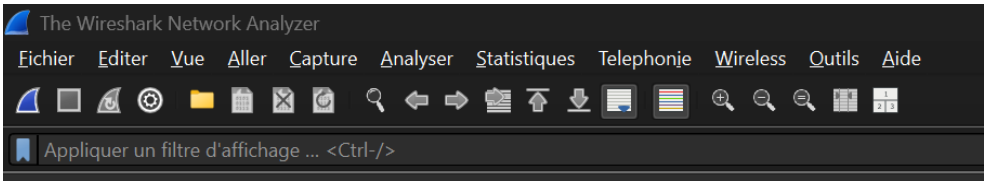
Rapport
OWASP: Atelier 02

SABRI OMAIMA

Atelier 02 :

1.Transmission de Données Sensibles en Clair (HTTP vs HTTPS)

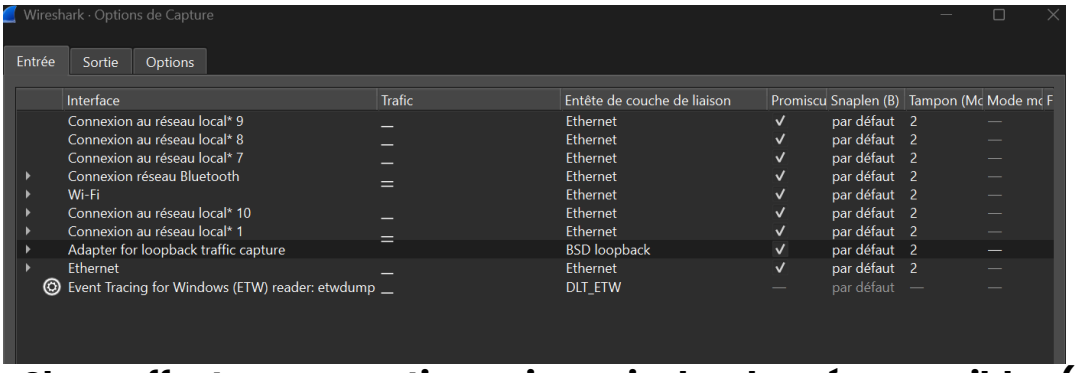
1.Démarrez un outil d'analyse réseau comme Wireshark ou Burp Suite.1.



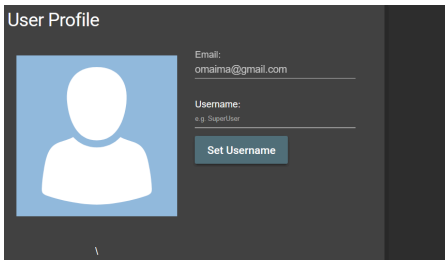
2. Configurez-le pour capturer le trafic de localhost:3000.

http://localhost:3000 Le trafic ne sort pas sur ton Wi-Fi.
Il reste à l'intérieur de ton PC.

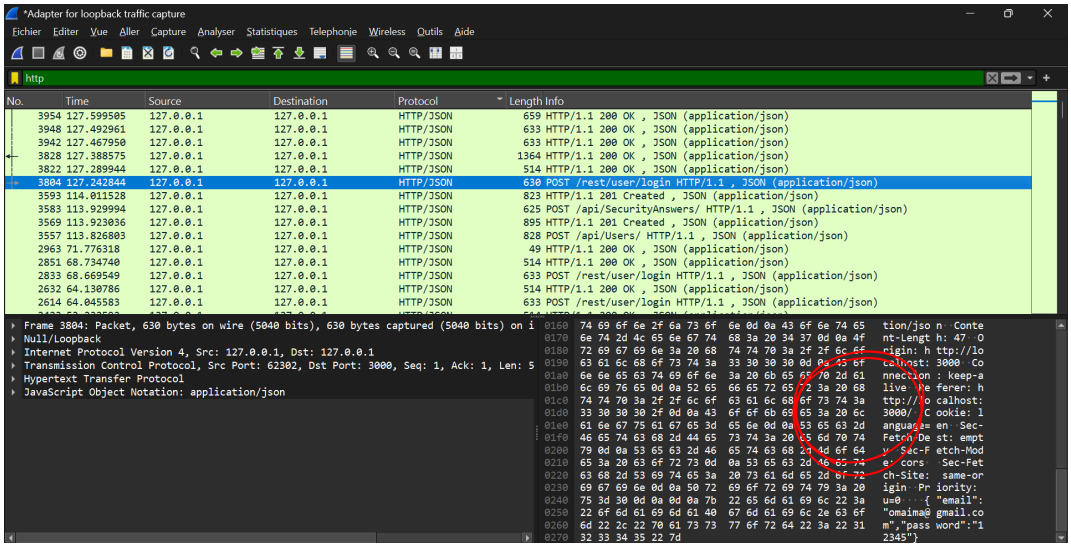
- localhost = communication interne
- Le Wi-Fi ne voit rien de ça
- Wireshark a besoin de l'interface NPcap Loopback Adapter (ou équivalent)
- C'est la seule interface qui capture le trafic vers 127.0.0.1 (port 3000, 4200, etc.)



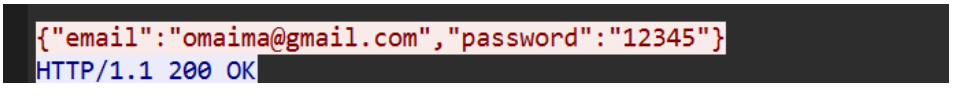
3.Sur Juice Shop, effectuez une action qui envoie des données sensibles (ex: connexion utilisateur).



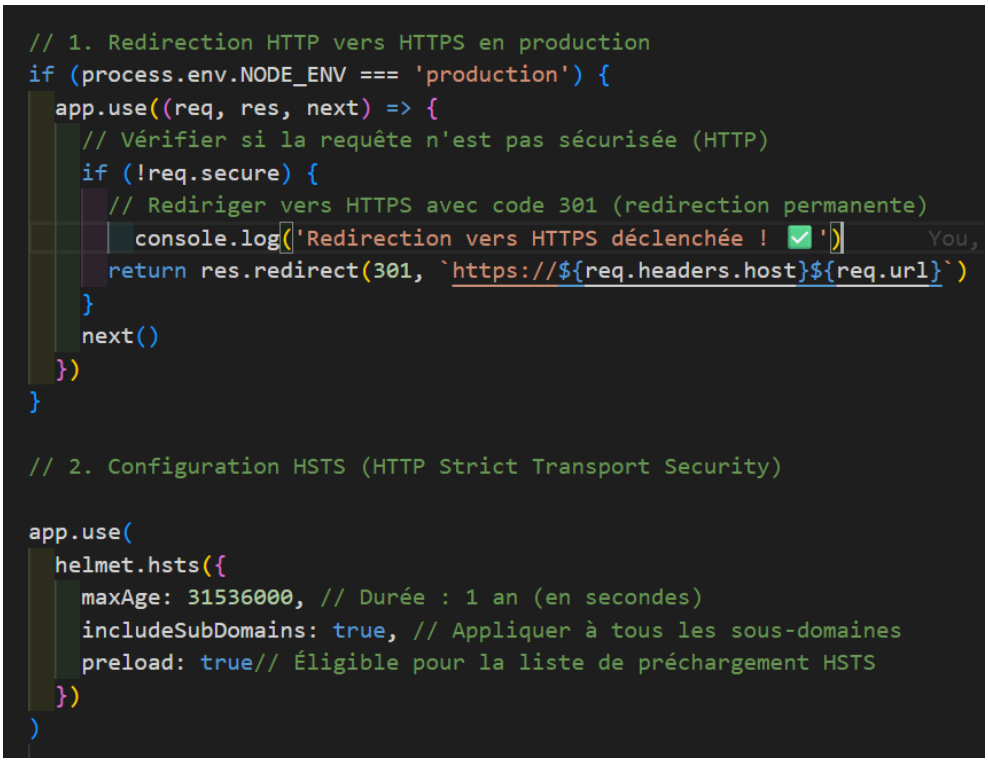
4.Dans le captureur, filtrez les paquets HTTP et recherchez les requêtes POST vers /rest/user/login.



5.Observez que les identifiants (email et mot de passe) sont visibles en clair dans la requête POST si HTTP est utilisé.



Correction Technique :



```
// 1. Redirection HTTP vers HTTPS en production
if (process.env.NODE_ENV === 'production') {
  app.use((req, res, next) => {
    // Vérifier si la requête n'est pas sécurisée (HTTP)
    if (!req.secure) {
      // Rediriger vers HTTPS avec code 301 (redirection permanente)
      console.log('Redirection vers HTTPS déclenchée ! ✅')
      return res.redirect(301, `https://${req.headers.host}${req.url}`)
    }
    next()
  })
}

// 2. Configuration HSTS (HTTP Strict Transport Security)

app.use(
  helmet.hsts({
    maxAge: 31536000, // Durée : 1 an (en secondes)
    includeSubDomains: true, // Appliquer à tous les sous-domaines
    preload: true// Éligible pour la liste de préchargement HSTS
  })
)
```

1-Vulnérabilité Identifiée
Nature de la faille :

- L'application Juice Shop utilise le protocole HTTP (sans chiffrement) pour transmettre les données entre le client et le serveur.
- Les identifiants de connexion (email et mot de passe) transitent en clair sur le réseau.

2-Preuve de la vulnérabilité :

- À l'aide de Wireshark, j'ai capturé le trafic réseau lors d'une connexion utilisateur.
- En filtrant les paquets HTTP (http filter), j'ai pu observer une requête POST /rest/user/login.
- Dans le corps de cette requête, les identifiants étaient visibles en texte brut :

3-Impact de cette vulnérabilité :

- Attaque Man-in-the-Middle (MITM) : Un attaquant sur le même réseau (WiFi public, réseau d'entreprise compromis) peut intercepter et lire toutes les données transmises.
- Vol d'identifiants : Les mots de passe peuvent être capturés et réutilisés pour accéder aux comptes utilisateurs.
- Violation du RGPD : La transmission de données personnelles sans chiffrement constitue une violation des obligations de sécurité.
- Perte de confiance : Les utilisateurs perdent confiance dans la sécurité de l'application.

4-Correction Appliquée
Principe de la correction : Pour protéger les données en transit, j'ai mis en œuvre deux mécanismes complémentaires :
A) Redirection HTTP → HTTPS (en production)

- Cette middleware vérifie si la requête entrante utilise HTTP (!req.secure).
- Si c'est le cas, elle effectue une redirection permanente (code 301) vers l'URL HTTPS équivalente.

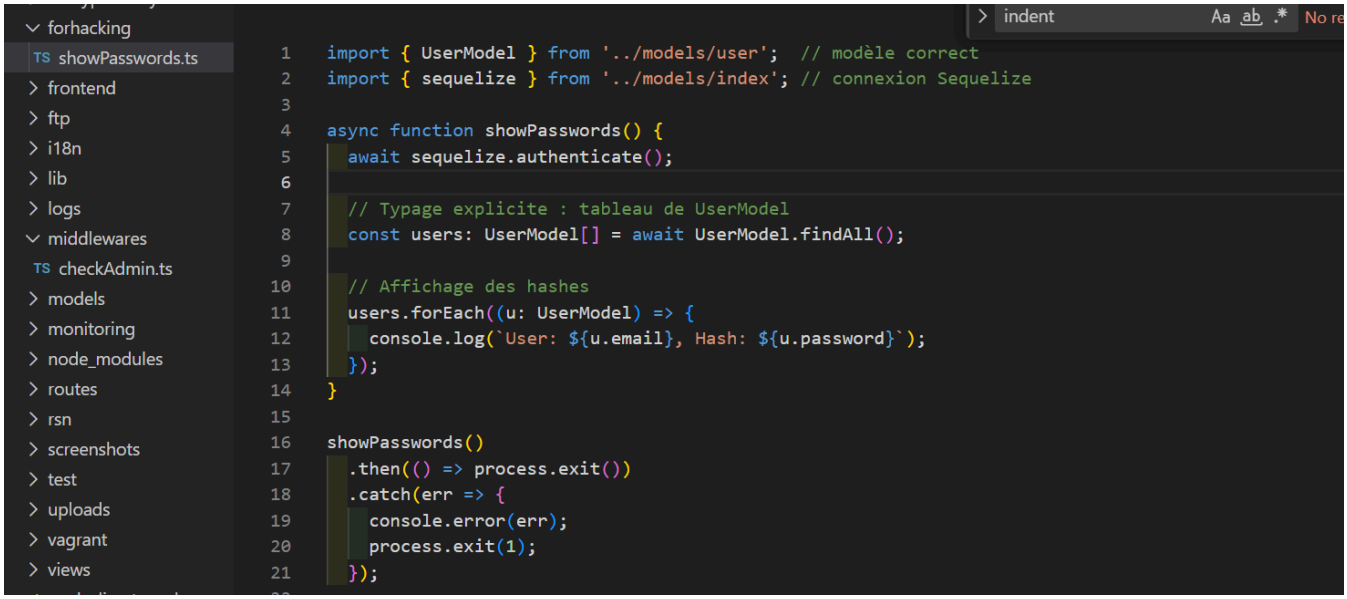
B) Implémentation de HSTS (HTTP Strict Transport Security)
- HSTS est un **header de sécurité HTTP** qui oblige les navigateurs à n'utiliser que HTTPS pour communiquer avec le serveur.

2.Hachage des Mots de Passe Faible ou Absent

1.Accédez à la base de données Juice Shop (SQLite par défaut), et exécutez la requête : SELECT * FROM Users;

```
PS C:\Users\Dell\Desktop\GI2-S1\Securité\Tps\Ateliers-OWASP\FormationOWASP\owasp-workshops\juice-shop> sqlite3 data/juiceshop.sqlite
SQLite version 3.51.0 2025-11-04 19:38:17
Enter ".help" for usage hints.
sqlite> SELECT * FROM Users;
1||admin@juice-sh.op|0192023a7bbd73250516f069df18b500|admin|||assets/public/images/uploads/defaultAdmin.png||1|2025-11-28 18:34:17.99
7 +00:00|2025-11-28 18:34:17.997 +00:00|
2||jim@juice-sh.op|e541ca7ecf72b8d1286474fc613e5e45|customer|||assets/public/images/uploads/default.svg||1|2025-11-28 18:34:17.997 +0
0:00|2025-11-28 18:34:17.997 +00:00|
3||bender@juice-sh.op|0c36e517e3fa95aabf1bbffc6744a4ef|customer|||assets/public/images/uploads/default.svg||1|2025-11-28 18:34:17.998
 +00:00|2025-11-28 18:34:17.998 +00:00|
4|bkimminich|bjoern.kimminich@gmail.com|6edd9d726cbdc873c539e41ae8757b8c|admin|||assets/public/images/uploads/defaultAdmin.png||1|202
5-11-28 18:34:17.998 +00:00|2025-11-28 18:34:17.998 +00:00|
5||ciso@juice-sh.op|861917d5fa5f1172f931dc700d81a8fb|deluxe|d715c2c75d4a42d3825a050e0a0163c1959b51165373f17bd8eed7b1e05bf20d||assets/
```

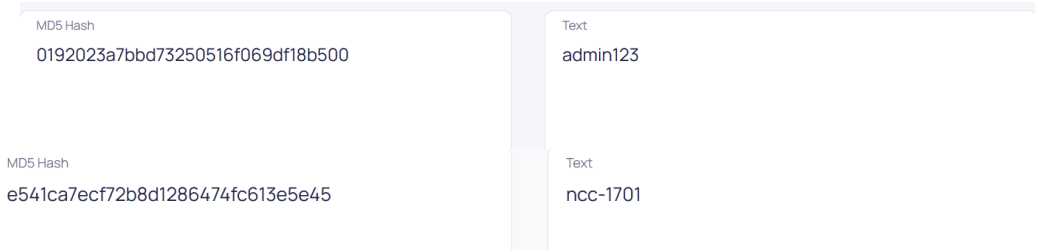
2.Ou bien créez un script « forhacking/showPasswords.ts » qui récupère tous les utilisateurs :



3.Exécutez-le : « npx ts-node .\forhacking\showPasswords.ts » et vous aurez ces lignes :
4.Observez que les mots de passe ont l'air d'être hachés avec un algorithme faible (MD5).

```
PS C:\Users\Dell\Desktop\GI2-S1\Securité\Tps\Ateliers-OWASP\FormationOWASP\owasp-workshops\juice-shop> npx ts-node .\forhac
king\showPasswords.ts
User: admin@juice-sh.op, Hash: 0192023a7bbd73250516f069df18b500
User: jim@juice-sh.op, Hash: e541ca7ecf72b8d1286474fc613e5e45
User: bender@juice-sh.op, Hash: 0c36e517e3fa95aabf1bbffc6744a4ef
User: bjoern.kimminich@gmail.com, Hash: 6edd9d726cbdc873c539e41ae8757b8c
User: ciso@juice-sh.op, Hash: 861917d5fa5f1172f931dc700d81a8fb
User: support@juice-sh.op, Hash: 3869433d74e3d0c86fd25562f836bc82
User: morty@juice-sh.op, Hash: f2f933d0bb0ba057bc8e33b8ebd6d9e8
User: mc.safesearch@juice-sh.op, Hash: b03f4b0ba8b458fa0acdc02cdb953bc8
User: J12934@juice-sh.op, Hash: 3c2abc04e4a6ea8f1327d0aae3714b7d
User: wurstbrot@juice-sh.op, Hash: 9ad5b0492bbe528583e128d2a8941de4
User: amy@juice-sh.op, Hash: 030f05e45e30710c3ad3c32f00de0473
User: bjoern@juice-sh.op, Hash: 7f311911af16fa8f418dd1a3051d6810
User: bjoern@owasp.org, Hash: 9283f1b2e9669749081963be0462e466
User: accountant@juice-sh.op, Hash: 963e10f92a70b4b463220cb4c5d636dc
User: uvogin@juice-sh.op, Hash: 05f92148b4b60f7dacd04cceebb8f1af
User: demo, Hash: fe01ce2a7fbac8fafaed7c982a04e229
User: john@juice-sh.op, Hash: 00479e957b6b42c459ee5746478e4d45
User: emma@juice-sh.op, Hash: 402f1c4a75e316afec5a6ea63147f739
User: stan@juice-sh.op, Hash: e9048a3f43dd5e094ef733f3bd88ea64
User: ethereum@juice-sh.op, Hash: 2c17c6393771ee3048ae34d6b380c5ec
User: testing@juice-sh.op, Hash: b616a64605a07941fbd31868aea3b54b
PS C:\Users\Dell\Desktop\GI2-S1\Securité\Tps\Ateliers-OWASP\FormationOWASP\owasp-workshops\juice-shop>
```

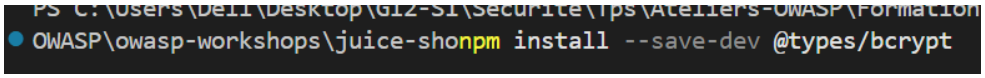
5.Utilisez un des outils en lignes pour décrypter quelques-uns



Correction Technique :

1.Utilier un algorithme de hashage plus fort (à savoir Bcrypt ou autre)

Commençons par l'installer dans le projet : npm install --save-dev @types/bcrypt
Puis, modifiez le fichier « insecurity.ts »

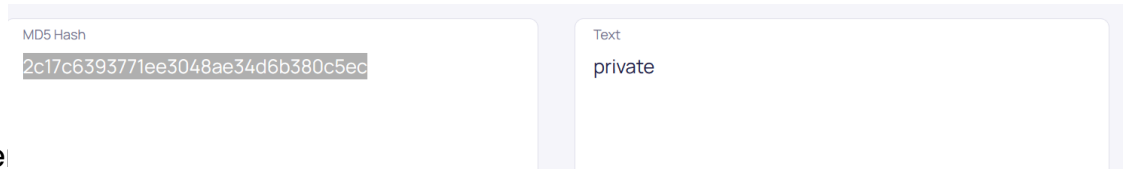



```
// export const hash = (data: string) => crypto.createHash('md5').update(data).digest('hex')
const saltRounds = 12
export const hash = (data: string): string => bcrypt.hashSync(data, saltRounds)
```

2. Créez un nouvel utilisateur, et réaffichez les utilisateurs pour voir les mots de passe. Vous allez remarquer que les mots de passe en changé. Essayez de les décrypter, ce n'est plus possible.
3.Maintenant les authentifications ne fonctionnent plus, car la vérification dans le fichier ligin.ts repose encore sur MD5. Pour corriger, ajouter la fonction compareHash dans le fichier « insecurity.ts » :

```
export const compareHash = (plain: string, hashed: string): boolean => {
  try {
    return bcrypt.compareSync(plain, hashed)
  } catch {
    return false
  }
}
```

4. Modifier ensuite le fichier « login.ts » remplacer le bloc suivant :
Apres les changements :



1. Vulné
Nature de la faille :

- L'application Juice Shop utilise l'algorithme MD5 pour hacher les mots de passe des utilisateurs avant de les stocker en base de données.
- MD5 est un algorithme cryptographique obsolète et vulnérable, considéré comme cassé depuis le début des années 2000.
- Absence de sel (salt) automatique : Les mêmes mots de passe produisent toujours les mêmes hashs, facilitant les attaques par rainbow tables.
- Rapidité excessive : MD5 peut générer des milliards de hashs par seconde, permettant des attaques par force brute très rapides.

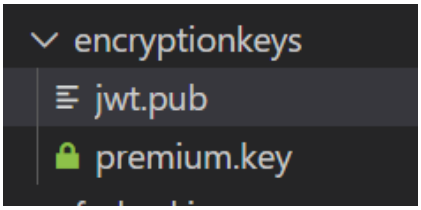
2. Impact de cette vulnérabilité

- Vol massif de comptes : Un attaquant avec accès à la base peut récupérer tous les mots de passe en minutes.
- Réutilisation d'identifiants : 65% des utilisateurs réutilisent leurs mots de passe → compromission d'autres comptes (email, banque).

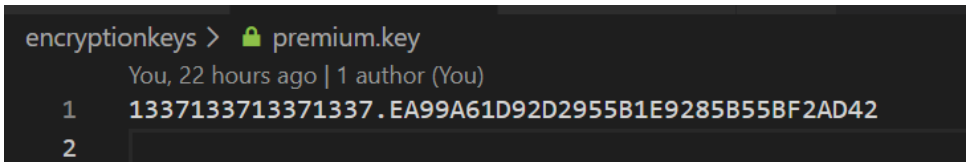
3. Correction Appliquée
Principe de la correction :
Remplacer MD5 (rapide, obsolète) par bcrypt (lent, sécurisé avec sel automatique).

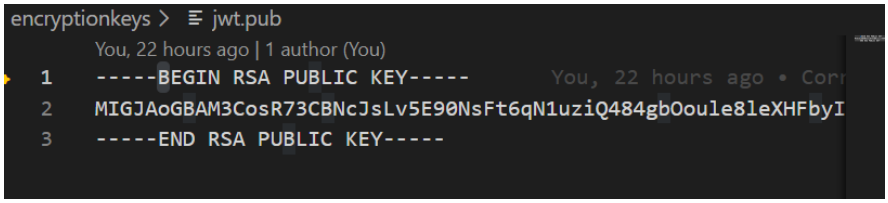
3.Secrets Codés en Dur dans le Code

1.Dans Juice Shop, explorez le répertoire /data/encryptionKeys, le fichier insecurity.ts cyber.yml dans le code source.

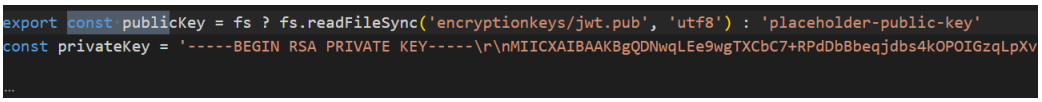


2.Recherchez des chaînes de caractères qui ressemblent à des « clés », « tokens » ou « mots de passe ». Avec ces secrets, on peut forger un JWT valide





3.Regardez aussi le fichier « insecurity.ts » et vous allez localiser la « clé privée » en clair.



Correction technique:

Solution 1 : fichier local non versionné « .env » contenant vos secrets.

Un fichier non versionné, c'est un fichier qui n'est pas poussé sur Git (ni GitHub, ni GitLab, ni autre dépôt).
Il reste local sur ta machine, volontairement ignoré par Git, pour éviter d'exposer des informations sensibles.

On met dans le .env :
- les clés privées
- les tokens secrets
- les API keys
- les passwords
- les JWT secrets

On NE met pas dans le .env :
- les clés publiques (pas obligatoires, mais parfois oui si utilisées par Node)
- les données non sensibles

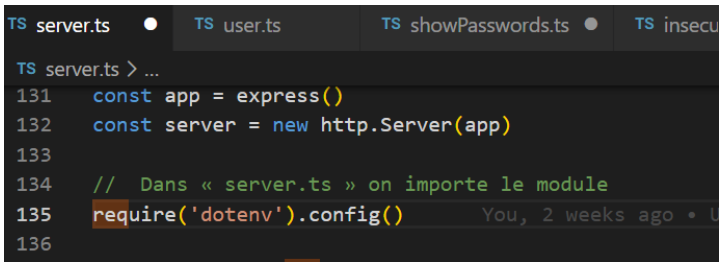
Le module dotenv sert à charger automatiquement les variables d'environnement depuis un fichier .env dans ton projet Node.js.
C'est un module petit mais extrêmement important pour sécuriser et organiser les secrets (API keys, password, tokens, clés privées…).

On installe le module dotenv :

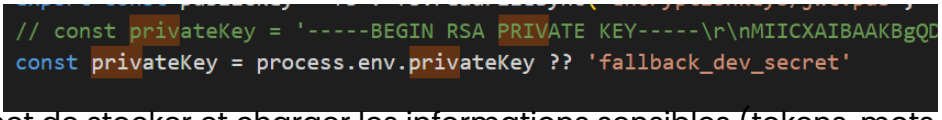
```
PS C:\Users\Dell\Desktop\GI2-S1\Securité\Tps\Ateliers-OWASP\FormationOWASP\owasp-workshops\juice-shop> npm install dotenv

added 1 package, and audited 2138 packages in 2m
```

Dans « server.ts » on importe le module :



et ON récupére la clé dans insecurity.ts.



👉 dotenv permet de stocker et charger les informations sensibles (tokens, mots de passe, clés, configs) dans un fichier .env au lieu de les mettre dans le code.

👉 On l'utilise pour sécuriser le projet, éviter d'exposer des secrets dans GitHub, et rendre la configuration facile à changer sans modifier le code.

Solution 2 : Utilisez un service cloud pour vos secrets.

Sur AWS (Amazon Cloud) par exemple, créez un secret nommé par exemple juice-shop-keys, et dedans mettez vos clés :

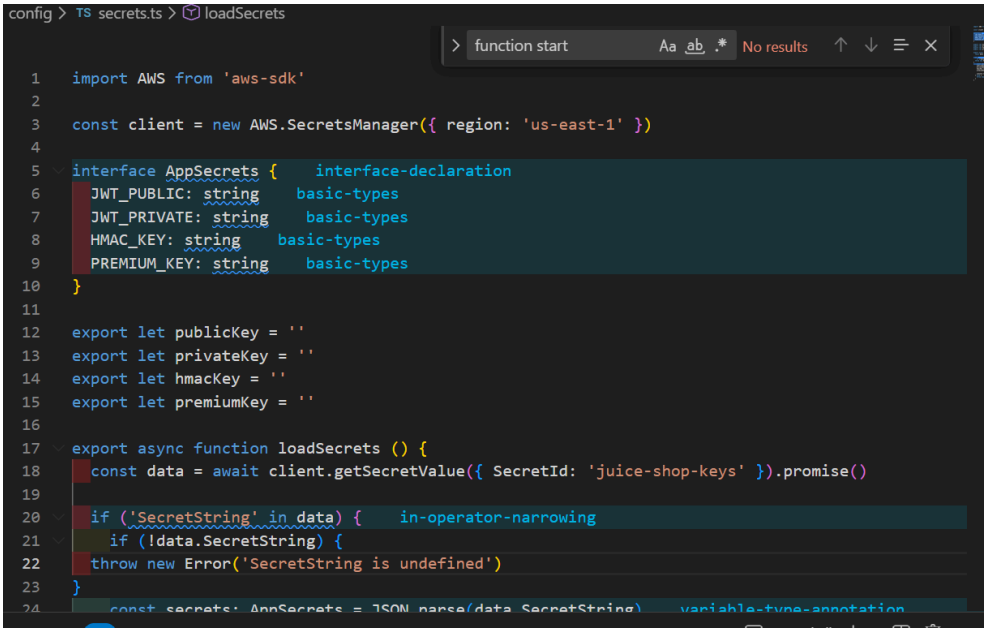
Key	Value
JWT_PUBLIC	contenu de jwt.pub
JWT_PRIVATE	contenu de la clé RSA
HMAC_KEY	pa4qacea4VK9t9nGv7yZtwmj
PREMIUM_KEY	contenu de premium.key

Le module aws (ou plus précisément aws-sdk / @aws-sdk/* dans les projets modernes) sert simplement à permettre à ton application Node.js d'accéder aux services Amazon Web Services, par exemple :

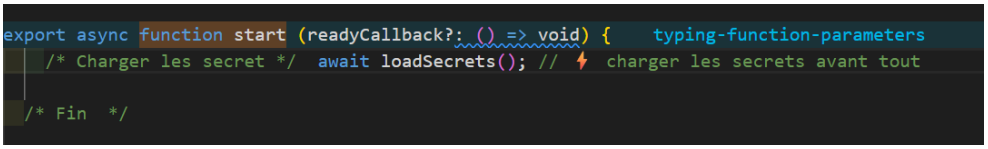
- AWS Secrets Manager → pour récupérer tes clés secrètes stockées dans le cloud
- AWS S3 → pour accéder à des fichiers
- AWS DynamoDB / RDS → pour accéder à des bases de données

Dans notre cas on va utiliser AWS Secrets Manager pour lire les clés secrètes.
Donc le fichier secrets.ts va importer le module AWS, se connecter à AWS Secrets Manager, et récupérer les secrets.

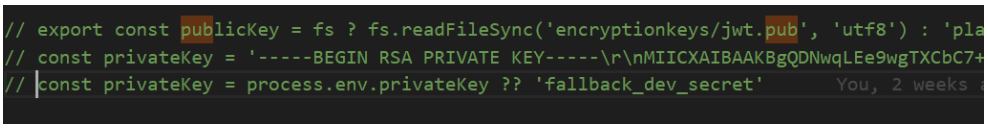
Créez un fichier « config/secrets.ts » qui va récupérer ces clés, mais avant, installez le module « aws » :



Et lancer la fonction avant le lancement du serveur :



N'oubliez pas de commenter ou supprimer les autres lignes contenant les déclarations des mots de passe



Différence principale entre solution 1 et 2 :

- .env → local, simple, pour développement ou petits projets.
- AWS Secrets Manager → centralisé, sécurisé, pour production ou projets où plusieurs développeurs ou serveurs doivent accéder aux secrets.

Quand choisir quoi

- .env : Pour des tests, développement local, projets personnels, ou si tu veux juste éviter de mettre des secrets en dur dans le code.
- AWS Secrets Manager : Pour un projet en production, avec plusieurs environnements, plusieurs serveurs ou collaborateurs, ou si tu veux une sécurité professionnelle et centralisée.

Résumé : Gestion sécurisée des secrets

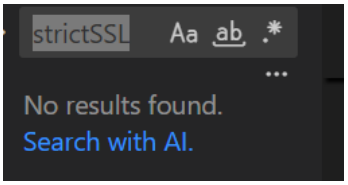
- Ne jamais mettre les secrets dans le code : mots de passe, clés API, tokens, etc.
- Utiliser des variables d'environnement ou un gestionnaire de secrets dédié (ex. HashiCorp Vault, AWS Secrets Manager, Azure Key Vault).
- Protéger les fichiers locaux contenant des secrets (.env) avec .gitignore pour éviter qu'ils soient versionnés.
- Scanner le code régulièrement avec des outils SAST/SCA (GitHub Advanced Security, GitLeaks, TruffleHog) pour détecter tout secret accidentel dans le dépôt.

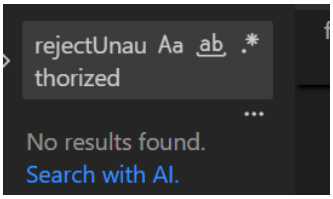
4.Validation des Certificats SSL Désactivée

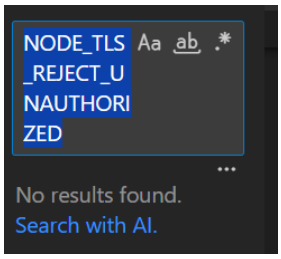
Type de faille : “MITM via désactivation de la vérification TLS/SSL”
Cette faille s’appelle :
👉 Attaque Man-in-the-Middle (MITM) due à la désactivation de la vérification des certificats SSL/TLS.
Elle apparaît lorsque le développeur écrit quelque chose comme : **rejectUnauthorized: false** ou **strictSSL: false** ou encore **NODE_TLS_REJECT_UNAUTHORIZED=0**
Ces options désactivent la vérification du certificat HTTPS, ce qui permet à un attaquant de présenter un certificat faux (auto-signé) → et la connexion sera acceptée quand même.

📩 Résultat : le trafic HTTPS peut être intercepté, lu, modifié → MITM complet.

Résultat de recherche : Aucune occurrence trouvée.







Conclusion :Le code source de Juice Shop ne contient pas de désactivation explicite de la validation SSL/TLS. Cette bonne pratique est respectée dans le projet.

Impact de cette vulnérabilité (si elle existait)

- Attaque Man-in-the-Middle : Un attaquant peut intercepter et modifier les communications HTTPS.
- Usurpation d'identité : L'attaquant peut se faire passer pour le serveur légitime avec un certificat auto-signé.
- Vol de données : Identifiants, tokens, données personnelles peuvent être capturés.
- Injection de code : L'attaquant peut modifier les réponses du serveur pour injecter du code malveillant.

Tip Métier :

Ne désactivez jamais la validation des certificats en production. Cette pratique annule toute la sécurité apportée par HTTPS. En développement, si vous avez des problèmes de certificat, configurez correctement un environnement avec des certificats de test valides signés par votre propre autorité de certification (CA) interne, et importez la racine de confiance dans votre magasin de certificats de test. N'utilisez jamais process.env.NODE_TLS_REJECT_UNAUTHORIZED = '0'.

5.Et pour Spring boot et ReactJS ...

Tip Métier :
Le chiffrement HTTPS est obligatoire en production, mais la configuration diffère selon l'environnement.
En Développement Local :

- Utilisez des certificats auto-signés créés avec OpenSSL pour tester HTTPS localement.
- Commande rapide : openssl req -x509 -newkey rsa:2048 -nodes -keyout key.pem -out cert.pem -days 365
- Acceptez les avertissements du navigateur (normal pour certificats auto-signés).

En Production :

- Ne JAMAIS utiliser de certificats auto-signés → Les navigateurs afficheront des avertissements aux utilisateurs.
- Utilisez Let's Encrypt (gratuit, automatique) ou un certificat commercial.
- Configurez HTTPS sur le reverse proxy (Nginx, Apache) plutôt que dans l'application Node.js directement.
- TLS 1.2 minimum (idéalement TLS 1.3) → Désactivez TLS 1.0 et 1.1 (obsolètes).
- Activez HSTS (HTTP Strict Transport Security) pour forcer HTTPS.
- Renouvelement automatique : Let's Encrypt expire tous les 90 jours → Automatisez avec Certbot.

Architecture Recommandée en Production :
Internet → Nginx (HTTPS + certificat) → Application Node.js (HTTP local)

- Nginx gère SSL/TLS (terminaison SSL)
- L'application communique en HTTP avec Nginx (réseau local sécurisé)
- Plus performant et plus facile à maintenir

Outils de Test :

- SSL Labs (1.Configurez votre service Nginx ou équivalent comme suit :) → Note de A à F pour votre configuration SSL
- Objectif : Obtenir un A ou A+
- Testez régulièrement (mensuellement) pour détecter les faiblesses

Erreurs Courantes à Éviter :

- ❌ Certificat expiré (surveillez les dates d'expiration)
- ❌ Mixed Content (charger du HTTP sur une page HTTPS)
- ❌ Chaîne de certificats incomplète
- ❌ Protocoles obsolètes activés (SSLv3, TLS 1.0)