Rapport d'analyse de sécurité

Date: 20/03/2025

Résumé exécutif

Cette analyse de sécurité a identifié plusieurs vulnérabilités critiques sur l'application web testée. L'analyse a révélé 8 vulnérabilités d'injection SQL, 2 vulnérabilités XSS, 4 vulnérabilités CSRF et 42 problèmes liés aux en-têtes de sécurité. Ces vulnérabilités représentent des risques importants pour la confidentialité, l'intégrité et la disponibilité des données. Des mesures correctives doivent être prises immédiatement pour remédier à ces problèmes.

1. Injections SQL (SQLi)

Page: http://127.0.0.1:5000/recherche

- Champ vulnérable: query
  \- Type: Error-based SQLi
  \- Payload testé: admin' #
  \- Temps de réponse: 0.066 secondes

Page: http://127.0.0.1:5000/login

- Champ vulnérable: username
  \- Type: SQLi bypass authentication (content change)
  \- Payload testé: admin'/*
  \- Temps de réponse: 0.07 secondes
  \- Formulaire d'authentification: Oui
- Champ vulnérable: password
  \- Type: SQLi bypass authentication (content change)
  \- Payload testé: ' OR '1'='1
  \- Temps de réponse: 0.052 secondes
  \- Formulaire d'authentification: Oui

Page: http://127.0.0.1:5000/inscription

- Champ vulnérable: username
  \- Type: Error-based SQLi
  \- Payload testé: ' UNION SELECT null, version()--
  \- Temps de réponse: 0.116 secondes

\- Formulaire d'authentification: Oui

- Champ vulnérable: password
  \- Type: SQLi bypass authentication (redirection)
  \- Payload testé: 1' OR '1' = '1
  \- Temps de réponse: 0.168 secondes
  \- Formulaire d'authentification: Oui

Page: http://127.0.0.1:5000/comments

- Champ vulnérable: comment
  \- Type: Error-based SQLi
  \- Payload testé: ' OR 1=1 --
  \- Temps de réponse: 0.141 secondes

Page: http://127.0.0.1:5000/profile

- Champ vulnérable: username
  \- Type: SQLi bypass authentication (redirection)
  \- Payload testé: ' AND (SELECT * FROM (SELECT(SLEEP(2)))a)--
  \- Temps de réponse: 0.049 secondes
  \- Formulaire d'authentification: Oui
- Champ vulnérable: password
  \- Type: SQLi bypass authentication (redirection)
  \- Payload testé: 'UNION user = 'admin' --
  \- Temps de réponse: 0.035 secondes
  \- Formulaire d'authentification: Oui

2. Cross-Site Scripting (XSS)

Page: http://127.0.0.1:5000/echo

- Type: reflected XSS
  \- Paramètre vulnérable: message
  \- Payload testé: <script>alert(37c99f)</script>

Page: http://127.0.0.1:5000/comments

- Type: stored XSS
  \- Paramètre vulnérable: comment
  \- Payload testé: <script>alert(e9f6e90)</script>
  \- URL d'affichage: http://127.0.0.1:5000/comments

3. Cross-Site Request Forgery (CSRF)

- URL vulnérable: http://127.0.0.1:5000/login
  \- Élément: Form targeting http://127.0.0.1:5000/login
  \- Méthode: POST
  \- Preuve: POST method without detectable CSRF protection
- URL vulnérable: http://127.0.0.1:5000/inscription
  \- Élément: Form targeting http://127.0.0.1:5000/inscription
  \- Méthode: POST
  \- Preuve: POST method without detectable CSRF protection
- URL vulnérable: http://127.0.0.1:5000/comments
  \- Élément: Form targeting http://127.0.0.1:5000/comments
  \- Méthode: POST
  \- Preuve: POST method without detectable CSRF protection
- URL vulnérable: http://127.0.0.1:5000/profile
  \- Élément: Form targeting http://127.0.0.1:5000/profile
  \- Méthode: POST
  \- Preuve: POST method without detectable CSRF protection

4. Problèmes d'en-têtes de sécurité

URL: http://127.0.0.1:5000

- En-tête manquant: Content-Security-Policy
- En-tête manquant: Referrer-Policy
- En-tête manquant: Strict-Transport-Security
- En-tête manquant: X-Content-Type-Options
- En-tête manquant: X-Frame-Options
- En-tête manquant: X-XSS-Protection

URL: http://127.0.0.1:5000/echo

- En-tête manquant: Content-Security-Policy
- En-tête manquant: Referrer-Policy
- En-tête manquant: Strict-Transport-Security
- En-tête manquant: X-Content-Type-Options
- En-tête manquant: X-Frame-Options
- En-tête manquant: X-XSS-Protection

URL: http://127.0.0.1:5000/recherche

- En-tête manquant: Content-Security-Policy
- En-tête manquant: Referrer-Policy
- En-tête manquant: Strict-Transport-Security
- En-tête manquant: X-Content-Type-Options
- En-tête manquant: X-Frame-Options
- En-tête manquant: X-XSS-Protection

URL: http://127.0.0.1:5000/login

- En-tête manquant: Content-Security-Policy
- En-tête manquant: Referrer-Policy
- En-tête manquant: Strict-Transport-Security
- En-tête manquant: X-Content-Type-Options
- En-tête manquant: X-Frame-Options
- En-tête manquant: X-XSS-Protection

URL: http://127.0.0.1:5000/inscription

- En-tête manquant: Content-Security-Policy
- En-tête manquant: Referrer-Policy
- En-tête manquant: Strict-Transport-Security
- En-tête manquant: X-Content-Type-Options
- En-tête manquant: X-Frame-Options
- En-tête manquant: X-XSS-Protection

URL: http://127.0.0.1:5000/comments

- En-tête manquant: Content-Security-Policy
- En-tête manquant: Referrer-Policy
- En-tête manquant: Strict-Transport-Security
- En-tête manquant: X-Content-Type-Options
- En-tête manquant: X-Frame-Options
- En-tête manquant: X-XSS-Protection

URL: http://127.0.0.1:5000/profile

- En-tête manquant: Content-Security-Policy
- En-tête manquant: Referrer-Policy
- En-tête manquant: Strict-Transport-Security
- En-tête manquant: X-Content-Type-Options
- En-tête manquant: X-Frame-Options
- En-tête manquant: X-XSS-Protection

Recommandations de sécurité

Pour les injections SQL:

- Utiliser des requêtes paramétrées (prepared statements) avec des paramètres liés
- Mettre en place un ORM (Object-Relational Mapping)
- Valider les entrées utilisateur en utilisant une liste blanche
- Limiter les privilèges de la base de données
- Utiliser des procédures stockées

Exemple de correction en Python (Flask/SQLAlchemy):

```
# Mauvaise pratique:
query = "SELECT * FROM users WHERE username = '" + username + "'"

# Bonne pratique avec SQLAlchemy:
user = db.session.query(User).filter(User.username == username).first()

# Bonne pratique avec requêtes paramétrées:
cursor.execute("SELECT * FROM users WHERE username = %s", (username,))
```

Pour les vulnérabilités XSS:

- Échapper les données de sortie en utilisant des bibliothèques d'échappement spécifiques au contexte
- Mettre en place une politique de sécurité de contenu (CSP)
- Utiliser des attributs HttpOnly et Secure pour les cookies
- Valider les entrées utilisateur côté serveur
- Utiliser des bibliothèques sécurisées pour le rendu HTML

Exemple de correction en Python (Flask):

```
# Mauvaise pratique:
@app.route('/echo')
def echo():
    message = request.args.get('message', '')
    return f"<p>{message}</p>"  # Vulnérable au XSS

# Bonne pratique:
from markupsafe import escape
@app.route('/echo')
def echo_secure():
    message = request.args.get('message', '')
    return f"<p>{escape(message)}</p>"  # Sécurisé
```

Pour les vulnérabilités CSRF:

- Utiliser des jetons anti-CSRF dans les formulaires
- Vérifier l'en-tête Referer pour les requêtes sensibles
- Implémenter l'en-tête SameSite=Strict pour les cookies
- Utiliser des mécanismes de double soumission de cookies

Exemple de correction en Python (Flask):

```
# Avec Flask-WTF:
app = Flask(__name__)
app.config['SECRET_KEY'] = 'clé-secrète-difficile-à-deviner'
csrf = CSRFProtect(app)

# Dans le formulaire:
@app.route('/profile', methods=['GET', 'POST'])
def profile():
    form = ProfileForm()
    if form.validate_on_submit():
        # Le jeton CSRF est automatiquement vérifié
        # Traitement du formulaire
        pass
    return render_template('profile.html', form=form)

# Dans le template HTML:
<form method="post">
    {{ form.csrf_token }}
    <!-- Autres champs du formulaire -->
    <button type="submit">Envoyer</button>
</form>
```

Pour les problèmes d'en-têtes de sécurité:

Exemple d'ajout d'en-têtes de sécurité en Python (Flask):

```
@app.after_request
def set_security_headers(response):
    response.headers['Content-Security-Policy'] = "default-src 'self'"
    response.headers['X-Content-Type-Options'] = 'nosniff'
    response.headers['X-Frame-Options'] = 'DENY'
    response.headers['X-XSS-Protection'] = '1; mode=block'
    response.headers['Strict-Transport-Security'] = 'max-age=31536000; includeSubDomains'
    response.headers['Referrer-Policy'] = 'no-referrer-when-downgrade'
    return response
```

- Content-Security-Policy: Définit les sources de contenu approuvées. Exemple: Content-Security-Policy: default-src 'self'
- X-Frame-Options: Prévient le clickjacking. Exemple: X-Frame-Options: DENY
- X-Content-Type-Options: Prévient le MIME-sniffing. Exemple: X-Content-Type-Options: nosniff
- X-XSS-Protection: Active la protection XSS du navigateur. Exemple: X-XSS-Protection: 1; mode=block

- Strict-Transport-Security: Force les connexions HTTPS. Exemple: Strict-Transport-Security: max-age=31536000; includeSubDomains
- Referrer-Policy: Contrôle les informations du référent. Exemple: Referrer-Policy: no-referrer-when-downgrade

Conclusion

L'analyse de sécurité a révélé plusieurs vulnérabilités critiques qui nécessitent une attention immédiate. Ces problèmes exposent l'application web à des risques significatifs, notamment: - Divulgation ou corruption de données sensibles via des injections SQL - Vol de session et usurpation d'identité via des attaques XSS - Exécution d'actions non autorisées via des attaques CSRF - Exposition à diverses attaques dues à l'absence d'en-têtes de sécurité Il est fortement recommandé de suivre les recommandations détaillées dans ce rapport et de mettre en place un processus de développement sécurisé comprenant: 1. Formation de l'équipe de développement aux bonnes pratiques de sécurité 2. Tests de sécurité réguliers (SAST, DAST, tests de pénétration) 3. Revues de code axées sur la sécurité 4. Utilisation de bibliothèques et frameworks sécurisés et maintenus 5. Mise en place d'un programme de gestion des vulnérabilités La sécurité est un processus continu qui nécessite une attention constante. Il est recommandé de procéder à une nouvelle analyse après l'application des correctifs pour vérifier leur efficacité.