TP Sécurité Web – JWT & HTTPS

# 1. Présentation du projet

Le projet consiste à mettre en place une application Express avec une authentification via JSON Web Tokens (JWT).

Objectifs du TP :

* - Créer un site web vulnérable à une faille d’authentification liée aux JWT
* - Mettre en place un serveur HTTPS
* - Implémenter une remédiation sécurisée
* - Démontrer les deux cas avec captures d’écran

# 2. Présentation de la faille

La version initiale du projet contient une vulnérabilité critique :  
- Lorsqu’un utilisateur se connecte, le backend lui renvoie un JWT avec le rôle 'admin', sans vérification.  
- La vérification du token utilise `jwt.decode(token)` qui ne vérifie pas la signature.  
- Un attaquant peut donc modifier le contenu du token et accéder à une route protégée.

# 3. Preuve d’exploitation

Étapes :

* - Connexion avec un nom quelconque
* - Récupération du token JWT via DevTools
* - Modification du token sur jwt.io
* - Injection du token falsifié dans localStorage
* - Accès accordé à la route /dashboard.html

# 4. Mise en HTTPS avec Ngrok

Let’s Encrypt étant contraignant localement, j’ai utilisé Ngrok pour créer un tunnel HTTPS :  
Commande utilisée : `ngrok http 3000`  
Exemple de lien obtenu : https://xxxxx.ngrok-free.app  
Cela permet de respecter la contrainte HTTPS du sujet sans passer par un VPS.

# 5. Remédiation : sécurisation du backend

Avant :

jwt.decode(token); // ❌ pas de vérification

Après :

jwt.verify(token, process.env.JWT\_SECRET); // ✅ signature vérifiée

Et dans admin.js : utilisation du middleware sécurisé.

# 6. Test avec token falsifié (après correction)

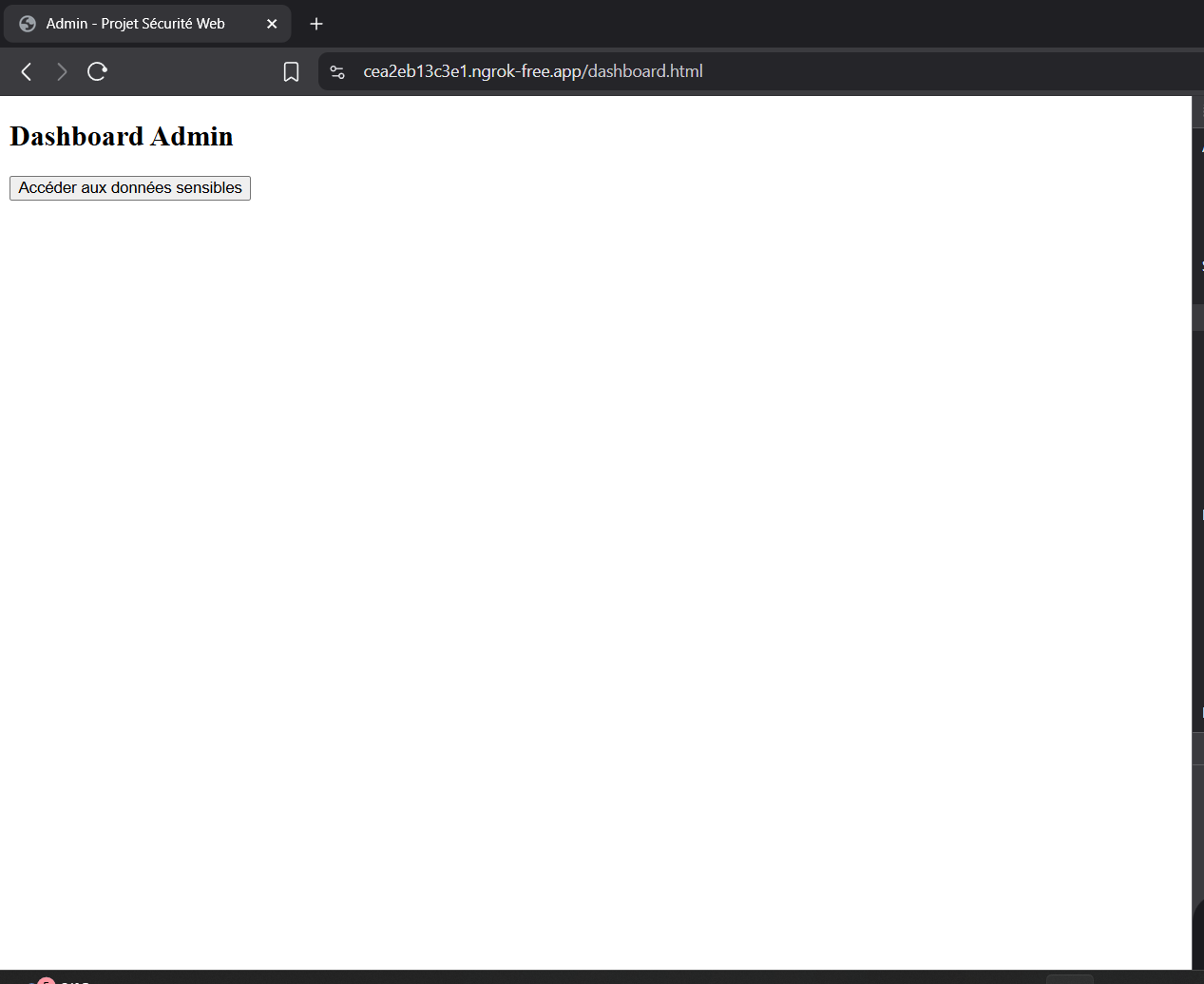
Un token modifié manuellement avec une signature invalide est désormais rejeté par le backend avec le message :  
{ "error": "Token invalide ou expiré" }

# 7. Conclusion

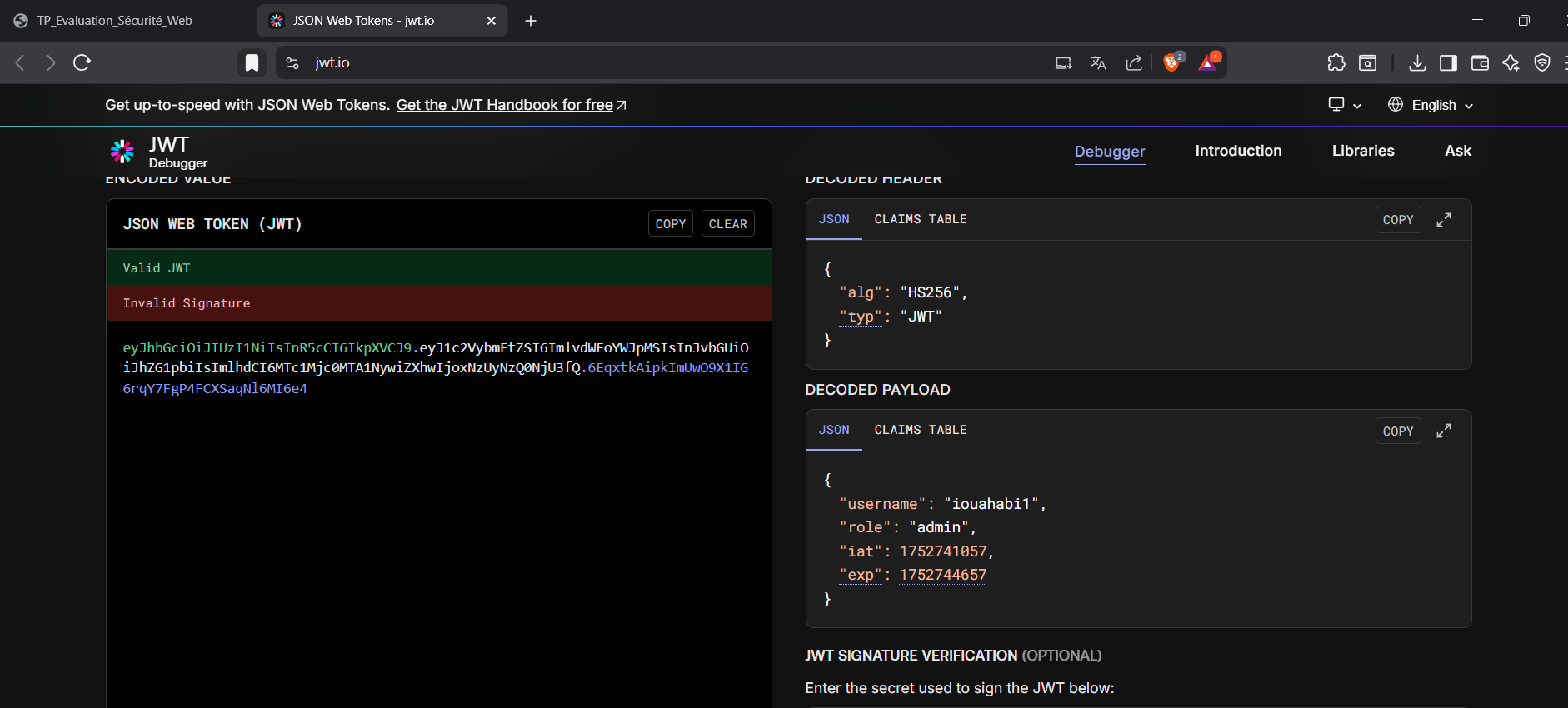
La faille de validation JWT a été exploitée avec succès. La remédiation via `jwt.verify()` empêche toute tentative de modification frauduleuse du token. Le tout a été testé en HTTPS via Ngrok pour respecter les contraintes du TP.

# 📸 Annexes – Captures d'écran

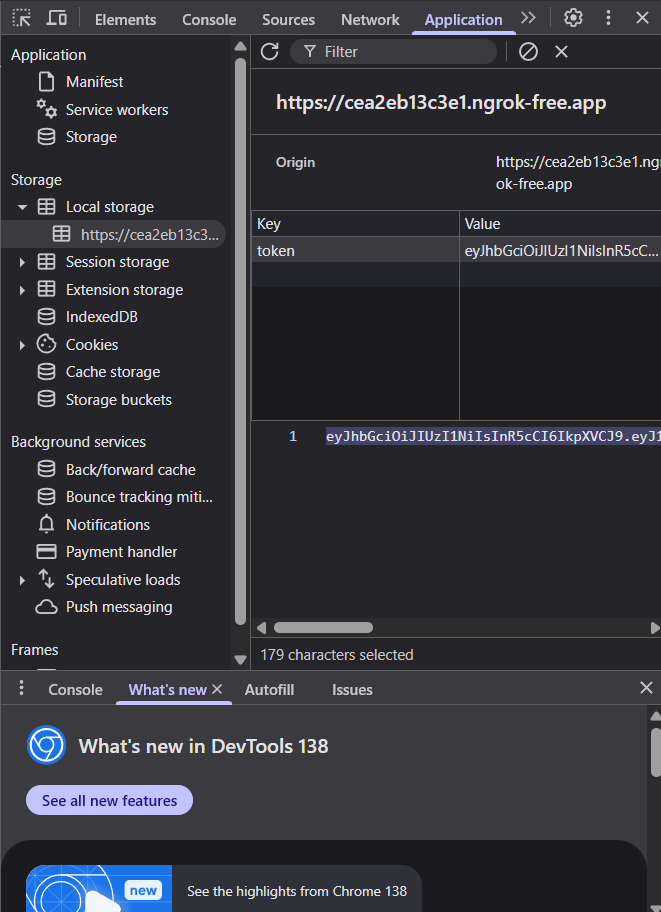
Accès au dashboard avec token modifié (version vulnérable)



Modification du rôle dans jwt.io



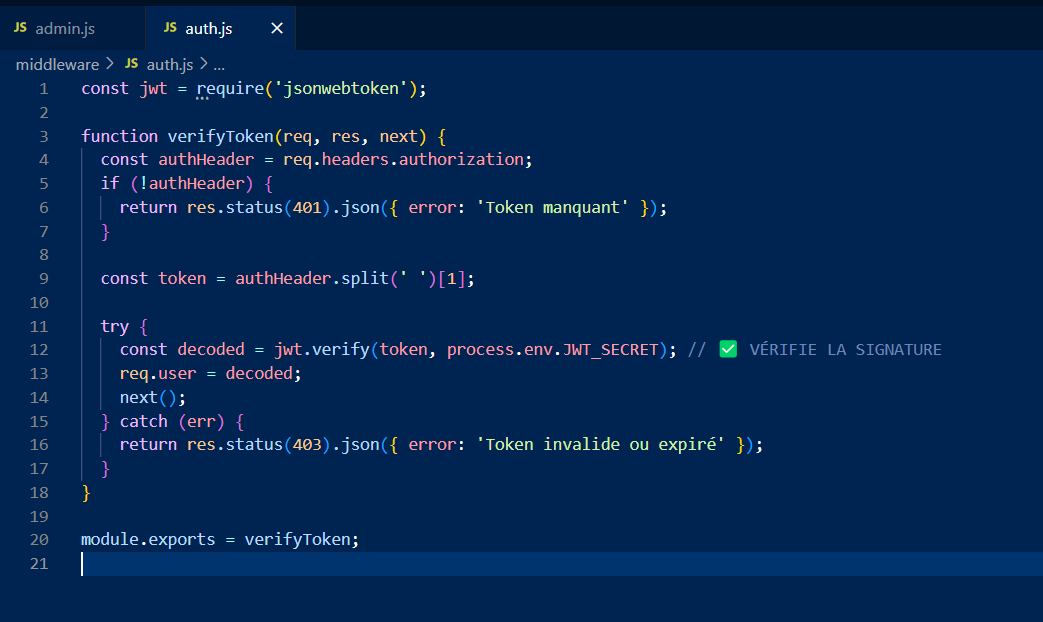
Récupération du token dans le localStorage



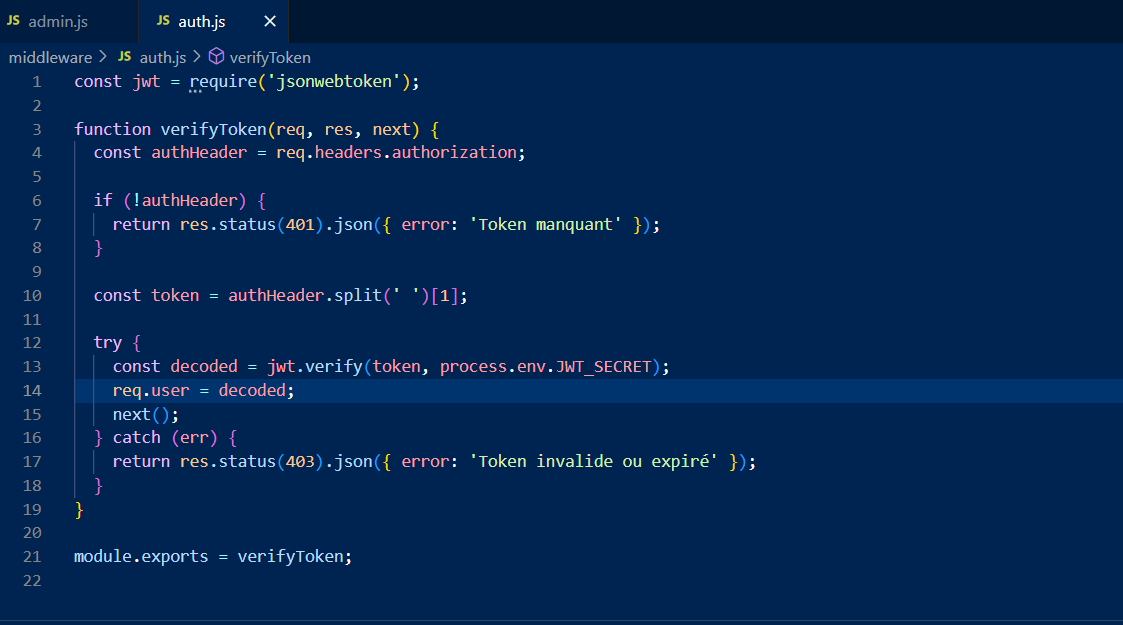
Accès refusé après correction (token modifié rejeté)



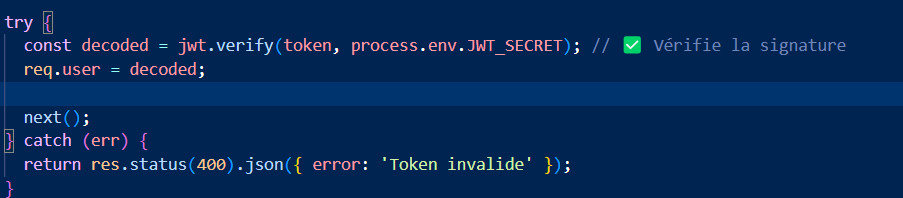
Code corrigé avec jwt.verify dans middleware



Code vulnérable avec jwt.decode



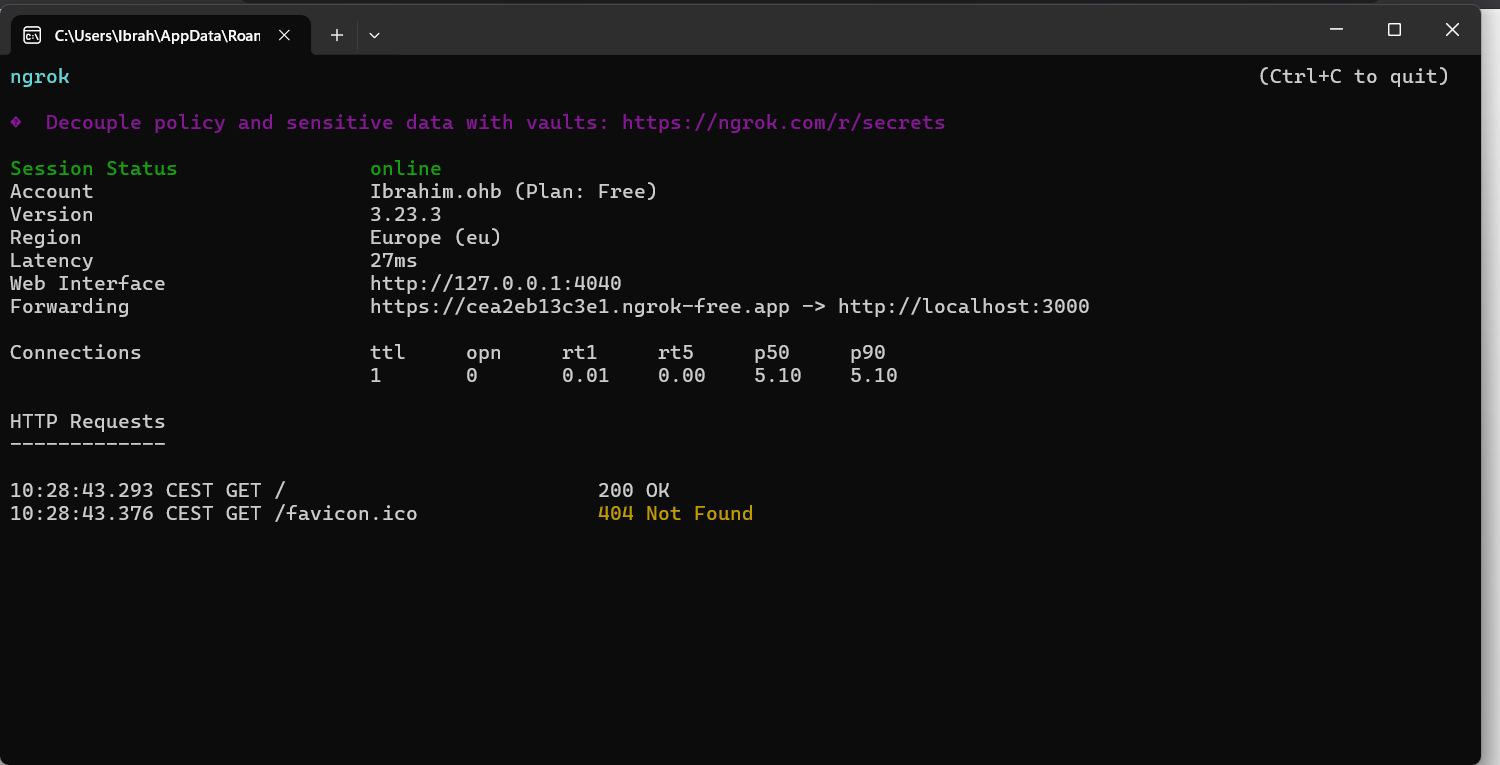
Utilisation du middleware sécurisé dans admin.js



Code admin.js après sécurisation



Serveur local exposé en HTTPS via Ngrok



TP Sécurité Web – JWT, HTTPS et Ngrok

# 📁 Structure du projet

Le projet est disponible sur GitHub à l'adresse suivante :  
🔗 https://github.com/Narutino10/TP-Secu  
  
Il contient deux dossiers :  
- `Projet-Secu` : version vulnérable (jwt.decode sans vérification)  
- `Projet-Secu-Fixe` : version corrigée (jwt.verify sécurisé)

# 🔧 Prérequis

- Node.js installé

- Ngrok installé (`npm install -g ngrok`)

- Un terminal ou éditeur (ex: Visual Studio Code)

# 🚀 Étapes pour lancer le projet

## 1. Cloner le dépôt

Dans un terminal :

git clone https://github.com/Narutino10/TP-Secu

## 2. Lancer la version vulnérable

1. Ouvrir un terminal dans le dossier `Projet-Secu` :

cd TP-Secu/Projet-Secu

2. Installer les dépendances :

npm install

3. Lancer le serveur :

npm run dev

4. Ouvrir un tunnel HTTPS avec ngrok :

ngrok http 3000

5. Ouvrir l’URL HTTPS dans un navigateur (ex: https://xxxxx.ngrok-free.app)

## 3. Lancer la version corrigée

Même procédure mais dans le dossier `Projet-Secu-Fixe` :

cd TP-Secu/Projet-Secu-Fixe

npm install

npm run dev

ngrok http 3000

# 🧪 Test de la faille (version vulnérable)

- Connecte-toi avec n’importe quel nom (aucune vérification)

- Récupère le token dans le localStorage (via DevTools)

- Modifie le payload sur jwt.io (par ex. role: 'admin')

- Colle le token falsifié dans le localStorage

- Accès autorisé même avec un faux token → FAILLE

# 🔒 Test de la correction (version sécurisée)

- Même manipulation

- Résultat : `Token invalide ou expiré` → faille corrigée avec jwt.verify