

Laboratoire 2:



LOG8100 - DevSecOps Opérations et dév. logiciel sécur

Yousra El Ghomari - 2235756

Ayah Farhat - 2153946

Youssef Ouarad - 2117136

Equipe 13

Soumis le 24 Octobre 2024

Soumis à Antoine Boucher

Introduction

Dans ce rapport, nous aborderons les tests de pénétration ainsi que les outils utilisés pour l'intégration et le déploiement continu. Nous commencerons par présenter une analyse approfondie de la sécurité et une évaluation des vulnérabilités d'une application sur laquelle nous avons réalisé des tests de pénétration. Ces tests ont permis d'identifier des failles potentielles et des axes d'amélioration en matière de sécurité. Ensuite, nous détaillerons la mise en place d'un pipeline d'intégration continue (CI) et d'un pipeline de déploiement continu (CD) sur GitHub. Ce pipeline a été conçu dans le but d'automatiser les processus de compilation, de test, et de déploiement de l'application, garantissant ainsi un flux de travail plus sécurisé, fiable et rapide.

Voici le lien vers notre dépôt [Github](#).

Analyse de sécurité

1. Résultats des Tests de Pénétration

Après avoir effectué une série de tests de pénétration sur l'application Web vulnérable hébergée à <http://localhost:3000>, plusieurs vulnérabilités ont été identifiées à l'aide d'OWASP ZAP. Ces tests ont impliqué l'exploration de l'application, la soumission de formulaires et des analyses de sécurité automatiques. Nous avons analysé 289 requêtes et identifié 12 [vulnérabilités](#). En voici 4 des plus élevées. La figure 1 présente les résultats du scan sur l'application ZAP.

2. Description des Vulnérabilités Identifiées

2.1. Bibliothèque JavaScript vulnérable (jQuery)

- **Description** : La bibliothèque JavaScript identifiée, jQuery version 3.2.1, présente plusieurs vulnérabilités connues comme *CVE-2020-11023*, *CVE-2020-11022* et *CVE-2019-11358*. Ces vulnérabilités peuvent être exploitées pour compromettre la sécurité de l'application.
- **Gravité** : Moyenne
- **Stratégies de mitigation** :
 - Mettre à jour la bibliothèque jQuery vers la version la plus récente afin de corriger ces vulnérabilités et renforcer la sécurité de l'application.

2.2. En-tête anti-clickjacking manquant

- **Description** : La réponse du serveur ne protège pas contre les attaques de type "ClickJacking". Cette attaque permet à un attaquant d'incorporer la page dans un cadre (iframe) sur un autre site et de tromper les utilisateurs pour qu'ils interagissent involontairement avec l'application. L'absence d'en-têtes comme *Content-Security-Policy* avec la directive *frame-ancestors* ou *X-Frame-Options* expose l'application à ce type d'attaque.
- **Gravité : Moyenne**
- **Stratégies de mitigation** :
 - Définir l'en-tête pour toutes les pages de notre site web.

2.3. En-tête Content Security Policy (CSP) non défini

- **Description** : La politique de sécurité de contenu (CSP) est une couche de sécurité supplémentaire qui aide à détecter et à atténuer certains types d'attaques, y compris les attaques de type Cross-Site Scripting (XSS) et les attaques par injection de données. Ces attaques peuvent entraîner des vols de données, des défigurations de sites, ou encore la distribution de malwares. CSP fournit un ensemble d'en-têtes HTTP standard permettant aux propriétaires de sites Web de déclarer les sources approuvées de contenu que les navigateurs doivent charger, y compris JavaScript, CSS, cadres HTML, polices, images et objets intégrés tels que les applets Java, ActiveX, fichiers audio et vidéo.
- **Gravité : Moyenne**
- **Stratégies de mitigation** :
 - S'assurer que notre serveur Web, serveur applicatif, ou équilibrage de charge est configuré pour définir l'en-tête **Content-Security-Policy**.

2.4. Utilisation de la directive wildcard (*) dans l'en-tête Content Security Policy (CSP)

- **Description** : La politique de sécurité de contenu (CSP) est une couche de sécurité supplémentaire qui aide à détecter et à atténuer certains types d'attaques, notamment les attaques de type Cross-Site Scripting (XSS) et les attaques par injection de données. Ces attaques peuvent être utilisées pour voler des données, défigurer un site ou distribuer des malwares. CSP offre un ensemble d'en-têtes HTTP standard qui permettent aux propriétaires de sites Web de déclarer les sources de contenu approuvées que les navigateurs doivent charger, couvrant des types de contenu tels que JavaScript, CSS, cadres HTML, polices, images et objets embarqués (comme des applets Java, ActiveX, fichiers audio et vidéo).

Cependant, l'utilisation de la directive wildcard (*) dans la CSP permet le chargement de contenu depuis n'importe quelle source, affaiblissant ainsi considérablement la sécurité et exposant l'application à des attaques externes.

- **Gravité : Moyenne**
- **Stratégies de mitigation :**
 - S'assurer que notre serveur web, serveur applicatif, ou équilibrage de charge est correctement configuré pour définir une **Content-Security-Policy** sans utiliser la directive *wildcard* (*).

The screenshot shows the ZAP interface with the following details:

- Request/Response Panel:**
 - Method: GET
 - URL: /sitemap.xml
 - Status: 404 Not Found
 - Headers:


```

X-Powered-By: Express
Content-Security-Policy: default-src 'none'
X-Content-Type-Options: nosniff
Content-Type: text/html; charset=utf-8
Content-Length: 150
Set-Cookie: connect.sid=s%3AvzKLZ9SPG61QWgegRH-PrY15UFxmj_Y.Gd%285mvP1%2BWhz60zXOUb2wsUDazJmimkK1PnhVutZdSI; Path=/; HttpOnly
Date: Thu, 17 Oct 2024 20:31:36 GMT
Connection: keep-alive
Keep-Alive: timeout=5
          
```
 - Body:


```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Cannot GET /sitemap.xml</pre>
</body>
</html>
          
```
- Alerts Panel:**
 - Alerts (13):
 - CSP: Wildcard Directive (8)
 - Content Security Policy (CSP) Header Not Set (2)
 - Missing Anti-clickjacking Header (2)
 - Vulnerable JS Library
 - Cookie without SameSite Attribute (5)
 - Cross-Domain JavaScript Source File Inclusion (6)
 - Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) (15)
 - X-Content-Type-Options Header Missing (5)
 - Authentication Request Identified
 - Information Disclosure - Suspicious Comments (3)
 - Loosely Scoped Cookie (19)
 - Session Management Response Identified (28)
 - User Agent Fuzzer (64)
- Alert Details Panel (CSP: Wildcard Directive):**
 - Attack: Evidence: default-src 'none'
 - CWE ID: 693
 - WASC ID: 15
 - Source: Passive (10055 - CSP)
 - Alert Reference: 10055-4
 - Input Vector:
 - Description: Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks. Including (but not limited to) Cross Site Scripting (XSS), and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on.
 - Other Info: The following directives either allow wildcard sources (or ancestors), are not defined, or are overly broadly defined: frame-ancestors, form-action
 - Solution:

Figure 1 : Résultats de la liste des alertes suite à notre scan sur ZAP

2. Utilisation de Orchestron

Nous avons aussi utilisé l'outil *Orchestron* pour visualiser de manière efficace [les vulnérabilités](#) de notre application. Nous avons passé le rapport du Scan automatique fait par ZAP en format XML via un webhook vers notre application *Orchestron* pour y parvenir.



Figure 2 : Rendu de notre scan ZAP sur Orchestron (1)

Open Vulnerabilities

8

Default View

Type to Search

Vulnerability Name	Tools
Content Security Policy (CSP) Header Not Set; CSP: Wildcard Directive - Multiple CWE : 693 Open For : 0 days ago	ZAP Show-Details
Vulnerable JS Library CWE : 829 Open For : 0 days ago	ZAP DVA
Missing Anti-clickjacking Header CWE : 1021 Open For : 0 days ago	ZAP DVA
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) CWE : 200 Open For : 0 days ago	ZAP DVA
X-Content-Type-Options Header Missing	ZAP

Figure 3 : Rendu de notre scan ZAP sur Orchestron (2)

Description du pipeline d'intégration continue (CI)

Étapes du Pipeline CI

1. Déclenchement du Pipeline:

- Le pipeline est déclenché automatiquement lorsqu'il y a un **push** ou une **pull request** sur la branche principale (main).

2. Vérification du Code et analyse de sécurité:

- Utilisation de **GitHub Actions** pour exécuter les étapes du pipeline. Cela comprend la vérification du code via les outils **SonarQube** et **CodeQL**.
- **SonarQube** effectue une analyse statique du code pour détecter les vulnérabilités et les problèmes de qualité.
- **CodeQL** analyse le code pour identifier les failles de sécurité et les erreurs potentielles.
- **Snyk** identifie les vulnérabilités dans les dépendances, effectue des tests et génère un rapport sur les failles détectées.

3. Gestion des Dépendances:

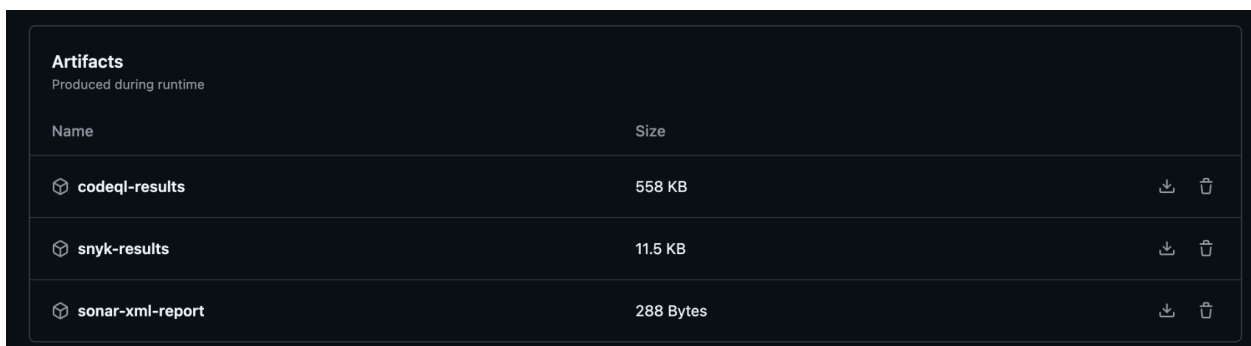
- **Dependabot** est utilisé pour surveiller et mettre à jour automatiquement les dépendances des projets. Les mises à jour sont planifiées quotidiennement.

4. Création et Déploiement de Docker:

- Une fois les tests et analyses terminés, le [pipeline](#) crée une image Docker à l'aide de docker build.
- L'image Docker est ensuite poussée vers [Docker Hub](#) avec le tag *latest*.

5. Rapports:

- Les résultats des analyses sont stockés et peuvent être consultés à tout moment dans la section '**Artifacts**' de notre workflow [security-tests.yml](#), comme illustré dans la Figure 4. Nous fournirons des échantillons des rapports sur GitHub Pages pour votre convenance à l'adresse suivante : <https://yelghom.github.io/dvna-master/security-tests-sample-reports/snyk-report>.






Artifacts	
Produced during runtime	
Name	Size
 codeql-results	558 KB
 snyk-results	11.5 KB
 sonar-xml-report	288 Bytes

Figure 4 : Artéfacts de notre workflow

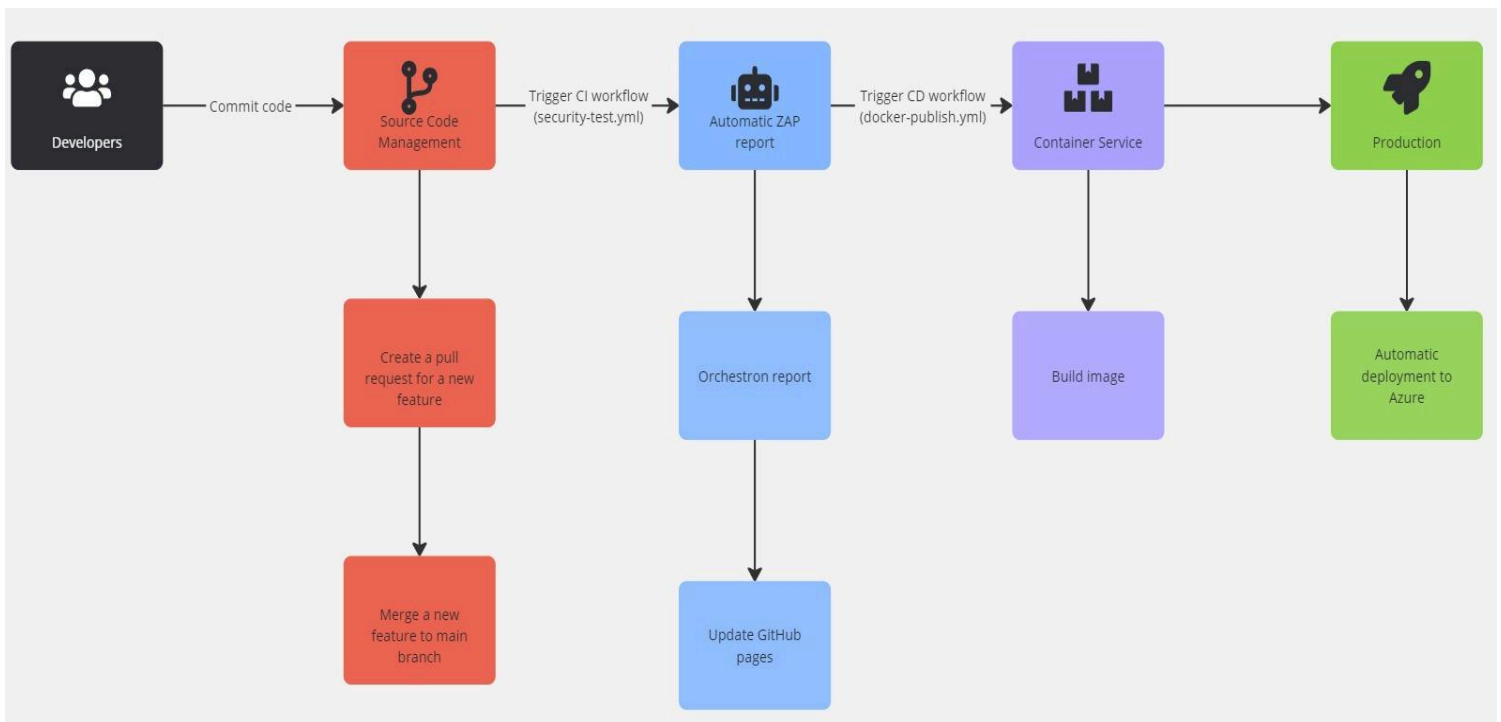


Figure 5 : Diagramme d'activité pour la séquence d'activité des pipelines CI et CD

Description du pipeline de déploiement continu (CD)

Étapes du Pipeline de Déploiement Continu (CD)

1. Déclenchement du Pipeline

- Le pipeline de déploiement continu (CD) se déclenche automatiquement dès qu'un push est effectué sur la branche principale (main) du dépôt GitHub. Cela permet de redéployer l'application chaque fois qu'un nouveau code ou une nouvelle image Docker est mis à jour.

2. Attendre le workflow de docker-publish

- Nous allons attendre [la fin du workflow](#) de création et publication de l'image Docker avant de commencer le déploiement sur Azure.

3. Connexion à Azure

- L'étape suivante consiste à établir une connexion sécurisée avec Azure en utilisant l'action `azure/login@v1`. Les informations d'identification nécessaires pour cette connexion sont stockées dans les **GitHub Secrets** sous le nom `AZURE_CREDENTIALS`.
- **Secrets Utilisés:**
 - `AZURE_CREDENTIALS`: Contient les informations d'identification d'Azure pour authentifier la connexion lors du déploiement de l'application.

4. Déploiement de l'Application vers Azure Container Apps

- Après la connexion à Azure, le pipeline procède au déploiement de l'image Docker la plus récente (générée et stockée sur Docker Hub) vers un service **Azure Container Apps**. L'action `azure/container-apps-deploy-action@v2` est utilisée pour ce processus. L'image Docker avec le tag `latest` (la plus récente) est extraite de Docker Hub et déployée dans le conteneur spécifié dans **Azure**.
- **Paramètres Utilisés:**
 - `containerAppName`: Le nom du conteneur dans lequel l'application est déployée (dans ce cas, `dvna-tp2-container`).
 - `resourceGroup`: Le groupe de ressources Azure où le conteneur est hébergé (`tp2`).
 - `imageToDeploy`: Spécifie l'image Docker à déployer. Dans ce pipeline, l'image `yelghom/dvna:latest` est extraite de Docker Hub et déployée.

Lien de l'application [ici](#).

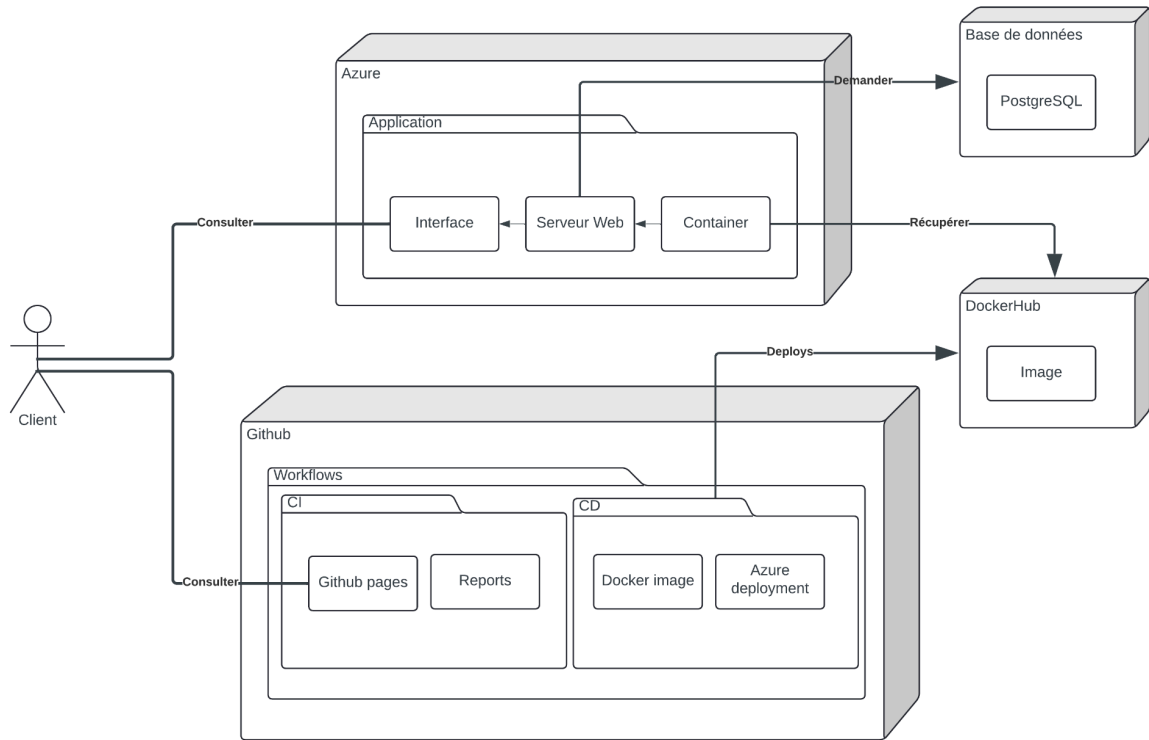


Figure 6 : Diagramme de déploiement de l'application

Conclusion

En conclusion, notre analyse a révélé plusieurs vulnérabilités de sécurité dans l'application, notamment des failles liées aux en-têtes de sécurité et à l'utilisation de bibliothèques JavaScript obsolètes. Pour renforcer la sécurité, nous recommandons de mettre à jour les bibliothèques vulnérables, d'ajouter des en-têtes comme Content Security Policy (CSP) et X-Content-Type-Options, ainsi que de configurer les cookies avec l'attribut SameSite pour prévenir les attaques. L'intégration d'outils de surveillance comme SonarQube et Dependabot permettra également de mieux gérer les vulnérabilités à l'avenir.