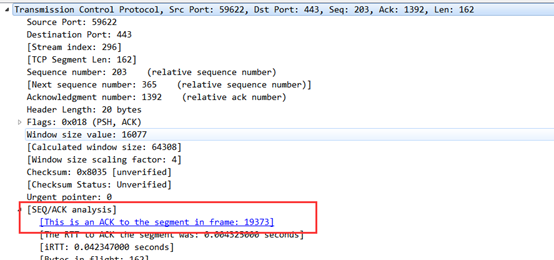
总结一下 在客户端向服务端打招呼以后，服务端主要发送了那些内容。首先是发送**服务端自己的随机数**给客户端，然后下发**服务端公钥证书**和下发**公钥的key**，最后服务端用**hellodone标示**此次内容全部发送完毕。以上发送过程我们都可以看得到，也就是说在这个过程中，客户端和服务端交互使用的都还是明文。

**Client keyExchange**

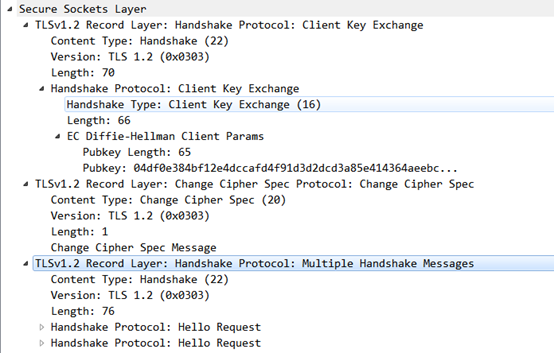
客户端接收到服务端的这些证书信息以后，解析来会进行回应服务端。可以看19374编号

https://static.oschina.net/uploads/space/2016/1125/134755_GHFK_2457218.png

查看传输层，传输层中明确写明了这是对19373的一个回应。



重点还是查看安全套接字层，具体来看回应的是什么



主要包含了如下几个方面

**Client key exchange**

包含了pre-master secret。这是在握手过程中生成的第三个随机数。分别是client hello阶段客户端的随机数，然后在server hello 阶段服务端的随机数。这一次是客户端接收到服务端的证书以后，生成的一个预加密因子（供对称加密使用）。

**Change cipher spec**

客户端通知服务端接下来就要使用加密的方式来进行通信了。

然后在19375中，客户端发送给服务端一段加密后的内容（**就是在这一步，数据的交互从明文开始转化为密文**），并且把本次会话信息中的所有细节做摘要发给服务端，以证明在本次过程中双方达成一致，未受到其他干扰



**服务端确认客户端加密请求**

然后服务端会在19376和19377中，服务端对客户端的发送的内容进行确认（19374），提供新的Session Ticket，然后发送了一段内容，并且对所有握手内容做摘要，发给客户端来验证通信过程是否被篡改。

最后在19378中，客户端对服务端发送的内容进行做确认

https://static.oschina.net/uploads/space/2016/1125/134819_wcos_2457218.png

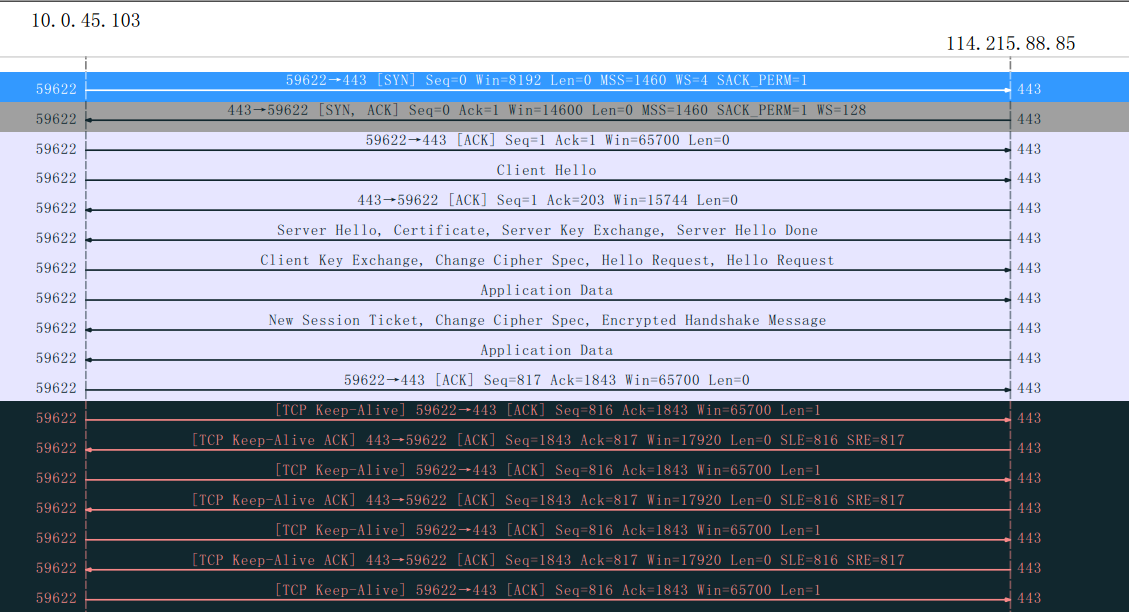
https://static.oschina.net/uploads/space/2016/1125/134825_hEmw_2457218.png

注意一下，这时候内容发送的协议已经不是SSL了，已经重新更改为了TCP

接下来就是在此加密信道上开始正常的业务数据传输过程。

（tcp.stream eq 296）

整个交互过程的概要可以用下图表示



小结：

本文大概的跟踪了以下https建立安全链接的过程，并未提到常用的加密算法什么的用到什么地方，在本文的最后提一下，https过程的建立，实际上就是一次一密，根本目的是为了协商出双方使用哪种对称加密算法，然后加密因子是多少，为了达到这个目的，双方使用了非对称加密进行协商，拿到双方的随机数，然后进行相关的计算得到一个随机对称加密的加密因子。为了确保客户端正在和真正的不是冒牌的服务器在协商，这个就用到了证书，由于是证书是可以自签名的，自签名是不可信的，所以就需要就需要找一个大家都信任的机构进行签名，例如CA。这样浏览器才会认可这个证书。还有就是为了保证在协商的过程中不受干扰，在协商的每一步都会把本次会话的信息做摘要传递给双方，用以校验。

当然了本文只是一个极其粗略的https流程的简介，https还涉及到很多其他的内容，例如一些https加密信道的重用，由于https的建立比较慢，所以google就发明了一种更快的协议QUIC。

由于作者水平有限，毕竟会有些疏漏以及理解不到位的地方，希望读者给予指正，以使其他人不被误解。

附：

https://git.oschina.net/tofu.wang/wireshark-https/tree/master

从这个里面可以下载到本次模拟的证书以及具体用wireshark抓包过来的包，然后打开wireshark进行导入，过滤器写ip.addr==114.215.88.85 即可