



# Rapport des Vulnérabilités

Mohamed Khachlaa

### 1. Stockage des mots de passe en texte clair :

Gravité: Critique

**Description :** Les mots de passe sont stockés en texte clair dans le fichier database. j son. Si un attaquant accède à ce fichier, il peut facilement lire toutes les informations d'identification des utilisateurs.

**Impact**: Compromission totale des comptes, vol d'identifiants et réutilisation potentielle des mots de passe sur d'autres services.

**Solution:** utilisation des outiles de hashage, (bcrypt)

### 2. Gestion de session non sécurisée :

Gravité: Élevée

**Description :** La configuration de session manque de paramètres de sécurité critiques :

- cookie.secure n'est pas défini, ce qui permet d'envoyer les cookies de session via HTTP (non sécurisé).
- cookie.http0nly n'est pas activé, ce qui rend les cookies accessibles au JavaScript côté client (augmente le risque de XSS).
- Aucun mécanisme d'expiration ou de rotation de session n'est implémenté.

**Impact**: Détournement de session, accès non autorisé et compromission de compte.

#### Solution:

- Envoyer les cookies uniquement via HTTPS en production
- Ajouter httpOnly: true
- Définir une expiration (ex. : 1 jour)

### 3. Absence de protection CSRF:

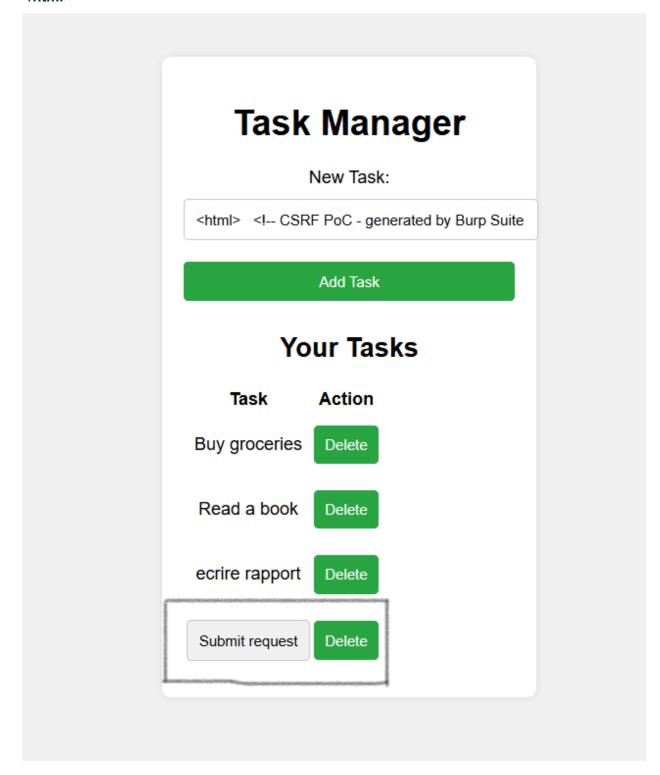
#### Gravité : Élevée

**Description :** L'application est vulnérable aux attaques **CSRF** (Cross-Site Request Forgery). Un attaquant peut tromper un utilisateur connecté pour qu'il effectue des actions non intentionnelles .

#### Exemple:

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
  <script>history.pushState(", ", '/')</script>
    <form method="GET" action="https://google.com">
        <input type="hidden" name="email" value="some@email.com" />
        <input type="submit" value="Submit request" />
        </form>
        <script>
        document.forms[0].submit();
        </script>
```

</body>



Impact : Actions non autorisées effectuées au nom de l'utilisateur.

#### Solution:

Utilisez une bibliothèque comme csurf pour ajouter une protection CSRF

## 4. Absence de validation et de nettoyage des entrées:

Gravité: Élevée

**Description :** Les entrées utilisateur (ex. : nom d'utilisateur, mot de passe, tâche) ne sont pas validées ni nettoyées. Cela rend l'application vulnérable à :

- XSS (Cross-Site Scripting) : Des scripts malveillants peuvent être injectés.
- Injection de commandes : Si les entrées utilisateur sont utilisées dans des commandes système.

Impact : Fuites de données, accès non autorisé et compromission du serveur.

#### Solution:

Utilisez une bibliothèque comme express-validator pour valider et nettoyer les entrées.

Utilisez les RegEx.

## 5. Absence de Limitation des Taux de Requêtes:

Gravité: Moyenne

**Description**: Il n'y a pas de limitation de débit sur les routes de connexion ou d'inscription, rendant l'application vulnérable aux attaques par force brute.

**Impact**: Compromission des comptes par tentatives répétées de connexion.

**Solution :** Utilisez une bibliothèque comme **express-rate-limit** pour limiter le nombre de requêtes.

### Exemple

```
const rateLimit = require('express-rate-limit');
const limiter = rateLimit({
   windowMs: 15 * 60 * 1000, // 15 minutes
   max: 5, // Limiter chaque IP à 5 requêtes par fenêtre de temps
});
app.use('/login', limiter);
app.use('/register', limiter);
```