HTTPS 基本过程

HTTPS 即 HTTP over TLS，是一种在加密信道进行 HTTP 内容传输的协议。

> TLS 的早期版本叫做 SSL。SSL 的 1.0, 2.0, 3.0 版本均已经被废弃，出于安全问题考虑广大浏览器也不再对老旧的 SSL 版本进行支持了，因此这里我们就统一使用 TLS 名称了。

TLS 的基本过程如下（取自 what-happens-when-zh_CN）：

- 客户端发送一个 `ClientHello` 消息到服务器端，消息中同时包含了它的 Transport Layer Security (TLS) 版本，可用的加密算法和压缩算法。
- 服务器端向客户端返回一个 `ServerHello` 消息，消息中包含了服务器端的 TLS 版本，服务器所选择的加密和压缩算法，以及数字证书认证机构（Certificate Authority，缩写 CA）签发的服务器公开证书，证书中包含了公钥。客户端会使用这个公钥加密接下来的握手过程，直到协商生成一个新的对称密钥。证书中还包含了该证书所应用的域名范围（Common Name，简称 CN），用于客户端验证身份。
- 客户端根据自己的信任 CA 列表，验证服务器端的证书是否可信。如果认为可信（具体的验证过程在下一节讲解），客户端会生成一串伪随机数，使用服务器的公钥加密它。这串随机数会被用于生成新的对称密钥
- 服务器端使用自己的私钥解密上面提到的随机数，然后使用这串随机数生成自己的对称主密钥
- 客户端发送一个 `Finished` 消息给服务器端，使用对称密钥加密这次通讯的一个散列值
- 服务器端生成自己的 hash 值，然后解密客户端发送来的信息，检查这两个值是否对应。如果对应，就向客户端发送一个 `Finished` 消息，也使用协商好的对称密钥加密
- 从现在开始，接下来整个 TLS 会话都使用对称秘钥进行加密，传输应用层（HTTP）内容

从上面的过程可以看到，TLS 的完整过程需要三个算法（协议），密钥交互算法，对称加密算法，和消息认证算法（TLS 的传输会使用 MAC(message authentication code) 进行完整性检查）。

我们以 Github 网站使用的 TLS 为例，使用浏览器可以看到它使用的加密为 `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`。其中密钥交互算法是 `ECDHE_RSA`，对称加密算法是 `AES_128_GCM`，消息认证（MAC）算法为 `SHA256`。

TLS 证书机制

HTTPS 过程中很重要的一个步骤，是服务器需要有 CA 颁发的证书，客户端根据自己的信任 CA 列表验证服务器的身份。现代浏览器中，证书验证的过程依赖于证书信任链。

所谓证书信任链，即一个证书要依靠上一级证书来证明自己是可信的，最顶层的证书被称为根证书，拥有根证书的机构被称为根 CA。

还是以 Github 为例，在浏览器中我们可以看到它的证书信任链如下：

```
DigiCert High Assurance EV Root CA -> DigiCert SHA2 Extended Validation Server CA -> Github.com
```

从上到下即 Root CA -> 二级 CA -> 网站。

前面提到，证书当中包括 CN(Common Name)，浏览器在验证证书的同时，也会验证 CN 的正确性。即不光需要验证“这是一个合法的证书”，还需要验证“这是一个用于 Github.com 的证书”。

既然所有的信任，最终要落到根 CA 上，根证书本身又是怎么获得的呢？答案也很简单，根证书一般是操作系统自带的。不管是桌面系统 Windows，macOS 还是移动端系统 Android, iOS 都会内置一系列根证书。随着操作系统本身的升级，根证书也会随着升级进行更新。

对浏览器而已，浏览器当然也有选择信任某个根证书的权利。Chrome 浏览器一般是跟随系统根证书信任的。Firefox 浏览器通常是使用自带的一套证书信任机制，不受系统证书的影响。

在使用 `curl` 等工具时，我们还可以自行选择证书进行信任。

> 有权威的信任，最终都要落到一个单点信任，不管是 Root CA，还是微软，苹果，谷歌等操作系统厂商。

中间人攻击

HTTPS 的过程并不是密不透风的，HTTPS 有若干漏洞，给中间人攻击（Man In The Middle Attack，简称 MITM）提供了可能。

所谓中间人攻击，指攻击者与通讯的两端分别建立独立的联系，并交换其所收到的数据，使通讯的两端认为他们正在通过一个私密的连接与对方直接对话，但事实上整个会话都被攻击者完全控制。在中间人攻击中，攻击者可以拦截通讯双方的通话并插入新的内容。

SSL 剥离

SSL 剥离即阻止用户使用 HTTPS 访问网站。由于并不是所有网站都只支持 HTTPS，大部分网站会同时支持 HTTP 和 HTTPS 两种协议。用户在访问网站时，也可能会在地址栏中输入 `http://` 的地址，第一次的访问完全是明文的，这就给了攻击者可乘之机。通过攻击 DNS 响应，攻击者可以将自己变成中间人。

> DNS 作为基于 UDP 的协议是相当不安全的，为了保证 DNS 的安全可以使用 DNS over TCP 等机制，这里不赘述了。

HSTS

为了防止上面说的这种情况，一种叫做 HSTS 的技术被引入了。HSTS（HTTP Strict Transport Security）是用于强制浏览器使用 HTTPS 访问网站的一种机制。它的基本机制是在服务器返回的响应中，加上一个特殊的头部，指示浏览器对于此网站，强制使用 HTTPS 进行访问：

```
Strict-Transport-Security: max-age=31536000; includeSubdomains; preload
```

可以看到如果这个过期时间非常长，就是导致在很长一段时间内，浏览器都会强制使用 HTTPS 访问该网站。

HSTS 有一个很明显的缺点，是需要等待第一个服务器的影响中的头部才能生效，但如果第一次访问该网站就被攻击呢？为了解决这个问题，浏览器中会带上一些网站的域名，被称为 HSTS preload list。对于在这个 list 的网站来说，直接强制使用 HTTPS。

伪造证书攻击

HSTS 只解决了 SSL 剥离的问题，然而即使在全程使用 HTTPS 的情况下，我们仍然有可能被监听。

假设我们想访问 `www.google.com`，但我们的 DNS 服务器被攻击了，指向的 IP 地址并非 Google 的服务器，而是攻击者的 IP。当攻击者的服务器也有合法的证书的时候，我们的浏览器就会认为对方是 Google 服务器，从而信任对方。这样，攻击者便可以监听我们和谷歌之前的所有通信了。

可以看到攻击者有两步需要操作，第一步是需要攻击 DNS 服务器。第二步是攻击者自己的证书需要被用户信任，这一步对于用户来说是很难控制的，需要证书颁发机构能够控制自己不滥发证书。

> 2015 年 Google 称发现赛门铁克旗下的 Thawte 未经同意签发了众多域名的数千个证书，其中包括 Google 旗下的域名和不存在的域名。当年 12 月，Google 发布公告称 Chrome、Android 及其他 Google 产品将不再信任赛门铁克旗下的"Class 3 Public Primary CA"根证书。

> 2016 年 Mozilla 发现沃通 CA 存在严重的信任问题，例如偷签 `github.com` 的证书，故意倒填证书日期绕过浏览器对 SHA-1 证书的限制等，将停止信任 WoSign 和 StartCom 签发的新证书。

HPKP

HPKP 技术是为了解决伪造证书攻击而诞生的。

HPKP（Public Key Pinning Extension for HTTP）在 HSTS 上更进一步，HPKP 直接在返回头中存储服务器的公钥指纹信息，一旦发现指纹和实际接受到的公钥有差异，浏览器就可以认为正在被攻击：

```
Public-Key-Pins: pin-sha256="base64=="; max-age=expireTime [; includeSubDomains][; report-uri="reportURI"]
```

和 HSTS 类似，HPKP 也依赖于服务器的头部返回，不能解决第一次访问的问题，浏览器本身也会内置一些 HPKP 列表。

> HPKP 技术仍然不能阻止第一次访问的攻击问题，部署和配置 HPKP 相当繁琐，一旦网站配置错误，就会导致网站证书验证失败，且在过期时间内无法有效恢复。HPKP 的机制也引来了一些安全性问题。Chrome 67 中废除了对 HPKP 的支持，在 Chrome 72 中 HPKP 被彻底移除。