LOG8100 - DevSecOps - Opérations et dév. logiciel sécuritaire

Laboratoire 2: Tests de pénétration et Outils pour Intégration et Déploiement continus

Dumitru Zlotea 2081654
Hakim Mektoub 1956925
Haowen Li 2074847

Date de remise *: 24-10-2024*

Introduction

Dans le cadre de ce travail pratique, nous avons analysé l'application DVNA. Cette application est une application web spécifiquement créée pour démontrer les vulnérabilités OWASP et comment les mitiger. Pour tester l'application, nous avons utilisé l'application ZAP. ZAP est un outil open-source qui permet de scanner une application et d'effectuer des tests de pénétrations sur cette application. Nous avons aussi exploré l'application manuellement en injectant des requêtes SQL.

Analyse de sécurité

Les tests de pénétrations que nous avons effectués manuellement et avec l'outil ZAP nous ont permis de déceler une dizaine de vulnérabilités dont 5 moyennes, comme on peut le voir dans la capture d'écran de l'outil Orchestron ci-dessous:

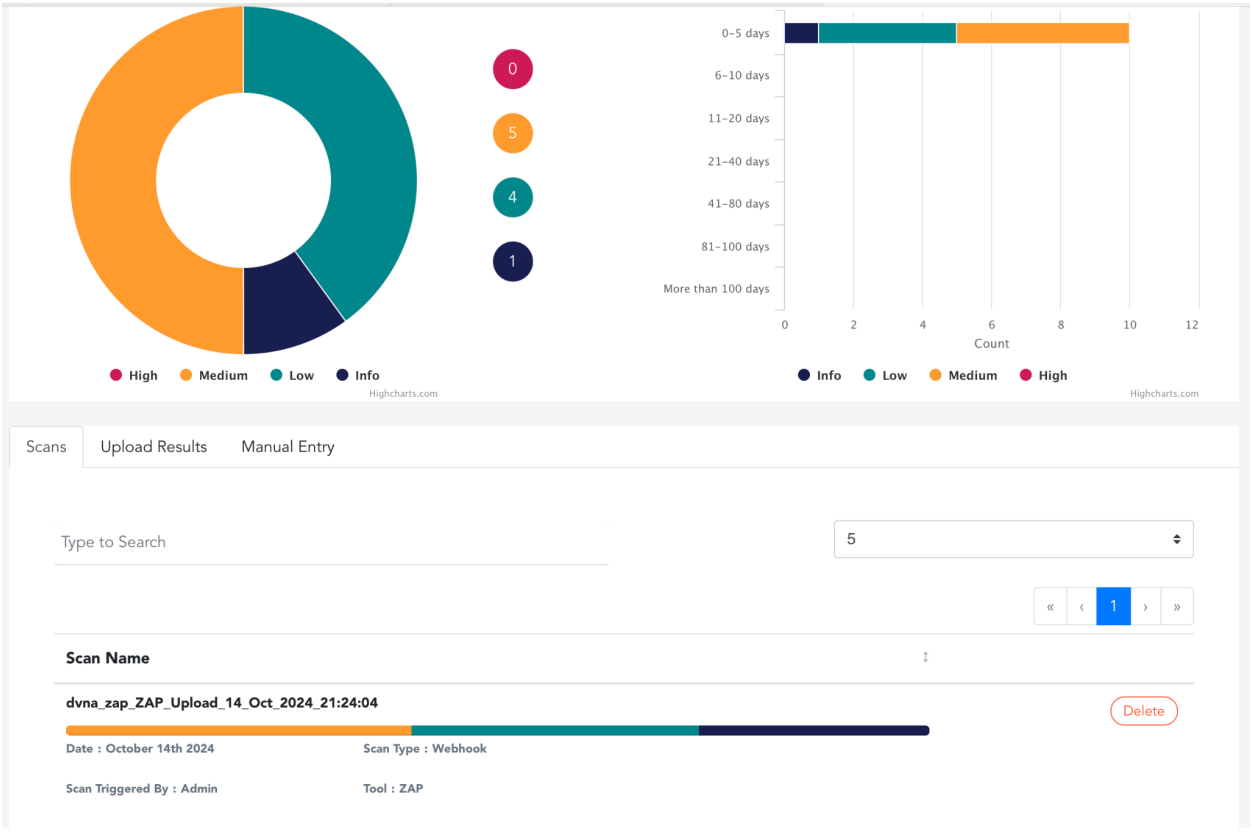
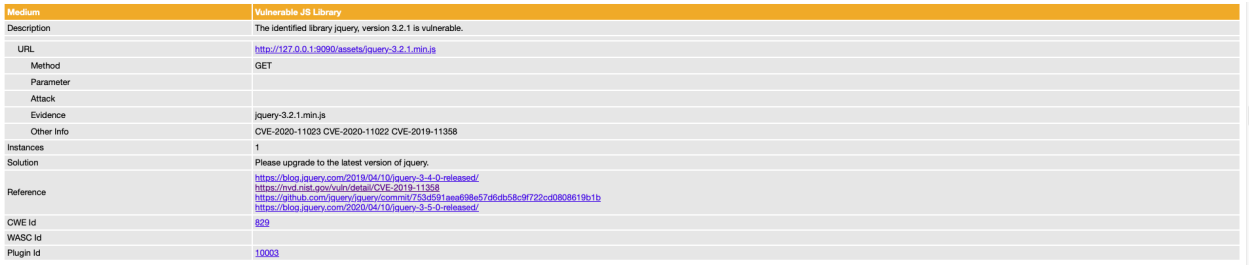


Figure 1. Capture d'écran d'Orchestron.

Parmi les vulnérabilités que nous avons relevées, nous allons en détailler 3 dans ce rapport.

1. Dans le TOP 10 OWASP, on retrouve A06-2021: Vulnerable and outdated components. Dans l'analyse roulée par ZAP, on a trouvé que l'application utilise une version vulnérable de JQuery, la version 3.2.1. Un CVE existe pour cette vulnérabilité: CVE-2019-11358. Pour mitiger cette vulnérabilité, il suffit de mettre à jour la version de JQuery. Jquery a fixé cette vulnérabilité à partir de la version 3.5.1. On recommande de mettre à jour à la version la plus récente de JQuery. Cette vulnérabilité est de gravité moyenne.



Medium	Vulnerable JS Library
Description	The identified library jquery, version 3.2.1 is vulnerable.
URL	http://127.0.0.1:9090/assets/jquery-3.2.1.min.js
Method	GET
Parameter	
Attack	
Evidence	jquery-3.2.1.min.js
Other Info	CVE-2020-11023 CVE-2020-11022 CVE-2019-11358
Instances	1
Solution	Please upgrade to the latest version of jquery.
Reference	https://blog.jquery.com/2019/04/10/jquery-3-4-0-released/ https://nvd.nist.gov/vuln/detail/CVE-2019-11358 https://github.com/jquery/jquery/commit/753d591aea698e57d6db58c9f722cd0808619b1b https://blog.jquery.com/2020/04/10/jquery-3-5-0-released/
CWE Id	829
WASC Id	
Plugin Id	10003

Figure 2. Résultat du rapport ZAP

2. En analysant l'application manuellement, nous avons aussi trouvé qu'elle est vulnérable à des injections SQL. Spécifiquement dans la page /app/usersearch. En injectant un apostrophe dans la barre de recherche, on voit qu'on reçoit une erreur. Ceci nous indique que les entrées ne sont pas validés. En injectant la requête suivante: *' UNION SELECT password,1 from Users --//*. On reçoit le mot de passe chiffré de l'utilisateur. Cette vulnérabilité est relevée dans le top 10 OWASP A03-2021. Pour mitiger cette vulnérabilité, il faut valider les entrées dans le code avant de soumettre les requêtes SQL. On peut par exemple valider que seuls les caractères A-Z, a-z et 0-9 sont autorisés dans les champs d'entrées utilisateurs. Il faut modifier le code dans: *core/appHandler.js*. Il s'agit d'une grave vulnérabilité.

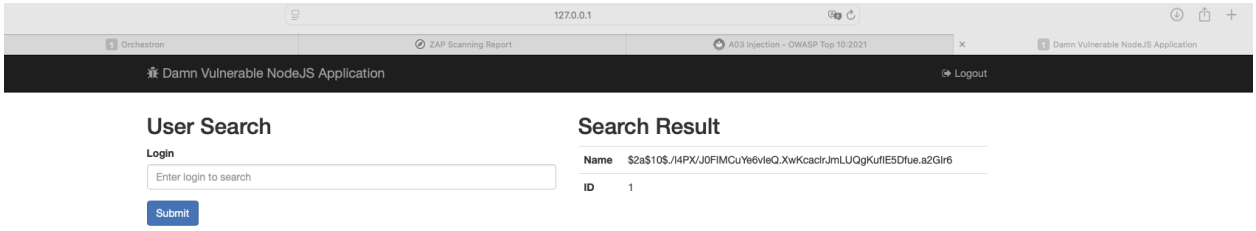


Figure 3. Résultat de l'injection SQL

3. Le rapport de l'analyse effectué par ZAP révèle aussi une vulnérabilité qui correspond au point de l'OWASP Top-10 : A-08-2021 - Software and Data Integrity Failures. En effet, on voit dans le rapport, que l'application utilise du code externe sans faire de validation sur l'intégrité de celui-ci. Ceci pourrait permettre à un utilisateur malicieux qui a eu accès au

serveur externe d'injecter du code malicieux dans l'application. On voit dans le rapport l'utilisation du code de bootstrap directement dans une directive script dans la page /forgotpw. Ceci est une vulnérabilité moyenne. On peut pallier cette vulnérabilité en s'assurant de valider l'intégrité du code provenant d'une tierce source en intégrant une signature digitale. Ci-dessous, on a l'extrait du rapport de ZAP.

Medium	Sub Resource Integrity Attribute Missing
Description	The integrity attribute is missing on a script or link tag served by an external server. The integrity tag prevents an attacker who have gained access to this server from injecting a malicious content.
URL	http://127.0.0.1:9090/forgotpw
Method	GET
Parameter	
Attack	
Evidence	<link id="bootstrap_styles" rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" type="text/css"/>
Other Info	
URL	http://127.0.0.1:9090/forgotpw
Method	GET
Parameter	
Attack	
Evidence	<script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
Other Info	
URL	http://127.0.0.1:9090/forgotpw
Method	GET
Parameter	
Attack	
Evidence	<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/jquery.bootstrapvalidator/0.5.3/js/bootstrapValidator.js"></script>
Other Info	
URL	http://127.0.0.1:9090/forgotpw
Method	GET
Parameter	
Attack	
Evidence	<script type="text/javascript" src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

Figure 4. Résultat du rapport ZAP

Description du pipeline d'intégration continue (CI)

Le pipeline d'intégration continue (CI) de notre projet repose sur trois outils principaux pour la sécurité et la gestion des dépendances : CodeQL, Snyk.io, et Dependabot. Chacun d'eux a un rôle spécifique et est intégré de manière à assurer que notre code est sécurisé, à jour, et exempt de failles ou de vulnérabilités potentielles avant toute mise en production. Lorsqu'un changement est détecté dans la branche principale, Ces trois outils seront déclenchés automatiquement à l'aide de Github Actions pour faire une analyse sur le code changé.

Analyse du code avec CodeQL

CodeQL aide à repérer les failles de sécurité et les erreurs directement dans le code. À chaque push ou pull request, il scanne tout le projet pour dénicher les mauvaises pratiques, les bugs, et les vulnérabilités. Il utilise des règles spécifiques pour fouiller dans la structure du code et identifier les soucis. Une fois l'analyse terminée, CodeQL génère un rapport avec une liste des problèmes et des suggestions pour les corriger. Tous les problèmes détectés par CodeQL peuvent être vus dans la page "Security" du repo Github.

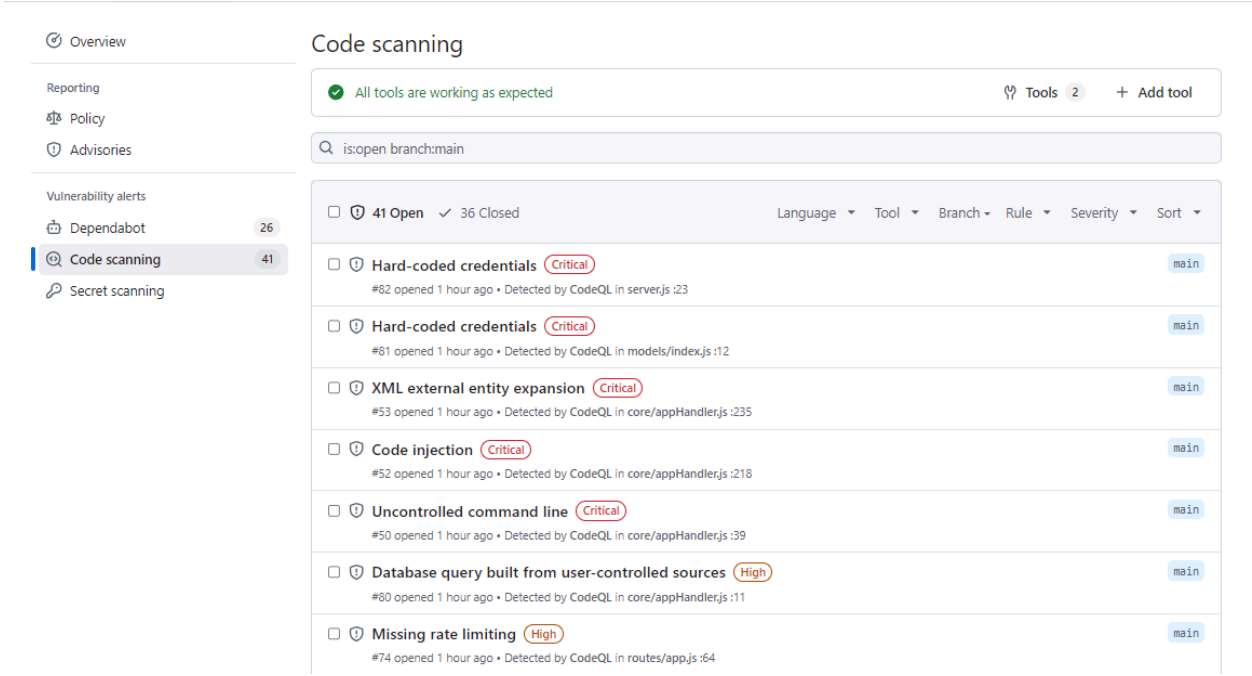


Figure 5. Résultat de CodeQL

Analyse de la sécurité avec Snyk.io

L'objectif de Snyk est de corriger les failles de sécurité dans les bibliothèques et dépendances de notre projet. À chaque fois qu'on pousse du code sur le dépôt, Snyk.io s'en charge en analysant nos dépendances, donc les librairies et packages qu'on utilise. Il les compare à une base de données de vulnérabilités connues pour identifier celles qui pourraient poser problème. Les résultats de l'analyse se trouvent dans la page "Actions" du repo Github.

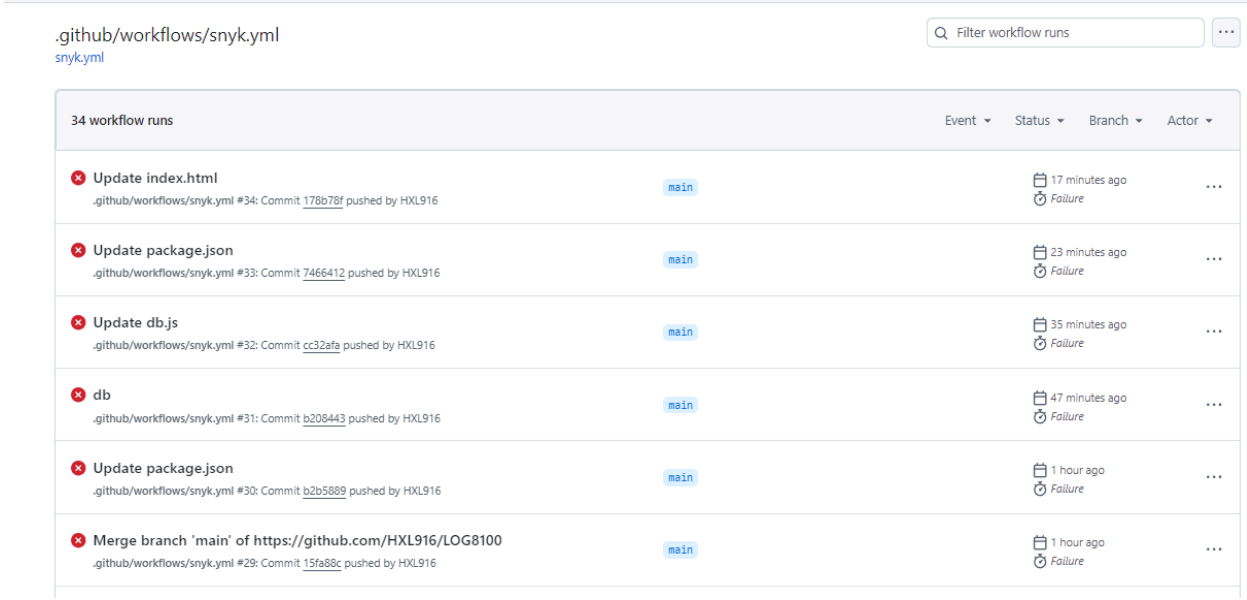


Figure 6. Résultat de snyk.yml

Mises à jour automatiques avec Dependabot

Dependabot permet de garantir que toutes les dépendances utilisées dans notre projet sont à jour, sécurisées et stables. Pour y parvenir, il surveille nos dépendances. Lorsqu'une nouvelle version d'une dépendance est publiée, Dependabot crée automatiquement une pull request pour proposer la mise à jour. Ces mises à jour peuvent inclure des corrections de bugs, des améliorations de performance ou des patchs de sécurité. Grâce à cette automatisation, les dépendances critiques sont toujours maintenues à jour sans intervention manuelle. Les vulnérabilités identifiées par Dependabot sont aussi dans la page “Security” du repo Github.

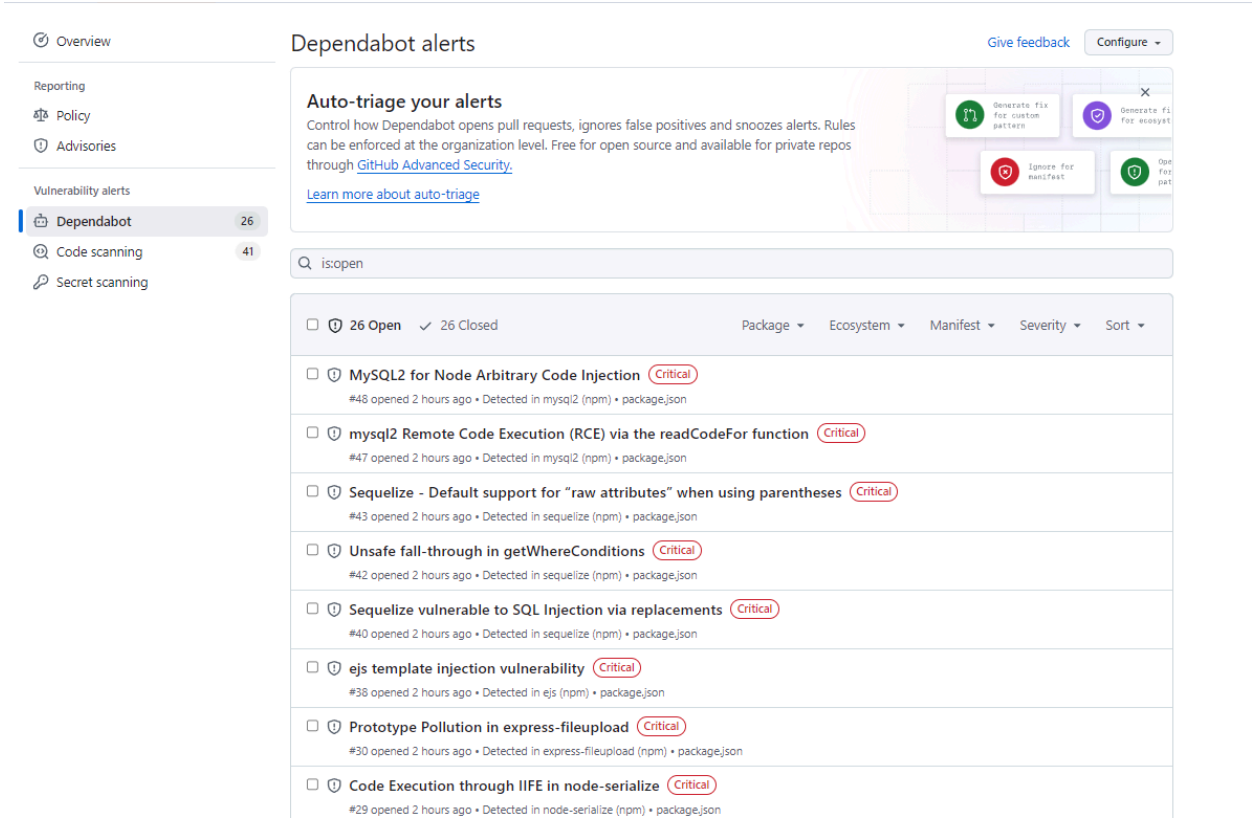


Figure 7. Résultat de Dependabot

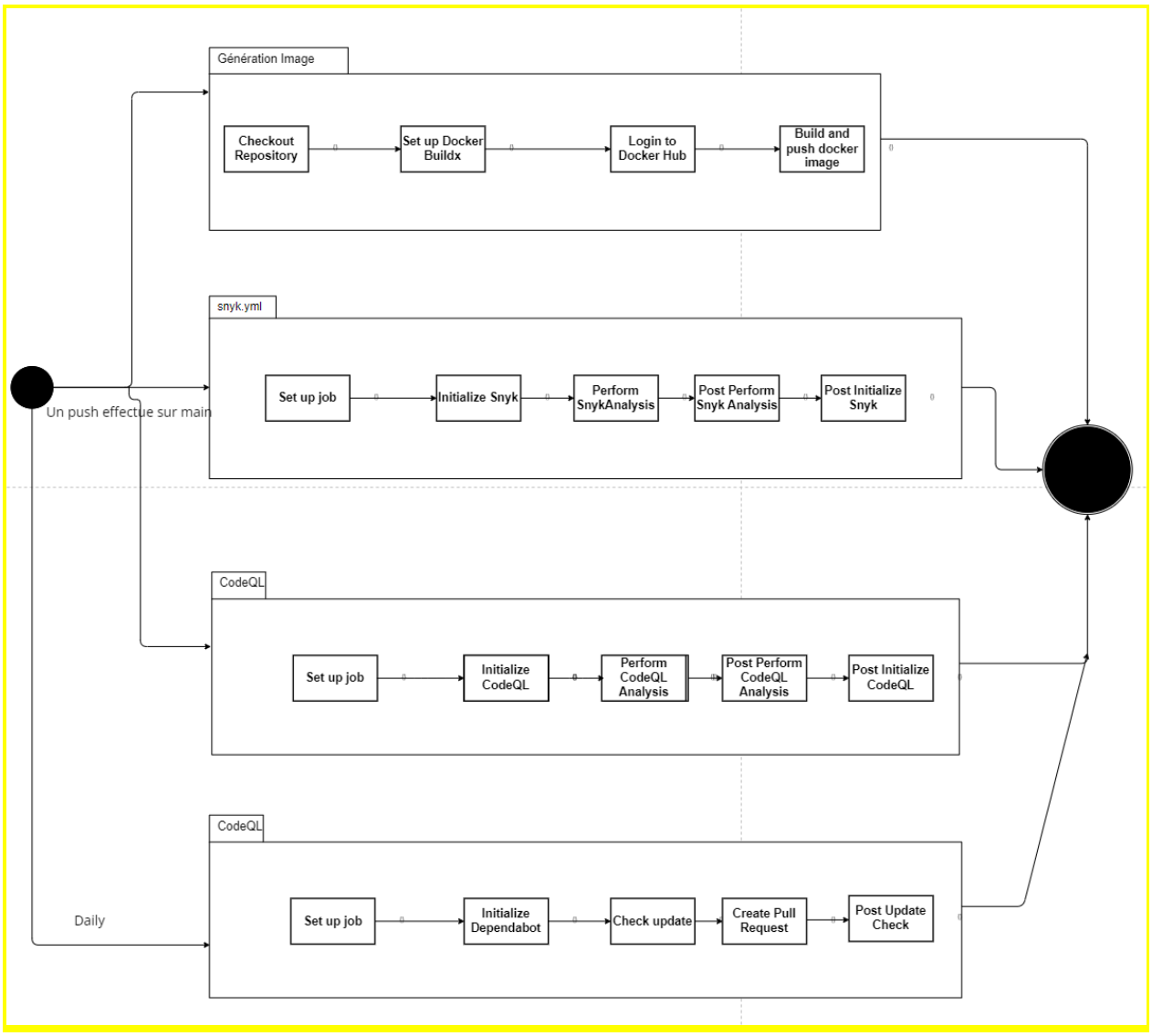


Figure 8. Diagramme d'état

Description du pipeline de déploiement continu (CD)

Étapes du processus de déploiement :

1. Développement local : Nous travaillons en local et poussons le code sur GitHub dès qu'une nouvelle fonctionnalité ou correction est prête.
2. Pipeline CI : GitHub Actions déclenche l'analyse du code (via Snyk.io, CodeQL, et Dependabot) pour s'assurer que tout est correct avant de continuer.
3. Création de l'image Docker : Une fois le code validé, GitHub Actions génère une nouvelle image Docker contenant notre application.
4. Push sur Docker Hub : L'image Docker est ensuite poussée sur Docker Hub, où elle est prête à être utilisée pour le déploiement.

5. Déploiement sur Heroku: Herokuutilise l'image Docker pour mettre à jour l'application déployé en ligne.

Pour commencer, nous utilisons Docker pour empaqueter notre application avec toutes ses dépendances. À chaque mise à jour, une nouvelle image Docker est créée, contenant tout ce qu'il faut pour exécuter l'application de manière reproductible, peu importe l'environnement. Nous utilisons également GitHub Actions pour automatiser la création de l'image Docker et son envoi vers Docker Hub. Ce processus est entièrement automatisé, configuré avec le fichier “build-and-deploy-image.yml”.

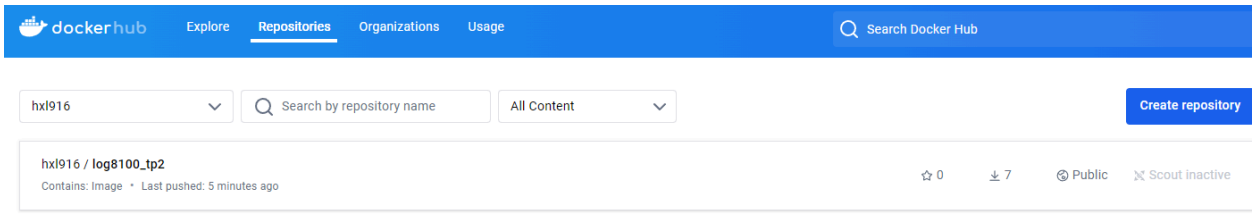


Figure 9. DockerHub

L'image Docker générée est ensuite utilisé pour mettre à jour l’application déployé sur Heroku. En ce qui concerne la gestion des données, notre base de données PostgreSQL est connectée à l'application une fois celle-ci mise en ligne, prenant en charge toutes les opérations liées aux données, comme le stockage et la récupération.

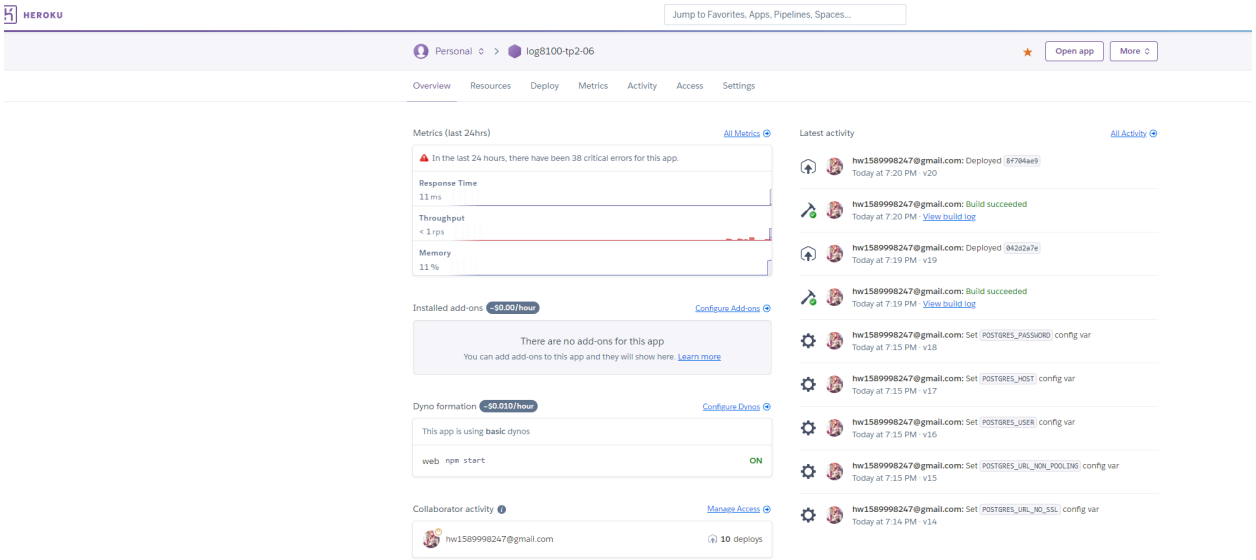


Figure 10. L’application déployé sur Heroku

Voici quelques lien utiles pour accéder à l'application:

[Lien Documentation]

[Lien Déploiement]

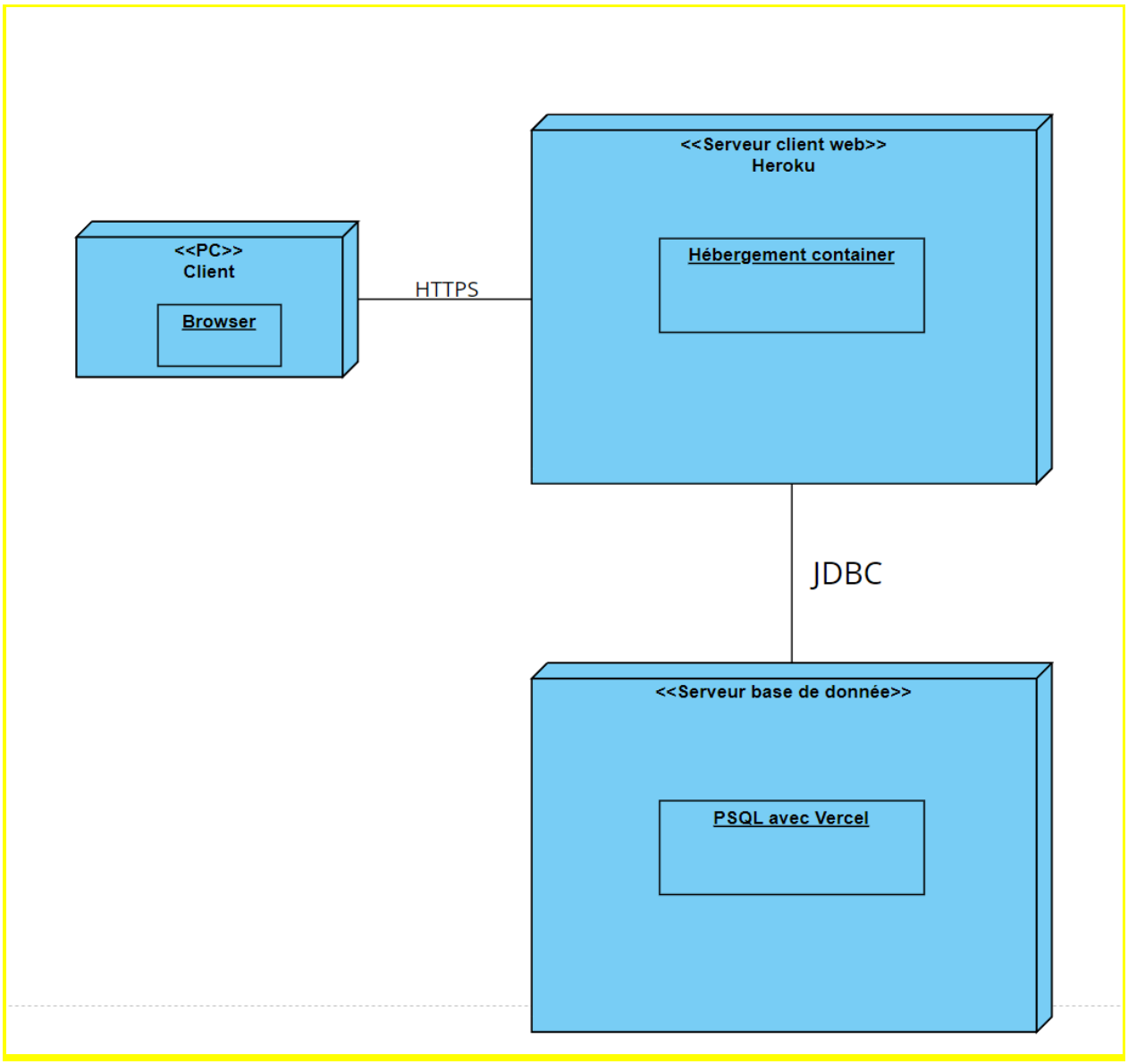


Figure 11. Diagramme de déploiement

Conclusion

En résumé, après avoir apporter quelques modifications mineures pour l'adapter à notre besoin. Nous avons aussi utiliser OWASP ZAP et Orchestron pour identifié des alertes de gravité varié. Suite à cela, nous avons intégré des outils comme DependaBot, CodeQL, synk.io et les avons ajouté dans notre pipeline de développement. En automatisant ces workflows il devient plus facile d'identifier les problèmes de sécurité peu importe à l'état de développement ou de production.