

Résumé des attaques CSRF

Définition

Une **CSRF** (*Cross-Site Request Forgery*, ou falsification de requête intersites) consiste à tromper le navigateur d'une personne déjà connectée pour qu'il envoie une requête non désirée à un site.

Le navigateur peut joindre automatiquement les cookies de session à cette requête. Le site risque alors de la traiter comme si la personne l'avait volontairement effectuée.

Exemple simplifié

Imaginons qu'un site permette de changer l'adresse e-mail avec une requête :

```
POST /modifier-email  
email=nouvelle-adresse@example.test
```

Si le site ne vérifie pas que la requête est légitime, une page malveillante pourrait inciter le navigateur d'une personne connectée à envoyer une requête de ce type. Le navigateur peut y joindre automatiquement le cookie de session.

Conditions favorables à une attaque

- La personne est connectée au site ciblé.
- Le navigateur possède un cookie ou autre identifiant de session.
- Le navigateur joint automatiquement cet identifiant à la requête.
- Le site n'effectue pas de vérification suffisante de l'origine ou de la légitimité de la requête.

Risques

Selon les fonctions du site, une CSRF peut entraîner une modification de paramètres, la publication d'un contenu, ou une autre action effectuée avec les droits de la personne connectée.

Prévention

- Utiliser un **jeton CSRF** imprévisible, associé à la session, et le vérifier côté serveur pour les opérations sensibles.
- Configurer les cookies de session avec l'attribut SameSite — généralement Lax ou Strict, selon les besoins de l'application. C'est une protection complémentaire, pas toujours suffisante à elle seule.
- Vérifier, lorsque c'est pertinent, les en-têtes Origin et Referer côté serveur.
- Ne jamais modifier l'état du compte avec une requête GET : réserver les changements aux méthodes appropriées, comme POST, PUT, PATCH ou DELETE.
- Pour les interfaces JavaScript, exiger un jeton CSRF ou un en-tête personnalisé vérifié côté serveur.
- Utiliser les protections CSRF intégrées au framework plutôt que de réinventer le mécanisme.

Exemple de formulaire avec jeton

Le serveur ajoute au formulaire un jeton imprévisible associé à la session :

```
<form method="post" action="/modifier-email">
  <input type="hidden" name="csrf_token" value="JETON_GENERE_PAR_LE_SERVEUR">
  <input type="email" name="email">
  <button type="submit">Modifier</button>
</form>
```

Lors de la réception, le serveur compare le jeton envoyé à celui attendu pour la session. **Un jeton réel doit être généré et vérifié par le serveur.**

Erreurs à éviter

- Croire que SameSite remplace toujours un jeton CSRF.
- Utiliser uniquement un champ caché sans vérifier sa valeur côté serveur.
- Permettre à une requête GET d'effectuer une action qui modifie des données.
- Supposer que CORS protège à lui seul contre toutes les CSRF.
- Oublier de protéger les requêtes asynchrones ou les points d'accès API utilisant des cookies.

À retenir

Une CSRF exploite le fait que le navigateur peut joindre automatiquement les informations de session à une requête. Pour s'en protéger, vérifiez côté serveur que les requêtes sensibles sont légitimes, notamment avec des **jetons CSRF**, et utilisez SameSite comme défense complémentaire.

From:

<http://slamwiki2.kobject.net/> - SlamWiki 2.1

Permanent link:

<http://slamwiki2.kobject.net/sio/bloc3/attaques/csrf>

Last update: **2026/10/12 07:56**

