Engineer

HTTPS

🔑 Mêmes requêtes, sur une connexion chiffrée

Ce que c’est

HTTPS est HTTP envoyé dans une connexion TLS. Requêtes et réponses restent les mêmes messages, avec les mêmes méthodes, codes de statut et en-têtes ; c’est la connexion qui les transporte qui devient chiffrée et authentifiée. Le schéma d’URI `https` est défini par le RFC 9110, spécification de la sémantique HTTP de juin 2022 qui a remplacé le RFC 2818, bref document de mai 2000 décrivant pour la première fois HTTP sur TLS. En l’absence de port explicite, HTTPS utilise le 443, contre le 80 pour HTTP en clair.

Le RFC 9110 associe deux règles à ce schéma. Le client DOIT vérifier que l’identité présentée par le serveur correspond de manière acceptable à l’hôte de l’adresse ; en pratique, le certificat doit couvrir ce nom. Une connexion au mauvais nom a échoué, quelle que soit la solidité de son chiffrement. Par ailleurs, `http://example.com/` et `https://example.com/` sont des origines distinctes avec des espaces de noms séparés. Pour le protocole, elles n’ont que le nom d’hôte en commun.

HTTPS n’est pas lié à une seule version de HTTP. HTTP/1.1 et HTTP/2 utilisent TLS sur TCP. HTTP/3 utilise QUIC, qui intègre la négociation TLS 1.3 au transport. Le RFC 9114 exclut de l’utiliser pour joindre directement un serveur `http://` en clair ; la version la plus récente arrive donc avec HTTPS ou pas du tout.

Une adresse montre ce que couvre le chiffrement. Pour `https://shop.example/orders/1234?ref=mail`, le chemin, la chaîne de requête, tous les en-têtes, y compris les cookies et `Authorization`, ainsi que les deux corps de message sont chiffrés. Le réseau entre visiteur et serveur voit encore l’adresse IP et le port du serveur, les instants et volumes des échanges et, dans la plupart des

connexions, le nom d'hôte `shop.example` : le navigateur l'a recherché dans DNS et le client le nomme dans le premier message TLS pour qu'une adresse puisse servir plusieurs sites. HTTPS masque ce qui a été demandé, pas à qui.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que HTTPS ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des comptes, mots de passe et droits attribués selon les besoins exacts de chaque poste, puis retirés le jour du départ.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 1 000 règles RBAC pour sécuriser Linux — 1,000 règles d'accès écrites pour combler une faille de sécurité

1 000 règles RBAC pour sécuriser Linux Contactez-nous →

Pourquoi c'est important

Pour un site public, la question actuelle est de savoir si quelque chose doit encore répondre en `http://` ; les navigateurs prennent désormais eux-mêmes la majeure partie de cette décision.

Le premier levier est le contexte sécurisé, terme du W3C désignant une page fournie par HTTPS ou depuis la machine elle-même. Les navigateurs réservent un ensemble croissant de fonctions à ces contextes. Les service workers en font partie : `navigator.serviceWorker` est déclaré pour les contextes sécurisés ; sur une page chargée par HTTP en clair, la propriété est donc absente, comme toutes les fonctions qui en dépendent, à commencer par le cache hors connexion. La spécification Secure Contexts recommande aussi aux auteurs de nouvelles spécifications de protéger ainsi les fonctions sensibles : la restriction devient le point de départ des nouveaux travaux, plutôt qu'une exception décidée au cas par cas.

Le second levier est le contenu mixte. Dès que la page est en HTTPS, le navigateur classe chacune de ses requêtes selon les règles Mixed Content du W3C. Les images, sons et vidéos demandés par `http://` passent automatiquement en `https://` et échouent si cette adresse ne répond pas. Scripts, feuilles de style et requêtes de données sont directement bloqués par une erreur réseau. Une seule adresse en clair oubliée dans un modèle peut donc arrêter une fonction entière, pas seulement une image.

Il reste ainsi peu de rôles au port 80. Un service à l'écoute qui répond à toute requête par une redirection permanente vers le même chemin en `https://` garde son utilité : les visiteurs saisissent des noms seuls et les anciens liens utilisent encore `http://`. Le défi HTTP de l'autorité de certification y arrive aussi lors de l'émission ou du renouvellement d'un certificat. HSTS permet ensuite au navigateur qui revient de sauter même la redirection. Tout autre usage du port 80 fournit une page sans les garanties précédentes.

La boucle locale reste la seule exception pour les sites ordinaires. La spécification considère `localhost`, `127.0.0.1` et `::1` comme potentiellement dignes de confiance : une page servie depuis la machine du développeur dispose des fonctions de contexte sécurisé sans certificat. Cette commodité de développement est à l'origine d'un des défauts ci-dessous.

À quoi cela ressemble en pratique

Le site de l'entreprise suit cette règle. Sur l'hôte de production, Caddy répond à toute requête web et renouvelle lui-même le certificat HTTPS, délivré par Let's Encrypt : sa validité relève du serveur web, pas d'une personne. Les mêmes pages sont publiées comme service onion Tor, dont l'adresse commence volontairement par `http://`. Une adresse onion encode la clé publique d'identité du service. Tor authentifie le service et chiffre la connexion avant tout échange HTTP ; c'est pourquoi ce miroir utilise HTTP sur Tor. La page des crédits renvoie aussi vers Qualys SSL Labs, qui évalue la configuration HTTPS en service pour toute personne qui le demande.

Le travail pour des clients montre l'autre face de la règle des contextes sécurisés. Pour une plateforme souvent utilisée par des gens de mer et du personnel de terrain sur téléphone, fréquemment hors connexion, l'entreprise a livré une prise en charge complète des applications web progressives, avec un service worker mettant en cache polices, images, sons, vidéos et ressources CDN selon une stratégie donnant priorité au cache. Les service workers ne fonctionnant réellement que sur HTTPS, le comportement hors connexion et l'installation ont été testés en HTTPS sur des appareils réels, au lieu d'être présumés corrects. Le même besoin apparaît dans l'outillage d'un monorepo polyglotte comptant 578 cibles de tâches : la couche go-task écrite par l'entreprise propose une variante HTTPS pour tester les PWA et les mobiles, à côté des modes de développement natif et Docker.

Où les problèmes surviennent

Les scripts s'arrêtent après le passage à HTTPS et certaines images disparaissent. Les pages contiennent encore des adresses `http://`, codées en dur dans un modèle, stockées dans le contenu du CMS ou copiées avec un extrait tiers. Le navigateur bloque scripts et feuilles de style et passe les images en HTTPS ; elles échouent là où l'autre hôte ne le propose pas. La console indique chaque adresse bloquée. Cherchez l'ancien schéma dans les modèles et le contenu stocké, puis réécrivez chaque occurrence en `https://` ou en chemin relatif au site :

```
grep -rnE '(src|href)="http://' templates/ content/
```

Un en-tête `Content-Security-Policy: upgrade-insecure-requests` permet aux pages de fonctionner pendant le nettoyage, mais laisse les adresses stockées inchangées.

Le site se redirige vers lui-même jusqu'à l'abandon du navigateur. Un CDN ou un répartiteur de charge termine TLS et transmet la requête à l'application en HTTP en clair. Celle-ci voit `http` et redirige vers `https` ; le proxy transmet encore la nouvelle tentative en `http`, jusqu'à l'erreur « too many redirects ». Cette même absence d'information fait produire des liens absolus commençant par `http://`. Le proxy doit transmettre le schéma d'origine dans l'en-tête standard `Forwarded: proto=https` du RFC 7239, ou l'ancien `X-Forwarded-Proto`. L'application ne devrait croire cet en-tête que s'il vient de l'adresse du proxy, car tout client peut l'envoyer. Confiez ensuite la redirection à une seule couche, généralement celle qui détient le certificat.

Un cookie de session défini en HTTPS circule en clair. Le RFC 9110 avertit que les cookies ne respectent pas sa séparation entre les deux origines. Selon les règles du RFC 6265, un cookie sans attribut `Secure` est aussi envoyé sur les requêtes `http://` au même hôte. Il suffit que le visiteur saisisse le nom seul, ou qu'un attaquant sur le réseau réponde à une requête en clair sans rapport par une redirection vers l'adresse `http://` du site. Définissez `Secure` sur chaque cookie émis et ajoutez HSTS pour que le navigateur n'effectue jamais la requête en clair.

Le service worker fonctionne sur le portable et manque sur le téléphone. Le développeur ouvre `http://localhost:3000`, contexte sécurisé, et tout fonctionne. Le téléphone du même réseau ouvre `http://192.168.1.20:3000`, qui n'en est pas

un : `navigator.serviceWorker` vaut undefined, une bibliothèque qui vérifie sa présence saute l'enregistrement et le mode hors connexion est absent sans avertissement. Servez aussi la version de développement en HTTPS. Caddy, par exemple, émet des certificats pour les noms et adresses IP locaux depuis sa propre autorité de certification locale et tente d'ajouter cette racine au magasin de confiance de la machine. Sa documentation avertit que ce n'est pas garanti et que l'installation dans chaque magasin relève de l'administrateur ; un téléphone de test doit recevoir cette racine à la main.

Pour aller plus loin

- RFC 9110, section 4.2.2 : schéma d'URI https — Schéma, port par défaut et obligation du client de vérifier l'identité du serveur.
- Contextes sécurisés — Pages considérées comme sécurisées, admissibilité de la boucle locale et conseil de restreindre les nouvelles fonctions.
- Contenu mixte — Requêtes d'une page HTTPS passées en HTTPS ou bloquées.
- RFC 7239 : extension HTTP Forwarded — En-tête standard permettant au proxy de transmettre le schéma employé par le visiteur.
- HTTPS automatique — Explication par Caddy de ses certificats, de sa redirection depuis HTTP et de son autorité de certification locale.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que HTTPS ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/https/