

# RAPPORT D'AUDIT DE SÉCURITÉ

Rapport d'audit pour notre Application Web  
Twitter Retro

Préparé par

BOUKEDJANI IMAD  
THEO FRATCZAK  
MERIBOUT AHMED  
ALLAMI BOUDKHIL

Encadré par

YANN ROTELLA



# Table des Matières

<b>1. Introduction</b>	<b>2</b>
<b>2. Contexte de l'Audit :</b>	<b>2</b>
<b>3. Méthodologie :</b>	<b>3</b>
<b>4. Démarche de l'audit:</b>	<b>4</b>
<b>5. Résultat D'audit :</b>	<b>10</b>

# 1. Introduction

Avant d'entrer dans les détails spécifiques de cet audit de sécurité de l'application web "Twitter Retro", il est essentiel de comprendre en quoi consiste un audit d'application web.

Un audit d'application web est un processus d'évaluation approfondie de la sécurité d'une application web, visant à identifier les vulnérabilités potentielles et à proposer des solutions pour les corriger. Il s'agit d'un élément crucial dans le développement et le maintien d'applications web sécurisées, surtout à une époque où les cybermenaces sont de plus en plus sophistiquées.

L'objectif principal d'un tel audit est de garantir que l'application web respecte les normes de sécurité les plus élevées et offre une protection adéquate des données et des transactions en ligne. Cela implique d'analyser chaque aspect de l'application, des fonctionnalités visibles par les utilisateurs aux composants backend, en passant par les protocoles de communication et les mécanismes d'authentification.

Dans le cadre de ce rapport d'audit, nous examinerons en détail les différentes facettes de l'application "Twitter Retro", en mettant l'accent sur les aspects liés à la sécurité. Nous identifierons les éventuelles vulnérabilités et proposerons des recommandations pour renforcer la sécurité de l'application, assurant ainsi une expérience utilisateur sécurisée et fiable.

## 2. Contexte de l'Audit :

Dans le cadre de cet audit de sécurité, nous nous penchons sur l'application web "Twitter Retro". Cette application a été développée avec le MERN Stack (MongoDB, Express.js, React.js, Node.js), offrant ainsi une expérience utilisateur moderne et réactive. Elle a été hébergée sur un serveur VPS, assurant ainsi un contrôle total sur son environnement de déploiement et de fonctionnement.

Twitter Retro vise à offrir une expérience utilisateur similaire à celle de son homologue actuel, tout en mettant un accent particulier sur la sécurité des données et des transactions en ligne. Cette application a été créée en Mai 2024 et utilise une combinaison de technologies frontend et backend pour fournir ses fonctionnalités.

L'environnement de développement et de déploiement de Twitter Retro repose sur des technologies modernes, garantissant ainsi une architecture robuste et évolutive. L'utilisation du MERN Stack permet une gestion efficace des composants frontend et backend, tandis que le déploiement sur un serveur VPS assure une disponibilité et des performances optimales.

Au cours de cet audit, nous examinerons en détail les différentes fonctionnalités de Twitter Retro, en mettant l'accent sur la sécurité des données et des transactions en ligne. Nous évaluerons les risques potentiels et proposerons des recommandations pour renforcer la sécurité de l'application, assurant ainsi une expérience utilisateur sécurisée et fiable.

### 3.Méthodologie :

Pour mener à bien notre audit de sécurité, nous avons suivi une approche méthodique combinant des tests manuels approfondis et l'utilisation d'outils d'analyse automatisée. Nos tests ont inclus une évaluation des dix principales vulnérabilités de l'OWASP, qui sont largement reconnues comme représentatives des risques les plus courants rencontrés dans les applications web. En nous concentrant sur ces vulnérabilités, nous avons pu identifier les points faibles de l'application et recommander des mesures correctives appropriées pour renforcer sa sécurité.

1. **Exploration des fonctionnalités** : Nous avons passé en revue chaque fonctionnalité de l'application Twitter Retro, en examinant attentivement les interactions utilisateur et les flux de données. Cela nous a permis de comprendre en profondeur le fonctionnement de l'application et d'identifier les points d'entrée potentiels pour les attaques.
2. **Analyse des données** : Nous avons analysé la manière dont les données sont traitées et stockées au sein de l'application, en mettant l'accent sur la sécurité de la couche de persistance des données. Nous avons examiné les pratiques de gestion des bases de données et évalué les mesures de protection mises en place pour garantir l'intégrité et la confidentialité des données.
3. **Test de sécurité** : Nous avons effectué des tests de sécurité approfondis pour identifier les vulnérabilités potentielles telles que les injections SQL, les attaques XSS (Cross-Site Scripting), et les failles d'authentification. Nous avons également examiné la robustesse des mécanismes de contrôle d'accès et de gestion des sessions pour détecter d'éventuelles faiblesses.
4. **Test de compatibilité** : Nous avons vérifié que l'application fonctionne correctement sur une gamme de navigateurs web et de dispositifs, en tenant compte des différences de rendu et de comportement.
5. **Analyse des flux réseau** : Nous avons analysé les flux réseau entre l'application et ses clients, en mettant l'accent sur la sécurisation des communications. Nous avons vérifié l'utilisation de protocoles sécurisés tels que HTTPS et évalué la résilience de l'application aux attaques de type MITM (Man-In-The-Middle).
6. **Évaluation des outils tiers** : Nous avons examiné les bibliothèques et les modules tiers utilisés dans l'application pour identifier les éventuelles vulnérabilités connues ou les

dépendances obsolètes. Nous avons également évalué la fréquence et la pertinence des mises à jour de sécurité appliquées à ces composants.

En combinant ces différentes approches, nous avons pu obtenir une vision complète de la posture de sécurité de l'application Twitter Retro et identifier les mesures nécessaires pour renforcer sa résilience face aux menaces potentielles.

## 4. Démarche de l'audit:

La première étape de notre audit de sécurité consiste à effectuer une Reconnaissance Passive en utilisant l'outil “**Whois**”. Cette démarche nous permet d'obtenir des informations précieuses telles que l'adresse IP de notre serveur hébergeant l'application et le fournisseur de services DNS associé. En comprenant l'infrastructure sous-jacente, nous sommes mieux préparés pour la phase suivante de notre évaluation de sécurité.

twitter-retro.fr				
DNS information				
<div>WhoisDNS RecordsDiagnostics</div>				
DNS Records for twitter-retro.fr				
Hostname	Type	TTL	Priority	Content
twitter-retro.fr	SOA	3600		twitter-retro.fr admin@pulseheberg.com 2024031760 604800 86400 2419200 604800
twitter-retro.fr	NS	3600		ns0.pulseheberg.net
twitter-retro.fr	NS	3600		ns1.pulseheberg.net
twitter-retro.fr	A	3600		185.216.26.172
www.twitter-retro.fr	A	3600		185.216.26.172
www.twitter-retro.fr	CNAME	3600		twitter-retro.fr

Nous passons ensuite à une Reconnaissance Active en utilisant l'outil “**Nmap**”. Cette étape implique l'analyse des ports ouverts et des services en cours d'exécution sur notre serveur. En utilisant les options -sV pour détecter les versions des services et -p- pour scanner tous les ports de 0 à 65536, nous obtenons un aperçu détaillé des vulnérabilités potentielles.

```
lmaaad@lmaaadSPC: ~  
SF:/1\.\1\x20400\x20Bad\x20Request\r\n\r\n");  
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel  
  
Service detection performed. Please report any incorrect results at https://nmap  
.org/submit/ .  
Nmap done: 1 IP address (1 host up) scanned in 119.19 seconds  
lmaaad@lmaaadSPC:~$ nmap -sV -p- 185.216.26.172  
Starting Nmap 7.80 ( https://nmap.org ) at 2024-05-05 15:28 CEST  
  
lmaaad@lmaaadSPC:~$  
lmaaad@lmaaadSPC:~$ nmap -sV -p- 185.216.26.172  
Starting Nmap 7.80 ( https://nmap.org ) at 2024-05-05 15:28 CEST  
Nmap scan report for 185.216.26.172  
Host is up (0.021s latency).  
Not shown: 65528 filtered ports  
PORT      STATE SERVICE      VERSION  
22/tcp    open  ssh          OpenSSH 8.2p1 Ubuntu 4ubuntu0.11 (Ubuntu Linux; prot  
ocol 2.0)  
25/tcp    closed smtp  
80/tcp    open  http         nginx 1.18.0 (Ubuntu)  
443/tcp   open  ssl/http     nginx 1.18.0 (Ubuntu)  
3001/tcp  open  nessus?  
3002/tcp  open  exlm-agent?  
8000/tcp  open  http         Node.js Express framework
```

Voici ce que chaque option signifie :

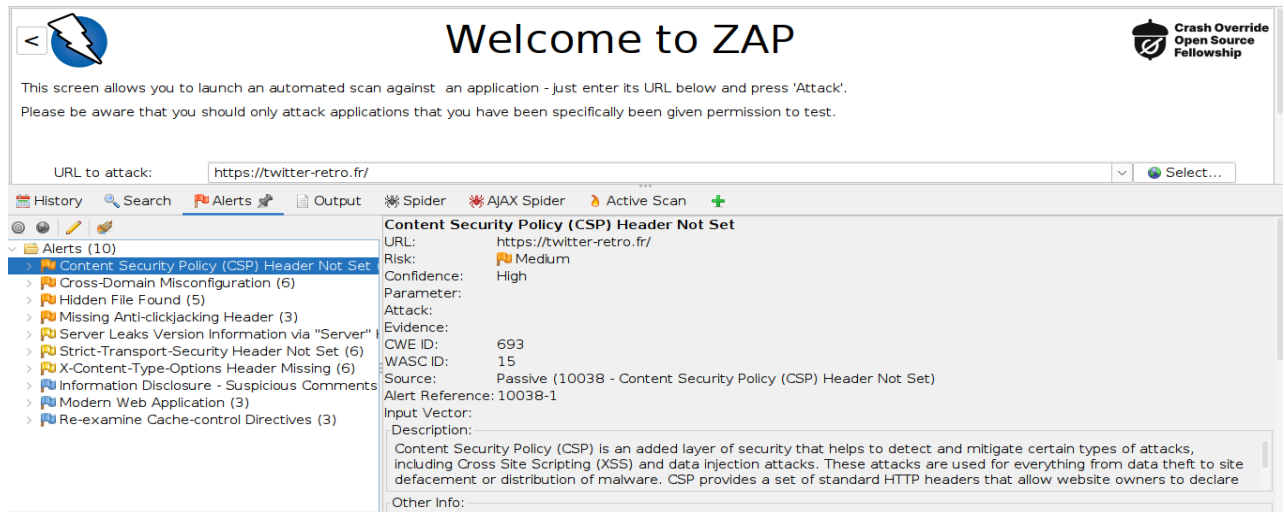
- `-sV` : Cette option permet de détecter la version des services en cours d'exécution sur les ports ouverts. Nmap envoie des requêtes pour identifier le service et la version qui y est associée en fonction de ses signatures de réponse.
- `-p-` : Cette option est pour scanner tous les ports de 0 jusqu' à 65536.

Lorsqu'on exécute cette commande avec une adresse IP spécifique, Nmap effectuera un scan de cette adresse IP en utilisant les options spécifiées.  
comme on peut voir que les ports 21 , 80 ,443,3001,3002,8000 sont ouverts .

L'énumération des vulnérabilités se poursuit avec l'utilisation d'outils automatisés tels que “**Nikto**”. Ce programme rapide et efficace permet de scanner les failles de sécurité du serveur web, fournissant ainsi une évaluation complète de la surface d'attaque. Les résultats obtenus nous guident dans la mise en œuvre de correctifs appropriés pour renforcer la sécurité de notre application.

```
lmaaad@lmaaadSPC:~$ nikto -host https://twitter-retro.fr/login  
- Nikto v2.1.5  
-----  
+ Target IP: 185.216.26.172  
+ Target Hostname: twitter-retro.fr  
+ Target Port: 443  
+ Start Time: 2024-05-02 15:51:48 (GMT2)  
-----  
+ Server: nginx/1.18.0 (Ubuntu)  
+ The anti-clickjacking X-Frame-Options header is not present.  
+ No CGI Directories found (use '-C all' to force check all possible dirs)  
+ 6544 items checked: 0 error(s) and 1 item(s) reported on remote host  
+ End Time: 2024-05-02 15:58:49 (GMT2) (421 seconds)  
-----  
+ 1 host(s) tested  
lmaaad@lmaaadSPC:~$
```


Pour une analyse plus approfondie, nous faisons appel à Owasp Zap, un scanner puissant qui identifie et remédier aux vulnérabilités potentielles dans les applications web. Les 10 alertes relevées après le test, réparties selon leur niveau de risque, orientent nos efforts de sécurisation vers les zones les plus critiques.



On peut voir qu'il y a 10 alerte après le test : 4 avec un risque moyen , 3 avec un risque bas , et 3 les derniers leurs risque c'est informative.

En poursuivant notre audit de **manière manuelle**, nous explorons les vulnérabilités courantes telles que les injections SQL. Étant donné que notre application repose sur MongoDB, une base de données NoSQL, nous menons des tests exhaustifs pour identifier et corriger les failles potentielles.

# Happening now.

 Unknow User, please retry

Email

Password

☐ Remember me [Forgot Password?](#)

Login

# Happening now.



Unknow User, please retry



Email

Bynawers.fratczak@gmail.com

Password

' or '1'='1

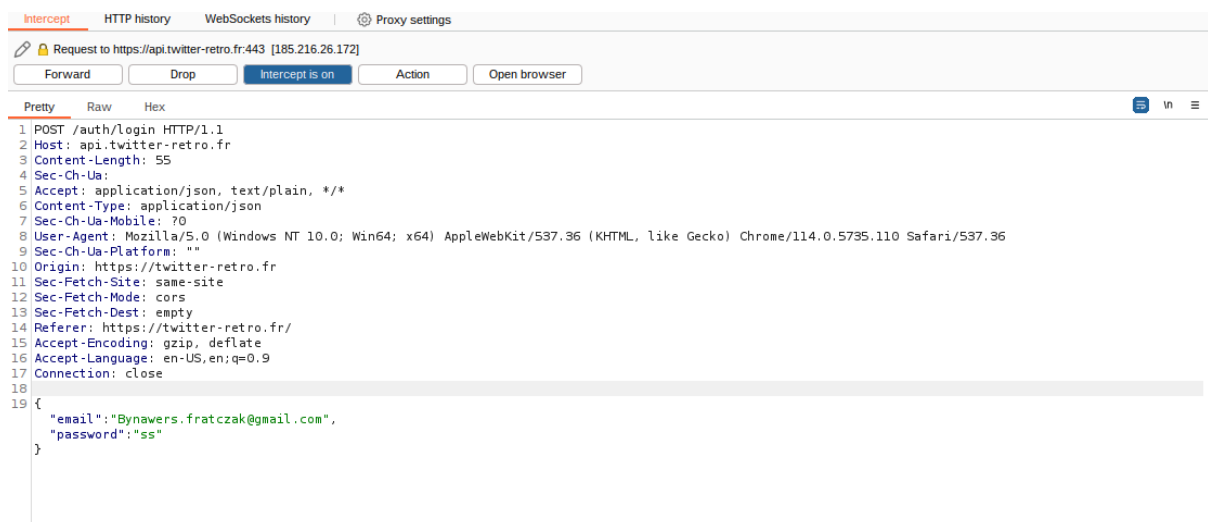
☐ Remember me

[Forgot Password?](#)

Login

Nous utilisons également Burp Suite, un outil de pentest, pour effectuer des injections directes dans les requêtes. Ce processus nous permet de découvrir des vulnérabilités qui pourraient être exploitées par des attaquants malveillants pour compromettre la sécurité de notre application.

Après ça on a essayé de faire l'injection directement dans la requêtes utilisant Burp suit qui est un outil utiliser pour le pentest :





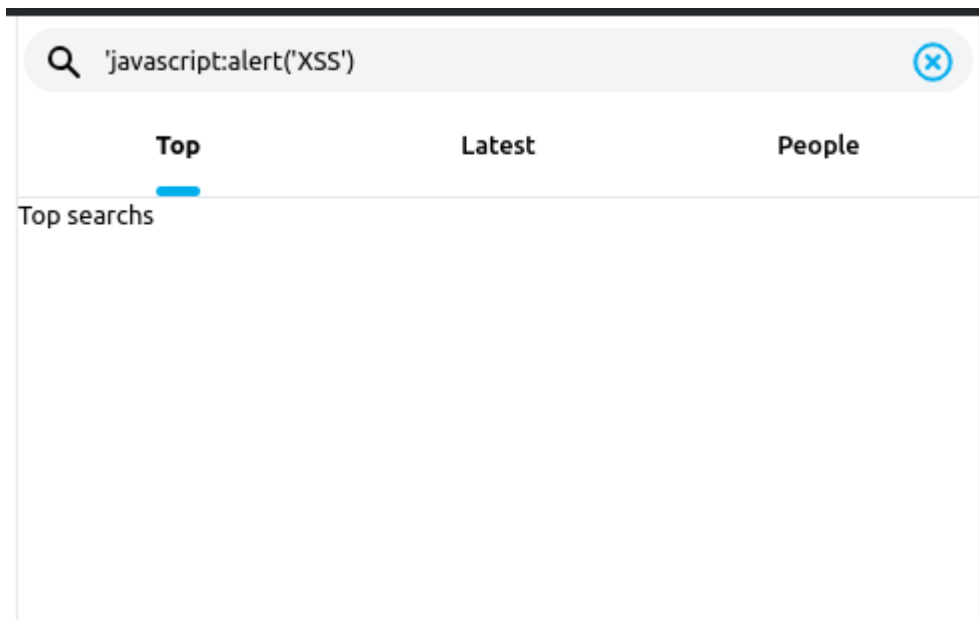
```
"email": "Bynawers.fratczak@gmail.com",
"password": {
  "$regex": ".*"
}
}
```

Response

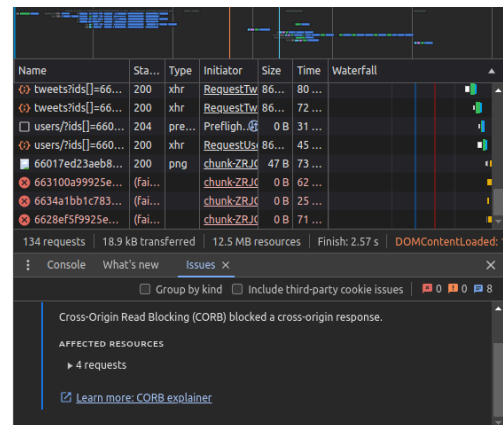
Pretty Raw Hex Render

```
9 cross-origin-resource-policy: cross-origin
10 Origin-Agent-Cluster: ?1
11 Referrer-Policy: no-referrer
12 Strict-Transport-Security: max-age=15552000; includeSubDomains
13 X-Content-Type-Options: nosniff
14 X-DNS-Prefetch-Control: off
15 X-Download-Options: noopen
16 X-Frame-Options: SAMEORIGIN
17 X-Permitted-Cross-Domain-Policies: none
18 X-XSS-Protection: 0
19 Access-Control-Allow-Origin: *
20 ETag: W/"29-b8aDslp/ZKhX9Ec/09IcWxdfJQo"
21
22 {"error": "data and hash must be strings"}
```

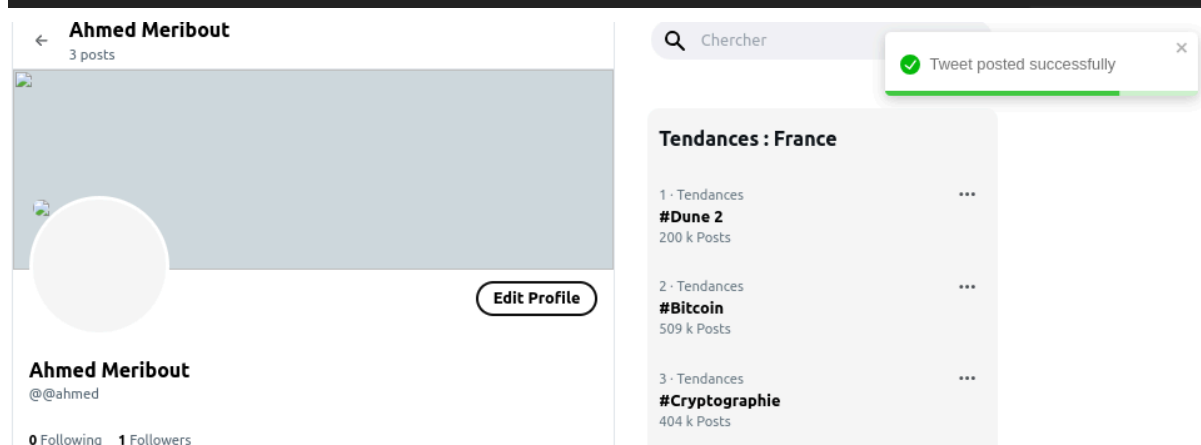
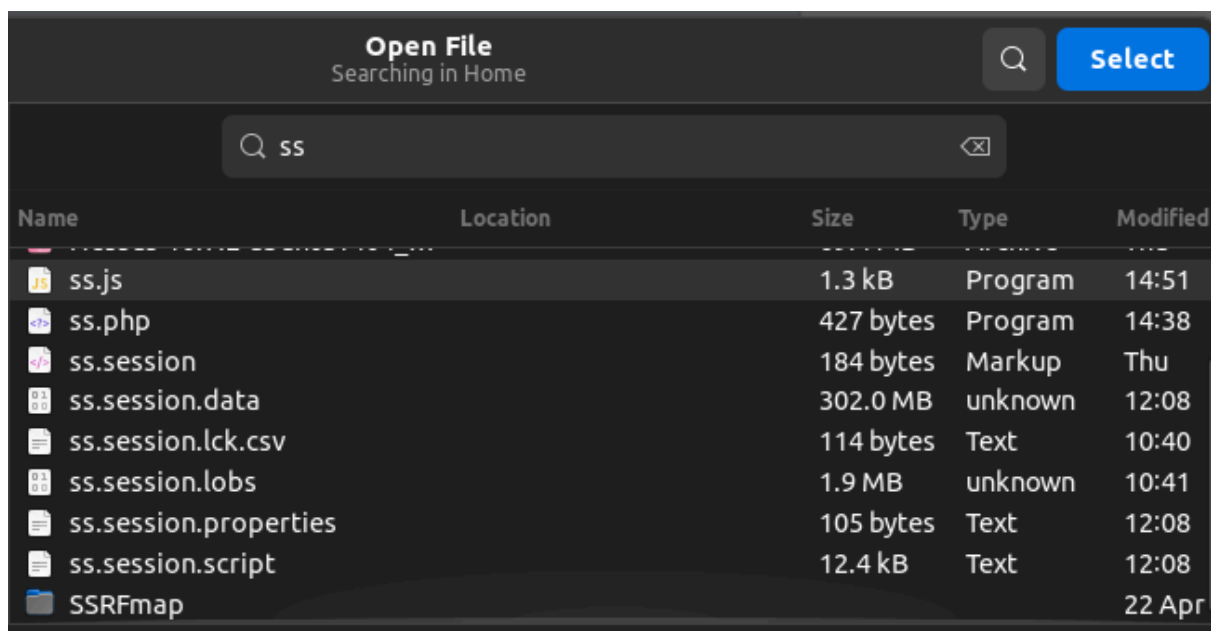
Dans le cadre de nos tests, nous explorons également les attaques XSS (Cross-Site Scripting), en recherchant des vulnérabilités qui pourraient permettre à des attaquants d'injecter du code malveillant dans les pages web. Malgré les contrôles de sécurité du navigateur, nous identifions des vulnérabilités qui nécessitent une attention immédiate.



On a essayé de faire d'autres tests XSS mais c'était bloquer par le navigateur a chaque fois avec le CORB (Cross-Origin Read Blocking ) :



Après qu'on a noté tous ça on a décidé de passer vers le code review et input validation and upload filtering , et on a trouvé pas male de chose , comme dans le upload profile , on peut uploader dans la place d'une photo de profile quelque soit le format qu'on veut qui peut servir a une Remote Code Execution (RCE) .



Chaque étape de notre audit de sécurité contribue à renforcer la robustesse de notre application et à protéger les données de nos utilisateurs contre les menaces potentielles. En prenant des mesures proactives pour identifier et corriger les vulnérabilités, nous garantissons une expérience en ligne sécurisée et fiable pour nos utilisateurs.

## 5. Résultat D'audit :

Dans le cadre de l'audit de sécurité de notre application, nous avons identifié plusieurs mesures critiques visant à renforcer la protection de nos données et à prévenir les attaques potentielles. Voici une explication détaillée de chaque recommandation :

### 1. Ajout du CSP Header :

Le Content Security Policy (CSP) est un mécanisme de sécurité qui permet de limiter les sources de contenu autorisées sur une page web. En ajoutant un CSP Header à notre application, nous spécifions les origines légitimes pour les ressources telles que les scripts, les styles et les images. Cela réduit considérablement le risque d'injections de scripts malveillants, comme les attaques XSS, en limitant les points d'entrée potentiels pour les attaquants.

### 2. Ajustement de l'option d'origine dans le middleware CORS :

Le Cross-Origin Resource Sharing (CORS) est un mécanisme qui contrôle les requêtes HTTP entre différents domaines. En ajustant l'option d'origine dans le middleware CORS pour inclure notre propre domaine, nous restreignons l'accès aux ressources de notre application uniquement aux domaines autorisés. Cela protège notre application contre les attaques malveillantes telles que les demandes CSRF (Cross-Site Request Forgery) en limitant les interactions avec des origines non fiables.

### 3. Suppression du répertoire .git du répertoire racine du site web :

Le répertoire .git peut contenir des informations sensibles telles que les journaux de validation, les identifiants de compte et d'autres données confidentielles. En supprimant ce répertoire du répertoire racine de notre site web, nous réduisons considérablement le risque de fuites d'informations et de compromission de la sécurité.

### 4. Configuration de l'en-tête X-Frame-Options :

L'en-tête X-Frame-Options permet de contrôler si une page web peut être chargée dans un cadre (frame) par un autre site. En définissant cette option sur 'SAMEORIGIN', nous limitons l'affichage de notre site web dans des cadres provenant uniquement du même domaine. Cela protège notre site contre les attaques de type Clickjacking, où un attaquant tente de tromper les utilisateurs en affichant notre site dans un cadre malveillant.

### 5. Configuration de l'en-tête X-Content-Type-Options :

L'en-tête X-Content-Type-Options avec la valeur 'nosniff' indique aux navigateurs de ne pas effectuer de "MIME-sniffing". Cela signifie qu'ils doivent respecter strictement le type de contenu déclaré dans les réponses HTTP, réduisant ainsi le risque d'exécution de contenu malveillant masqué sous une fausse extension de fichier.

### 6. Configuration de l'en-tête Strict-Transport-Security (HSTS) :

L'en-tête HSTS oblige les navigateurs à utiliser HTTPS pour toutes les communications avec notre site web. Cela renforce la sécurité des échanges de données en chiffrant le trafic et en protégeant contre les attaques de type Man-in-the-Middle. En imposant l'utilisation de HTTPS, nous réduisons le risque d'interception ou de modification malveillante des données sensibles.

#### **7. Filtrage des fichiers téléchargés :**

Il est crucial de mettre en place un filtrage strict pour les fichiers téléchargés sur notre application. En n'acceptant que les types de fichiers autorisés tels que les images PNG, JPG et JPEG, nous réduisons considérablement le risque d'exécution de code malveillant via des fichiers téléchargés. De plus, en vérifiant les en-têtes des fichiers pour confirmer leur format, nous renforçons encore la sécurité de notre application contre les attaques basées sur des fichiers téléchargés.

#### **8. Vérification de la complexité des mots de passe :**

En définissant des critères de complexité pour les mots de passe des utilisateurs, tels que l'inclusion d'au moins une minuscule, une majuscule, un chiffre et un symbole, nous renforçons la sécurité des comptes utilisateur. Cela rend les mots de passe plus difficiles à deviner pour les attaquants, réduisant ainsi le risque de compromission des comptes utilisateur et des données sensibles associées.

En mettant en œuvre ces recommandations de sécurité, nous améliorons significativement la robustesse de notre application et réduisons les risques d'exploitation par des attaquants malveillants. Ces mesures renforcent la protection de nos données et garantissent une expérience utilisateur sécurisée et fiable.

## **6 .Conclusion:**

Dans notre conclusion, nous soulignons l'importance pour notre groupe d'étudiants de maintenir la sécurité de notre application en intégrant des pratiques d'audit et de test régulières. Voici comment nous pouvons formuler cela :

En tant qu'équipe responsable de la sécurité de notre application, nous comprenons l'importance de rester vigilants face aux menaces potentielles. Notre audit de sécurité a mis en lumière plusieurs vulnérabilités et lacunes de sécurité qui pourraient compromettre la confidentialité et l'intégrité de nos données.

Pour assurer la protection continue de notre application, il est essentiel que nous adoptions une approche proactive en matière de sécurité. Cela inclut la réalisation régulière de scans d'audit et de tests de sécurité pour identifier et corriger les vulnérabilités avant qu'elles ne soient exploitées par des individus malveillants.

En intégrant des pratiques d'audit et de test régulières dans nos processus de développement et de maintenance, nous pouvons renforcer la sécurité de notre application et protéger efficacement nos données et nos utilisateurs contre les menaces potentielles.

En conclusion, il est de notre responsabilité de maintenir la sécurité de notre application en adoptant une approche proactive en matière de sécurité. En restant vigilants et en menant des audits de sécurité réguliers, nous pouvons assurer la fiabilité et la sécurité de notre application pour l'ensemble de nos utilisateurs.