Engineer

Subresource Integrity

ブラウザーがファイルの内容を照合するためのハッシュ値

どのようなものか

Subresource Integrityは、スクリプトやスタイルシートに含まれるべきバイト列をページが事前に指定し、届いた内容が異なればブラウザーにファイルを破棄させる仕組みです。通常はSRIと略します。ページは、ファイルの暗号学的ダイジェストを `integrity` 属性に書きます。この属性を付けるのは、ファイルを読み込む `script` 要素や `link` 要素です。応答が届くと、ブラウザーは受け取ったバイト列に同じハッシュ関数を適用し、その結果を要素上のダイジェストと比較します。一致すれば、ファイルは通常どおり実行されるか、ページの装飾に使われます。異なれば、ブラウザーは応答をネットワークエラーとして扱い、スクリプトを実行せず、スタイルシートも適用しません。

値は、ハッシュ関数名、ハイフン、base64形式のダイジェストをつないだものです。完成したファイルに一つのコマンドを実行すれば作れます。二度実行してみると、わずかな変更で結果が変わることが分かります。

```
$ printf "alert('Hello, world.');" > hello.js
$ openssl dgst -sha384 -binary hello.js | openssl base64 -A
H8BRh8j48O9oYatfu5AZzq6A9RINhZO5H16dQZngK7T62em8MUt1FLm52t+eX6xO
$ echo >> hello.js
$ openssl dgst -sha384 -binary hello.js | openssl base64 -A
qu8P7hCLFcBpas8BOnGmaANCkfXJIpFttDEaxOj0Pq9H6Q8NYjsMb+PVqwuJygVe
```

最初の出力の前に `sha384-` を付けると、その一行のスクリプトに対する属性値になります。末尾に改行を一つ加えただけで、最初とはまったく違うダイジェストになり、最初の値を持つブラウザーは、二つ目のファイルを拒否します。

仕様が認めるハッシュ関数は、いずれもSHA-2系のSHA-256、SHA-384、SHA-512の三つです。執筆時点では、SHA-384を基準として適切な選択肢としています。一つの属性に、空白で区切った複数のダイジェストを並べることもできま

す。ブラウザーは、その中で最も強い関数を選び、その関数で作られたいずれかのダイジェストに一致すれば、ファイルを受け入れます。そのため、ページ上に誤ったダイジェストしかない期間を作らずに、より強いアルゴリズムへ移行したり、切り替え中にライブラリーの二つのリリースを受け入れたりできます。

属性は `script` と `link` で使え、ファイルを先読みする `Link` 応答ヘッダーにも同じ値を載せられます。仕様の現行文書は第2版で、Web Application Security Working GroupによるW3C作業草案です。第1版は2016年にW3C勧告になりました。

概念を判断に生かす

「Subresource Integrity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

コンテンツ配信ネットワーク、またはサイトの運営者が管理していないホストからライブラリーを読み込むページは、そのホストにページ上でコードを実行する権限を渡します。HTTPSは、ネットワーク上の転送中のファイルを保護しますが、ホストがそもそも何を送ることにしたかについては保証しません。CDNの保存領域が侵害されたり、公開用のアカウントが乗っ取られたり、見慣れたアドレスのファイルがひそかに差し替えられたりすると、そのファイルを読み込むすべてのページが、ページの全権限で新しいコードを実行します。ページ自身のコードには、その違いを見分ける手段がありません。

SRIは、その判断をページ側へ戻します。サイトの運営者が、自分で読んだ、またはビルドしたファイルからダイジェストを計算し、それをサイト自身のサーバーから送るページ内に含めます。ホストが別の内容を返しても、そのファイルは実行されずに拒否されます。誰かが確認するまでページの機能は失われますが、原因が明らかな不具合として表面化し、他人のコードをひそかに実行することを避けられます。

ダイジェストは、バージョンも固定します。 `/lib/latest.js` のようなアドレスは、ページ側が知らない間に、いつでも新しいリリースを返すようになりますが、ダイジェストはそれに追従しません。運営者が選んでいない更新は、特定の

要素の読み込み拒否になり、エラーイベントで報告されます。

SRIが行わないことも、正確に押さえる必要があります。ファイルをダイジェストと照合するだけで、安全なファイルかどうかを判断するわけではありません。すでに悪意ある一行を含むライブラリーのダイジェストは、その一行も忠実に保護します。インラインスクリプトは対象外で、それはContent Security Policyの役割です。また、ダイジェストを載せたページを信頼するため、ページ自体も改ざんされずに届く必要があります。

実際の活用例

このサイトは、自身のスタイルシートにSRIを使っています。セキュリティーページにも、スタイルシートへのSubresource Integrityの適用を記載しています。サイトのビルド時にバンドルのフィンガープリントとハッシュを生成するため、読者のブラウザーは、内容が変わらず届いたことを検証できます。実行可能なJavaScriptは公開せず、第三者へのリクエストもありません。ブラウザーのネットワークパネルでホストを数えると、サイト自身の一つだけです。

クレジットページは、同じファイルをビルド側の視点で説明しています。ビルド時に連結してフィンガープリントを付けた一つのCSSバンドルを、SRIのハッシュとともに配信します。フィンガープリントは、ファイル内容のハッシュを名前に含めるものです。変更したスタイルシートには新しいアドレスが付き、ブラウザーが古いキャッシュを現行版と取り違えることを防ぎます。一方、integrityの値によって、そのアドレスの先にあるバイト列をブラウザーが検証できます。ビルドの同じ段階で同じバイト列から両方を生成するため、どのリリースでもスタイルシートとダイジェストが一緒に配信されます。その結果、次の節で説明する最も一般的な失敗、つまりハッシュ計算後のファイル変更が入り込む余地は、非常に小さくなります。

問題が起きるところ

ビルドとブラウザーの間でファイルが書き換わる。 スクリプトを縮小するCDN、応答の内容に何かを挿入するプロキシ、ダイジェストの計算後にファイルを書き換えるデプロイ処理は、内容のバイト列を変えます。攻撃がなくても、ブラウザーは結果を拒否します。内容を変えないgzipやBrotliでの再圧縮では、この不一致は起こりません。検証の対象は、ネットワーク上で送られた圧縮済みのバイト列ではなく、デコードされた応答本文だからです。仕様は、内容を変える中継者に義務を課しています。配信する内容に合わせてダイジェストも更新するか、元の内容を変えずに配信しなければなりません。一方、配信元には、ファイ

ルを変更しないよう求める `Cache-Control: no-transform` を送ることが推奨されています。確実な手順は、ブラウザーが検証するデコード後の内容とまったく同じバイト列から、最後にダイジェストを計算することです。

別オリジンのファイルが、CORSなしで届く。 仕様は、別のオリジンのファイルの完全性を検証するにはCORSを使うことを要求し、使わずに試みるのは論理上の誤りだとしています。例えば、要素に `crossorigin="anonymous"` を指定し、ホストはファイルの共有を許すヘッダーで応答する必要があります。`crossorigin="use-credentials"` もCORSを使いますが、サーバーのヘッダーで、要求元のオリジンと認証情報の利用を明示的に許可します。CORSリクエストと、それに対応するサーバー側のポリシーがなければ、ブラウザーは検証せずに読み込むのではなく、ファイルをブロックします。この規則は、ほかのサイトを守ります。規則がなければ、攻撃者は推測したダイジェストで非公開の応答を読み込もうとし、その成否から、例えば読者がそのサイトにログインしているかどうかを知ることができるためです。

ダイジェストが平文のHTTPで届く。 SRIは、ファイル配信ホストの侵害からページを守りますが、ネットワーク経路上の攻撃者からは守りません。HTTPSを使わずに配信するページでは、転送中のファイルを書き換えられる者は、ページ内のダイジェストも書き換えたり削除したりできます。そのため、仕様は、完全性メタデータをセキュアコンテキストだけに配信するよう推奨しています。

ブラウザーが無視するハッシュ関数を使う。 仕様が認めるのはSHA-256、SHA-384、SHA-512だけであり、MD5やSHA-1のように弱いと分かっている関数は、ブラウザーが拒否するよう推奨しています。意外なのは、一覧にないトークンの扱いです。ブラウザーはそれを無視し、無視されるトークンだけの属性は、メタデータがまったくないものとして扱われるため、ファイルは検証されずに読み込まれます。Hugoのように、MD5でフィンガープリントを作れるビルドツールでは、誤ってその状態を配信しやすくなります。

拒否されたファイルとともに、ページも動かなくなる。 検証が失敗すると、ブラウザーは要素上でエラーイベントを発生させ、スクリプトやスタイルシートは単に欠けた状態になります。ページの動作に欠かせないライブラリーなら、仕様は、イベントを捕捉し、遅くても信頼できる配信元、例えばサイト自身のサーバーから代替コピーを読み込むことを提案しています。代替がなければ、防御は設計どおりに働いていても、読者には壊れたページが届きます。

ダイジェストのある要素だけが検証される。 一つのスクリプトにintegrity属性があっても、後から誰かが属性なしで追加した次のスクリプトについては何も保証しません。integrity属性のあるスクリプトだけをレビューしても、属性を欠くス

クリプトは見つかりません。草案は、これをサーバー側で扱う応答ヘッダーを加えています。`Integrity-Policy` に `blocked-destinations=(script)` を指定すると、完全性メタデータなしで届く外部スクリプトをすべてブロックします。`Integrity-Policy-Report-Only` は、先にその一覧を報告し、ブロックする前に修正できるようにします。

参考資料

- Subresource Integrity — 検証アルゴリズム、`Integrity-Policy` ヘッダー、プロキシ、平文HTTP、クロスオリジンの情報漏えいに関する注記を含む、現行の草案です。
- Subresource Integrity、W3C勧告2016年版 — 草案の基礎となった、確定した勧告である第1版です。
- Fetch Standard — リクエストの完全性メタデータと応答本文を照合する処理、および別オリジンの検証で必要なCORSプロトコルを定めています。
- Hugo：resources.Fingerprint — 静的サイトジェネレーターがビルド時にファイルのフィンガープリントを作り、対応するintegrity値を返す仕組みです。
- FIPS 180-4、Secure Hash Standard — SHA-256、SHA-384、SHA-512を定義する連邦標準です。

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

「Subresource Integrity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/sri/