Engineer

Content Security Policy

ページで実行できるコードや、リソースの取得元・送信先をブラウザーへ指定する規則

どのようなものか

Content Security Policyは、ページが何を実行し、取得し、送信できるかについて、ウェブサイトからブラウザーへ渡す規則です。ブラウザーが、サイトに代わってその規則を適用します。ほぼ常にCSPと略します。サイトは、`Content-Security-Policy`応答ヘッダーで規則を送るか、ページの`meta`要素に書き込みます。その時点から、許可されていないスクリプト、スタイルシート、画像、フォント、フレーム、接続は、読み込み前に拒否します。

ポリシーは、セミコロンで区切るディレクティブを並べた、短い文字列です。各ディレクティブは、リソースまたは振る舞いの一種類と、受け入れる出所を定めます。厳格なポリシーは、次のようになります。読みやすいように一行に一つずつ示しますが、通信時には、一つのヘッダーの値です。

```
default-src 'self';
script-src 'nonce-q8Zt3VwR1mKx7NpL2cYfJA==' 'strict-dynamic';
object-src 'none';
base-uri 'none';
form-action 'self';
frame-ancestors 'none'
```

使う語彙は少数です。`'self'`は、ページ自身のオリジンを指します。`https://static.example.com`のようなホストや、`https:`のようなスキームを指定すると、許可する範囲が広がります。`'none'`は、何も許可しません。nonceは、サーバーが一つの応答のために生成する乱数値で、ポリシーと、その応答で信頼を保証する各スクリプト要素の両方に書きます。ハッシュは、インラインスクリプトの正確なテキストから計算するダイジェストです。そのスクリプトだけを実行し、それ以外は実行させません。`'unsafe-inline'`と`'unsafe-eval'`という二つのキーワードは、保護を再び無効にします。その名前自体が、十分な注意喚起です。

`default-src` は、ポリシーが省略した取得用ディレクティブの代わりになります。 `script-src` 、 `style-src` 、 `img-src` 、 `connect-src` などが対象です。代わりになるのは、取得用ディレクティブに限られます。取得ではなく振る舞いを制御する `base-uri` 、 `form-action` 、 `frame-ancestors` などは、明記しなければ無制限のままです。

監視だけを行うモードでも使えます。 `Content-Security-Policy-Report-Only` として送ると、何も遮断せず、代わりに違反の一件ごとに、 `report-to` で指定したエンドポイントへ報告します。ブラウザーのReporting APIを使う仕組みです。古い `report-uri` ディレクティブは非推奨で、代わりにこの仕組みが推奨されています。現在の規格本文は、Web Application Security Working GroupによるW3C作業草案、Content Security Policy Level 3です。

概念を判断に生かす

「Content Security Policy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

CSPは、クロスサイトスクリプティングに対抗するために作られました。コメント欄、HTMLに表示し直す検索語、侵害された依存先などを通して、誰かが自分の選んだテキストをページへ入れます。ブラウザーは、それを、サイト自身のコードと同じ権限を持つスクリプトとして実行します。ページを読み、ログインした利用者になり代わって操作し、見つけたものをどこへでも送れます。第一の防御は、出力を正しくエスケープすることです。すべてのテンプレートのすべての行で、毎回、正しい必要があります。CSPは第二の防御です。エスケープの誤りがすり抜けても、注入されたスクリプトは、ポリシーも通過しなければなりません。その応答のnonceを持つスクリプトだけを実行するポリシーなら、拒否できます。

実行を認めたスクリプトの到達先も制限します。 `connect-src` は、スクリプトが通信できるホストを決め、窃取用スクリプトが収集した情報を送れる先を絞ります。 `frame-ancestors` は、どのサイトがページをフレーム内に表示できるかを決めます。別のページの上にページを重ね、利用者の意図しない場所へクリックを

届ける、クリックジャッキングへの標準的な対策です。`form-action` は、フォームが送信できる先を決めます。`base-uri` は、注入された `base` 要素が、ページ内のすべての相対アドレスを、気付かれないまま別の先へ向けるのを防ぎます。

CSPが誰を守るかも、正確に捉える必要があります。ブラウザー内で動くため、ページを読む人を、攻撃者が仕込んだコンテンツから守ります。一方、その同じ人からサーバーを守ることはありません。自分のブラウザーなら、数秒でヘッダーを編集したり、取り除いたりできるためです。ポリシーは、ページを信用する範囲を狭める理由にはなりますが、リクエストをより信用する理由にはなりません。

主なコストは、使っているものの洗い出しです。ポリシーを設けるには、ページがリソースを読み込む元となるすべてのオリジン、すべてのインラインスクリプト、属性として書いたすべてのイベントハンドラーを、チームが把握する必要があります。新しいサイトなら、最初のコミットから厳格なポリシーに合わせて作るコストは、わずかです。インラインハンドラー、タグマネージャー、複数のウィジェットを持つ古いサイトでは、先に整理のコストを払います。通常、本当のプロジェクトになるのは、その整理です。

実際の活用例

このサイトも、ポリシーを適用しています。セキュリティページには、ページ自体に宣言する厳格なContent Security Policyを記載しています。フォームをどこにも送信しないため、`form-action 'none'` を使います。すべてのHTMLページが、`script-src 'none'` を明記します。実行可能なJavaScriptも、サービスワーカーも動きません。第三者へのリクエストはなく、ページに必要なものは、すべて自身のホストから届くため、許可するものはごくわずかです。

ログインする利用者を持つアプリケーションには、短い許可リスト以上のものが必要です。職業や組織に関するデータを扱うプラットフォームでは、HSTSやSameSite Cookieとともに、Next.jsのプロキシミドルウェアが、リクエストごとのnonceと `strict-dynamic` を使うContent-Security-Policyを設定します。応答ごとに新しいnonceを使えば、その応答でサーバーが印を付けたスクリプト要素だけを実行します。ポリシーに `strict-dynamic` があれば、nonceによって信頼するスクリプトは、さらに別のスクリプトを読み込めます。そのため、バンドルが分割されたコードを取得するすべてのホストを、ポリシーで列挙する必要はありません。このプラットフォームは、クライアントが悪意を持つという前提で動きます。そのため、各リクエストの権限はサーバーで改めて確認し、フロントエンドの制御は、利用体験のためだけに使います。CSPは、注入されたコードから利用者を守る、複数ある層の一つです。認可は、別の場所で決めます。

問題が起きるところ

ポリシーを有効にすると、サイトの半分が動かなくなります。 厳格なポリシーをいきなり強制すると、誰も覚えていなかったインラインスクリプトや、誰かが二年前に追加した追跡タグを遮断します。まず、`Content-Security-Policy-Report-Only` として配信します。報告が落ち着くまで読み、挙がった出所を一つずつ許可するか取り除き、その後で強制します。報告専用ヘッダーは、実際のHTTPヘッダーでなければなりません。仕様は、`meta` 要素での指定を認めていません。

許可リストが、ホスト全体を信用します。 `script-src https://cdn.example.com` は、そのホストのすべてのファイルを許可します。古いバージョンのライブラリーや、呼び出し元の入力をスクリプトとして返すエンドポイントも含みます。仕様自体が、クロスサイトスクリプティングに有効なポリシーの導入は難しいと述べ、Strict CSPと呼ぶ構成を推奨しています。nonceかハッシュと `'strict-dynamic'` で `script-src` を構成し、`base-uri` を `'self'` か `'none'` にします。

`'unsafe-inline'` が、すべてのインラインスクリプトを許すと思ってしまいます。 同じソースリストにnonceかハッシュが現れると、それらを理解するブラウザーは、`'unsafe-inline'` を無視します。この規則によって、nonceを知らない古いブラウザー向けには、代替として `'unsafe-inline'` を残し、現在のブラウザーには、厳格な一覧を適用できます。一つのスクリプトへnonceを加えた途端、ほかのインラインスクリプトが遮断されるため、チームを驚かせる場合があります。

nonceが、秘密ではなくなります。 仕様は、ポリシーを送るたびに新しい値を使うことを必須とし、暗号学的に安全な生成器による128ビット以上の値を推奨しています。テンプレートへ固定値を書き込んだり、nonceを含むページ全体を共有キャッシュへ保存し、全員に配信したりすると、攻撃者に、自分のスクリプトへ付けるための値を渡してしまいます。

`meta` 要素内のポリシーでは、同じポリシーをヘッダーで送る場合より、できることが減ります。 この方法で届く `frame-ancestors`、`report-uri`、`sandbox` は、ブラウザーが無視します。文書内でその要素より後だけにポリシーを適用するため、head内でそれより上にあるものは、検査されずに読み込まれます。応答ヘッダーを変更できないサイトでも、headの可能な限り早い位置へ `meta` ポリシーを置けば、実際に保護できます。ただし、フレームへの埋め込みを防ぐには、ヘッダーが必要です。

二つ目のポリシーは、緩和せず、制限を強めます。 一つの応答には、フレームワーク、プロキシ、CDNからの複数のポリシーを載せられます。リソースは、そのすべてを通過する必要があります。ポリシーを追加してできるのは、制限を強

めることだけです。遮断されたリクエストを直すため、後段で緩いヘッダーを追加しても、効果はありません。遮断した側のポリシーを直す必要があります。

参考資料

- Content Security Policy Level 3 — 規格そのものです。Strict CSPの推奨と、nonceが盗まれる仕組みを説明する節を含みます。
- HTML標準：meta要素 — ページ内で宣言するポリシーを適用するプラグマ、無視するディレクティブ、それより前に取得したリソースが保護をすり抜ける理由を説明します。
- Reporting API — 強制する前に違反を調べるための、`report-to` の背後にあるエンドポイントの仕組みです。
- Trusted Types — `require-trusted-types-for` ディレクティブで有効にする関連規格です。DOMを通して注入されるスクリプトを対象にします。

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

「Content Security Policy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/csp/