Formulaire sécurisé

Security By Design

12 mai

M1 FT EPSI
Créé par : KUETI Yann

epsi
l'école d'ingénierie informatique

Table des matières

Introduction

Dans un contexte numérique où les cyberattaques augmentent chaque année, la sécurisation des données utilisateur est devenue un impératif critique. Ce projet s'inscrit dans une démarche de *Security by Design*, visant à concevoir un formulaire de contact intégrant nativement les meilleures pratiques de cybersécurité.

Nous avons développé une solution complète répondant aux enjeux suivants :

- Protection des données personnelles
- Prévention des attaques courantes (XSS, CSRF, injections)
- Authentification robuste des administrateurs
- Traçabilité des actions sensibles

L'architecture combine des technologies Node.js, Express, reCAPTCHA avec des mécanismes cryptographiques modernes comme AES-256-GCM, bcrypt pour offrir une solution à la fois performante et sécurisée.

1. Objectifs du Projet

1.1 Objectif Principal

L'objectif était de développer un formulaire de contact sécurisé en environnement local, en appliquant les principes de "Security by Design". Ce formulaire devait permettre de collecter les données suivantes : nom, email et message, tout en assurant leur protection contre les principales vulnérabilités web.

1.2 Objectifs Spécifiques

Fonctionnalité	Technologie Employée
Chiffrement des messages	AES-256-GCM avec salage
Authentification admin	bcrypt (12 rounds)
Protection anti-bots	reCAPTCHA v2 + rate limiting
Sécurisation HTTPS	Helmet + CSP (Content Security Policy) strict

2. Architecture Sécurisée

Formulaire_de_contact_securise/
├── config/stocke la configuration du reCAPTCHA et les certificats SSL.
│   ├── ssl/
│   ├── crypto.json
│   └── recaptcha.js
├── logs/ archive les logs d'accès et les erreurs et les données du formulaire cryptées
│   ├── access.log
│   ├── error.log
│   └── messages.json
├── node_modules/
├── public/ contient les fichiers HTML, CSS et JS statiques.

│   ├── css/
│   ├── js/
│   ├── contact.html
│   └── login.html
├── src/ contient le serveur Node.js (server.js) et les scripts de chiffrement/déchiffrement.
│   ├── decrypt.js
│   └── server.js
├── package.json
├── package-lock.json
└── README.md

3. Fonctionnalités sécuritaires mises en place et risques couverts

3.1 Transmission sécurisée via HTTPS

- Génération d'un certificat SSL local via OpenSSL.
- Configuration du serveur Express pour être uniquement accessible en HTTPS.
- **Risque couvert** : interception des données par un tiers (attaque de type Man-in-the-Middle).

3.2 Sécurisation des entêtes HTTPS

- Utilisation du middleware helmet avec une politique CSP personnalisée limitant les sources aux seules nécessaires (reCAPTCHA, ressources locales).
- **Risque couvert** : attaques XSS, clickjacking et injection de contenu malveillant.

3.3 Protection contre les attaques automatisées (bots)

- Intégration de Google reCAPTCHA v2 côté client et vérification côté serveur via une API.
- **Risque couvert** : spams, bots envoyant massivement des requêtes au serveur.

3.4 Gestion des sessions

- Utilisation de express-session avec des cookies HttpOnly et Secure activés.
- Session utilisateur créée uniquement si authentification réussie.
- **Risque couvert** : vol ou manipulation de session.

3.5 Hachage des mots de passe

- Le mot de passe de l'administrateur est préalablement haché avec bcrypt (12 rounds).
- **Risque couvert** : vol ou fuite de mot de passe en clair.

3.6 Chiffrement des messages avant stockage

- Utilisation de l'algorithme AES-256-GCM via le module crypto de Node.js.
- Les données envoyées via le formulaire de contact sont chiffrées et enregistrées dans un fichier JSON (message.json). Le déchiffrement par l'administrateur se fait en exécutant le **fichier src/decrypt.js**
- **Risque couvert** : vol de données sensibles en cas d'accès non autorisé au stockage.

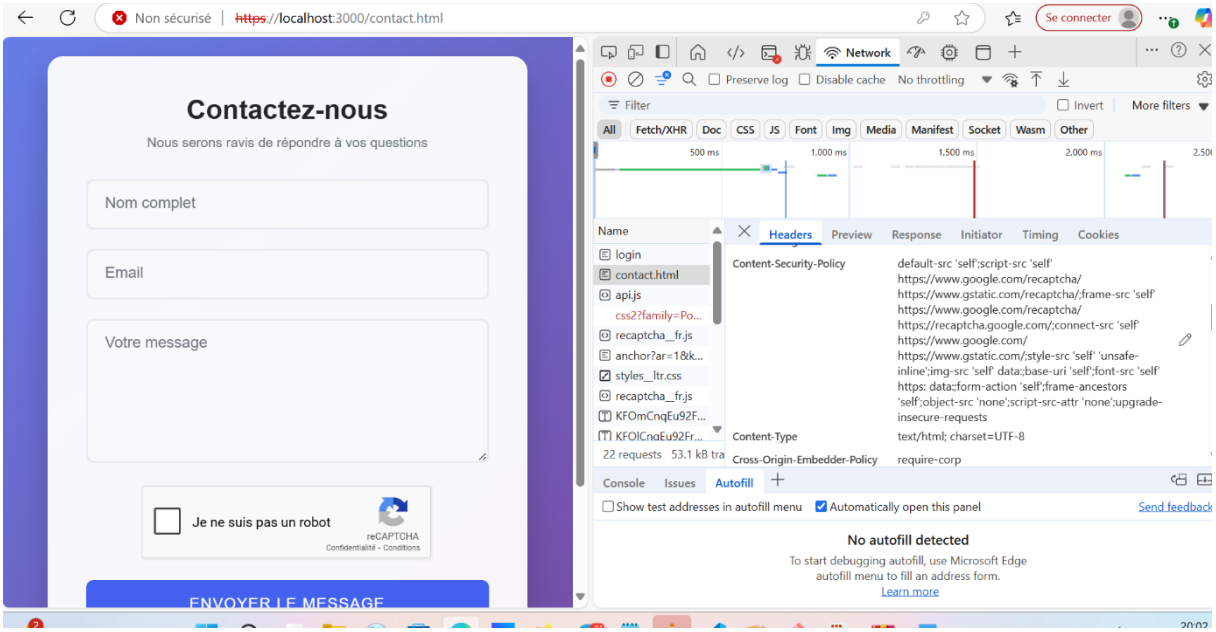
3.7 Logs d'accès et d'erreurs

- Chaque requête entrante est loggée avec date et méthode dans logs/access.log.
- Les erreurs (ex : reCAPTCHA invalide) sont enregistrées dans logs/error.log.
- **Risque couvert** : manque de traçabilité en cas d'incident ou de tentative d'intrusion.

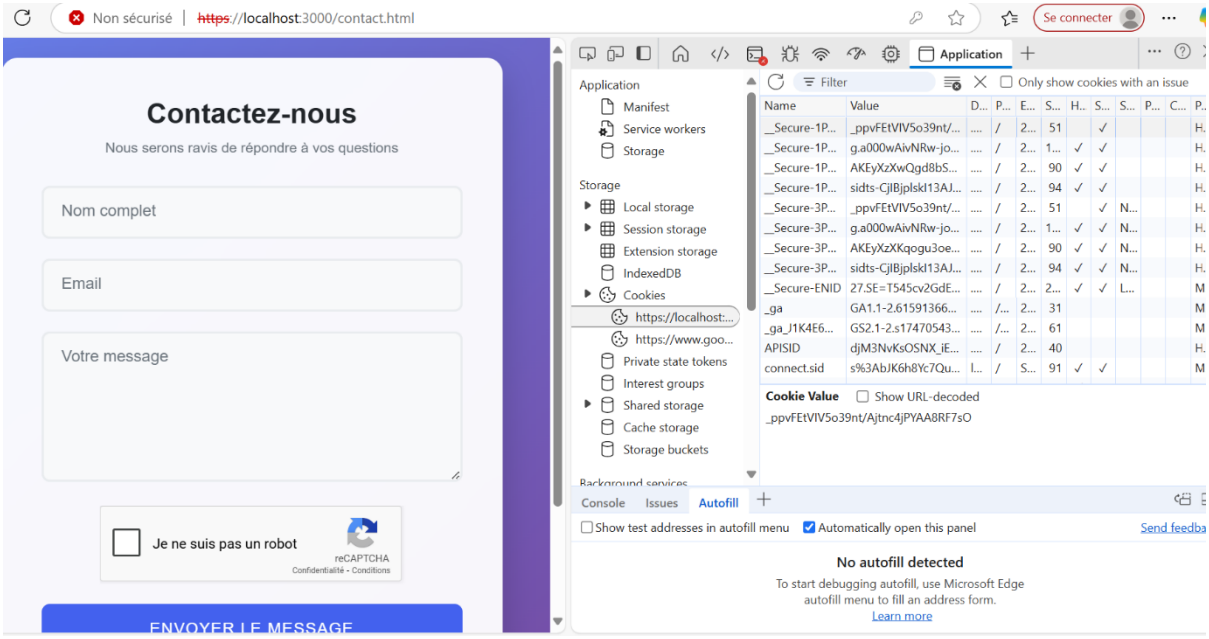
4. Résultats des Tests

4.1 DevTools (Application / Network)

Headers HTTPS : les en-têtes de sécurité (CSP, X-Content-Type, X-Frame-Options) sont bien présents.

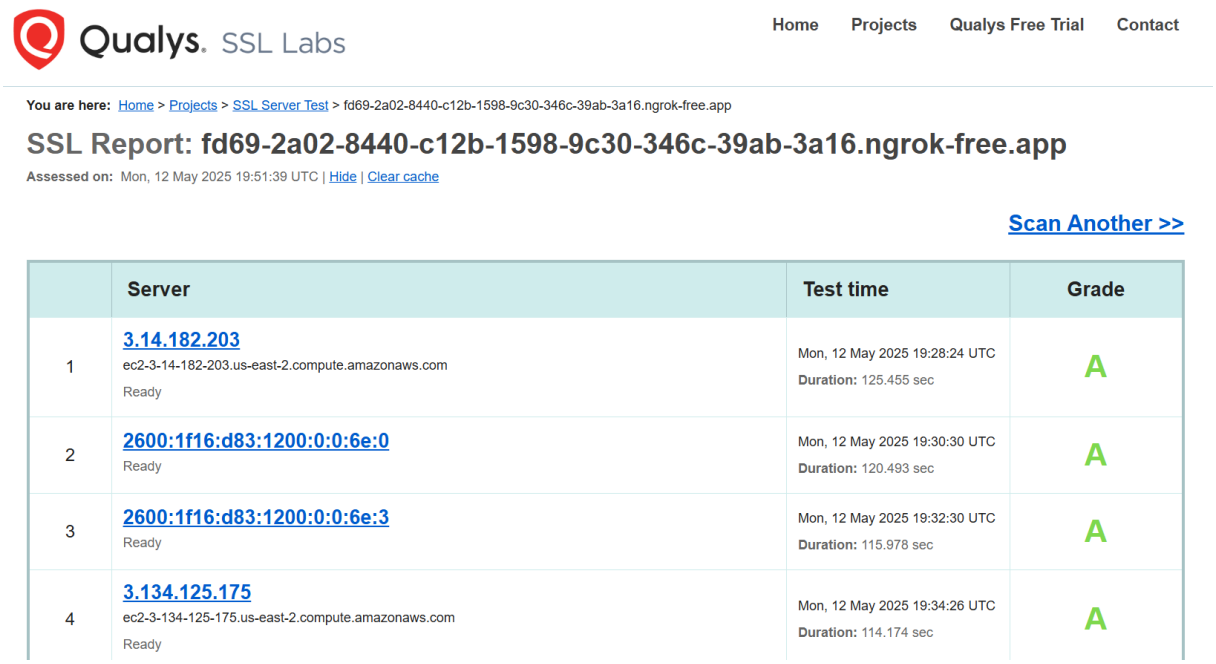


Cookies : les flags Secure et HttpOnly sont bien activés.



4.2 configuration HTTPS avec SSL Labs

Elle est bien très bien faite



5. Problèmes rencontrés et résolutions

Nous avons rencontrés plusieurs difficultés qui nous ont d'ailleurs fait recommencer le travail à 2 reprises.

Problème	Solution mise en place
Erreur MODULE_NOT_FOUND lors de l'exécution de decrypt.js	Création du fichier manquant et positionnement correct dans src/
Conflit de CSP lors du chargement du reCAPTCHA (surtout)	Ajustement fin des directives script-src, frame-src et connect-src dans helmet
Session non persistante malgré l'authentification	Ajout des options cookie: { secure: true, httpOnly: true }
Impossible de tester sans HTTPS	Forçage de redirection vers HTTPS dans server.js et utilisation d'un certificat auto-signé

Conclusion

En somme, le formulaire répond à l'ensemble des exigences de l'énoncé. Il intègre les bonnes pratiques de sécurité web modernes : HTTPS, reCAPTCHA, hachage des mots de passe, chiffrement des données sensibles, politiques CSP strictes, cookies sécurisés, et logs d'activités. L'ensemble a été testé et validé avec des outils professionnels (DevTools, SSL Labs).
En définitive, ce formulaire ne se contente pas de répondre à des exigences techniques : il incarne une philosophie où chaque ligne de code participe à la protection fondamentale de la vie privée des utilisateurs.