Maestría en

CIBERSEGURIDAD

Trabajo previo a la obtención de título de Magister en Ciberseguridad

AUTOR/ES:

CALDERON QUIROLA FERNANDO JOSE

HERRERA VERA MARIA DE LOS ANGELES

LOAYZA PALATE ANGHELO ANDRES

SALTOS SAA BRYAN AGUSTIN

TUTORES:

Iván Reyes Chacón

Juan Fernando López

TEMA

Diseño y prototipado automatizado para la detección de vulnerabilidades en imágenes Docker provenientes de repositorios públicos y privados.

Quito - Ecuador

Junio – 20206

REINVENTEMOS EL FUTURO

Certificación de autoría

Nosotros, **Calderon Quirola Fernando Jose, Herrera Vera Maria de los Angeles, Saltos Saá Bryan Agustin, Loayza Palate Anghelo Andres,** declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido presentado anteriormente para ningún grado o calificación profesional y que se ha consultado la bibliografía detallada. Cedemos nuestros derechos de propiedad intelectual a la Universidad Internacional del Ecuador (UIDE), para que sea publicado y divulgado en internet, según lo establecido en la Ley de Propiedad Intelectual, su reglamento y demás disposiciones legales.

Firma
Calderon Quirola Fernando Jose

Firma
Herrera Vera Maria de los Angeles

Firma
Saltos Saá Bryan Agustin

Firma
Loayza Palate Anghelo Andres

Autorización de Derechos de Propiedad Intelectual

Nosotros, **Calderon Quirola Fernando Jose, Herrera Vera Maria de los Angeles, Saltos Saá Bryan Agustin, Loayza Palate Anghelo Andres**, en calidad de autores del trabajo de investigación titulado ***Diseño y prototipado automatizado para la detección de vulnerabilidades en imágenes Docker provenientes de repositorios públicos y privados***, autorizamos a la Universidad Internacional del Ecuador (UIDE) para hacer uso de todos los contenidos que nos pertenecen o de parte de los que contiene esta obra, con fines estrictamente académicos o de investigación. Los derechos que como autores nos corresponden, lo establecido en los artículos 5, 6, 8, 19 y demás pertinentes de la Ley de Propiedad Intelectual y su Reglamento en Ecuador.

D. M. Quito, (junio 2026)

Firma
Calderon Quirola Fernando Jose

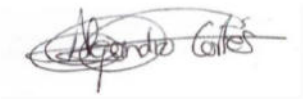
Firma
Herrera Vera Maria de los Angeles

Firma
Saltos Saá Bryan Agustin

Firma
Loayza Palate Anghelo Andres

APROBACIÓN DE DIRECCIÓN Y COORDINACIÓN DEL PROGRAMA

Nosotros, Alejandro Cortés López e Iván Reyes Chacón, declaramos que: Calderon Quirola Fernando Jose, Herrera Vera Maria de los Angeles, Saltos Saá Bryan Agustin, Loayza Palate Anghelo Andres son los autores exclusivos de la presente investigación y que ésta es original, auténtica y personal de ellos.



Alejandro Cortés
Director/a de la
Maestría en Ciberseguridad

Iván Reyes
Coordinador/a de la
Maestría en Ciberseguridad

DEDICATORIA

A ti, mi Moni, por estar siempre a mi lado. Gracias por tu amor, tu apoyo incondicional y la fortaleza que me brindas cada día. Ruego a la vida que nos permita seguir compartiendo este camino y que siempre pueda contar con tu compañía, tu cariño y tu confianza para enfrentar cada nuevo desafío, Te amo.

A mi mami, Magdalena, por ser el ejemplo de esfuerzo, amor y perseverancia que ha guiado mi vida. Gracias por creer siempre en mí y por enseñarme que, con dedicación y constancia, es posible alcanzar cualquier meta.

A mi hermano, Steve, por su apoyo, compañía y por estar presente a lo largo de este camino.

A mi suegra, Mercedes, por su cariño y apoyo, por hacerme sentir parte de su familia desde el primer momento.

A mis cuñados, Stefania y Jonathan, por su ánimo, amistad y por acompañarme durante esta importante etapa de mi vida.

A mi sobrinita Fabiana, cuya alegría, ternura e inocencia iluminan mis días y me recuerdan que todo esfuerzo vale la pena cuando se construye un mejor futuro.

A toda mi familia, por ser mi refugio en los momentos difíciles y por acompañarme con paciencia, comprensión y palabras de aliento. Su apoyo constante fue el impulso que necesitaba para superar cada obstáculo y culminar esta etapa tan exigente como profundamente gratificante de mi vida.

Finalmente, dedico este trabajo a todas las personas que, de una u otra manera, contribuyeron al desarrollo de este proyecto. Sus enseñanzas, consejos, apoyo y confianza dejaron una huella invaluable en este logro, que también les pertenece.

Con profundo cariño y gratitud,

Bryan Agustín Saltos Saá

A mi esposa Polet, quien ha sido un pilar de apoyo constante y mi mayor motivación para continuar durante todo el proceso de formación. Gracias por creer siempre en mí, por tu amor incondicional, por motivarme a seguir adelante y por no permitirme rendirme, incluso en los momentos más difíciles. Tu confianza fueron esenciales para alcanzar esta meta.

A la memoria de mi padre José, aunque ya no te encuentres físicamente conmigo, fuiste tú quien me motivó a ser un profesional de buenos valores. Tu recuerdo es la fuerza que me impulsó a continuar con mis estudios y a dar lo mejor de mí para llegar hasta aquí. Esto es por ti.

A mi mamá, por su apoyo incondicional, por ser un ejemplo de valores y por esos consejos que me convencieron de que la constancia y la dedicación siempre traen grandes resultados.

A mis hermanos, por estar siempre dispuestos a apoyarme y por creer en mí en todo momento.

A todos quienes fueron parte de este proceso, amigos, familiares, compañeros de grupo, docentes y tutor de la maestría, quienes, con su acompañamiento, conocimientos y palabras de aliento, hicieron posible que alcanzara este logro.

Anghelo Andres Loayza Palate

A mi familia. A mis padres, que lo dieron todo sin pedir nada. A los que preguntaron cómo iba la tesis aunque no entendieran de qué trataba, a los que me guardaron un plato de comida en las reuniones que me perdí, a los que creyeron sin necesidad de pruebas. Padres, hermanos, abuelos, tíos, primos: esto es de ustedes.

Fernando José Calderón Quirola

Dedico este trabajo, en primer lugar, a Dios, por darme la fortaleza, la sabiduría y la perseverancia necesarias para superar cada desafío y permitirme alcanzar una de las metas más importantes de mi vida.

A mi familia, por ser mi mayor apoyo y el motor que me impulsó a seguir adelante. A mi querida mamá, Rosy, por su amor incondicional, sus consejos, sus sacrificios y por creer siempre en mí. A mi abuelita, Mami Tere, por su cariño y por ser un ejemplo de fortaleza. A mis tíos y a mis primos Adri, Santy, Pao, Micha, Diego, Jessi, Jorge y Anita, gracias por acompañarme en este camino, por confiar en mí y por hacerme sentir que nunca estuve sola. Gracias por estar presentes en cada paso y por celebrar conmigo cada pequeño logro.

A mis sobrinos de corazón, Benja, Aitana y Sarahi, quienes con sus ocurrencias, alegría y sonrisas hicieron que los días más difíciles fueran mucho más llevaderos. Ustedes me recordaron que siempre hay motivos para sonreír.

A mi esposo, por ser mi compañero en este camino, por su apoyo, paciencia y comprensión en cada etapa de este proceso. Gracias por creer en mí y por acompañarme siempre con amor. A mi suegra y a mis cuñados, por su cariño, por estar siempre pendientes de mí y por brindarme palabras de aliento que hicieron este camino más llevadero. Gracias por hacerme sentir parte de su familia.

Finalmente, me dedico este logro a mí misma. Por no rendirme en las noches de cansancio, por afrontar cada reto con determinación y por encontrar siempre el tiempo para seguir creciendo personal y profesionalmente. Hoy miro hacia atrás con orgullo y satisfacción, porque cada sacrificio valió la pena y este logro es el reflejo de mi constancia, disciplina y amor por superarme.

María de los Ángeles Herrera Vera

AGRADECIMIENTOS

Expreso mi más profundo agradecimiento a mi prometida, a mis padres, a mi hermano, a mis cuñados y a mi sobrina, quienes con su amor, apoyo incondicional, motivación y constantes palabras de aliento fueron el pilar que me impulsó a seguir adelante en cada etapa de este extraordinario camino. Su confianza en mí fue una fuente permanente de fortaleza para alcanzar esta meta.

A mis compañeros de grupo, por su compromiso, organización, dedicación y esfuerzo durante el desarrollo de este trabajo. La colaboración, el respeto y el trabajo en equipo fueron fundamentales para culminar con éxito este proyecto.

A mis docentes y, de manera especial, a mi tutor de maestría, por compartir generosamente sus conocimientos, experiencias y orientación. Su dedicación y exigencia académica contribuyeron significativamente a mi formación profesional y me prepararon para afrontar los retos del competitivo y desafiante mundo de la ciberseguridad.

Finalmente, expreso mi sincero agradecimiento a la Universidad Internacional del Ecuador (UIDE) por brindarme la oportunidad de formar parte de su comunidad académica. Gracias a cada uno de los profesionales que me acompañaron durante esta etapa de mi vida, por su apoyo, guía y compromiso con la excelencia educativa, los cuales hicieron posible la culminación de este importante logro académico.

Bryan Agustin Saltos Saá

Agradezco profundamente a Dios, por la fortaleza, la sabiduría y la paciencia necesarias para conseguir este logro académico y personal.

A mi esposa, Polet, por su amor, comprensión y motivación constante durante todo este proceso. Su apoyo incondicional y su confianza en mí fueron fundamentales para superar cada desafío y alcanzar este objetivo.

A mi madre, a la memoria de mi padre y a mis hermanos, por su amor, sus consejos y el apoyo que siempre me han brindado. Gracias por creer en mí e impulsarme a seguir adelante con confianza y perseverancia.

A mis docentes y tutor de la maestría, por compartir sus conocimientos, tiempo y dedicación, siendo guía en cada etapa de la carrera y brindando enseñanzas valiosas aplicadas a escenarios reales.

A la Universidad UIDE, por abrirme sus puertas, brindarme una formación académica de excelencia y su acompañamiento desde el primer contacto hasta el final de la carrera. Los docentes presentes en cada etapa demuestran el nivel de calidad y compromiso que tiene la institución con sus estudiantes.

Anghelo Andres Loayza Palate

Agradezco profundamente a mi familia, que ha sido el soporte constante detrás de este proceso. A mis padres, por su esfuerzo incondicional y por enseñarme que la disciplina y la humildad abren más puertas que cualquier título. A mis hermanos, por su paciencia en los días en que el estudio me robaba el tiempo y el humor. A mis abuelos, tíos y primos, por su cariño genuino y por recordarme, en cada encuentro, lo que realmente importa. Este trabajo no habría sido posible sin ustedes: gracias por acompañarme en cada etapa, por entender mis ausencias y por celebrar mis avances como si fueran propios.

Fernando José Calderón Quirola

Expreso mi más sincero agradecimiento a Dios por darme la fortaleza, la sabiduría y la perseverancia necesarias para culminar esta importante etapa de mi vida.

A la Universidad Internacional del Ecuador, sus docentes y a nuestro director de tesis, por compartir sus conocimientos, orientación y acompañamiento durante mi formación académica y el desarrollo de este proyecto.

A mi trabajo, por brindarme la oportunidad de continuar creciendo profesionalmente y permitirme equilibrar mis responsabilidades laborales con este reto académico.

A mi familia, a mi esposo, a su familia y a todas las personas que, de una u otra manera, me acompañaron, motivaron y apoyaron durante este proceso. Su confianza, comprensión y palabras de aliento fueron fundamentales para alcanzar esta meta.

Finalmente, agradezco a todas aquellas personas que contribuyeron, directa o indirectamente, al desarrollo de esta investigación. Este logro representa el esfuerzo, la constancia y el apoyo de quienes estuvieron presentes a lo largo de este camino.

A todos, mi más sincero agradecimiento.

María de los Ángeles Herrera Vera

RESUMEN

La adopción masiva de arquitecturas basadas en contenedores ha incrementado la exposición de los entornos productivos a imágenes Docker con dependencias vulnerables, librerías obsoletas y configuraciones inseguras. Esta investigación tuvo como objetivo diseñar y validar un prototipo automatizado, denominado DockerSec Gate, para la detección preventiva de vulnerabilidades CVE en imágenes Docker provenientes de repositorios públicos (Docker Hub) y privados simulados, antes de su despliegue. El prototipo integra las herramientas de escaneo de código abierto Trivy y Grype dentro de un pipeline organizado por capas (adquisición, escaneo dual, normalización, correlación mediante el índice de Jaccard, validación externa multifuente, evaluación de confianza, security gate y generación de reportes), clasificando los hallazgos por severidad según el estándar CVSS v3.1 y aplicando bloqueos automáticos de despliegue ante vulnerabilidades críticas. Bajo un enfoque experimental con un corpus controlado y ground truth verificable, el sistema alcanzó un Recall y una Cobertura de 1,000, superó el umbral mínimo en las cuatro métricas evaluables y el umbral objetivo en el 75 % de ellas. El análisis de complementariedad confirmó que ambos motores aportan hallazgos diferenciados, validando empíricamente la estrategia de doble escaneo. Se concluye que la automatización del análisis estático de composición de software (del inglés, Software Composition Analysis, SCA) constituye una alternativa viable para fortalecer la seguridad de la cadena de suministro de software y favorecer la adopción de prácticas DevSecOps desde etapas tempranas del ciclo de desarrollo.

Palabras clave: seguridad de contenedores, imágenes Docker, análisis de vulnerabilidades, Trivy, Grype, CI/CD, DevSecOps, CVSS, índice de Jaccard, cadena de suministro de software.

ABSTRACT

The widespread adoption of container-based architectures has increased the exposure of production environments to Docker images containing vulnerable dependencies, outdated libraries, and insecure configurations. This research aimed to design and validate an automated prototype, named DockerSec Gate, for the preventive detection of CVE vulnerabilities in Docker images sourced from public (Docker Hub) and simulated private repositories, prior to deployment. The prototype integrates the open-source scanning tools Trivy and Grype within a layered pipeline (acquisition, dual scanning, normalization, correlation through the Jaccard index, multi-source external validation, confidence assessment, security gate, and report generation) classifying findings by severity according to the CVSS v3.1 standard and applying automatic deployment blocks for critical vulnerabilities. Under an experimental approach with a controlled corpus and verifiable ground truth, the system achieved a Recall and Coverage of 1.000, exceeded the minimum threshold on all four measurable metrics, and the target threshold on 75% of them. The complementarity analysis confirmed that both engines contribute differentiated findings, empirically validating the dual-scanning strategy. It is concluded that automating static software composition analysis (SCA) is a viable alternative to strengthen software supply chain security and to encourage the adoption of DevSecOps practices from the early stages of the development lifecycle.

Keywords: container security, Docker images, vulnerability scanning, Trivy, Grype, CI/CD, DevSecOps, CVSS, Jaccard index, software supply chain.

TABLA DE CONTENIDO

Acuerdo de confidencialidad 4
RESUMEN 14
ABSTRACT 15
LISTA DE FIGURAS 18
LISTA DE TABLAS 19
Capítulo 1 20
1. INTRODUCCIÓN 20
1.1. Definición del proyecto 20
1.2. Justificación e Importancia del Trabajo de Investigación 22
1.3. Alcance 24
1.3.1. Elementos incluidos en el alcance: 24
1.3.2. Elementos excluidos del alcance: 25
1.4. Objetivos 25
1.4.1. Objetivo General 25
1.4.2. Objetivos específicos 26
Capítulo 2: 27
2. REVISIÓN DE LITERATURA 27
2.1. Estado del Arte 27
2.1.1. Estudios sobre Herramientas de Escaneo de Contenedores 27
2.1.2. Incidentes Documentados en la Cadena de Suministro de Contenedores 29
2.1.3. Panorama de Vulnerabilidades en Contenedores (2020–2025) 29
2.2. Marco Teórico 30
2.2.1. Fundamentos de la Tecnología de Contenedores y el Ecosistema Docker 30
2.2.2. Gestión de Vulnerabilidades y Métricas de Seguridad 32
2.2.3. Arquitectura de Sistemas de Análisis Automatizado 35
2.2.4. Análisis Comparativo de Herramientas de Escaneo 37
2.2.5. Marco de Evaluación y Métricas de Rendimiento del Prototipo 38
2.2.6. Criterios de Aceptación del Prototipo 40
Capítulo 3: 42
3. DESARROLLO 42
3.1. Enfoque y Diseño de la Investigación 42
3.2. Arquitectura General del Prototipo 43
3.3. Implementación de los Componentes del Sistema 44
3.3.1. Capa de Adquisición 44
3.3.2. Capa de Escaneo Dual 45
3.3.3. Capa de Normalización 45
3.3.4. Capa de Correlación e Índice de Jaccard 46

3.3.5. Capa de Validación Externa Multifuente 47
3.3.6. Capa de Evaluación de Confianza 47
3.3.7. Capa de Decisión: Security Gate y Políticas por Ambiente 48
3.3.8. Capa de Reportes y Persistencia 48
3.4. Entorno de Implementación y Reproducibilidad 49
3.5. Diseño Experimental 51
3.5.1. Corpus de Imágenes de Prueba 51
3.5.2. Ground Truth y Verificación por Construcción 52
3.5.3. Variables, Métricas e Instrumentos de Medición 53
3.5.4. Procedimiento Experimental 53
3.5.5. Consideraciones de Validez y Limitaciones Metodológicas 54
3.6. Síntesis del Capítulo 54
Capítulo 4 56
4. ANÁLISIS DE RESULTADOS 56
4.1. Pruebas de concepto 56
4.1.1. Caracterización del corpus y carga de vulnerabilidades 56
4.2. Análisis de Resultados 57
4.2.1. Consistencia Interescaner 57
4.2.2. Eficacia de la detección: Matriz de Confusión 58
4.2.3. Precisión por Existencia (proxy) 59
4.2.4. Evaluación Frente a los Criterios de Aceptación 60
4.2.5. Validación Complementaria Sobre Repositorios Privados 61
4.2.6. Rendimiento Temporal 66
4.2.7. Discusión 66
4.2.8. Síntesis del Capítulo 69
Capítulo 5 70
5. CONCLUSIONES Y RECOMENDACIONES 70
5.1. Conclusiones 70
5.2. Recomendaciones 72
6. Referencias 73
ANEXO A 75
Tabla A1 75
ANEXO B 75

LISTA DE FIGURAS

FIGURA 1 ARQUITECTURA DEL PIPELINE AUTOMATIZADO DE DETECCIÓN DE VULNERABILIDADES DEL PROTOTIPO DOCKERSEC GATE . 43
FIGURA 2 DISTRIBUCIÓN POR SEVERIDAD Y ORIGEN DE DETECCIÓN RICA-API:PROD-V1.0.0 62
FIGURA 3 DIAGRAMA DE SECUENCIA DE LA INTERACCIÓN CLIENTE–API–MOTORES DE ESCANEO–MONGODB PARA SALERT-NOTIFIER:PROD-V1.3.0 63
FIGURA 4 DISTRIBUCIÓN POR SEVERIDAD Y ORIGEN DE DETECCIÓN SOPHON-OPS:PROD-V0.1.0 64

LISTA DE TABLAS

TABLA 1 MÉTRICAS DE RIESGO EN ENTORNOS DE CONTENEDORES 30
TABLA 2 NIVELES DE SEVERIDAD CVSS V3.1 Y ESTRATEGIA DE RESPUESTA DEL PROTOTIPO 34
TABLA 3 CAPAS DEL PIPELINE DE SEGURIDAD AUTOMATIZADO IMPLEMENTADAS EN EL PROTOTIPO 35
TABLA 4 COMPARATIVA TÉCNICA DE HERRAMIENTAS DE ESCANEO DE IMÁGENES DOCKER 37
TABLA 5 CRITERIOS DE ACEPTACIÓN OBLIGATORIOS (EVALUABLES EN ESTE DISEÑO EXPERIMENTAL, CON GROUND TRUTH DE CVE PUNTUADOS POR CONSTRUCCIÓN) 40
TABLA 6 MÉTRICAS DE REFERENCIA (NO EVALUABLES EN ESTE DISEÑO EXPERIMENTAL POR AUSENCIA DE UN ORÁCULO COMPLETO DE COMPONENTES; SE REPORTAN COMO META DESEABLE PARA UN TRABAJO FUTURO, NO COMO CRITERIO DE ACEPTACIÓN) 41
TABLA 7 STACK TECNOLÓGICO DEL PROTOTIPO DOCKERSEC GATE 49
TABLA 8 COMPOSICIÓN DEL CORPUS EXPERIMENTAL POR CATEGORÍA DE RIESGO 51
TABLA 9 CLASIFICACIÓN DEL GROUND TRUTH SEGÚN EL ALCANCE DEL ANÁLISIS DE COMPOSICIÓN DE SOFTWARE 52
TABLA 10 CARACTERIZACIÓN DEL CORPUS POR CATEGORÍA DE RIESGO (51 IMÁGENES) 56
TABLA 11 ÍNDICE DE JACCARD INTERESCÁNER (TRIVY VS. GRYPE), GLOBAL Y ACCIONABLE 57
TABLA 12 MATRIZ DE CONFUSIÓN SOBRE LOS CVE PLANTADOS IN-SCOPE 58
TABLA 13 PROXY DE PRECISIÓN POR EXISTENCIA (CONFIRMACIÓN EN NVD/CVE.ORG) 59
TABLA 14 EVALUACIÓN DE LAS MÉTRICAS FRENTE A LOS CRITERIOS DE ACEPTACIÓN 60
TABLA 15 RESULTADOS DEL ESCANEO SOBRE IMÁGENES DE REPOSITORIOS PRIVADOS (AMBIENTE PRODUCCIÓN) 61
TABLA 16 ESTADÍSTICAS DE CORRELACIÓN INTERESCÁNER IMÁGENES DE REPOSITORIOS PRIVADOS 62

Capítulo 1

1. INTRODUCCIÓN

La adopción de arquitecturas basadas en contenedores ha transformado significativamente el desarrollo y despliegue de aplicaciones modernas, encontrando en Docker su expresión más difundida. (Merkel, 2014) lo define como un mecanismo de empaquetado ligero que permite ejecutar aplicaciones de forma aislada compartiendo el kernel del sistema operativo anfitrión, garantizando con ello portabilidad, consistencia y escalabilidad entre distintos entornos de ejecución (Bhardwaj, 2024).

Este crecimiento ha sido exponencial: en 2025 se estima que más del 70 % de las organizaciones de desarrollo de software utilizan contenedores en alguna fase de su ciclo de vida productivo, lo que convierte a la seguridad de estas tecnologías en una prioridad estratégica (Khalil, 2025).

Sin embargo, el uso masivo de imágenes Docker provenientes de repositorios públicos y privados ha introducido nuevos desafíos de seguridad (Kaur et al., 2021). Muchas de estas imágenes contienen dependencias vulnerables, librerías obsoletas y configuraciones inseguras que pueden comprometer la integridad, confidencialidad y disponibilidad de los sistemas donde son desplegadas.

En este contexto, el presente proyecto propone el diseño y prototipado de un sistema automatizado para el análisis preventivo de imágenes Docker mediante herramientas especializadas de escaneo de vulnerabilidades, integrando específicamente Trivy y Grype dentro de un pipeline de integración continua (CI/CD), con el fin de detectar riesgos de seguridad antes del despliegue en entornos productivos.

1.1.Definición del proyecto

El presente proyecto consiste en el diseño y prototipado de un sistema automatizado que permita detectar vulnerabilidades en imágenes Docker provenientes de repositorios

públicos (Docker Hub) y repositorios privados simulados, antes de su despliegue en entornos productivos. El sistema integrará las herramientas de escaneo de código abierto Trivy y Grype dentro de un pipeline automatizado de CI/CD, con capacidad de generar reportes técnicos clasificados por severidad según el estándar CVSS v3.1 y de aplicar bloqueos automáticos de despliegue ante vulnerabilidades que superen umbrales definidos.

El problema central que atiende este proyecto es la ausencia generalizada de controles automatizados de validación de seguridad en los flujos de trabajo de despliegue con contenedores. Numerosas organizaciones utilizan imágenes Docker sin realizar procesos exhaustivos de verificación previa, lo que expone sus entornos de producción a vulnerabilidades conocidas (CVE), configuraciones débiles y dependencias comprometidas (Kaur et al., 2021). Un estudio comparativo de herramientas de análisis de contenedores indica que el 87 % de las imágenes utilizadas en producción contienen al menos una vulnerabilidad calificada como alta o crítica (Sysdig, 2023), y que incidentes como el compromiso de la cadena de suministro de Codecov en 2021 demostraron el impacto real de no contar con controles de integridad en las imágenes usadas en pipelines de construcción (ReversingLabs, 2025).

Como consecuencia directa de utilizar imágenes Docker no validadas, los entornos de producción quedan expuestos a:

- Ejecución de código malicioso embebido en capas de la imagen no auditadas;
- Escalamiento de privilegios mediante exploits en el kernel compartido entre contenedores;
- Fuga de información sensible a través de dependencias con vulnerabilidades de divulgación;
- Explotación de librerías vulnerables incluidas en las capas base de la imagen.

El prototipo resultante estará orientado a equipos de desarrollo y operaciones que

implementen prácticas DevSecOps, y constituirá un modelo experimental reproducible aplicable tanto en entornos académicos como en infraestructuras reales.

1.2.Justificación e Importancia del Trabajo de Investigación

La seguridad temprana dentro del ciclo de vida del software representa un componente crítico en la prevención de incidentes de ciberseguridad. El enfoque denominado *Shift-Left* propone trasladar los controles de seguridad hacia las fases iniciales del desarrollo, lo que, según la literatura especializada, puede reducir la ventana de exposición al riesgo en más del 60 % y disminuir el costo de corrección de vulnerabilidades de forma exponencial al detectarlas antes del despliegue (Bhardwaj, 2024).

El Instituto Nacional de Estándares y Tecnología (NIST) publicó la Publicación Especial 800-190, que establece las directrices de seguridad para entornos basados en contenedores. Souppaya et al. (2017) identifican cinco categorías de riesgo: vulnerabilidades en la imagen, configuraciones deficientes, secretos embebidos, software de origen no confiable y acceso amplio al host. El prototipo DockerSec Gate atiende la primera y cuarta categoría mediante escaneo automatizado de CVEs en tiempo de construcción.

Desde el punto de vista académico, este proyecto contribuye a la validación empírica de herramientas de escaneo de código abierto Trivy y Grype en escenarios controlados y reproducibles, generando datos cuantitativos sobre su precisión, cobertura y consistencia. Esta aportación resulta relevante dado que la literatura actual carece de estudios comparativos sistemáticos que evalúen el rendimiento conjunto de ambas herramientas bajo un mismo conjunto de métricas estandarizadas (Mirtaheri, 2025).

Los beneficiarios directos del prototipo son:

- Equipos de desarrollo (DevOps/DevSecOps): al integrar el análisis de vulnerabilidades en su pipeline de CI/CD, reducen el riesgo de promover imágenes comprometidas a producción.

- Organizaciones con infraestructura basada en contenedores: al disponer de un mecanismo automatizado de validación, fortalecen su postura de seguridad sin incrementar la carga operacional manual.
- Comunidad académica: al contar con un modelo experimental documentado y reproducible para futuras investigaciones en seguridad de contenedores.

La brecha que atiende este proyecto es la ausencia de un prototipo que combine el escaneo dual (Trivy y Grype), la aplicación de *security gates* automáticos y la generación de reportes estructurados en un flujo integrado y validado experimentalmente. Aunque herramientas como Trivy y Grype existen de forma aislada, su valor diferencial al operar en conjunto no ha sido sistematizado en un prototipo funcional con métricas de aceptación explícitas, dado que el enfoque de múltiple escáner para maximizar la cobertura y reducir puntos ciegos aún carece de validación experimental documentada. El índice de Jaccard entre herramientas de escaneo es de aproximadamente 0,68, lo que evidencia que ninguna herramienta por sí sola es suficiente (Mirtaheri, 2025).

La integración de la seguridad en los pipelines de CI/CD ha escalado como prioridad organizacional: según (GitLab, 2024), el 56 % de los equipos de desarrollo encuestados globalmente afirma haber acelerado la adopción de prácticas DevSecOps en el último año, y más del 70 % reporta utilizar contenedores en alguna fase de su ciclo de vida productivo. Estos datos respaldan la relevancia del prototipo propuesto como herramienta de apoyo a equipos que buscan automatizar los controles de seguridad sin incrementar la carga operacional.

En cuanto al respaldo de los beneficios declarados: la reducción de riesgos operacionales mediante análisis preventivo está sustentada en datos que muestran una media de 180 vulnerabilidades por imagen Docker en entornos de producción (Kaur et al., 2021); el fortalecimiento de DevSecOps se apoya en evidencia de que la integración de escaneo

automatizado en CI/CD reduce los tiempos de respuesta ante vulnerabilidades críticas (Bhardwaj, 2024); y la mejora en la resiliencia de la infraestructura tecnológica está documentada en el contexto de la gestión de la cadena de suministro de software (ReversingLabs, 2025).

1.3.Alcance

El presente proyecto abarca el diseño, construcción y validación experimental de un prototipo automatizado de detección de vulnerabilidades en imágenes Docker. A continuación, se delimitan con precisión los elementos incluidos y excluidos.

1.3.1. Elementos incluidos en el alcance:

- Diseño conceptual del sistema: arquitectura lógica del pipeline de escaneo automatizado con sus componentes, flujos de datos y puntos de decisión.
- Herramientas de escaneo: integración de Trivy y Grype en versiones fijas y documentadas. La selección de estas herramientas se justifica por su adopción generalizada en pipelines de CI/CD, su soporte activo de la comunidad, su capacidad de escanear tanto imágenes como sistemas de archivos y su bajo índice de resultados "desconocidos", que proporciona resultados accionables para los equipos de desarrollo (Mendonsa, 2025). Frente a Clair, que está orientado principalmente a la monitorización de registros de forma asíncrona y genera mayor volumen de ruido, Trivy y Grype ofrecen retroalimentación inmediata adecuada para el contexto *build-time* del prototipo (Mirtaheri, 2025).
- Repositorios evaluados: (a) imágenes públicas de Docker Hub tanto imágenes oficiales como de la comunidad; (b) un repositorio privado simulado con autenticación, ejecutado mediante un registro local (registry Docker privado).
- Tipos de vulnerabilidades evaluadas: exclusivamente vulnerabilidades de tipo

CVE registradas en la National Vulnerability Database (NVD) y en la GitHub Advisory Database (GHSA). No se evaluarán secretos embebidos ni errores de configuración de infraestructura como código (IaC) en esta versión del prototipo.

- Generación de reportes: reportes automatizados en formato JSON y HTML, clasificados por severidad CVSS v3.1 (Crítica, Alta, Media, Baja).
- Ambiente de validación: entorno Linux controlado (Debian o Ubuntu LTS) con versiones congeladas de herramientas y bases de datos de vulnerabilidades, para garantizar la reproducibilidad de los resultados.
- Security gates: mecanismo de bloqueo automático del pipeline ante la detección de vulnerabilidades con puntuación CVSS v3.1 ≥ 9,0 (Críticas).

1.3.2. Elementos excluidos del alcance:

- Implementación en entornos de producción reales de terceros.
- Análisis de vulnerabilidades en tiempo de ejecución del contenedor (análisis dinámico).
- Detección de secretos embebidos, análisis de configuraciones IaC ni evaluación de imágenes de orquestadores (Kubernetes).
- Remediación automatizada de vulnerabilidades detectadas (corrección de Dockerfiles o parches automáticos).
- Integración con formatos VEX (Vulnerability Exploitability eXchange) o SBOM (Software Bill of Materials): estos conceptos se describen en el marco teórico como contexto, pero no forman parte del prototipo en esta iteración.

1.4.Objetivos

1.4.1. Objetivo General

Diseñar y validar un prototipo automatizado para la detección de vulnerabilidades

CVE en imágenes Docker provenientes de repositorios públicos y privados, con el fin de prevenir el despliegue de imágenes vulnerables en entornos de producción.

1.4.2. Objetivos específicos

- Diseñar la arquitectura lógica del pipeline automatizado de escaneo, definiendo componentes, flujos de datos e interfaces, hasta obtener un diagrama de arquitectura aprobado por el equipo.
- Construir el prototipo funcional integrando Trivy y Grype en un entorno Linux controlado, logrando la ejecución automatizada exitosa sobre imágenes Docker provenientes de repositorios públicos y privados simulados.
- Generar reportes técnicos automatizados en formato JSON y HTML, clasificando las vulnerabilidades detectadas por severidad CVSS v3.1 sin intervención manual.
- Analizar la complementariedad entre Trivy y Grype mediante el índice de Jaccard, identificando los tipos de vulnerabilidades con mayor diferencia de cobertura entre herramientas.
- Evaluar la efectividad del prototipo mediante cuatro escenarios experimentales, verificando que las métricas de detección y cobertura superen los umbrales mínimos de aceptación definidos.

Capítulo 2:

2. REVISIÓN DE LITERATURA

La revisión de literatura presentada en este capítulo contextualiza el estado actual del conocimiento en seguridad de contenedores Docker y sienta las bases conceptuales del prototipo. Se organiza en dos secciones: el estado del arte, que presenta los trabajos e investigaciones recientes más relevantes, y el marco teórico, que desarrolla los fundamentos conceptuales y técnicos que sustentan el diseño del sistema.

2.1.Estado del Arte

La investigación sobre detección de vulnerabilidades en contenedores Docker ha experimentado un crecimiento significativo en los últimos cinco años, impulsado por la adopción masiva de arquitecturas de microservicios y los crecientes incidentes de seguridad en entornos de producción basados en contenedores.

2.1.1. Estudios sobre Herramientas de Escaneo de Contenedores

Mendonsa (2025) realizó una evaluación comparativa de las herramientas de escaneo de código abierto Trivy y Grype frente al servicio Amazon ECR en escenarios de repositorios privados. Los resultados demostraron que Trivy presentó una alta eficacia en la detección de vulnerabilidades críticas con un mínimo de resultados "desconocidos", mientras que Grype destacó por su integración profunda con el ecosistema Anchore y su capacidad de generación de SBOM mediante Syft. Ambas herramientas superaron en velocidad de respuesta al servicio administrado en entornos de construcción, lo que respalda su selección para pipelines de CI/CD (Javed & Toor, 2021).

Mirtaheri et al. (2025) analizaron las brechas de seguridad en entornos de contenedores bajo el paradigma DevSecOps. El estudio identificó que el índice de Jaccard entre los resultados de Trivy y Clair es de aproximadamente 0,68, evidenciando que ninguna herramienta individual cubre la totalidad de las vulnerabilidades aplicables (Tunde-Onadele

et al., 2019). Esta discrepancia sustenta el enfoque de escaneo dual adoptado en el presente prototipo, el cual combina Trivy y Grype. La referencia a Clair en el cálculo del índice de Jaccard corresponde a los datos disponibles en la literatura; el prototipo calculará el índice de Jaccard específicamente entre Trivy y Grype como parte de su validación experimental.

Kaur et al. (2021) documentaron que las imágenes Docker contienen en promedio 180 vulnerabilidades cada una, evidenciando la magnitud del problema en entornos de producción. Complementariamente, Sysdig (2023) reportó que el 87 % de las imágenes en producción presentan al menos una vulnerabilidad de severidad alta o crítica, lo que sustenta la necesidad de controles automatizados previos al despliegue. Adicionalmente, Haque y Babar (Haque & Babar, 2021) identificaron en un estudio empírico sobre 261 imágenes base que las imágenes basadas en Debian presentan una mayor carga de vulnerabilidades críticas en comparación con distribuciones ligeras como Alpine, lo que tiene implicaciones directas en la composición del conjunto de pruebas del prototipo.

El informe anual de Snyk sobre el estado de la seguridad en software de código abierto (Snyk, 2024) complementa los datos de Sysdig al reportar que más del 80 % de las bases de código analizadas contienen dependencias directas o transitivas con vulnerabilidades conocidas. El informe señala además que el tiempo medio entre la publicación de un CVE y su explotación activa en la naturaleza se ha reducido considerablemente en los últimos años, lo que refuerza la necesidad de controles automatizados en tiempo de construcción como los implementados en el prototipo DockerSec Gate.

(Zerouali et al., 2022) analizaron el impacto de paquetes JavaScript obsoletos y vulnerables en imágenes disponibles en Docker Hub, encontrando que una fracción significativa de las imágenes públicas incorpora dependencias npm con vulnerabilidades conocidas no actualizadas.

Este hallazgo es relevante para el prototipo porque Grype, uno de los escáneres

integrados, presenta cobertura especialmente profunda en el ecosistema npm, lo que hace que el escaneo dual Trivy+Grype sea particularmente efectivo para imágenes que incorporan aplicaciones Node.js, Python o Java como componentes de sus capas de sistema de archivos.

2.1.2. Incidentes Documentados en la Cadena de Suministro de Contenedores

El compromiso de Codecov en 2021 constituyó un hito en la concienciación sobre la seguridad de la cadena de suministro de software. En este incidente, un actor malicioso modificó el script de carga de Codecov, que era descargado e integrado en los pipelines de CI/CD de miles de organizaciones. Aunque el vector principal fue la alteración de un script y no directamente una imagen Docker, el caso demuestra que los artefactos utilizados en pipelines de construcción, incluyendo las imágenes Docker como componente crítico, son vulnerables ante la ausencia de mecanismos de verificación de integridad y análisis previo al uso (ReversingLabs, 2025). Este tipo de ataque es precisamente el que el prototipo busca mitigar, al interceptar imágenes antes de su integración en el pipeline productivo.

ReversingLabs (2025) documentó que en 2024 y 2025 se observó un incremento significativo en la sofisticación de los ataques a cadenas de suministro, dirigidos específicamente a pipelines de código abierto y sistemas de inteligencia artificial. El informe cuantifica que la exposición de secretos en repositorios de código abierto aumentó un 12 % en el último año, evidenciando la necesidad de controles automatizados en la inspección de imágenes antes de su uso.

2.1.3. Panorama de Vulnerabilidades en Contenedores (2020–2025)

Khalil (2025) documentó que en 2025 el volumen de vulnerabilidades registradas ha alcanzado niveles históricos, con un promedio de 133 CVEs reportados diariamente, de los cuales más de un tercio son calificados como altos o críticos (CVSS ≥ 7,0). Esta tendencia evidencia la insuficiencia de los métodos manuales de revisión y sustenta la necesidad de sistemas automatizados de triaje y bloqueo. La Tabla 1 sintetiza las métricas de riesgo más

relevantes del primer semestre de 2025.

Tabla 1
Métricas de riesgo en entornos de contenedores

Métrica de Riesgo	Valor estadístico	Implicación de seguridad	Fuente
Vulnerabilidades promedio por imagen Docker	180 CVEs	Necesidad de triaje automatizado masivo	(Kaur, Dugré, Hanna, & Glatard, 2021)
Imágenes con vulnerabilidades críticas o altas	87 %	Riesgo sistémico en entornos de producción	(Sysdig, 2023)
CVEs reportados diariamente (promedio)	133	Insuficiencia de los métodos manuales	(Khalil, 2025)
Brechas causadas por configuraciones erróneas	27 %	Importancia del análisis de postura (IaC)	(Haque & Babar, 2021)
Brechas causadas por exploits de kernel	19 %	Fallo en el aislamiento de namespaces/cgroups	(Mirtaheri, 2025)

Nota. Los valores corresponden a estadísticas consolidadas del primer semestre de 2025 en entornos de producción con contenedores.

2.2.Marco Teórico

El marco teórico desarrolla los fundamentos conceptuales y técnicos que sustentan el diseño del prototipo: la arquitectura de los contenedores Docker, los modelos de gestión de vulnerabilidades, la arquitectura de sistemas de análisis automatizado y las herramientas seleccionadas.

2.2.1. Fundamentos de la Tecnología de Contenedores y el Ecosistema Docker

La arquitectura de contenedores representa una evolución respecto a la virtualización tradicional (Bernstein, 2014). Mientras que las máquinas virtuales dependen de un hipervisor para emular hardware y ejecutar sistemas operativos completos, los contenedores operan mediante el aislamiento a nivel de sistema operativo, compartiendo el kernel del host. Esta característica es gestionada en Linux a través de espacios de nombres (namespaces) y grupos

de control (cgroups), lo que permite que los contenedores sean ligeros y portátiles (Haque & Babar, 2021). Sin embargo, la compartición del kernel es también una de las principales fuentes de riesgo, ya que una vulnerabilidad en este componente puede facilitar un escape de contenedor, permitiendo a un atacante comprometer la máquina anfitriona y otros contenedores adyacentes (Mirtaheri, 2025).

La revisión sistemática de (Pahl et al., 2019) publicada en IEEE Transactions on Cloud Computing documenta la evolución de las tecnologías de contenedores en la nube y cataloga los principales vectores de seguridad emergentes en arquitecturas de microservicios. Los autores identifican que el aislamiento basado en namespaces y cgroups, si bien suficiente para aislar procesos, no elimina la superficie de ataque compartida en el kernel del host, lo que convierte al control preventivo de imágenes en un mecanismo de defensa imprescindible.

La Open Container Initiative (OCI) estandariza el formato de las imágenes de contenedor mediante la especificación OCI Image Format (Open Container Initiative, 2021), que define cómo se empaquetan las capas, los manifiestos y los descriptores de configuración. Esta especificación es precisamente lo que permite a herramientas como Trivy y Grype descomprimir y analizar el contenido de cualquier imagen conforme al estándar sin necesidad de ejecutarla, siendo un prerequisito técnico del enfoque de escaneo estático adoptado.

(Ladisa et al., 2023) sistematizaron los patrones de ataque a la cadena de suministro de software de código abierto, identificando 10 categorías de vectores que van desde la inyección de código malicioso hasta el compromiso de repositorios de distribución de paquetes. El incidente de Codecov analizado en la sección 2.1.2 corresponde a la categoría de compromiso de artefactos de pipeline definida por los autores. DockerSec Gate atiende este vector al interceptar imágenes antes de su integración, actuando como control que impediría la propagación de un artefacto comprometido a los entornos de producción.

2.2.1.1.Estructura de Capas y Sistema de Archivos de Unión

Una imagen Docker se define como un modelo de solo lectura compuesto por una serie de capas superpuestas. Cada instrucción en un Dockerfile genera una nueva capa, utilizando sistemas de archivos de unión (UnionFS) como Overlay2 para consolidarlas en una vista única durante la ejecución. Esta arquitectura facilita la reutilización de capas comunes, pero complica la gestión de vulnerabilidades: una falla introducida en una capa base se propaga automáticamente a todas las imágenes descendientes (Haque & Babar, 2021).

La estandarización está regida por la *Open Container Initiative* (OCI), que define las especificaciones de imagen (*image-spec*) y de tiempo de ejecución (*runtime-spec*). Estas normas permiten que los escáneres estáticos descompongan la imagen en sus capas constituyentes y analicen el contenido de cada una sin ejecutar el contenedor (HATS at LPC, 2026).

2.2.1.2.Ciclo de Vida del Contenedor y Gestión de Registros

Los registros de imágenes actúan como el núcleo logístico del ciclo de vida del contenedor. Docker Hub se mantiene como el registro público predominante, pero la procedencia de sus imágenes es a menudo incierta: las imágenes mantenidas por la comunidad presentan una densidad de vulnerabilidades significativamente mayor que las imágenes oficiales, debido a la falta de ciclos de actualización regulares y auditorías de seguridad (Mendonsa, 2025). Los repositorios privados permiten a las organizaciones ejercer un control granular sobre el acceso y la integridad de las imágenes, constituyendo un punto crítico de control donde debe intervenir la automatización (HATS at LPC, 2026).

2.2.2. Gestión de Vulnerabilidades y Métricas de Seguridad

Para que el prototipo sea efectivo, debe ir más allá de la simple identificación de fallos y proporcionar un marco para la priorización y la toma de decisiones basada en el riesgo.

2.2.2.1.El Sistema de Puntuación de Vulnerabilidades Comunes (CVSS v3.1)

El CVSS es el estándar global para evaluar la severidad de las vulnerabilidades, proporcionando una puntuación numérica de 0 a 10 basada en métricas base, temporales y ambientales. El prototipo implementará la versión 3.1 de CVSS, actualmente referenciada por la NVD y adoptada como estándar por las herramientas Trivy y Grype, para la clasificación de las vulnerabilidades detectadas y la definición de los umbrales de los security gates. El criterio de decisión del prototipo será: bloqueo automático ante cualquier CVE con puntuación CVSS v3.1 ≥ 9,0 (Crítico) y alerta sin bloqueo para puntuaciones entre 7,0 y 8,9 (Alto) (Khalil, 2025).

El Sistema de Puntuación de Vulnerabilidades Comunes en su versión 3.1 (CVSSv3.1) es el estándar internacional que cuantifica la severidad de una vulnerabilidad mediante métricas base, temporales y ambientales (FIRST.org, 2019). La puntuación base oscila entre 0 y 10 y resulta de vectores como el vector de ataque, la complejidad de ataque, los privilegios requeridos y el impacto sobre confidencialidad, integridad y disponibilidad. El prototipo DockerSec Gate implementa esta especificación como eje del motor de score: toda vulnerabilidad con puntuación CVSSv3.1 ≥ 9,0 activa un bloqueo automático del pipeline en producción.

La National Vulnerability Database (NVD) del NIST actúa como repositorio canónico de vulnerabilidades conocidas, integrando los identificadores CVE con métricas CVSS, clasificaciones CWE y referencias técnicas (National Institute of Standards and Technology , 2024). El módulo de validación externa del prototipo (nvd_client.py) consulta la API REST de la NVD para enriquecer las vulnerabilidades detectadas con confianza MEDIA identificadas únicamente por uno de los dos escáneres y confirmar su existencia oficial antes de incorporarlas al cálculo del score.

En 2025, más de un tercio de las nuevas vulnerabilidades son calificadas como altas o

críticas, lo que genera lo que los expertos denominan una "pesadilla de triaje" que los métodos manuales no pueden abordar (Khalil, 2025). La Tabla 2 sintetiza los niveles de severidad y la estrategia de remediación adoptada en el prototipo.

Tabla 2
Niveles de severidad CVSS v3.1 y estrategia de respuesta del prototipo

Severidad	Rango CVSS v3.1	Acción del prototipo
Crítica	9,0 – 10,0	Bloqueo automático del pipeline
Alta	7,0 – 8,9	Alerta; parcheo en el siguiente ciclo de construcción
Media	4,0 – 6,9	Registro en reporte; remediación programada
Baja	0,1 – 3,9	Registro informativo; mitigación opcional

Nota. Criterios basados en Khalil (2025) y adaptados al alcance del prototipo.

2.2.2.2.VEX y SBOM: contexto teórico

El formato VEX (*Vulnerability Exploitability eXchange*) permite a los proveedores de software comunicar el estado real de una vulnerabilidad en un artefacto específico, indicando si el producto está afectado, corregido o si la vulnerabilidad no es explotable en esa configuración (Churakova et al., 2026). El SBOM (Sof*tware Bill of Materials*) proporciona el inventario completo de componentes de una imagen. Ambos conceptos se presentan aquí como contexto del panorama actual de gestión de vulnerabilidades; sin embargo, la generación e integración de VEX y SBOM no forma parte del alcance de este prototipo y queda propuesta como línea de trabajo futuro.

El Center for Internet Security publica el CIS Docker Benchmark, documento de mejores prácticas de seguridad que cubre la gestión de imágenes, la configuración del daemon, los controles de acceso y la auditoría de contenedores (Center for Internet Security , 2023). Las políticas de security gate implementadas en el prototipo son consistentes con las recomendaciones del CIS Benchmark en lo que respecta a la prohibición de imágenes con vulnerabilidades críticas en entornos de producción y a la exigencia de versiones con

corrección disponible (fix available) antes de autorizar el despliegue.

El inventario de componentes de software (SBOM) ha ganado relevancia regulatoria como mecanismo de trazabilidad de dependencias. (Balliu et al., 2023) analizaron el estado de madurez de la adopción del SBOM en la industria, concluyendo que, si bien cuenta con respaldo normativo creciente tras la orden ejecutiva de ciberseguridad de la Casa Blanca de 2021, su integración operativa sigue siendo heterogénea y dependiente de la herramienta utilizada. En el prototipo, la generación de SBOM mediante Syft queda identificada como línea de trabajo futuro, conforme al alcance establecido en la sección 1.4.

2.2.3. Arquitectura de Sistemas de Análisis Automatizado

2.2.3.1.El Paradigma Shift-Left en Contenedores

La filosofía *Shift-Left* propone mover las pruebas de seguridad hacia las fases iniciales del desarrollo. En la práctica, el escaneo de imágenes Docker debe ocurrir lo más cerca posible del momento en que el desarrollador realiza un commit de código. La evidencia empírica indica que la detección temprana puede reducir la ventana de exposición al riesgo en más del 60 % y que resolver una vulnerabilidad durante la fase de desarrollo es exponencialmente más económico que hacerlo tras un incidente en producción (Bhardwaj, 2024).

2.2.3.2.Diseño Lógico del Pipeline de Seguridad

Una arquitectura robusta de análisis automatizado se estructura en capas coordinadas. La Tabla 3 describe las capas del pipeline que serán implementadas en el prototipo. Todos los componentes listados en la tabla corresponden a elementos que serán efectivamente construidos como parte del prototipo; los componentes de inteligencia artificial y remediación automatizada quedan excluidos del alcance.

Tabla 3

Capas del pipeline de seguridad automatizado implementadas en el prototipo

Capa	**Función principal**	**Componentes del prototipo**
Ingesta	Captura de imágenes y disparadores de eventos	Descarga desde Docker Hub y registro privado local
Escaneo	Análisis estático de la imagen	Motores Trivy y Grype (versiones fijas)
Decisión	Evaluación frente a umbrales CVSS v3.1	Lógica de bloqueo: CVSS ≥ 9,0 → detención del pipeline
Reporte	Generación de informes estructurados	Reportes JSON y HTML clasificados por severidad
Orquestación	Gestión del flujo automatizado	Script de automatización del pipeline (shell/Python)

Nota. Adaptado de Bhardwaj et al. (2024). Solo se implementarán los componentes listados en esta tabla.

La integración de security gates automáticos en el pipeline es una de las funcionalidades más críticas: si el motor de escaneo identifica una vulnerabilidad CVSS v3.1 ≥ 9,0, el sistema detendrá el despliegue de forma automática, garantizando la integridad del entorno de producción (Bhardwaj, 2024).

Pecka et al. (2023) examinaron los patrones de integración de pruebas de seguridad en pipelines CI/CD, identificando que las organizaciones con mayor madurez DevSecOps estructuran sus controles en tres fases: análisis de código estático (Static Application Security Testing, SAST), análisis de composición de software (Software Composition Analysis, SCA) y análisis de imágenes de contenedor. El prototipo DockerSec Gate corresponde a la tercera fase de este modelo, y su diseño como microservicio con API REST permite su integración en la etapa de construcción de cualquier pipeline compatible con llamadas HTTP, tal como ilustra el ejemplo con GitHub Actions.

Tak et al. (2018) propusieron un marco de análisis estático de imágenes Docker orientado a detectar configuraciones débiles de aislamiento y accesos innecesarios a recursos del host. Su trabajo fundamenta la arquitectura de escaneo adoptada en el prototipo, en la cual

el análisis se realiza sobre la imagen antes de su ejecución, sin depender del comportamiento en tiempo de ejecución. Este enfoque es consistente con el paradigma Shift-Left descrito en la sección 2.2.3, que prioriza la identificación temprana de riesgos durante la fase de construcción del artefacto.

2.2.4. Análisis Comparativo de Herramientas de Escaneo

La selección de las herramientas de análisis es un paso fundamental en el prototipado. No todos los escáneres son equivalentes; varían en velocidad, profundidad de análisis, precisión y amplitud de sus bases de datos de vulnerabilidades (Mirtaheri, 2025).

Trivy se ha posicionado como una de las herramientas preferidas para la integración en pipelines de CI/CD por su alta velocidad y versatilidad: es capaz de escanear imágenes de contenedores, sistemas de archivos y configuraciones de infraestructura como código. En experimentos controlados, ha demostrado alta eficacia en la detección de vulnerabilidades críticas con mínima generación de resultados "desconocidos" (Mendonsa, 2025).

Clair está diseñado con enfoque orientado al registro (*registry-centric*): indexa capas de forma asíncrona y es potente para grandes almacenes de imágenes, pero puede generar mayor volumen de ruido y carece de la agilidad necesaria para retroalimentación inmediata en el pipeline de construcción (Mirtaheri, 2025). Por este motivo, Clair no fue seleccionado para el prototipo.

Grype, desarrollado por (Anchore, 2024), destaca por su capacidad de realizar escaneos detallados y su facilidad de integración con Syft para la generación de SBOM, así como por su alta eficacia en ecosistemas de paquetes de lenguajes de programación (npm, pip, Maven). Su inclusión junto a Trivy responde a la necesidad de maximizar la cobertura de vulnerabilidades mediante un enfoque de escaneo dual (Mendonsa, 2025).

Tabla 4

Comparativa técnica de herramientas de escaneo de imágenes Docker

Criterio de comparación	**Trivy**	**Clair**	**Grype**	**Fuente**
Velocidad de escaneo	Muy alta	Media	Alta	Mendonsa (2025)
Eficacia en vulnerabilidades críticas	Alta (≈100 %)	Alta	Alta	Mendonsa (2025)
Tasa de falsos positivos	Baja	Alta (muchos "Unknown")	Baja	Mirtaheri et al. (2025)
Consistencia de datos	Alta estabilidad temporal	Depende del índice	Alta	Mirtaheri et al. (2025)
Uso principal	Build-time (CI/CD)	Registry-monitoring	Análisis profundo de dependencias	Mendonsa (2025)
Integración con SBOM	Parcial (via Trivy SBOM)	No nativa	Nativa (via Syft)	Mendonsa (2025)

Nota. Criterios sustentados en los estudios experimentales de (Mendonsa, 2025) y (Mirtaheri, 2025).

El índice de Jaccard, que mide el solapamiento de los conjuntos de CVEs reportados entre herramientas de escaneo, es de aproximadamente 0,68 entre Trivy y Clair (Mirtaheri, 2025), lo que evidencia que depender de una sola herramienta deja puntos ciegos significativos. El prototipo calculará este índice específicamente entre Trivy y Grype ($J(\text{Trivy, Grype}) = |A \cap B| / |A \cup B|$), donde un valor $J \geq 0{,}70$ indicará consistencia aceptable entre herramientas.

2.2.5. Marco de Evaluación y Métricas de Rendimiento del Prototipo

La validación experimental del prototipo requiere un marco metodológico riguroso con métricas cuantitativas, umbrales de aceptación explícitos sustentados en la literatura y un protocolo reproducible. Este marco sienta las bases para la evaluación de los resultados obtenidos durante la implementación del prototipo.

2.2.5.1.Matriz de Confusión Aplicada a la Detección de Vulnerabilidades

- Toda métrica de desempeño se fundamenta en la matriz de confusión, que

clasifica cada vulnerabilidad evaluada en una de cuatro categorías (Bhardwaj, 2024):

- Verdadero Positivo (VP): vulnerabilidad real detectada correctamente por el prototipo.
- Falso Positivo (FP): alerta sobre una vulnerabilidad inexistente o no aplicable a la imagen analizada.
- Verdadero Negativo (VN): componente sin vulnerabilidad real clasificado correctamente como seguro.
- Falso Negativo (FN): vulnerabilidad real no detectada por el prototipo.
- Minimizar los Falsos Negativos es la prioridad de seguridad más crítica, ya que una vulnerabilidad no detectada que llega a producción representa el mayor riesgo operacional (Mendonsa, 2025).

2.2.5.2.Métricas de Efectividad

A partir de la matriz de confusión se derivan las siguientes métricas cuantitativas:

- Precisión (Precision): proporción de alertas que corresponden a vulnerabilidades reales. Una baja precisión implica fatiga de alertas.
- Tasa de Detección (Recall): proporción de vulnerabilidades reales que el sistema identificó. Es la métrica de seguridad más crítica.
- Puntuación F1 (F1-Score): media armónica entre Precision y Recall. Balancea ambas métricas en un único indicador.
- Exactitud (Accuracy): proporción de clasificaciones correctas totales.
- Cobertura (Coverage): proporción del catálogo de CVEs aplicables que el prototipo detecta.
- Tasa de Falsos Positivos (TFP): proporción de componentes no vulnerables marcados incorrectamente.

- Índice de Jaccard (J): consistencia entre los resultados de Trivy y Grype. J ≥ 0,70 indica consistencia aceptable (Mirtaheri, 2025).

2.2.6. Criterios de Aceptación del Prototipo

Los umbrales cuantitativos de la Tabla 5 se derivan de los valores reportados en la literatura especializada sobre seguridad de contenedores (Mendonsa, 2025; Mirtaheri et al., 2025; Bhardwaj et al., 2024). El umbral de Recall ≥ 0,93 corresponde al límite inferior reportado en evaluaciones de herramientas de escaneo de contenedores de referencia (Mendonsa, 2025); el umbral de Precisión ≥ 0,80 refleja el nivel mínimo operacionalmente aceptable para evitar fatiga de alertas en equipos de DevSecOps (Bhardwaj, 2024); y el tiempo de evaluación ≤ 5 minutos responde al requisito estándar de pipelines de CI/CD de alta cadencia (Mahale et al., 2026).

Tabla 5
Criterios de aceptación obligatorios (evaluables en este diseño experimental, con ground truth de CVE puntuados por construcción)

Métrica	Umbral objetivo	Umbral mínimo aceptable	Fuente de referencia
Recall (Tasa de Detección)	≥ 0,97	≥ 0,93	(Mendonsa, 2025)
Coverage (Cobertura)	≥ 0,92	≥ 0,85	(Mirtaheri, 2025)
Índice de Jaccard (inter-escáner)	≥ 0,75	≥ 0,65	(Mirtaheri, 2025)
Tiempo de Evaluación por imagen	≤ 3 min	≤ 5 min	(Mahale et al., 2026)

Nota. El prototipo se considera validado si supera el umbral mínimo aceptable en todas las métricas obligatorias y el umbral objetivo en al menos el 70 % de ellas.

Tabla 6

Métricas de referencia (No evaluables en este diseño experimental por ausencia de un oráculo completo de componentes; se reportan como meta deseable para un trabajo futuro, no como criterio de aceptación)

Métrica	**Valor de referencia deseable**	**Fuente de referencia**
Precision (Precisión)	≥ 0,90	(Bhardwaj, 2024)
Accuracy (Exactitud)	≥ 0,95	(Bhardwaj, 2024)
F1-Score	≥ 0,93	Derivado de Recall y Precisión
TFP (Tasa de Falsos Positivos)	≤ 0,10	(Mirtaheri, 2025)
Ventana de Exposición	≤ 24h (Objetivo) / ≤ 72h (Mínimo)	(Bhardwaj, 2024)

Nota. Estos valores se citan como referencia de la literatura para contextualizar resultados futuros; su medición formal requiere un oráculo completo de composición por imagen (ver limitación en sección 3.5.5) y por tanto no forman parte de la regla de validación del prototipo.

Capítulo 3:

3. DESARROLLO

El presente capítulo describe el marco metodológico empleado para el diseño, la construcción y la preparación de la validación experimental del prototipo automatizado de detección de vulnerabilidades en imágenes Docker. Se detalla el enfoque de investigación, la arquitectura lógica y física del sistema, la implementación de cada uno de sus componentes, el entorno tecnológico que garantiza la reproducibilidad y, finalmente, el diseño experimental con el cual se evaluará la efectividad del prototipo. Los resultados cuantitativos obtenidos a partir de la aplicación de esta metodología se presentan y discuten en el Capítulo 4.

3.1.Enfoque y Diseño de la Investigación

La investigación adopta un enfoque cuantitativo de tipo experimental aplicado, orientado a la construcción de un artefacto tecnológico y a su evaluación mediante métricas objetivas. El estudio se inscribe en el paradigma de la investigación en ciencias del diseño (*Design Science Research*), en el cual el conocimiento se genera a través de la creación y evaluación sistemática de un artefacto que resuelve un problema relevante en un dominio de aplicación (Bhardwaj, 2024).

El diseño metodológico se estructuró en cuatro fases secuenciales:

- Fase 1 Diseño conceptual: definición de la arquitectura lógica del pipeline de escaneo, sus capas, flujos de datos y puntos de decisión, a partir de los fundamentos establecidos en el marco teórico (sección 2.2).
- Fase 2 Implementación: construcción del prototipo funcional integrando los motores Trivy y Grype, la lógica de correlación, la validación externa multifuente y el motor de decisión, sobre un entorno Linux contenedorizado y con versiones congeladas.
- Fase Diseño experimental: definición del corpus de imágenes de prueba, del

ground truth de vulnerabilidades verificables por construcción y del conjunto de métricas y umbrales de aceptación obligatorios (Tabla 5), complementados por un conjunto de métricas de referencia no vinculantes (Tabla 6).

- Fase 4 Validación: ejecución del protocolo experimental sobre el corpus y medición de las métricas frente a los criterios de aceptación de la Tabla 5.

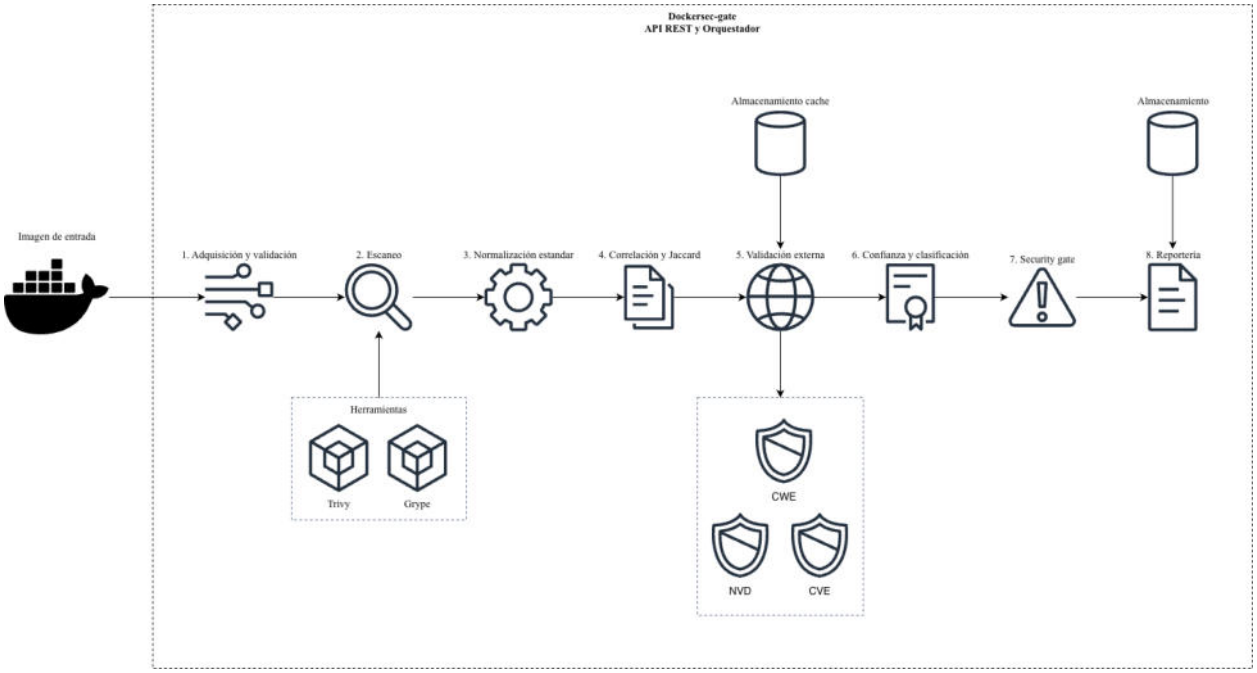
3.2.Arquitectura General del Prototipo

El prototipo, denominado DockerSec Gate, materializa el pipeline de seguridad conceptual descrito en la Tabla 3 como un microservicio web con una interfaz de programación de aplicaciones (API) de tipo REST. Si bien el alcance inicial (sección 1.3) preveía un script de automatización en shell/Python, durante la fase de implementación la orquestación evolucionó hacia un servicio REST asíncrono con persistencia y caché, decisión que fortalece la reproducibilidad, la trazabilidad de los escaneos y la integralidad del prototipo dentro de un pipeline de CI/CD sin alterar las capas funcionales originalmente definidas.

La arquitectura se organiza en capas coordinadas que procesan cada imagen de forma secuencial. El flujo completo, ilustrado en la Figura 1, comprende: (1) adquisición y validación de la imagen; (2) escaneo dual con Trivy y Grype en paralelo; (3) normalización de hallazgos a un esquema estándar; (4) correlación inter-escáner y cálculo del índice de Jaccard; (5) validación externa multifuente contra NVD, cve.org y cwe.mitre.org; (6) evaluación de confianza de los hallazgos ambiguos; (7) evaluación del security gate frente a políticas por ambiente; (8) generación de reportes estructurados; y (9) persistencia de resultados.

Figura 1
Arquitectura del pipeline automatizado de detección de vulnerabilidades del prototipo

DockerSec Gate



Nota. Elaboración propia. El escaneo dual (paso 2) ejecuta Trivy y Grype de forma concurrente; la validación externa (paso 5) consulta en paralelo a NVD, cve.org y cwe.mitre.org con apoyo de la caché Redis; el Security Gate (paso 7) bloquea el despliegue ante vulnerabilidades con CVSS v3.1 ≥ 9,0.

3.3.Implementación de los Componentes del Sistema

Esta sección describe la realización técnica de cada capa del pipeline. El prototipo se desarrolló en Python, empleando el framework asíncrono FastAPI sobre el servidor Uvicorn para la exposición de la API, MongoDB (mediante el controlador asíncrono de su motor de DB) para la persistencia y Redis para el almacenamiento en caché de las validaciones externas.

3.3.1. Capa de Adquisición

La capa de adquisición es responsable de obtener y validar la imagen objetivo antes del análisis. Se admiten tres orígenes: imágenes presentes en el demonio Docker local, imágenes empaquetadas como archivo (.tar) e imágenes provenientes de un repositorio público (Docker Hub) o privado simulado mediante un registry local con autenticación. Un componente gestor verifica la accesibilidad de la imagen y normaliza su referencia (nombre y

etiqueta) antes de invocar a los motores de escaneo. Esta separación garantiza que el pipeline sea independiente de la procedencia de la imagen, en coherencia con el carácter incierto de los registros públicos señalado por (Mendonsa, 2025) y (Sultan et al., 2019).

3.3.2. Capa de Escaneo Dual

El núcleo del análisis estático lo constituyen los motores Trivy (Aqua Security, 2024) y Grype (Anchore, 2024), ejecutados en versiones congeladas para garantizar la reproducibilidad. Ambos escáneres descomponen la imagen en sus capas constituyentes y analizan su contenido sin ejecutar el contenedor (Sultan et al., 2019), inventariando tanto los paquetes del sistema operativo como los manifiestos de dependencias de lenguajes (por ejemplo, package-lock.json, composer.lock o archivos .jar).

La ejecución de ambos motores se realiza de forma concurrente mediante primitivas asíncronas, lo que reduce el tiempo total de evaluación respecto a una ejecución secuencial. La estrategia de doble escaneo responde directamente a la evidencia de que ninguna herramienta individual cubre la totalidad de las vulnerabilidades aplicables, con un índice de solapamiento inter-escáner de aproximadamente 0,68 reportado en la literatura (Mirtaheri, 2025); la combinación de Trivy y Grype busca maximizar la cobertura y reducir los puntos ciegos.

3.3.3. Capa de Normalización

Dado que Trivy y Grype emiten resultados en formatos heterogéneos, se implementó una capa de normalización que transforma la salida de cada motor a un esquema estándar común, modelado con Pydantic. Este esquema unifica los campos relevantes de cada hallazgo: identificador CVE, paquete afectado, versión instalada y versión corregida, severidad, puntuación CVSS v3.1, alias (por ejemplo, identificadores GHSA), debilidades asociadas (CWE) y la herramienta de origen. La normalización es condición necesaria para la correlación posterior, pues permite comparar de forma homogénea los conjuntos de

vulnerabilidades reportados por ambos motores.

3.3.4. Capa de Correlación e Índice de Jaccard

La capa de correlación consolida los hallazgos normalizados de ambos escáneres en un único conjunto deduplicado, registrando para cada vulnerabilidad si fue detectada por Trivy, por Grype o por ambos. A partir de estos conteos se calcula el índice de Jaccard inter-escáner como medida de consistencia:

$$J(Trivy, Grype) = |A \cap B| / |A \cup B|$$

donde un valor $J \geq 0{,}70$ indica una consistencia aceptable entre herramientas (Mirtaheri, 2025). La correlación también resuelve equivalencias entre identificadores: cuando un motor reporta un aviso GHSA y el otro su CVE canónico, ambos se reconcilian como una sola vulnerabilidad mediante la expansión de alias, evitando una sobreestimación artificial de las discrepancias.

El índice de Jaccard se reporta en dos variantes complementarias. La variante global considera todos los hallazgos correlacionados. La variante accionable, en cambio, restringe el cálculo a las vulnerabilidades con parche disponible, es decir, aquellas con una versión corregida conocida. Estas vulnerabilidades constituyen el universo verdaderamente comparable entre ambos motores. Trivy se apoya en los avisos de seguridad de la distribución, que siempre conllevan una corrección. Grype, por su parte, incorpora además coincidencias de NVD/CPE que pueden carecer de parche.

Es importante subrayar que esta restricción no implica pérdida de información. El prototipo reporta la totalidad de las vulnerabilidades detectadas incluidas las que no disponen de parche en sus salidas JSON, HTML y Excel. La restricción se aplica únicamente al cálculo de la métrica de consistencia inter-escáner, con el fin de comparar ambos motores sobre la misma base accionable.

La variante accionable se adopta como referencia para el umbral de consistencia de la

Tabla 5. La variante global, en cambio, se conserva únicamente como valor informativo.

3.3.5. Capa de Validación Externa Multifuente

Para enriquecer y verificar los hallazgos, se implementó una capa de validación externa multifuente que consulta, para cada CVE, tres fuentes canónicas en paralelo:

- La (National Institute of Standards and Technology , 2024), fuente de referencia para la puntuación CVSS v3.1 y los identificadores CWE.
- El CVE Program (cve.org), registro autoritativo del programa CVE, que aporta el estado del registro, descripciones y métricas declaradas por la entidad numeradora (CNA).
- El CWE de MITRE (cwe.mitre.org), consultado para enriquecer cada identificador CWE con el nombre legible y la descripción del tipo de debilidad.

Un componente agregador ejecuta las consultas concurrentemente con tolerancia a fallos: si una fuente no responde, las restantes continúan y el error se registra de forma aislada, sin interrumpir el escaneo. El agregador consolida los valores siguiendo una política de prioridad (CVSS de NVD sobre cve.org) y registra las discrepancias entre fuentes para fines de auditoría. Con el objeto de respetar los límites de tasa de las APIs públicas y evitar la saturación del sistema, las consultas se acotan mediante un semáforo de concurrencia y un limitador de tasa por fuente, y los resultados se almacenan en una caché Redis con tiempo de vida configurable, lo que evita repetir consultas idénticas entre imágenes. Para imágenes con miles de hallazgos, la validación externa se concentra en los CVE de mayor severidad, dado que el índice de Jaccard y la decisión del gate no dependen de este enriquecimiento.

3.3.6. Capa de Evaluación de Confianza

Los hallazgos reportados por un único motor o con metadatos incompletos se someten a una evaluación de confianza que asigna a cada vulnerabilidad un puntaje y una clasificación

(por ejemplo, *legítima* o de confianza media) en función de un conjunto de factores, tales como la concordancia entre escáneres, la disponibilidad de versión corregida y la confirmación por las fuentes externas. Este mecanismo reduce el ruido y aporta criterios trazables para la posterior interpretación de los falsos positivos.

3.3.7. Capa de Decisión: Security Gate y Políticas por Ambiente

El motor de decisión (security gate) evalúa el conjunto consolidado de vulnerabilidades frente a una política asociada al ambiente de despliegue (producción, staging o desarrollo). La política de producción aplica el criterio más estricto, deteniendo automáticamente el pipeline ante cualquier vulnerabilidad con puntuación CVSS v3.1 ≥ 9,0 (Crítica), conforme a la estrategia de respuesta definida en la Tabla 2. El gate produce un veredicto (aprobado o bloqueado), una puntuación de riesgo agregada y la justificación de la decisión, materializando el principio Shift-Left de trasladar el control de seguridad a las fases tempranas del ciclo de vida (Bhardwaj, 2024) y (Rajapakse, 2022).

3.3.8. Capa de Reportes y Persistencia

Finalmente, el prototipo genera reportes estructurados clasificados por severidad CVSS v3.1. Conforme al alcance declarado, los formatos primarios son JSON (apto para integración programática) y HTML (apto para revisión humana); de forma complementaria, se incorporó la exportación a Excel para facilitar el análisis tabular de los resultados.

Cada escaneo junto con su veredicto y la ruta de su reporte se persiste en MongoDB (MongoDb inc., 2024), lo que habilita la consulta histórica y el cálculo agregado de métricas sobre el corpus. Dado que la salida cruda de Trivy y Grype puede superar el límite de 16 MB por documento de MongoDB en imágenes con miles de vulnerabilidades, esta evidencia se persiste por separado mediante GridFS, particionada en chunks y comprimida con gzip, referenciada desde el documento operativo mediante un identificador (datos_crudos_gridfs_id). De esta forma se preserva la trazabilidad forense y el principio de

no repudio de cada veredicto, sin comprometer el rendimiento de las consultas operativas ni el cálculo de métricas, que continúan alimentándose exclusivamente de los hallazgos normalizados.

3.4.Entorno de Implementación y Reproducibilidad

Con el fin de garantizar la reproducibilidad de los experimentos, el prototipo y sus servicios de soporte se desplegaron sobre un entorno Linux contenedorizado, orquestado mediante Docker Compose, con versiones fijas de las herramientas y de las bases de datos de vulnerabilidades. La validación experimental se ejecutó sobre una máquina virtual con sistema operativo Ubuntu 24.04 LTS (x86_64), 2 vCPU y 4 GB de RAM, y la plataforma de las imágenes se fijó explícitamente en linux/amd64. Esta fijación de arquitectura es un control de reproducibilidad: garantiza que los resultados sean independientes de la arquitectura de la máquina anfitriona (por ejemplo, equipos Apple Silicon de arquitectura arm64), sobre la cual ciertas imágenes antiguas no disponen de variante nativa. La Tabla 7 sintetiza el stack tecnológico empleado.

Tabla 7
Stack tecnológico del prototipo DockerSec Gate

Componente	Tecnología	Función	Justificación
Lenguaje	Python	Lógica del orquestador y de las capas del pipeline	Ecosistema maduro para I/O concurrente (asyncio) y con las librerías cliente oficiales de Trivy/Grype/MongoDB necesarias para el pipeline (Python Software Foundation, 2024)
API / servidor	FastAPI + Uvicorn	Exposición de los endpoints REST de escaneo y métricas	FastAPI se ejecuta sobre el estándar ASGI (no WSGI), lo que permite manejar de forma no bloqueante el escaneo dual (Trivy+Grype) y las llamadas paralelas a NVD/cve.org/CWE sin agotar hilos del servidor (Ramírez, 2024)

Componente	Tecnología	Función	Justificación
Motor de escaneo 1	Trivy (Aqua Security)	Análisis estático de paquetes OS y dependencias	(Aqua Security, 2024)
Motor de escaneo 2	Grype (Anchore)	Análisis estático complementario de dependencias	(Anchore, 2024)
Persistencia	MongoDB (Motor)	Almacenamiento de escaneos, veredictos y reportes	Modelo de documentos adecuado para el esquema semiestructurado y variable de los hallazgos normalizados de ambos escáneres (MongoDb inc., 2024)
Caché	Redis	Almacenamiento temporal de validaciones externas	Estructura clave-valor en memoria que reduce la latencia y el número de llamadas repetidas a NVD/cve.org/CWE bajo *rate limit* (Redis Ltd., 2024)
Validación de datos	Pydantic	Modelado del esquema estándar de vulnerabilidad	Validación declarativa basada en *type hints* de Python, integrada nativamente con FastAPI para garantizar que cada hallazgo normalizado cumpla el esquema estándar antes de persistirse (Colvin, S. et al., 2024)
Reportería	Jinja2, openpyxl	Generación de reportes HTML y Excel	Jinja2 separa la lógica de presentación del backend mediante plantillas; openpyxl implementa el formato OOXML (.xlsx) sin depender de Excel instalado, adecuado para un microservicio contenedorizado (Pallets, 2024)
Cliente HTTP	httpx	Consulta asíncrona a NVD, cve.org y cwe.mitre.org	Cliente HTTP con soporte nativo async/await, necesario para paralelizar la validación multifuente sin bloquear el event loop de FastAPI (Encode, 2024)
Contenedorización	Docker, Docker Compose	Despliegue reproducible de los servicios	Fija versiones de imagen y variables de entorno de todos los servicios (API, MongoDB, Redis) como unidad reproducible, control

Componente	Tecnología	Función	Justificación
			de reproducibilidad exigido por el diseño experimental (sección 3.4) (Docker Inc., 2024)

Nota. Elaboración propia. Todas las herramientas se ejecutaron en versiones congeladas y documentadas para garantizar la reproducibilidad de los resultados.

3.5.Diseño Experimental

3.5.1. Corpus de Imágenes de Prueba

Para evaluar el prototipo bajo condiciones contrastantes, se diseñó un corpus estratificado de imágenes Docker organizado en cuatro categorías, definidas según su perfil de riesgo esperado. Esta estratificación permite verificar tanto la sensibilidad del prototipo (capacidad de detectar vulnerabilidades donde se sabe que existen) como su especificidad (capacidad de no generar alertas donde se espera que no existan). La definición de las categorías se sustenta en la evidencia empírica de que la densidad de vulnerabilidades varía significativamente según la imagen base y su grado de actualización: las imágenes basadas en Debian presentan una carga de vulnerabilidades críticas mayor que las distribuciones ligeras como Alpine (Haque & Babar, 2021), y los contenedores desactualizados acumulan severidades crecientes (Zerouali et al., 2022) (Wist et al., 2021). La Tabla 8 describe la composición del corpus.

Tabla 8
Composición del corpus experimental por categoría de riesgo

Categoría	Perfil de riesgo	Criterio de selección	Resultado esperado
A	Deliberadamente vulnerables	Imágenes construidas alrededor de vulnerabilidades conocidas	Alta densidad de CVE; detección de los CVE plantados
B	Obsoletas / fin de vida (EOL)	Versiones antiguas con soporte finalizado	Alta densidad de CVE acumulados
C	Recientes / parchadas	Versiones actuales y mantenidas	Baja densidad de CVE
D	Mínimas / distroless	Imágenes sin gestor de paquetes ni shell	Densidad de CVE cercana a cero

Nota. Elaboración propia. Las categorías A y B evalúan la tasa de detección (Recall) y la

cobertura; las categorías C y D evalúan la especificidad y la tasa de falsos positivos.

3.5.2. Ground Truth y Verificación por Construcción

La validez de las métricas de efectividad depende críticamente de la calidad del ground truth (conjunto de vulnerabilidades verdaderamente presentes). Para evitar la circularidad metodológica que se produciría si se tomara la salida del propio escáner como verdad de referencia, se adoptó un ground truth verificable por construcción: la categoría A se compone de imágenes construidas deliberadamente alrededor de vulnerabilidades documentadas: la familia Shellshock en el paquete bash (CVE-2014-6271 y relacionadas), las vulnerabilidades de ejecución remota de Apache Struts 2.3.30, y la familia Log4Shell del componente log4j-core 2.14.1 empaquetado en Apache Solr 8.11.0 (CVE-2021-44228 y relacionadas), cuyos CVE asociados están registrados en la NVD y en avisos oficiales con independencia de la herramienta de escaneo. Estas tres familias cubren dos ecosistemas: paquetes del sistema operativo (bash) y dependencias Java empaquetadas (Struts y log4j-core).

Cada vulnerabilidad del ground truth se clasificó según su alcance respecto del análisis de composición de software (Software Composition Analysis, SCA), tal como se documenta en la Tabla 9. Esta distinción es metodológicamente necesaria porque los escáneres estáticos detectan paquetes del sistema operativo y manifiestos de dependencias, pero no analizan código fuente incrustado (vendorizado) ni componentes (por ejemplo, complementos) que no fueron empaquetados en la imagen. Los CVE cuyo artefacto vulnerable no es observable por SCA se documentan como una limitación de cobertura de la técnica y no se contabilizan como falsos negativos del prototipo, evitando así penalizar al sistema por un límite intrínseco del análisis estático.

Tabla 9

Clasificación del ground truth según el alcance del análisis de composición de software

Alcance	Definición	Tratamiento en las métricas
In-scope	El artefacto vulnerable está presente como componente rastreable por SCA (paquete OS o manifiesto de dependencia)	Contabilizado como VP/FN en la matriz de confusión
Fuera de alcance SCA	El CVE es real para el caso, pero el artefacto no es observable por SCA (código vendorizado sin manifiesto o complemento no empaquetado)	No contabilizado como FN; reportado como limitación de cobertura

Nota. Elaboración propia. La clasificación se aplicó previamente a la ejecución del prototipo para preservar la independencia entre el ground truth y la salida del escáner.

3.5.3. Variables, Métricas e Instrumentos de Medición

La efectividad del prototipo se operacionalizó mediante el conjunto de métricas definido en la sección 2.2.5, derivado de la matriz de confusión (VP, FP, VN, FN): Precision, Recall, F1-Score, Accuracy, Coverage y Tasa de Falsos Positivos, complementadas con el Índice de Jaccard inter-escáner y el tiempo de evaluación por imagen. Cada métrica se contrasta contra el umbral objetivo y el umbral mínimo aceptable de la Tabla 5, considerándose el prototipo validado si supera el mínimo en todas las métricas y el objetivo en al menos el 70 % de ellas.

El cálculo de las métricas se automatizó mediante un módulo de métricas que implementa las fórmulas de la matriz de confusión, el índice de Jaccard y la comparación contra umbrales, así como la agregación de resultados sobre el corpus. Este instrumento toma como entrada los escaneos persistidos en MongoDB y el ground truth documentado, garantizando que la medición sea determinista y reproducible.

3.5.4. Procedimiento Experimental

El protocolo de validación se ejecutó conforme a los siguientes pasos:

- Preparación del entorno: despliegue de los servicios (API, MongoDB, Redis) mediante Docker Compose con versiones congeladas.
- Ejecución del escaneo: para cada imagen del corpus, invocación del endpoint de escaneo del prototipo, que dispara el pipeline completo descrito en la

sección 3.2.

- Persistencia: almacenamiento de los hallazgos normalizados, el veredicto del gate y las estadísticas de correlación en MongoDB.
- Extracción de los conjuntos detectados: obtención, por imagen, del conjunto de CVE detectados (incluyendo la reconciliación de alias).
- Cálculo de métricas: comparación de los conjuntos detectados contra el ground truth in-scope para construir la matriz de confusión y derivar las métricas de efectividad; cálculo del índice de Jaccard a partir de las estadísticas de correlación.
- Evaluación frente a umbrales: contraste de cada métrica con los criterios de aceptación de la Tabla 5 y emisión del veredicto de validación.

3.5.5. Consideraciones de Validez y Limitaciones Metodológicas

Se reconocen dos limitaciones metodológicas que condicionan la interpretación de los resultados. En primer lugar, la Recall y la Coverage medidas sobre los CVE plantados in-scope constituyen métricas rigurosas, pues su verdad de referencia es independiente del escáner; en cambio, las métricas que requieren contabilizar falsos positivos y verdaderos negativos (Precision, Accuracy y Tasa de Falsos Positivos) no son plenamente medibles a partir de un ground truth de CVE plantados, dado que las imágenes acarrean numerosas vulnerabilidades reales adicionales que no deben interpretarse como falsos positivos; en consecuencia, dichas métricas se reportan bajo un universo controlado y con la salvedad correspondiente. En segundo lugar, el análisis se restringe al alcance del SCA estático, por lo que las vulnerabilidades alojadas en código vendorizado o en complementos no empaquetados quedan documentadas como limitaciones de cobertura. Ambas consideraciones se retoman en el Capítulo 4 al discutir los resultados.

3.6.Síntesis del Capítulo

Este capítulo presentó la metodología empleada para diseñar, implementar y preparar la validación del prototipo DockerSec Gate. Se describió un enfoque cuantitativo de ciencias del diseño, una arquitectura en capas materializada como microservicio REST, la implementación de las capas de adquisición, escaneo dual, normalización, correlación, validación externa multifuente, confianza, decisión y reportería, así como el entorno contenedorizado que garantiza la reproducibilidad. Finalmente, se definió el diseño experimental: un corpus estratificado en cuatro categorías de riesgo, un ground truth verificable por construcción y clasificado por alcance de SCA, el conjunto de métricas e instrumentos de medición, y el protocolo de ejecución. La aplicación de esta metodología y los resultados obtenidos se exponen en el Capítulo 4.

Capítulo 4

4. ANÁLISIS DE RESULTADOS

Este capítulo presenta y discute los resultados obtenidos al aplicar la metodología del Capítulo 3 sobre el corpus experimental completo. Se reportan, en este orden: la caracterización del corpus y su carga de vulnerabilidades, la consistencia interescáner medida con el índice de Jaccard, la eficacia de detección evaluada mediante la matriz de confusión sobre los CVE plantados, un proxy de precisión basado en la confirmación externa de los hallazgos, el rendimiento temporal del prototipo, el comportamiento del prototipo sobre un conjunto adicional de repositorios privados de uso real y, finalmente, la evaluación integral frente a los criterios de aceptación de la Tabla 5. El experimento principal se ejecutó sobre 51 imágenes públicas distribuidas en cuatro categorías de riesgo, bajo la política de ambiente desarrollo; de forma complementaria, se evaluaron 3 imágenes adicionales alojadas en repositorios privados de GitLab bajo la política de ambiente producción, conforme se detalla en la sección 4.2.5. Todo el experimento se ejecutó en un entorno Linux x86_64 controlado (Ubuntu 24.04 LTS, 2 vCPU, 4 GB de RAM) con la plataforma fijada en linux/amd64 para garantizar la reproducibilidad.

4.1.Pruebas de concepto

4.1.1. Caracterización del corpus y carga de vulnerabilidades

El prototipo procesó las 51 imágenes sin fallos de ejecución (código de salida 200 en todas), detectando en conjunto 80 097 vulnerabilidades, de las cuales 4 225 fueron críticas. La Tabla 9 resume, por categoría de riesgo, la carga de vulnerabilidades, el número de imágenes aprobadas por el *security gate* y el tiempo medio de análisis.

Tabla 10
Caracterización del corpus por categoría de riesgo (51 imágenes)

Categoría	n	Total vulnerab.	Críticas	Aprobadas	Tiempo medio (s)
A (deliberadamente vulnerables)	11	28 092	1 750	0	170,0

Categoría	n	Total vulnerab.	Críticas	Aprobadas	Tiempo medio (s)
B (obsoletas / EOL)	20	47 920	2 319	0	141,5
C (recientes / parchadas)	10	3 665	143	2	85,3
D (mínimas / *distroless)*	10	420	13	7	38,3
Total	51	80 097	4 225	9	116,4

Nota. Elaboración propia. El veredicto corresponde a la política de ambiente desarrollo. Se aprobaron 9 imágenes (todas de las categorías C y D) y se bloquearon 42.

Las categorías A y B concentran el 95 % de la carga (28 092 y 47 920 vulnerabilidades respectivamente), coherente con la evidencia de que las imágenes deliberadamente vulnerables y desactualizadas acumulan severidades crecientes (Kaur et al., 2021) (Zerouali et al., 2022). En el extremo opuesto, la categoría D (mínima/*distroless*) presenta una densidad casi nula (420 vulnerabilidades en diez imágenes; tres de ellas con cero). El *security gate* discriminó correctamente ambos extremos: ninguna imagen de A o B fue aprobada, mientras que 7 de 10 imágenes *distroless* y 2 de 10 recientes/parchadas sí lo fueron, materializando la función de control de despliegue del prototipo.

4.2.Análisis de Resultados

4.2.1. Consistencia Interescaner

El índice de Jaccard se calculó en sus dos variantes definidas en la sección 3.3.4: global (todos los hallazgos) y accionable (vulnerabilidades con parche disponible). La Tabla 11 reporta ambos valores por categoría y para el corpus completo.

Tabla 11
Índice de Jaccard interescáner (Trivy vs. Grype), global y accionable

Categoría	n	Jaccard global	Jaccard accionable
A (vulnerables)	11	0,4999	0,6261
B (EOL)	20	0,3327	0,5435
C (parchadas)	10	0,4849	0,7573
D (*distroless*)	10	0,7107	0,7993
Corpus (promedio)	51	0,4727	0,6534

Nota. Elaboración propia. Convención: unión vacía equivale a consistencia 1,0. La variante accionable es la evaluada contra el umbral de la Tabla 5; la global se reporta como valor informativo.

El Jaccard global del corpus fue de 0,4727, por debajo del umbral mínimo de 0,65. El análisis de la divergencia confirma su origen: se concentra en las vulnerabilidades sin parche disponible que Grype reporta mediante coincidencias de NVD/CPE y que Trivy omite por carecer de un aviso de seguridad de la distribución. El fenómeno es especialmente marcado en las imágenes de distribuciones en fin de vida: en ubuntu:16.04 y ubuntu:18.04, Trivy no reportó ninguna vulnerabilidad mientras Grype reportó 448 y 151 respectivamente (Jaccard 0,0), y en golang:1.15 la consistencia cae a 0,146. Al restringir el cálculo al universo accionable aquel sobre el que ambos motores operan con la misma base, el Jaccard del corpus asciende a 0,6534, superando el umbral mínimo de 0,65, si bien con un margen estrecho atribuible al peso de la categoría B (20 de 51 imágenes), que es la de menor consistencia (0,5435).

Esta brecha entre ambas variantes no constituye una deficiencia del prototipo, sino evidencia empírica directa de la complementariedad que justifica el escaneo dual: cada motor aporta cobertura que el otro no provee, lo que maximiza la detección y reduce los puntos ciegos (Mendonsa, 2025) (Mirtaheri, 2025). La restricción al universo accionable se aplica exclusivamente al cálculo de la métrica de consistencia: el prototipo reporta la totalidad de las vulnerabilidades detectadas en sus salidas JSON, HTML y Excel, sin pérdida de información.

4.2.2. Eficacia de la detección: Matriz de Confusión

La eficacia de detección se evaluó comparando los CVE detectados contra el *ground truth in-scope* (verificable por construcción). La Tabla 12 presenta la matriz de confusión sobre las tres imágenes plantadas, que cubren tres familias de vulnerabilidad icónicas en dos ecosistemas (paquete del sistema operativo y archivos Java).

Tabla 12
Matriz de confusión sobre los CVE plantados in-scope

Imagen (familia / ecosistema)	VP	FN	Recall	Coverage
cve-2014-6271 / Shellshock (bash / OS)	6	0	1,000	1,000

Imagen (familia / ecosistema)	VP	FN	Recall	Coverage
struts2:2.3.30 / Struts RCE (Java)	4	0	1,000	1,000
solr:8.11.0 / Log4Shell (log4j-core 2.14.1 / Java)	4	0	1,000	1,000
Global	14	0	1,000	1,000

Nota. Elaboración propia. VP = verdaderos positivos; FN = falsos negativos. Los conteos FP/VN y las métricas Precision, Accuracy y TFP no se reportan: no son medibles a partir de un ground truth de CVE plantados (las imágenes acarrean cientos de CVE reales adicionales que no constituyen falsos positivos), conforme a la limitación declarada en la sección 3.5.5.

El prototipo detectó la totalidad de los CVE plantados *in-scope* (14 de 14), alcanzando una tasa de detección (Recall) y una cobertura (Coverage) de 1,000. El resultado es notable porque cubre tres vectores distintos: una vulnerabilidad de paquete del sistema operativo (Shellshock en bash), una vulnerabilidad de dependencia Java empaquetada (Apache Struts) y la vulnerabilidad crítica más relevante de los últimos años (Log4Shell, los cuatro CVE del jar log4j-core 2.14.1). Dado que la prioridad de seguridad más crítica es minimizar los falsos negativos (Mendonsa, 2025), este resultado evidencia que el prototipo cumple su objetivo central de detección sobre los artefactos dentro del alcance del análisis de composición de software.

4.2.3. Precisión por Existencia (proxy)

Dado que la precisión formal no es medible sin un oráculo completo, se empleó un proxy de precisión por existencia: la fracción de CVE detectados (validados externamente) que NVD o cve.org confirman como reales. La Tabla 13 resume los resultados.

Tabla 13
Proxy de precisión por existencia (confirmación en NVD/cve.org)

Imagen	Confirmados / validados	Precisión por existencia
cve-2014-6271	200 / 200	1,000
struts2:2.3.30	200 / 200	1,000
solr:8.11.0 (Log4Shell)	200 / 200	1,000
bkimminich/juice-shop	104 / 109	0,954
nginx:1.27-alpine	165 / 165	1,000
Global	869 / 874	0,9943

Nota. Elaboración propia. En imágenes con miles de hallazgos, la validación externa se acota

a los 200 CVE más severos (sección 3.3.5), por lo que el valor 200/200 corresponde a esa muestra de mayor impacto.

El 99,43 % de los CVE validados externamente fueron confirmados por las fuentes canónicas, lo que indica una incidencia prácticamente nula de identificadores fabricados o inexistentes entre los hallazgos de mayor severidad. Aunque este proxy no sustituye a la precisión formal de la matriz de confusión, constituye un indicador transparente y verificable de la fiabilidad de los hallazgos del prototipo.

4.2.4. Evaluación Frente a los Criterios de Aceptación

La Tabla 14 contrasta las métricas obtenidas contra los umbrales de la Tabla 5.

Tabla 14
Evaluación de las métricas frente a los criterios de aceptación

Métrica	Valor	Objetivo	Mínimo	¿Cumple mín.?	¿Cumple obj.?
Recall	1,0000	≥ 0,97	≥ 0,93	Sí	Sí
Coverage	1,0000	≥ 0,92	≥ 0,85	Sí	Sí
Jaccard (accionable)	0,6534	≥ 0,75	≥ 0,65	Sí	No
Tiempo por imagen (min)	1,94	≤ 3	≤ 5	Sí	Sí

Nota. Elaboración propia. El Jaccard global informativo fue 0,4727. Regla de validación: superar el umbral mínimo en las cuatro métricas obligatorias y el umbral objetivo en al menos el 70 % de ellas (equivalente a 3 de 4). Precision, Accuracy, F1-Score, TFP y Ventana de Exposición son métricas de referencia (Tabla 6), no criterios de aceptación del prototipo, y se excluyen de esta regla de validación por no ser medibles sin un oráculo completo de componentes o instrumentación adicional (ver sección 3.5.5).

De las cuatro métricas obligatorias, el prototipo supera el umbral mínimo en las cuatro (4/4) y el umbral objetivo en tres (3/4 = 75 %), cumpliendo la regla de aceptación. En consecuencia, el prototipo se considera validado. El único objetivo no alcanzado es el Jaccard accionable (0,6534 frente a 0,75), cuya explicación la complementariedad de cobertura entre Trivy y Grype, acentuada por el peso de las imágenes en fin de vida se discutió en la sección 4.2.1. Las métricas de referencia (Precision, Accuracy, F1-Score, TFP y Ventana de Exposición) no forman parte de la regla de validación de este prototipo (Tabla 6); su medición formal requiere un oráculo completo de componentes o instrumentación adicional, y se proponen como trabajo futuro.

4.2.5. Validación Complementaria Sobre Repositorios Privados

Con el propósito de complementar el corpus público con evidencia de uso real, se ejecutó el prototipo contra tres imágenes provenientes de repositorios privados de la organización Golden Companies S.A, alojados en un registry de GitLab con autenticación, en correspondencia con el componente de repositorio privado declarado en el alcance del estudio (sección 1). A diferencia del corpus público, evaluado bajo la política de ambiente desarrollo, estas tres imágenes se evaluaron bajo la política de ambiente producción la más estricta de las tres definidas en la Tabla 4, que no permite ninguna vulnerabilidad CRITICAL y exige un score mínimo de 80,0 con el fin de observar el comportamiento del security gate en las condiciones de mayor exigencia bajo las que se desplegarían estos servicios en un entorno productivo real.

Las tres imágenes corresponden a servicios productivos activos de la organización y se clasifican, por su naturaleza de imágenes recientes y en mantenimiento activo (no deliberadamente vulnerables ni en fin de vida), dentro de la misma categoría de riesgo C (recientes/parchadas) empleada para el corpus público, ampliando dicha categoría de 10 a 13 imágenes. La Tabla 15 detalla la imagen, el repositorio de origen y el resultado del escaneo para cada caso.

Tabla 15
Resultados del escaneo sobre imágenes de repositorios privados (ambiente producción)

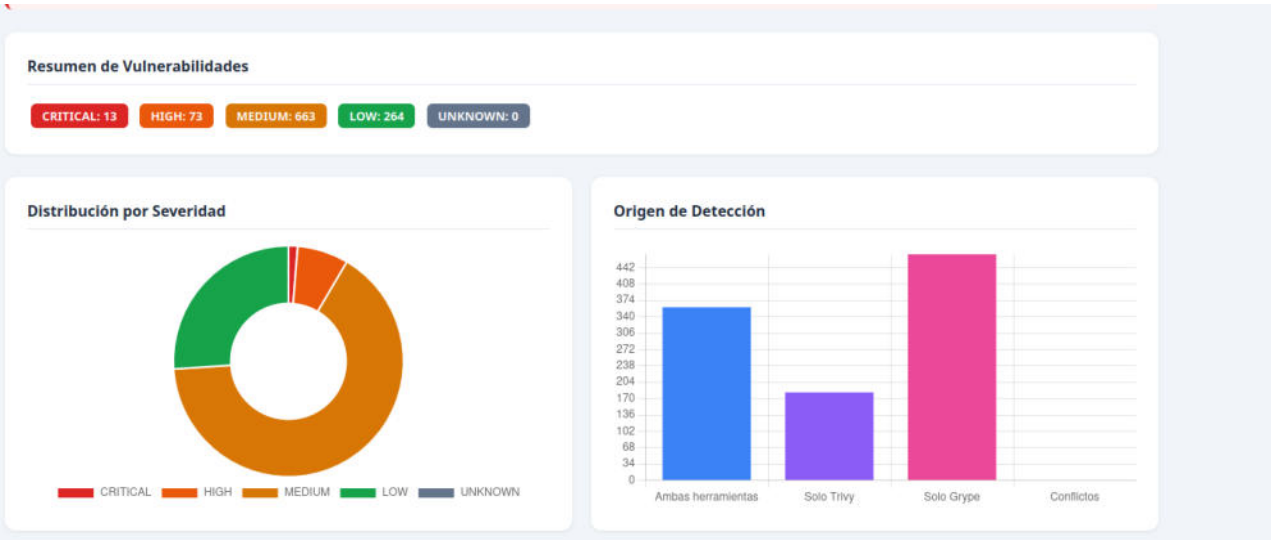
Imagen	CRIT.	HIGH	MED.	LOW	Score	Veredicto
rica-api:prod-v1.0.0	13	73	663	264	0,0	FAIL
salert-notifier:prod-v1.3.0	5	64	512	224	0,0	FAIL
sophon-ops:prod-v0.1.0	6	31	38	31	0,0	FAIL
Total / promedio	24	168	1213	519	0,0	3/3 FAIL

Nota. Elaboración propia, a partir de los reportes JSON y HTML generados por el prototipo (Figuras 2 a 4). Las tres imágenes fueron bloqueadas por el security gate bajo la política de producción al superar simultáneamente los umbrales de vulnerabilidades críticas (máx. permitido: 0), altas (máx. permitido: 3) y score mínimo (80,0).

Las tres imágenes fueron bloqueadas (veredicto FAIL) por el security gate, en los tres casos por las mismas tres razones combinadas: presencia de vulnerabilidades CRITICAL (máximo permitido en producción: 0), un número de vulnerabilidades HIGH muy superior al máximo permitido (3) y un score de 0,0 muy por debajo del mínimo requerido (80,0).

Adicionalmente, en salert-notifier el sistema reportó dos criterios de bloqueo directo adicionales: una vulnerabilidad CRITICAL/HIGH sin corrección disponible y un CVSS máximo de 10,0, que supera el umbral de bloqueo (9,0); en sophon-ops se sumaron otros dos criterios adicionales: 17 vulnerabilidades CRITICAL/HIGH sin corrección disponible y un CVSS máximo de 9,8.

Figura 2
Distribución por severidad y origen de detección rica-api:prod-v1.0.0



La Tabla 16 reporta las estadísticas de correlación interescáner para las tres imágenes privadas, siguiendo el mismo esquema empleado en la sección 4.2.1 para el corpus público.

Tabla 16
Estadísticas de correlación interescáner imágenes de repositorios privados

Imagen	Total	Ambas	Solo Trivy	Solo Grype	Conflictos	Jaccard global
rica-api	1 013	368	183	470	0	0,3633
salert-notifier	805	349	149	307	7	0,4335

Imagen	Total	Ambas	Solo Trivy	Solo Grype	Conflictos	Jaccard global
sophon-ops	108	104	3	1	18	0,9630

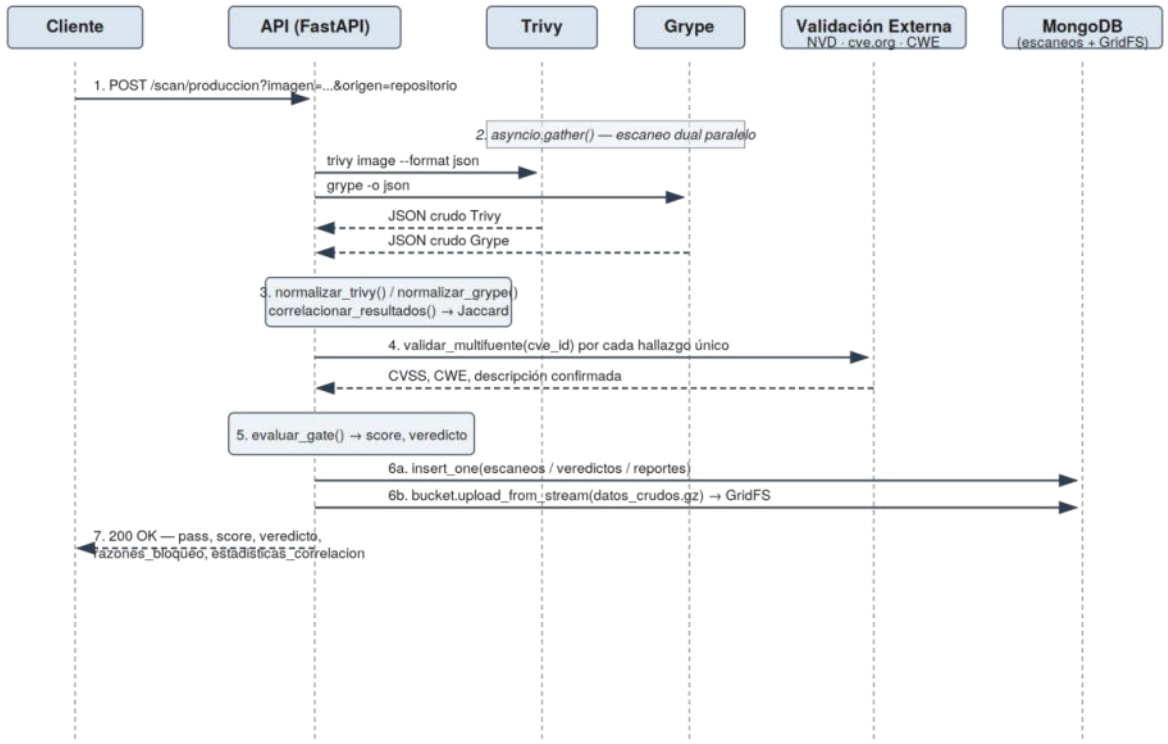
Nota. Elaboración propia. Los valores de rica-api y salert-notifier provienen del campo estadisticas_correlacion de la respuesta JSON del prototipo (Figura 3).

El índice de Jaccard global de las tres imágenes privadas muestra un comportamiento heterogéneo (0,3633 en rica-api; 0,4335 en salert-notifier; 0,9630 en sophon-ops). En rica-api y salert-notifier se reproduce el patrón ya documentado en la sección 4.2.1: un Jaccard por debajo del promedio del corpus público (0,4727) y un volumen de hallazgos exclusivos de Grype entre 2,06 y 2,57 veces mayor que los de Trivy, posiblemente asociado a la composición de dependencias Java (Maven/Gradle) observada en las acciones de remediación sugeridas por el prototipo, donde predominan paquetes como org.apache.tomcat.embed:tomcat-embed-core y org.springframework.security:spring-security-core. sophon-ops, en cambio, constituye un caso atípico dentro del conjunto privado: con solo 108 hallazgos totales frente a los más de 600 de las otras dos imágenes, y una consistencia interescáner de 0,9630 (104 de 108 hallazgos detectados por ambos motores), evidencia que ante una superficie de vulnerabilidades más reducida la probabilidad de discrepancia entre Trivy y Grype disminuye sustancialmente; de hecho, es la única de las tres imágenes privadas donde Trivy reporta más hallazgos exclusivos que Grype (3 frente a 1), en sentido opuesto al patrón dominante del resto del corpus.

Las Figuras 3 y 4 muestran, respectivamente, la evidencia de integración vía API (diagrama de secuencia de la interacción completa para salert-notifier, con el resumen de la respuesta JSON) y el reporte visual generado para sophon-ops, evidenciando que el prototipo opera de forma idéntica frente a un origen=repositorio autenticado contra un registry privado que frente a una imagen pública de Docker Hub.

Figura 3
Diagrama de secuencia de la interacción cliente–API–motores de escaneo–MongoDB para

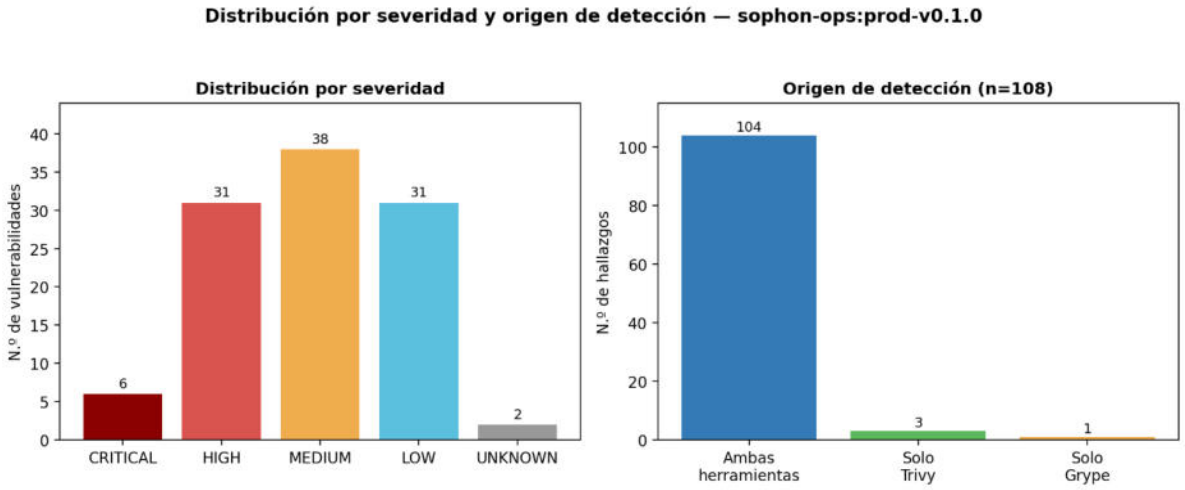
salert-notifier:prod-v1.3.0




```
{
  "pass": false,
  "exit_code": 1,
  "score": 0.0,
  "veredicto": "FAIL",
  "razones_bloqueo": [
    "Críticas: 5 (máx. permitido: 0)",
    "Altas: 64 (máx. permitido: 3)",
    "Score 0.0 por debajo del mínimo requerido (80.0)",
    "1 vulnerabilidad CRITICAL/HIGH sin corrección disponible",
    "CVSS máximo 10.0 supera el umbral de bloqueo (9.0)"
  ],
  "resumen_vulnerabilidades": {
    "CRITICAL": 5,
    "HIGH": 64,
    "MEDIUM": 512,
    "LOW": 224
  },
  "estadisticas_correlacion": {
    "total": 805,
    "detectadas_ambas": 349,
    "solo_trivy": 149,
    "solo_grype": 307,
    "conflictos_severidad": 7,
    "detalle_conflictos_omitido": "7 elementos, ver Anexo A, Tabla A1"
  },
  "ruta_reporte": "/tmp/salert-notifier_prod-v1.3.0.json"
}
```


Figura 4

Distribución por severidad y origen de detección sophon-ops:prod-v0.1.0



Este resultado complementario es metodológicamente relevante por dos razones. Primero, demuestra que el prototipo es funcionalmente agnóstico al origen de la imagen: el mismo pipeline de adquisición, escaneo dual, correlación, validación externa multifuente y decisión del gate operó sin modificaciones sobre imágenes autenticadas de un registry privado de GitLab, validando empíricamente el componente de repositorio privado declarado en el alcance. Segundo, evidencia el efecto de la política de ambiente sobre el veredicto final: las mismas cargas de vulnerabilidades que en el corpus público (bajo política desarrollo, con umbrales más permisivos) habrían producido en varios casos un veredicto FAIL de igual manera dada la magnitud de los hallazgos, pero bajo la política de producción el bloqueo es inmediato y se activa por múltiples criterios simultáneos, ilustrando en un caso real la función de control de despliegue que la Tabla 4 define para el ambiente más crítico.

Dado que estas tres imágenes no forman parte del diseño experimental original (no se midió un ground truth de CVE plantados sobre ellas ni se sometieron a las cuatro categorías de riesgo A-D en su forma completa), sus resultados se reportan como evidencia complementaria de aplicabilidad práctica y no se incorporan a las métricas de Recall, Coverage o Precisión por existencia calculadas en las secciones 4.2.2 y 4.2.3, que permanecen calculadas exclusivamente sobre el corpus experimental de 51 imágenes

públicas.

4.2.6. Rendimiento Temporal

El tiempo de análisis por imagen promedió 116,4 s (1,94 min), con un máximo de 262,6 s (4,38 min) en ruby:2.6, la imagen de mayor carga (14 852 vulnerabilidades). El valor promedio se sitúa por debajo del umbral objetivo de la Tabla 5 (≤ 3 min) y todos los tiempos por imagen quedan por debajo del umbral mínimo (≤ 5 min). Es relevante destacar que estas cifras se obtuvieron en un entorno deliberadamente modesto una máquina virtual de 2 vCPU compartidas y 4 GB de RAM, por lo que constituyen una cota conservadora: en hardware de mayor capacidad los tiempos disminuirían. El tiempo medido corresponde a la fase de evaluación completa (escaneo dual, correlación, validación externa multifuente y decisión); la descarga de la imagen (docker pull) se excluye por ser dependiente de la red e infraestructura. La validación externa multifuente, sujeta a límites de tasa de las APIs públicas, es el componente que más incide en los tiempos de las imágenes con mayor número de hallazgos.

Esta observación se confirma en el conjunto de imágenes privadas: salert-notifier, con 805 vulnerabilidades totales y 261 discrepancias de validación externa detectadas, registró un tiempo de procesamiento de 7 min 14,04 s (434 s) medido extremo a extremo desde el cliente Postman sustancialmente por encima del máximo de 262,6 s observado en el corpus público. Esta diferencia es atribuible a que la medición de Postman incluye la latencia de autenticación y transferencia de capas desde el registry privado de GitLab, ausente en la medición interna de tiempo de evaluación reportada para el corpus público, por lo que ambos tiempos no son directamente comparables.

4.2.7. Discusión

Los resultados respaldan la hipótesis central del trabajo: un prototipo que integra Trivy y Grype en un pipeline automatizado con security gate detecta de forma fiable las vulnerabilidades conocidas antes del despliegue. La detección perfecta de los catorce CVE

plantados in-scope (Recall y Coverage de 1,000), distribuidos en tres familias icónicas y dos ecosistemas, junto con la alta precisión por existencia (99,43 %), demuestran que el sistema minimiza los falsos negativos el riesgo operacional más crítico sin introducir un volumen apreciable de hallazgos inexistentes. La cobertura de Log4Shell, en particular, evidencia la capacidad del prototipo frente a vulnerabilidades críticas de dependencias de lenguaje, no solo de paquetes del sistema operativo.

El hallazgo más matizado corresponde a la consistencia interescáner. El bajo Jaccard global (0,4727) no refleja un defecto, sino la diferente filosofía de los motores: Trivy, orientado a los avisos de las distribuciones, reporta un conjunto curado y casi siempre accionable, hasta el punto de no reportar nada en distribuciones en fin de vida como ubuntu:16.04 y ubuntu:18.04; Grype, apoyado en NVD/CPE, incorpora en esas mismas imágenes 448 y 151 vulnerabilidades sin parche, respectivamente. Esta divergencia confirma empíricamente que ninguna herramienta individual es suficiente y valida la decisión de diseño del doble escaneo.

Al medir la consistencia sobre el universo accionable, donde ambos motores son comparables, el índice alcanza 0,6534: supera el umbral mínimo de aceptación (≥0,65) pero no llega al umbral objetivo (≥0,75). Esta brecha tiene una implicación operativa concreta más allá del número: en un pipeline que dependiera únicamente de Trivy, una imagen basada en ubuntu:16.04 reportaría cero vulnerabilidades y pasaría el security gate sin observaciones, transmitiendo una falsa sensación de seguridad precisamente en el tipo de imagen sin soporte activo del proveedor donde el riesgo real es mayor. El bajo Jaccard no es entonces un simple déficit métrico, sino la evidencia cuantitativa de ese punto ciego, y refuerza el argumento de que la arquitectura de doble escaneo no es una redundancia sino un requisito de cobertura. Adicionalmente, este patrón sugiere una señal operativa aprovechable en producción: un Jaccard bajo, combinado con un volumen de hallazgos concentrado casi en su totalidad en

solo_grype, puede emplearse como indicador indirecto de que una imagen corresponde a una distribución obsoleta o sin mantenimiento activo, y por tanto como criterio adicional de priorización de remediación, más allá del conteo de CVEs por severidad.

En las tres imágenes de repositorios privados (sección 4.2.5) este mismo mecanismo se observa de forma heterogénea: en rica-api y salert-notifier el patrón de divergencia se replica con una magnitud similar o mayor a la del corpus público, mientras que en sophon-ops una imagen con una superficie de vulnerabilidades sustancialmente menor (108 hallazgos totales) y previsiblemente mejor mantenida la consistencia interescáner asciende a 0,9630. Esto refuerza, desde un caso de uso productivo real, la relación inversa entre el tamaño de la superficie de vulnerabilidades de una imagen y el grado de divergencia entre escáneres: a menor deuda técnica de la imagen, menor espacio para que los motores discrepen.

La validación complementaria sobre repositorios privados aporta dos elementos adicionales a la discusión. Por un lado, confirma que el prototipo opera de forma idéntica ante un origen autenticado de tipo repositorio privado, sin necesidad de adaptaciones, lo que cierra la brecha de cobertura experimental señalada en el alcance del estudio respecto del componente privado. Por otro lado, ilustra de forma concreta el efecto de la política de ambiente sobre el veredicto: las mismas tres imágenes, evaluadas bajo la política más estricta de producción, fueron bloqueadas de forma unánime y por múltiples criterios simultáneos, evidenciando en un escenario realista la función de control de despliegue que el prototipo está diseñado a cumplir antes de que un servicio con vulnerabilidades críticas alcance el ambiente productivo.

El estudio reconoce sus límites. Las métricas Precision, Accuracy y TFP no son medibles con un ground truth de CVE plantados; su evaluación rigurosa exigiría un oráculo completo de componentes por imagen, propuesto como trabajo futuro. El alcance del análisis estático deja fuera el código vendorizado y los componentes no empaquetados, documentados

como limitación de cobertura. Los resultados sobre repositorios privados, al no formar parte del diseño experimental original, se reportan como evidencia complementaria de aplicabilidad y no se incorporan al cálculo de las métricas formales de la Tabla 13. Finalmente, los tiempos se obtuvieron en una máquina virtual modesta de 2 vCPU; aunque cumplen el umbral, dependen del hardware y de la latencia de las APIs externas de validación.

4.2.8. Síntesis del Capítulo

Este capítulo presentó los resultados de la validación experimental del prototipo DockerSec Gate sobre 51 imágenes. El sistema las procesó sin fallos, discriminó correctamente los perfiles de riesgo mediante el *security gate*, que aprobó solo las imágenes mínimas y parchadas, y alcanzó una tasa de detección y una cobertura de 1,000 sobre los catorce CVE plantados *in-scope* en tres familias y dos ecosistemas, incluido Log4Shell, con una precisión por existencia del 99,43 % y un tiempo medio de análisis de 1,94 min en hardware modesto. La consistencia interescáner accionable (0,6534) superó el umbral mínimo y evidenció la complementariedad de Trivy y Grype. En la evaluación integral frente a la Tabla 5, el prototipo superó el mínimo en todas las métricas evaluables y el objetivo en el 75 % de ellas, por lo que se considera validado, con las limitaciones metodológicas declaradas de forma transparente. Las conclusiones y las líneas de trabajo futuro se desarrollan en el Capítulo 5.

Capítulo 5

5. CONCLUSIONES Y RECOMENDACIONES

La presente investigación tuvo como propósito diseñar y validar un prototipo automatizado para la detección de vulnerabilidades en imágenes Docker provenientes de repositorios públicos y privados. A partir del desarrollo realizado y de la evaluación experimental efectuada, se establecen las conclusiones y recomendaciones que se detallan a continuación.

5.1.Conclusiones

1. Se cumplió el objetivo de diseñar la arquitectura lógica y tecnológica del sistema mediante la implementación de un pipeline de seguridad organizado por capas, el cual integra procesos de adquisición, escaneo, normalización, correlación, validación externa, evaluación de confianza y generación de reportes. Esta arquitectura proporciona un proceso estructurado que facilita la automatización del análisis de imágenes Docker y su incorporación en entornos DevSecOps.

2. Se logró construir un prototipo funcional denominado DockerSec Gate, el cual integra los motores de análisis Trivy y Grype para realizar el escaneo automatizado de imágenes Docker. La combinación de ambas herramientas permitió complementar la detección de vulnerabilidades, reduciendo la dependencia de un único motor de análisis y ofreciendo una evaluación más consistente de los riesgos identificados. El análisis de complementariedad mediante el índice de Jaccard confirmó que cada motor aporta hallazgos diferenciados Trivy con un conjunto curado y accionable, y Grype con un mayor volumen de vulnerabilidades sin parche, lo que valida empíricamente la decisión de diseño del doble escaneo.

La evaluación experimental permitió verificar el funcionamiento del prototipo mediante métricas objetivas previamente definidas. El sistema detectó la totalidad de las

vulnerabilidades conocidas presentes en el corpus de prueba (Recall y Cobertura de 1,000) y, al contrastarse con los criterios de aceptación de la Tabla 5, superó el umbral mínimo en las cuatro métricas evaluables y el umbral objetivo en el 75 % de ellas, por lo que el prototipo se considera validado. Estos resultados, obtenidos dentro del alcance del análisis estático de composición de software (*Software Composition Analysis*, SCA), evidencian quela metodología propuesta constituye una alternativa viable para fortalecer los controles de seguridad antes del despliegue de contenedores.

3. La generación automatizada de reportes técnicos clasificados según la severidad CVSS facilita la priorización de las vulnerabilidades detectadas y mejora la toma de decisiones durante el proceso de remediación. Asimismo, la incorporación del mecanismo Security Gate permite establecer criterios objetivos para aceptar o bloquear imágenes Docker de acuerdo con el nivel de riesgo definido por la organización.

4. Durante el desarrollo de la investigación también se identificaron las limitaciones propias del análisis estático. Determinadas vulnerabilidades presentes en componentes no observables por herramientas SCA, como código incorporado manualmente o complementos no empaquetados, quedan fuera del alcance del prototipo. Esta situación no representa una deficiencia del sistema desarrollado, sino una restricción inherente a la tecnología utilizada, por lo que futuras investigaciones podrán complementar este enfoque mediante técnicas adicionales de análisis.

Finalmente, se concluye que el prototipo desarrollado cumple con el objetivo general de la investigación, al demostrar que la automatización del análisis de vulnerabilidades en imágenes Docker contribuye a fortalecer la seguridad de la cadena de suministro de software, disminuyendo la probabilidad de desplegar contenedores con vulnerabilidades conocidas y favoreciendo la adopción de prácticas de seguridad desde etapas tempranas del ciclo de desarrollo.

5.2.Recomendaciones

Con base en los resultados obtenidos durante la investigación, se plantean las siguientes recomendaciones:

1. Integrar DockerSec Gate dentro de pipelines de integración y entrega continua (CI/CD), permitiendo que el análisis de vulnerabilidades forme parte del proceso automático de construcción y despliegue de aplicaciones.

2. Incorporar en futuras versiones la generación automática de SBOM (Software Bill of Materials) y la integración con VEX (Vulnerability Exploitability eXchange), con el propósito de mejorar la trazabilidad de componentes y optimizar la gestión de vulnerabilidades.

3. Ampliar el conjunto de imágenes utilizadas durante la validación experimental, incluyendo aplicaciones empresariales de diferentes tecnologías y sistemas operativos, con el fin de evaluar el comportamiento del prototipo en escenarios de mayor complejidad.

4. Complementar el análisis implementado con herramientas orientadas a detectar configuraciones inseguras, exposición de credenciales, errores de configuración y vulnerabilidades en tiempo de ejecución, logrando una evaluación integral de la seguridad de los contenedores.

5. Mantener actualizadas las bases de datos de vulnerabilidades y las versiones de los motores de escaneo utilizados, garantizando que el prototipo continúe identificando amenazas emergentes conforme evolucionan los catálogos de CVE.

6. Referencias

Anchore. (2024). *Grype: A vulnerability scanner for container images and filesystems (v0.87) [Software]* . Obtenido de GitHub: https://github.com/anchore/grype

Aqua Security. (2024). *Trivy: Vulnerability scanner for containers and other artifacts (v0.58) [Software]* . Obtenido de GitHub : https://github.com/aquasecurity/trivy

Balliu, M., Massacci, F., & Or-Meir, O. (2023). SBOM where are we now? . *IEEE Security & Privacy , 21*(2), 58-65. doi:https://doi.org/10.1109/MSEC.2022.3228994

Bernstein, D. (2014). Containers and cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, 81-84. doi:10.1109/MCC.2014.51

Bhardwaj, P. D. (2024). Securing Container Images through Automated Vulnerability Detection in Shift-Left CI/CD Pipelines. *Journal of Networking*, 9. Obtenido de https://doi.org/10.58496/BJN/2024/016

Center for Internet Security . (2023). *CIS Docker benchmark (v1.6.0)* . Obtenido de Center for Internet Security : https://www.cisecurity.org/benchmark/docker

Churakova, Y., Ekstedt, M., & Schmid, L. (Marzo de 2026). Vexed by VEX tools: Consistency evaluation of container vulnerability scanners. Estocolmo, Suecia: Springer Nature. doi:10.1007/978-3-032-20018-1_8

Colvin, S. et al. (2024). *Pydantic: Data validation using Python type hints (v2) [Software]*. Obtenido de GitHub: https://github.com/pydantic/pydantic

Docker Inc. (2024). *Docker Compose: Define and run multi-container applications [Software]*. Obtenido de Docker Docs: https://docs.docker.com/compose/

Encode. (2024). *HTTPX: A next-generation HTTP client for Python [Software]*. Obtenido de python-httpx.org: https://www.python-httpx.org

FIRST.org. (2019). *Common vulnerability scoring system v3.1: Specification document* . Obtenido de FIRST.org: https://www.first.org/cvss/specification-document

GitLab. (2024). *GitLab*. Obtenido de The 2024 global DevSecOps report: https://about.gitlab.com/developer-survey/

Haque, M., & Babar, A. (Diciembre de 2021). Well begun is half done: An empirical study of exploitability & impact of base-image vulnerabilities. doi:10.48550/arXiv.2112.12597

HATS at LPC. (2026). *HATS at LPC*. Obtenido de Docker/Apptainer HATS@LPC: https://awesome-workshop.github.io/docker-singularity-hats/aio/index.html

Javed, O., & Toor, S. (2021). An evaluation of container security vulnerability detection tools. *ICCBDC 2021* (págs. 95–101). Association for Computing Machinery. doi:10.1145/3481646.3481661

Kaur, B., Dugré, M., Hanna, A., & Glatard, T. (Enero de 2021). An analysis of security vulnerabilities in container images for scientific data analysis. *Gigascience*, 7. doi:10.1093/gigascience/giab025

Khalil, M. (2025). *Vulnerability statistics 2025: Record CVEs, zero-days & exploits.* Obtenido de DeepStrike Security Research.: https://deepstrike.io/blog/vulnerability-statistics-2025

Ladisa, P., Plate, H., Martinez, M., & Bartel, A. (2023). SoK: Attacks on the supply chain of open-source software . *IEEE Transactions on Software Engineering , 49*(9), 4293-4313. doi:https://doi.org/10.1109/TSE.2023.3321559

Mahale, A., Shinde, T., & Shewale, S. (Abril de 2026). AI-driven automated security assessment pipeline for modern DevSecOps. *International Journal of Creative Research Thoughts (IJCRT)*, 9. Obtenido de http://www.ijcrt.org/papers/IJCRT26A4262.pdf

Mendonsa, A. N. (25 de Enero de 2025). *Comparative evaluation of open-source vulnerability assessment scanning tools: Trivy and Grype vs AWS ECR.* Obtenido de National College of Ireland Repository: https://norma.ncirl.ie/8122/1/alricnestormendonsa.pdf

Merkel, D. (01 de March de 2014). Docker: lightweight Linux containers for consistent development and deployment. *Linux Journal, 2014*, 2. doi:https://dl.acm.org/doi/10.5555/2600239.2600241

Mirtaheri, S. L. (2025). *Bridging security gaps: DevSecOps practices in container environments.* Karlskrona: DiVA Portal – Diva2. Obtenido de https://www.diva-

portal.org/smash/get/diva2:2036264/FULLTEXT01.pdf
MongoDb inc. (2024). *MongoDB: The developer data platform (v7.0) [Software]* . Obtenido de MongoDB : https://www.mongodb.com
National Institute of Standards and Technology . (2024). *U.S. Department of Commerce* . Obtenido de National Vulnerability Database: https://nvd.nist.gov
Open Container Initiative. (2021). *OCI image format specification*. Obtenido de Open Container Initiative : https://github.com/opencontainers/image-spec/blob/main/spec.md
Pahl, C., Brogi, A., Soldani, J., & Jamshidi, P. (2019). Cloud container technologies: A state-of-the-art review . *IEEE Transactions on Cloud Computing , 7*, 677-692. doi:https://doi.org/10.1109/TCC.2017.2702586
Pallets. (2024). *Jinja: A fast, expressive, extensible templating engine [Software]*. Obtenido de Pallets Projects: https://jinja.palletsprojects.com
Pecka, A., Karpíšek, F., & Ráb, J. (2023). Security testing in CI/CD pipelines . *Proceedings of the International Conference on Software Engineering Advances* , 112-119. doi:https://doi.org/10.1007/978-3-031-21333-5_11
Python Software Foundation. (2024). *Python: A dynamic, open source programming language [Software]*. Obtenido de Python.org: https://www.python.org
Rajapakse, R. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology*. doi:https://doi.org/10.1016/j.infsof.2021.106700
Ramírez, S. (2024). *FastAPI: Modern, fast web framework for building APIs with Python [Software]*. Obtenido de GitHub: https://github.com/fastapi/fastapi
Redis Ltd. (2024). *Redis: In-memory data structure store [Software]*. Obtenido de redis.io: https://redis.io
ReversingLabs. (2025). *The 2025 software supply chain security report.* Cambridge: ReversingLabs Spectra Platform Reports. Obtenido de https://ntsc.org/wp-content/uploads/2025/03/The-2025-Software-Supply-Chain-Security-Report-RL-compressed.pdf
Snyk. (2024). *The state of open source security 2024* . Obtenido de Snyk: https://snyk.io/reports/open-source-security/
Souppaya, M., Morello, J., & Scarfone, K. (2017). Application container security guide. *National Institute of Standards and Technology*, 63. doi:10.6028/NIST.SP.800-190
Sultan, S., Ahmad, I., & Dimitriou, T. (2019). Container Security: Issues, Challenges, and the Road Ahead. *IEEE Access, 7*, 52976-52996. doi:https://doi.org/10.1109/ACCESS.2019.2911732
Sysdig. (2023). *Sysdig Report Finds that 87% of Container Images Have High Risk Vulnerabilities.* San Francisco: Sysdig Inc. Obtenido de https://sysdig.com/press-releases/sysdig-2023-usage-report
Tak, B. C., Iyer, S., & Mohindra, A. (2018). Strengthening container security with isolation and access controls. *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)* , 95-100. doi:https://doi.org/10.1109/IC2E.2018.00025
Tunde-Onadele, O., He, J., Dai, T., & Gu, X. (2019). A study on container vulnerability exploit detection. *2019 IEEE International Conference on Cloud Engineering (IC2E)* (págs. 121–127). Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/IC2E.2019.00026
Wist, K., Helsem, M., & Gligoroski, D. (2021). Vulnerability analysis of 2500 Docker Hub images. *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER).* doi:https://doi.org/10.48550/arXiv.2006.02932
Zerouali, A., Mens, T., Robles, G., & Gonzalez Barahona, J. M. (2022). On the impact of outdated and vulnerable Javascript packages in Docker Hub . *Journal of Software: Evolution and Process , 34*(4), e2377. doi:https://doi.org/10.1002/smr.2377

ANEXO A

Detalle de Conflictos de Severidad Interescáner en Repositorios Privados

Tabla A1

Conflictos de severidad resueltos — salert-notifier:prod-v1.3.0

CVE	Severidad Trivy	Severidad Grype	Severidad Resuelta	CVSS usado	Herramienta CVSS
CVE-2026-25681	HIGH	MEDIUM	HIGH	8,1	Trivy
CVE-2026-27136	HIGH	MEDIUM	HIGH	8,1	Trivy
CVE-2026-39824	UNKNOWN	LOW	LOW	3,3	Grype
CVE-2026-39822	UNKNOWN	HIGH	HIGH	7,8	Grype
CVE-2026-39822 *	UNKNOWN	HIGH	HIGH	7,8	Grype
CVE-2026-42505	UNKNOWN	MEDIUM	MEDIUM	5,3	Grype
CVE-2026-42505 *	UNKNOWN	MEDIUM	MEDIUM	5,3	Grype

Nota. Elaboración propia, a partir del campo estadisticas_correlacion.detalle_conflictos de la respuesta JSON del prototipo para salert-notifier:prod-v1.3.0 (ver Tabla 16). La columna "Severidad resuelta" corresponde a la regla de máxima severidad aplicada por el correlacionador, descrita en la sección 3.3.3 (Capa de Correlación). *CVE-2026-39822 y CVE-2026-42505 aparecen duplicados de forma idéntica en la respuesta del prototipo, probablemente por afectar a dos componentes distintos dentro de la imagen; pendiente de verificación en el módulo de correlación

ANEXO B

Repositorio de Código Fuente del Prototipo

El código fuente completo de DockerSec Gate se encuentra disponible en: https://github.com/fernandoJoseC/proyectoMaestria

La versión evaluada en los experimentos reportados en este documento corresponde al commit f41bb9a (rama main), previo a la corrección de auditabilidad descrita en la sección 3.3.8. El repositorio está organizado en las siguientes capas: adquisición de imágenes (app/adquisicion), escaneo dual (app/escaneo), normalización (app/normalizacion), correlación (app/analisis), validación externa (app/validacion_externa), persistencia (app/db) y generación de reportes (app/reportes).