Engineer

HTTPS

🔑 ті самі запити через шифроване сполучення

Що це таке

HTTPS — це HTTP, переданий через TLS-з'єднання. Запити й відповіді лишаються тими самими повідомленнями, що й раніше, з тими самими методами, кодами стану й заголовками; змінюється з'єднання під ними, яке тепер зашифроване й автентифіковане. Схему URI `https` визначає RFC 9110, специфікація семантики HTTP від червня 2022 року, яка прийшла на зміну RFC 2818 — короткому документу з травня 2000 року, що вперше описав HTTP поверх TLS. Типовий порт — 443, тоді як звичайний HTTP використовує 80.

RFC 9110 пов'язує зі схемою два правила. Клієнт мусить перевірити, що ідентичність сервера прийнятно збігається з хостом в адресі. На практиці це означає, що сертифікат має покривати це ім'я, а з'єднання з неправильним ім'ям вважається невдалим, хоч яким сильним було б шифрування. І `http://example.com/` та `https://example.com/` — різні походження (origins) з окремими просторами імен. Для протоколу в них спільне ім'я хоста, і на цьому спільне закінчується.

HTTPS не прив'язаний до однієї версії HTTP. HTTP/1.1 і HTTP/2 працюють поверх TLS на TCP. HTTP/3 працює поверх QUIC, що несе рукостискання TLS 1.3 усередині самого транспорту, і RFC 9114 не дозволяє використовувати його для прямого доступу до звичайного `http://` -сервера, тож найновіша версія приходить разом із HTTPS або не приходить зовсім.

Що саме покриває шифрування, найлегше побачити на адресі. Для `https://shop.example/orders/1234?ref=mail` зашифровані шлях, рядок запиту, усі заголовки, зокрема cookies та `Authorization`, і тіла обох повідомлень. Мережа між відвідувачем і сервером усе одно бачить IP-адресу й порт сервера, час і обсяг трафіку, а в більшості з'єднань ще й ім'я хоста

`shop.example` : браузер шукав його в DNS, а клієнт називає його в першому повідомленні TLS, щоб одна адреса могла обслуговувати багато сайтів. HTTPS приховує, про що питали, а не кого питали.

Застосуйте поняття до рішення

Застосуйте «Що таке HTTPS?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Облікові записи, паролі й права налаштовані так, щоб доступ відповідав роботі, а звільнені втрачали його того ж дня.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Написала 1 000 правил RBAC для Linux-серверів — 1,000 правил доступу написано, щоб усунути вразливість

Написала 1 000 правил RBAC для Linux-серверів Зв'язатися з нами →

Чому це важливо

Відкрите питання для публічного сайту — чи має щось узагалі відповідати на `http://` , і більшу частину цього рішення браузери тепер ухвалюють самі.

Перший важіль — безпечний контекст (secure context), термін W3C для сторінки, доставленої через HTTPS або з самої машини. Браузери залишають дедалі більше функцій лише для безпечних контекстів. Service workers — одна з них: `navigator.serviceWorker` оголошено тільки для безпечних контекстів, тож на сторінці, завантаженій звичайним HTTP, цієї властивості просто немає, як і всього, що на ній побудовано, починаючи з офлайн-кешування. Специфікація Secure Contexts також радить авторам нових специфікацій так само захищати чутливі функції, тож обмеження стає типовим для нової роботи, а не винятком від випадку до випадку.

Другий важіль — змішаний вміст (mixed content). Коли сторінка сама працює через HTTPS, браузер розподіляє кожен її запит за правилами W3C щодо змішаного вмісту. Зображення, аудіо й відео, запитані через `http://` , автоматично підвищуються до `https://` і не завантажуються, якщо та адреса не відповідає. Скрипти, таблиці стилів і запити даних блокуються повністю з мережевою помилкою. Одна забута незашифрована адреса в шаблоні може вимкнути цілу функцію, а не лише картинку.

Тож порту 80 лишається зовсім мало. Слухач, що на кожен запит відповідає постійним переспрямуванням на той самий шлях за `https://` , досі виправдовує себе, бо люди вводять імена без схеми, а старі посилання досі ведуть на `http://` . Туди ж приходить HTTP-перевірка центру сертифікації,

коли сертифікат випускають або поновлюють. HSTS потім дозволяє браузеру, що повертається, пропустити навіть переспрямування. Усе інше на порту 80 — це сторінка, віддана без гарантій, описаних вище.

Loopback — єдиний сталий виняток для звичайних сайтів. Специфікація вважає `localhost`, `127.0.0.1` і `::1` потенційно надійними, тож сторінка з власної машини розробника отримує функції безпечного контексту без сертифіката. Ця зручність для розробників і є джерелом одного зі збоїв нижче.

Як це виглядає на практиці

Власний сайт компанії дотримується цього правила. На робочому хості Caddy відповідає на кожен вебзапит і сам поновлює HTTPS-сертифікат від Let's Encrypt, тож стежити, щоб сертифікат був чинним, — робота вебсервера, а не людини. Ті самі сторінки опубліковано як onion-сервіс Tor, і ця адреса навмисно починається з `http://`. Сторінка подяк пояснює чому: onion-адреса містить закодований відкритий ключ ідентифікації сервісу. Tor перевіряє справжність сервісу й шифрує з'єднання ще до обміну HTTP, тому це дзеркало використовує HTTP через Tor. Та сама сторінка посилається на Qualys SSL Labs, який оцінює живу HTTPS-конфігурацію для кожного, хто попросить.

Клієнтська робота показує правило безпечних контекстів з іншого боку. Для платформи, якою часто користуються моряки й польовий персонал із телефонів, нерідко без зв'язку, компанія зробила повну підтримку Progressive Web App із service worker, що кешує шрифти, зображення, аудіо, відео та ресурси CDN за принципом cache-first. Service workers поводяться як слід лише через HTTPS, тому офлайн-режим і встановлення перевіряли через HTTPS на реальних пристроях, а не брали на віру. Та сама потреба видна в інструментах монорепозиторію на кількох мовах програмування з 578 цілями task, де написаний компанією шар go-task ставить HTTPS-варіант для тестування PWA і мобільних пристроїв поруч із режимами розробки native і Docker.

Де виникають проблеми

Після переходу на HTTPS скрипти перестають працювати, а частина зображень зникає. На сторінках досі є адреси `http://`, жорстко прописані в шаблоні, збережені у вмісті CMS або вставлені разом зі стороннім

фрагментом коду. Браузер блокує скрипти й таблиці стилів і переводить запити зображень на HTTPS; зображення не завантажуються там, де інший хост не підтримує HTTPS. Консоль браузера називає кожну заблоковану адресу. Знайдіть стару схему в шаблонах і збереженому вмісті та перепишіть кожен збіг на `https://` або на шлях відносно сайту:

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

Заголовок `Content-Security-Policy: upgrade-insecure-requests` допомагає сторінкам пережити прибирання, але збережені адреси лишає такими, як були.

Сайт переспрямовує сам на себе, доки браузер не здається. CDN або балансувальник навантаження термінує TLS і передає запит застосунку звичайним HTTP. Застосунок бачить `http`, переспрямовує на `https`, проксі знову передає повторний запит як `http`, і цикл закінчується помилкою «too many redirects». Через ту саму сліпоту застосунок будує абсолютні посилання, що починаються з `http://`. Проксі має передавати початкову схему — у стандартному заголовку `Forwarded: proto=https` з RFC 7239 або в старішому `X-Forwarded-Proto`, — а застосунок має вірити цьому заголовку лише тоді, коли той приходить з адреси самого проксі, бо надіслати його може будь-який клієнт. Далі нехай переспрямуванням володіє один шар, зазвичай той, що тримає сертифікат.

Сесійний cookie, встановлений через HTTPS, передається відкритим текстом. RFC 9110 попереджає, що cookies не поважають межу, яку він проводить між двома походженнями (origins). За правилами cookies з RFC 6265 cookie без атрибута `Secure` надсилається й у незашифрованих запитах `http://` до того самого хоста. Достатньо, щоб відвідувач ввів ім'я без схеми, або щоб зловмисник у мережі відповів на будь-який інший незашифрований запит переспрямуванням на `http://`-адресу сайту. Ставте `Secure` на кожен cookie, який видає сайт, і додайте HSTS, щоб браузер узагалі не робив незашифрованого запиту.

Service worker працює на ноутбуці й відсутній на телефоні. Розробник відкриває `http://localhost:3000`, тобто безпечний контекст, і все працює. Телефон у тій самій мережі відкриває `http://192.168.1.20:3000`, а це вже не безпечний контекст, тож `navigator.serviceWorker` дорівнює `undefined`, бібліотека, що його перевіряє, пропускає реєстрацію, і офлайн-режиму тихо

немає. Віддавайте через HTTPS і збірку для розробки. Caddy, наприклад, випускає сертифікати для локальних імен та IP-адрес від власного локального центру сертифікації й намагається додати його корінь до сховища довіри машини. Його документація попереджає, що цей крок спрацьовує не гарантовано і що додати корінь до кожного сховища довіри — справа адміністратора, а тестовому телефону корінь доводиться встановлювати вручну.

Що почитати далі

- RFC 9110, розділ 4.2.2: https URI Scheme — схема, її типовий порт і правило, що клієнт мусить перевірити ідентичність сервера.
- Secure Contexts — які сторінки вважаються безпечними, чому до них належить loopback, і порада обмежувати нові функції.
- Mixed Content — які запити зі сторінки HTTPS підвищуються, а які блокуються.
- RFC 7239: Forwarded HTTP Extension — стандартний заголовок, яким проксі передає схему, використану відвідувачем.
- Automatic HTTPS — власний опис сертифікатів Caddy, його переспрямування з HTTP і локального центру сертифікації.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке HTTPS?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/https/