**Informe Técnico**

**Diagnóstico y Propuesta de Mejora en una Pipeline CI/CD con Falencias de Seguridad – CodeSecure Ltd.**

**1. Escenario**

La empresa **CodeSecure Ltd.** implementó un pipeline básico de integración continua (**CI/CD**) para su aplicación web. Actualmente, este pipeline:

* Compila el código y lo prueba con Jest.
* Despliega automáticamente a producción.
* Carece de validación de seguridad, control de dependencias y análisis de infraestructura.
* No tiene control de roles ni aprobación manual antes del despliegue.

El riesgo principal es la introducción de código vulnerable en producción sin controles previos.

**2. Análisis de Debilidades**

| **Nº** | **Debilidad Detectada** | **Herramienta / Técnica Propuesta** | **Justificación** |
| --- | --- | --- | --- |
| 1 | Ausencia de análisis de código fuente (SAST) | **SonarQube, Semgrep** | Permiten detectar inyecciones, hardcode de credenciales y malas prácticas antes del build. |
| 2 | Falta de control de dependencias y librerías vulnerables (SCA) | **Snyk, OWASP Dependency-Check** | Detectan CVEs conocidos en librerías externas y bloquean su uso en producción. |
| 3 | No se realiza escaneo dinámico (DAST) a la aplicación desplegada | **OWASP ZAP, Burp Suite CLI** | Simulan ataques reales a la aplicación para descubrir XSS, SQLi y otros vectores. |
| 4 | No existe análisis de infraestructura, imágenes y configuración | **Trivy, Checkov, Docker Scout** | Identifican vulnerabilidades en contenedores, malas configuraciones y errores en IaC. |
| 5 | Despliegue automático sin aprobación ni segregación de funciones | **Gate de aprobación manual + RBAC en CI/CD** | Reduce el riesgo de que cambios maliciosos o inseguros lleguen a producción sin revisión humana. |

**3. Propuesta de Pipeline CI/CD Seguro**

**Flujo recomendado bajo enfoque DevSecOps:**

1. **Commit & Push** → Revisión de código con **pre-commit hooks**.
2. **Build** → Ejecución de pruebas unitarias + **SAST** (SonarQube).
3. **Análisis de Dependencias (SCA)** → Snyk u OWASP Dependency-Check.
4. **Escaneo de Contenedores e Infraestructura** → Trivy + Checkov.
5. **Despliegue en entorno de staging**.
6. **DAST automatizado** → OWASP ZAP CLI sobre staging.
7. **Gate de aprobación manual** (roles definidos en GitLab CI/Jenkins).
8. **Despliegue a producción** solo si todas las fases anteriores son exitosas.

**4. Beneficios Estratégicos del Pipeline Seguro**

1. **Reducción de Riesgos Operativos:** disminuye la probabilidad de que vulnerabilidades críticas lleguen a producción.
2. **Cumplimiento Normativo:** facilita la alineación con marcos de seguridad como **OWASP SAMM, NIST 800-53 y ISO 27001**.
3. **Mayor confianza del cliente y stakeholders:** demuestra un enfoque proactivo hacia la seguridad.
4. **Automatización eficiente:** integra seguridad sin frenar la agilidad del desarrollo, reforzando la cultura DevSecOps.

**5. Conclusión**

El pipeline actual de **CodeSecure Ltd.** presenta debilidades críticas que pueden comprometer la seguridad de la aplicación en producción. La propuesta de integración de **SAST, SCA, DAST, análisis de infraestructura y controles de aprobación manual** fortalece la cadena CI/CD, alineándola con principios de **DevSecOps**. Implementar estas medidas garantiza mayor resiliencia frente a ataques, mejora continua y reducción del riesgo organizacional.