



**POLYTECHNIQUE
MONTRÉAL**

UNIVERSITÉ
D'INGÉNIERIE

DevSecOps

LOG8100– AUTOMNE 2024

Travail pratique 2

**Tests de pénétration et
Outils pour Intégration et
Déploiement continu**

Soumis par

Nina Cussac

Alek Delisle

Bryan Junior Ngatshou

Soumis le

24 octobre 2024

Table des matières

Introduction	2
Vulnérabilités trouvées	3
Rapport de vulnérabilités avec le scan automatique de OWASP ZAP	6
Rapport de test avec Orchestron	7
Description de la pipeline d'intégration en continu	8
Description de la pipeline de déploiement en continue	10
Annexe	13

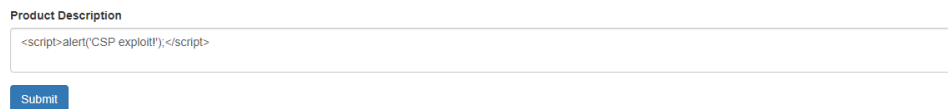
Introduction

Ce rapport présente l'analyse de sécurité effectuée sur une application nommée ***Damn Vulnerable NodeJS Application (DVNA)***. C'est une application NodeJS conçue pour démontrer les vulnérabilités du Top 10 OWASP tout en offrant des solutions pour les corriger et les éviter. Sur cette application nous avons étudié les résultats de PenTest obtenus à l'aide de deux applications : OWASP ZAP et Orchestron. OWASP ZAP est un outil de test de pénétration permettant de détecter les vulnérabilités dans les applications web et Orchestron a été utilisé pour générer des rapports de vulnérabilités à partir des résultats de tests obtenus avec OWASP ZAP et pour centraliser la gestion des résultats de sécurité dans un environnement CI/CD.

Vulnérabilités trouvées

1. Injection de script (Cross-Site Scripting - XSS)

Dans la section pour ajouter et éditer des produits, il est possible d'injecter des scripts dans les champs de texte. Cette vulnérabilité est de type Nous donnons ici un exemple pour démontrer que le script est exécuté une fois que le formulaire est soumis. Bien que ce script ne soit pas dangereux pour les utilisateurs de l'application dans ce cas-ci, des scripts pouvant rediriger l'utilisateur vers un site web malicieux ou rediriger la communication vers un autre serveur pourraient être ajoutés, ce qui rendrait les utilisateurs vulnérables à des attaques.



Product Description

`<script>alert("CSP exploitt!"); </script>`

Submit

Figure 1. Ajout du script causant une alerte dans la page "List Products"

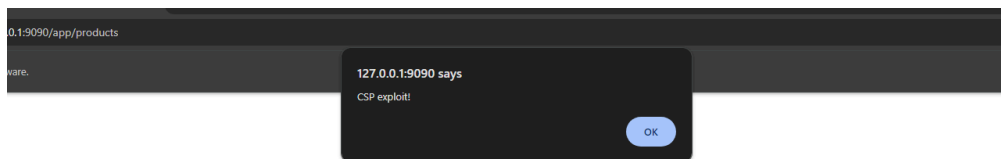
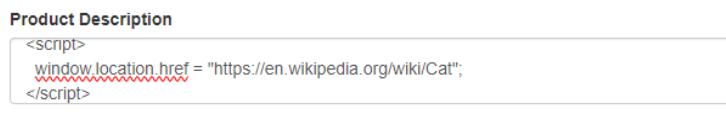


Figure 2. Alerte sur la page "Liste Products" dès qu'elle est ouverte



Product Description

`<script>
window.location.href = "https://en.wikipedia.org/wiki/Cat";
</script>`

Figure 3. Exemple de script qui redirige l'utilisateur vers un autre site web

Cette vulnérabilité est définie, par ZAP, comme étant un risque moyen pour la sécurité de l'application. C'est une vulnérabilité de type CWE-79 et elle fait partie du OWASP Top 10 dans la catégorie: "A07:2021 - Identification et authentification manquantes". Pour contrer cette vulnérabilité, il suffit d'ajouter une validation des entrées, et/ou de transformer les caractères pouvant causer problème, exemple changer ">" pour ">".

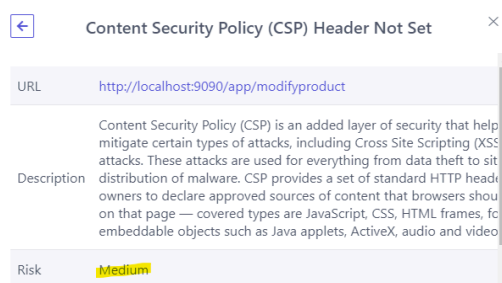


Figure 4. Classification de la vulnérabilité par ZAP

2. Exposition de données sensibles

À l'aide d'une fouille manuel en utilisant ZAP, nous avons découvert une vulnérabilité importante qui expose les hachages de mots de passe. En effet, il suffit de se rendre sur l' URL suivant: **https://127.0.0.1:9090/app/admin/usersapi**. Sur cette page on retrouve la table des utilisateurs, avec entre autres, leur mots de passe hachés, leur email, etc. Cette vulnérabilité est un mélange de deux éléments du Top 10 OWASP: A01:2021 - Broken Access Control et A02:2021 - Cryptographic Failures. En effet, les accès aux données sensibles sont mal gérés et les données sensibles ne sont pas encryptées. C'est une vulnérabilité à risque élevé pour la sécurité, car les informations sont facilement accessibles et par l'importance de ces informations pour l'application.

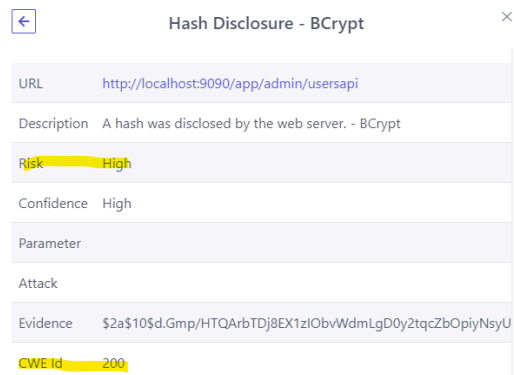


Figure 5. Résumé de la vulnérabilité trouvée avec la fonctionnalité “Manual Explore” de ZAP

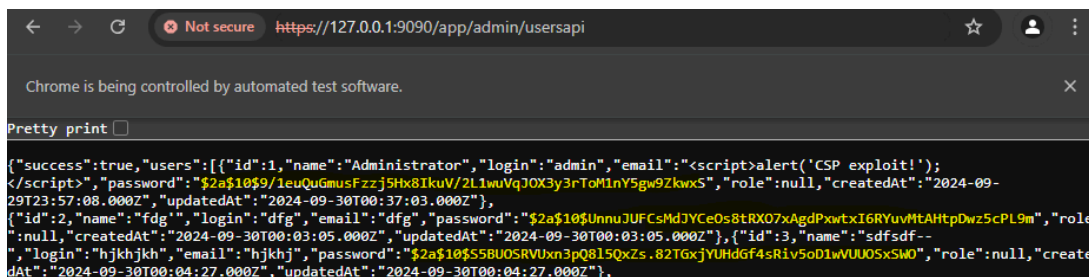


Figure 6. Page comportant la table des utilisateurs (données sensibles)

Malgré le fait que les mots de passe soit hachés, ceci permet aux attaquants d'avoir des opportunités d'attaques, par exemple en utilisant des “rainbow tables” pour déchiffrer les mots de passe.

Plusieurs contre-mesures pour cette vulnérabilité sont possibles. Une première solution serait de s'assurer que seul les personnes ayant un rôle d'administrateur ont accès à l'information contenu dans cette API et, pour ajouter une couche de sécurité, chiffrer ces données en plus du hachage de mots de passe. Sinon, si possible, simplement ne pas avoir d'API divulguant cette information.

3. Injection SQL

Dans l'application dans la route `/app/usersearch` il est possible de trouver l'id et le nom de tous les utilisateurs. En effet, si l'on fait une injection SQL comme celle-ci: `' or '1'='1' and id>'1`, on peut retrouver tous les utilisateurs un par un. La partie `" ' or '1' = '1' "` permet de faire la recherche suivante: **SELECT name,id FROM Users WHERE login=" or '1' = '1'**, la condition sera donc toujours vraie et il sera donc possible d'obtenir tous les utilisateurs. Cependant, la fonction donnant les résultats de la recherche, ne montre que le premier élément, alors on ajoute la partie `"and id>'1"` pour éliminer le id 1 dans ce cas si et obtenir le id=2, ensuite si l'on veut le id=3 nous pourrions remplacer le 1 dans l'injection par 2 et ainsi de suite.

User Search

Login

Submit

Search Result

Name	a
ID	2

Figure 7. Injection SQL pour trouver le nom pour l'utilisateur ayant l'id=2

```
module.exports.userSearch = function (req, res) {
  var query = "SELECT name,id FROM Users WHERE login='" + req.body.login + "'";
  db.sequelize.query(query, {
    model: db.User
  }).then(user => {
    if (user.length) {
      var output = {
        user: {
          name: user[0].name,
          id: user[0].id
        }
      }
      res.render('app/usersearch', {
        output: output
      })
    }
  })
}
```

Figure 8. Partie du code permettant l'injection dans le fichier `appHandler.js`

Cette vulnérabilité est considérée comme ayant un risque moyen sur l'application. Elle fait partie du top 10 de l'OWASP: A03:2021 - Injections et est catégorisée comme étant une CWE-89: "SQL injection". Une manière sécuritaire de régler le problème d'injection SQL dans cette partie du code, serait d'utiliser une requête paramétrée au lieu de directement concaténer le texte brute dans la requête SQL.

```
module.exports.userSearch = function (req, res) {
  var query = "SELECT name, id FROM Users WHERE login = :login";
  db.sequelize.query(query, {
    replacements: { login: req.body.login },
    model: db.User
  })
}
```

Figure 9. Solution pour empêcher les injection SQL pour la recherche d'utilisateur

L'utilisation de **replacements** dans **sequelize** permet d'échapper les caractères spéciaux comme les guillemets, donc la tentative d'injection SQL, vu plutôt, ressemblerait à ceci:

SELECT name, id FROM Users WHERE login = '\ ' OR\ '1\='1\ AND\ id\ >\ '1\'. Il deviendrait donc impossible d'effectuer des injections SQL pour la recherche d'utilisateurs.

Rapport de vulnérabilités avec le scan automatique de OWASP ZAP

Par la suite, nous avons utilisé la fonctionnalité de Scan automatique de l'application avec OWASP ZAP. Voici les résultats obtenus :

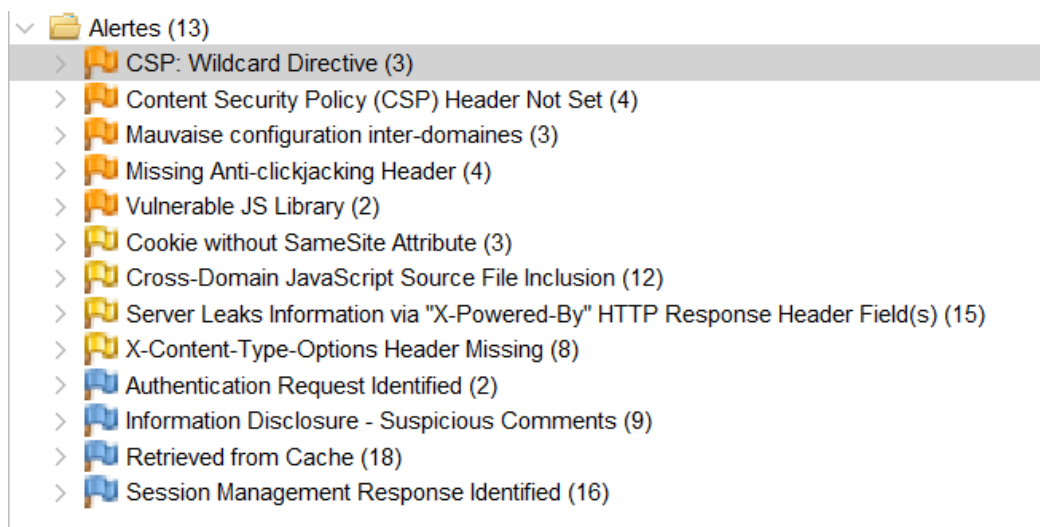


Figure 10. Résultats du scan automatique sur notre application avec ZAP

En ne prenant que les vulnérabilités appartenant au top 10 de l'OWASP voici les vulnérabilités que nous avons détecté dans l'application:

Concernant la navigation sur le site :

- Content Security Policy (CSP) Header Not Set : A05:2021 Security Misconfiguration
L'absence d'en-têtes CSP peut exposer le site à des attaques XSS et inclusion de contenu non sécurisé.
- Cookie without SameSite Attribute : A05:2021 Security Misconfiguration
Les cookies sans attribut SameSite peuvent exposer l'application à des attaques CSRF.
- Cross-Domain JavaScript Source File Inclusion : A08:2021 Software and Data Integrity Failures
Inclure des fichiers JavaScript provenant de domaines croisés peut exposer à des attaques si les fichiers externes sont compromis.

Concernant la soumission de formulaires:

- Authentication Request Identified : A07:2021 Identification and Authentication Failures
Les faiblesses dans les mécanismes d'authentification peuvent conduire à l'usurpation d'identité ou à des accès non autorisés.
- Session Management Response Identified : A07:2021 Identification and Authentication Failure
Les faiblesses dans la gestion des sessions peuvent entraîner le vol de sessions et l'usurpation d'identité.

Rapport de test avec Orchestron

Après avoir analysé notre application avec OWASP ZAP, on a généré un rapport en xlm et l'avons donné à Orchestron, voici l'analyse obtenue avec cet outil :

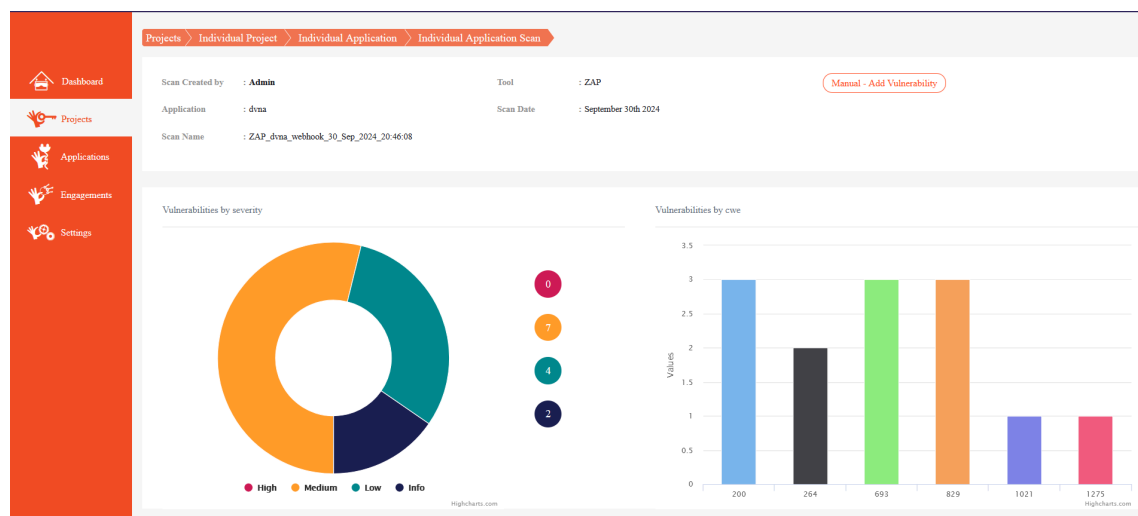


Figure 11. Résultats de l'analyse obtenue sur Orchestron (partie 1)

List of Vulnerabilities

Type to Search

« < 1 2 3 > »

Vulnerability Name
Information Disclosure - Suspicious Comments CWE : 200
Cross-Domain JavaScript Source File Inclusion CWE : 829
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) CWE : 200
Missing Anti-clickjacking Header CWE : 1021
Cookie without SameSite Attribute CWE : 1275

Figure 12. Résultats de l'analyse obtenue sur Orchestron (partie 2)

On constate donc un total de 11 vulnérabilités. Cependant, aucune vulnérabilité n'est présentée comme ayant une gravité forte sur la sécurité de l'application. Il est toutefois important de résoudre les vulnérabilités présentées dans le rapport, car celles-ci, en particulier celles faisant partie de la catégorie médium, peuvent être une porte d'entrée pour les attaquants.

Description de la pipeline d'intégration en continu

Une des méthodes de développement de logiciel DevOps par laquelle les développeurs intègrent régulièrement leurs modifications dans un répertoire centralisé. Pour ce travail, le répertoire utilisé est celui de Github¹ et nous avons utilisé la plateforme *GitHub Actions* offerte par ce répertoire et permettant d'automatiser la construction et les tests pendant l'intégration du code.

Tout d'abord cette automatisation se fait en écrivant un fichier `.yaml`² décrivant le flux de travail (workflow) qui doit être automatisé. Pour ce travail, nous avons choisi d'automatiser les tests de sécurité et de qualité de code, en utilisant les outils Snyk³ et Sonarqube⁴.

Ensuite nous avons configuré les différents outils utilisés afin de les intégrer dans notre flow.

- Pour Sonarqube, nous utilisons la version community que nous hébergeons dans une machine virtuelle Azure. Nous avons ensuite récupéré les codes secrets que nous utiliserons pour l'analyse du code. Ces codes secrets sont enregistrés dans github.

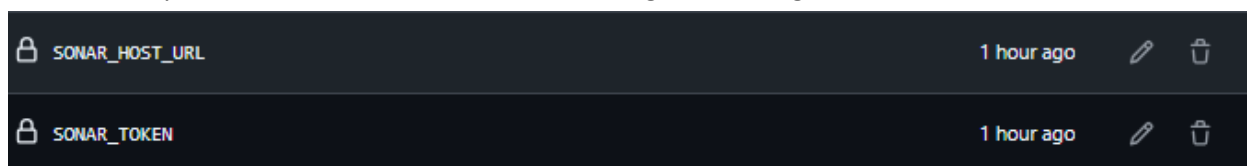


Figure 13. Codes secrets de Sonarqube utilisés dans le flux de travail

- Pour Snyk, nous avons créé un compte et ensuite utilisé le jeton qui nous était fourni afin de l'intégrer dans notre pipeline



Figure 14. Code secrets de de Snyk utilisé dans le flux de travail

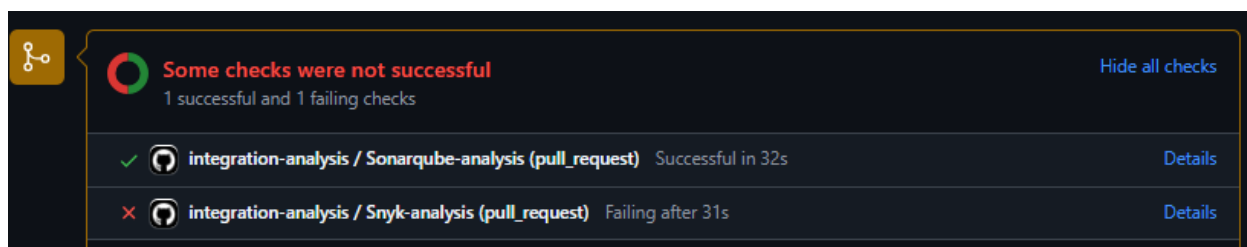


Figure 15. Flux de travaux d'analyse utilisant Sonarqube et Snyk

¹ Service d'hébergement de projet

² Fichiers communément utilisés pour créer des configurations

³ <https://snyk.io/fr/>

⁴ <https://www.sonarsource.com/products/sonarqube/>

De plus, nous avons ajouté un pipeline d'intégration qui automatise la gestion de dépendances en utilisant le dépendaBot (Annexe C). Ce workflow est effectué 1 fois par semaine après un pull-request. Comme on peut le voir dans la figure 15. Ce flux se fait donc en deux travaux (jobs) et se déclenche à chaque fois qu'un des développeurs veut intégrer son travail dans la branche main (par *pull-request* ou par *push*). Chaque travail est effectué dans une machine virtuelle fournie par la plateforme, utilisant la dernière version du système d'exploitation ubuntu. Selon la configuration que nous avons donnée dans le fichier de configuration. Nous pouvons également observer que le travail effectué par Snyk échoue, indiquant la présence de vulnérabilités dans le code. Il est à noter que ce projet utilise une vieille version de node (node 8) et corriger les paquets utilisés dans celui-ci, un par un, n'est pas à la portée de ce travail. Le fichier de ce flux de travail est présenté dans l'annexe A.

La figure 16 présente en diagramme d'activité, les étapes configurées pour chaque flux.

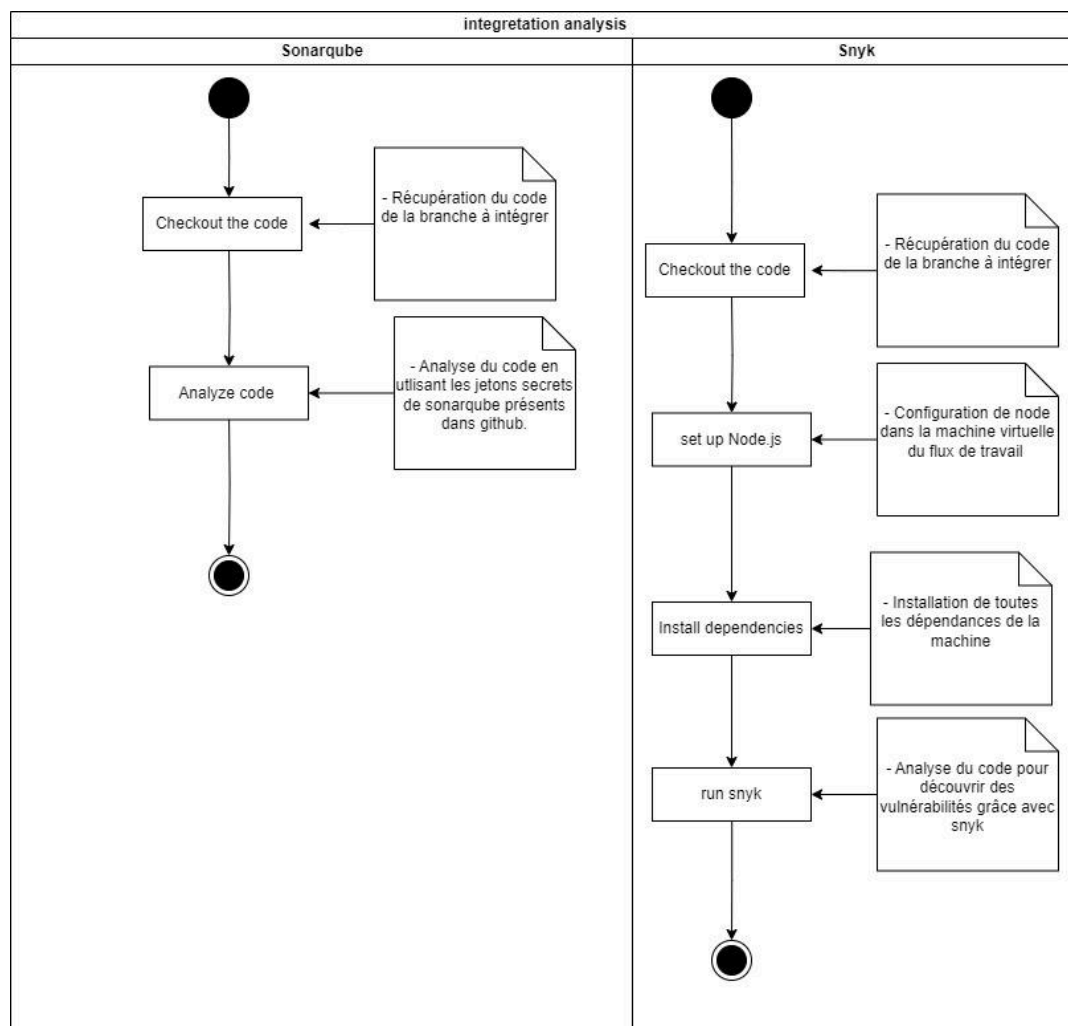


Figure 16. Diagramme d'activités de l'analyse du code intégré par pull request

Description de la pipeline de déploiement en continue

Après avoir automatisé l'intégration du code. Il serait donc nécessaire d'automatiser son déploiement en continue dans un environnement de production.

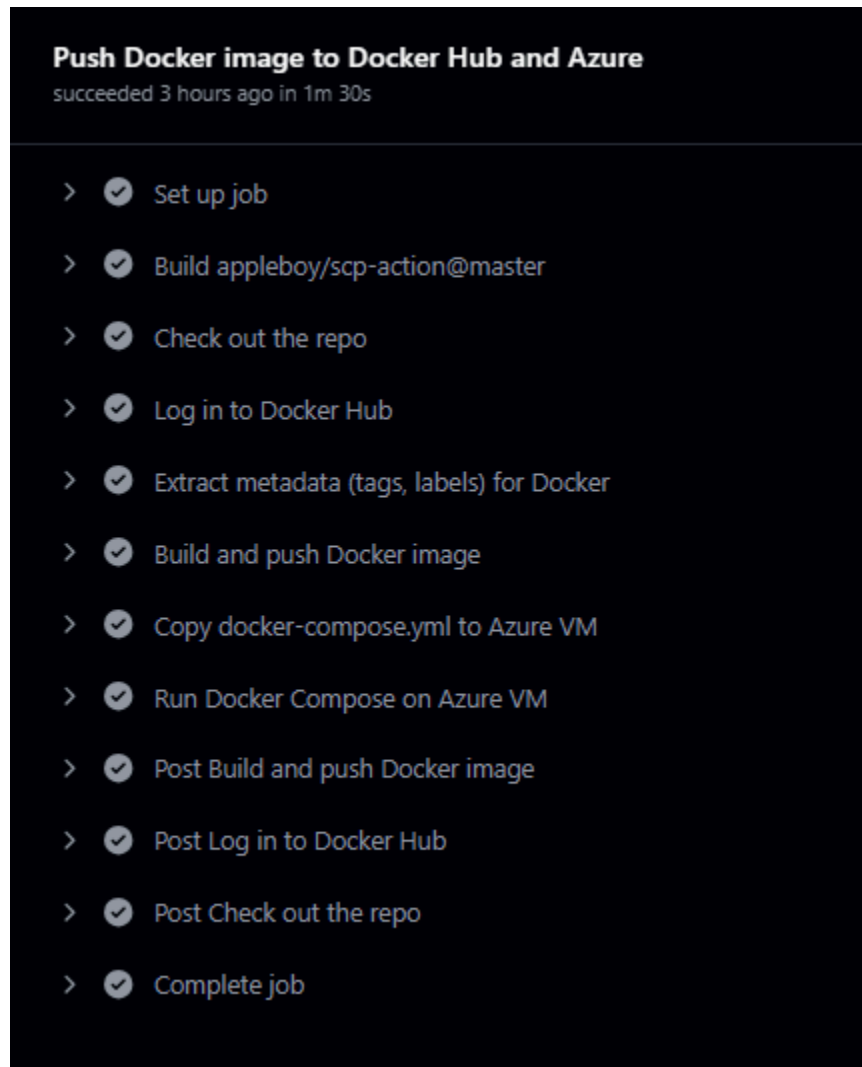


Figure 17: étapes de flux de travail du déploiement de l'application vers la machine virtuelle hébergée sur azure

Pour ce travail, nous avons choisi de déployer une image docker du projet dans une machine virtuelle azure. Les étapes de ce processus, configuré dans le fichier de flux de travail de déploiement (Annexe B) et présentées dans la figure 17, sont les suivantes:

1. Récupérer le code à déployer

Dans cette étape le code intégré et testé dans la branche main est récupéré par la flux de travail grâce à des outils disponibles dans la plateforme github-actions. Dans notre cas, *actions/checkout@v4*⁵

2. Connexion au DockerHub ⁶

Cette étape se fait après la création d'un compte dans le hub Docker. Les informations d'identifications sont ensuite récupérées et sauvegardées comme secret de projet dans github.

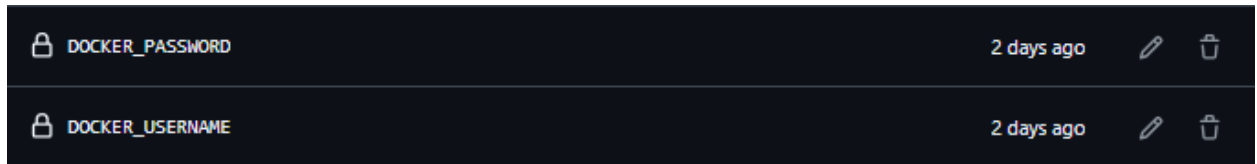


Figure 18. Secrets de connexion vers Docker Hub utilisés dans le flux de travail de déploiement.

3. Extraction des métadonnées de l'image

Dans cette étape, le flux génère des métadonnées utilisée pour identifier l'image construite telles que l'étiquette (tag en anglais) ou encore le libellé (label en anglais)

4. Construction de l'image et mise en place dans le Docker Hub

L'image est ainsi construite en utilisant le fichier de configuration de l'image Dockerfile présent dans le répertoire de code ainsi que les métadonnées générées dans l'étape précédente.

5. Copie du fichier docker-compose.yml dans la machine virtuelle

Il peut arriver que le fichier de lancement des images pour le projet puisse changer. C'est la raison pour laquelle nous avons choisi d'intégrer son transfert vers la machine virtuelle à chaque déploiement. La copie se fait par scp⁷ grâce aux informations d'identification de la machine contenu sur github.

6. Rouler docker compose dans la machine virtuelle:

Dans cette étape, nous nous connectons par ssh⁸ à la machine virtuelle et démarrons le script permettant de supprimer l'ancienne version de l'image, d'installer une nouvelle version et de la lancer.

Après toutes ces étapes, les utilisateurs pourront avoir accès à l'application via internet en suivant le lien suivant: <https://inf8100-tp2.nbj-dev.me>. La figure 18 montre le diagramme de déploiement de l'application.

⁵ Outil de github actions permettant de récupérer le code dans une machine virtuelle temporaire

⁶ Service fournit par docker permettant de répertorier, trouver et partager des images docker

⁷ Protocole de copie sécuritaire permettant de transférer des fichiers entre deux machines distantes.

⁸ Permet d'avoir accès à un ordinateur à distance de façon sécurisé dans un réseau non sécurisé

Le lien vers le repo github est le suivant <https://github.com/bgsub/TP2-DVNA> et pour avoir accès à la documentation déployée via github pages: <https://bgsub.github.io/TP2-DVNA/>.

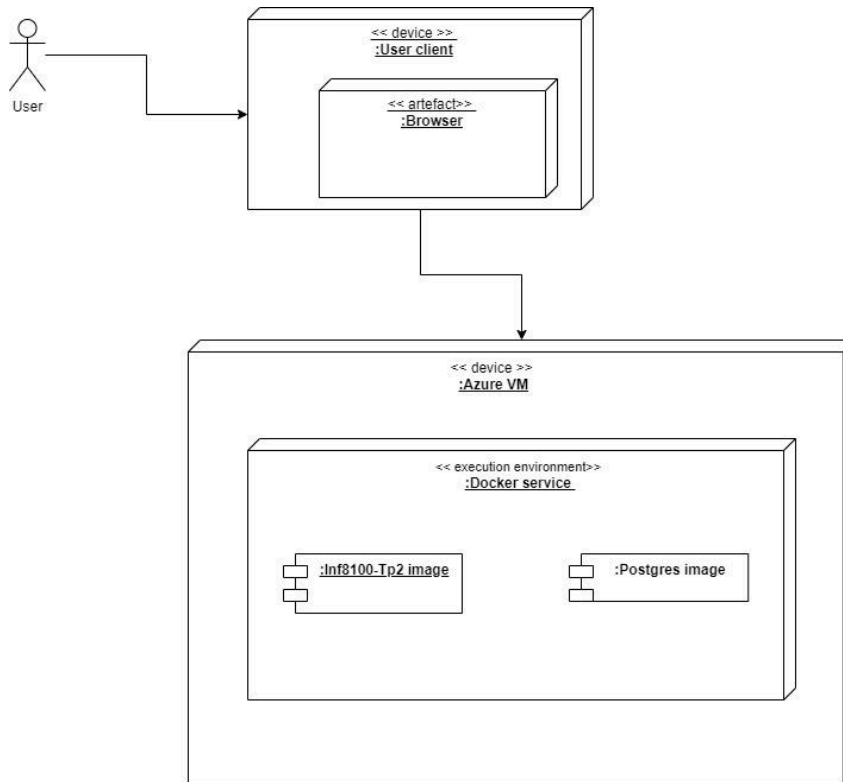


Figure 19. Diagramme de déploiement de l'application utilisé pour le tp2 du cours INF8100

Conclusion

Ce travail pratique nous a permis d'explorer en profondeur les vulnérabilités de sécurité associées à l'application étudiée *Damn Vulnerable NodeJS Application (DVNA)* et de mettre en place des outils pour les identifier et les corriger. En utilisant *OWASP ZAP* et *Orchestron*, nous avons pu identifier des vulnérabilités critiques comme les injections SQL, les expositions de données sensibles, et les scripts intersites (XSS). Puis nous avons intégré une pipeline CI/CD, qui inclut des tests de sécurité automatisés avec *Sonarqube* et *Snyk*. Cela nous a permis de constater l'efficacité et l'importance de l'automatisation des processus de détection et de correction des vulnérabilités. Nous avons démontré que l'utilisation de bonnes pratiques de développement est essentielle pour améliorer la robustesse des applications.

Annexe

```
on:
  pull_request:
    types: [opened, synchronize, reopened]
  push:
    branches: [main]

name: integration-analysis
jobs:
  sonarqube:
    name: Sonarqube-analysis
    runs-on: ubuntu-latest
    steps:
      - name: checkout code
        uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - name: analyze code
        uses: sonarsource/sonarqube-scan-action@v3
        env:
          SONAR_TOKEN: ${ secrets.SONAR_TOKEN }
          SONAR_HOST_URL: ${ secrets.SONAR_HOST_URL }

  security:
    name: Snyk-analysis
    runs-on: ubuntu-latest
    steps:
      - name: Check out code
        uses: actions/checkout@v4

      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '10'
```

```

- name: Install dependencies
  run: npm install

- name: Run Snyk to check for vulnerabilities
  uses: snyk/actions/node@master
  env:
    SNYK_TOKEN: ${ secrets.SNYK_TOKEN }

```

Figure 20. Fichier de flux de travail de d'intégration en continue

```

name: Publish Docker image

on:
  push:
    branches: ["main"]

jobs:
  build-and-deploy:
    name: Push Docker image to Docker Hub and Azure
    runs-on: ubuntu-latest
    permissions:
      packages: write
      contents: read
      attestations: write
      id-token: write
    steps:
      - name: Check out the repo
        uses: actions/checkout@v4

      - name: Log in to Docker Hub
        uses: docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a
        with:
          username: ${ secrets.DOCKER_USERNAME }
          password: ${ secrets.DOCKER_PASSWORD }

      - name: Extract metadata (tags, labels) for Docker
        id: meta
        uses: docker/metadata-action@9ec57ed1fcd8bf14dcef7dfbe97b2010124a938b7
        with:
          images: bgsub/inf8100

```

```

- name: Build and push Docker image
  id: push
uses: docker/build-push-action@3b5e8027fcad23fda98b2e3ac259d8d67585f671
with:
  context: .
  file: ./Dockerfile
  push: true
  tags: bgsub/inf8100:latest

- name: Copy docker-compose.yml to Azure VM
  uses: appleboy/scp-action@master
  with:
    host: ${ secrets.AZURE_IP }
    username: ${ secrets.AZURE_USER }
    key: ${ secrets.AZURE_SSH_KEY }
    source: "docker-compose.yml"
    target: "~/ "

- name: Run Docker Compose on Azure VM
  uses: appleboy/ssh-action@master
  with:
    host: ${ secrets.AZURE_IP }
    username: ${ secrets.AZURE_USER }
    key: ${ secrets.AZURE_SSH_KEY }
    script: |
      docker stop azureuser-inf8100-tp2-1
      docker system prune -a -f
      docker-compose up -d

```

Figure 21. Fichier de flux de travail de déploiement en continue

```

version: 2
updates:
- package-ecosystem: "npm"
  directory: "/"
  schedule:
    interval: "weekly"

```

Figure 22. Fichier yml de la pipeline du dependaBot