API CINEMA

Express REST MVC • PostgreSQL (Prisma) • MongoDB (Mongoose)

Sécurisation avant mise en production

Par Mathis BOHEC & Marie COURTOIS

Sommaire

I- Besoin & cas d’usage

II- Architecture technique

III- Modèles de données

IV- Sécurisation (mesures + preuves)

V- OWASP (mapping)

VI- Outils (SONAR / zap / npm audit)

VII- Démo

VIII- Bilan & amélioration

I- Besoin & cas d’usage

Expression de besoin

- API pour gérer un catalogue de films
- Consommée par un client web/mobile
- Fonctions clés : consulter, rechercher, s'inscrire, noter

Rôles

- Visiteur : consulter & rechercher
- Utilisateur : noter + profil
- Admin : CRUD films + gestion users

Scénarios (exemple)

- Lister/consulter un film (public)
- Filtrer par titre/genre/réalisateur
- Login token JWT
- Noter un film (auth)
- Admin gère le catalogue

II- Architecture technique 3-tiers

Client
(Web / Mobile)

→

API Express
REST ◦ MVC

→

Données
PostgreSQL + MongoDB

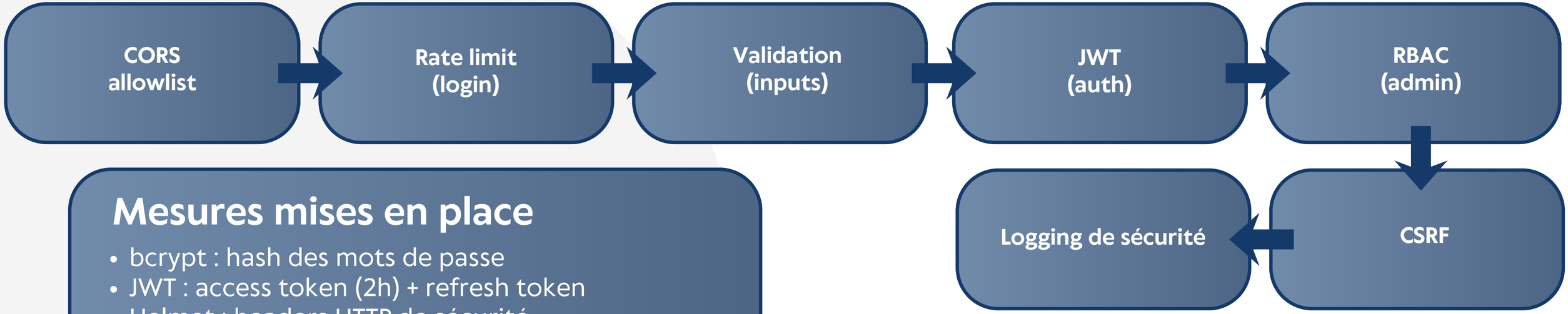
Pourquoi SQL + NoSQL ?

- PostgreSQL : données structurées + relations (films, acteurs, réalisateurs)
- MongoDB : logs/historiques & documents flexibles
- Chacun utilisé là où il est le plus efficace

Stack sécurité

- CORS allowlist · Helmet · Rate limiting
- JWT (access + refresh) · bcrypt
- Validation des entrées · erreurs centralisées
- CSRF token (1h, usage unique)
- Logging sécurité (accès sensibles / auth)

Sécurisation



Mesures mises en place

- bcrypt : hash des mots de passe
- JWT : access token (2h) + refresh token
- Helmet : headers HTTP de sécurité
- Erreurs centralisées : pas de fuite d'infos
- Refresh token : rotation + max 5 sessions actives
- CSRF : token 1h + 403 si absent/invalide/expiré + usage unique
- Logging : traces des actions sensibles (userId/role/ip/path)
- Front tokens : access token en mémoire (pas de storage), refresh token en sessionStorage, refresh auto + logout = révocation

Stack sécurité

- Brute force & spam (rate limit)
- Injection (validation + ORM)
- Accès non autorisés (RBAC)
- Misconfiguration (CORS/headers)

Sécurisation - Extraits de code

Bcrypt

```
85  const hashedPassword = await bcrypt.hash(password, 10);
86
87  const newUser = await User.create({
88      username: sanitizedUsername,
89      email: sanitizedEmail,
90      password: hashedPassword
91  });
92
93  const userResponse = {
94      id: newUser._id,
95      username: newUser.username,
96      email: newUser.email,
97      role: newUser.role
98  };
```

JWT

```
147         const token = jwt.sign(
148             { userId: user._id, role: user.role },
149             process.env.JWT_SECRET,
150             { expiresIn: "2h" }
151         );
152
153         res.json({
154             token,
155             user: {
156                 id: user._id,
157                 username: user.username,
158                 email: user.email,
159             },
160         });
161
```

Payload : { userId, role } + expiresIn: "2h"

Sécurisation - Extraits de code

Refresh Token

```
142  const refreshToken = crypto.randomBytes(64).toString('hex');
143
144  // Limiter à 5 refresh tokens actifs par utilisateur (rotation)
145  const userWithTokens = await User.findById(user._id).select('+refreshTokens');
146  const refreshTokens = userWithTokens.refreshTokens || [];
147  if (refreshTokens.length >= 5) {
148      refreshTokens.shift();
149  }
150  refreshTokens.push(refreshToken);
151
152  await User.findByIdAndUpdate(user._id, { refreshTokens });
153
154  securityLogger.logLoginAttempt(req, true);
155
156  res.json({
157      accessToken,
158      refreshToken,
159      user: {
160          id: user._id,
161          username: user.username,
162          email: user.email,
163      },
164  });
```

Rotation : max 5 refresh tokens / user (éviction des plus anciens)

Rate limit

```
20  app.use(rateLimit({ windowMs: 60_000, max: 100 }));
21  app.use("/auth/login", rateLimit({ windowMs: 15*60_000, max: 10 }));
```

Sécurisation - Extraits de code

Validate

```js
const { validationResult } = require("express-validator");

function validate(req, res, next) {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(422).json({ errors: errors.array() });
  }
  next();
}

```

CORS

```js
app.use(cors({
    origin: function (origin, callback) {
        // Autoriser les requêtes sans origine (mobile apps, Postman, etc.) en développement
        if (!origin && process.env.NODE_ENV !== 'production') {
            return callback(null, true);
        }
        if (allowedOrigins.indexOf(origin) !== -1 || !origin) {
            callback(null, true);
        } else {
            callback(new Error('Non autorisé par CORS'));
        }
    },
    credentials: true,
    allowedHeaders: ["Authorization", "Content-Type", "X-Requested-With"],
    methods: ["GET", "POST", "PUT", "DELETE", "PATCH", "OPTIONS"]
}));
```

Sécurisation - Extraits de code

CSRF

```js
const crypto = require("node:crypto");

const csrfTokens = new Map();
const CSRF_TOKEN_EXPIRY = 60 * 60 * 1000; // 1 heure

/**
 * Génère un token CSRF
 */
function generateCSRFToken() {
    return crypto.randomBytes(32).toString('hex');
}


function generateCSRF(req, res, next) {
    const token = generateCSRFToken();
    const expiry = Date.now() + CSRF_TOKEN_EXPIRY;

    csrfTokens.set(token, {
        expiry,
        ip: req.ip || req.connection.remoteAddress
    });

    if (csrfTokens.size > 1000) {
        const now = Date.now();
        for (const [t, data] of csrfTokens.entries()) {
            if (data.expiry < now) {
                csrfTokens.delete(t);
            }
        }
    }

    res.locals.csrfToken = token;
    res.setHeader('X-CSRF-Token', token);
    next();
}
```

CSRF

```js
function verifyCSRF(req, res, next) {

    if (req.path.startsWith('/api/user/login') ||
        req.path.startsWith('/api/user/register') ||
        req.path.startsWith('/api/user/refresh')) {
        return next();
    }


    const token = req.headers['x-csrf-token'] || req.body?.csrfToken;

    if (!token) {

        if (process.env.NODE_ENV === 'development') {
            return next();
        }
        return res.status(403).json({ error: "Token CSRF manquant" });
    }

    const tokenData = csrfTokens.get(token);

    if (!tokenData) {
        return res.status(403).json({ error: "Token CSRF invalide" });
    }

    if (tokenData.expiry < Date.now()) {
        csrfTokens.delete(token);
        return res.status(403).json({ error: "Token CSRF expiré" });
    }

    csrfTokens.delete(token);

    next();
}
```

Limite : stockage en mémoire → en prod multi-instance, store partagé (Redis)

Sécurisation - Extraits de code

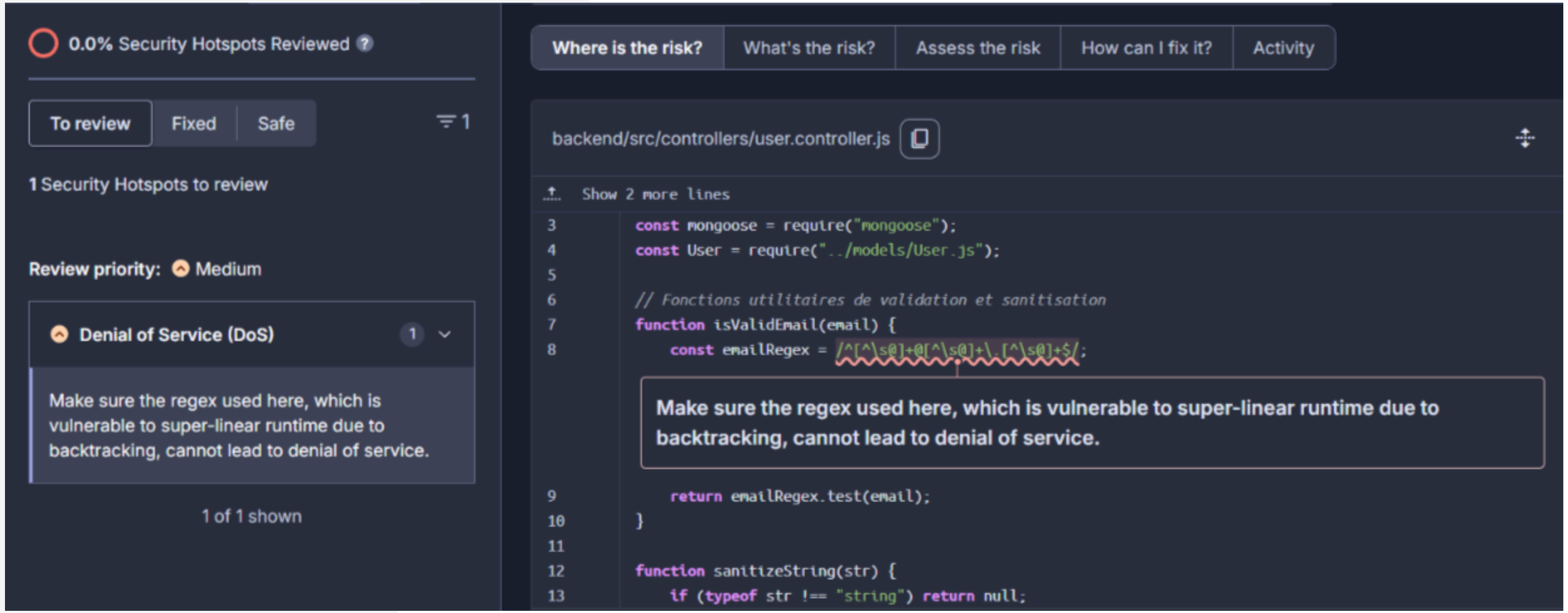
RBAC

```
logSensitiveAccess(req, resource, action) {
    const logData = {
        timestamp: new Date().toISOString(),
        event: 'SENSITIVE_ACCESS',
        userId: req.user?.userId || 'N/A',
        role: req.user?.role || 'N/A',
        resource,
        action,
        ip: req.ip || req.connection.remoteAddress,
        userAgent: req.get('user-agent'),
        method: req.method,
        path: req.path
    };

    console.log(`[SECURITY] 🔒 Accès sensible: ${action} sur ${resource} par utilisateur ${logData.userId} (${logData.role})`);
},
```

Sécurisation - Extraits de code

Helmet (headers sécurisés)

Sécurisation - Extrait de Sonar



Hotspot : risque ReDoS (backtracking regex) → fix : limitation de longueur avant regex

OWASP

A01 Broken Access Control

- Mesures : JWT + RBAC + logs accès sensibles
- Preuve : 403 sur /admin + logging SENSITIVE_ACCESS

A07 Identification & Authentication Failures

- Mesures : JWT exp + rate limit login
- Preuve : expiresIn + 10/15min
- + refresh token + rotation + stockage tokens front

A05 Misconfiguration / A02 Crypto

- A05 : CORS allowlist + Helmet
- A02 : bcrypt + HTTPS local

Outils de sécurité

SONARQUBE

- Hotspots : 1 (regex ReDoS / backtracking)
- Correction : limitation de longueur avant regex
- Autre correction : validation des types renforcée (bug) + entrées mieux contrôlées

OWASP ZAP

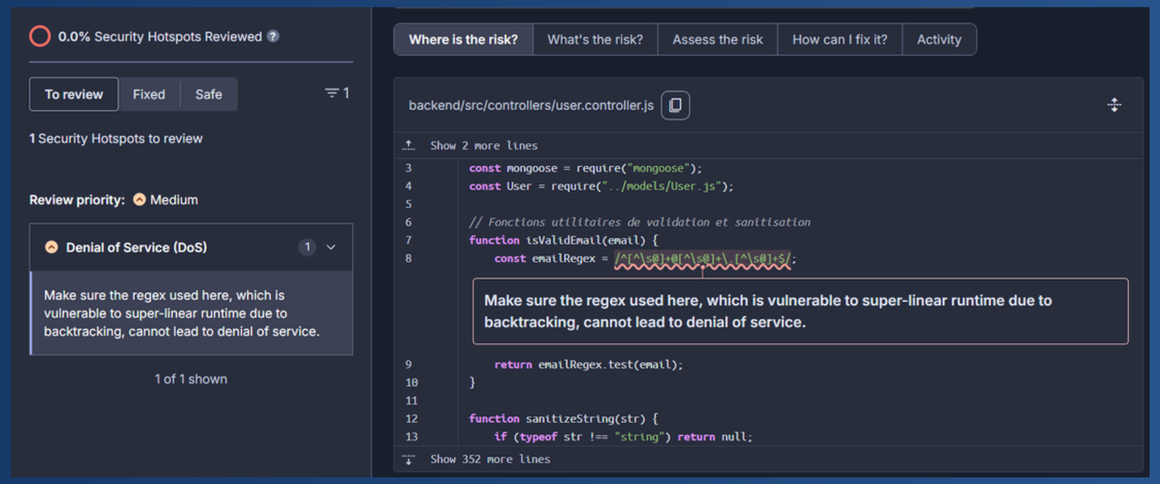
- Scan : baseline (passif) sur https://localhost:3000
- Alerte : Re-examine Cache-control Directives (Info / Low)
- Décision : on réduit le risque via tokens non persistants + recommandation Cache-Control: no-store sur routes sensibles

npm audit

- 4 vulnérabilités (2 moderate / 2 high)
- Paquets vus : body-parser, glob, js-yaml, jws
- Action : npm audit fix + upgrades ciblés / justification si contrainte

Outils de sécurité

SONARQUBE

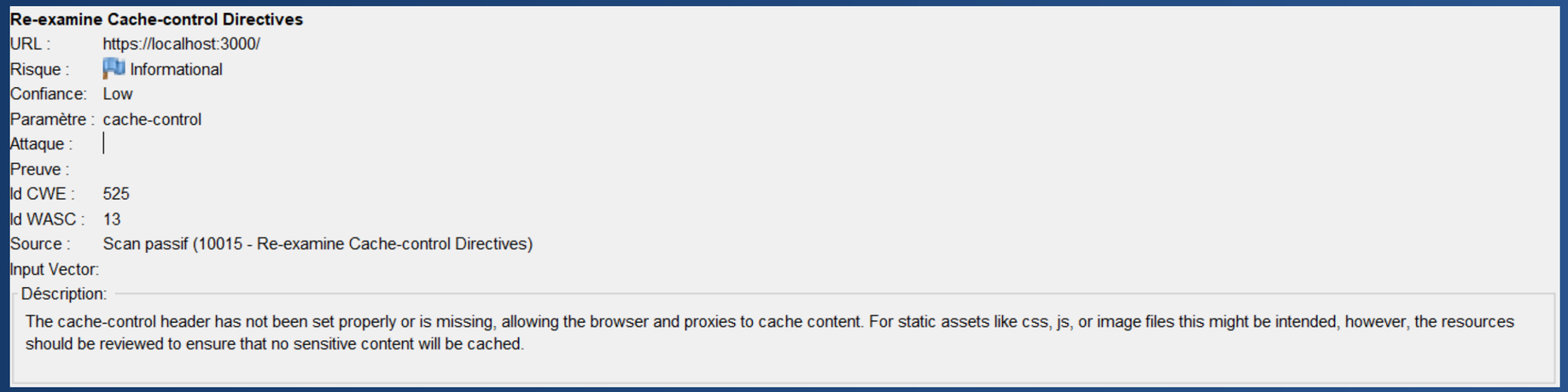


npm audit

```
# npm audit report

body-parser  2.2.0
Severity: moderate
body-parser is vulnerable to denial of service when url encoding is used - https://github.com/advisories/GHSA-wqch-xfxh-vrr4
fix available via `npm audit fix`
node_modules/body-parser

glob  10.2.0 - 10.4.5
Severity: high
glob CLI: Command injection via -c/--cmd executes matches with shell:true - https://github.com/advisories/GHSA-5j98-mcp5-4vw2
fix available via `npm audit fix`
node_modules/@jest/reporters/node_modules/glob
node_modules/jest-config/node_modules/glob
node_modules/jest-runtime/node_modules/glob

js-yaml  <3.14.2 || >=4.0.0 <4.1.1
Severity: moderate
js-yaml has prototype pollution in merge (<<) - https://github.com/advisories/GHSA-mh29-5h37-fv8m
js-yaml has prototype pollution in merge (<<) - https://github.com/advisories/GHSA-mh29-5h37-fv8m
fix available via `npm audit fix`
node_modules/@istanbuljs/load-nyc-config/node_modules/js-yaml
node_modules/js-yaml

jws  <3.2.3
Severity: high
auth0/node-jws Improperly Verifies HMAC Signature - https://github.com/advisories/GHSA-869p-cjfg-cm3x
fix available via `npm audit fix`
node_modules/jws

4 vulnerabilities (2 moderate, 2 high)

To address all issues, run:
  npm audit fix
```

Outils de sécurité

OWASP ZAP



Re-examine Cache-control Directives

URL : https://localhost:3000/

Risque : Informational

Confiance: Low

Paramètre : cache-control

Attaque :

Preuve :

Id CWE : 525

Id WASC : 13

Source : Scan passif (10015 - Re-examine Cache-control Directives)

Input Vector:

Déscription:

The cache-control header has not been set properly or is missing, allowing the browser and proxies to cache content. For static assets like css, js, or image files this might be intended, however, the resources should be reviewed to ensure that no sensitive content will be cached.

Sécurisation - Extraits de code

Ce qui est bon

- Architecture MVC claire + séparation responsabilités
- RBAC admin/user + routes protégées
- Validation + erreurs centralisées
- Outils : Sonar / ZAP / npm audit (preuves)

A améliorer

- Stocker hash des refresh tokens (au lieu de clair)
- CSRF : store partagé Redis en prod multi-instance
- Logs structurés + centralisation (ELK / Loki)
- Tests d'intégration sécurité (RBAC/injection/bruteforce)

Merci pour votre écoute !

Mathis BOHEC & Marie COURTOIS