关于启用 HTTPS 的一些经验分享（一）

**文章目录**

* [理解 Mixed Content](https://imququ.com/post/sth-about-switch-to-https.html#toc-0)
  + [早期的 IE](https://imququ.com/post/sth-about-switch-to-https.html#toc-0-0)
  + [比较新的 IE](https://imququ.com/post/sth-about-switch-to-https.html#toc-0-1)
  + [现代浏览器](https://imququ.com/post/sth-about-switch-to-https.html#toc-0-2)
  + [移动浏览器](https://imququ.com/post/sth-about-switch-to-https.html#toc-0-3)
* [合理使用 CSP](https://imququ.com/post/sth-about-switch-to-https.html#toc-1)
  + [block-all-mixed-content](https://imququ.com/post/sth-about-switch-to-https.html#toc-1-0)
  + [upgrade-insecure-requests](https://imququ.com/post/sth-about-switch-to-https.html#toc-1-1)
* [合理使用 HSTS](https://imququ.com/post/sth-about-switch-to-https.html#toc-2)
  + [HSTS 基本使用](https://imququ.com/post/sth-about-switch-to-https.html#toc-2-0)
  + [HSTS Preload List](https://imququ.com/post/sth-about-switch-to-https.html#toc-2-1)
* [CDN 安全](https://imququ.com/post/sth-about-switch-to-https.html#toc-3)
  + [合理使用 SRI](https://imququ.com/post/sth-about-switch-to-https.html#toc-3-0)
  + [了解 Keyless SSL](https://imququ.com/post/sth-about-switch-to-https.html#toc-3-1)

随着国内网络环境的持续恶化，各种篡改和劫持层出不穷，越来越多的网站选择了全站 HTTPS。就在今天，免费提供证书服务的 [Let's Encrypt](https://letsencrypt.org/) 项目也正式开放，HTTPS 很快就会成为 WEB 必选项。HTTPS 通过 TLS 层和证书机制提供了内容加密、身份认证和数据完整性三大功能，可以有效防止数据被查看或篡改，以及防止中间人冒充。本文分享一些启用 HTTPS 过程中的经验，重点是如何与一些新出的安全规范配合使用。至于 HTTPS 的部署及优化，之前写过很多，本文不重复了。

**理解 Mixed Content**

HTTPS 网页中加载的 HTTP 资源被称之为 Mixed Content（混合内容），不同浏览器对 Mixed Content 有不一样的处理规则。

**早期的 IE**

早期的 IE 在发现 Mixed Content 请求时，会弹出「是否只查看安全传送的网页内容？」这样一个模态对话框，一旦用户选择「是」，所有 Mixed Content 资源都不会加载；选择「否」，所有资源都加载。

**比较新的 IE**

比较新的 IE 将模态对话框改为页面底部的提示条，没有之前那么干扰用户。而且默认会加载图片类 Mixed Content，其它如 JavaScript、CSS 等资源还是会根据用户选择来决定是否加载。

**现代浏览器**

现代浏览器（Chrome、Firefox、Safari、Microsoft Edge），基本上都遵守了 W3C 的 [Mixed Content](http://www.w3.org/TR/mixed-content/) 规范，将 Mixed Content 分为 Optionally-blockable 和 Blockable 两类：

Optionally-blockable 类 Mixed Content 包含那些危险较小，即使被中间人篡改也无大碍的资源。现代浏览器默认会加载这类资源，同时会在控制台打印警告信息。这类资源包括：

* 通过 <img> 标签加载的图片（包括 SVG 图片）；
* 通过 <video> / <audio> 和 <source> 标签加载的视频或音频；
* 预读的（Prefetched）资源；

除此之外所有的 Mixed Content 都是 Blockable，浏览器必须禁止加载这类资源。所以现代浏览器中，对于 HTTPS 页面中的 JavaScript、CSS 等 HTTP 资源，一律不加载，直接在控制台打印错误信息。

**移动浏览器**

前面所说都是桌面浏览器的行为，移动端情况比较复杂，当前大部分移动浏览器默认允许加载所有 Mixed Content。也就是说，对于移动浏览器来说，HTTPS 中的 HTTP 资源，无论是图片还是 JavaScript、CSS，默认都会加载。

补充：上面这段结论源自于我大半年前的测试，本文评论中的 ayanamist 同学反馈现状已经有所变化。我又做了一些测试，果然随着操作系统的升级，移动浏览器都开始遵循 Mixed Content 规范了。最新测试表明，对于 Blockable 类 Mixed Content：

* iOS 9 以下的 Safari，以及 Android 5 以下的 Webview，默认会加载；
* Android 各版本的 Chrome，iOS 9+ 的 Safari，Android 5+ 的 Webview，默认不会加载；

一般选择了全站 HTTPS，就要避免出现 Mixed Content，页面所有资源请求都走 HTTPS 协议才能保证所有平台所有浏览器下都没有问题。

**合理使用 CSP**

CSP，全称是 Content Security Policy，它有非常多的指令，用来实现各种各样与页面内容安全相关的功能。这里只介绍两个与 HTTPS 相关的指令，更多内容可以看我之前写的《[Content Security Policy Level 2 介绍](https://imququ.com/post/content-security-policy-level-2.html)》。

**block-all-mixed-content**

前面说过，对于 HTTPS 中的图片等 Optionally-blockable 类 HTTP 资源，现代浏览器默认会加载。图片类资源被劫持，通常不会有太大的问题，但也有一些风险，例如很多网页按钮是用图片实现的，中间人把这些图片改掉，也会干扰用户使用。

通过 CSP 的 block-all-mixed-content 指令，可以让页面进入对混合内容的严格检测（Strict Mixed Content Checking）模式。在这种模式下，所有非 HTTPS 资源都不允许加载。跟其它所有 CSP 规则一样，可以通过以下两种方式启用这个指令：

HTTP 响应头方式：

Content-Security-Policy: block-all-mixed-content

<meta> 标签方式：

<meta http-equiv="Content-Security-Policy" content="block-all-mixed-content">

**upgrade-insecure-requests**

历史悠久的大站在往 HTTPS 迁移的过程中，工作量往往非常巨大，尤其是将所有资源都替换为 HTTPS 这一步，很容易产生疏漏。即使所有代码都确认没有问题，很可能某些从数据库读取的字段中还存在 HTTP 链接。

而通过 upgrade-insecure-requests 这个 CSP 指令，可以让浏览器帮忙做这个转换。启用这个策略后，有两个变化：

* 页面所有 HTTP 资源，会被替换为 HTTPS 地址再发起请求；
* 页面所有站内链接，点击后会被替换为 HTTPS 地址再跳转；

跟其它所有 CSP 规则一样，这个指令也有两种方式来启用，具体格式请参考上一节。需要注意的是 upgrade-insecure-requests 只替换协议部分，所以只适用于 HTTP/HTTPS 域名和路径完全一致的场景。

**合理使用 HSTS**

在网站全站 HTTPS 后，如果用户手动敲入网站的 HTTP 地址，或者从其它地方点击了网站的 HTTP 链接，依赖于服务端 301/302 跳转才能使用 HTTPS 服务。而第一次的 HTTP 请求就有可能被劫持，导致请求无法到达服务器，从而构成 HTTPS 降级劫持。

**HSTS 基本使用**

这个问题可以通过 HSTS（HTTP Strict Transport Security，[RFC6797](https://tools.ietf.org/html/rfc6797)）来解决。HSTS 是一个响应头，格式如下：

Strict-Transport-Security: max-age=expireTime [; includeSubDomains] [; preload]

**max-age**，单位是秒，用来告诉浏览器在指定时间内，这个网站必须通过 HTTPS 协议来访问。也就是对于这个网站的 HTTP 地址，浏览器需要先在本地替换为 HTTPS 之后再发送请求。

**includeSubDomains**，可选参数，如果指定这个参数，表明这个网站所有子域名也必须通过 HTTPS 协议来访问。

**preload**，可选参数，后面再介绍它的作用。

HSTS 这个响应头只能用于 HTTPS 响应；网站必须使用默认的 443 端口；必须使用域名，不能是 IP。而且启用 HSTS 之后，一旦网站证书错误，用户无法选择忽略。

**HSTS Preload List**

可以看到 HSTS 可以很好的解决 HTTPS 降级攻击，但是对于 HSTS 生效前的首次 HTTP 请求，依然无法避免被劫持。浏览器厂商们为了解决这个问题，提出了 HSTS Preload List 方案：内置一份可以定期更新的列表，对于列表中的域名，即使用户之前没有访问过，也会使用 HTTPS 协议。

目前这个 Preload List 由 Google Chrome 维护，Chrome、Firefox、Safari、IE 11 和 Microsoft Edge 都在使用。如果要想把自己的域名加进这个列表，首先需要满足以下条件：

* 拥有合法的证书（如果使用 SHA-1 证书，过期时间必须早于 2016 年）；
* 将所有 HTTP 流量重定向到 HTTPS；
* 确保所有子域名都启用了 HTTPS；
* 输出 HSTS 响应头：
  + max-age 不能低于 18 周（10886400 秒）；
  + 必须指定 includeSubdomains 参数；
  + 必须指定 preload 参数；

即便满足了上述所有条件，也不一定能进入 HSTS Preload List，更多信息可以[看这里](https://hstspreload.appspot.com/)。通过 Chrome 的 chrome://net-internals/#hsts 工具，可以查询某个网站是否在 Preload List 之中，还可以手动把某个域名加到本机 Preload List。

对于 HSTS 以及 HSTS Preload List，我的建议是只要你不能确保永远提供 HTTPS 服务，就不要启用。因为一旦 HSTS 生效，你再想把网站重定向为 HTTP，之前的老用户会被无限重定向，唯一的办法是换新域名。

**CDN 安全**

对于大站来说，全站迁移到 HTTPS 后还是得用 CDN，只是必须选择支持 HTTPS 的 CDN 了。如果使用第三方 CDN，安全方面有一些需要考虑的地方。

**合理使用 SRI**

HTTPS 可以防止数据在传输中被篡改，合法的证书也可以起到验证服务器身份的作用，但是如果 CDN 服务器被入侵，导致静态文件在服务器上被篡改，HTTPS 也无能为力。

W3C 的 [SRI](http://www.w3.org/TR/SRI/)（Subresource Integrity）规范可以用来解决这个问题。SRI 通过在页面引用资源时指定资源的摘要签名，来实现让浏览器验证资源是否被篡改的目的。只要页面不被篡改，SRI 策略就是可靠的。

有关 SRI 的更多说明请看我之前写的《[Subresource Integrity 介绍](https://imququ.com/post/subresource-integrity.html)》。SRI 并不是 HTTPS 专用，但如果主页面被劫持，攻击者可以轻松去掉资源摘要，从而失去浏览器的 SRI 校验机制。

**了解 Keyless SSL**

另外一个问题是，在使用第三方 CDN 的 HTTPS 服务时，如果要使用自己的域名，需要把对应的证书私钥给第三方，这也是一件风险很高的事情。

CloudFlare 公司针对这种场景研发了 Keyless SSL 技术。你可以不把证书私钥给第三方，改为提供一台实时计算的 Key Server 即可。CDN 要用到私钥时，通过加密通道将必要的参数传给 Key Server，由 Key Server 算出结果并返回即可。整个过程中，私钥都保管在自己的 Key Server 之中，不会暴露给第三方。

CloudFlare 的这套机制已经开源，如需了解详情，可以查看他们官方博客的这篇文章：[Keyless SSL: The Nitty Gritty Technical Details](https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/)。

好了，本文先就写到这里，需要注意的是本文提到的 CSP、HSTS 以及 SRI 等策略都只有最新的浏览器才支持，详细的支持度可以去 [CanIUse](http://caniuse.com/) 查。切换到 HTTPS 之后，在性能优化上有很多新工作要做，这部分内容我在之前的博客中写过很多，这里不再重复，只说最重要的一点：既然都 HTTPS 了，赶紧上 HTTP/2 才是正道。

本文来自：https://imququ.com/post/sth-about-switch-to-https.html