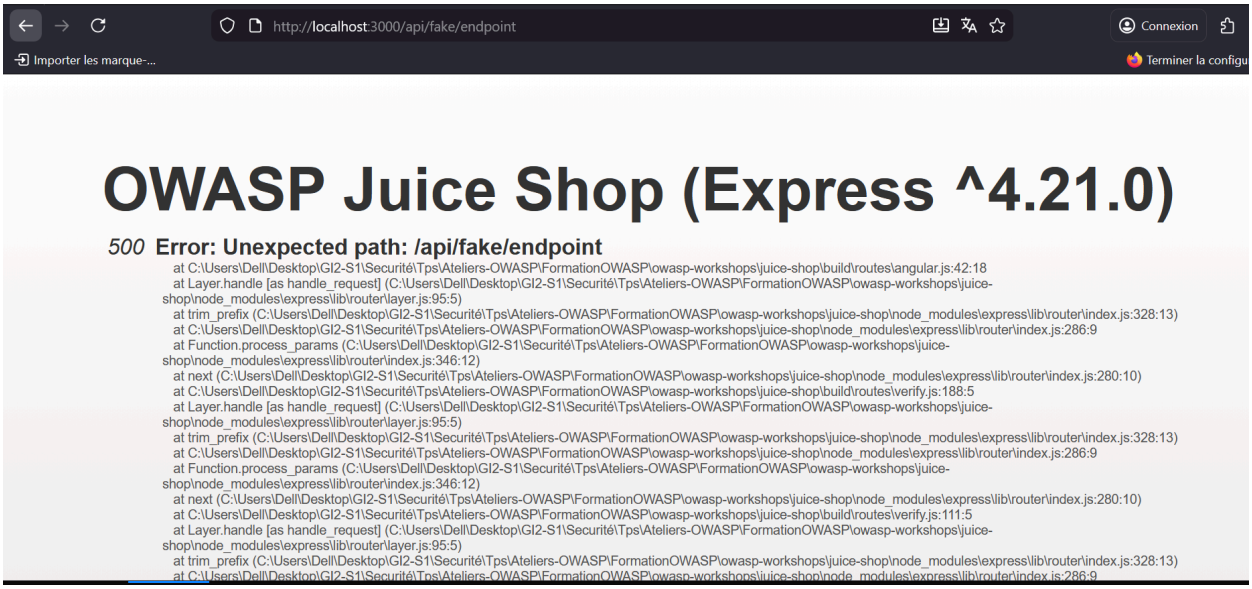
Rapport
Atelier 5:
A05 – Security Misconfiguration

SABRI OMAIMA

1.Stack Trace Visible (Mode Debug)

Reproduction :·

- **Accédez à une URL inexistante, par exemple : http://localhost:3000/api/fake/endpoint**
- **Vérifiez si la stack trace du serveur est visible dans la réponse JSON et dans le navigateur web :**



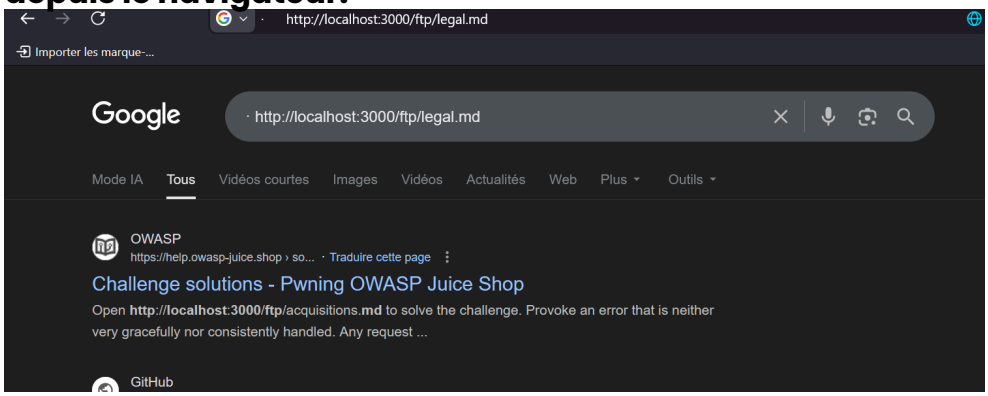

```
info: Restored 2-star loginAdminChallenge (Login Admin)
info: Restored 1-star scoreBoardChallenge (Score Board)
Error: Unexpected path: /api/fake/endpoint
    at C:\Users\Dell\Desktop\GI2-S1\Securité\Tps\Ateliers-OWASP\FormationOWASP\owasp-workshops\juice-shop\build\routes\angular.js:42:18
    at Layer.handle [as handle_request] (C:\Users\Dell\Desktop\GI2-S1\Securité\Tps\Ateliers-OWASP\FormationOWASP\owasp-workshops\juice-shop\node_modules\express\lib\router\layer.js:95:5)
    at trim_prefix (C:\Users\Dell\Desktop\GI2-S1\Securité\Tps\Ateliers-OWASP\FormationOWASP\owasp-workshops\juice-shop\node_modules\express\lib\router\index.js:328:13)
    at C:\Users\Dell\Desktop\GI2-S1\Securité\Tps\Ateliers-OWASP\FormationOWASP\owasp-workshops\juice-shop\node_modules\express\lib\router\index.js:286:9
    at Function.process_params (C:\Users\Dell\Desktop\GI2-S1\Securité\Tps\Ateliers-OWASP\FormationOWASP\owasp-workshops\juice-shop\node_modules\express\lib\router\index.js:346:12)
```

```
// ================Correction A05 exo1-Gestion des erreurs=================

  // Middleware global pour la gestion des erreurs
  app.use((err: Error, req: Request, res: Response, next: NextFunction) => {
    // Log complet côté serveur (utile pour le debugging interne)
    console.error(err.stack)

    // Réponse générique côté client (sans stack trace)
    res.status(500).json({
      error: 'Erreur interne du serveur. Veuillez réessayer plus tard.'
    })
})
```

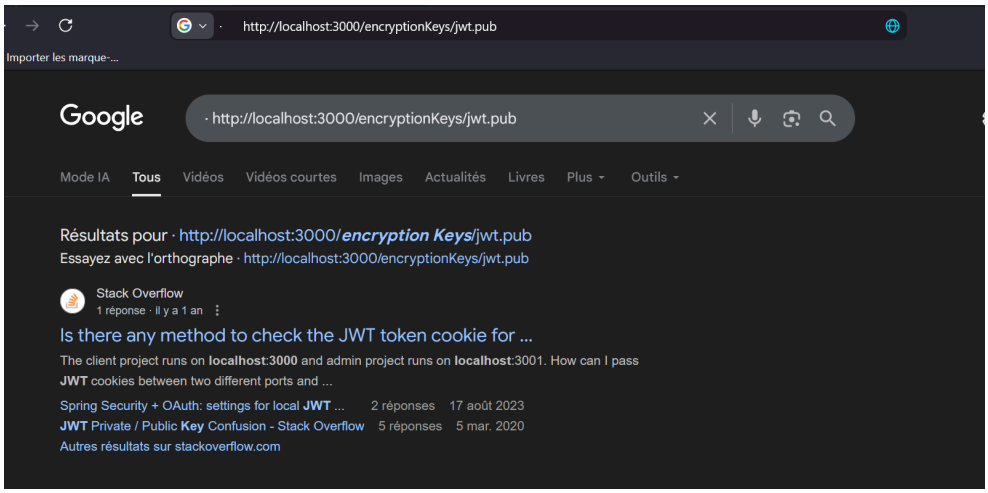
2. Fichiers Sensibles Exposés

Reproduction

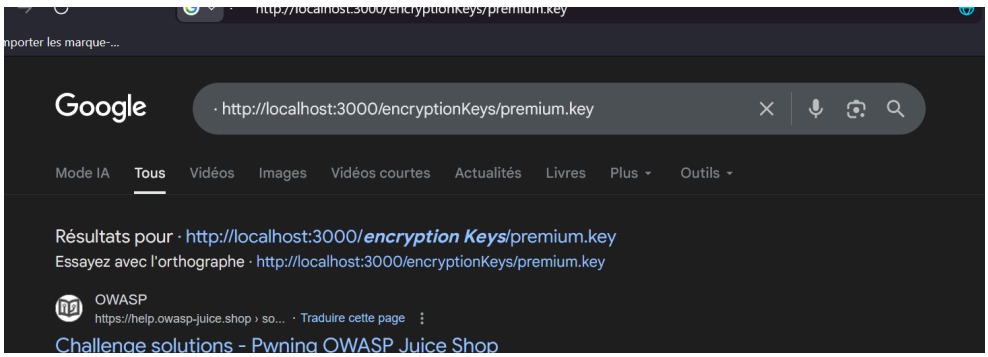
Dans votre navigateur, essayez d'accéder à ces URLs :
1) http://localhost:3000/ftp/legal.md
👉 Ce que c'est
C'est un fichier interne du serveur, placé dans un dossier qui simule un dossier FTP.
👉 À quoi il sert dans Juice Shop
C'est un fichier Markdown (legal.md) qui n'est normalement pas censé être directement accessible depuis le navigateur.



2) http://localhost:3000/encryptionKeys/jwt.pub
👉 Ce que c'est
C'est la clé publique JWT utilisée pour vérifier les signatures des tokens JWT du site.
👉 À quoi elle sert
Le serveur signe les JWT avec une clé privée
Le client peut vérifier le JWT avec la clé publique



3) http://localhost:3000/encryptionKeys/premium.key
👉 Ce que c'est
C'est une clé privée utilisée dans Juice Shop pour certains challenges (ex : déchiffrer un fichier premium, ou manipuler un abonnement).
👉 Pourquoi c'est grave
Une clé privée ne doit JAMAIS être accessibles depuis le web.
Ici, Juice Shop simule une mauvaise configuration :



Correction :

```
// vuln-code-snippet start directoryListingChallenge accessLogDisclosureChallenge
/* /ftp directory browsing and file download */ // vuln-code-snippet neutral-line directoryListingChallenge
// app.use('/ftp', serveIndexMiddleware, serveIndex('ftp', { icons: true })) // vuln-code-snippet vuln-line directoryListingChallenge
// app.use('/ftp(?!/quarantine)/:file', servePublicFiles()) // vuln-code-snippet vuln-line directoryListingChallenge
// app.use('/ftp/quarantine/:file', serveQuarantineFiles()) // vuln-code-snippet neutral-line directoryListingChallenge
app.use('/ftp', (req, res) => res.status(403).send('Accès interdit'));
```

Faites de même avec les autres fichiers/dossiers
Vous devez aussi bloquer les accès aux fichier sql, zip, csv, etc, .env, .git, etc.
Si vous utilisez un reverse proxy (nginx, Apache), il faut l'utiliser pour bloquer l'accès à ces fichiers (fichier htaccess).

On ajout des mislwares ;

```
  //  BLOQUER L'ACCÈS AUX FICHIERS SENSIBLES (sql, zip, csv, env, etc.)
app.use((req, res, next) => {
  const forbiddenExtensions = [
    '.sql', '.zip', '.csv', '.tar', '.gz',
    '.env', '.key', '.pem', '.crt', '.gitignore'
  ]
  if (forbiddenExtensions.some(ext => req.path.endsWith(ext))) {
    return res.status(403).send('Accès interdit')
  }
  next()
})
```

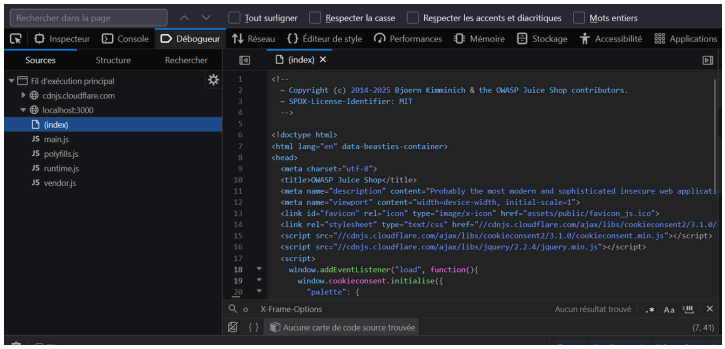
```
// BLOQUER L'ACCÈS AUX FICHIERS SENSIBLES PAR NOM
const forbiddenFiles = [
  '/.env',
  '/.git/config',
  '/jwt.pub',
  '/premium.key',
  '/database.sql'
]
for (const file of forbiddenFiles) {
  app.get(file, (req, res) => res.status(403).send('Accès interdit'))
}
```

1.Headers de Sécurité Manquants

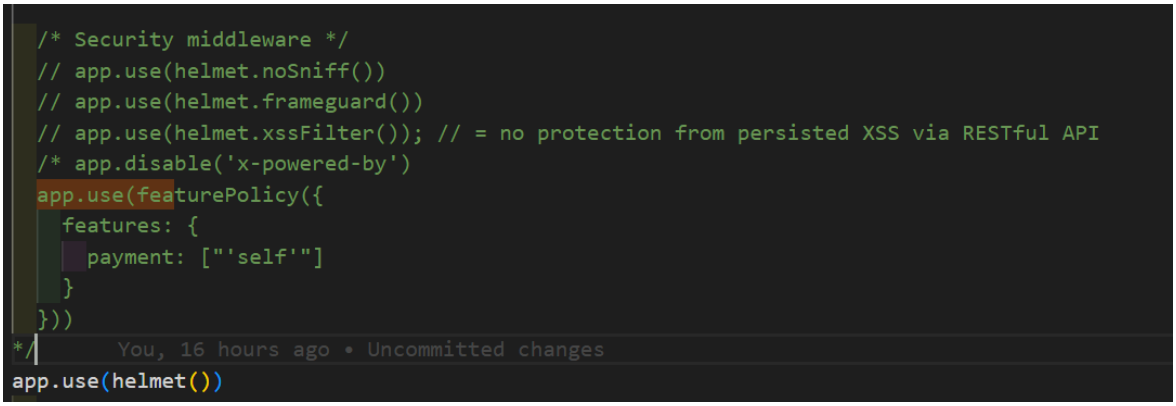
· Cliquez sur index.html
· Vérifiez si les headers suivants sont absents :

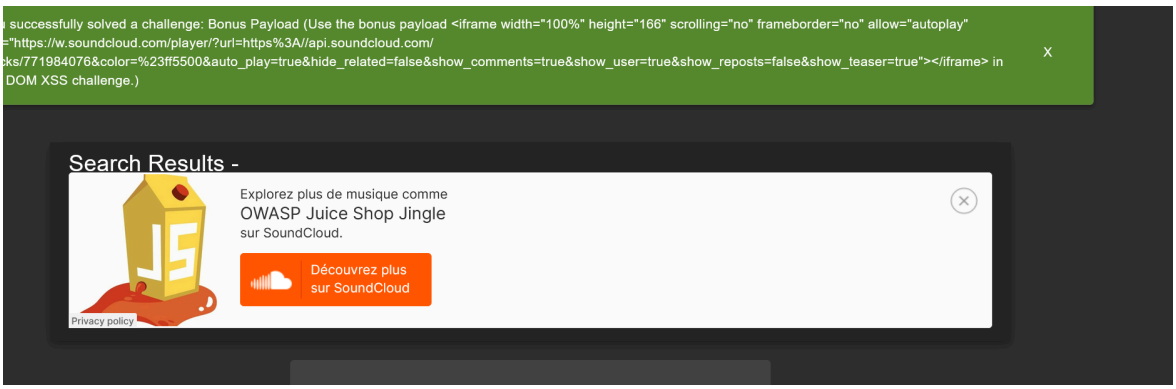
o X-Frame-Options: DENY (empêche l'intégration du site dans un iframe et bloque les attaques de type clickjacking)
oContent-Security-Policy
o X-Content-Type-Options: nosniff (Empêche le navigateur d'interpréter le type des fichiers lui même)
o X-XSS-Protection (Active la protection XSS côté client pour les anciens navigateurs)
o Strict-Transport-Security (HSTS) (Force le navigateur à n'utiliser que HTTPS (protection MITM)).
o Referrer-Policy (Contrôle les informations transmises dans le header Referer)
o Permissions-Policy (anciennement Feature-Policy)
o Content-Security-Policy (CSP) (pour bloquer les attaques XSS)
o Cross-Origin Resource Policy (CORP) et Cross-Origin Opener Policy (COOP)
oCache-Control / Pragma / Expires (Empêche la mise en cache de données sensibles (login, API).)
o X-Powered-By (Évite d'exposer la techno utilisée (Express))

Oui



On a désactivé Helmet pour rendre l'application vulnérable et pouvoir tester l'absence des protections.





En résumé, ces headers de sécurité sont des barrières essentielles qui protègent ton application contre de nombreuses attaques courantes : clickjacking, XSS, fuite de données sensibles, exploitation des permissions du navigateur, et attaques MITM.
Les désactiver (comme avec Helmet dans ton test) rend l'application vulnérable, ce qui est utile pour apprendre et tester, mais en production, ils doivent toujours être activés pour assurer la sécurité des utilisateurs et des données.