Rapport de Patch

Sécurisation du Frontend

Projet : IPSSI_PATCH

Auteur : GAVI Holali David

Date : 11 décembre 2025

IPSSI - Mastère Cybersécurité et Cloud Computing

Table des matières

1 Introduction

Le frontend du projet `IPSSI_PATCH`, développé en React, interagit avec un backend Express/SQLite. Après la correction du backend, plusieurs modifications étaient nécessaires pour :

— Aligner le frontend avec les nouvelles routes sécurisées
— Corriger les appels API obsolètes
— Protéger le rendu des données
— Éviter l'affichage d'informations sensibles

Ce document détaille les vulnérabilités du frontend, leur impact et les correctifs appliqués, avec les extraits de code correspondants.

2 Vue d'ensemble des corrections

Le tableau ci-dessous présente une synthèse des problèmes identifiés et des correctifs appliqués au frontend.

Problème	**Gravité**	**Correction apportée**
Envoi de requêtes SQL brutes	Critique	Remplacement par JSON `{id: ...}`
Content-Type incorrect	Moyen	Passage à `application/json`
Soumission des commentaires en texte brut	Haut	Envoi d'un JSON `{content: ...}`
Affichage du mot de passe	Critique	Suppression de l'affichage
Absence de validation dans les inputs	Moyen	Validation frontend + backend
Risque XSS dans les commentaires	Bas	React protège, backend sanitize

TABLE 1 – Récapitulatif des problèmes et correctifs frontend

3 Détails des vulnérabilités et correctifs

3.1 Envoi de SQL brut à la route /user

3.1.1 Code vulnérable

```
axios.post(
  'http://localhost:8000/user',
  `SELECT id, name FROM users WHERE id = ${queryId}`,
  {
    headers: { "Content-Type" : 'text/plain' }
  }
);
```

3.1.2 Pourquoi c'est dangereux ?

— On envoyait directement une requête SQL construite par le frontend
— Avant patch backend, cette requête était exécutée telle quelle → risque de commande malveillante, même involontaire

3.1.3 Correctif appliqué

```
const response = await axios.post(
  'http://localhost:8000/user',
  { id: queryId }, // << JSON securise
  {
    headers : {
      "Content-Type" : 'application/json'
    }
  }
);
```

Pourquoi ce correctif ?
— Aligné sur le backend qui attend `{id: ...}`
— Empêche tout envoi de SQL
— Respecte les bonnes pratiques REST/JSON

3.2 Envoi du commentaire en texte brut

3.2.1 Code vulnérable

```
await axios.post(
  'http://localhost:8000/comment',
  newComment,
  { headers: { "Content-Type": 'text/plain' } }
);
```

3.2.2 Risque

— Le backend ne recevait pas un JSON mais du texte brut → incohérence d'API
— Impossible d'utiliser les middlewares de sanitization correctement

3.2.3 Correctif

```
await axios.post(
  'http://localhost:8000/comment',
  { content: newComment }, // JSON propre
  { headers: { "Content-Type": 'application/json' } }
);
```

Impact positif :
— Le middleware `sanitizeComment` fonctionne parfaitement
— Alignement total frontend ↔ backend

3.3 Affichage du mot de passe dans l'UI

3.3.1 Code vulnérable

```
ID: {u.id}      Name: {u.name}      Password: {u.password}
```

3.3.2 Pourquoi c'est dangereux ?

— Même si le backend renvoyait encore un mot de passe haché, on n'affiche jamais un mot de passe
— Risque pédagogique d'exposition involontaire
— Mauvaise pratique UI/UX et sécurité

3.3.3 Correctif

```
ID: {u.id}      Name: {u.name}
```

Pourquoi ?
— Les mots de passe appartiennent strictement à l'utilisateur
— Même hachés, ils ne doivent pas apparaître à l'écran

3.4 Risque XSS dans les commentaires

3.4.1 Analyse

React échappe automatiquement la plupart du HTML :

```
{comment.content}
```

⇒ Pas de risque tant que `dangerouslySetInnerHTML` n'est pas utilisé.

3.4.2 Correctif indirect

Le backend sanitize déjà :

```
content = content
  .replace(/&/g, "&amp;")
  .replace(/</g, "&lt;")
  .replace(/>/g, "&gt;")
  .replace(/"/g, "&quot;");
```

Résultat :
Impossible pour un attaquant d'injecter :

```
<script>alert("XSS")</script>
```

4 Code complet FRONTEND corrigé

(Uniquement les parties modifiées, compact pour documentation)

4.1 Query sécurisée

```
const response = await axios.post(
  'http://localhost:8000/user',
  { id: queryId },
  {
    headers : {
      "Content-Type" : 'application/json'
    }
  }
);
```

4.2 Commentaire envoyé en JSON

```
await axios.post(
  'http://localhost:8000/comment',
  { content: newComment },
  { headers: { "Content-Type": "application/json" } }
);
```

4.3 Suppression de l'affichage du mot de passe

```
<p key={u.id}>
  ID: {u.id}     Name: {u.name}
</p>
```

5 Résumé des correctifs appliqués au frontend

Correctif	**Rôle**
Envoi JSON pour `/user`	Compatibilité backend + sécurité
Envoi JSON pour `/comment`	Utilisation correcte du middleware `sanitizeComment`
Suppression affichage mot de passe	Confidentialité
Vérification `queryId` côté UI	UX + validité
Acceptation automatique du XSS sanitize backend	Sécurité additionnelle

Table 2 – Synthèse des correctifs frontend

6 Conclusion

Le frontend est désormais entièrement aligné avec le backend sécurisé :

Les entrées utilisateur sont validées

Les appels API sont corrects

Aucun mot de passe n'est visible

Les commentaires sont protégés contre la XSS

La logique de sécurité est respectée

Ces corrections rendent l'architecture plus robuste et totalement compatible avec les principes de sécurité enseignés en cybersécurité.

7 Références

— React Security Best Practices : https://react.dev/learn/security
— OWASP XSS Prevention Cheat Sheet : https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
— Axios Documentation : https://axios-http.com/docs/intro
— MDN Web Docs - Content-Type : https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type