




# Audit de Sécurité du tchat Manao (OWASP ZAP)

À la suite des scans de l'outil d'audit de sécurité d'API ZAP nous nous retrouvons avec 3 alertes de sécurité de catégorie « intermédiaire » :

- >  .htaccess Information Leak (602)
- >  Absence de Jetons Anti-CSRF (1240)
- >  CSP: Wildcard Directive (1287)

# 1/ .htaccess Information Leak

Le fichier .htaccess permet de modifier la configuration d'un server Apache et d'y prendre la main en activant/désactivant des fonctionnalités additionnelles sur le server. Afin de contrer cette menace il faut limiter l'accès au fichier .htaccess, pour cela :

- Dans le fichier .htaccess rajoutez ces lignes :

```
# Deny access to .htaccess
<Files .htaccess>
Order allow,deny
Deny from all
</Files>
```

**Problème** : Nous utilisons un server web Express et non Apache directement. Nous n'avons pas accès directement au .htaccess.

**Source** : <https://www.opentechguides.com/how-to/article/apache/115/htaccess-file-dir-security.html>

- Importer le module express-htaccess-middleware avec npm et rajouter dans l'index.js de notre application :

```
var RewriteMiddleware = require('express-htaccess-middleware');
var RewriteOptions = {
  file: path.resolve(__dirname, '.htaccess'),
  verbose: (process.env.NODE_ENV == 'development'),
  watch: (process.env.NODE_ENV == 'development'),
};

app.use(RewriteMiddleware(RewriteOptions));
```

**Problème** : pour une raison inconnue le module n'a pas l'effet attendu et la menace est toujours active

**Source** : <https://www.npmjs.com/package/express-htaccess-middleware>

## 2/ Absence de jetons anti-CSRF

La contrefaçon de requête intersites (Cross Site Request Forgery - CSRF) est une attaque qui consiste à forcer une victime à envoyer une requête HTTP vers une destination cible, sans qu'elle n'en ait ni connaissance ni intention, afin d'effectuer une action en se faisant passer pour la victime. La cause originelle est que les fonctionnalités de l'application sont appelées à l'aide d'URL ou d'actions de formulaires prévisibles et reproductibles. La nature de l'attaque est que le CSRF exploite la confiance qu'un site internet accorde à un utilisateur. En revanche, le cross-site scripting (XSS) exploite la confiance que l'utilisateur porte à un site internet. Comme XSS, les attaques CSRF ne sont pas nécessairement multi-sites, mais elles peuvent l'être. La contrefaçon de requête intersite est également connue sous les noms CSRF, XSRF, attaques-en un clic (one-click attack), session riding, confused deputy et sea surf.

Pour résoudre ce problème :

- Utilisez une librairie ou un framework approuvé qui ne permet pas cette vulnérabilité, ou qui contient des fonctionnalités permettant d'éviter plus facilement cette vulnérabilité.  
Utilisez par exemple des librairies anti-CSRF telles que CSRFGuard de l'OWASP.
- Assurez-vous que votre application soit exempte de problèmes de cross-site scripting, parce que la plupart des défenses contre le CSRF peuvent être contournées en utilisant des scripts contrôlés par le pirate.
- Générez une valeur à usage unique pour chaque formulaire, placez-la dans le formulaire et vérifiez-la à la réception du formulaire. Assurez-vous que cette valeur unique ne soit pas prévisible (CWE-330). Notez que ceci peut aussi être contourné en utilisant XSS (Chapitre 1).
- Identifiez les opérations particulièrement dangereuses. Quand l'utilisateur exécute une opération dangereuse, envoyez une requête de confirmation distincte pour vérifier que l'utilisateur veut effectivement effectuer cette opération. Notez que ceci peut aussi être contourné en utilisant XSS.

Utilisez la librairie de gestion de session ESAPI. Cette librairie comprend un composant pour le contrôle de CSRF.

N'utilisez pas la méthode GET pour les requêtes entraînant un changement d'état.

- Vérifiez l'en-tête HTTP Referer pour voir si la requête provient d'une page attendue. Ceci pourrait toutefois restreindre la fonctionnalité de l'application, car les utilisateurs ou les serveurs proxy pourraient avoir désactivé le renvoi du HTTP Referer pour des raisons de confidentialité.

**Problème** : Manque de temps pour l'implémentation et d'expérience technique pour un sujet assez complexe d'architecture de sécurité d'API

**Source** : <https://www.zaproxy.org/docs/alerts/10202/>

### 3/ CSP: wildcard directive

Content Security Policy (CSP) est une couche de sécurité qui aide à détecter et lutter contre certains types d'attaques. Incluant (ce n'est une liste exhaustive) le Cross Site Scripting (XSS – chapitre 1) et les injections de données. Ces types d'attaques sont utilisés pour du vol de données ou la propagation de malwares. Le CSP fournit un set de headers HTTP qui permet au propriétaire du site de déclarer des sources de contenus que le navigateur serait autorisé à charger sur une page. (JavaScript, CSS, HTML frames, polices, images and objets intégrables comme des applets Java, ActiveX, fichiers audios et vidéos)

Pour résoudre ce problème :

- Configurer une stratégie CSP nécessite d'utiliser un en-tête HTTP Content-Security-Policy pour une page web et de spécifier une valeur pour contrôler les ressources que le navigateur est autorisé à charger pour cette page. Ainsi, une page qui charge et affiche des images peut autoriser les images stockées n'importe où mais n'autoriser les envois de formulaires que vers certaines adresses (par exemple). Pour un site dont tout le contenu est fourni par le site lui-même ou par les sous-domaines de source-sure.example.net (qui peut être un autre site) :

```
Content-Security-Policy: default-src 'self' *.source-sure.example.net
```

**Source** : <https://developer.mozilla.org/fr/docs/Web/HTTP/CSP>

**Problème** : l'ajout du header par différents moyens ne change rien à l'alerte CSP : wildcard directive, comme s'il n'était pas réellement implémenté.

**Moyen 1** : <https://github.com/frux/express-csp-header>

**Moyen 2** :

```
app.use(function (req, res, next) {  
  res.setHeader(  
    'Content-Security-Policy',  
    "default-src 'self'; font-src 'self'; img-src 'self'; script-src 'self'; style-src 'self'  
  );  
  next();  
});
```

**Source** : <https://stackoverflow.com/questions/21048252/nodejs-where-exactly-can-i-put-the-content-security-policy>