Cross-Site Request Forgery

Description

Le {Cross-Site Request Forgery} (CSRF ou XSRF) est une technique d'attaque Web qui force un utilisateur authentifié sur une application vulnérable à réaliser une action en abusant de la confiance qu'a l'application envers les requêtes de ses clients.

L'action forcée dépend du contexte, mais peut être le changement de certains paramètres du compte, le virement bancaire vers un compte arbitraire ou encore l'attribution de nouveaux privilèges à un utilisateur de l'application. Étant donné que la cible est un utilisateur et non l'application en elle-même, cette attaque appartient à la famille des vulnérabilités dites “côté client”.

Ces attaques existent car les navigateurs envoient automatiquement les cookies associés aux sessions authentifiées, sans vérifier l'origine de la requête. Pour la réaliser, un utilisateur malveillant va héberger, au travers d'une page Web, un formulaire HTML permettant d'effectuer la requête réalisant l'action désirée. Ce formulaire est accompagné d'un script JavaScript qui va soumettre le formulaire dès que la page est chargée.

```
<h1>Page hébergée sur hackerman.hack pour CSRF sur example.ex</h1>
<form action="https://example.ex/edit-account" method="post">
  <input type="password" name="password" value="easypass123">
  <input type="password" name="confirm-password" value="easypass123">
  <button type="submit">Edit</button>
</form>
<script>
    document.forms[0].submit();
</script>
```

Exemple de charge utilisée pour forcer le changement de mot de passe sur **exemple.ex**.

Si l'utilisateur qui consulte la page malveillante est authentifié sur l'application ciblée, la requête contiendra son cookie de session. La requête est ensuite reçue par le serveur qui ne fait pas de distinction entre une requête légitime et malicieuse. Enfin, l'action désirée par l'attaquant est effectuée.

L'attaque nécessite que la cible se dirige sur la page hébergée par l'attaquant. En général, c'est au travers des mails de phishing que le CSRF est réalisé. Cependant, d'autres moyens peuvent être utilisés pour diffuser l'attaque, notamment via un service de tickets interne ou encore via une messagerie compromise d'un des employés de l'entreprise ciblée.

Aujourd'hui, plusieurs protections existent pour se protéger des attaques par CSRF. L'attaque étant peu connue des développeurs, celle-ci reste donc très présente dans les applications actuelles.

Pré-requis d’exploitation

Pour mettre en œuvre cette attaque, il est nécessaire d'avoir accès au minimum à un compte de l'application à tester. Ce compte doit avoir les privilèges nécessaires pour effectuer l'action vulnérable aux attaques CSRF.

Compétences nécessaires

- Connaissances de base du protocole HTTP ;
- Notions sur les langages JavaScript et HTML.

Outils nécessaires

- Accès à une console de navigateur ou à un outil de modification et/ou d’interception de requêtes comme **Burp Suite**.

Flux d’exécution

Explorer

La détection d'un CSRF est assez simple, il suffit de parcourir les pages de l'application web avec un compte utilisateur légitime (idéalement deux pour effectuer des tests) à la recherche de fonctionnalités intéressantes (transferts d’argent, changement de mot de passe, etc.) en accord avec les points suivants :

- **Une action exploitable :** l'action réalisée doit être sensible. Un CSRF sur un changement de filtre de recherche n'aura à priori aucune répercussion utile pour l'attaquant, tandis qu'un changement d'adresse email est bien plus intéressant d'un point de vue offensif ;
- **L'authentification est basée uniquement sur un cookie :** l'attaque par CSRF repose sur le fait de réaliser une requête HTTP sous le nom de la victime. Si l'application utilise d'autres facteurs de vérification pour effectuer l'action, il est probable que celle-ci ne passe pas. Par exemple pour un virement bancaire vulnérable qui demanderait à l'utilisateur de confirmer celui-ci sur l'application de son téléphone, rendant ainsi l'attaque impossible ;
- **Le formulaire ne contient que des paramètres prédictibles :** le formulaire ne doit pas envoyer de jeton particulier ou d'informations imprédictibles. Une valeur imprédictible pour l'attaquant dans le formulaire rendrait l'attaque caduque, car l'application n'accepterait pas d'effectuer l'action, un des champs étant erroné. Par exemple, un changement de mot de passe qui demanderait l'ancien mot de passe pour réaliser le changement.

Expérimenter

Si l'on trouve une fonctionnalité qui respecte les trois points de l'étape précédente, il est relativement probable qu'un CSRF soit possible. Pour s'en assurer totalement, il est nécessaire de le tester. Pour cela, le mieux est d'utiliser deux comptes obtenus légitimement. L'un servira à mettre en place l'attaque, l'autre à tester :

- Créer le code HTML / JavaScript qui permet d'effectuer automatiquement l'action ciblée avec des valeurs prédéfinies dans les champs.
- Mettre en place le code sur un serveur accessible par la victime. En CTF, on peut utiliser des services Web publics tels que Beeceptor, Pipedream ou encore ngrok.
- Avec le compte de la victime authentifiée sur l'application, visiter la page qui contient la charge d'attaque.
- Si après avoir navigué sur la page, l'action est réalisée sans avoir eu d'interaction supplémentaire, l'attaque est possible.

> **Pro-tips** : pour utiliser deux comptes authentifiés sur la même application, vous pouvez utiliser la navigation privée, ou les containers tabs. Sinon, il est possible d'utiliser l'extension Pwnfox disponible seulement pour le navigateur Firefox.

Exploiter

Conséquences potentielles

Les conséquences potentielles de ce type d'attaque dépendent grandement de la nature de l'application ciblée. Par exemple, dans le cas d'une application bancaire vulnérable, il peut être possible :

- D'usurper un compte utilisateur (par exemple en exploitant un formulaire de changement de mot de passe ou d'email) ;
- De modifier les informations de compte (telles que les adresses de contact ou encore les informations personnelles) ;
- De déclencher des transactions financières (comme des achats ou des paiements) sans le consentement de l'utilisateur ;
- D'ajouter des bénéficiaires ou des comptes tiers pour les transactions financières.

Contres-mesures

Les contre-mesures suivantes peuvent être mises en œuvre :

- **Utiliser des jetons cryptographiques pour associer une demande à une action spécifique.** Le jeton peut être régénéré à chaque demande, de sorte que si une demande avec un jeton invalide est reçue par le serveur, elle peut être écartée de manière fiable. Le jeton est considéré comme invalide s'il est arrivé avec une demande autre que l'action à laquelle il était censé être associé.
- **Utiliser un second facteur d'authentification.** L'utilisateur peut être invité à confirmer une action chaque fois qu'une action concernant des données potentiellement sensibles est invoquée. De cette façon, même si l'attaquant parvient à faire en sorte que l'utilisateur clique sur un lien malveillant et demande l'action souhaitée, l'utilisateur peut encore refuser la confirmation et ainsi être averti que son compte a potentiellement été compromis.
- **Mettre en place l'option 'Same-Site' sur les cookies de session.** L'option **Same-Site** peut être précisée lors de l'attribution d'un cookie par le serveur. Cet attribut permet de réduire le domaine de diffusion du cookie associé selon trois niveaux de politiques : **None**, **Lax** et **Strict**.

Comment cela fonctionne ?

Exemple 1

Le formulaire HTML ci-dessous permet à un utilisateur légitime de modifier l'email avec lequel son compte est associé :

```
<form action="/account/edit-email" method="post">
    <input type="email" name="email">
    <button type="submit">Change Email</button>
</form>
```

La requête envoyée via ce formulaire se présente comme ceci :

```
POST /account/edit-email HTTP/2
Host: example.ex
Cookie: session=JV1WLK4mrRS3o4zX64NOS2fmADrFKIgo

email=new-email@root-me.org
```

À la réception de cette requête, le serveur modifie l'email du compte sans plus de vérifications. Ce formulaire respecte bien les 3 prérequis présentés dans le point Explorer. Pour mettre en place l'attaque, on ré-utilise le formulaire HTML légitime que l'on modifie légèrement pour l'héberger sur un serveur que l'on contrôle :

```
<form id="malicious" action="https://example.ex/account/edit-email" method="post">
    <input type="email" name="email" value="hackerman@root-me.org" />
    <button type="submit">Change Email</button>
</form>
<script>
    document.getElementById('malicious').submit();
</script>
```

> **Attention** : le formulaire d'exploitation est hébergé sur un serveur dont le nom de domaine est différent de celui à destination de la requête. Il est important de bien préciser le domaine de destination dans le champ **action** de la balise **form**. Sinon, la requête sera dirigée vers
>
> `/account/edit-email`
>
> du domaine appartenant à l'attaquant.

Le JavaScript permet d'automatiquement soumettre le formulaire dès que la page est chargée. Si la personne visitant la page est un utilisateur de l'application

`example.ex`

, la requête envoyée contiendra son cookie de session.

Notre attaque est prête à être exploitée. On transmet à la victime l'URL contenant la charge ci-dessus et on patiente jusqu'à ce que celle-ci visite la page. Une fois fait, il ne reste alors qu'à utiliser la fonctionnalité d'oubli de mot de passe en précisant le mail tout juste changé

`hackerman@root-me.org`

pour compromettre totalement le compte de la victime.

Exemple 2

Dans ce nouvel exemple, le formulaire est protégé par un jeton cryptographique. Ce jeton est créé par le serveur à la réception d'une requête vers

`/account`

et transmis dans un champ caché du formulaire (nommé dans l'exemple

`csrf-token`

) avec le reste de la réponse. Le formulaire ressemble à ceci :

```
<form action="/account/edit-email" method="post">
    <input type="hidden" name="csrf-token" value="4db830b9ca7301988ed14e3ed04a47c1">
    <input type="email" name="email">
    <button type="submit">Change Email</button>
</form>
```

Un utilisateur légitime voulant modifier son email, va renvoyer le même jeton que celui transmis par le serveur. Ce-dernier va pouvoir faire la liaison entre la première requête vers

```
/account
```

et la seconde requête vers

```
/account/edit-mail
```

qui contient le même jeton. Dans le cas où ces jetons ne seraient pas égaux, cela signifie que la seconde requête ne provient pas d'une première requête vers

```
/account
```

, et donc que la requête est probablement illégitime ; l'email n'est alors pas modifié par le serveur.

```
POST /account/edit-email HTTP/2
Host: example.ex
Cookie: session=JV1WLK4mrRS3o4zX64NOS2fmADrFKIgo

csrf-token=4db830b9ca7301988ed14e3ed04a47c1&email=new-email@root-me.org
```

Il existe malgré tout des techniques pour contourner ce jeton. Imaginons que ce dernier n'est pas lié à un utilisateur, mais commun à tous. Cela signifie que n'importe quel utilisateur réalisant une requête vers

```
/account
```

génère un jeton qui peut être utilisé par n'importe qui pour légitimer une requête vers

```
/account/edit-mail
```

.

Un attaquant peut alors générer un jeton valide au préalable de son exploitation et le placer comme valeur dans sa charge malveillante :

```
<form id="malicious" action="https://example.ex/account/edit-email" method="post">
    <input type="hidden" name="csrf-token" value="f7cc02f6ba84fbe0d2d90ca3fee3946d">
    <input type="email" name="email" value="hackerman@root-me.org" />
    <button type="submit">Change Email</button>
</form>
<script>
    document.getElementById('malicious').submit();
</script>
```

Les jetons étant communs à tous les utilisateurs, le serveur va simplement vérifier que le token est dans sa base de données. Étant générée en amont sur le compte de l'attaquant de manière légitime, la requête est validée et l'attaque réussie.

pour_que_la_verification_par_jeton_csrf_soit_efficace_il_est_important_que_celui-ci_soit_genere_de_maniere_aleatoire_unique_pour_chaque_utilisateur_et_etre_associe_a_une_duree_de_validite_limitee

Retour fiches vulnérabilités

- Cyber fiches vulnérabilités