Engineer

HTTPS

🔑 同じ要求と応答を、暗号化した接続でやり取りするHTTP

どのようなものか

HTTPSは、TLS接続を通して送るHTTPです。要求と応答は、それまでと同じメッセージで、メソッド、状態コード、ヘッダーも変わりません。変わるのは、その下の接続が、暗号化され、認証されることです。`https` というURIスキームは、2022年6月のHTTPの意味に関する仕様RFC 9110で定義されています。HTTP over TLSを初めて記述した、2000年5月の短い文書RFC 2818から引き継ぎました。既定ポートは443で、平文HTTPは80を使います。

RFC 9110は、このスキームに二つの規則を付けます。クライアントは、サーバーが提示する識別情報が、アドレス内のホストと、許容できる形で一致すると確認しなければなりません。実際には、証明書がその名前を対象に含む必要があります。暗号がどれだけ強くても、名前が違えば接続は失敗です。また、`http://example.com/` と `https://example.com/` は、別々の名前空間を持つ、異なるオリジンです。プロトコル上は、ホスト名以外には何も共有しません。

HTTPSは、HTTPの一つの版に結び付いていません。HTTP/1.1とHTTP/2は、TCP上のTLSで動きます。HTTP/3は、TLS 1.3のハンドシェイクを転送方式の内部で運ぶQUIC上で動きます。RFC 9114は、平文の `http://` サーバーへ直接接続する用途を認めていません。そのため、最新のHTTPを使うにはHTTPSが必要です。

暗号化する範囲は、アドレスを見ると分かりやすくなります。`https://shop.example/orders/1234?ref=mail` では、パス、クエリー文字列、Cookieや `Authorization` を含む全ヘッダー、双方のメッセージ本文が暗号化されます。閲覧者とサーバーの間のネットワークからは、サーバーのIPアドレスとポート、通信のタイミングと大きさ、ほとんどの接続ではホスト名 `shop.example` も見えます。ブラウザーがDNSで調べ、クライアントがTLSの最初のメッセージでその名前を示すことで、一つのアドレスで多数のサイトを提供できるためです。HTTPSが隠すのは、何を求めたかであり、誰に求めたかではありません。

概念を判断に生かす

「HTTPSとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 アカウント、パスワード、権限を整理し、業務に必要なアクセスだけを付与し、退職者の権限は当日に解除します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 Linuxサーバーを守る1,000件のRBACルールを作成 — 1,000 セキュリティ上の穴を塞ぐアクセスルール

Linuxサーバーを守る1,000件のRBACルールを作成 お問い合わせ →

なぜ重要なのか

公開サイトで今問うべきなのは、なお `http://` で応答するものを残すべきかです。今では、ブラウザーが、その判断の大半を自ら行います。

第一の仕組みは、安全なコンテキストです。W3Cの用語で、HTTPSか、そのマシン自身から配信するページを指します。ブラウザーが安全なコンテキストだけに認める機能は、増え続けています。Service Workerも、その一つです。`navigator.serviceWorker` は、安全なコンテキスト用として宣言されているため、平文HTTPで読み込むページには、このプロパティー自体がなく、オフラインキャッシュをはじめ、その上の機能もありません。Secure Contexts仕様は、新しい仕様の著者にも、機微な機能を同様に制限することを勧めます。そのため、この制限は、個別の例外ではなく、新しい開発の既定方針です。

第二は、混在コンテンツです。ページ自身がHTTPSになると、ブラウザーは、各取得要求をW3CのMixed Content規則で分類します。 `http://` で求める画像、音声、動画は、自動的に `https://` へ切り替え、そのアドレスが応答しなければ失敗します。スクリプト、スタイルシート、データの要求は、ネットワークエラーとして無条件に遮断します。そのため、テンプレートに平文アドレスが一つ残るだけで、画像一枚だけでなく、機能全体が止まり得ます。

ポート80に残る用途は、ほとんどありません。すべての要求へ、 `https://` の同じパスへの恒久的転送を返す待ち受けには、まだ役割があります。人は名前だけを入力し、古いリンクには `http://` が残るためです。証明書の発行や更新時には、認証局のHTTPチャレンジも、そこへ届きます。HSTSを使えば、再訪するブラウザーは、その転送さえ省けます。それ以外にポート80で提供するページには、以上の保証がありません。

普通のサイトで常に認められる例外は、ループバックです。仕様は、`localhost` 、 `127.0.0.1` 、 `::1` を、潜在的に信頼できるものと数えます。そのため、開発者自身のマシンから提供するページでは、証明書なしで安全なコン

テキストの機能を使えます。この開発上の便利さが、後述する問題の一つを生みます。

実際の活用例

会社自身のサイトも、この規則に従います。本番ホストでは、Caddyがすべてのウェブ要求へ応答し、HTTPS証明書を自ら更新します。Let's Encryptの証明書を使い、有効に保つ仕事は、人ではなくウェブサーバーが担います。同じページをTorのonionサービスでも公開し、そちらのアドレスが `http://` で始まるのは、意図的です。onionアドレスには、サービスの識別用公開鍵が符号化されています。HTTPをやり取りする前に、Torがサービスを認証し、接続を暗号化します。このミラーでHTTP over Torを使うのは、そのためです。クレジットのページには、誰でも実際のHTTPS設定の評価を依頼できるQualys SSL Labsへのリンクもあります。

顧客向けの業務は、安全なコンテキストの規則を、反対側から示します。利用者の多くが、スマートフォンを使い、しばしばオフラインになる船員や現場担当者の基盤で、会社はProgressive Web Appへの全面対応を実装しました。Service Workerが、フォント、画像、音声、動画、CDNのアセットを、キャッシュを優先して取得する方式でキャッシュします。Service Workerが本来の動作をするのはHTTPSであるため、オフラインとインストールの動作は、推測で済ませず、HTTPSで実機検証しました。同じ必要性は、578のタスクターゲットを持つ複数言語のモノレポのツールにも現れます。会社が書いたgo-task層は、ネイティブとDockerの開発モードに並べて、PWAとモバイルを試すHTTPS版を用意します。

問題が起きるところ

HTTPSへ切り替えると、スクリプトが止まり、一部の画像が消えます。 テンプレートへの直書き、CMSのコンテンツ、外部の断片コードの貼り付けに、`http://` アドレスが残っています。ブラウザーはスクリプトやスタイルシートを遮断し、画像はHTTPSへ切り替えるため、相手のホストがHTTPSを提供しないところで壊れます。ブラウザーのコンソールには、遮断した各アドレスが出ます。テンプレートと保存済みのコンテンツで古いスキームを検索し、一つずつ `https://` か、サイト内の相対パスへ書き換えます。

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

`Content-Security-Policy: upgrade-insecure-requests` ヘッダーは、整理の間もページを動かしますが、保存したアドレスは元のままです。

ブラウザーが諦めるまで、サイトが自分自身へ転送します。 CDNやロードバランサーがTLSを終端し、要求を平文HTTPでアプリケーションへ渡します。アプリは `http` を見て `https` へ転送し、プロキシーが再試行を再び `http` で渡し、最後は「too many redirects」（転送回数が多すぎる）というエラーになります。同じ見落としで、アプリは `http://` で始まる絶対リンクも作ります。プロキシーは、RFC 7239の標準ヘッダー `Forwarded: proto=https` か、古い `X-Forwarded-Proto` で、元のスキームを渡す必要があります。どのクライアントでも送れるヘッダーなので、アプリが信じるのは、プロキシー自身のアドレスから届いた場合だけにします。その上で、通常は証明書を持つ層一つだけに、転送を任せます。

HTTPSで設定したセッションCookieが、平文で流れます。 RFC 9110は、Cookieが、二つのオリジンの間に引いた境界に従わないと警告します。RFC 6265のCookie規則では、 `Secure` 属性のないCookieを、同じホストへの平文 `http://` 要求でも送ります。訪問者が名前だけを入力しても送られますし、ネットワーク上の攻撃者が、無関係な平文要求へ、サイトの `http://` アドレスへの転送で応答しても送られます。サイトが発行するすべてのCookieへ `Secure` を付け、HSTSも加え、そもそも平文要求を出さないようにします。

ノートパソコンでは動くService Workerが、スマートフォンでは見つかりません。 開発者は、安全なコンテキストである `http://localhost:3000` を開き、すべて動きます。同じネットワークのスマートフォンが開く `http://192.168.1.20:3000` は、そうではありません。そのため、 `navigator.serviceWorker` の値はundefined（未定義）で、それを検査するライブラリーが登録を省き、オフラインモードが黙って欠けます。開発用ビルドもHTTPSで提供します。たとえばCaddyは、自身のローカル認証局から、ローカル名やIPアドレス用の証明書を発行し、そのルートをマシンの信頼ストアへ追加しようとします。文書は、その処理は保証されず、各信頼ストアへルートを入れるのは管理者の仕事と警告します。試験用のスマートフォンにも、そのルートを手で渡す必要があります。

参考資料

- RFC 9110、第4.2.2節：https URIスキーム — スキーム、既定ポート、クライアントによるサーバーの識別情報確認の規則です。
- 安全なコンテキスト（Secure Contexts） — 安全と数えるページ、ループバックが該当する理由、新機能の制限に関する助言です。
- 混在コンテンツ（Mixed Content） — HTTPSページの要求のうち、HTTPSへ切り替えるものと遮断するものです。

- RFC 7239：転送HTTP拡張 — 訪問者が使ったスキームを、プロキシーが渡すための標準ヘッダーです。
- 自動HTTPS — 証明書、HTTPからの転送、ローカル認証局についての、Caddy自身の説明です。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「HTTPSとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/https/