Universidad Tecnológica de Panamá
Facultad de Ingeniería de Sistemas Computacionales
Departamento de Arquitectura y Redes de Computadoras
Licenciatura en Ciberseguridad

Integrantes:
Juan Ojeda
Cristian Mendoza

Proyecto Semestral

Materia:
Ciberseguridad V

Profesor:
Eldezel García

2025

Objetivo General

El objetivo principal de este proyecto es diseñar e implementar un flujo completo de integración y entrega continua (CI/CD) basado en la metodología DevSecOps, que incluya análisis automatizados de seguridad para garantizar la calidad y protección del software durante todo su ciclo de vida. Este flujo se centra en automatizar y unificar distintos tipos de análisis de seguridad tales como el Análisis Estático de Seguridad (SAST), Análisis de Composición de Software (SCA) y Análisis Dinámico de Seguridad (DAST), mediante la integración de herramientas especializadas como SonarQube, Trivy y OWASP ZAP dentro de pipelines configurados en GitLab CI/CD. Además, se busca evaluar vulnerabilidades no solo en el código fuente sino también en las dependencias y contenedores usados para la aplicación, con el fin de demostrar cómo la seguridad se integra de forma práctica y eficiente en entornos DevOps, asegurando que cada etapa del proceso de desarrollo aporte valor a la seguridad sin afectar la agilidad y automatización del despliegue.

Introducción a DevSecOps

DevSecOps es una metodología que incorpora la seguridad a lo largo del ciclo de vida del desarrollo de software, automatizando pruebas, análisis y despliegues para asegurar calidad y protección constante. Su propósito es garantizar que la seguridad no sea un paso aislado al final del proceso, sino un componente esencial desde la planificación, codificación, pruebas, despliegue y mantenimiento.

Arquitectura del Proyecto

La arquitectura del proyecto está diseñada para soportar un completo pipeline DevSecOps que integra herramientas de análisis de seguridad y despliegue automatizado, todo orquestado mediante contenedores Docker y gestionado a través de GitLab CI/CD.

El proyecto utiliza Docker Compose para levantar y conectar varios servicios clave:

- **GitLab CE:** Provee la plataforma de repositorios Git y la gestión integral del ciclo CI/CD. Se ejecuta en un contenedor con configuración persistente en volúmenes para mantener la configuración, logs y datos.

- **GitLab Runner:** Servicio que ejecuta los jobs del pipeline definidos en .gitlab-ci.yml. Está configurado para ejecutarse en modo Docker privilegiado, lo que permite construir imágenes y ejecutar contenedores dentro del pipeline.

- **SonarQube:** Plataforma de análisis estático de código (SAST) integrada con el pipeline para evaluar la calidad y seguridad del código fuente. Su interfaz web es accesible en el entorno local para revisión de resultados.
- **Trivy Server:** Herramienta de análisis de vulnerabilidades en dependencias y contenedores (SCA). Se ejecuta como servidor para acelerar y centralizar los escaneos durante el pipeline.

- **OWASP ZAP Proxy:** Herramienta para realizar análisis dinámicos de seguridad (DAST) instalable con su interfaz WebSwing para facilitar la automatización y generación de reportes HTML.

El directorio principal del proyecto contiene la definición del docker-compose.yml que orquesta estos servicios en una red Docker privada llamada devsecops-net, garantizando la comunicación segura y aislada entre los contenedores. Los servicios usan volúmenes Docker para persistir datos y configuraciones importantes, como la base de datos de SonarQube o la caché de Trivy.

La aplicación vulnerable de prueba está ubicada en la carpeta proyecto/final y cuenta con un Dockerfile para construir su imagen personalizada que será desplegada durante la etapa de despliegue en el pipeline. Este diseño modular permite desarrollar, analizar, testar y desplegar la aplicación en un flujo continuo sin perder la integración entre las diferentes etapas de seguridad.

Gracias a esta arquitectura basada en contenedores y la integración continua, se logra un entorno replicable y escalable, donde cada componente puede ser actualizado, revisado y gestionado de forma independiente, facilitando la automatización del ciclo DevSecOps y la colaboración ágil entre equipos.

Metodología DevSecOps

El pipeline está definido en .gitlab-ci.yml con las siguientes etapas:

- **Test:** Instalación de dependencias y pruebas básicas en Node.js.

```
# ============================
# 1. TEST
# ============================
test_app:
  stage: test
  image: node:18
  script:
    - cd final
    - npm install
    - npm test
  only:
    - main
```

- **Build:** Construcción de imagen Docker de la aplicación.

```
# ============================
# 2. BUILD
# ============================
build_image:
  stage: build
  image: docker:latest
  services:
    - docker:dind
  variables:
    DOCKER_TLS_CERTDIR: ""
  script:
    - cd final
    - docker build -t $DOCKER_IMAGE .
  only:
    - main
```

- **SAST:** Análisis estático con SonarQube, configurado con token CI.

```
# ============================
# 3. SAST — SonarQube
# ============================
sast_sonarqube:
  stage: sast
  image: sonarsource/sonar-scanner-cli:latest
  script:
    - rm -rf .scannerwork

    - |
      sonar-scanner \
        -Dsonar.projectKey=proyecto-final \
        -Dsonar.sources=final \
        -Dsonar.host.url=$SONAR_HOST_URL \
        -Dsonar.token=$SONAR_TOKEN
  only:
    - main
```

- **SCA:** Escaneo de dependencias y sistema de archivos con Trivy.

```
# ============================
# 4. SCA
# ============================
sca_trivy:
  stage: sca
  image: alpine:3.18
  before_script:
    - apk add --no-cache curl
    - curl -sfL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh | sh -s
-- -b /usr/local/bin
  script:
    - trivy fs --severity HIGH,CRITICAL --format json -o trivy-fs-report.json .
  artifacts:
    paths:
      - trivy-fs-report.json
  only:
    - main
```

- **DAST:** Análisis dinámico con OWASP ZAP proxy para escaneo en ejecución.

```
# ==========================
# 5. DAST – ZAP
# ==========================
dast_zap:
  stage: dast
  image: alpine:3.18
  script:
    - apk add --no-cache curl

    - echo "Iniciando análisis ZAP contra http://192.168.233.138:3001 ..."
    - echo "Conectando a ZAP API en http://192.168.233.138:8080"

    - curl "http://192.168.233.138:8080/JSON/spider/action/scan/?url=http://192.168.233.138:3001"

    - curl "http://192.168.233.138:8080/OTHER/core/other/htmlreport/" -o zap_report.html
  artifacts:
    paths:
      - zap_report.html
  only:
    - main
```

- **Deploy:** Despliegue del contenedor con la aplicación vulnerable para pruebas.

```
# ============================
# 7. DEPLOY
# ============================
deploy_app:
  stage: deploy
  image: docker:latest
  services:
    - docker:dind
  variables:
    DOCKER_TLS_CERTDIR: ""
  script:
    - docker rm -f node_final || true
    - docker run -d --name node_final -p 3001:3000 $DOCKER_IMAGE
  only:
    - main
```

Implementación

- Configuración del repositorio

El repositorio principal del proyecto está organizado para facilitar la integración continua y la automatización de análisis y despliegue. Se utiliza GitLab como sistema de control de versiones y plataforma para la ejecución de pipelines CI/CD, usando principalmente la rama main, donde se centralizan los commits que activan el flujo de despliegue y las pruebas en cada push.

- Estructura del proyecto

La estructura del proyecto está diseñada para mantener separados los componentes de configuración, código fuente y scripts de despliegue, facilitando la gestión y mantenimiento:

- **docker-compose.yml:** Define los servicios Docker principales para levantar GitLab CE, GitLab Runner, SonarQube y Trivy, además de configurar redes y volúmenes persistentes.

```
juan@juan:~$ cat docker-compose.yml
version: '3.8'

services:
  gitlab-ce:
    image: gitlab/gitlab-ce:latest
    container_name: gitlab-ce
    restart: always
    hostname: '192.168.233.138'
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        external_url 'http://192.168.233.138'
    ports:
      - '80:80'
      - '443:443'
      - '2222:22'
    volumes:
      - ./gitlab/config:/etc/gitlab
      - ./gitlab/logs:/var/log/gitlab
      - ./gitlab/data:/var/opt/gitlab
    networks:
      - devsecops-net

  gitlab-runner:
    image: gitlab/gitlab-runner:latest
    container_name: gitlab-runner
    restart: always
    depends_on:
      - gitlab-ce
    volumes:
      - ./gitlab-runner/config:/etc/gitlab-runner
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - devsecops-net
```

- **docker-compose.zap.yml:** Configura específicamente el servicio para ejecutar OWASP ZAP como un proxy para pruebas dinámicas de seguridad.

- **gitlab-runner/config/config.toml:** Archivo de configuración del GitLab Runner donde se establecen parámetros del ejecutor Docker, tokens y red.

- **proyecto/final/:** Carpeta que contiene el código fuente de la aplicación Node.js vulnerable junto a su Dockerfile, package.json y archivos JavaScript. Esta carpeta es el objetivo de los análisis estáticos y dinámicos.

```
juan@juan:~/proyecto$ ls -la
total 36
drwxrwxr-x 4 juan juan 4096 nov 24 23:35 .
drwxr-x--- 8 juan juan 4096 nov 23 14:13 ..
drwxr-xr-x 2 root root 4096 nov 23 03:50 final
drwxrwxr-x 8 juan juan 4096 nov 24 23:36 .git
-rw-r--r-- 1 root root 2441 nov 24 23:35 .gitlab-ci.yml
-rw-r--r-- 1 root root 2908 nov 22 23:27 .gitlab-ci.yml.save
-rw-rw-r-- 1 juan juan 6118 nov 20 04:58 README.md
-rw-r--r-- 1 root root  187 nov 23 03:49 sonar.project.properties
```

- **.gitlab-ci.yml:** Archivo que define el pipeline CI/CD con etapas, variables y scripts para pruebas, construcción, análisis con SonarQube, Trivy, ZAP y despliegue automático.

```
juan@juan:~/proyecto$ cat .gitlab-ci.yml
stages:
  - test
  - build
  - sast
  - sca
  - dast
  - deploy

variables:
  DOCKER_IMAGE: gitlab-ci-cd-lab:latest
  DOCKER_DRIVER: overlay2

  SONAR_HOST_URL: "http://192.168.233.138:9000"

  TRIVY_SERVER_URL: "http://192.168.233.138:4954"

# ============================
# 1. TEST
# ============================
test_app:
  stage: test
  image: node:18
  script:
    - cd final
    - npm install
    - npm test
  only:
    - main

# ============================
# 2. BUILD
# ============================
build_image:
  stage: build
  image: docker:latest
  services:
```

Dentro del directorio proyecto/ también se encuentran otros archivos relevantes como el README con documentación principal y el archivo sonar.project.properties que configura SonarQube para el análisis del código fuente.

Este diseño modular y organizado permite que cada componente sea fácilmente identificable, modificable y testeable, garantizando que los cambios en el código se reflejen inmediatamente en la ejecución del pipeline y en la seguridad del proyecto.

Herramientas utilizadas

- **GitLab CE y GitLab Runner:** Es la plataforma principal para el control de versiones y la gestión del ciclo de vida del desarrollo, facilitando la integración continua y entrega continua (CI/CD). Permite alojar repositorios Git y definir pipelines automatizados. GitLab Runner es el ejecutor de jobs que procesa las tareas definidas en los pipelines, aquí configurado en modo Docker para permitir la construcción y pruebas de contenedores de forma aislada y replicable.

- **SonarQube:** Es una herramienta de análisis estático de código (SAST) que examina el código fuente en busca de vulnerabilidades, errores y malas prácticas de programación. Se integra en el pipeline para ofrecer reportes detallados sobre la calidad y seguridad del código, ayudando a identificar y corregir problemas antes del despliegue. SonarQube también soporta reglas personalizables y se conecta con el flujo devsecops mediante tokens y configuración vía CI.

- **Trivy:** Es un escáner de vulnerabilidades para contenedores y dependencias, utilizado aquí para análisis de composición de software (SCA). Detecta vulnerabilidades conocidas en librerías y paquetes usados en la aplicación, así como en la imagen Docker generada. Su integración como servicio y uso en pipelines permite automatizar la detección proactiva de riesgos en el entorno de ejecución.

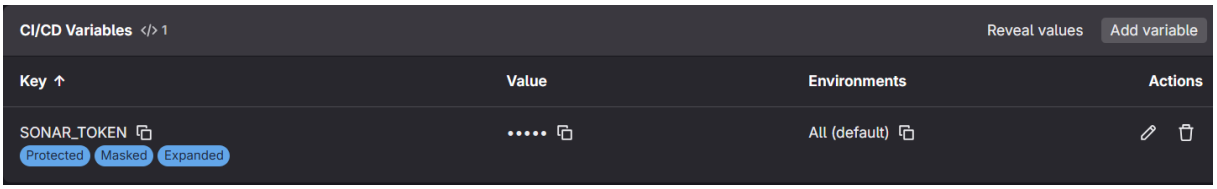
- **OWASP ZAP (Zed Attack Proxy):** Es una herramienta para pruebas de seguridad dinámicas (DAST) que realiza análisis de aplicaciones en ejecución buscando vulnerabilidades explotables, como inyecciones, XSS, y otros ataques comunes. Aquí se configura como un proxy con interfaz WebSwing para su automatización dentro del pipeline, generando reportes que evidencian la seguridad real del software desplegado.

- **Docker y Docker Compose:** Se utiliza para contenerizar cada uno de los servicios y la aplicación bajo prueba, asegurando portabilidad y aislamiento. Docker Compose facilita la orquestación de múltiples contenedores relacionados, gestionando redes, volúmenes y dependencias entre ellos. Esto permite montar un entorno completo de CI/CD local replicable para desarrollo y pruebas integradas.

Configuración específica

En la configuración del pipeline DevSecOps es fundamental definir variables de entorno que permitan la integración y comunicación segura entre los distintos servicios desplegados en contenedores. Por ejemplo:

SONAR_TOKEN: Token de autenticación generado en SonarQube para permitir que el pipeline pueda enviar análisis y recibir reportes de manera segura.



TRIVY_SERVER_URL: URL del servicio Trivy Server donde se ejecutan los escaneos de vulnerabilidades en dependencias y contenedores, accesible desde el pipeline.

Estas variables se definen en el archivo `.gitlab-ci.yml` para que los jobs dispongan de ellas al momento de ejecutar las herramientas de análisis.

El archivo `docker-compose.yml` establece la orquestación de los servicios con varios detalles importantes:

- Volúmenes: Permiten persistir datos y configuraciones esenciales, como la base de datos y configuración de SonarQube (`sonarqube_data`, `sonarqube_extensions`), o la caché de Trivy (`trivy-cache`). También se montan directorios del host a los contenedores en GitLab y GitLab Runner para compartir configuraciones y el socket de Docker que permite la ejecución de contenedores anidados.

```
volumes:
  sonarqube_data:
  sonarqube_extensions:
  trivy-cache:
```

- **Redes**: Se crea una red personalizada llamada `devsecops-net` que aísla la comunicación entre los contenedores, asegurando que sólo los servicios autorizados puedan comunicarse entre sí en el entorno cerrado del pipeline.

```
networks:
  - devsecops-net
```

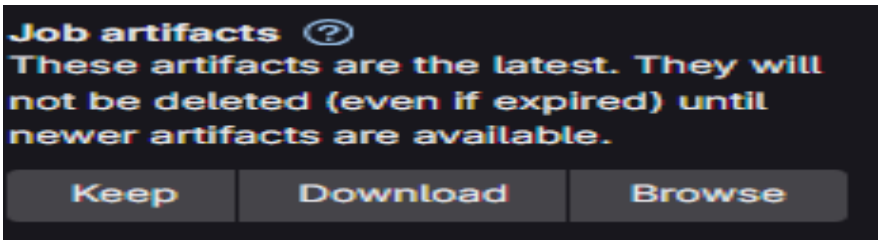
Esta configuración modular y aislada facilita la escalabilidad y replicabilidad del entorno DevSecOps, asegurando que cada herramienta puede interactuar con las demás sin conflictos y con persistencia de datos adecuada.

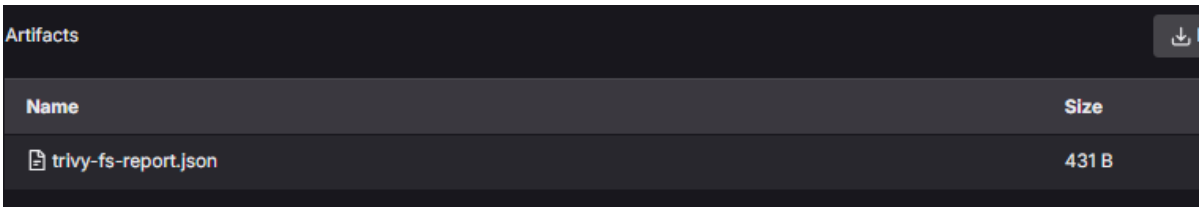
Resultados

Los reportes generados durante el pipeline ofrecen una visión completa del estado de seguridad y calidad del código y la infraestructura de la aplicación:

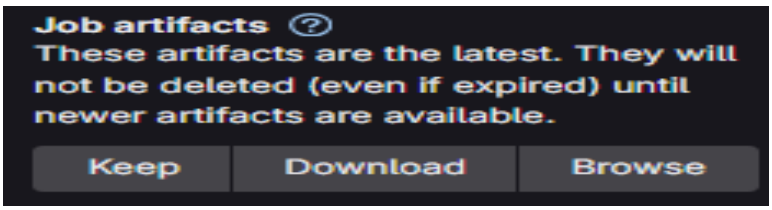
- Los reportes de SonarQube están disponibles en su interfaz web, mostrando resultados de análisis estático de código con métricas detalladas como vulnerabilidades, código duplicado, cobertura, y errores potenciales. Estos informes facilitan la identificación rápida y la gestión de riesgos en el código fuente, además de apoyar la mejora continua del desarrollo.

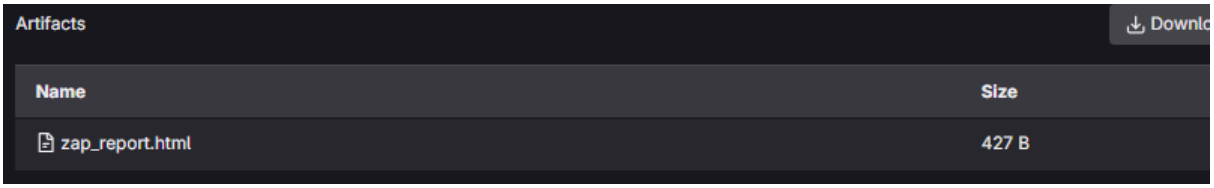
- Los reportes JSON de Trivy son generados durante el escaneo del sistema de archivos y dependencias, y se almacenan como artefactos del pipeline para su revisión posterior. Estos reportes permiten detectar vulnerabilidades conocidas en librerías y componentes utilizados por la aplicación y contenedores.



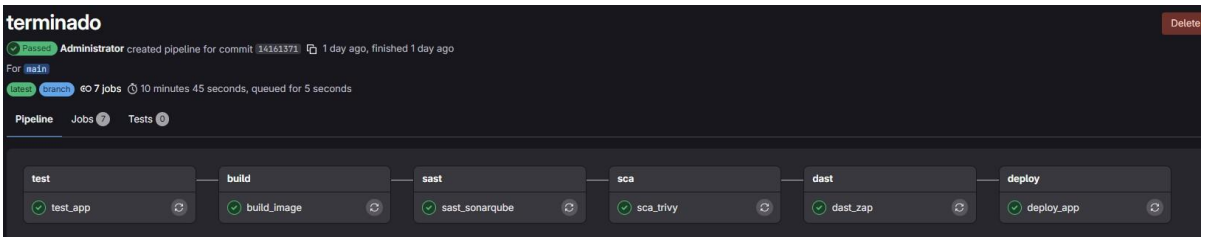


- Los reportes HTML de OWASP ZAP se generan tras realizar el escaneo dinámico de seguridad sobre la aplicación desplegada, identificando posibles fallos explotables en tiempo de ejecución, como inyección de código o ataques XSS. Estos informes se guardan también como artefactos accesibles para análisis y auditoría.





- La ejecución del pipeline es exitosa, automatizando cada etapa —desde la prueba inicial del código, pasando por su construcción, análisis estático y dinámico, hasta el despliegue final— garantizando la integración continua y el aseguramiento de la calidad y seguridad en todo momento..



Conclusiones

El proyecto demuestra la viabilidad y beneficios de integrar seguridad automatizada en pipelines DevOps usando herramientas estándar open source. Se obtuvo un proceso eficiente para detectar vulnerabilidades desde el desarrollo hasta el despliegue, lo que refuerza la importancia de aplicar metodologías DevSecOps en entornos académicos y profesionales. Este enfoque permite que la seguridad sea parte integral del ciclo de vida del software, evitando que se convierta en una tarea aislada o tardía.

Además, se recomienda seguir mejorando con análisis y validaciones más profundas, así como ampliar la cobertura de pruebas para fortalecer aún más la detección de riesgos. La experiencia obtenida evidencia que la automatización y el uso de herramientas accesibles no solo optimizan el trabajo de los equipos, sino que también promueven una cultura de desarrollo seguro y responsable, capaz de adaptarse a las necesidades reales de la industria tecnológica.