Automatización de la seguridad en el desarrollo del SW

Jordi Nogués López

Máster en Ciberseguridad y Privacidad

Área de Análisis de Datos

Tutor
Joan Caparrós Ramírez

Profesor
Pau Perea Paños

Fecha Entrega
7 de enero de 2025

Universitat Oberta de Catalunya

A mi esposa Ana, por su apoyo y paciencia a lo largo de este Máster. Sin su aliento constante, este logro no habría sido posible.

A mi hija Anna, por entender mis ausencias en sus competiciones de rítmica. Su comprensión es un reflejo de su madurez y cariño.

A Joan Caparrós, mi tutor en este proyecto, cuya guía experta y dedicación han sido clave para desarrollar un trabajo que espero sea de utilidad para estudiantes y profesionales de la ciberseguridad.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Automatización de la seguridad en el desarrollo del Software*
Nombre del autor:	*Jordi Nogués López*
Nombre del consultor/a:	*Joan Caparrós Ramírez*
Nombre del PRA:	*Pau Perea Paños*
Fecha de entrega (mm/aaaa):	01/2025
Titulación:	*Máster Universitario en Ciberseguridad y Privacidad*
Área del Trabajo Final:	*Análisis de datos*
Idioma del trabajo:	*Castellano*
Palabras clave	*Ciberseguridad, DevSecOps, SAST, SCA*

Resumen del Trabajo:

Este Trabajo Fin de Máster aborda la integración de herramientas de análisis de seguridad de aplicaciones en un *pipeline* automatizado *DevSecOps*, con el objetivo de garantizar la identificación y mitigación temprana de vulnerabilidades durante el ciclo de desarrollo de software. El proyecto se centra en herramientas SAST y SCA, configuradas en un entorno CI/CD mediante GitHub Actions.

La memoria documenta las etapas clave: investigación, selección de herramientas, diseño del entorno de laboratorio y configuración del pipeline. Se detalla cómo estas herramientas automatizan pruebas específicas en puntos críticos del flujo de desarrollo, mejorando la eficiencia en la detección de vulnerabilidades y facilitando la priorización de remediaciones.

Adicionalmente, se realiza un análisis introductorio del *OWASP Benchmark Project*, y se identifican posibles líneas de evolución, como la integración de herramientas DAST e IAST. Entre las lecciones aprendidas, destacan la relevancia de la automatización, la importancia de adaptar las herramientas al flujo de trabajo del desarrollador y la necesidad de priorizar vulnerabilidades según el riesgo.

Este proyecto contribuye al entendimiento práctico de *DevSecOps* y proporciona una base replicable para integrar seguridad en entornos de desarrollo modernos.

Abstract:

This Master's Thesis addresses the integration of application security testing tools into an automated *DevSecOps pipeline*, with the goal of ensuring the early identification and mitigation of vulnerabilities during the software development lifecycle. The project focuses on SAST and SCA tools, configured within a CI/CD environment using GitHub Actions.

The thesis documents the key stages: research, tool selection, laboratory environment design, and pipeline configuration. It details how these tools automate specific tests at

critical points in the development workflow, improving the efficiency of vulnerability detection and facilitating the prioritization of remediations.

Additionally, an introductory analysis of the OWASP Benchmark Project is conducted, and potential lines of evolution are identified, such as the integration of DAST and IAST tools. Among the lessons learned, the importance of automation, the adaptation of tools to the developer's workflow, and the prioritization of vulnerabilities based on risk stand out.

This project contributes to the practical understanding of *DevSecOps* and provides a replicable foundation for integrating security into modern development environments.

Índice

1. INTRODUCCIÓN 1

1.1 CONTEXTO Y JUSTIFICACIÓN DEL TRABAJO 1
1.2 OBJETIVOS DEL TRABAJO 3
1.2.1. Objetivo general 3
1.2.2. Objetivos específicos 3
1.3 ENFOQUE Y MÉTODO SEGUIDO 4
1.3.1. Evaluación de metodologías 4
1.3.2. Metodología seleccionada 5
1.4 PLAN DE TRABAJO 6
1.4.1. Listado de tareas a realizar 6
1.4.2. Planificación de las tareas 8
1.4.3. Diagrama de Gantt 9
1.5 BREVE REVISIÓN DEL ESTADO DEL ARTE 10
1.5.1. Visión general 10
1.5.2. Líderes del mercado 13
1.5.3. La Inteligencia artificial aplicada al ámbito AST 15
1.6 LISTADO DE POSIBLES RIESGOS Y SOLUCIONES 16
1.7 LISTADO DE COSTES ASOCIADOS AL MATERIAL 17
1.8 IMPLICACIONES LEGALES O ÉTICAS 18
1.8.1. Sostenibilidad 18
1.8.2. Comportamiento Ético y Responsabilidad Social 18
1.8.3. Diversidad, Género y Derechos Humanos 19

2. FASE DE INVESTIGACIÓN 20

2.1 INTRODUCCIÓN 20
2.2 FUNDAMENTOS DE SEGURIDAD EN EL DESARROLLO DEL SOFTWARE 20
2.2.1. Modelado de amenazas 20
2.2.2. Integración temprana de la seguridad en el ciclo de desarrollo 25
2.2.3. Conclusiones 27
2.3 HERRAMIENTAS DE LA GUÍA OWASP 28
2.3.1. Introducción 28
2.3.2. Análisis estático de seguridad 29
2.3.3. Test Interactivo (IAST) 32
2.3.4. Test Dinámico de la Aplicación (DAST) 34
2.3.5. Conclusiones 35
2.4 EVALUACIÓN DE LAS HERRAMIENTAS 36
2.4.1. Casos de uso 36
2.4.2. Criterios de evaluación 38
2.4.3. Ranking de fabricantes 39
2.5 HERRAMIENTAS SELECCIONADAS 42
2.5.1. Selección de la plataforma DevSecOps 43
2.5.2. Selección de herramientas AST para el entorno empresarial 45
2.5.3. Selección de herramientas AST para el entorno de laboratorio 47
2.6 DESCRIPCIÓN DEL ENTORNO DE LABORATORIO 49
2.6.1. Entorno de desarrollo y ejecución 49
2.6.2. Pipeline CI/CD 50

3. FASE DE IMPLEMENTACIÓN 51

3.1 INTRODUCCIÓN 51
3.2 CONFIGURACIÓN DE LA PLATAFORMA *DEVSECOPS* 52
3.2.1. Selección de la aplicación objetivo 52
3.2.2. Creación del repositorio en GitHub 53

3.2.3. Configuración del entorno local de desarrollo y ejecución ... 53
3.3 Pruebas SAST ... 54
3.3.1. Preparación de la herramienta ... 54
3.3.2. Ejecución del escaneo SAST ... 54
3.3.3. Análisis de las vulnerabilidades detectadas en el escaneo SAST ... 57
3.3.4. Remediación de vulnerabilidades con GitHub Copilot Autofix ... 57
3.4 Pruebas SCA ... 58
3.4.1. Preparación de la herramienta ... 58
3.4.2. Ejecución del escaneo SCA ... 59
3.4.3. Análisis de las vulnerabilidades detectadas en el escaneo SCA ... 61
3.4.4. Remediación de las vulnerabilidades SCA ... 62
3.5 Integración de otras herramientas en GitHub Actions ... 64
3.5.1. Escaneo gestionado desde la herramienta AST ... 64
3.5.2. Escaneo integrado en el pipeline CI/CD ... 64
3.6 Evaluación de impacto tras la implementación ... 66
3.6.1. Mejora en la detección de vulnerabilidades ... 66
3.6.2. Reducción de tiempos de desarrollo y mantenimiento ... 66
3.6.3. Incremento en la estabilidad del sistema ... 67
3.6.4. Experiencia del desarrollador ... 67
3.6.5. Conclusión ... 67

4. CONCLUSIONES ... 68

4.1 Introducción ... 68
4.2 Evaluación del cumplimiento de objetivos ... 68
4.3 Lecciones aprendidas ... 68
4.4 Posibles trabajos futuros ... 70

5. GLOSARIO DE TÉRMINOS Y ACRÓNIMOS ... 71

6. BIBLIOGRAFÍA ... 73

7. ANEXOS ... 75

7.1. Matriz de herramientas propuestas por OWASP (por orden alfabético) ... 76
7.2. Matriz de herramientas propuestas por OWASP (por categoría) ... 77
7.3. Plantillas para la integración de herramientas AST en GitHub Actions ... 78
7.4. Introducción al OWASP Benchmark Project ... 83
7.5. Importación del repositorio en GitHub ... 86
7.6. Configuración del entorno local de desarrollo y ejecución ... 88
7.6.1. Configuración del entorno local de desarrollo ... 88
7.6.2. Instalación del entorno de ejecución ... 89
7.7. Configuración de CodeQL ... 91
7.8. Experiencia de usuario en el escaneo SAST ... 93
7.9. Archivo SARIF: Función y estructura ... 95
7.10. Gestión de vulnerabilidades con GitHub ... 97
7.11. Caso de uso de Copilot Autofix ... 100
7.12. Configuración de Dependabot ... 101
7.13. Gestión del *pull-request* en GitHub ... 104
7.14. Listado de vulnerabilidades SCA ... 106
7.15. Remediación automática con Dependabot ... 107
7.16. Escaneo del repositorio GitHub desde la aplicación Semgrep ... 111
7.17. Integración de la plantilla YAML predefinida ... 113
7.18. Creación del token de acceso para la conexión con Semgrep ... 115
7.19. Configuración del archivo de workflow (.yml) ... 117

Lista de ilustraciones

Ilustración 1. Diagrama de Gantt 9
Ilustración 2. Software Assurance Maturity Model. CISSP Official Study Guide (Chapple, 2024) 10
Ilustración 3. Metodología DevSecOps. Fuente: (OWASP Foundation, 2024) 12
Ilustración 4. Magic Quadrant for App Security Testing. Fuente: Gartner (Mark Horvath, 2024) 13
Ilustración 5. Modelo DFD. Fuente "Threat Modeling: Designing for Security" (Shostack, 2014) 21
Ilustración 6. STRIDE, flujo de interacciones (Shostack, 2014) 23
Ilustración 7. Integración de la seguridad en el desarrollo (Rich Mogull, Mike Rothman, 2024) 26
Ilustración 8. Necesidades de seguridad dentro del SDLC (Manjunath Bhat, 2023) 43
Ilustración 9. Entorno de laboratorio. Fuente: Elaboración propia 49
Ilustración 10. Pipeline CI/CD. Fuente: Elaboración propia 50
Ilustración 11. Flujo de trabajo en el escaneo SAST. Fuente: Elaboración propia 55
Ilustración 12. Flujo de trabajo en el escaneo SCA. Fuente: Elaboración propia 60
Ilustración 13. Vulnerabilidad "Command Injection" en una librería. Fuente: Elaboración propia 61
Ilustración 14. Cierre de la alerta detectada por Dependabot. Fuente: Elaboración propia 62
Ilustración 15. Vulnerabilidad "Verification Bypass" en una librería. Fuente: Elaboración propia 63
Ilustración 16. Guía de interpretación de resultados del OWASP Benchmark 84
Ilustración 17. Importación del repositorio mediante GitHub. Fuente: Elaboración propia 86
Ilustración 18. Archivos de workflow en GitHub Actions. Fuente: Elaboración propia 87
Ilustración 19. Configuración de GitHub Desktop. Fuente: Elaboración propia 88
Ilustración 20. Arranque de la aplicación OWASP Juice Shop. Fuente: Elaboración propia 90
Ilustración 21. Página de inicio de la aplicación OWASP Juice Shop. Fuente: Elaboración propia 90
Ilustración 22. Cambio de la visibilidad del repositorio. Fuente: Elaboración propia 91
Ilustración 23. Configuración de CodeQL. Fuente: Elaboración propia 92
Ilustración 24. Commit en el repositorio local. Fuente: Elaboración propia 93
Ilustración 25. Push de los cambios al repositorio remoto. Fuente: Elaboración propia 94
Ilustración 26. Consulta de los resultados del trabajo de CodeQL. Fuente: Elaboración propia 94
Ilustración 27. Exportación del archivo SARIF. Fuente: Elaboración propia 95
Ilustración 28. Estructura del archivo SARIF. Fuente: Elaboración propia 96
Ilustración 29. Listado de vulnerabilidades detectadas por CodeQL. Fuente: Elaboración propia 97
Ilustración 30. Gestión de vulnerabilidades en GitHub. Fuente: Elaboración propia 98
Ilustración 31. CWE-843: Type Confusion (MITRE, 2024) 99
Ilustración 32. Autofix de Copilot para CWE-843. Fuente: Elaboración propia 100
Ilustración 33. Activación del escaneo de Dependabot. Fuente: Elaboración propia 101
Ilustración 34. Gestión del pull-request en GitHub. Fuente: Elaboración propia 104
Ilustración 35. Listado de alertas detectadas por Dependabot. Fuente: Elaboración propia 106
Ilustración 36. Actualización automática de la librería jsonwebtoken. Fuente: Elaboración propia 108
Ilustración 37. Opciones de fusionado en GitHub. Fuente: Elaboración propia 109
Ilustración 38. Página de confirmación de la acción de fusionado. Fuente: Elaboración propia 110
Ilustración 39. Consentimiento de acceso desde Semgrep. Fuente: Elaboración propia 111
Ilustración 40. Activación del escaneo gestionado desde Semgrep. Fuente: Elaboración propia 111
Ilustración 41. Listado de alertas en la consola de Semgrep. Fuente: Elaboración propia 112
Ilustración 42. Integración de una plantilla YAML predefinida. Fuente: Elaboración propia 114
Ilustración 43. Generación del token de acceso en Semgrep. Fuente: Elaboración propia 115
Ilustración 44. Configuración del secreto en GitHub. Fuente: Elaboración propia 116
Ilustración 45. Configuración del archivo de workflow .yml. Fuente: Elaboración propia 118

Lista de tablas

Tabla 1. STRIDE, ejemplos de amenaza por categoría 22
Tabla 2. STRIDE, análisis de interacciones (Shostack, 2014) 24
Tabla 3. Medidas de seguridad proactivas en el SSDLC (Rich Mogull, Mike Rothman, 2024) 26
Tabla 4. Ponderación por casos de uso (Gartner, 2024) 37
Tabla 5. Puntuación de fabricantes por categoría 40
Tabla 6. Puntuación ponderada en el escenario empresarial 40
Tabla 7. Ejemplos de herramientas DevSecOps por funcionalidad (Manjunath Bhat, 2023) 44
Tabla 8. Mejor fabricante por categoría 45
Tabla 9. Casos de prueba por área de vulnerabilidad OWASP 83

1. Introducción

1.1 Contexto y justificación del Trabajo

En la era digital actual, la seguridad del software es una prioridad crítica para las organizaciones de cualquier tamaño. Las vulnerabilidades en el software pueden ser explotadas para llevar a cabo ataques cibernéticos, resultando en daños económicos y reputacionales. La creciente complejidad de las aplicaciones y la rapidez con la que se despliegan nuevas funcionalidades amplifican los riesgos asociados a la seguridad.

La seguridad en el desarrollo de aplicaciones es esencial no solo desde una perspectiva técnica, sino también desde diferentes puntos de vista que afectan a las organizaciones en su conjunto:

- **Perspectiva empresarial**: Las brechas de seguridad pueden provocar pérdidas financieras significativas, afectar la continuidad del negocio y dañar la reputación de la organización.
- **Perspectiva legal y regulatoria**: Las organizaciones deben cumplir con leyes y regulaciones que protegen la privacidad y seguridad de los datos, como el GDPR o la Ley de Protección de Datos. El incumplimiento puede resultar en sanciones severas.
- **Perspectiva del usuario**: Los clientes esperan que sus datos personales y transacciones estén seguros. La confianza del usuario es fundamental para mantener y atraer clientes.
- **Perspectiva operativa**: Las vulnerabilidades pueden llevar a interrupciones del servicio, pérdida de datos y costes adicionales para la recuperación y mitigación de incidentes.

La implementación efectiva de prácticas de verificación del software antes de que sea desplegado en producción es fundamental para prevenir ataques y asegurar la integridad y confidencialidad de la información.

Sin embargo, para muchas organizaciones resulta complejo integrar estas prácticas de verificación de seguridad de manera efectiva en sus procesos de desarrollo y despliegue. Algunos de los principales obstáculos a los que se enfrentan son:

- **Falta de automatización**: La ausencia de procesos automatizados dificulta la integración de pruebas de seguridad en ciclos de desarrollo ágiles y continuos.
- **Falta de estandarización**: El uso de múltiples herramientas no integradas puede generar inconsistencias y complicar el análisis de resultados.
- **Desconexión entre equipos**: La falta de comunicación y colaboración entre los equipos de desarrollo, operaciones y seguridad puede conducir a retrasos en la detección y corrección de vulnerabilidades.

Estas situaciones pueden resultar en una detección tardía de vulnerabilidades y en un incremento en los costes y esfuerzos necesarios para corregirlas, impactando negativamente en los plazos de entrega y la calidad del software.

Por tanto, es necesario desarrollar enfoques que permitan integrar de manera eficiente y automatizada las herramientas de verificación de seguridad dentro del ciclo de desarrollo, en línea con las prácticas recomendadas por organizaciones líderes como OWASP[1]**.**

[1] La Open Web Application Security Project (OWASP) es una organización mundial sin ánimo de lucro dedicada a mejorar la seguridad del software. Proporciona recursos gratuitos, como herramientas, documentación y guías, que ayudan a las organizaciones a desarrollar y mantener aplicaciones seguras.

1.2 Objetivos del Trabajo

En este apartado se presentan los objetivos que compondrán este Trabajo Final de Máster. Estos objetivos han sido priorizados en **objetivos primarios**, que constituyen el producto mínimo viable del proyecto, y **objetivos secundarios**, que aportan valor adicional si se dispone de tiempo suficiente. Cada objetivo se considera una tarea EPIC, de la cual derivarán las tareas necesarias para su consecución, detalladas en el listado de tareas.

1.2.1. Objetivo general

El objetivo principal de este proyecto es ofrecer un análisis del panorama de herramientas de Application Security Testing (AST), es decir, herramientas destinadas a probar y garantizar la seguridad de las aplicaciones durante su desarrollo, y demostrar cómo se puede implementar un pipeline *DevSecOps* automatizado que integre dichas herramientas en el ciclo de vida de una aplicación.

Para alcanzar este objetivo general, se han establecido los siguientes objetivos específicos:

1.2.2. Objetivos específicos

Objetivos primarios

Los objetivos primarios que se han identificado en este proyecto son:

1. Estudiar y analizar las distintas herramientas AST mencionadas en la *OWASP DevSecOps Guideline* (OWASP Foundation, 2024).

   Se investigarán las herramientas de seguridad recomendadas por OWASP, comprendiendo sus características y funcionalidades.

2. Seleccionar un conjunto acotado de herramientas para su posterior estudio en profundidad.

   Basándose en el análisis anterior y en los criterios de evaluación de capacidades críticas establecidos por Gartner (Gartner, 2024), se elegirán herramientas relevantes y adecuadas para entornos empresariales.

3. Configurar una plataforma *DevSecOps* que integre las tipologías más críticas de las herramientas de análisis.

   Se implementará un pipeline automatizado que integre las herramientas seleccionadas, facilitando su ejecución en el ciclo de vida del desarrollo.

4. Desarrollar un entorno de prueba representativo.

   Utilizando una aplicación con vulnerabilidades incorporadas intencionadamente para realizar pruebas y validar la eficacia de la integración, y analizar el impacto en la calidad y seguridad del software.

Objetivos secundarios

Los objetivos secundarios son los siguientes:

1. Establecer un marco de evaluación de herramientas, basado en criterios de eficiencia técnica.

   Se definirá un marco que permita evaluar las herramientas AST considerando aspectos como ratios de falsos positivos y negativos, y rendimiento.

2. Incluir la integración de otras tipologías de herramientas de análisis.

   Se ampliará el pipeline para integrar herramientas adicionales, si el tiempo lo permite.

1.3 Enfoque y método seguido

En este apartado se describe el enfoque y la metodología que se seguirán para desarrollar el proyecto. Se ha considerado la aplicación de diferentes metodologías de gestión de proyectos, evaluando sus ventajas y desventajas en el contexto específico de este Trabajo Final de Máster.

1.3.1. Evaluación de metodologías

Para determinar la metodología más adecuada, se han analizado las siguientes opciones:

Metodología en cascada

Se trata de un modelo secuencial en que cada fase del proyecto debe completarse antes de pasar a la siguiente. Las fases típicas incluyen análisis de requisitos, diseño, implementación y estabilización.

- Ventajas:
  - Claridad en los objetivos y entregables de cada fase.
  - Facilidad de planificación y seguimiento.
- Desventajas:
  - Rigidez ante cambios o descubrimientos posteriores.
  - Riesgo de identificar los problemas demasiado tarde.

Metodologías ágiles

Son enfoques iterativos e incrementales que permiten adaptarse a cambios y promover la colaboración continua. Ejemplos incluyen Scrum y Kanban.

- Ventajas:
  - Flexibilidad y capacidad de respuesta ante cambios.
  - Entregas parciales y continuas de valor.

    - Mayor interacción y feedback.
- Desventajas:
    - Requiere experiencia en su aplicación.
    - Puede ser menos predictiva en plazos y recursos.

1.3.2.Metodología seleccionada

Tras evaluar las metodologías anteriores, se ha optado por un enfoque híbrido que combina elementos de la metodología en cascada y las metodologías ágiles, adaptándose a las necesidades específicas de cada fase del proyecto:

Fases de planificación e investigación

Estas fases requieren un análisis detallado y secuencial, donde se establecen los fundamentos teóricos y se sientan las bases para el resto del proyecto. La **metodología en cascada** es adecuada aquí, ya que permite estructurar el trabajo de manera lógica y ordenada.

Fase de implementación

La implementación del pipeline DevSecOps y la integración de las herramientas AST pueden beneficiarse de un enfoque ágil. Se podrá iterar sobre el Producto Mínimo Viable (MVP), integrando primero las herramientas esenciales y, si el tiempo lo permite, ampliando el alcance para incluir herramientas adicionales y otros objetivos secundarios.

El **enfoque ágil** permite adaptarse a posibles desafíos técnicos y aprovechar oportunidades de mejora.

Aplicación de la metodología en las fases del proyecto

A continuación, se detalla cómo se aplicará la metodología híbrida en cada fase:

- Fases de planificación e investigación – **Metodología en cascada**
    - Definición del proyecto y objetivos.
    - Revisión bibliográfica y desarrollo de fundamentos teóricos.
    - Análisis de la guía de OWASP y selección de herramientas

- Fase de implementación - **Metodología ágil**
    - **Sprint 1**
        - Tareas de implementación asociadas a los objetivos primarios.
    - **Sprint 2**
        - Tareas de implementación asociadas a los objetivos secundarios.
- Fase de cierre – **Metodología en cascada**
    - Análisis final de resultados y conclusiones.
    - Documentación de la memoria.

- Preparación de la presentación del trabajo.

Gestión de tareas y seguimiento

Para asegurar un desarrollo eficiente del proyecto:

- Se utilizarán herramientas de gestión de proyectos para planificar y hacer seguimiento de las tareas, ajustando el plan según sea necesario.
- Se mantendrá una comunicación regular con el tutor para compartir avances y recibir *feedback* oportuno.

1.4 Plan de trabajo

En este apartado se detalla el plan de trabajo diseñado para alcanzar los objetivos establecidos en este TFM. Se presentan las tareas asociadas a cada objetivo, agrupadas en **objetivos primarios** y **secundarios**, y se muestra su planificación en fases, con entregas parciales al final de cada una.

1.4.1. Listado de tareas a realizar

Para alcanzar los objetivos definidos, se ha establecido un conjunto de tareas organizadas en función de su importancia dentro del alcance del TFM. En primer lugar, se abordarán las tareas relacionadas con los objetivos primarios, que constituyen el producto mínimo viable (MVP) y aseguran la cobertura de las necesidades fundamentales del proyecto. Posteriormente, se describirán las tareas correspondientes a los objetivos secundarios, concebidas como ampliaciones opcionales que aportan valor añadido al MVP y, si la planificación lo permite, podrían abordarse en una segunda fase.

Tareas asociadas a los objetivos primarios

Estas actividades son las que garantizan la base conceptual y técnica del proyecto. Su realización asegurará una comprensión sólida de las herramientas AST, la selección adecuada de soluciones, la configuración de un entorno *DevSecOps* funcional y la implementación de un laboratorio de pruebas realista:

1. Estudiar y analizar las distintas herramientas de seguridad AST.
   - Realizar un análisis introductorio sobre los fundamentos de seguridad en el desarrollo del software, para comprender el contexto en el que se enmarcan las herramientas de Application Security Testing.
   - Investigar las herramientas mencionadas en la *OWASP DevSecOps Guideline* (OWASP Foundation, 2024)
   - Construir una matriz de herramientas y capacidades que facilite su comparación.

2. Seleccionar un conjunto acotado de herramientas para su posterior estudio en mayor profundidad.
    - Basarse en la matriz de herramientas y capacidades y en los criterios de evaluación de capacidades críticas establecidos por Gartner (Gartner, 2024).
    - Elegir herramientas líderes en el mercado que sean relevantes y adecuadas para entornos empresariales.

3. Seleccionar y configurar una plataforma *DevSecOps* que integre herramientas de Análisis Estático de Seguridad de Aplicaciones (SAST) y de Análisis de Composición de Software (SCA).
    - Implementar la integración de las herramientas seleccionadas en un pipeline automatizado.
    - Configurar la ejecución automática de estas herramientas en las operaciones de *commit*.

4. Desarrollar un entorno de prueba representativo.
    - Utilizar una aplicación con vulnerabilidades intencionales, como *OWASP Juice Shop* (OWASP Foundation, 2024), para realizar pruebas.
    - Validar la eficacia de la integración de las herramientas y analizar el impacto en la calidad y seguridad del software.

Tareas asociadas a los objetivos secundarios

Las tareas asociadas a los objetivos secundarios se plantean como extensiones opcionales que profundizan en aspectos complementarios de la seguridad de aplicaciones. Aunque no son indispensables para completar el producto mínimo viable, estas actividades pueden aportar un valor añadido, ya sea realizando un análisis más detallado del rendimiento de las herramientas AST mediante una metodología de benchmarking, o integrando capacidades DAST que amplíen el alcance funcional del pipeline *DevSecOps*. Su realización dependerá de la disponibilidad de tiempo, así como de las prioridades establecidas durante el proyecto.

1- Analizar una metodología de benchmarking de aplicaciones AST.
    - Estudiar una herramienta de benchmarking de referencia, como la *OWASP Benchmark* (OWASP Foundation, 2024)
    - Desarrollar una guía de uso.

2- Ampliar el pipeline *DevSecOps* integrando características de Análisis Dinámico de Seguridad de Aplicaciones (DAST).
    - Configurar la integración de la herramienta DAST seleccionada.
    - Automatizar su ejecución en las etapas apropiadas del ciclo de desarrollo.

1.4.2.Planificación de las tareas

La planificación de las tareas se ha realizado considerando un esfuerzo total de 300 horas, acorde con los créditos asignados al TFM. Este esfuerzo está calculado sobre un promedio de 4 horas de dedicación diarias durante un período de 75 días. Se han tenido en cuenta periodos de mayor disponibilidad, como fines de semana y vacaciones, para avanzar en las etapas más demandantes.

	Duración (días)	**Esfuerzo (horas)**
1. Fase de planificación	**9**	**36**
1.1. Definición del proyecto	*2*	*8*
1.2. Enfoque y metodología	*2*	*8*
1.3. Plan de trabajo	*2*	*8*
1.4. Estado de la cuestión	*2*	*8*
1.5. Análisis de riesgos	*1*	*4*
2. Fase de investigación	**16**	**64**
2.1. Revisión bibliográfica	*5*	*20*
2.2. Desarrollo de los fundamentos teóricos	*2*	*8*
2.3. Estudio de la guía de OWASP	*3*	*12*
2.4. Definición de los criterios de evaluación	*2*	*8*
2.5. Ranking de fabricantes	*2*	*8*
2.6. Selección de herramientas	*2*	*8*
3. Fase de implementación	**25**	**100**
3.1. Sprint 1		
3.1.1. Construcción del repositorio de código	*3*	*12*
3.1.2. Diseño y configuración del pipeline	*3*	*12*
3.1.3. Integración herramientas SAST, SCA	*5*	*20*
3.1.4. Ejecución y verificación de las pruebas	*3*	*12*
3.2. Sprint 2 (opcional)		
3.2.1. Plataformas de benchmarking	*3*	*8*
3.2.3. Integración herramienta DAST	*5*	*20*
3.2.4. Ejecución y verificación de las pruebas	*3*	*12*
4. Fase de cierre	**25**	**100**
4.1. Análisis de resultados	*5*	*20*
4.2. Conclusiones	*5*	*20*
4.3. Documentación de la memoria	*10*	*40*
4.4. Preparación del vídeo de presentación del trabajo	*5*	*20*
TOTAL	***75***	***300***

1.4.3.Diagrama de Gantt

El siguiente cronograma muestra el plan de trabajo señalando en color rojo los hitos clave:

Nombre de tarea	Duración	Comienzo	Fin
Trabajo Final de Máster	**75 días**	**mié 25/09/24**	**mar 07/01/25**
Fase de Planificación	**9 días**	**mié 25/09/24**	**lun 07/10/24**
Definición del proyecto	2 días	mié 25/09/24	jue 26/09/24
Enfoque y metodología	2 días	vie 27/09/24	lun 30/09/24
Plan de trabajo	2 días	mar 01/10/24	mié 02/10/24
Estado de la cuestión	2 días	jue 03/10/24	vie 04/10/24
Análisis de riesgos	1 día	lun 07/10/24	lun 07/10/24
Entrega PAC1	0 días	lun 07/10/24	lun 07/10/24
Fase de Investigación	**16 días**	**mar 08/10/24**	**mar 29/10/24**
Revisión bibliográfica	5 días	mar 08/10/24	lun 14/10/24
Desarrollo de los fundamentos teóricos	2 días	mar 15/10/24	mié 16/10/24
Estudio de la guía de OWASP	3 días	jue 17/10/24	lun 21/10/24
Definición de los criterios de evaluación	2 días	mar 22/10/24	mié 23/10/24
Ranking de fabricantes	2 días	jue 24/10/24	vie 25/10/24
Selección de herramientas	2 días	lun 28/10/24	mar 29/10/24
Entrega PAC2	0 días	mar 29/10/24	mar 29/10/24
Fase de Implementación	**19 días**	**mié 30/10/24**	**lun 25/11/24**
SPRINT 1 (MVP)	**14 días**	**mié 30/10/24**	**lun 18/11/24**
Construcción del repositorio de código	3 días	mié 30/10/24	vie 01/11/24
Diseño y configuración del pipeline	3 días	lun 04/11/24	mié 06/11/24
Integración herramientas SAST, SCA	5 días	jue 07/11/24	mié 13/11/24
Ejecución de las pruebas	3 días	jue 14/11/24	lun 18/11/24
Entrega PAC3.1	0 días	lun 18/11/24	lun 18/11/24
SPRINT 2 (Opcional)	**5 días**	**mar 19/11/24**	**lun 25/11/24**
Plataformas de benchmarking	5 días	mar 19/11/24	lun 25/11/24
Entrega PAC3.2	0 días	lun 25/11/24	lun 25/11/24
Fase de Cierre	**31 días**	**mar 26/11/24**	**mar 07/01/25**
Análisis de resultados	10 días	mar 26/11/24	lun 09/12/24
Conclusiones	5 días	mar 10/12/24	lun 16/12/24
Completar la memoria	10 días	mar 17/12/24	lun 30/12/24
Entrega PAC4	0 días	lun 30/12/24	lun 30/12/24
Preparación del vídeo	6 días	mar 31/12/24	mar 07/01/25
Entrega vídeo	0 días	mar 07/01/25	mar 07/01/25
Fin TFM	0 días	mar 07/01/25	mar 07/01/25

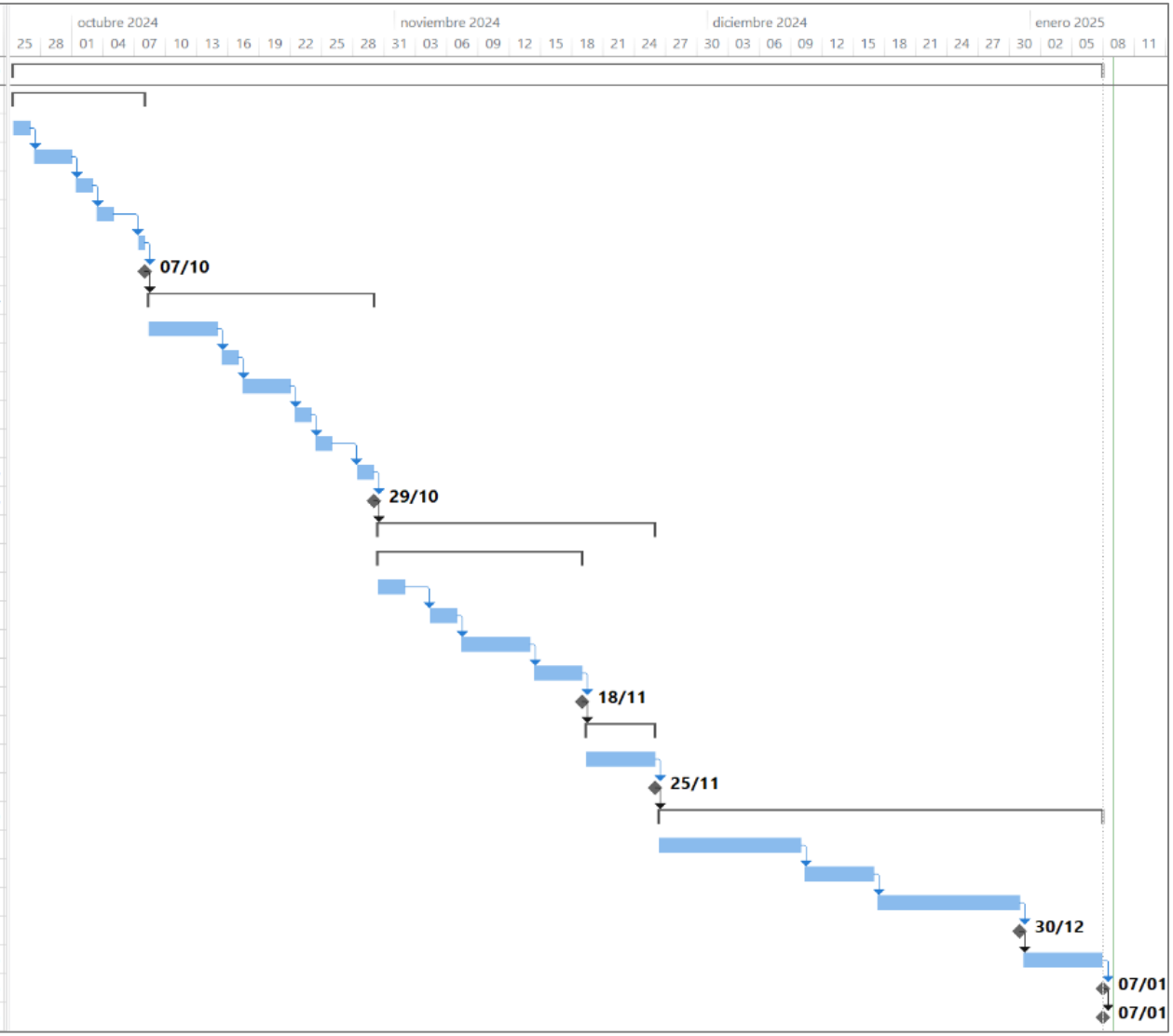


Ilustración 1. Diagrama de Gantt

1.5 Breve revisión del estado del arte

1.5.1. Visión general

En este apartado se explorarán los modelos y marcos de referencia que ayudan a las organizaciones a mejorar sus prácticas de seguridad en el desarrollo de software, poniendo especial énfasis en las propuestas de OWASP.

Para abordar los desafíos de seguridad en el desarrollo de software, es esencial adoptar un enfoque estructurado y maduro que permita a las organizaciones evaluar y mejorar sus prácticas de seguridad. Una herramienta valiosa en este sentido es el Modelo de Madurez para el Aseguramiento del Software de OWASP (en inglés, *Software Assurance Maturity Model*). Este modelo proporciona un marco de referencia que ayuda a las organizaciones a mejorar sus prácticas de seguridad en el desarrollo de software.

El modelo de madurez de OWASP se compone de cinco funciones de negocio: Gobernanza, Diseño, Implementación, Verificación y Operaciones. Cada una de estas funciones incluye actividades de seguridad clave que, en conjunto, permiten a las organizaciones avanzar en su capacidad para producir software seguro.

A continuación, se presenta una ilustración del modelo de madurez para el aseguramiento del software de OWASP, que muestra las diferentes funciones de negocio y cómo se integran en el ciclo de desarrollo seguro.

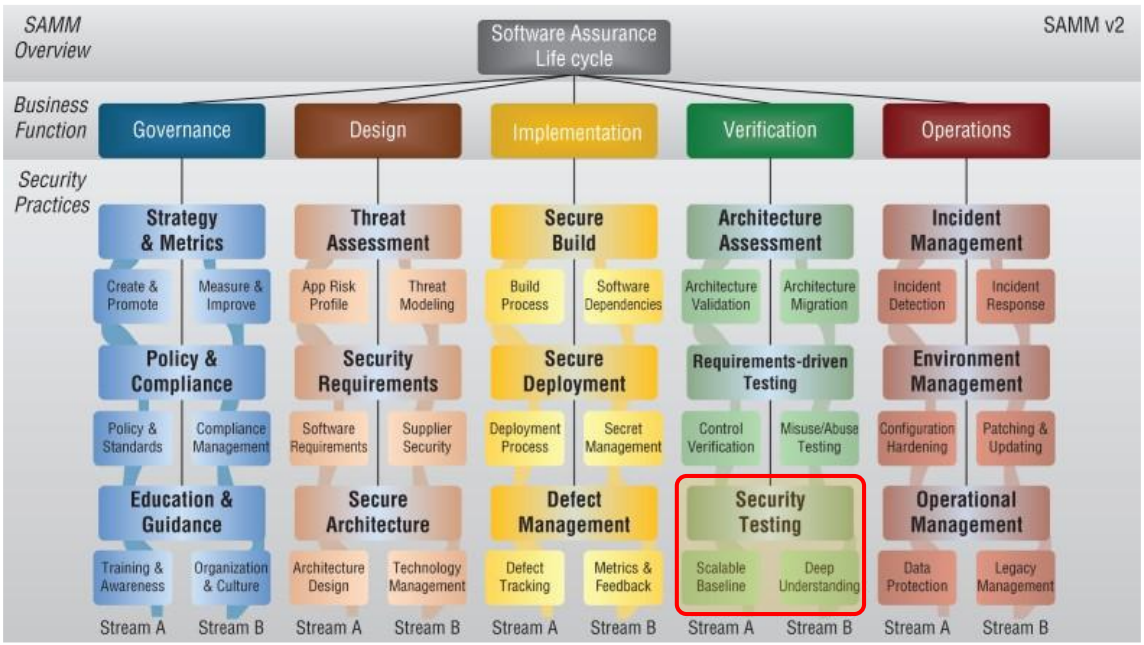


Ilustración 2. Software Assurance Maturity Model. CISSP Official Study Guide (Chapple, 2024)

Dentro de este modelo, la función de **Verificación** es crucial, ya que se centra en garantizar que el software desarrollado cumple con los requisitos de seguridad establecidos. Las actividades de Verificación incluyen la realización de pruebas de seguridad utilizando herramientas como:

- **SAST** (Static Application Security Testing),

- **DAST** (Dynamic Application Security Testing),
- **IAST** (Interactive Application Security Testing) y
- **SCA** (Software Composition Analysis).

Estas herramientas permiten identificar vulnerabilidades tanto en el código propio como en los componentes de terceros, facilitando la mitigación de riesgos antes de que el software sea desplegado en producción. La implementación efectiva de estas prácticas de verificación es fundamental para prevenir ataques y asegurar la integridad y confidencialidad de la información.

La integración temprana de la seguridad en el ciclo de desarrollo del software es esencial para prevenir vulnerabilidades y reducir riesgos. El enfoque de "Shift-Left" en seguridad promueve la incorporación de prácticas y herramientas de seguridad desde las primeras etapas del desarrollo, permitiendo una detección y corrección más temprana de las vulnerabilidades.

Adicionalmente, la *OWASP DevSecOps Guideline* (OWASP Foundation, 2024) ofrece un marco detallado para integrar la seguridad en las prácticas de DevOps, estructurando sus recomendaciones en torno a tres áreas principales: Personas, Procesos y Gobierno.

- **Personas**: El factor humano es fundamental en *DevSecOps*. La guía enfatiza la necesidad de incluir expertos en seguridad desde las fases iniciales del proyecto y de formar a los equipos en prácticas de código seguro. La colaboración y comunicación efectiva entre los equipos de desarrollo, operaciones y seguridad son esenciales para el éxito.
- **Procesos**: Se divide en cuatro segmentos principales:
    - **Diseño**: Incluye actividades como el Modelado de Amenazas, que permite identificar y mitigar riesgos potenciales desde las etapas iniciales del desarrollo.
    - **Código**: Aborda prácticas como la gestión de secretos, análisis estático de código (**SAST**), análisis interactivo (**IAST**) y pruebas en tiempo real durante el desarrollo.
    - **Construcción**: Incluye pruebas dinámicas de aplicaciones (**DAST**), seguridad de APIs, verificación de configuraciones y escaneo de infraestructura como código (**IaC**).
    - **Operación**: Enfoca la gestión de claves y secretos, protección de aplicaciones nativas de la nube, gestión de vulnerabilidades y monitorización continua para detectar y responder a incidentes de seguridad.
- **Gobierno**: Implica la supervisión y control de la implementación de las prácticas de seguridad, asegurando el cumplimiento de políticas, normativas y estándares de la industria. Esto incluye auditorías de cumplimiento, protección de datos y la implementación de políticas de seguridad codificadas.

La guía de OWASP también destaca la importancia de integrar herramientas AST en cada fase del ciclo de vida del desarrollo, proporcionando recomendaciones sobre su uso y las mejores prácticas para su implementación efectiva.

A continuación, se presenta una ilustración que resume la metodología *DevSecOps* según OWASP:

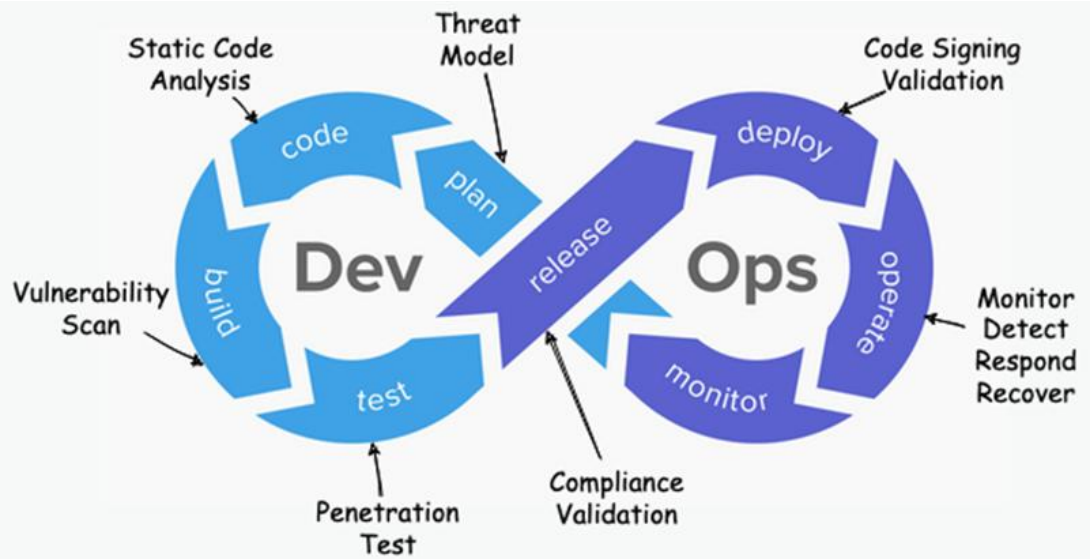


Ilustración 3. Metodología DevSecOps. Fuente: (OWASP Foundation, 2024)

Para seleccionar las herramientas más adecuadas en cada categoría, es fundamental contar con evaluaciones objetivas y actualizadas. Los informes de **Gartner**, como el Magic Quadrant for Application Security Testing (Mark Horvath, 2024) y el análisis de Capacidades Críticas para el Testing de Seguridad de Aplicaciones (Gartner, 2024), ofrecen una visión detallada del mercado y las capacidades de las herramientas AST.

Estos análisis permiten identificar las fortalezas y debilidades de las herramientas en función de las necesidades específicas, facilitando una selección informada.

La adopción de plataformas CI/CD como Jenkins, GitLab CI/CD y GitHub Actions facilita la automatización y ejecución de estas herramientas en los pipelines de desarrollo.

La tendencia actual se centra en automatizar la seguridad sin afectar la velocidad y eficiencia del desarrollo, promoviendo prácticas que permitan una detección temprana de vulnerabilidades y una respuesta ágil ante posibles amenazas, en consonancia con los principios establecidos por OWASP.

1.5.2.Líderes del mercado

La última actualización del “Magic Quadrant” de Gartner sobre Application Security Testing (Mark Horvath, 2024), clasifica a los 12 principales fabricantes de herramientas comerciales de AST según se muestra en la imagen siguiente:

Ilustración 4. Magic Quadrant for App Security Testing. Fuente: Gartner (Mark Horvath, 2024)

Es importante tener presente que Gartner se basa en estudios de mercado y solo incluye a los fabricantes que registran ingresos anuales muy sustanciales por productos AST, asegurando así que las empresas evaluadas tienen una presencia financiera sólida en el sector, además de cumplir con los siguientes requisitos técnicos:

- Sus soluciones deben soportar, como mínimo, capacidades SAST y SCA.
- Para las herramientas SAST, deben proporcionar soporte para los principales lenguajes de programación (Python, Java, C#, PHP, JavaScript…).
- Para las herramientas SCA, deben proporcionar las siguientes capacidades de escaneo:
  - Vulnerabilidades comunes conocidas

    - Librerías vulnerables obsoletas
    - Licencias inapropiadas
- Deben proporcionar sus propias capacidades de testing, sin depender de otras herramientas.
- Deben ofrecer plantillas para reportar las vulnerabilidades detectadas con acuerdo al *OWASP Top Ten* (OWASP Foundation, 2024) y a otros estándares.
- Deben proporcionar guías para la remediación de las vulnerabilidades.

Gartner distingue en el cuadrante de **líderes** a los cinco fabricantes que se indica a continuación. También se incluye un resumen de las principales fortalezas que posicionan a cada uno en ese cuadrante:

- **Synopsys**. Conocido por su amplio rango de capacidades AST, que incluye SAST, DAST, IAST y SCA, ofrecidos tanto en entornos on-premises como en soluciones SaaS. Su plataforma Polaris, un servicio SaaS, destaca por integrar todas las capacidades de AST. Synopsys ha fortalecido su capacidad de escaneo dinámico con adquisiciones estratégicas, como la de WhiteHat Security, complementando sus herramientas de análisis estático e interactivo. Synopsys también se distingue por su sólida oferta en la seguridad de la cadena de suministro, que es algo crítico en la actualidad.

- **Veracode.** Se distingue por ser una solución exclusivamente SaaS que ofrece un conjunto completo de herramientas AST, incluyendo SAST, DAST, IAST y seguridad de contenedores. Veracode también es reconocida por su certificación FedRAMP, lo que la convierte en una opción fuerte para industrias altamente reguladas y el sector gubernamental. Además, Veracode añade valor a través de sus ofertas de formación para desarrolladores y su capacidad para ayudar a las organizaciones a comparar su madurez en seguridad con los estándares de la industria.

- **Checkmarx**. Destaca por su sólida integración con los desarrolladores a lo largo de todo el ciclo de vida del desarrollo. La empresa ha mejorado la experiencia del desarrollador con características como DevHub, que facilita la identificación y corrección de vulnerabilidades en el código abierto. Además, su motor Checkmarx Fusion permite una integración a nivel de repositorio, proporcionando a los desarrolladores una vista completa de la seguridad de sus aplicaciones.

- **OpenText** (Fortify). Se distingue por sus múltiples opciones de despliegue, que incluyen on-premises, SaaS, nube privada y servicios gestionados. Un aspecto clave es la incorporación de aprendizaje automático para mejorar la precisión en la identificación de vulnerabilidades y reducir los falsos positivos. Fortify también sobresale en el análisis de la composición del software (SCA) y en la seguridad de la cadena de suministro, con asociaciones estratégicas como la de Sonatype.

- **Snyk**. Se diferencia por sus capacidades de seguridad para aplicaciones nativas en la nube, que abarcan SAST, SCA, IaC y escaneo de contenedores. La fortaleza de Snyk radica en su enfoque centrado en los desarrolladores, con una estrecha integración en los flujos de trabajo de DevOps y proporcionando escaneo automatizado y capacidades de remediación. Además, Snyk cuenta con una amplia base de datos de vulnerabilidades y sigue innovando con adquisiciones como Fugue, que amplía sus capacidades de gestión de la postura de seguridad en la nube.

1.5.3.La Inteligencia artificial aplicada al ámbito AST

En los últimos años, la Inteligencia Artificial ha comenzado a revolucionar campos muy diversos, y el ámbito de las herramientas de análisis de seguridad no ha sido la excepción. La capacidad de las tecnologías de IA para procesar grandes volúmenes de datos, identificar patrones complejos y aprender de manera autónoma ha abierto nuevas posibilidades para mejorar la detección de vulnerabilidades en el código y reducir los falsos positivos.

A medida que estas herramientas avanzan, van integrando capacidades de IA, distintas según el tipo de herramienta. A continuación, se presentan ejemplos de cómo diferentes herramientas AST están incorporando características de IA para optimizar sus capacidades (Mark Horvath, 2024):

- **HCLSoftware**:

  HCL AppScan utiliza técnicas de Machine Learning y procesamiento de lenguaje natural para mejorar la precisión y reducir los falsos positivos en sus hallazgos. Las características de Intelligent Findings Analytics (IFA) e Intelligent Code Analytics (ICA) mejoran el proceso de análisis de seguridad agrupando hallazgos e investigando nuevas y desconocidas APIs.

- **Veracode**:

  Ha comenzado a integrar IA para mejorar la precisión en la identificación de vulnerabilidades y reducir falsos positivos en su análisis estático (SAST).

- **Checkmarx:**

  Está desarrollando capacidades avanzadas que incluyen el uso de IA para mejorar la eficiencia en la detección de vulnerabilidades en sus herramientas SAST e IAST.

- **SonarQube:**

  Aunque SonarQube tradicionalmente no se ha basado en IA, existen plugins y extensiones que permiten aprovechar modelos de IA para mejorar la detección de defectos y vulnerabilidades en el código.

- **Contrast Security:**

  Utiliza IA para analizar el comportamiento de las aplicaciones en tiempo real, mejorando la detección de vulnerabilidades en sus herramientas de IAST.

- **Synopsys BlackDuck:**

  Synopsys ha integrado capacidades de IA en su herramienta BlackDuck para la identificación más eficiente de vulnerabilidades en componentes de código abierto (SCA).

- **Snyk:**

  En su enfoque de análisis de componentes de terceros (SCA), también utiliza IA para identificar rápidamente vulnerabilidades en dependencias y sugerir parches de manera automatizada.

1.6 Listado de posibles riesgos y soluciones

A continuación, se identifican los principales riesgos asociados al desarrollo de este proyecto, junto con las medidas propuestas para reducir su impacto en el progreso y el cumplimiento de los objetivos:

Riesgo	**Descripción**	**Impacto**	**Medidas**
Falta de familiaridad con las herramientas AST y las plataformas CI/CD	La integración de las herramientas podría ser más compleja de lo previsto, comprometiendo el alcance o los plazos del proyecto.	Alto	Realizar pruebas de concepto tempranas, y descartar herramientas que planteen un nivel de dificultad excesivo en relación con los objetivos del proyecto.
Imposibilidad de acceder a herramientas comerciales	La falta de acceso a versiones completas de las herramientas comerciales podría limitar la profundidad del análisis o la validez de los resultados obtenidos.	Medio	Priorizar el uso de versiones gratuitas, o de evaluación, que permitan completar los experimentos requeridos sin tener que afrontar costes significativos.
Factores externos o personales que interfieran con el calendario de entregas	Pueden surgir imprevistos que afecten el cumplimiento del cronograma o la calidad del trabajo final.	Bajo	Planificar márgenes de tiempo adecuados, y mantener un control riguroso del plan de trabajo mediante Microsoft Project para ajustar los plazos en caso de imprevistos. Mantener comunicación constante con el tutor para priorizar tareas clave si es necesario.

1.7 Listado de costes asociados al material

En este apartado se detallan los costes asociados a los recursos materiales necesarios para el desarrollo de este Trabajo Final de Máster. Se describirán los componentes de hardware y software utilizados, así como las herramientas y servicios requeridos.

Dado que el proyecto prioriza el uso de tecnologías de código abierto y plataformas gratuitas, los costes asociados son mínimos y se buscará optimizar los recursos disponibles para reducir al máximo el presupuesto necesario.

- Ordenador personal con acceso a Internet

  Se utilizará un ordenador doméstico de tipo medio, ya que se priorizará el uso de tecnología cloud y no será necesaria la instalación de software pesado en el cliente.

  Debido a que el ordenador y la conexión a Internet también se utilizarán para otros fines, los costes imputables específicamente a este proyecto se consideran despreciables.

- Plataforma CI/CD

  Se utilizarán entornos de desarrollo de acceso gratuito, como Jenkins, GitLab o GitHub.

- Herramientas AST

  Siempre que sea posible, se emplearán herramientas de código abierto o herramientas comerciales con licencias de uso gratuitas o versiones de evaluación.

Coste total estimado

El coste total estimado es mínimo, prácticamente nulo, ya que se priorizará el uso de herramientas y servicios gratuitos. Este coste podría aumentar si fuera necesario recurrir a licencias de pago en algún momento.

1.8 Implicaciones legales o éticas

En el desarrollo de este Trabajo de Fin de Máster se tomarán en cuenta diversas consideraciones legales y éticas, alineadas con la **Guía Transversal de la CCEG** de la UOC.

A continuación, se detallan las implicaciones más relevantes en tres dimensiones clave: sostenibilidad, comportamiento ético y responsabilidad social, y diversidad, género y derechos humanos.

1.8.1.Sostenibilidad

En el contexto de un proyecto relacionado con la ciberseguridad, es importante evaluar el impacto medioambiental del uso de herramientas automáticas y procesos en la nube, y evitar la necesidad de mantener servidores continuamente activos. Esta política está enmarcada en los Objetivos de Desarrollo Sostenible (ODS), en particular en el ODS 7 (Energía asequible y no contaminante). Para contribuir a este objetivo, se prioriza el uso de servicios en la nube que permiten el apagado automático de recursos no utilizados, reduciendo significativamente el consumo energético cuando no se están ejecutando tareas activas. En particular, los *tests* estáticos de seguridad se ejecutarán en máquinas virtuales efímeras proporcionadas por Azure. Estas máquinas se crean y destruyen según la demanda, optimizando el uso de los recursos y evitando el consumo innecesario de energía.

Además, se promoverá el uso de servicios en la nube de proveedores que certifiquen su compromiso con prácticas sostenibles, como Microsoft, que se ha comprometido a ser carbono negativo para 2030 (Carbon Leadership Forum, 2024) a pesar del reto que suponen las nuevas cargas de trabajo requeridas por la Inteligencia Artificial.

Este enfoque también fomenta la optimización de los ciclos de integración continua (CI/CD) y la adopción de entornos virtualizados, lo que permite reducir la infraestructura física y, por lo tanto, la huella de carbono. Estas prácticas están alineadas con el ODS 13 (Acción por el clima), promoviendo un uso más eficiente de los recursos. Concretamente, la implementación de pipelines en GitHub Actions permite ejecutar análisis solo cuando se detectan cambios en el código, minimizando así el uso de recursos innecesarios.

El impacto sostenible de las herramientas de seguridad también puede observarse en la reducción de residuos digitales mediante la disminución de la necesidad de actualizaciones relacionadas con brechas de seguridad. Al mejorar la seguridad de los sistemas, se disminuyen las pérdidas económicas por ciberataques y se fomenta una economía más estable y sostenible.

1.8.2.Comportamiento Ético y Responsabilidad Social

Este proyecto implica el uso de herramientas de análisis de seguridad, que tienen el potencial de detectar vulnerabilidades en aplicaciones web. Por ello, se implementarán medidas para asegurar que los resultados obtenidos no se utilicen para fines ilícitos o que vulneren la privacidad o la seguridad de los sistemas analizados.

Para garantizar un comportamiento ético, se adoptarán las siguientes medidas:

- **Protección de datos**: Durante todo el desarrollo del TFM, se garantizará el cumplimiento de las normativas de protección de datos (GDPR), evitando el uso indebido de información sensible.
- **Prevención de uso indebido**: A través del copyright de este trabajo, se establece una protección explícita sobre los resultados obtenidos durante los análisis de seguridad. Esto implica que cualquier uso de los resultados, como la explotación de vulnerabilidades detectadas, queda prohibido sin la autorización previa del autor. Esta medida asegura que los conocimientos adquiridos y los hallazgos del proyecto sean utilizados únicamente con fines educativos y de mejora de la seguridad, evitando su uso malintencionado en aplicaciones reales. Estas acciones se alinean con el ODS 16 (Paz, justicia e instituciones sólidas), promoviendo un uso responsable, ético y seguro de las tecnologías desarrolladas, y asegurando la integridad y confidencialidad de los datos de análisis.

El impacto ético-social de estas herramientas es positivo en cuanto a la mayor protección de los datos personales, lo que contribuye a una mayor confianza en el mundo digital. Un software más seguro favorece el derecho fundamental a la privacidad, mientras que la ciberseguridad refuerza la confianza y la inclusión digital y social.

1.8.3.Diversidad, Género y Derechos Humanos

El proyecto considerará la importancia de la inclusividad y la accesibilidad en el desarrollo y uso de herramientas de seguridad. Aunque las herramientas AST utilizadas en el proyecto se centran en la detección de vulnerabilidades técnicas, se prestará atención a la accesibilidad en su uso y a la promoción de un entorno de trabajo inclusivo.

Se evitará el uso de herramientas o enfoques que refuercen sesgos discriminatorios, garantizando que el código y los análisis sean neutrales desde el punto de vista de género y raza. Asimismo, se seguirán las guías de estilo de la UOC para garantizar una redacción inclusiva y respetuosa en la memoria del TFM, promoviendo un lenguaje que no perpetúe estereotipos de género ni discriminación. Estos esfuerzos están alineados con los ODS 5 (Igualdad de género) y 10 (Reducción de las desigualdades).

2. Fase de investigación

2.1 Introducción

Este capítulo aborda los conceptos fundamentales y las herramientas clave que facilitan la integración de la seguridad en el ciclo de vida del desarrollo de software (SDLC). Se exploran los fundamentos de seguridad en el desarrollo de software, reconociendo la importancia de metodologías como el modelado de amenazas y el enfoque de "*Shift-Left*", que promueve la incorporación de prácticas de seguridad desde las etapas más tempranas del desarrollo.

Además, se analizan las herramientas recomendadas por la guía OWASP, destacando su aplicación en diferentes fases del SDLC y su relevancia en la identificación y mitigación de vulnerabilidades mediante técnicas como SAST, DAST e IAST. También se evalúan diversas herramientas de seguridad de aplicaciones (AST), utilizando criterios establecidos por analistas reconocidos, y se establece un ranking de fabricantes basado en sus capacidades para satisfacer las necesidades de seguridad en el contexto empresarial.

Finalmente, se presentan las herramientas seleccionadas tanto para entornos empresariales como para entornos de laboratorio. En el contexto empresarial, se destacan soluciones que ofrecen un equilibrio óptimo entre funcionalidad y eficiencia. Para el entorno de laboratorio, se eligen herramientas accesibles y prácticas que permiten implementar ejercicios de seguridad con fines didácticos, considerando también la elección de una plataforma *DevSecOps* adecuada para las pruebas.

2.2 Fundamentos de seguridad en el desarrollo del software

En el desarrollo de software actual, es fundamental integrar la seguridad desde las etapas más tempranas para prevenir vulnerabilidades y reducir riesgos.

Esta sección aborda los fundamentos esenciales de la seguridad en el ciclo de vida del software, centrándose en el modelado de amenazas y el concepto de "Shift-Left", que promueve la incorporación de prácticas de seguridad desde el inicio del proceso de desarrollo.

2.2.1. Modelado de amenazas

El principal objetivo de este TFM es el análisis de las metodologías y herramientas de pruebas que deben utilizarse dentro del ciclo del desarrollo del software para verificar su seguridad y su calidad. Sin embargo, conviene hacer una parada preliminar para introducir el concepto de “Modelado de Amenazas” o “***Threat Modeling***”.

Se trata de una actividad fundamental que debe abordarse desde la etapa de diseño de la aplicación, con el objetivo de identificar todas las medidas de seguridad con las que deberá contar tanto el propio software del aplicativo como el sistema que lo

hospede, y es donde se determinará, además, cuáles son los casos de prueba que se requiere implementar para poder hacer las verificaciones precisas.

Es decir, las herramientas de pruebas sólo resultarán de plena utilidad si previamente se ha determinado qué es lo que se debe probar, y el modelado de amenazas es lo que permitirá dar respuesta a esa cuestión.

El libro "*Threat Modeling: Designing for Security*" (Shostack, 2014), obra de referencia en este ámbito, propone un método de modelado estructurado en cuatro pasos diferenciados ("*The Four-Step Framework*"):

1- **Modelar** el sistema: ¿Qué es lo que se está construyendo?

2- **Identificar** las amenazas: ¿Qué es lo que puede ir mal una vez esté construido el sistema?

3- **Tratar** las amenazas: ¿Qué debe hacerse al respecto de las cosas que pueden ir mal?

4- **Verificar**: ¿Se ha hecho el análisis correcto?

Modelar el sistema

Este primer paso consiste en dibujar un diagrama que represente el sistema desde el punto de vista del flujo de datos que se va a producir a través de las entidades y los procesos subyacentes, y delimitando las áreas que vayan a estar controladas por actores diferentes ("*trust boundaries*").

Una de las técnicas más recomendadas es el Diagrama de Flujo de Datos (DFD), como el ejemplo que se muestra en la imagen siguiente:

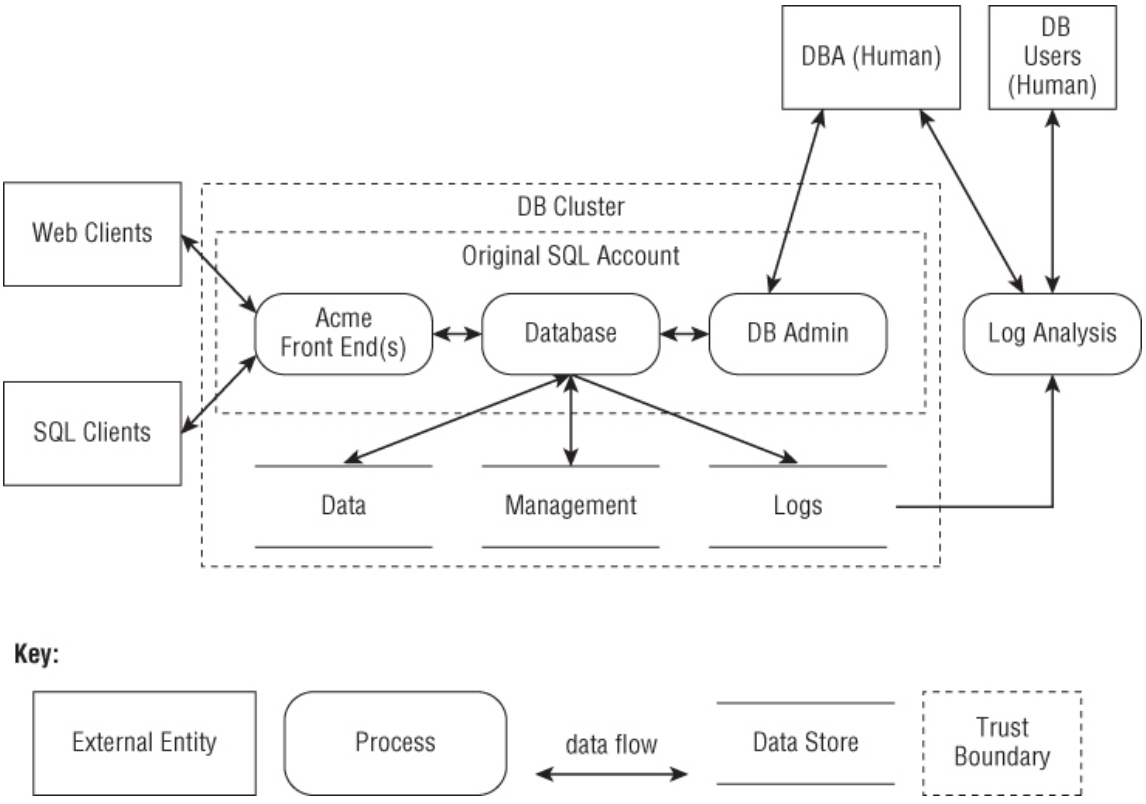


Ilustración 5. Modelo DFD. Fuente "Threat Modeling: Designing for Security" (Shostack, 2014)

Identificar las amenazas

Una vez se dispone del diagrama que representa el sistema que se va a construir, la pregunta clave a plantearse es **¿Qué puede ir mal?**

Para encontrar las respuestas se pueden utilizar diferentes estrategias, y una de las más extendidas es el uso del nemónico **STRIDE**, que establece las 6 categorías de amenazas principales:

Spoofing | **T**ampering | **R**epudiation | **I**nformation disclosure | **D**enial of service | **E**levation of privileges

La *Threat Modeling Cheat Sheet* de OWASP (OWASP Foundation, s.f.) muestra cual es la propiedad de la seguridad de la información que se ve transgredida por cada una de esas categorías de amenazas, e incorpora un ejemplo de cada una para facilitar su comprensión, tal como se aprecia en la tabla siguiente:

Categoría	**Viola…**	**Ejemplos**
Spoofing	Autenticidad	Un atacante roba el token de autenticación de un usuario legítimo y lo utiliza para hacerse pasar por él.
Tampering	Integridad	Un atacante abusa de la aplicación para realizar actualizaciones no intencionadas en una base de datos.
Repudiation	No-repudio	Un atacante manipula los registros para ocultar sus acciones.
Information Disclosure	Confidencialidad	Un atacante extrae datos de una base de datos que contiene información de cuentas de usuario.
Denial of Service	Disponibilidad	Un atacante bloquea a un usuario legítimo de su cuenta realizando muchos intentos fallidos de autenticación.
Elevation of Privileges	Autorización	Un atacante manipula un token JWT para cambiar su rol.

Tabla 1. STRIDE, ejemplos de amenaza por categoría

Existen diferentes herramientas de soporte, como el ***Microsoft Threat Modeling Tool***, ***IriusRisk***, o el ***OWASP Threat Dragon***, que pueden ser utilizadas en el entorno de desarrollo para facilitar la identificación de las amenazas presentes en cada elemento y en cada una de las posibles interacciones de ese elemento con el sistema. Aunque no se integran directamente en los entornos de desarrollo integrados (IDE), estas herramientas permiten crear y gestionar modelos de amenazas que pueden ser referenciados durante el ciclo de vida del desarrollo de software, ayudando a guiar la implementación de medidas de seguridad.

La imagen siguiente muestra un sistema de ejemplo en el que interviene un browser, un flujo de datos iniciado por un interactuante externo, 2 procesos (Contoso.exe y Fabrikam.dll), y una base de datos:

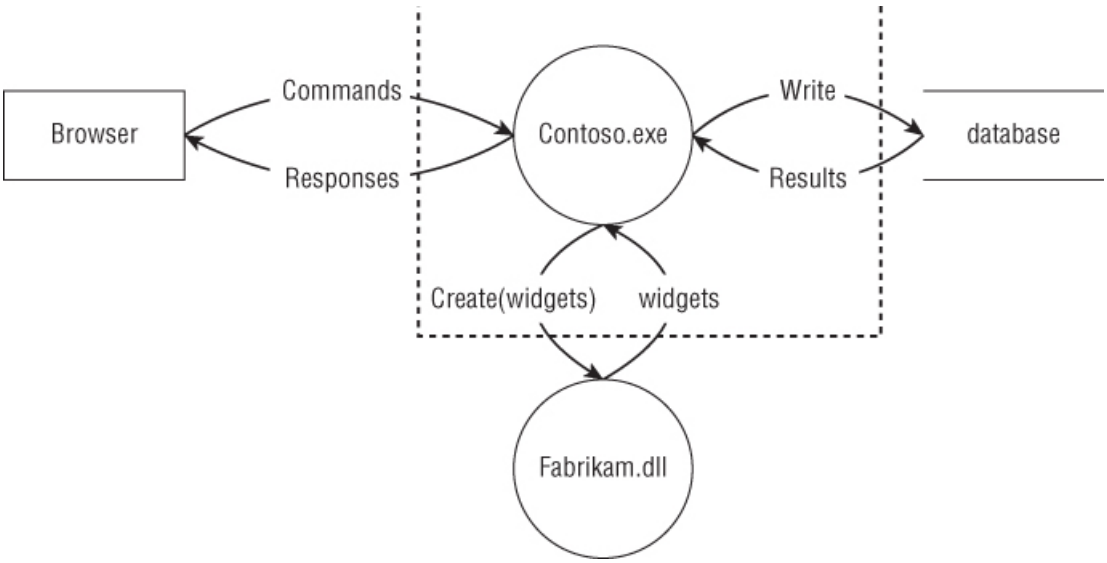


Ilustración 6. STRIDE, flujo de interacciones (Shostack, 2014)

La técnica consiste en desglosar las posibles interacciones de cada elemento, e identificar la tipología de amenazas existente para cada una, tal como muestra la tabla a continuación (Shostack, 2014):

#	**Elemento**	**Interacción**	**S**	**T**	**R**	**I**	**D**	**E**
1	Proceso (Contoso)	El proceso tiene un flujo de datos saliente hacia el almacén de datos.	x		x			
2		El proceso envía salida a otro proceso.	x		x	x	x	x
3		El proceso envía salida a un interactuante externo (código).	x		x	x	x	
4		El proceso envía salida a un interactuante externo (humano).			x			
5		El proceso tiene un flujo de datos entrante desde el almacén de datos.	x	x		x	x	
6		El proceso tiene un flujo de datos entrante desde un proceso.	x		x		x	x
7		El proceso tiene un flujo de datos entrante desde un interactuante externo.	x			x	x	
8	Flujo de Datos (comandos/ respuestas)	Cruza el límite de la máquina.		x		x	x	

9	Almacén de Datos (base de datos)	El proceso tiene un flujo de datos saliente hacia el almacén de datos.	x	x	x	x	x	
10		El proceso tiene un flujo de datos entrante desde el almacén de datos.		x	x	x		
11	Interactuante Externo (navegador)	El interactuante externo pasa entradas al proceso.	x		x	x		
12		El interactuante externo recibe entradas del proceso.	x					

Tabla 2. STRIDE, análisis de interacciones (Shostack, 2014)

Se ha señalado en verde la interacción número 6, correspondiente al flujo de datos entrante en Contoso procedente de Fabrikam, para utilizarlo a continuación como ejemplo ilustrativo de esta técnica, profundizando en las amenazas identificadas en esa interacción:

- **S (Spoofing - Suplantación):** Contoso cree que está recibiendo datos de Fabrikam, pero en realidad podría ser un atacante suplantando la identidad de Fabrikam.
- **T (Tampering - Manipulación):** No está presente en esta interacción.
- **R (Repudiation - Repudio):** Contoso puede negar (repudiar) haber recibido datos de Fabrikam.
- **I (Information Disclosure - Divulgación de Información):** No está presente en esta interacción.
- **D (Denial of Service - Denegación de Servicio):** Contoso se bloquea/detiene debido a la interacción con Fabrikam.
- **E (Elevation of Privileges - Elevación de Privilegios):** Fabrikam pasa datos o argumentos que le permiten cambiar el flujo de ejecución de Contoso para ganar privilegios o controlar su comportamiento.

Tratar las amenazas

Según la Guía de Estudio de CompTIA Security+ (Mike Chapple D. S., 2024), las metodologías de Gestión de Riesgos distinguen 4 posibles tipos de acción a tomar frente a una amenaza:

- **Mitigarla**: Aplicar controles de seguridad para reducir la probabilidad y/o la magnitud del riesgo.
- **Evitarla**: Modificar las prácticas de negocio (es decir, características funcionales de la aplicación) para eliminar por completo la posibilidad de que se materialice el riesgo.
- **Transferirla**: Transferir la responsabilidad a un tercero, por ejemplo, comprando una póliza de seguro que cubra los daños que se puedan producir.

- **Aceptarla**: Aplica cuando el coste de mitigar un riesgo es mayor que el impacto del propio riesgo. En ese caso, se autoriza una excepción.

La acción más común será mitigar el riesgo mediante la aplicación de controles de seguridad. Por ejemplo, frente a una amenaza de suplantación ("*spoofing*") de la identidad de una persona, la estrategia de mitigación consistirá en establecer un mecanismo robusto de identificación con múltiple factor de autenticación (algo que el usuario sepa, tenga, o sea).

Verificar el trabajo

Este último paso, sobre el que normalmente se iterará antes de la primera puesta en Producción y de los sucesivos despliegues, consiste en revisar el modelo, verificar que se ha identificado cada posible amenaza, y comprobar que se han contemplado y realizado las pruebas pertinentes.

A modo de checklist, se propone reflexionar sobre los siguientes aspectos:

1- Verificar el modelo
    a. ¿Es lo suficientemente completo y detallado?
    b. ¿Cubre todas las decisiones de seguridad que se han tomado?
    c. ¿Se puede abordar la siguiente versión sin incorporar cambios al diagrama?
2- Verificar cada amenaza
    a. ¿Se ha tomado alguna acción específica para cada amenaza?
    b. ¿Se ha tomado la acción correcta en cada caso?
3- Verificar las pruebas
    a. ¿Están alineadas las pruebas de seguridad con las demás pruebas del software y con los riesgos que podrían surgir en caso de fallo?

2.2.2.Integración temprana de la seguridad en el ciclo de desarrollo

La Guía de Seguridad de la CSA (Rich Mogull, Mike Rothman, 2024) introduce el concepto de ***Shift-Left*** en el contexto de la seguridad en el ciclo de vida del desarrollo de software (SSDLC). Este enfoque surge como respuesta a la necesidad de mover las actividades de seguridad a la izquierda, es decir, a las fases más tempranas del desarrollo, en lugar de esperar hasta la fase de mantenimiento o, peor aún, después de que ocurra un incidente de seguridad en producción.

El *Shift-Left* busca integrar la seguridad desde las primeras etapas del SSDLC, garantizando que cada fase del desarrollo esté alineada con los principios de seguridad. Esto no solo mejora la seguridad del producto final, sino que también es una estrategia más eficiente en términos de costes y recursos, al identificar y mitigar vulnerabilidades antes de que se conviertan en problemas graves.

La guía de la CSA proporciona un marco claro sobre cómo implementar esta integración de seguridad en un entorno *DevSecOps*, abarcando desde la arquitectura y el desarrollo inicial hasta la producción:

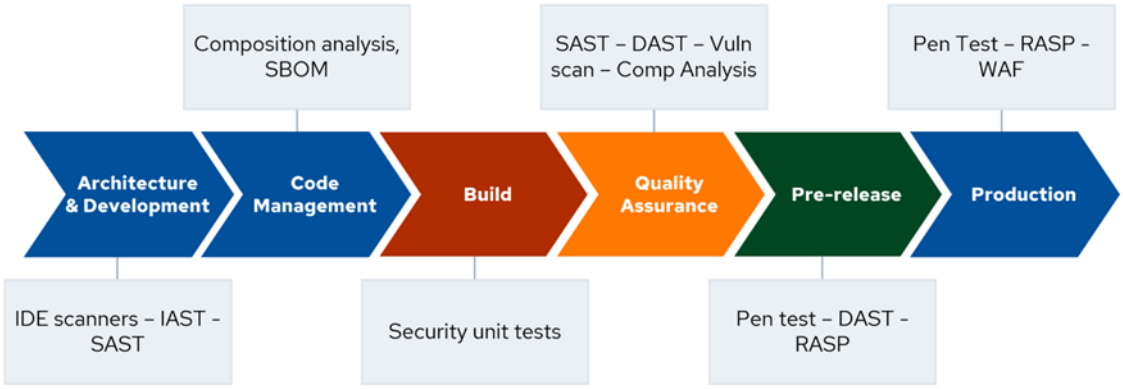


Ilustración 7. Integración de la seguridad en el desarrollo (Rich Mogull, Mike Rothman, 2024)

La tabla siguiente resume las medidas de seguridad proactivas aplicadas en varias etapas del ciclo de vida del software, indicando y justificando dónde se implementa cada técnica:

Where?	What?	Why?
Integrated Development Environments (IDE)	SAST	Detects source code vulnerabilities and provides the developer real-time feedback as they code
Repository	Software Composition Analysis (SCA)	Detects vulnerable dependencies and libraries
Build phase	Security Unit Tests	Detects module-level security vulnerabilities
Quality Assurance phase	SAST IAST DAST	Detects static code and operation vulnerabilities
Pre-release phase	DAST	Detects operation vulnerabilities
Production	Web Application Firewall (WAF) Runtime Application Self Protection (RASP)	Monitors and prevents attacks

Tabla 3. Medidas de seguridad proactivas en el SSDLC (Rich Mogull, Mike Rothman, 2024)

En definitiva, la CSA destaca la importancia de un enfoque de seguridad continuo en el desarrollo de software. Integrar herramientas como SAST, DAST, IAST y SCA en el SDLC no solo fortalece la seguridad del software, sino que también permite una

detección temprana de vulnerabilidades, optimizando el proceso de desarrollo y asegurando productos más robustos y seguros desde su concepción hasta su despliegue en producción.

2.2.3.Conclusiones

El modelado de amenazas y la integración temprana de la seguridad en el ciclo de desarrollo son pilares fundamentales en la construcción de aplicaciones seguras.

Adoptar el enfoque de "*Shift-Left*", como propone la Cloud Security Alliance (CSA), permite identificar, gestionar y mitigar riesgos potenciales desde las etapas iniciales del desarrollo, en lugar de afrontarlos en fases tardías o tras incidentes en producción. Este enfoque proactivo no solo mejora la seguridad del producto final, sino que también optimiza recursos al prevenir costosas correcciones posteriores.

La aplicación del modelado de amenazas, estructurado en cuatro pasos y apoyado en metodologías como STRIDE, facilita una cobertura integral de las posibles vulnerabilidades. Al integrar herramientas específicas como SAST, DAST, IAST y SCA desde el diseño hasta el despliegue, se fortalece la capacidad para detectar y neutralizar amenazas antes de que se materialicen. La guía de la CSA destaca la importancia de este enfoque continuo de seguridad, asegurando que cada fase del ciclo de vida del desarrollo de software esté alineada con los principios de seguridad.

En resumen, el modelado de amenazas y la integración temprana y continua de prácticas y herramientas de seguridad son esenciales para desarrollar y mantener aplicaciones resilientes frente a las amenazas actuales y futuras.

2.3 Herramientas de la Guía OWASP

2.3.1. Introducción

En el primer capítulo se introducía la *OWASP DevSecOps Guideline* (OWASP Foundation, 2024), definiéndola como un marco de trabajo para la integración de la seguridad en torno a tres áreas principales: personas, procesos y gobierno.

En esta sección se profundizará en el área de Procesos, y más específicamente en los segmentos de Código y de Construcción -siguiendo la terminología de la Guía-, puesto que es en estos dos segmentos donde entran en juego las herramientas AST, que son el objetivo de este proyecto.

Las actividades de cada uno de estos segmentos se estructuran de la manera siguiente:

Código

Este segmento aborda la gestión de secretos y la revisión de código antes de su integración, así como la detección temprana de vulnerabilidades de código fuente. Contempla las siguientes actividades:

- Pre-Commit: Implica la gestión de secretos y la revisión del código para asegurar que no contenga información sensible ni violaciones de seguridad.
- Análisis Estático (SAST): El análisis estático es una técnica de revisión de código que no requiere la ejecución del programa. Se utiliza para identificar errores de programación, violaciones de estándares de codificación, y vulnerabilidades de seguridad.
- Test Interactivo (IAST): Evaluación de aplicaciones en tiempo real durante el desarrollo para identificar vulnerabilidades sin necesidad de explotarlas.

Construcción (Build)

En este otro segmento las actividades se centran en la evaluación de la seguridad en etapas más avanzadas del ciclo de desarrollo, una vez que la aplicación se encuentra empaquetada y lista para su despliegue.

- Test Dinámico de la Aplicación (DAST): Técnica de "caja negra" que identifica vulnerabilidades en aplicaciones en ejecución.
- Test de Aplicaciones Móviles (MAST): Herramientas específicas como MobSF se utilizan para la seguridad de aplicaciones móviles.
- Test de Seguridad de las APIs o Servicios: Evaluación de la seguridad de las APIs utilizando herramientas como Postman con Apigee.
- Verificación de Configuraciones: Herramientas como AWS Config permiten asegurar que las configuraciones de infraestructura cumplan con las políticas de seguridad.

Las siguientes subsecciones tratarán sobre las herramientas que propone OWASP para los distintos usos o categorías.

2.3.2.Análisis estático de seguridad

Dentro del ciclo de desarrollo seguro de software en un entorno *DevSecOps*, el análisis estático de seguridad (Static Application Security Testing, SAST) juega un papel crítico en la identificación temprana de vulnerabilidades y en la fortificación de los componentes de software, antes de que estos sean desplegados en producción.

La sección de Análisis Estático de la OWASP *DevSecOps* Guideline cubre cuatro áreas principales:

- Escaneo y fortificación de contenedores
- Test estático del código propio (SAST)
- Análisis del código de terceros (Software Component/Composition Analysis, SCA)
- Escaneo de scripts de "Infrastructure as Code" (IaC).

Escaneo y fortificación de Contenedores

El escaneo de contenedores se realiza para identificar y corregir vulnerabilidades en las imágenes de contenedores que se usan durante la fase de compilación y antes de que estas sean enviadas a un registro de contenedores confiable. La fortificación de contenedores asegura que las imágenes son seguras y cumplen con las mejores prácticas recomendadas.

- **Capacidades del Escaneo de Contenedores**

  Para maximizar la seguridad en entornos basados en contenedores, las herramientas de escaneo ofrecen capacidades específicas que abordan desde la detección de configuraciones erróneas hasta la verificación de actualizaciones críticas. Estas capacidades son fundamentales para garantizar un entorno seguro y conforme a estándares:

  - Bibliotecas desactualizadas.
  - Contenedores configurados incorrectamente.
  - Sistemas operativos desactualizados.
  - Validaciones de cumplimiento.
  - Buenas prácticas sugeridas.

- **Herramientas Open-Source**

  Las siguientes herramientas Open-Source ofrecen funcionalidades adaptadas a diferentes necesidades, desde escaneos básicos hasta gestión avanzada de riesgos:

  - **Clair**: Análisis estático de vulnerabilidades para contenedores.
  - **Anchore**: Análisis profundo de imágenes Docker.
  - **Dagda**: Escaneo de vulnerabilidades y amenazas en contenedores Docker.
  - **Falco**: Detección de amenazas en tiempo de ejecución para Kubernetes.
  - **Harbor**: Registro de contenedores seguro y escaneo de imágenes.
  - **Trivy**: Escáner de vulnerabilidades y configuraciones erróneas.
  - **Kubescape**: Herramienta de seguridad para Kubernetes que incluye análisis de riesgos y cumplimiento.

- **Herramientas Comerciales**

  En el ámbito comercial, la *OWASP DevSecOps Guideline* recomienda la siguiente herramienta:

  - **Aquasec**: Escaneo avanzado de vulnerabilidades y gestión de riesgos en contenedores.

- **Buenas Prácticas para la Fortificación de Contenedores**

  OWASP recomienda las siguientes prácticas para la fortificación de contenedores:

  - Usar solo imágenes base confiables y mantenerlas actualizadas.
  - Aplicar parches de seguridad de manera regular.
  - Implementar el principio de mínimo privilegio en la configuración de contenedores.
  - Definir limitaciones de recursos para evitar ataques de agotamiento.
  - Usar escáneres de vulnerabilidades de imágenes y configurar la red de forma segura.
  - Endurecer los sistemas host y aplicar medidas de seguridad en el *runtime* de contenedores.
  - Monitorizar la actividad de los contenedores y asegurar los registros de contenedores.

Test Estático del Código Propio (SAST)

El análisis estático del código fuente (SAST) es una técnica fundamental que permite identificar vulnerabilidades y errores en el código sin necesidad de ejecutarlo. Es parte del proceso de revisión de código y se utiliza para detectar violaciones de sintaxis, vulnerabilidades de seguridad, errores de programación, violaciones de estándares de codificación, y valores indefinidos.

- **Herramientas Open-Source**

  El ecosistema de herramientas SAST incluye tanto soluciones de código abierto como comerciales, cada una con características únicas para adaptarse a diferentes lenguajes y necesidades de desarrollo. A continuación, se listan las de código abierto:

  - **CodeSec by Contrast** Security: Herramienta SAST gratuita y de alta velocidad, diseñada para desarrolladores e integración en CI/CD.
  - **SonarQube**: Herramienta web-based que soporta más de 20 lenguajes y permite la integración de plugins.
  - **Brakeman**: Escáner de vulnerabilidades específico para aplicaciones Ruby on Rails.
  - **CodeQL**: Motor de análisis semántico líder para descubrir vulnerabilidades en bases de código.

- **Herramientas Comerciales**

  En el ámbito comercial, la oferta es muy amplia:

  - **Veracode**: Herramienta SAST basada en el modelo SaaS, centrada en la seguridad del código.
  - **Security Code Scan**: Detector de patrones de vulnerabilidades para C# y VB.NET.
  - **Enlightn**: Escáner de vulnerabilidades para aplicaciones PHP Laravel.

- **Inquisition**: Conjunto de herramientas para análisis técnico de aplicaciones web construidas con Ruby y Ruby on Rails.
- **CodeSweep**: Herramienta de análisis estático para GitHub, gratuita y capaz de escanear código en más de 20 lenguajes.
- **HCL AppScan on Cloud**: Herramienta SAST como servicio que también realiza SCA e IaC.
- **Semgrep**: Análisis estático ligero para múltiples lenguajes.
- **Checkmarx SAST** y **Fortify**: Escáneres de vulnerabilidades de seguridad en el código.
- **PT Application Inspector**: Analizador de código fuente que proporciona herramientas para confirmar vulnerabilidades y facilita la colaboración entre especialistas en seguridad y desarrolladores.

Análisis del Código de Terceros (Software Composition Analysis, SCA)

El análisis de la composición de software (SCA) es un proceso automatizado que gestiona la seguridad de componentes de terceros y open-source en el código. SCA ayuda a identificar componentes potencialmente vulnerables, previniendo riesgos significativos como los ataques a la cadena de suministro, que pueden explotar vulnerabilidades en la red de proveedores y componentes de software.

- **Herramientas Open-Source**

  Las herramientas SCA, disponibles en versiones gratuitas y comerciales, ofrecen funcionalidades que facilitan la identificación de vulnerabilidades y el cumplimiento de licencias en los componentes de terceros utilizados en el desarrollo. A continuación, las opciones de código abierto recomendadas por OWASP:

  - **OWASP Dependency-Track:** Plataforma de análisis de componentes que permite identificar y reducir riesgos en la cadena de suministro de software.
  - **OWASP Dependency-Check:** Herramienta SCA que soporta múltiples lenguajes como Java, .NET, y JavaScript.
  - **OWASP CycloneDX:** Formato estándar para SBOM con generadores compatibles.
  - **OWASP dep-scan:** Herramienta de auditoría basada en vulnerabilidades conocidas, que genera documentos SBOM y CSAF.
  - **RetireJS:** Herramienta específica para analizar dependencias JavaScript.
  - **Safety:** Herramienta de verificación de dependencias Python.
  - **bundler-audit:** Verificación de parches para Bundler en Ruby.

- **Herramientas Comerciales**

  En el ámbito comercial, OWASP destaca las siguientes herramientas:

  - **Hakiri**: Herramienta comercial para la verificación de dependencias en proyectos Ruby y Rails.
  - **HCL AppScan on Cloud**: Herramienta que combina SAST, SCA y escaneo de IaC.
  - **Snyk**: Herramienta SCA ofrecida como solución SaaS.
  - **WhiteSource**: Identifica todos los componentes open-source en el software y asegura la protección contra vulnerabilidades.
  - **Synopsys BlackDuck**: Permite la gestión automatizada de políticas y el cumplimiento de licencias a lo largo del ciclo de vida del desarrollo de software.

Escaneo de Scripts de "Infrastructure as Code" (IaC)

El escaneo de scripts IaC es una técnica que verifica el código utilizado para configurar y gestionar infraestructuras, buscando problemas de seguridad y errores antes del despliegue de la infraestructura. Esto permite asegurar que la infraestructura cumpla con las normas de seguridad y políticas de la empresa desde el principio.

- **Herramientas (Open-Source) para el escaneo de scripts IaC:**

  Las herramientas disponibles para el escaneo de IaC permiten detectar configuraciones erróneas, vulnerabilidades y problemas de cumplimiento en scripts que gestionan infraestructuras, proporcionando un nivel adicional de seguridad en entornos *DevSecOps*:

  - **Checkov**: Prevención de configuraciones erróneas en la nube durante la fase de compilación.
  - **ansible-lint**: Verificador de buenas prácticas para Ansible.
  - **puppet-lint**: Verificación del cumplimiento del estilo de guía en Puppet.
  - **tfsec**: Escáner de seguridad para Terraform.
  - **terrascan**: Detección de violaciones de cumplimiento y seguridad en IaC.
  - **tflint**: Linter de Terraform con capacidad de integración.
  - **Trivy**: Escáner con políticas integradas para detectar problemas de configuración en Docker, Kubernetes, Terraform y CloudFormation.
  - **KICS**: Herramienta (de Checkmarx) para encontrar vulnerabilidades y problemas de cumplimiento en IaC.

En conjunto, el análisis estático de seguridad es la piedra angular de la práctica de *DevSecOps*, proporcionando un enfoque robusto para la identificación y mitigación de vulnerabilidades en el código y los componentes de software antes de que entren en producción.

La correcta implementación de estas herramientas y prácticas puede ayudar a proteger las aplicaciones de amenazas tanto internas como externas, mejorando notablemente la seguridad general del ciclo de vida del desarrollo de software.

2.3.3.Test Interactivo (IAST)

El Test Interactivo de Seguridad de Aplicaciones (IAST, por sus siglas en inglés) es una solución avanzada para la detección de vulnerabilidades en aplicaciones durante su ejecución. Esta metodología combina características de análisis estático y dinámico, lo que permite obtener una visión completa de los posibles problemas de seguridad, tanto desde la perspectiva del código como de su comportamiento en tiempo real.

A continuación, se detallan su funcionamiento, características principales, los datos que monitoriza y las herramientas más destacadas que ofrecen soporte para esta tecnología.

Funcionamiento del IAST

El IAST se instala en los servidores de aplicaciones y APIs, donde monitoriza y analiza continuamente el comportamiento de la aplicación en ejecución. Esto se logra

mediante la instrumentación del código con sensores que observan directamente el comportamiento relevante para la seguridad.

Características del IAST

El IAST aporta beneficios únicos que lo distinguen de otras metodologías de análisis de seguridad. Desde su capacidad para integrarse en los flujos de trabajo del desarrollo hasta su optimización para pruebas rápidas y precisas, esta tecnología ofrece una solución robusta para abordar los riesgos de seguridad en entornos modernos.

Las siguientes características ilustran su enfoque innovador:

- Instrumentación del código: IAST funciona al integrar sensores directamente en el código de la aplicación. Estos sensores monitorizan en tiempo real el comportamiento de la aplicación, detectando actividades sospechosas o vulnerabilidades a medida que ocurren.
- Optimización y velocidad: Utiliza técnicas probadas similares a las utilizadas por herramientas de gestión de rendimiento de aplicaciones (APM) y perfiles, pero optimizadas para la seguridad y con un enfoque en la rapidez y eficiencia.
- Sin Impacto en el desarrollo: No requiere cambios en la forma en que los equipos construyen, prueban o despliegan código. Esto significa que las pruebas de seguridad se pueden realizar en tiempo real durante el desarrollo normal y las pruebas, sin necesidad de explotar las vulnerabilidades identificadas.
- Implementación versátil: El IAST se puede desplegar en servidores de desarrollo, en pipelines de CI/CD, en servidores de aseguramiento de la calidad, e incluso en entornos de producción. Es particularmente eficaz para las pruebas de seguridad de APIs, superando los desafíos que enfrentan SAST y DAST con el código y los datos complejos de las APIs.

Acceso y Datos que monitoriza el IAST

Los sensores IAST tienen acceso a:

- Código Completo: Permite un análisis exhaustivo de la aplicación.
- Peticiones y Respuestas HTTP Completas: Monitoreo de las interacciones de la aplicación con el usuario y el entorno.
- Flujo y Control de Datos: Rastreo detallado de cómo los datos se mueven y se procesan dentro de la aplicación.
- Datos de Configuración: Evaluación de cómo la configuración puede afectar la seguridad.
- Librerías y *Frameworks*: Análisis de cómo se utilizan y su impacto en la seguridad.
- Conexiones de Back-End: Monitoreo de las interacciones de la aplicación con bases de datos y otros servicios de back-end.

Herramientas IAST (comerciales)

El mercado ofrece varias soluciones comerciales para la implementación de IAST, entre las que se incluyen:

- **Contrast Assess y Contrast Community Edition**: Ofrecen una integración profunda en el ciclo de vida del desarrollo de software, permitiendo pruebas de seguridad continuas y detalladas.
- **Checkmarx Interactive Application Security Testing (CxIAST)**: Proporciona capacidades avanzadas para la identificación de vulnerabilidades en tiempo real dentro de las aplicaciones en ejecución.

- **Seeker Interactive Application Security Testing**: Enfocado en la detección precisa de vulnerabilidades en aplicaciones dinámicas y complejas, proporcionando una visión detallada de los riesgos.
- **HCL AppScan on Cloud**: Ofrece un enfoque integral para la seguridad de aplicaciones, incluyendo capacidades de IAST, SAST y DAST, adaptándose a diversos entornos y requisitos de seguridad.

2.3.4.Test Dinámico de la Aplicación (DAST)

El Test Dinámico de la Aplicación (DAST) es una técnica de prueba de seguridad que se realiza en aplicaciones en ejecución, utilizando un enfoque de "caja negra". Esto significa que el análisis se lleva a cabo sin conocimiento del código fuente de la aplicación, lo que permite detectar vulnerabilidades desde la perspectiva de un atacante externo. DAST es esencial para identificar una amplia gama de problemas de seguridad, especialmente aquellos relacionados con la interacción del usuario y la configuración del servidor.

Funcionamiento del DAST

El DAST opera inyectando cargas útiles maliciosas en una aplicación en funcionamiento para identificar vulnerabilidades y debilidades potenciales. Este tipo de prueba es particularmente efectivo para encontrar problemas como:

- Validación de entrada/salida: Verificación de que la aplicación maneja correctamente las entradas y salidas, previniendo ataques como inyecciones de SQL y XSS.
- Problemas de autenticación: Detección de fallos en los mecanismos de autenticación que podrían permitir el acceso no autorizado.
- Errores de configuración del servidor: Identificación de configuraciones inseguras que podrían ser explotadas por un atacante.

Características del DAST

Las herramientas DAST se caracterizan por su capacidad para analizar aplicaciones en ejecución desde una perspectiva externa, simulando el comportamiento de un atacante. Estas soluciones ofrecen una visión integral de las vulnerabilidades relacionadas con la interacción del usuario, la configuración del servidor y el flujo de datos en la aplicación.

A continuación, se describen las principales características que hacen del DAST una técnica necesaria en los entornos de seguridad modernos.

- Escaneo Extensivo: Las herramientas DAST pueden realizar escaneos exhaustivos desde el lado del cliente y del servidor, sin requerir acceso al código fuente o conocimiento del *framework* sobre el cual está construida la aplicación.
- Interacción Mínima del Usuario: Aunque la configuración de las herramientas DAST puede requerir experiencia, una vez configuradas, los escaneos suelen requerir mínima interacción por parte del usuario y pueden ejecutarse como parte de escaneos nocturnos automatizados.
- Tipos de Herramientas: Entre las herramientas DAST más comunes se incluyen los escáneres de seguridad dinámica, los *fuzzer* (herramientas que inyectan datos aleatorios para encontrar fallos) y los *proxies* de ataque.

Herramientas Open-Source

Las herramientas DAST de código abierto ofrecen soluciones accesibles y flexibles para realizar pruebas dinámicas de seguridad. Estas herramientas suelen ser ideales para proyectos que requieren un enfoque económico, permitiendo realizar evaluaciones exhaustivas de la seguridad de aplicaciones web en ejecución.

A continuación, se presentan las opciones recomendadas por OWASP en esta categoría, ampliamente utilizadas en la comunidad de ciberseguridad.

- **ZED Attack Proxy (ZAP)**: Desarrollado por OWASP, ZAP es una de las herramientas open-source más populares para realizar pruebas de seguridad. Proporciona un entorno completo para escanear y encontrar vulnerabilidades en aplicaciones web.
- **OWASP Nettacker**: Una herramienta automatizada de recopilación de información y escaneo de vulnerabilidades que incluye una interfaz API y Web UI.

Herramientas Comerciales

En el ámbito comercial, las herramientas DAST destacan por su capacidad para escanear aplicaciones complejas a gran escala, ofreciendo soporte técnico y características avanzadas.

A continuación, se describen las herramientas comerciales recogidas en la *OWASP DevSecOps Guideline*:

- **Acunetix**: Un escáner automático de seguridad web que audita de manera precisa todas las aplicaciones web, incluyendo HTML5, JavaScript y aplicaciones de página única (SPA).
- **Netsparker**: Capaz de identificar vulnerabilidades en todo tipo de aplicaciones web modernas, independientemente de la arquitectura o plataforma subyacente.
- **InsightAppSec (AppSpider)**: Una herramienta para pruebas de seguridad en la web moderna, capaz de realizar escaneos dinámicos en aplicaciones complejas.
- **Veracode Dynamic Analysis**: Ofrece análisis dinámico a escala, ayudando a las empresas a escanear sus aplicaciones web en busca de vulnerabilidades explotables.
- **Burp Suite**: Plataforma integrada para realizar pruebas de seguridad en aplicaciones web. Sus herramientas trabajan de manera conjunta para mapear y analizar la superficie de ataque de una aplicación y encontrar y explotar vulnerabilidades de seguridad.
- **HCL AppScan on Cloud**: Herramienta DAST construida como un servicio, capaz de escanear tanto aplicaciones públicas como privadas. Puede explorar y probar aplicaciones web modernas, manejar escenarios complejos de inicio de sesión y aprovechar pasos manuales grabados.
- **Nuclei**: Escáner de vulnerabilidades rápido y personalizable basado en un simple lenguaje de especificación (DSL) basado en YAML.
- **Dastardly**: Escáner de seguridad web ligero, diseñado para ejecutarse en pipelines de CI/CD.

2.3.5.Conclusiones

A lo largo de esa sección, se han explorado las herramientas de Test de Seguridad de Aplicaciones bajo la perspectiva de OWASP, estructuradas en cuatro enfoques

clave: el Análisis Estático de Seguridad (**SAST**), el Análisis de Composición de Software (**SCA**), el Test Interactivo de Seguridad de Aplicaciones (**IAST**) y el Test Dinámico de Aplicaciones (**DAST**).

Cada uno de estos enfoques cumple un rol específico dentro del entorno de *DevSecOps*, proporcionando diferentes perspectivas y capacidades para asegurar la integridad y seguridad del software.

Mientras que SAST y SCA se centran en la seguridad del código y los componentes antes de que la aplicación entre en producción, IAST y DAST permiten análisis dinámicos. DAST se emplea típicamente en aplicaciones ya desplegadas, evaluando su comportamiento desde una perspectiva externa, mientras que IAST analiza la seguridad durante el proceso de desarrollo o pruebas, monitorizando la ejecución del código con acceso directo a su estructura interna.

Los anexos Matriz de herramientas propuestas por OWASP (por orden alfabético) y Matriz de herramientas propuestas por OWASP (por categoría) recopilan las 51 herramientas recogidas actualmente en la Guía de OWASP.

2.4 Evaluación de las herramientas

En esta sección se presenta el enfoque de un analista autorizado, como Gartner, quien para evaluar la eficacia de las diversas herramientas AST, destaca la importancia de considerar cómo las capacidades de estas herramientas se alinean y responden a los requerimientos específicos de cada caso de negocio o uso particular.

2.4.1. Casos de uso

En su análisis de capacidades críticas (Gartner, 2024) , Gartner distingue cinco casos de uso:

1. **Empresarial**: Este caso de uso considera las necesidades de organizaciones de gran escala, típicamente con una variedad de tipos de aplicaciones y enfoques de desarrollo. Requiere un enfoque comprensivo hacia la seguridad de aplicaciones que abarque todas las etapas del ciclo de desarrollo y múltiples tecnologías implicadas.
2. **Movilidad y cliente**: En este caso, los equipos de seguridad de aplicaciones se centran específicamente en probar aplicaciones en puntos finales, como dispositivos móviles, dentro de navegadores y otros escenarios similares. Es crucial para las organizaciones que priorizan la interacción directa con el cliente a través de aplicaciones accesibles públicamente.
3. **Seguridad de la cadena de suministro de software**: Este caso de uso enfatiza una combinación de funciones de seguridad de la cadena de suministro junto con las capacidades básicas de las herramientas AST. Es particularmente relevante para aquellas organizaciones que buscan asegurar la integridad del software recibido, utilizado y distribuido, en conjunto con un programa de seguridad de aplicaciones fundacional.

4. ***DevSecOps***: Este caso de uso resalta los requerimientos de organizaciones con una adopción significativa de *DevOps* y otras metodologías de desarrollo rápidas e iterativas. La integración de pruebas de seguridad en estos entornos ágiles y continuos es esencial para mantener la seguridad sin obstaculizar la velocidad de entrega.
5. **Aplicaciones cloud nativas**: Este caso de uso se enfoca en la prueba de seguridad para arquitecturas de aplicaciones y estilos de despliegue más modernos, incluyendo contenedores, APIs, microservicios y computación sin servidor. Es vital para organizaciones que emplean tecnologías de vanguardia y plataformas de nube para garantizar la seguridad en entornos dinámicos y altamente escalables. Este enfoque ayuda a abordar las particularidades de la seguridad en la nube, como la gestión de configuraciones, el control de acceso y la vigilancia continua, que son cruciales dado el modelo de amenazas en constante evolución en estos entornos.

Cada escenario tiene sus propias necesidades; es decir, la relevancia de cada capacidad o categoría de herramientas depende del caso de uso en el que se apliquen.

La tabla siguiente muestra el peso que Gartner asigna a las distintas capacidades AST en cada escenario:

Critical Capabilities	Enterprise	Cloud	DevSecOps	Mobile	Supply Chain
Static AST	15%	10%	10%	5%	10%
Dynamic AST	10%	5%	5%	5%	0%
Interactive AST	5%	10%	10%	5%	0%
Software Composition Analysis	15%	10%	10%	15%	30%
Mobile AST	5%	0%	5%	30%	0%
Infrastructure as Code	5%	15%	10%	0%	0%
Container Security Scanning	5%	15%	10%	0%	10%
Fuzzing	5%	0%	5%	5%	0%
API Testing and Discovery	5%	15%	10%	20%	0%
Software Supply Chain Security	10%	5%	5%	5%	35%
Application Security Posture Mgmt.	10%	5%	5%	5%	10%
Developer Enablement	10%	10%	15%	5%	5%

Tabla 4. Ponderación por casos de uso (Gartner, 2024)

Respecto a las capacidades SAST, DAST, IAST y SCA -en las que se centra este TFM-, el análisis de Gartner destaca el peso de **SAST** y **DAST** en los entornos empresariales, que representan el escenario más común y significativo en la práctica profesional. Estas capacidades son las más críticas en entornos empresariales grandes por su efectividad en la identificación y mitigación de vulnerabilidades a gran escala.

Por otro lado, **IAST** se revela como una capacidad importante en los casos de uso de aplicaciones nativas de la nube y *DevSecOps*, donde su valor radica en la integración y el análisis en tiempo real dentro de metodologías de desarrollo ágil y operaciones continuas.

Finalmente, la capacidad **SCA** juega un papel clave tanto en los entornos empresariales como en los casos de uso de aplicaciones móviles y de cliente. Su importancia radica en la prevalencia de dependencias y componentes de terceros. Sin embargo, SCA es especialmente crucial en la seguridad de la cadena de

suministro de software (Software Supply Chain Security), donde se le asigna un peso del 30%. Esto subraya la necesidad de esta capacidad para identificar y gestionar vulnerabilidades en componentes de terceros, lo cual es esencial para prevenir ataques y asegurar la integridad del software en cadenas de suministro, donde los riesgos de seguridad pueden tener graves consecuencias.

2.4.2.Criterios de evaluación

A continuación, se detallan los criterios de evaluación que Gartner aplica para valorar la eficacia y adecuación de las herramientas AST en sus diferentes categorías:

Criterios SAST

Las herramientas SAST se centran en el análisis estático del código fuente antes de su ejecución, lo que permite detectar vulnerabilidades y errores desde etapas tempranas del desarrollo. Sus criterios de evaluación incluyen:

- Capacidad de analizar múltiples lenguajes de programación y frameworks.
- Eficiencia en la detección de vulnerabilidades en el código fuente, bytecode o binario.
- Reducción de falsos positivos y capacidad de personalización según las prácticas de codificación.
- Facilidad de integración en el entorno de desarrollo y automatización de pruebas.
- Disponibilidad de SAST en la nube o como herramienta local.

Criterios DAST

Las herramientas DAST evalúan la seguridad de la aplicación en su estado de ejecución, simulando ataques y detectando vulnerabilidades visibles únicamente en tiempo de ejecución. Sus criterios clave son:

- Capacidad para simular ataques en aplicaciones en su estado de ejecución.
- Efectividad en la detección de vulnerabilidades en APIs y aplicaciones web.
- Capacidad para realizar pruebas sin acceso al código fuente (black box testing).
- Identificación de vulnerabilidades solamente visibles en tiempo de ejecución.
- Compatibilidad con aplicaciones modernas y entornos web complejos.

Criterios IAST

Las herramientas IAST combinan atributos de SAST y DAST, instrumentando la aplicación durante su ejecución para identificar vulnerabilidades mientras se realizan pruebas funcionales. Sus criterios son los siguientes:

- Capacidad para instrumentar la aplicación mientras se ejecuta y observar su comportamiento en tiempo real.
- Eficacia en la identificación de vulnerabilidades mientras el código es probado por otras herramientas.
- Capacidad para analizar vulnerabilidades tanto en el código como en el entorno operativo.
- Integración con otras herramientas de pruebas para ampliar la cobertura.

Criterios SCA

Las herramientas SCA se centran en la seguridad de dependencias y componentes de terceros, comprobando que el software no incluya bibliotecas vulnerables o con licencias inapropiadas. Sus criterios de evaluación abarcan:

- Capacidad para identificar componentes de código abierto y comerciales utilizados en el software.
- Eficiencia en la detección de vulnerabilidades conocidas en bibliotecas de terceros.
- Capacidad de gestionar riesgos de licencias y evaluar la seguridad de los componentes de código abierto.
- Provisión de guías claras para resolver vulnerabilidades.
- Integración con herramientas de desarrollo para verificar la seguridad en el proceso de incorporación de dependencias.

2.4.3.Ranking de fabricantes

Contexto

Basándose en los casos de uso y los criterios de evaluación definidos, se procede a analizar las capacidades de los principales fabricantes de herramientas AST según el informe de Gartner (Gartner, 2024).

Este análisis proporciona una comprensión de las fortalezas y debilidades de las soluciones líderes en el mercado. Sin embargo, la selección de las herramientas que serán objeto de estudio detallado en los capítulos posteriores también considerará factores prácticos, como su adecuación al contexto específico de este trabajo.

Puntuación de las capacidades

La tabla siguiente muestra la puntuación que Gartner otorga a las distintas capacidades que ofrece cada fabricante, en una escala de 1 a 5 y con acuerdo a los criterios especificados:

Herramientas	**SAST**	**DAST**	**IAST**	**SCA**
Snyk	3.9	1.0	1.0	4.3
Checkmarx	4.5	2.4	3.0	3.8
Contrast Security	3.0	2.7	3.9	3.9
GitHub	2.7	1.0	1.0	3.9
GitLab	3.0	3.5	1.0	3.6
HCLSoftware	4.2	4.2	3.5	3.2
Mend.io	3.4	1.0	1.0	3.9
Onapsis	3.0	1.5	1.0	1.3
OpenText	4.0	3.9	2.7	4.3
Sonatype	3.3	1.0	1.0	4.5

Synopsys	4.3	2.7	4.3	4.6
Veracode	4.3	3.0	2.0	4.1

Tabla 5. Puntuación de fabricantes por categoría

En esa escala de 1 a 5, cada valor tiene el significado que se indica a continuación:

1- Pobre o ausente: No se cubre ninguno o prácticamente ninguno de los requisitos especificados para la capacidad.
2- Regular: Algunos requisitos no se cumplen.
3- Bueno: Cumple con los requisitos.
4- Excelente: Cumple, y excede algunos requisitos.
5- Sobresaliente: Supera significativamente los requisitos.

Puntuación de fabricantes en el escenario empresarial

Teniendo en cuenta el peso que tiene cada capacidad en el caso de uso empresarial (SAST 15%, DAST 10%, IAST 5% y SCA 15% en el conjunto de las 12 capacidades, o 33%, 22%, 11% y 33% respectivamente, centrando el estudio en estas 4 capacidades), la valoración general de cada herramienta en este escenario es la siguiente:

Fabricante	SAST	DAST	IAST	SCA	*Promedio ponderado*
Synopsys	4,3	2,7	4,3	4,6	***4,0***
OpenText	4	3,9	2,7	4,3	***3,9***
HCLSoftware	4,2	4,2	3,5	3,2	***3,8***
Veracode	4,3	3	2	4,1	*3,7*
Checkmarx	4,5	2,4	3	3,8	*3,6*
Contrast Security	3	2,7	3,9	3,9	*3,3*
GitLab	3	3,5	1	3,6	*3,1*
Snyk	3,9	1	1	4,3	*3,1*
Sonatype	3,3	1	1	4,5	*2,9*
Mend.io	3,4	1	1	3,9	*2,8*
GitHub	2,7	1	1	3,9	*2,5*
Onapsis	3	1,5	1	1,3	*1,9*

Tabla 6. Puntuación ponderada en el escenario empresarial

Tal como se puede observar en la tabla de arriba, **Synopsys** es el fabricante que obtiene una mejor puntuación media (y es el más destacado en el MQ de Gartner (Mark Horvath, Magic Quadrant for Application Security Testing, 2024), aunque sin embargo, obtiene una puntuación inferior a 3 en la capacidad DAST, dado que no cumple con todos los requisitos establecidos para esa capacidad.

En segundo lugar figura **OpenText**, con el único handicap de su cobertura incompleta en los requisitos de la capacidad IAST.

El tercer lugar de este ranking lo consigue **HCLSoftware**, que, si bien no aparece en el cuadrante de líderes de Gartner debido a que no ofrece soluciones específicas para la protección contra ataques en la cadena de suministro del software, es el único fabricante que cumple todos los requisitos de las cuatro capacidades objeto de este estudio.

A continuación, se proporciona una visión general de la estrategia con la que estos tres fabricantes se posicionan en el mercado de AST, y de sus herramientas.

Estrategia de Synopsys

Synopsys (Synopsys, 2024) destaca por su puntuación general y por su liderazgo en el cuadrante mágico de Gartner.

La empresa ofrece una gama completa de soluciones que abarcan todas las áreas clave de pruebas de seguridad de aplicaciones:

- **Coverity** para SAST, que permite identificar vulnerabilidades en el código fuente durante las primeras etapas del desarrollo.
- **WhiteHat Dynamic** para DAST, una solución mejorada tras la adquisición de WhiteHat Security en junio de 2022, que proporciona pruebas de seguridad en entornos de producción para cualquier aplicación web.
- **Black Duck** para SCA, que ayuda a gestionar y asegurar el uso de componentes de código abierto y de terceros.
- **Seeker** para IAST, que combina la ejecución de la aplicación con el análisis de seguridad en tiempo real.
- **Polaris**, una plataforma basada en la nube que integra capacidades de AST, incluyendo las ofertas fAST Static y fAST SCA, proporcionando una solución unificada y escalable como servicio (SaaS).
- **Code Sight**, un complemento para entornos de desarrollo integrado (IDE) que facilita a los desarrolladores la detección y corrección de vulnerabilidades directamente desde su entorno de trabajo.

Estrategia de OpenText

Opentext (OpenText, 2024) ocupando el segundo lugar en el ranking, adquirió Fortify de Micro Focus en 2022, lo que le ha permitido fortalecer su oferta de AST, con una cartera de soluciones robustas, como:

- **Fortify** (SAST, DAST y SCA): Fortify proporciona capacidades amplias y maduras para el análisis estático y dinámico de seguridad. Fortify on Demand es una solución SaaS que permite a las empresas ejecutar pruebas de seguridad sin necesidad de infraestructura local.
- **Fortify WebInspect** (DAST): Una herramienta diseñada para detectar vulnerabilidades en aplicaciones web de forma automatizada, siendo especialmente útil en entornos ágiles y *DevSecOps*.
- **Fortify Software Composition Analysis** (SCA): Identifica riesgos en componentes de código abierto, brindando visibilidad sobre las vulnerabilidades en las dependencias de software.

Estrategia de HCLSoftware

HCLSoftware (HCLSoftware, 2024), a pesar de no aparecer en el cuadrante de "*Leaders*" de Gartner, sino en el de "*Challengers*", es el único fabricante que cumple con todos los requisitos de las cuatro capacidades analizadas.

Destaca por su conjunto de herramientas AppScan, que proporciona una experiencia de usuario unificada en diversas técnicas de pruebas de seguridad de aplicaciones. Ha lanzado una solución SCA propietaria y ha ampliado sus capacidades de análisis híbrido en su solución SAST.

Sus principales soluciones son:

- **AppScan** (SAST y DAST): AppScan es conocido por su capacidad de escanear aplicaciones de manera precisa, utilizando algoritmos de machine learning para reducir falsos positivos. Se integra bien en pipelines de CI/CD, ofreciendo versiones on-premise y en la nube.
- **AppScan on Cloud**: Una versión SaaS que facilita la integración y ejecución de pruebas de seguridad directamente desde la nube, permitiendo la escalabilidad y el acceso remoto para equipos distribuidos.
- **AppScan Enterprise** (IAST): Ofrece una profunda integración con herramientas de desarrollo y operaciones para proporcionar análisis en tiempo real mientras se ejecuta el código en entornos de desarrollo ágiles.

2.5 Herramientas seleccionadas

En el marco de este trabajo, es necesario distinguir claramente entre las herramientas seleccionadas por su relevancia y capacidad en entornos empresariales profesionales, y aquellas que resultan más prácticas y accesibles para ser utilizadas en un entorno de laboratorio con fines didácticos.

Esta diferenciación responde tanto a la diversidad de objetivos en ambos contextos como a la necesidad de adaptarse a las limitaciones técnicas y económicas presentes en cada escenario.

En el **entorno empresarial**, la elección de herramientas AST está guiada por criterios de eficacia, escalabilidad, soporte técnico y cumplimiento de normativas. Estas herramientas se seleccionan por su capacidad para integrarse en procesos empresariales complejos y, generalmente, requieren inversiones significativas en licencias e infraestructura técnica. La prioridad en este entorno es garantizar la seguridad a gran escala, mantener la calidad del software y mitigar riesgos que podrían tener repercusiones legales o económicas para la organización.

Por otra parte, en el **entorno de laboratorio**, especialmente dentro del contexto de este proyecto, se busca maximizar la eficiencia en la integración y la configuración de las herramientas, minimizando los costes y garantizando que sean accesibles para su implementación en escenarios de prueba con fines principalmente pedagógicos. Este enfoque permite centrarse en la parte práctica del aprendizaje, utilizando herramientas que, aunque puedan tener limitaciones en cuanto a funcionalidades avanzadas, son suficientes para ilustrar su funcionamiento y aportar valor didáctico.

2.5.1.Selección de la plataforma *DevSecOps*

Aunque este proyecto no contempla un análisis formal de las herramientas *DevSecOps* disponibles en el mercado, conviene aplicar ciertos criterios para seleccionar la plataforma que se utilizará en la implementación del ejercicio práctico.

Como apoyo para la toma de esta decisión, el informe de Gartner titulado "*How to Select DevSecOps Tools for Secure Software Delivery*" (Manjunath Bhat, 2023) proporciona un conjunto de criterios sobre las características que deben valorarse en dichas herramientas. Estas características se resumen en la siguiente imagen:

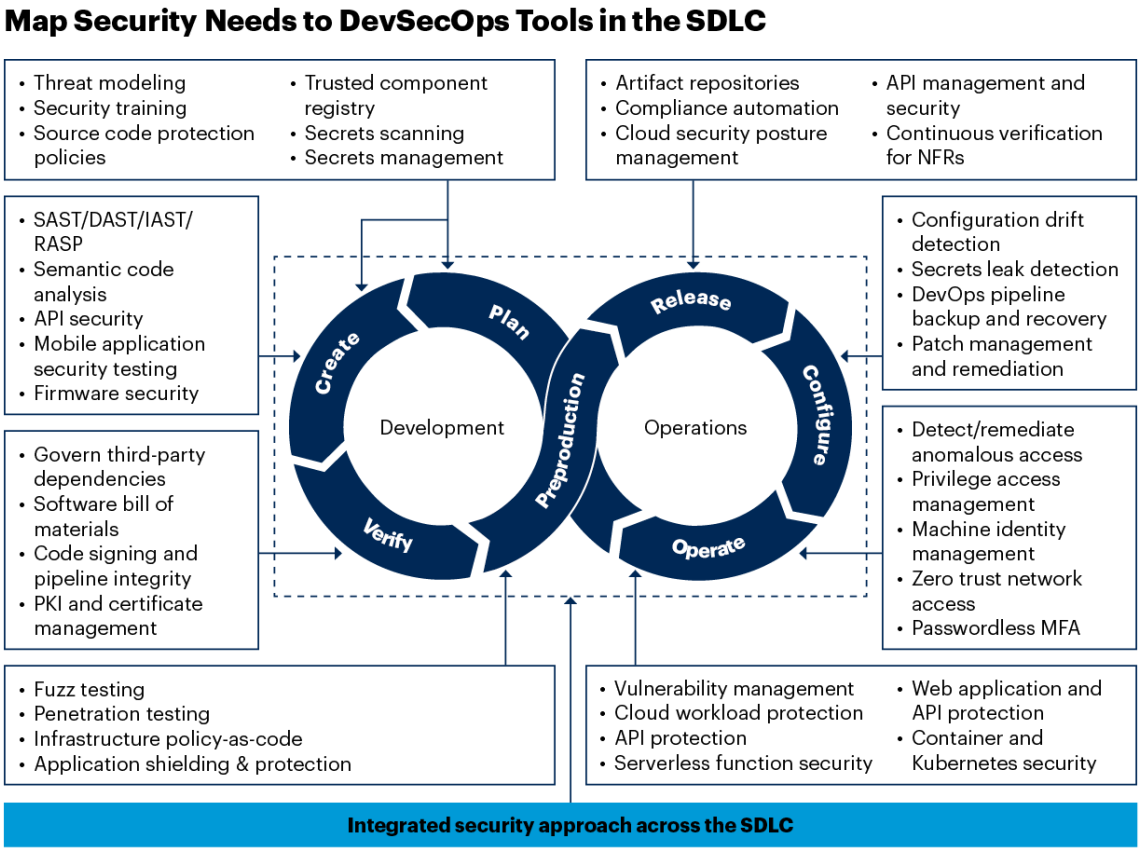


Ilustración 8. Necesidades de seguridad dentro del SDLC (Manjunath Bhat, 2023)

La siguiente tabla muestra un resumen de las herramientas mencionadas a modo de ejemplo en el informe, asociadas a las principales características *DevSecOps*:

Necesidades de seguridad en el SDLC	Ejemplos de herramientas que abordan la necesidad
Repositorios de artefactos	AWS CodeArtifact, Azure Artifacts, CloudRepo, Cloudsmith, **GitHub**, GitLab, Google Cloud Artifact Registry, Inedo, JFrog, Packagecloud, Sonatype
SCA	**GitHub**…
SAST/DAST	**GitHub**…

Escaneo de secretos	Amazon CodeGuru Reviewer, Bitbucket, BluBracket, Check Point (Spectral), Cycode, GitGuardian, **GitHub**, GitLab, Nightfall.ai, Opsera (GitCustodian), Palo Alto Networks (Prisma Cloud), Soteri
Análisis semántico de código	**GitHub** Advanced Security (Semmle), Snyk Code, Sonatype Lift (MuseDev)
Políticas de protección del código fuente	See branch protection policies in source code repositories, such as Bitbucket, **GitHub** and GitLab.
Registro de componentes confiables	**GitHub**, GitLab, Google Cloud Assured Open Source Software service, JFrog (Artifactory), Sonatype, Tidelift

Tabla 7. Ejemplos de herramientas DevSecOps por funcionalidad (Manjunath Bhat, 2023)

En la tabla anterior, se observa que GitHub aparece recurrentemente entre los ejemplos de herramientas que cubren las funcionalidades *DevSecOps*.

Es importante señalar que GitHub debe considerarse principalmente como un repositorio de código con capacidades de control de versiones, mientras que **GitHub Actions** es la herramienta nativa que GitHub integra para la automatización de flujos en el pipeline CI/CD.

Aunque existen otras herramientas comunes de automatización de tareas CI/CD de uso común que también se pueden utilizar con GitHub, como **Jenkins** o **GitLab CI/CD**, se ha optado por **GitHub Actions** por las siguientes razones:

- Integración nativa con GitHub (plataforma líder para el hospedaje de código): facilita la gestión del código y la automatización dentro de un único entorno, simplificando la operativa.
- Amplia comunidad y soporte: existen abundantes recursos en forma de documentación, tutoriales y ejemplos, además de una comunidad activa que facilita la resolución de problemas.
- Acceso a GitHub Education con el carnet universitario: GitHub Education ofrece beneficios adicionales, como **GitHub Copilot**, GitHub Codespaces, o un tiempo de ejecución de flujos prácticamente ilimitado, entre otros (GitHub Education, 2024).
- Integración nativa de herramientas SAST y SCA: **CodeQL** permite realizar análisis de código estático directamente en el repositorio, identificando las vulnerabilidades. **Dependabot** automatiza la actualización de dependencias y alerta sobre vulnerabilidades conocidas en librerías externas.
- **Plantillas YAML** para integrar otras herramientas de análisis de código: GitHub Actions ofrece plantillas y acciones predefinidas para integrar una amplia variedad de herramientas de análisis. El anexo Plantillas para la integración de herramientas AST en GitHub Actions muestra las herramientas actualmente disponibles a través de la integración con plantillas YAML.
- Respaldo corporativo: La adquisición de GitHub en 2018 por parte de **Microsoft**, es garantía para su continuidad sin comprometer su carácter Open Source, y también facilita la integración con servicios de Azure y otras soluciones Microsoft.

2.5.2.Selección de herramientas AST para el entorno empresarial

Esta sección se basa en el Ranking de fabricantes elaborado anteriormente, y tiene el propósito de destacar el fabricante mejor valorado en cada categoría, y señalar la herramienta que le otorga dicho reconocimiento.

Fabricante	SAST	DAST	IAST	SCA
Checkmarx	**4,5**	2,4	3	3,8
HCLSoftware	4,2	**4,2**	3,5	3,2
Synopsys	4,3	2,7	**4,3**	**4,6**
OpenText	4	3,9	2,7	4,3
Veracode	4,3	3	2	4,1
Contrast Security	3	2,7	3,9	3,9
GitLab	3	3,5	1	3,6
Snyk	3,9	1	1	4,3
Sonatype	3,3	1	1	4,5
Mend.io	3,4	1	1	3,9
GitHub	2,7	1	1	3,9
Onapsis	3	1,5	1	1,3

Tabla 8. Mejor fabricante por categoría

Mejor fabricante SAST: Checkmarx

Herramienta: Checkmarx SAST (Checkmarx, Checkmarx SAST, 2024)**.**

Sus características clave son:

- Alta precisión en la detección de vulnerabilidades: reduce significativamente los falsos positivos, optimizando el tiempo y esfuerzo en la remediación.
- Soporte para una amplia variedad de lenguajes de programación: permite analizar proyectos heterogéneos sin necesidad de herramientas adicionales.
- Integración fluida con entornos de desarrollo integrados (IDE) y pipelines CI/CD: facilita la adopción por parte de los desarrolladores y su incorporación en flujos de trabajo existentes.

Mejor fabricante DAST: HCLSoftware

Herramienta: HCL AppScan Standard (HCLSoftware, 2024)**.**

Características clave:

- Potente capacidad para identificar vulnerabilidades en aplicaciones web y móviles: cubre una amplia gama de tecnologías y plataformas.

- Análisis profundo que simula ataques reales: emula técnicas utilizadas por atacantes, aumentando la efectividad en la detección de vulnerabilidades.
- Interfaz intuitiva y fácil de usar: reduce la curva de aprendizaje y mejora la eficiencia del equipo de seguridad.

Mejor fabricante IAST: Synopsys

Herramienta: Seeker IAST (Synopsys, 2024).

Sus características clave son:

- Monitorización en tiempo real durante las pruebas funcionales: detecta vulnerabilidades mientras se ejecutan las pruebas, sin necesidad de escaneos adicionales.
- Alta precisión con mínimos falsos positivos: ofrece resultados fiables, ahorrando tiempo en la validación de las vulnerabilidades detectadas.
- Información detallada para facilitar la remediación: proporciona contexto y trazabilidad hasta el código fuente, ayudando a los desarrolladores a aplicar las correcciones con agilidad.

Mejor fabricante SCA: Synopsys

Herramienta: Black Duck (Synopsys, Black Duck SCA, 2024)

Características clave:

- Amplia base de datos de componentes de código abierto: mantiene una extensa y bien actualizada lista de vulnerabilidades y licencias de software.
- Detección de vulnerabilidades y gestión de licencias: ayuda a mitigar riesgos de seguridad y cumplimiento legal asociados al uso de software de terceros.
- Integración con múltiples herramientas y plataformas: se adapta fácilmente a diversos entornos de desarrollo y despliegue.

Ventajas de usar una misma suite de herramientas

Aunque es importante conocer cuál es la mejor herramienta en cada categoría, también se deben valorar las ventajas de optar por una *suite* integrada:

- Integración y compatibilidad: Las herramientas están diseñadas para trabajar juntas, lo que reduce problemas de compatibilidad y facilita la integración en el flujo de trabajo.
- Gestión centralizada: Unifica la administración de las herramientas, simplificando la monitorización y el mantenimiento.
- Soporte y actualización: Un único proveedor proporciona soporte técnico y actualizaciones, garantizando coherencia en el servicio.

- Eficiencia en costes: Puede resultar más rentable adquirir una suite completa que licencias individuales de diferentes proveedores, optimizando la inversión.

Analizando desde esta perspectiva, la *suite* que ofrece el mejor equilibrio entre todas sus capacidades es la de **HCLSoftware**, con altas puntuaciones en SAST, DAST e IAST, y una propuesta adecuada en SCA.

2.5.3.Selección de herramientas AST para el entorno de laboratorio

En esta sección se presentan herramientas accesibles desde el punto de vista económico y técnico, que facilitan la realización de ejercicios prácticos de seguridad en un entorno de laboratorio. Estas herramientas se han seleccionado por su facilidad de integración y por ser gratuitas o de bajo coste, sin comprometer la capacidad de enseñar los conceptos clave de las pruebas de seguridad de aplicaciones.

A continuación, se detallan las herramientas seleccionadas para cada categoría:

SAST: **CodeQL** (GitHub, Inc., 2024)

Se ha seleccionado CodeQL como herramienta SAST por los siguientes motivos:

- Integración nativa con GitHub: CodeQL está completamente integrado en GitHub, lo que facilita su configuración y uso sin necesidad de instalaciones adicionales o configuraciones complejas. Esto permite ejecutar análisis de código estático de manera automática en los repositorios de GitHub, optimizando el flujo de trabajo de los desarrolladores.
- Amplia cobertura de lenguajes y reglas predefinidas**:** Soporta múltiples lenguajes de programación, y cuenta con un amplio conjunto de reglas predefinidas que permiten detectar una extensa gama de vulnerabilidades.
- Personalización y comunidad activa: La herramienta permite crear consultas personalizadas en su propio lenguaje (QL) para adaptarse a necesidades específicas y cuenta con el respaldo de una comunidad activa que comparte conocimientos y actualizaciones, asegurando su evolución continua.

SCA: **Dependabot** (GitHub, Inc., 2024)

Se ha seleccionado Dependabot como herramienta SCA debido a las siguientes características diferenciales:

- Integración nativa con GitHub: Se integra directamente en los repositorios de GitHub, lo que permite la detección automática de vulnerabilidades en las dependencias y facilita su gestión sin necesidad de herramientas adicionales.
- Automatización de actualizaciones de dependencias: Genera automáticamente solicitudes de extracción (*pull-requests*) para actualizar las dependencias a versiones más seguras y recientes, agilizando el mantenimiento del proyecto.
- Alertas de seguridad: Proporciona notificaciones claras y detalladas sobre vulnerabilidades conocidas en las dependencias utilizadas y la configuración

puede ajustarse a las necesidades específicas de cada proyecto a través de archivos YAML.

DAST: *Fuera de alcance*

Se ha decidido excluir las pruebas DAST del alcance del proyecto para mantener un enfoque claro en la automatización de SAST y SCA, que representan modelos de integración más críticos y alineados con el objetivo principal de este TFM. Las pruebas DAST, al centrarse en aplicaciones desplegadas y no en el código fuente, no se integran de forma natural en el pipeline CI/CD ni en el repositorio de código, lo que las hace menos relevantes en este contexto. Además, las herramientas DAST no ofrecen una integración sencilla de sus resultados en la plataforma de gestión del proyecto, donde las vulnerabilidades se analizan y remedian como parte del ciclo de desarrollo. Por ello, se prioriza un enfoque más simple que destaque el impacto de la automatización en el ciclo de desarrollo.

IAST: *Fuera de alcance*

Las pruebas IAST también se excluyen del ejercicio práctico por motivos similares, debido a la falta de soporte para su integración en las plataformas *DevSecOps*.

2.6 Descripción del entorno de laboratorio

En la fase de implementación de este proyecto se creará un entorno que permita realizar pruebas de seguridad con herramientas AST integradas en un pipeline CI/CD. A continuación, se detalla el entorno y el pipeline de integración.

2.6.1. Entorno de desarrollo y ejecución

Se creará un repositorio en **GitHub.com**, donde se alojará el código de la aplicación sobre la que se realizarán las pruebas de seguridad. Este repositorio será la fuente central de control de versiones para el proyecto.

Además, se configurará un entorno local, que consistirá en una réplica del repositorio remoto. La gestión y sincronización del repositorio local con el repositorio remoto se realizará mediante **GitHub Desktop**.

La aplicación se instalará en un **servidor web** para comprobar su funcionamiento.

GitHub Actions estará configurado de forma que las herramientas de *test* se ejecuten en máquinas virtuales efímeras, dentro de un entorno Azure. En función de los resultados de la prueba, las herramientas propondrán la implantación de cambios a través de acciones *pull request* (PR).

El siguiente diagrama muestra los principales elementos del entorno de trabajo:

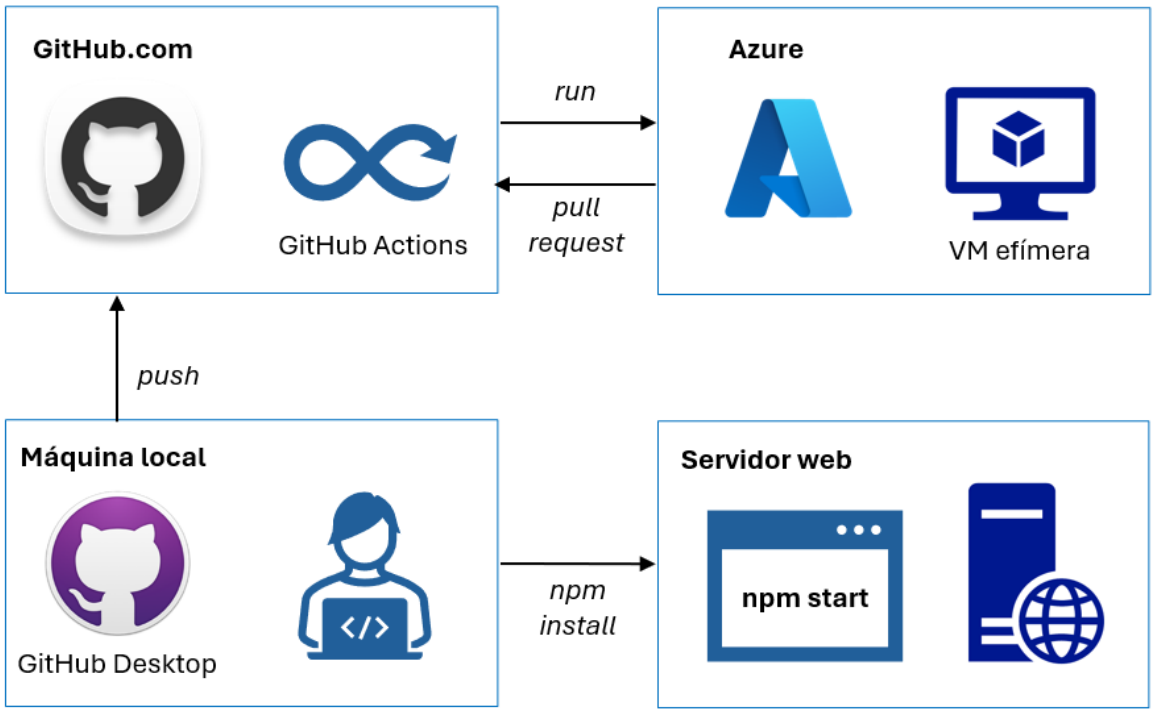


Ilustración 9. Entorno de laboratorio. Fuente: Elaboración propia

2.6.2. *Pipeline* CI/CD

Se configurará un *pipeline* de automatización, o ***workflow*** en términos de GitHub Actions, con 3 trabajos (***jobs***) diferenciados, uno para cada tipología de prueba. Esto es debido a que cada tipo de prueba requiere una herramienta distinta, y a que, además, debe realizarse en un momento distinto dentro del ciclo de vida de la aplicación, tal como se ha descrito en la sección Integración temprana de la seguridad en el ciclo de desarrollo.

Job 1 – SAST

Este trabajo se disparará automáticamente cada vez que se realice un ***push*** desde el repositorio local hacia el repositorio remoto, o un ***pull-request***. desde una rama de desarrollo a la rama principal.

Este paso ejecutará la aplicación **CodeQL**, que analizará el código fuente desplegándolo en una máquina virtual efímera. La función de esta máquina se conoce como "***runner***".

Los resultados del escaneo se almacenarán en un archivo con formato **SARIF** (Static Analysis Results Interchange Format), el cual será exportado automáticamente al repositorio de GitHub, permitiendo su visualización y análisis en la pestaña de seguridad del repositorio.

Job 2 – SCA

Este trabajo se diseñará para dispararse en dos situaciones específicas:

- Cuando que se realice un ***push*** o un ***pull-request*** de un **archivo de dependencias**. Es decir, no debe ejecutarse ante cualquier cambio de código, sino solamente cuando el desarrollador instala o actualiza alguna dependencia.
- De forma programada (vía ***cron***) en intervalos regulares, para comprobar que las librerías actualizadas siguen estando en vigencia y se consideran seguras.

Esto asegura que cualquier cambio en las dependencias sea analizado, minimizando el riesgo de incorporar vulnerabilidades a través de librerías de terceros, y también permite mantener las dependencias bajo control de forma continua.

Al activarse el flujo, se ejecutará la aplicación **Dependabot**.

El siguiente diagrama muestra de forma gráfica el *pipeline* descrito, a alto nivel:

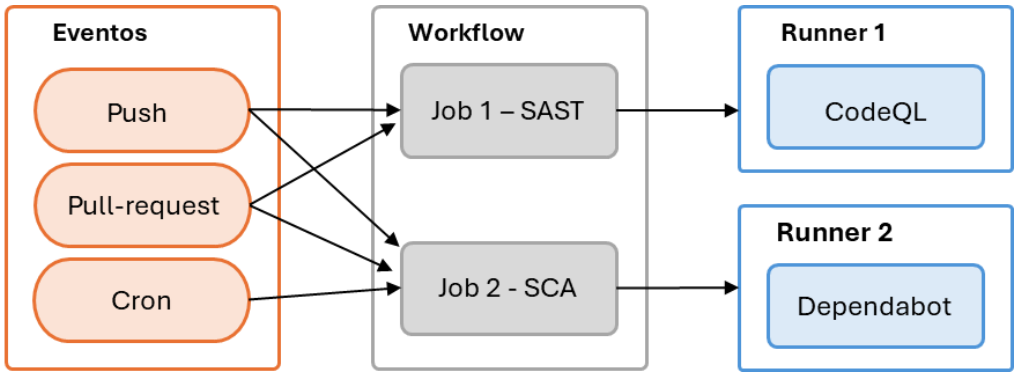


Ilustración 10. Pipeline CI/CD. Fuente: Elaboración propia

3. Fase de implementación

3.1 Introducción

Tras el amplio análisis de herramientas y metodologías realizado en la fase de investigación, donde se identificaron las soluciones AST más adecuadas y se profundizó en las prácticas óptimas para el desarrollo seguro de software, esta fase de implementación se centra en aplicar dichos conocimientos en un entorno real. El objetivo es materializar la integración de las herramientas seleccionadas dentro de un entorno *DevSecOps* avanzado, utilizando **GitHub Actions** como plataforma para automatizar y optimizar los procesos de seguridad en el ciclo de vida del desarrollo de software (CI/CD).

Si bien la fase previa permitió comprender las capacidades y beneficios de diversas herramientas AST, así como las estrategias para su selección y evaluación, en esta etapa se aborda específicamente la configuración y despliegue de flujos de trabajo automatizados que incorporen análisis de seguridad de manera eficiente y continua.

No se pretende comparar la calidad o eficiencia de las herramientas ni medir su rendimiento frente a otros productos del mercado, ya que existen recursos especializados, como informes de analistas y herramientas de *benchmarking* como OWASP Benchmark, que cumplen esa función (ver el anexo Introducción al OWASP Benchmark Project para más información).

En cambio, el enfoque se orienta a descubrir y documentar los mecanismos prácticos de integración de las herramientas AST seleccionadas, fortaleciendo la seguridad del proceso de desarrollo y proporcionando un modelo replicable en muchos proyectos.

Además, se busca que esta experiencia resulte útil para la comunidad de estudiantes y profesionales dedicados al desarrollo de software, proporcionando una guía práctica sobre la integración de herramientas AST en un entorno *DevSecOps*. Al documentar tanto los retos encontrados como las soluciones aplicadas, se espera facilitar el aprendizaje de los mecanismos de integración, contribuyendo a una comprensión más amplia y profunda de cómo implementar la seguridad en cada fase del desarrollo de software.

Este capítulo se organiza en secciones que describen, paso a paso, la configuración y el uso de las herramientas AST seleccionadas durante la fase de investigación para implementar este ejercicio de laboratorio.

El capítulo concluye con una sección que proporciona las pautas clave para integrar cualquier otra herramienta AST en el pipeline CI/CD, enfocándose en la habilitación de permisos de acceso y la configuración de los archivos YAML necesarios para la correcta ejecución de los flujos de trabajo en el entorno de GitHub Actions.

Para ilustrar esta capacidad, se utilizará la herramienta SAST de Semgrep, un fabricante que ofrece una versión gratuita (Community Edition), que se integra fácilmente en entornos CI/CD y permite a los desarrolladores definir y personalizar reglas de análisis, lo cual la convierte en una opción flexible y práctica para entornos de laboratorio.

3.2 Configuración de la plataforma *DevSecOps*

Antes de integrar las herramientas de seguridad en el ciclo de vida del desarrollo de software, es necesario establecer una plataforma *DevSecOps* adecuada.

Esta sección detalla los pasos para crear el repositorio en GitHub y configurar el entorno de ejecución local de la aplicación objetivo.

3.2.1.Selección de la aplicación objetivo

Para facilitar la verificación de los resultados proporcionados por las herramientas AST, es fundamental utilizar una aplicación diseñada específicamente con fines didácticos que, de forma intencionada, contenga una amplia variedad de vulnerabilidades. Con este criterio, se han evaluado las siguientes aplicaciones:

- OWASP Benchmark Project (OWASP Foundation, 2024)
- OWASP Juice Shop (OWASP Foundation, 2024)
- OWASP WebGoat (OWASP Foundation, 2024)

La elección de la aplicación se ha basado en criterios como la cobertura de vulnerabilidades, la facilidad de integración en un entorno *DevSecOps* y el uso de tecnologías modernas. Tras analizar estas opciones, se ha seleccionado **OWASP Juice Shop** por los siguientes motivos:

Cobertura integral de vulnerabilidades

Abarca todas las categorías del *OWASP Top Ten* (OWASP Foundation, 2024), lo que permite evaluar de manera amplia y completa la efectividad de las herramientas AST en distintos tipos de vulnerabilidades.

A diferencia de *OWASP Benchmark Project*, que está más enfocado en medir el rendimiento y precisión de las herramientas AST sobre un conjunto predefinido de vulnerabilidades, *OWASP Juice Shop* se asemeja más a una aplicación real con una variedad más rica de escenarios de ataque.

Por su parte, *OWASP WebGoat*, aunque también cubre muchas vulnerabilidades, está orientada a la enseñanza mediante ejercicios prácticos y no ofrece un entorno tan completo y automatizable para la integración de pruebas en un pipeline *DevSecOps*.

Facilidad de integración en un entorno DevSecOps

El código fuente de *OWASP Juice Shop* está alojado en GitHub, lo que permite importarlo sin complicaciones, facilitando la configuración automática del entorno *DevSecOps* seleccionado para el proyecto: GitHub Actions.

En comparación, *OWASP Benchmark Project* es menos adecuado para un entorno de integración continua (CI/CD), ya que su estructura se centra en la evaluación de herramientas específicas y no en la ejecución fluida dentro de un ciclo DevOps.

OWASP WebGoat, por su parte, está diseñada principalmente para enseñar a los usuarios a identificar y explotar vulnerabilidades de forma manual, y no se adapta fácilmente a flujos de trabajo automatizados como los que permite GitHub Actions.

Tecnología moderna

OWASP Juice Shop está construida utilizando tecnologías modernas como Angular y Node.js, proporcionando un entorno que refleja de manera realista las prácticas actuales de desarrollo, y la efectividad de las pruebas.

En cambio, tanto *OWASP WebGoat* como *OWASP Benchmark Project* están construidas con el SDK de Java 11 y pueden presentar limitaciones en la integración con las herramientas AST seleccionadas para este proyecto.

3.2.2.Creación del repositorio en GitHub

Para facilitar el acceso al código fuente y la gestión de flujos de trabajo automatizados, se importará el repositorio original del proyecto "OWASP Juice Shop" a GitHub.

En el anexo Importación del repositorio en GitHub se detalla el proceso de importación del repositorio, especificando la URL de origen y el nombre de repositorio en destino. También se destaca la ubicación de los flujos de automatización en la carpeta ".github/workflows", que han sido importados junto con la aplicación.

3.2.3.Configuración del entorno local de desarrollo y ejecución

Para gestionar de forma cómoda y eficiente tanto el código de la aplicación como los flujos de CI/CD, se utilizará **GitHub Desktop** como entorno de desarrollo local.

El anexo Configuración del entorno local de desarrollo y ejecución muestra el procedimiento para configurar Git en la máquina local y los pasos necesarios para instalar y ejecutar la aplicación, incluyendo el despliegue de un **servidor web** local.

Este proceso de instalación incluirá la generación de archivos de dependencias, en los que se documentan las versiones exactas de cada paquete necesario para el correcto funcionamiento de la aplicación. Estos archivos desempeñarán un papel fundamental más adelante, en el análisis SCA, al ser utilizados por las herramientas SCA para identificar vulnerabilidades conocidas en las dependencias y sugerir actualizaciones a versiones más seguras, garantizando así la seguridad de las bibliotecas externas que utiliza la aplicación.

3.3 Pruebas SAST

En esta sección se analizará el uso de **CodeQL**, la herramienta SAST que GitHub incorpora de forma nativa para la detección y gestión de vulnerabilidades de seguridad en el código propio. CodeQL no solo permite identificar vulnerabilidades en el código fuente, sino también gestionarlas de manera eficiente. Una vez detectadas, las vulnerabilidades pueden clasificarse y priorizarse: es posible descartarlas si se considera que son falsos positivos o representan un riesgo aceptable, asignarlas como tarea a un tercero (ya sea un desarrollador o un analista de seguridad) para su revisión o remediación, o proceder a corregirlas directamente.

Para facilitar esta remediación, CodeQL proporciona documentación detallada y ejemplos prácticos, que permiten entender y solucionar cada vulnerabilidad de forma manual. Además, en los casos en los que sea posible, **GitHub Copilot** se puede emplear como soporte automatizado, sugiriendo e incluso aplicando correcciones de forma automática. Esta combinación de herramientas simplifica el proceso de detección y corrección de vulnerabilidades, y agiliza el flujo de trabajo al ofrecer múltiples opciones de gestión adaptadas a las necesidades de cada proyecto.

GitHub ofrece una configuración predeterminada para integrar CodeQL en el pipeline, que se puede habilitar directamente desde el menú de configuración del repositorio, sin necesidad de crear manualmente un archivo YAML de configuración. Esta configuración predeterminada se dispara en eventos comunes como *push* y *pull-request*. Para usuarios que requieren personalización avanzada, GitHub facilita un asistente que permite modificar y adaptar determinados parámetros de CodeQL, como los lenguajes de programación que debe analizar, el conjunto de consultas (básicas o extendidas) y los eventos de ejecución.

3.3.1. Preparación de la herramienta

Para habilitar CodeQL en este proyecto, es necesario que el repositorio tenga visibilidad pública o que el usuario disponga de una licencia Enterprise en GitHub.

En el anexo Configuración de CodeQL se describe el procedimiento para cambiar la visibilidad del repositorio y los pasos necesarios para habilitar CodeQL en un repositorio público. Este proceso incluye la edición de su configuración para adaptar los parámetros de análisis según los requisitos del proyecto.

3.3.2. Ejecución del escaneo SAST

El proceso de integración del análisis SAST con CodeQL comienza desde el momento en que un desarrollador realiza cambios en el código fuente y continúa hasta que estos se integran en la rama principal del repositorio.

Para ello, se ha configurado CodeQL de manera que el escaneo se active automáticamente cada vez que se produzca un evento *push* o un *pull-request* en el repositorio. Este flujo combina acciones manuales y automáticas que garantizan la detección temprana de vulnerabilidades, su revisión y, en caso necesario, su corrección antes de incorporar los cambios definitivos.

El diagrama que se presenta a continuación ilustra este flujo de trabajo, destacando las interacciones clave entre los desarrolladores, el repositorio en GitHub y el pipeline de CI/CD. En él se muestra cómo CodeQL se integra de manera transparente en este

proceso, no solo para detectar y gestionar vulnerabilidades de seguridad, sino también para facilitar su remediación, asegurando así un desarrollo seguro y eficiente:

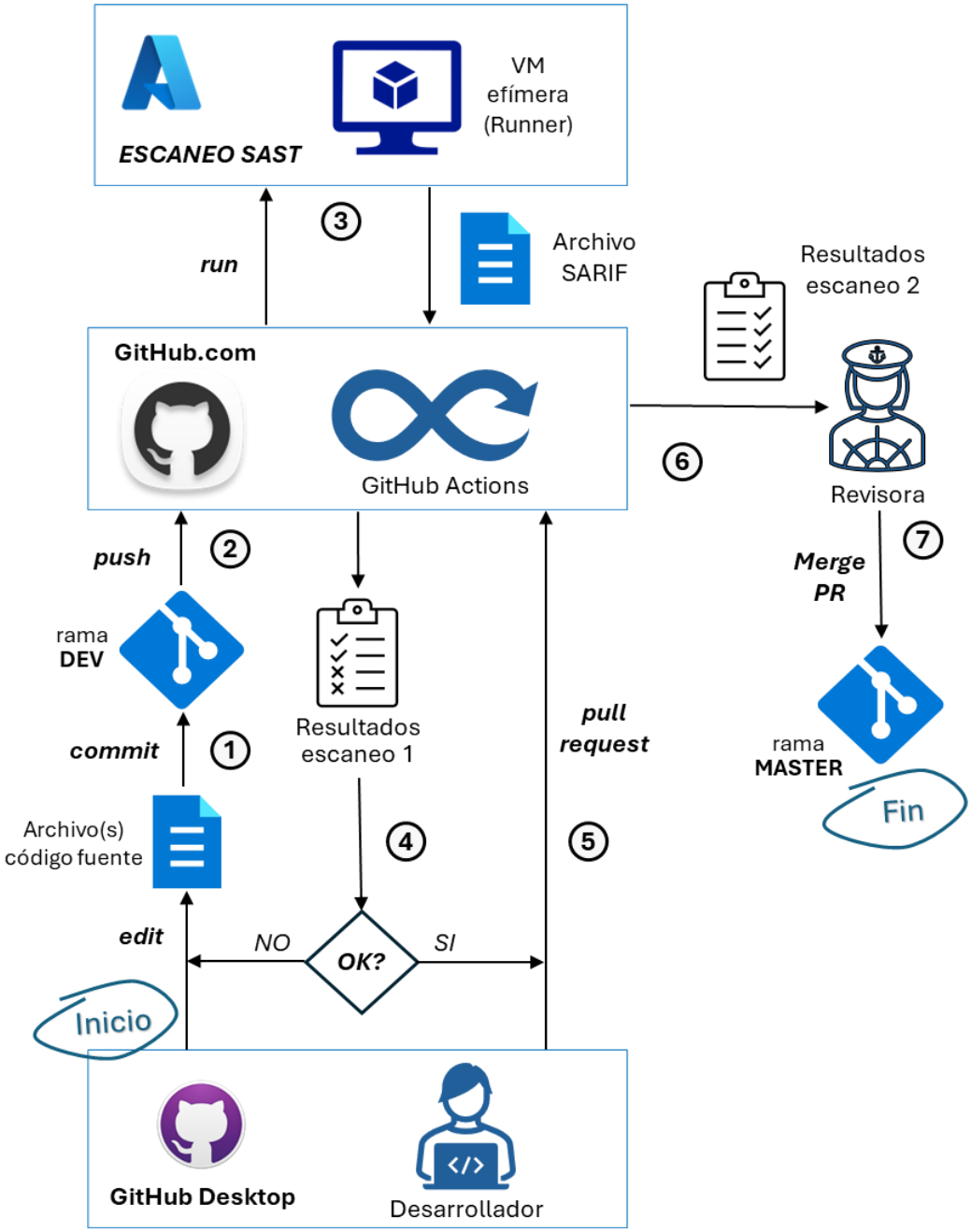


Ilustración 11. Flujo de trabajo en el escaneo SAST. Fuente: Elaboración propia

Paso 1: Edición y commit del código en el repositorio local

El desarrollador realiza cambios en uno o varios archivos de código fuente en la rama de desarrollo (**DEV**) de su repositorio local. Utilizando GitHub Desktop, realiza un ***commit***, incorporando la nueva versión de los archivos al historial de versiones en su máquina local.

Paso 2: Push al repositorio remoto (DEV)

Tras el *commit*, el desarrollador sincroniza los cambios con el repositorio remoto en GitHub mediante un ***push*** a la misma rama (**DEV**). Esto actualiza el repositorio en la nube con los cambios realizados localmente.

Paso 3: Ejecución del flujo de GitHub Actions (escaneo SAST tras el push)

La acción de ***push*** activa un flujo de GitHub Actions que ejecuta un escaneo SAST en la rama de desarrollo para verificar si el código contiene vulnerabilidades de seguridad. El escaneo se realiza en un entorno temporal virtualizado en Azure, donde se clona el repositorio. Durante este proceso, la herramienta SAST genera un archivo SARIF (Static Analysis Results Interchange Format) que contiene información detallada sobre todas las vulnerabilidades identificadas. Al finalizar el escaneo, el archivo SARIF se transfiere automáticamente a GitHub, permitiendo que los resultados se visualicen en la pestaña de Seguridad del repositorio.

El anexo Archivo SARIF: Función y estructura explica cómo acceder a este archivo a través de la interfaz de GitHub Actions.

Cabe señalar que en este paso pueden coexistir otras herramientas de test automático que se hayan configurado junto con CodeQL.

Paso 4: Revisión de los resultados del escaneo SAST

Si el escaneo SAST identifica vulnerabilidades, el desarrollador revisa los resultados en GitHub y realiza los ajustes necesarios en el código, repitiendo el ciclo de ***commit*** y ***push***. Si el escaneo no reporta problemas, el flujo de trabajo continúa sin interrupciones.

Paso 5: Creación de un pull request hacia la rama principal (MASTER)

El desarrollador crea un ***pull-request*** en GitHub, solicitando que los cambios realizados en la rama de desarrollo (**DEV**) se integren en la rama principal (**MASTER**). Este *pull-request* será revisado y aprobado antes de proceder con la integración.

Paso 6: Ejecución del flujo de GitHub Actions (escaneo SAST tras el pull request)

La creación del ***pull-request*** activa nuevamente el flujo de GitHub Actions, que ejecuta otro escaneo SAST sobre el código para asegurar que no se hayan introducido vulnerabilidades antes de fusionarse en la rama principal, y se generarán unos nuevos resultados, que estarán disponibles en GitHub.

Paso 7: Revisión y aprobación del pull request

Un revisor o revisora asignada verifica los cambios realizados y los resultados del escaneo SAST. Si todo es satisfactorio, aprobará el *pull-request*, realizando el ***merge*** hacia la rama **MASTER**. Con esta acción, los cambios se integrarán en la rama principal del repositorio remoto, **completándose así el flujo de trabajo**.

El anexo Experiencia de usuario en el escaneo SAST muestra la interacción del desarrollador con la interfaz gráfica de las herramientas que utiliza, desde que realiza y publica una modificación en el código fuente de la aplicación, hasta la ejecución del escaneo SAST y la correspondiente generación de alertas.

3.3.3.Análisis de las vulnerabilidades detectadas en el escaneo SAST

El análisis SAST realizado con CodeQL ha identificado diversas vulnerabilidades en el código fuente de la aplicación, cuyo detalle se encuentra en el anexo Gestión de vulnerabilidades con GitHub.

El anexo muestra cómo, a través de la pestaña “Security” del repositorio en GitHub, es posible acceder a una interfaz avanzada que permite visualizar y gestionar cada alerta. Además de mostrar una descripción detallada de la vulnerabilidad detectada y su localización en el código, esta interfaz ofrece varias opciones de gestión:

1- Descartar la alerta en caso de que se considere un falso positivo o un riesgo aceptable.
2- Crear una incidencia para asignar la remediación de la vulnerabilidad a un miembro del equipo.
3- Utilizar la función GitHub Copilot Autofix para aplicar correcciones de forma automática en los casos donde sea viable.
4- Acceder a información adicional en la base de datos de MITRE, donde se detallan las características y el tratamiento recomendado de la vulnerabilidad identificada mediante su código CWE.

El anexo también incluye un análisis detallado de un ejemplo específico de vulnerabilidad (CWE-843: Type Confusion), en la que se explora cómo un atacante podría aprovechar una debilidad en el tratamiento de parámetros HTTP. Esta vulnerabilidad se ilustra con capturas de pantalla de la herramienta, destacando las recomendaciones y las acciones de remediación propuestas por CodeQL y MITRE para abordarla eficazmente.

3.3.4.Remediación de vulnerabilidades con GitHub Copilot Autofix

GitHub Copilot Autofix es una herramienta que facilita la corrección automática de vulnerabilidades en el código fuente con el apoyo de la Inteligencia Artificial. Esta funcionalidad integrada en GitHub permite, además de detectar el código vulnerable, aplicar soluciones automáticas basadas en buenas prácticas de seguridad, en aquellos casos donde sea seguro y viable.

El anexo Caso de uso de Copilot Autofix ofrece una explicación detallada de esta funcionalidad, usando un ejemplo específico de vulnerabilidad detectada en la aplicación OWASP Juice Shop. La imagen en el anexo muestra cómo Copilot Autofix identifica una vulnerabilidad del tipo CWE-843 (*Type Confusion*) y sugiere una corrección automatizada. Esta corrección puede aplicarse directamente en la rama principal o en una rama secundaria, permitiendo al equipo de desarrollo gestionar la solución de manera flexible y ágil.

3.4 Pruebas SCA

En esta sección se analizará el uso de **Dependabot**, la herramienta SCA que GitHub incorpora de forma nativa para la detección y gestión de vulnerabilidades en las dependencias de terceros. Dependabot no solo permite identificar vulnerabilidades en las librerías y componentes utilizados por el proyecto, sino también gestionarlas de manera eficiente. Una vez detectadas, las vulnerabilidades pueden revisarse y priorizarse. Dependabot facilita la configuración de actualizaciones automáticas para dependencias vulnerables, la revisión manual de los cambios propuestos o incluso la posibilidad de ignorar ciertas alertas si se consideran falsos positivos o riesgos aceptables.

Para facilitar la remediación, Dependabot proporciona documentación detallada y enlaces a los informes de vulnerabilidades, permitiendo entender el impacto de cada problema y las versiones disponibles que lo corrigen. Además, cuando es posible, Dependabot genera automáticamente los *pull-requests* que actualizan las dependencias a versiones seguras, simplificando el proceso de mantenimiento y asegurando que el proyecto esté protegido contra vulnerabilidades conocidas.

GitHub ofrece una configuración predeterminada para integrar Dependabot en el repositorio, que se puede habilitar directamente desde el menú de configuración sin necesidad de crear manualmente un archivo de configuración. Esta integración permite que Dependabot monitorice continuamente las dependencias y notifique sobre cualquier vulnerabilidad detectada. Para desarrolladores que requieren personalización avanzada, GitHub facilita la edición de un archivo dependabot.yml, en el que es posible modificar parámetros como la frecuencia de actualización, los registros de paquetes que se deben monitorizar y las ramas del repositorio que se utilizarán para las actualizaciones.

3.4.1. Preparación de la herramienta

Al estar integrada de forma nativa en GitHub, para activar Dependabot basta con acceder al menú de configuración del repositorio, dirigirse a la sección “Code security” y habilitar la opción “Dependabot alerts”. El anexo Configuración de Dependabot muestra esta página y detalla todas las características de personalización que aporta la herramienta para gestionar las alertas de una manera eficiente, a la medida de las necesidades del proyecto.

Para proyectos que requieren configuraciones avanzadas, las herramientas SCA generalmente permiten establecer opciones mediante archivos YAML ubicados en la carpeta .github/workflows/ del repositorio.

Estos archivos permiten especificar detalles como la frecuencia de actualización, las ramas en las que se aplicarán las actualizaciones, y los nombres de los archivos de manifiesto que contienen las dependencias de terceros.

Las dependencias que estas herramientas SCA monitorizan son las que están definidas en los archivos de manifiesto o “*lockfiles*”, cuya estructura depende del gestor de paquetes utilizado en el proyecto. Entre los más comunes se encuentran:

- npm: package-lock.json y package.json
- Yarn: yarn.lock
- Python (pip): requirements.txt
- Pipenv (Python): pipfile y pipfile.lock
- Maven (Java): pom.xml

A continuación, se presenta un extracto del código YAML necesario para especificar los archivos de manifiesto en un escaneo SCA:

```
- name: Escaneo SCA
  run: |
    <sca-tool> scan --sarif --output=sca.sarif --config=auto \
      --include="**/*.lock" \
      --include="**/package-lock.json" \
      --include="**/requirements.txt" \
      --include="**/pom.xml"
```

3.4.2.Ejecución del escaneo SCA

Dado que el análisis SCA se centra en verificar la seguridad y la vigencia de las librerías de terceros que utiliza el proyecto, no es necesario ejecutar el escaneo ante cualquier cambio en el código, sino solamente cuando se instala o actualiza alguna dependencia.

Adicionalmente, para asegurar que las dependencias siguen siendo seguras y están actualizadas, es conveniente realizar el escaneo de manera programada, incluso cuando no haya cambios en los archivos de manifiesto.

Para una gestión eficiente del escaneo, el workflow distingue entre los archivos cuyos cambios deben activar el proceso (“***paths***”) y los archivos que serán objeto del análisis (“***include***”). Aunque en este caso ambos tipos de archivos suelen coincidir, conceptualmente son objetos distintos, y por ello se gestionan con cláusulas diferentes.

A continuación, se muestra un ejemplo de configuración del archivo YAML que especifica tanto los eventos como los archivos que disparan el *workflow*:

```
name: Escaneo SCA
on:
  push:                 # Se dispara cuando se hace push de cualquiera de los…
    paths:              # …archivos de dependencias
      - 'package-lock.json'
      - 'package.json'
      - 'yarn.lock'
      - 'requirements.txt'
      - 'pom.xml'
  pull_request:         # Se dispara en pull-requests que incluyen cambios en…
    paths:              # …cualquier archivo de dependencias
      - 'package-lock.json'
      - 'package.json'
      - 'yarn.lock'
      - 'requirements.txt'
      - 'pom.xml'
  schedule:
    - cron: '0 2 * * *'     # Se dispara automáticamente todos los días a las 2:00 UTC
```

El diagrama que se presenta a continuación describe el flujo de trabajo implementado para integrar el análisis SCA con Dependabot en el ciclo de desarrollo de software. Este flujo se centra en la gestión de dependencias de terceros, destacando las etapas clave desde la actualización o instalación de una librería hasta la verificación automática de su seguridad mediante GitHub Actions. Los recuadros sombreados señalan las diferencias respecto al flujo SAST, como la incorporación de escaneos programados, que garantizan la seguridad continua del proyecto frente a vulnerabilidades conocidas en librerías externas:

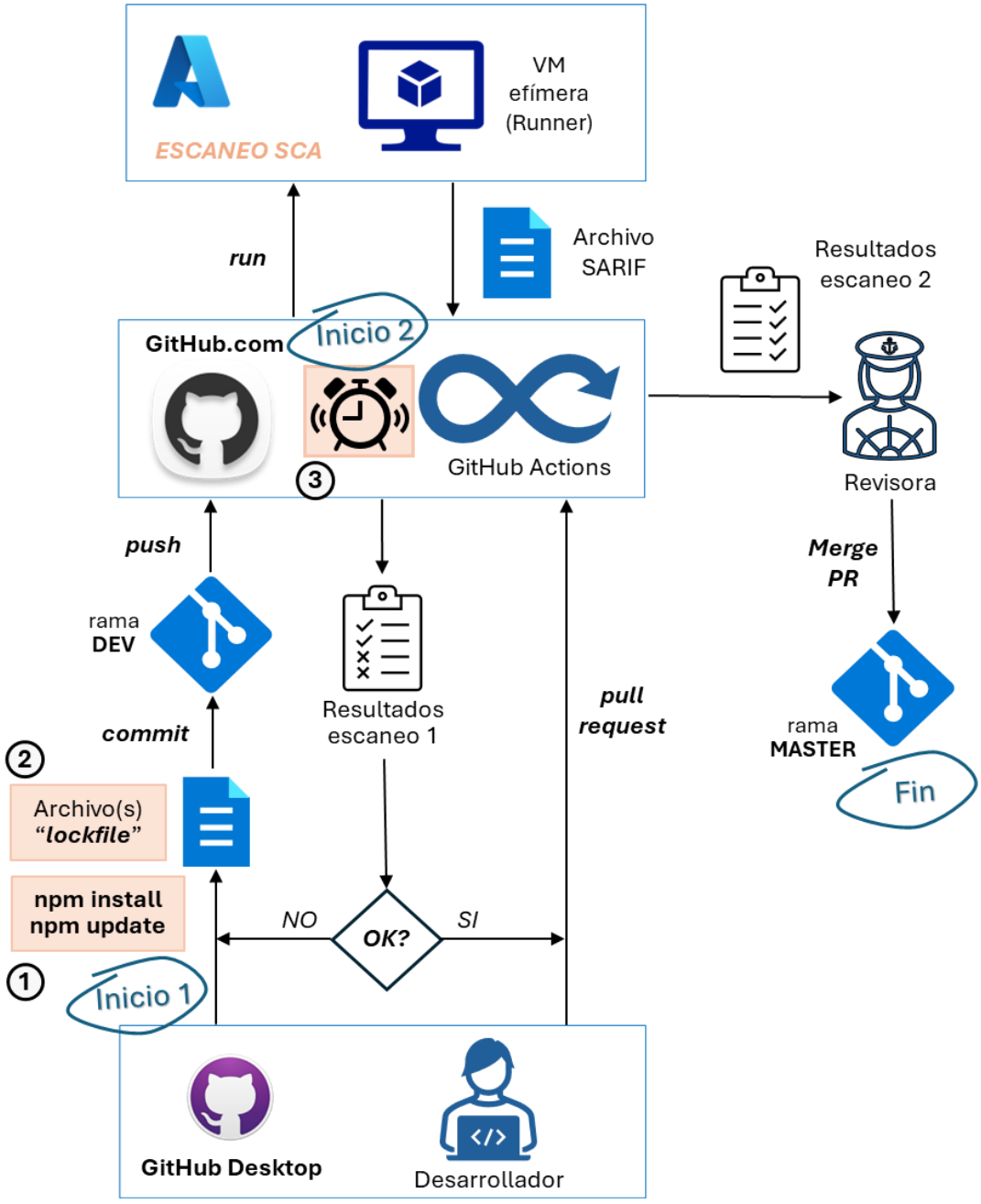


Ilustración 12. Flujo de trabajo en el escaneo SCA. Fuente: Elaboración propia

Las diferencias respecto al diagrama del flujo SAST son las siguientes:

1, 2 : Actualización de las dependencias del proyecto

El desarrollador añade una nueva librería al proyecto (mediante el comando “npm install”) o actualiza la versión de una librería ya instalada (mediante “npm update”).

Esto modifica el archivo de *lockfile,* que registra las versiones exactas de todas las dependencias. GitHub Desktop detecta automáticamente este cambio en el *lockfile,* y cuando el desarrollador realice el *commit* y *push* hacia la rama de desarrollo, el flujo de GitHub Actions se activará, de forma similar al caso del escaneo SAST.

3: Flujo programado

El flujo SCA también se iniciará de forma automática desde GitHub Actions a la hora programada, independientemente de si ha habido o no cambios en las librerías.

Esto permite verificar regularmente que todas las librerías de terceros siguen siendo seguras y están actualizadas, garantizando así la seguridad del proyecto.

El anexo Gestión del pull-request en GitHub muestra la interacción de la revisora con la interfaz gráfica de GitHub, donde se verifica la seguridad de las dependencias del proyecto antes de integrarlas en la rama principal.

3.4.3.Análisis de las vulnerabilidades detectadas en el escaneo SCA

Una vez completado el escaneo, los resultados están disponibles en la pestaña “Security” de GitHub, tal como se ilustra en el anexo Listado de vulnerabilidades SCA.

A continuación, se muestra una captura de la página de detalle correspondiente a la primera vulnerabilidad detectada: una inyección de código arbitrario en la librería “marsdb”.

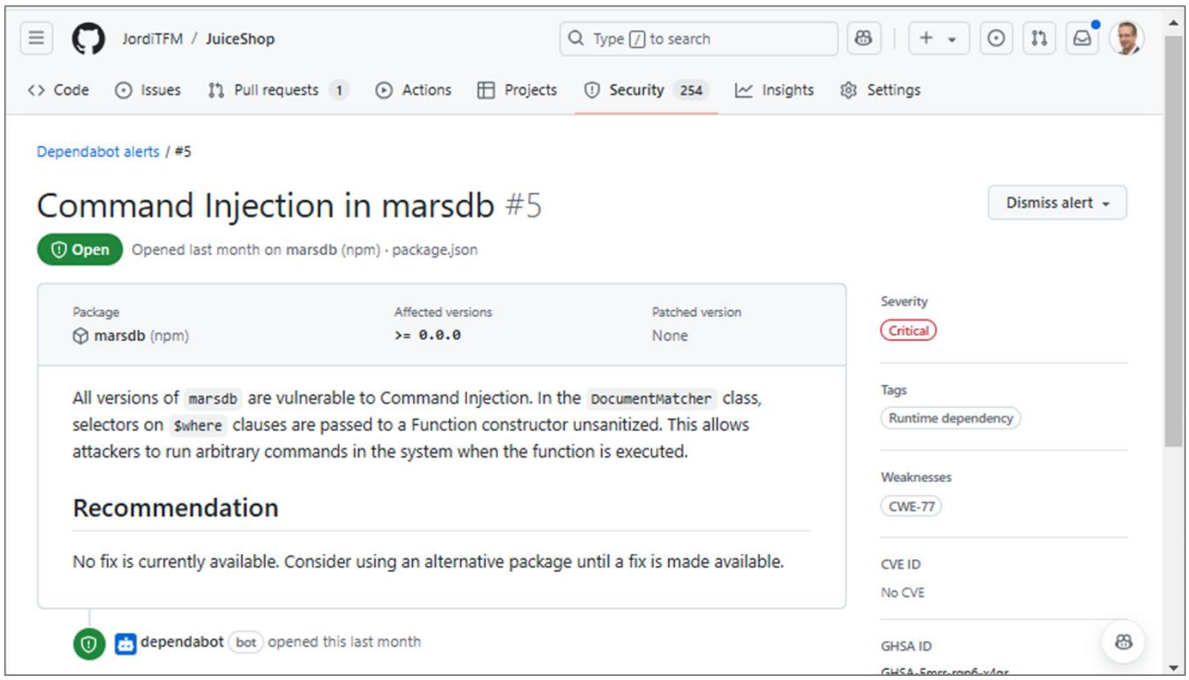


Ilustración 13. Vulnerabilidad "Command Injection" en una librería. Fuente: Elaboración propia

MarsDB (Artemyev, 2024) es una biblioteca ligera para bases de datos en memoria, diseñada para simular la funcionalidad de MongoDB en entornos donde no se requiere un motor de base de datos completo, y resulta apropiada para aplicaciones pequeñas o escenarios de pruebas basados en Node.js.

La vulnerabilidad detectada (CWE-77) tiene una puntuación de 9.8 (crítica) según la métrica CVSS v3, y reside en la clase “DocumentMatcher” del paquete, que permite pasar selectores a través de la cláusula “$where” directamente a un constructor de funciones sin realizar la sanitización adecuada de los datos.

Esto expone el sistema a ataques de inyección de comandos, en los que un atacante puede introducir código malicioso en los selectores, que luego se ejecuta en el sistema comprometiendo la seguridad del servidor y permitiendo actividades maliciosas, como la ejecución de comandos no autorizados, el robo de datos sensibles, o la minería de criptomonedas.

3.4.4.Remediación de las vulnerabilidades SCA

Actualmente, no existe un parche para esta vulnerabilidad en MarsDB, y Dependabot recomienda reemplazar esta biblioteca por una alternativa más segura.

La imagen siguiente muestra cómo se puede cerrar la alerta en GitHub, indicando el motivo y añadiendo un comentario justificativo. En este caso, se especifica que ya se ha iniciado la tarea para reemplazar MarsDB por SQLite:

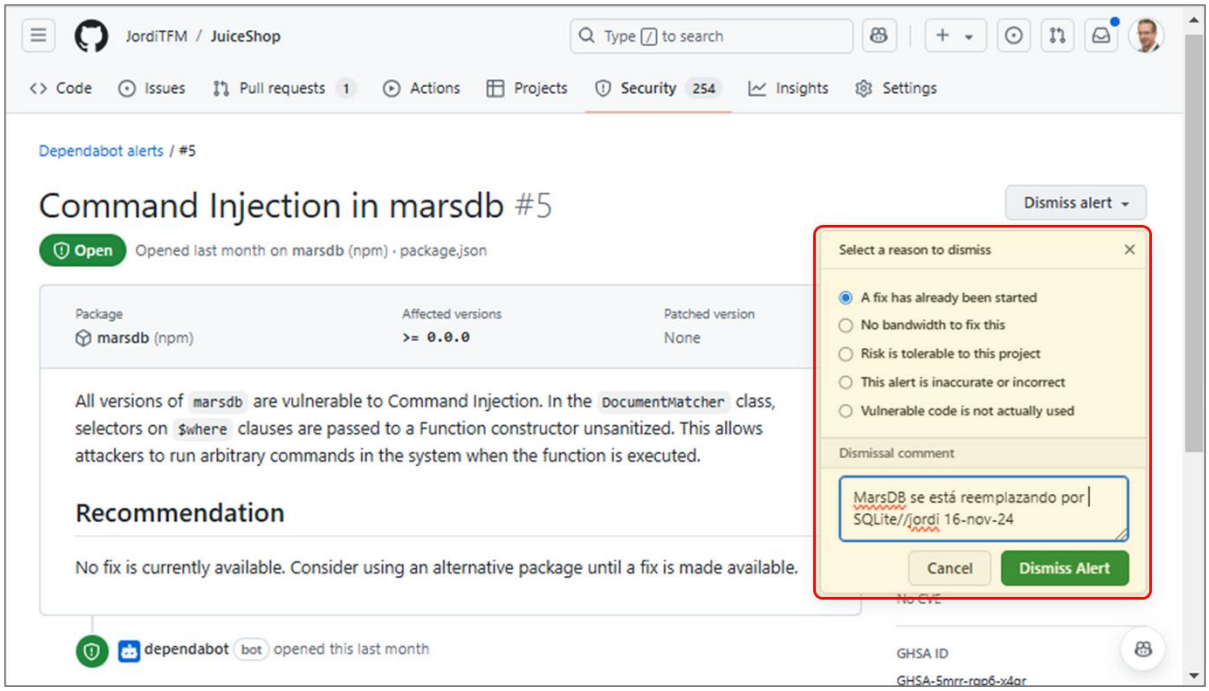


Ilustración 14. Cierre de la alerta detectada por Dependabot. Fuente: Elaboración propia

Por otra parte, la segunda alerta identificada en el listado de vulnerabilidades, tiene una solución más sencilla ya que no requiere sustituir la librería afectada.

Se trata de una vulnerabilidad de tipo CWE-20 (“Improper Input Validation”) que afecta a todas las versiones de la librería “jsonwebtoken” anteriores a la 4.2.2. El problema radica en que estas versiones no validaban correctamente la cabecera del JWT, lo que permitía a un atacante omitir el algoritmo de firma para evitar su verificación, y poder así manipular el contenido del token sin que se detectara la alteración.

La captura siguiente muestra el detalle de esta vulnerabilidad, y, además, permite observar que Dependabot ha generado automáticamente un *pull-request* para actualizar la librería desde la versión actual (0.4.0) a la versión 9.0.0:

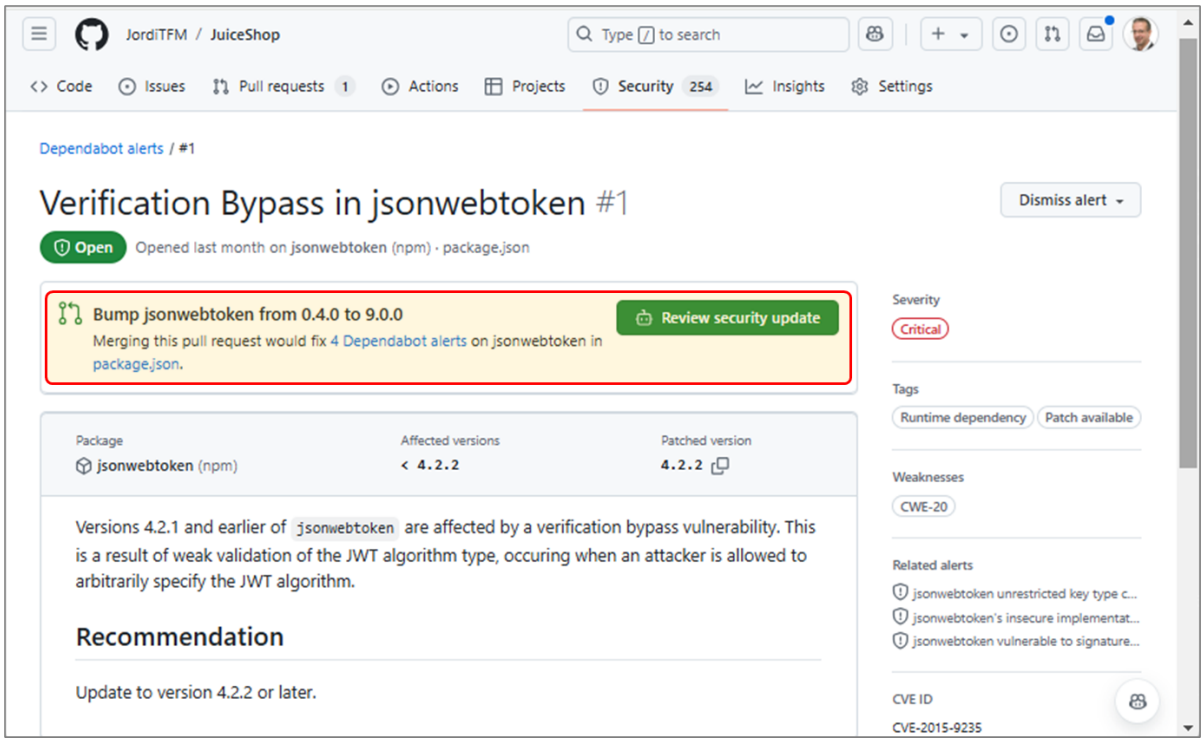


Ilustración 15. Vulnerabilidad "Verification Bypass" en una librería. Fuente: Elaboración propia

Con la aplicación del *pull-request* quedará resuelta la incidencia, tal como se detalla en el anexo Remediación automática con Dependabot y, al mismo tiempo, se resolverán las otras 3 alertas que también estaban relacionadas con el mismo componente, tal como se destaca en el recuadro amarillo de la imagen de arriba.

3.5 Integración de otras herramientas en GitHub Actions

En el libro "*Learning GitHub Actions: Automation and Integration of CI/CD with GitHub*" (Laster, 2023), se exploran en profundidad las capacidades y mecanismos de automatización que GitHub Actions ofrece para la integración de flujos CI/CD. Este enfoque permite definir y gestionar workflows mediante archivos de configuración en lenguaje YAML (.yml), lo que facilita la personalización de cada paso del pipeline.

Gracias a esta flexibilidad, es posible integrar prácticamente cualquier herramienta AST en los pipelines de GitHub, aprovechando la estructura modular y adaptable que GitHub Actions proporciona.

En esta sección, se analizará un ejemplo práctico con **Semgrep** (Semgrep, 2024), una herramienta de análisis de código recomendada en la *OWASP DevSecOps Guideline*, que cuenta con una versión de evaluación gratuita.

Este caso práctico demostrará dos métodos de integración de Semgrep: un escaneo gestionado directamente desde la herramienta, y una integración completa dentro del pipeline CI/CD en GitHub Actions.

3.5.1. Escaneo gestionado desde la herramienta AST

El anexo titulado Escaneo del repositorio GitHub desde la aplicación Semgrep detalla el proceso de conexión que debe configurarse en Semgrep para habilitar el escaneo del repositorio GitHub.

En este proceso, se requiere la autenticación en Semgrep y la concesión de permisos para acceder al repositorio. Los resultados de esta integración se visualizan en la interfaz de Semgrep.

Sin embargo, este tipo de integración presenta ciertas limitaciones:

- Gestión limitada en GitHub: No permite visualizar ni gestionar las alertas directamente desde el entorno de GitHub.
- Falta de integración en el ciclo de vida CI/CD: No integra las pruebas AST en eventos clave del ciclo de vida del software (como *push* o *pull-request*), lo cual limita su utilidad para una supervisión continua dentro del *pipeline* CI/CD de GitHub.
- Interfaz más básica: La consola de Semgrep es menos completa que la interfaz de Security de GitHub, lo cual limita las opciones de clasificación, priorización y asignación de alertas. GitHub, en cambio, permite una gestión más centralizada y avanzada, lo que facilita el flujo de trabajo para los equipos de desarrollo y seguridad.

Debido a su simplicidad, este tipo de integración es útil para pruebas manuales o análisis puntuales, pero no es óptima para implementaciones en un flujo de trabajo CI/CD completamente automatizado. Para este tipo de implementaciones, resulta beneficioso aprovechar la interfaz avanzada de Security de GitHub para la gestión continua y colaborativa de las alertas de seguridad.

3.5.2. Escaneo integrado en el pipeline CI/CD

Para conseguir una integración plena de Semgrep en el *pipeline* CI/CD de GitHub Actions, se requiere una configuración adicional más allá de la simple instalación de la plantilla YAML predefinida.

A continuación, se explican los pasos adicionales necesarios para lograr una integración completa:

Selección de la plantilla YAML predefinida de Semgrep

GitHub facilita una plantilla básica predefinida para Semgrep, accesible desde el menú "Security". Esta plantilla se puede instalar fácilmente en la carpeta de *workflows* del repositorio, tal como se describe en el anexo Integración de la plantilla YAML predefinida.

Sin embargo, la versión predefinida aporta un valor muy escaso para una integración funcional, ya que no incluye varios ajustes esenciales. Por lo tanto, si una herramienta AST carece de plantilla predefinida en el catálogo de GitHub Actions, esto no representa una barrera significativa para su integración.

Creación y configuración del token de acceso

Para que GitHub pueda ejecutar Semgrep de manera automática, es necesario crear un token de acceso en Semgrep y configurarlo como un secreto en GitHub. Esto permite una comunicación segura entre GitHub Actions y Semgrep.

El anexo Creación del token de acceso para la conexión con Semgrep detalla los pasos para generar y configurar este token, que es imprescindible para la autenticación en *workflows* automatizados.

Personalización del archivo YAML

La plantilla YAML debe ser editada para incluir el token de acceso y otros ajustes, necesarios para que los escaneos de Semgrep se activen y ejecuten correctamente dentro del flujo CI/CD de GitHub Actions.

El anexo Configuración del archivo de workflow (.yml) muestra el archivo “semgrep.yml” final, con la configuración de todos los parámetros necesarios para un correcto funcionamiento del workflow.

3.6 Evaluación de impacto tras la implementación

La fase de implementación ha sido fundamental para materializar los conocimientos adquiridos durante la investigación, demostrando la eficacia y practicidad de integrar herramientas AST en un entorno *DevSecOps* utilizando GitHub Actions.

Esta implementación ha tenido un impacto significativo en la calidad del software y en la eficiencia del proceso de desarrollo.

A continuación, se destacan las principales áreas de mejora observadas:

3.6.1. Mejora en la detección de vulnerabilidades

La integración de CodeQL y Dependabot ha permitido identificar de manera sistemática vulnerabilidades críticas tanto en el código fuente como en las dependencias del proyecto. Algunos ejemplos destacados son:

- CodeQL detectó 88 vulnerabilidades en el código de OWASP Juice Shop, incluyendo casos complejos como CWE-843 (confusión de tipos), cuya remediación fue asistida por GitHub Copilot Autofix, reduciendo significativamente el tiempo necesario para corregir errores de seguridad.
- Dependabot identificó varios problemas críticos, como una vulnerabilidad de tipo CWE-77 (inyección de comandos) en la librería "marsdb" y vulnerabilidades de tipo CWE-20 (validación incorrecta de parámetros de entrada) en la librería "jsonwebtoken". En este último caso, la automatización generó pull-requests que actualizaron la dependencia a una versión segura, resolviendo múltiples alertas de una sola vez.

Estos resultados demuestran que la adopción de herramientas AST mejora sustancialmente la cobertura de seguridad, permitiendo identificar vulnerabilidades que fácilmente pasarían desapercibidas con métodos manuales.

3.6.2. Reducción de tiempos de desarrollo y mantenimiento

La automatización de los flujos de trabajo ha optimizado significativamente el proceso de desarrollo:

- La detección temprana de vulnerabilidades en eventos clave (*push*, *pull-request*) evitó que los problemas de seguridad se acumularan en etapas avanzadas, donde su corrección sería más costosa y difícil de gestionar.
- Herramientas como Copilot Autofix redujeron el tiempo dedicado a remediar vulnerabilidades comunes, como validaciones incorrectas o sanitización de entradas, permitiendo a los desarrolladores centrarse en tareas más complejas y creativas.

Asimismo, las actualizaciones automatizadas de Dependabot eliminaron la necesidad de revisiones manuales de dependencias, mejorando la agilidad del mantenimiento.

3.6.3.Incremento en la estabilidad del sistema

La integración del análisis de dependencias (SCA) ha tenido un impacto directo en la estabilidad y seguridad del sistema al garantizar que las bibliotecas externas utilizadas están actualizadas y libres de vulnerabilidades conocidas:

- La sustitución de librerías vulnerables, como "marsdb", por alternativas más seguras (SQLite) redujo los potenciales vectores de ataque.
- Los escaneos programados permitieron detectar vulnerabilidades nuevas incluso cuando no se habían realizado cambios en el código base, reforzando la seguridad continua del proyecto.

3.6.4.Experiencia del desarrollador

La combinación de una interfaz gráfica avanzada en GitHub y las capacidades automatizadas de las herramientas AST ha mejorado la experiencia de los desarrolladores, permitiendo:

- Gestionar y priorizar alertas de forma centralizada desde la pestaña "Security" de GitHub, con documentación detallada sobre cada vulnerabilidad.
- Aplicar correcciones automáticas o generar incidencias asignables en un entorno colaborativo, lo que facilita la integración de la seguridad en el flujo de trabajo diario.

3.6.5.Conclusión

La implementación de herramientas AST en un pipeline *DevSecOps* no solo mejora la calidad y seguridad del software, sino que también incrementa la eficiencia del desarrollo y el mantenimiento. Este impacto positivo valida el enfoque adoptado en este proyecto y subraya la importancia de integrar herramientas AST en el ciclo de vida del desarrollo de software.

La experiencia adquirida y los beneficios observados refuerzan la recomendación de adoptar prácticas *DevSecOps* y perfeccionar la integración de herramientas de seguridad. Esto impulsará una nueva generación de aplicaciones más seguras, fortaleciendo la competitividad y la confianza de los usuarios en los productos desarrollados.

4. Conclusiones

4.1 Introducción

En este proyecto se ha trabajado en la integración de herramientas de análisis de seguridad de aplicaciones en un pipeline automatizado *DevSecOps,* como parte del ciclo de vida del desarrollo de software.

El objetivo principal ha sido garantizar la identificación y mitigación temprana de vulnerabilidades, destacando la importancia de incorporar la seguridad desde las primeras etapas del desarrollo.

Este capítulo evalúa el grado de cumplimiento de los objetivos propuestos, analiza la planificación seguida, resume las lecciones aprendidas y presenta posibles líneas de trabajo futuro que podrían ampliar el alcance del proyecto.

4.2 Evaluación del cumplimiento de objetivos

Los objetivos primarios del proyecto, centrados en la selección, configuración e integración de herramientas SAST y SCA en un pipeline CI/CD mediante GitHub Actions, se han alcanzado plenamente.

Estas herramientas han demostrado ser eficaces para la detección y gestión de vulnerabilidades, como se evidencia en la documentación del entorno de laboratorio diseñado, alineado con los estándares OWASP.

En cuanto a los objetivos secundarios:

- Se completó un análisis introductorio del OWASP Benchmark Project, lo que permitió comprender el rol de las plataformas de benchmarking en el contexto del proyecto.
- Aunque inicialmente se consideró integrar herramientas DAST, este objetivo se dejó como una posible evolución del proyecto, debido a las limitaciones técnicas y a la necesidad de mantener un enfoque claro y pedagógico.

En términos de planificación, se respetaron en general los plazos establecidos en el diagrama de Gantt, y la iteración propuesta a través de sprints permitió ajustes según los retos surgidos durante la implementación.

4.3 Lecciones aprendidas

A lo largo de este TFM, se han identificado una serie de lecciones clave que subrayan la importancia de integrar herramientas de seguridad en el ciclo de vida del desarrollo de software, así como los retos y estrategias para maximizar su efectividad.

Estas lecciones, extraídas tanto de la etapa de investigación como de la implementación, pueden resumirse en los siguientes puntos:

1. **La importancia de la automatización en *DevSecOps***

   Las revisiones de seguridad hechas de forma manual son ineficientes, y representan un obstáculo para la velocidad que el negocio digital le exige al desarrollo del software.

   La integración de herramientas como CodeQL (SAST) y Dependabot (SCA) mediante GitHub Actions permitió automatizar las pruebas de seguridad sin ralentizar el flujo de desarrollo y sin tener que esperar a la integración en la rama principal de la aplicación para poder hacerlo.

   Además, las pruebas realizadas en el capítulo de Implementación demostraron que la incorporación de la Inteligencia Artificial, a través de GitHub Copilot, aporta un notable valor añadido al proceso de remediación. Copilot sugiere soluciones en tiempo real, facilitando que perfiles no especializados en ciberseguridad comprendan las vulnerabilidades y apliquen correcciones con agilidad. De este modo, la IA no solo acelera la resolución de problemas, sino que democratiza el acceso a buenas prácticas de seguridad, consolidando la automatización como un componente clave del enfoque *DevSecOps*.

2. **Responsabilidad compartida entre desarrollo y seguridad**

   Los equipos de desarrollo deben asumir un papel de responsabilidad en los procesos relacionados con la seguridad, aunque carezcan de conocimientos o experiencia previa en ciberseguridad.

   Este trabajo demostró que integrando las herramientas AST en el pipeline CI/CD, los desarrolladores pueden asumir un papel más activo en la identificación y mitigación de las vulnerabilidades, sin necesidad de ser expertos en ciberseguridad.

3. **Priorizar las vulnerabilidades más críticas**

   Los desarrolladores deben focalizarse en primer lugar en las vulnerabilidades conocidas de mayor riesgo, utilizando herramientas que proporcionen métricas de riesgo. Asimismo, es más importante identificar vulnerabilidades conocidas en el software de terceros, que centrarse exclusivamente en los problemas del código propio.

   Las pruebas realizadas en el proyecto demostraron que las alertas generadas por CodeQL y Dependabot en la pestaña “Security” de GitHub aportaban la información necesaria para priorizar la remediación de las vulnerabilidades en base a la severidad de las mismas.

4. **Adaptar las herramientas al desarrollador**

   Para asegurar la experiencia de usuario y su productividad, los procesos y las herramientas deben adaptarse al desarrollador, y no a la inversa. El éxito de las herramientas AST radica en su capacidad para integrarse en la plataforma de desarrollo y en el sistema de control de versiones.

   En el proyecto se ha demostrado una integración de las herramientas AST sin fisuras en la plataforma GitHub, mediante la cual el desarrollador obtenía feedback

continuo sobre los resultados de las verificaciones de seguridad, de la misma forma en la que obtiene el feedback de cualquier otro tipo de validaciones, y con la misma operativa de remediación.

En resumen, estas lecciones aprendidas destacan cómo un enfoque centrado en la automatización, la priorización del riesgo y la integración fluida de herramientas puede transformar la seguridad en un elemento inherente al desarrollo, sin comprometer la productividad ni la agilidad.

4.4 Posibles trabajos futuros

Este proyecto sienta las bases para futuras investigaciones y desarrollos. Entre las posibles líneas de evolución se incluyen:

1. **Integración de herramientas DAST**

   Ampliar el pipeline actual con herramientas de análisis dinámico que permitan evaluar aplicaciones en ejecución.

   Además, sería interesante explorar métodos para transformar los resultados generados por estas herramientas al formato SARIF, facilitando su integración en GitHub y su visualización junto a los análisis SAST y SCA.

2. **Integración de herramientas IAST**

   Las herramientas IAST aportan un valor añadido al combinar atributos de SAST y DAST, proporcionando visibilidad tanto interna como externa de la aplicación. Su capacidad para correlacionar la ejecución del código con su estructura permite un equilibrio óptimo entre la reducción de falsos positivos y la precisión en la cobertura de código.

   Sin embargo, su adopción sigue siendo limitada debido a restricciones en el soporte de plataformas, lo que abre un área de investigación sobre su implementación en distintos entornos.

3. **Benchmarking de herramientas**

   Ampliar el análisis del OWASP Benchmark Project mediante pruebas que evalúen la eficiencia de las herramientas SAST utilizadas, midiendo ratios de falsos positivos y negativos, así como su rendimiento.

   Aunque este análisis puede ser una guía útil para seleccionar herramientas en proyectos futuros, es importante considerar que el OWASP Benchmark Project actualmente solo es compatible con código Java.

5. Glosario de términos y acrónimos

Amenaza	Toda circunstancia o evento que potencialmente pueda afectar de forma adversa a la aplicación bloqueando el servicio o provocando la destrucción, alteración o divulgación indebida de la información.
API	Application Programming Interface.
AST	Application Security Testing.
Atacante	Individuo, grupo, organización o gobierno que realiza o tiene la intención de realizar un ataque.
Benchmark	Estándar utilizado para evaluar el rendimiento de herramientas o sistemas mediante pruebas en un entorno controlado con parámetros definidos.
Commit	Acción que guarda un conjunto de cambios en el historial del repositorio local, permitiendo llevar un registro de cada modificación realizada en el código. Cada *commit* crea un punto de control que documenta el cambio y la razón que lo ha motivado.
CSA	Cloud Security Alliance.
CVSS	Common Vulnerability Scoring System.
DAST	Dynamic Application Security Testing.
EPIC	Tarea EPIC o épica, en *Scrum*, es un elemento de trabajo o requisito de alto nivel demasiado grande para ser completado en una sola iteración o *sprint*.
IAST	Interactive Application Security Testing.
IaC	Infrastructure as Code.
JSON	JavaScript Object Notation.
JWT	JSON Web Token.
Lockfile	Archivo que registra las versiones exactas de las dependencias instaladas en un proyecto en el momento de su creación.
Machine Learning	Rama de la Inteligencia Artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan.
MAST	Mobile Application Security Testing.
Merge	Operación que consolida los cambios de dos ramas distintas del repositorio de código.
OWASP	Open Web Application Security Project.
Pull	Acción que permite al desarrollador obtener la última versión del código de un repositorio remoto en su entorno local.
Pull-request	Solicitud para que se revise un conjunto de cambios realizados en una rama específica y se fusione con otra rama, normalmente la principal.

Push	Operación que envía los cambios confirmados (*commits*) desde el repositorio local al repositorio remoto.
SAMM	Software Assurance Maturity Model.
SARIF	Static Analysis Results Interchange Format.
SAST	Static Application Security Testing.
SBOM	Software Bills Of Materials.
SCA	Software Composition Analysis.
Scrum	Marco de trabajo para el desarrollo ágil de software.
SSDLC	Secure Software Development Lifecycle.
Vulnerabilidad	Debilidad de un activo que puede ser explotada por una o más amenazas.

6. Bibliografía

Artemyev, A. (16 de noviembre de 2024). *GitHub*. Obtenido de https://github.com/c58/marsdb

Carbon Leadership Forum. (26 de octubre de 2024). *Emisiones de carbono incorporadas en Microsoft.* Obtenido de https://carbonleadershipforum.org/es/como-microsoft-esta-reduciendo-el-carbono-incorporado/

Chapple, M. (2024). *ISC2 CISSP Certified Information Systems Security Professional Official Study Guide, 10th Edition.* Canada and the United Kingdom: John Wiley & Sons, Inc.

Checkmarx. (12 de octubre de 2024). *Checkmarx SAST*. Obtenido de https://checkmarx.com/cxsast-source-code-scanning/

Checkmarx. (19 de octubre de 2024). *ZAP*. Obtenido de https://www.zaproxy.org/

Gartner. (2024). *Critical Capabilities for Application Security Testing.* Stamford: Gartner. Recuperado el 29 de septiembre de 2024, de https://www.gartner.com/document/4368599?ref=ddisp&refval=4366399

GitHub Education. (13 de octubre de 2024). Obtenido de https://docs.github.com/en/education/explore-the-benefits-of-teaching-and-learning-with-github-education/github-education-for-students

GitHub, Inc. (26 de octubre de 2024). *CodeQL documentation*. Obtenido de https://codeql.github.com/docs/

GitHub, Inc. (26 de octubre de 2024). *Dependabot quickstart guide*. Obtenido de https://docs.github.com/en/code-security/getting-started/dependabot-quickstart-guide

HCLSoftware. (12 de octubre de 2024). Obtenido de https://www.hcl-software.com

HCLSoftware. (12 de octubre de 2024). *HCL AppScan Standard*. Obtenido de https://hcl-software.com/appscan/products/appscan-standard

Laster, B. (2023). *Learning GitHub Actions - Automation and Integration of CI/CD with GitHub.* Sebastopol, California: O'Reilly.

Manjunath Bhat, M. H. (2023). *How to Select DevSecOps Tools for Secure Software Delivery.* Stamford. Obtenido de https://www.gartner.com/document/code/772114

Mark Horvath, D. G. (2024). *Magic Quadrant for Application Security Testing.* Stamford: Gartner Group. Recuperado el 29 de septiembre de 2024, de https://www.gartner.com/interactive/mq/4366399

Mike Chapple, D. S. (2024). *CompTIA Security+ Study Guide.* New Jersey: John Wiley & Sons.

MITRE. (2 de noviembre de 2024). *Common Weakness Enumeration*. Obtenido de https://cwe.mitre.org/data/definitions/843.html

OpenText. (12 de octubre de 2024). Obtenido de http://www.opentext.com

OWASP Foundation. (27 de septiembre de 2024). *Benchmark Project*. Obtenido de https://owasp.org/www-project-benchmark/

OWASP Foundation. (19 de octubre de 2024). *Dependency-Check*. Obtenido de https://owasp.org/www-project-dependency-check/

OWASP Foundation. (28 de septiembre de 2024). *DevSecOps Guideline*. Obtenido de https://github.com/OWASP/DevSecOpsGuideline/tree/master/current-version

OWASP Foundation. (28 de septiembre de 2024). *Juice Shop Project*. Obtenido de https://owasp.org/www-project-juice-shop/

OWASP Foundation. (12 de octubre de 2024). *OWASP Top Ten*. Obtenido de https://owasp.org/www-project-top-ten/

OWASP Foundation. (12 de octubre de 2024). *OWASP WebGoat*. Obtenido de https://owasp.org/www-project-webgoat/

OWASP Foundation. (s.f.). *Threat Modeling Cheat Sheet*. (OWASP) Obtenido de https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html

Rich Mogull, Mike Rothman. (27 de septiembre de 2024). *Security Guidance for Critical Areas of Focus in Cloud Computing v5.* (C. S. Alliance, Ed.) Obtenido de https://cloudsecurityalliance.org/artifacts/security-guidance-v5

Semgrep. (19 de octubre de 2024). Obtenido de https://semgrep.dev/

Shostack, A. (2014). *Threat Modeling: Designing for Security.* Indianapolis: John Wiley & Sons.

Sonar. (19 de octubre de 2024). *SonarQube Community Edition*. Obtenido de https://www.sonarsource.com/open-source-editions/sonarqube-community-edition/

Synopsys. (12 de octubre de 2024). Obtenido de http://www.synopsys.com

Synopsys. (12 de octubre de 2024). *Black Duck SCA.* Obtenido de https://www.blackduck.com/software-composition-analysis-tools/black-duck-sca.html

Synopsys. (12 de octubre de 2024). *Seeker IAST.* Obtenido de https://www.synopsys.com/content/dam/synopsys/sig-assets/datasheets/interactive-application-security-testing-datasheet.pdf

7. Anexos

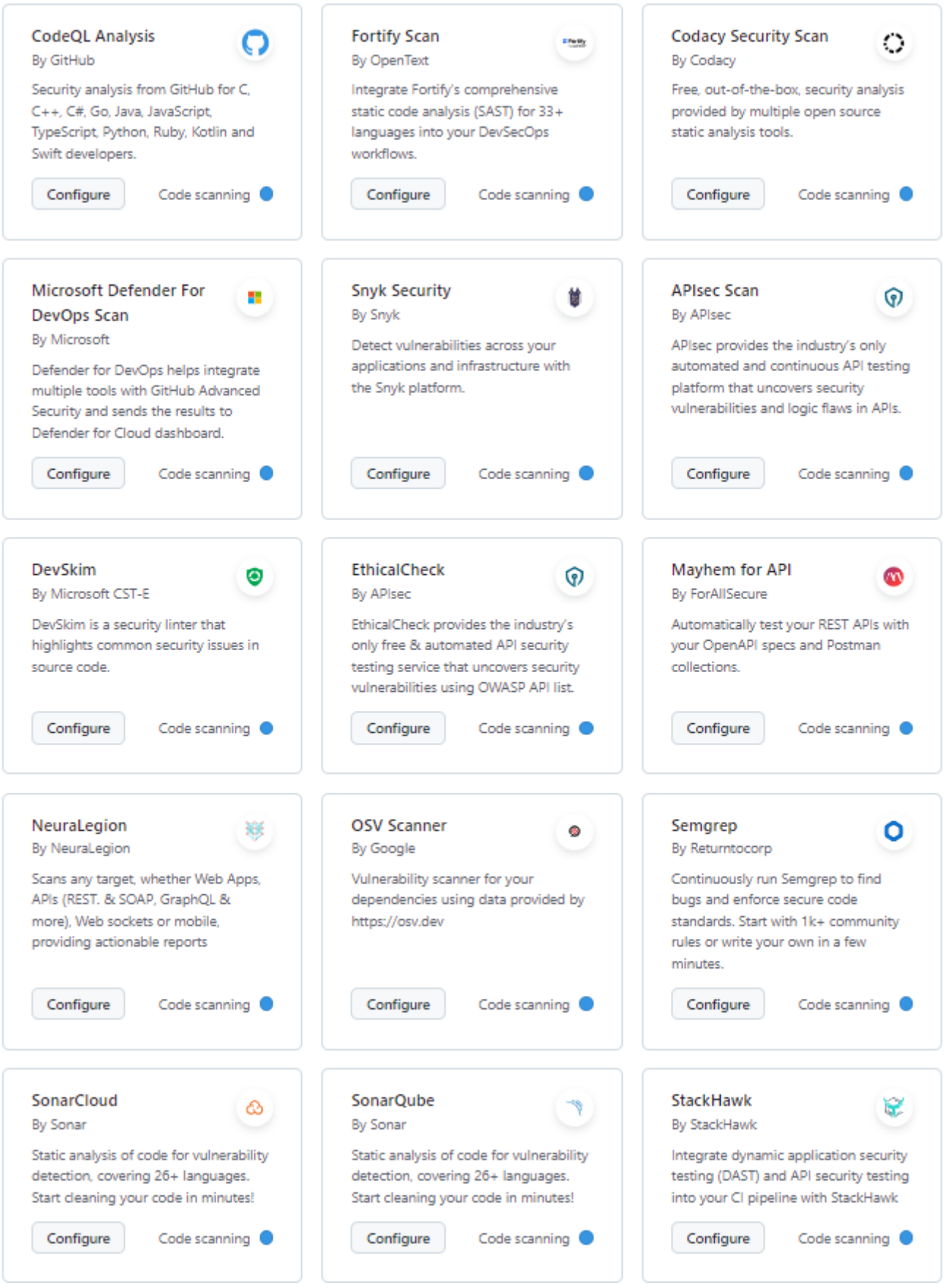
7.1. Matriz de herramientas propuestas por OWASP (por orden alfabético)

Herramienta	SAST	DAST	IAST	SCA	Containe	IaC
Acunetix	No	Sí	No	No	No	No
Anchore	No	No	No	No	Sí	No
ansible-lint	No	No	No	No	No	Sí
Aquasec	No	No	No	No	Sí	No
Brakeman	Sí	No	No	No	No	No
bundler-audit	No	No	No	Sí	No	No
Burp Suite	No	Sí	No	No	No	No
Checkmarx IAST (CxIAST)	No	No	Sí	No	No	No
Checkmarx SAST	Sí	No	No	No	No	No
Checkov	No	No	No	No	No	Sí
Clair	No	No	No	No	Sí	No
CodeQL	Sí	No	No	No	No	No
CodeSec by Contrast Security	Sí	No	No	No	No	No
CodeSweep	Sí	No	No	No	No	No
Contrast Assess y Contrast Community Edition	No	No	Sí	No	No	No
Dagda	No	No	No	No	Sí	No
Dastardly	No	Sí	No	No	No	No
Enlightn	Sí	No	No	No	No	No
Falco	No	No	No	No	Sí	No
Fortify	Sí	No	No	No	No	No
Hakiri	No	No	No	Sí	No	No
Harbor	No	No	No	No	Sí	No
HCL AppScan on Cloud	Sí	Sí	Sí	Sí	No	No
Inquisition	Sí	No	No	No	No	No
InsightAppSec (AppSpider)	No	Sí	No	No	No	No
KICS	No	No	No	No	No	Sí
Kubescape	No	No	No	No	Sí	Sí
Netsparker	No	Sí	No	No	No	No
Nuclei	No	Sí	No	No	No	No
OWASP CycloneDX	No	No	No	Sí	No	No
OWASP Dependency-Check	No	No	No	Sí	No	No
OWASP Dependency-Track	No	No	No	Sí	No	No
OWASP dep-scan	No	No	No	Sí	No	No
OWASP Nettacker	No	Sí	No	No	No	No
PT Application Inspector	Sí	No	No	No	No	No
puppet-lint	No	No	No	No	No	Sí
RetireJS	No	No	No	Sí	No	No
Safety	No	No	No	Sí	No	No
Security Code Scan	Sí	No	No	No	No	No
Seeker Interactive Application Security Testing	No	No	Sí	No	No	No
Semgrep	Sí	No	No	No	No	No
Snyk	No	No	No	Sí	No	Sí
SonarQube	Sí	No	No	No	No	No
Synopsys BlackDuck	No	No	No	Sí	No	No
terrascan	No	No	No	No	No	Sí
tflint	No	No	No	No	No	Sí
tfsec	No	No	No	No	No	Sí
Trivy	No	No	No	No	Sí	Sí
Veracode	Sí	Sí	No	No	No	No
WhiteSource	No	No	No	Sí	No	No
ZED Attack Proxy (ZAP)	No	Sí	No	No	No	No
51	14	10	4	12	8	10

7.2. Matriz de herramientas propuestas por OWASP (por categoría)

Herramienta	SAST	DAST	IAST	SCA	Containe	IaC
HCL AppScan on Cloud	Sí	Sí	Sí	Sí	No	No
Veracode	Sí	Sí	No	No	No	No
Checkmarx SAST	Sí	No	No	No	No	No
Brakeman	Sí	No	No	No	No	No
CodeQL	Sí	No	No	No	No	No
CodeSec by Contrast Security	Sí	No	No	No	No	No
CodeSweep	Sí	No	No	No	No	No
Enlightn	Sí	No	No	No	No	No
Fortify	Sí	No	No	No	No	No
Inquisition	Sí	No	No	No	No	No
PT Application Inspector	Sí	No	No	No	No	No
Security Code Scan	Sí	No	No	No	No	No
Semgrep	Sí	No	No	No	No	No
SonarQube	Sí	No	No	No	No	No
Acunetix	No	Sí	No	No	No	No
Burp Suite	No	Sí	No	No	No	No
Dastardly	No	Sí	No	No	No	No
InsightAppSec (AppSpider)	No	Sí	No	No	No	No
Netsparker	No	Sí	No	No	No	No
Nuclei	No	Sí	No	No	No	No
OWASP Nettacker	No	Sí	No	No	No	No
ZED Attack Proxy (ZAP)	No	Sí	No	No	No	No
Checkmarx IAST (CxIAST)	No	No	Sí	No	No	No
Contrast Assess y Contrast Community Edition	No	No	Sí	No	No	No
Seeker Interactive Application Security Testing	No	No	Sí	No	No	No
Snyk	No	No	No	Sí	No	Sí
bundler-audit	No	No	No	Sí	No	No
Hakiri	No	No	No	Sí	No	No
OWASP CycloneDX	No	No	No	Sí	No	No
OWASP Dependency-Check	No	No	No	Sí	No	No
OWASP Dependency-Track	No	No	No	Sí	No	No
OWASP dep-scan	No	No	No	Sí	No	No
RetireJS	No	No	No	Sí	No	No
Safety	No	No	No	Sí	No	No
Synopsys BlackDuck	No	No	No	Sí	No	No
WhiteSource	No	No	No	Sí	No	No
Kubescape	No	No	No	No	Sí	Sí
Trivy	No	No	No	No	Sí	Sí
Anchore	No	No	No	No	Sí	No
Aquasec	No	No	No	No	Sí	No
Clair	No	No	No	No	Sí	No
Dagda	No	No	No	No	Sí	No
Falco	No	No	No	No	Sí	No
Harbor	No	No	No	No	Sí	No
ansible-lint	No	No	No	No	No	Sí
Checkov	No	No	No	No	No	Sí
KICS	No	No	No	No	No	Sí
puppet-lint	No	No	No	No	No	Sí
terrascan	No	No	No	No	No	Sí
tflint	No	No	No	No	No	Sí
tfsec	No	No	No	No	No	Sí
51	14	10	4	12	8	10

7.3. Plantillas para la integración de herramientas AST en GitHub Actions



Appknox
By Appknox

Use Appknox action for faster and precise security assessments of your iOS and Android apps developed using any programming language

Configure Code scanning

Bearer
By Bearer

Continuously run Bearer code security scanning tool (SAST) to discover, filter and prioritize security and privacy risks.

Configure Code scanning

ESLint
By GitHub Actions

A tool for identifying and reporting the problems found in ECMAScript/JavaScript code.

Configure Code scanning

njsscan
By NodeJSScan

nodejsscan is a static security code scanner that finds insecure code patterns in your Node.js applications.

Configure Code scanning

NowSecure Mobile SBOM
By NowSecure

Generate a Mobile SBOM for an application and submit to Dependency Graph

Configure Code scanning

NowSecure
By NowSecure

The NowSecure Action delivers fast, accurate, automated security analysis of iOS and Android apps coded in any language

Configure Code scanning

Xanitizer
By RIGS IT

Automatically scan your code for vulnerabilities and generate compliance reports with the static security analysis tool Xanitizer (SAST).

Configure Code scanning

Jscrambler Code Integrity
By Jscrambler

Protect your JavaScript Application with polymorphic obfuscation, code locks, and self-defensive techniques

Configure Code scanning

Sysdig Inline Scan
By Sysdig

Performs analysis on locally built container image and posts the results in SARIF report

Configure Code scanning

Checkmarx
By Checkmarx

Beat vulnerabilities with more secure code.Scan your code with Checkmarx One and see results in the GitHub code scanning.

Configure Code scanning

CxSAST
By Checkmarx

Scan your code with Checkmarx CxSAST and see your results in the GitHub security tab.

Configure Code scanning

Debricked Scan
By OpenText

Integrate with Debricked's state of the art AI-powered Software Composition Analysis to automate your security.

Configure Code scanning

Endor Labs scan
By Endor Labs

Identify, prioritize and address open source and code governance issues with Endor Labs.

Configure Code scanning

OSSAR
By GitHub

Run multiple open source security static analysis tools without the added complexity with OSSAR (Open Source Static Analysis Runner).

Configure Code scanning

Synopsys Action
By Synopsys

The Synopsys GitHub Action allows you to configure your pipeline to run Synopsys security testing and take action on the security results

Configure Code scanning

Synopsys Intelligent Security Scan Action

By Synopsys

The Synopsys Intelligent Security Scan Action helps selectively perform SAST and SCA scans, triggered during a variety of GitHub Platform events

Configure Code scanning

Veracode Static Analysis

By Veracode

Get fast feedback on flaws with Veracode Static Analysis and the pipeline scan. Break the build based on flaw severity and CWE category.

Configure Code scanning

CodeScan

By CodeScan Enterprises, LLC

CodeScan allows for better visibility on your code quality checks based on your custom rulesets.

Configure Code scanning

Contrast Scan

By Contrast Security Inc

Scans Pull Requests on each push for the introduction and/or resolution of vulnerabilities to the repository.

Configure Code scanning

pmd

By pmd

PMD is a static source code analyzer. It supports Java, JavaScript, Apex and Visualforce, Modelica, PLSQL, Apache Velocity, XML, XSL, Scala.

Configure Code scanning

Anchore Syft SBOM Scan

By Anchore

Produce Software Bills of Materials based on Anchore's open source Syft tool.

Configure Code scanning

Anchore Grype Vulnerability Scan

By Anchore

Produce source and container vulnerability reports based on Anchore's open source Grype tool.

Configure Code scanning

Bandit Scan

By abirismyname

Bandit is free software designed to find common security issues in Python code, maintained by PyCQA

Configure Code scanning

Red Hat CodeReady Dependency Analytics

By Red Hat

Scan your project's dependencies with CodeReady Dependency Analytics.

Configure Code scanning

Frogbot Scan and Fix

By JFrog

Automatically creates pull requests with fixes for vulnerable project dependencies. Uses JFrog Xray to scan the project. Included as part of JFrog's free subscription.

Configure Code scanning

Frogbot Scan Pull Request

By JFrog

Automatically scans new pull requests for security vulnerabilities. Uses JFrog Xray to scan the project. Included as part of JFrog's free subscription.

Configure Code scanning

Haskell Dockerfile Linter

By GitHub Actions

A smarter Dockerfile linter that helps you build best practice Docker images.

Configure Code scanning

Policy Validator for CloudFormation

By Amazon Web Services

Validate AWS IAM Policies in CloudFormation Templates powered IAM Access Analyzer

Configure Code scanning

Policy Validator for Terraform

By Amazon Web Services

Validate AWS IAM Policies in Terraform Templates powered IAM Access Analyzer

Configure Code scanning

Pyre

By Meta

Pyre is a performant type checker for Python compliant with PEP 484. Pyre can analyze codebases with millions of lines of code incrementally – providing instantaneous feedback to developers as they write code.

Configure Code scanning

Pysa
By Meta

Python Static Analyzer (Pysa) is a security-focused static analysis tool that tracks flows of data from where they originate to where they terminate in a dangerous location.

Configure Code scanning

Snyk Container
By Snyk

Detect vulnerabilities in your container images and surface the issues in GitHub code scanning.

Configure Code scanning

Trivy
By Aqua Security

Scan Docker container images for vulnerabilities in OS packages and language dependencies with Trivy from Aqua Security.

Configure Code scanning

Brakeman
By Brakeman

Brakeman is a static analysis security vulnerability scanner for Ruby on Rails applications.

Configure Code scanning

clj-holmes
By Matheus Bernardes

A Static Application Security Testing tool to find vulnerable Clojure code via rules that use a simple pattern language.

Configure Code scanning

clj-watson
By GitHub Actions

Scan Clojure/Clojurescript projects for vulnerable direct/transitive dependencies.

Configure Code scanning

cloudrail
By Indeni Cloudrail

Cloudrail can be used to scan your infrastructure-as-code files for potential security and compliance issues.

Configure Code scanning

Credo Scan
By Credo

Credo is a static code analysis tool for the Elixir language with a focus on teaching and code consistency.

Configure Code scanning

42Crunch API Security Audit
By 42Crunch

Use the 42Crunch Audit to perform static API security testing (SAST) on OpenAPI/Swagger files.

Configure Code scanning

Datree
By Datree

Detect misconfigurations in your Kubernetes manifests and present them in Github code scanning

Configure Code scanning

Detekt
By Detekt

Static code analysis for Kotlin

Configure Code scanning

Flawfinder
By David A. Wheeler

Flawfinder is a simple program that scans C/C++ source code and reports potential security flaws.

Configure Code scanning

Kubesec
By Controlplane

Security risk analysis for Kubernetes resources. Submit pod-types (such as deployment, cronjob) to receive an itemised security risk score.

Configure Code scanning

lintr
By GitHub Actions

lintr provides static code analysis for R.

Configure Code scanning

mobsf
By mobsf

Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.

Configure Code scanning

Microsoft C++ Code Analysis

By Microsoft

Code Analysis with the Microsoft C & C++ Compiler for CMake based projects.

Configure Code scanning

PHPMD

By GitHub Actions

A spin-off project of PHP Depend and aims to be a PHP equivalent of the well known Java tool PMD.

Configure Code scanning

PSScriptAnalyzer

By Microsoft Corporation

A static code checker for PowerShell modules and scripts. PSScriptAnalyzer checks the quality of PowerShell code by running a set of rules.

Configure Code scanning

Prisma Cloud IaC Scan

By Palo Alto Prisma Cloud

Scan your Infrastructure as Code files with Prisma Cloud to detect security issues

Configure Code scanning

Psalm Security Scan

By psalm

Psalm is a static analysis tool for finding errors in PHP applications

Configure Code scanning

puppet-lint

By GitHub Actions

Puppet Lint tests Puppet code against the recommended Puppet language style guide.

Configure Code scanning

RuboCop Linting

By arthurnn

A Ruby static code analyzer and formatter, based on the community Ruby style guide.

Configure Code scanning

rust-clippy

By GitHub Actions

A collection of lints to catch common mistakes and improve your Rust code.

Configure Code scanning

OSSF Scorecard

By Open Source Security Foundation (OpenSSF)

Scorecard is a static supply-chain security analysis tool to assess the security posture of your project

Configure Code scanning

SecurityCodeScan

By @security-code-scan

Vulnerability Patterns Detector for C# and VB.NET

Configure Code scanning

Snyk Infrastructure as Code

By Snyk

Detect vulnerabilities in your infrastructure as code files and surface the issues in GitHub code scanning.

Configure Code scanning

Sobelow

By nccgroup

Sobelow is a security-focused static analysis tool for the Phoenix framework.

Configure Code scanning

SOOS DAST Scan

By SOOS

SOOS DAST is the easy-to-integrate no-limit web vulnerability scanner. Integrate SOOS DAST with your CI pipeline to find vulnerabilities by scanning a web app or APIs.

Configure Code scanning

tfsec

By tfsec

A static analysis security scanner for your Terraform code. Discover problems with your infrastructure before hackers do.

Configure Code scanning

Zscaler IaC Scan

By Zscaler CWP

Scan your Infrastructure as Code files using Zscaler Infrastructure as Code (IaC) Scan app

Configure Code scanning

7.4. Introducción al OWASP Benchmark Project

En el ámbito de la seguridad de aplicaciones existen numerosas herramientas para realizar pruebas estáticas, dinámicas e interactivas con el fin de identificar fallos de seguridad. Sin embargo, sin un método de evaluación estandarizado, es difícil comparar objetivamente su efectividad. El OWASP Benchmark Project proporciona esta referencia objetiva al ofrecer una plataforma de prueba unificada, exhaustiva y de código abierto.

El OWASP Benchmark es una aplicación web desarrollada en Java que incluye más de 2.700 casos de prueba en su versión actual (1.2). Cada caso de prueba está asociado a una categoría de vulnerabilidad específica (CWE) y puede ser analizado por diversas herramientas de seguridad. Las vulnerabilidades incluidas son explotables, lo que permite una evaluación realista de las capacidades de las herramientas para detectar y gestionar fallos de seguridad.

Las áreas de vulnerabilidad abarcadas incluyen, entre otras, inyección de comandos, criptografía débil, inyección SQL y XSS (Cross-Site Scripting). Cada caso de prueba es un servlet Java diseñado para ejecutarse de forma que las herramientas AST puedan analizar la aplicación en un entorno seguro y controlado.

A continuación, se muestra una tabla con el número de casos de prueba disponibles en la versión actual del benchmark, desglosada por área de vulnerabilidad:

Área de vulnerabilidad	# Tests	CWE ID
Command Injection	251	78
Weak Cryptography	246	327
Weak Hashing	236	328
LDAP Injection	59	90
Path Traversal	268	22
Secure Cookie Flag	67	614
SQL Injection	504	89
Trust Boundary Violation	126	501
Weak Randomness	493	330
XPATH Injection	35	643
XSS (Cross-Site Scripting)	455	79
Total Casos de Prueba	**2.740**	

Tabla 9. Casos de prueba por área de vulnerabilidad OWASP

Métricas de evaluación

El OWASP Benchmark Project se basa en dos métricas clave: la Tasa de Verdaderos Positivos (True Positive Rate, TPR) y la Tasa de Falsos Positivos (False Positive Rate, FPR).

Estas métricas permiten entender el desempeño de una herramienta tanto en la identificación de vulnerabilidades reales como en la generación de falsos positivos.

La siguiente gráfica presenta un espacio en el que se miden ambos índices. Cada punto en el gráfico representa el rendimiento de una herramienta específica en términos de TPR y FPR:

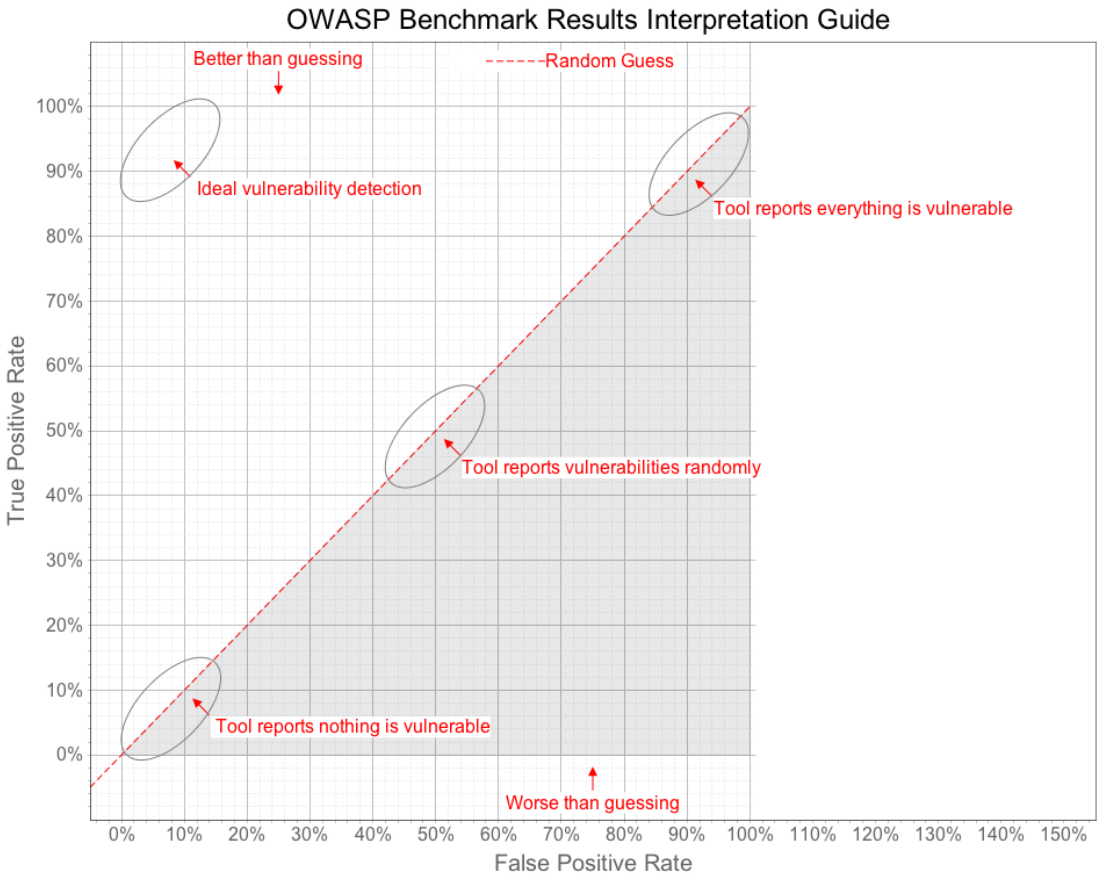


Ilustración 16. Guía de interpretación de resultados del OWASP Benchmark

En la gráfica:

- La parte superior izquierda representa el "ideal", donde la herramienta identifica correctamente muchas vulnerabilidades (alto TPR) y genera pocos falsos positivos (bajo FPR).
- La diagonal central indica un rendimiento similar al de una predicción aleatoria.
- La parte inferior derecha muestra un desempeño en el que la herramienta reporta todo como vulnerable, incluso cuando no lo es, lo cual indica un rendimiento incluso peor que el de una suposición aleatoria.

Además de esta representación gráfica, OWASP utiliza el Benchmark Accuracy Score, basado en el Índice de Youden, para calcular una puntuación de precisión para cada herramienta. El Índice de Youden se calcula como:

Índice de Youden = Sensibilidad + Especificidad – 1, donde la sensibilidad equivale al TPR y la especificidad a 1−FPR.

Un índice de 1 indica una precisión perfecta, mientras que un índice de 0 muestra que la herramienta no tiene capacidad para discriminar entre vulnerabilidades reales y falsas.

Este sistema de evaluación proporciona a los usuarios del OWASP Benchmark una visión clara y objetiva del rendimiento de distintas herramientas AST, facilitando una comparación imparcial basada en la precisión en la detección y clasificación de vulnerabilidades.

7.5. Importación del repositorio en GitHub

En este anexo se describe el proceso de importación de la aplicación “OWASP Juice Shop” en el repositorio del proyecto, y se detalla la ubicación de los workflows GitHub Actions importados junto con la aplicación.

La imagen siguiente muestra cómo puede importarse el código fuente de la aplicación de una manera sencilla indicándole al asistente de GitHub la dirección del repositorio de origen y el nombre que se desea darle al repositorio en destino:

- URL de origen: https://github.com/juice-shop/juice-shop.git
- Nombre del repositorio en destino: **JuiceShop**.

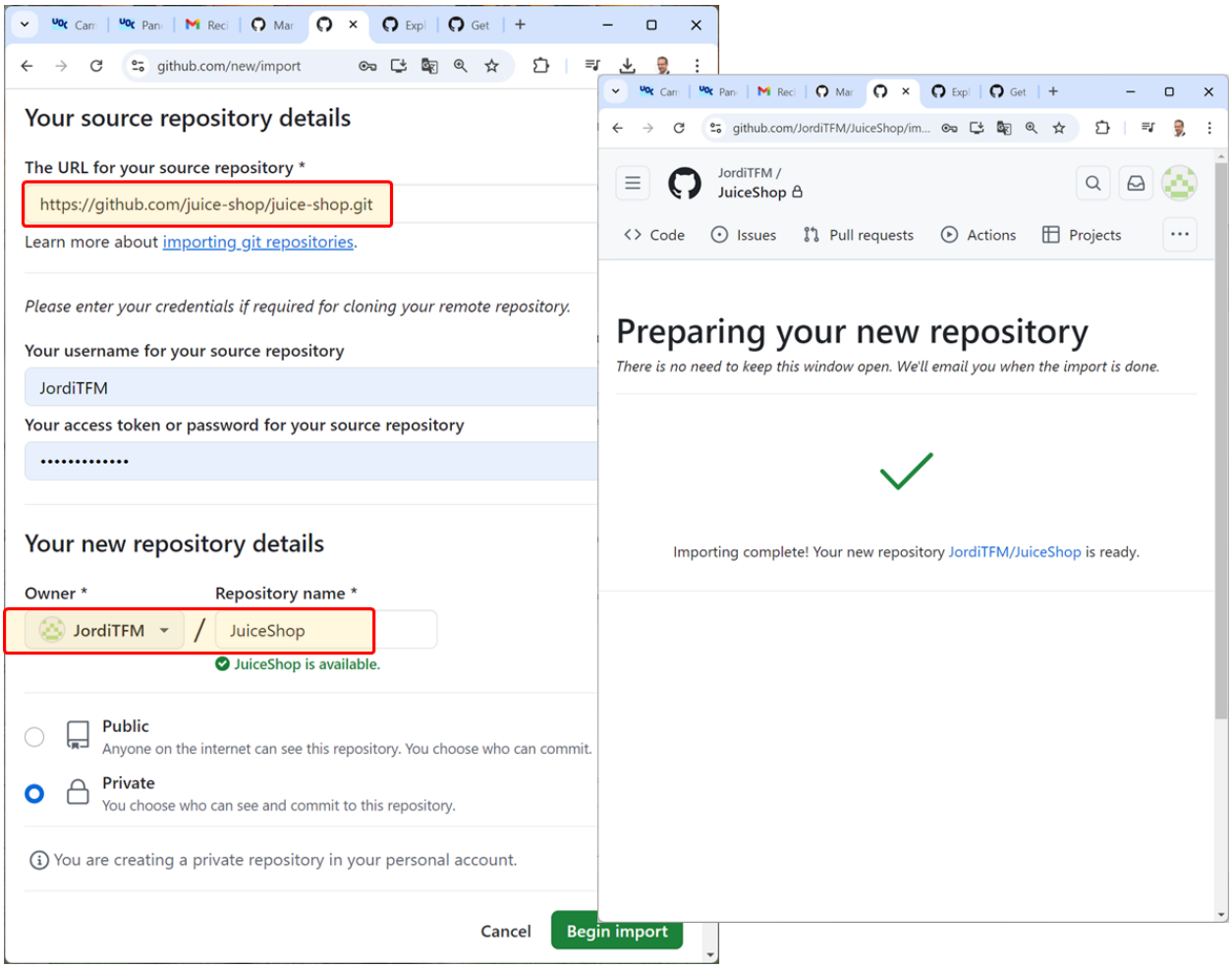


Ilustración 17. Importación del repositorio mediante GitHub. Fuente: Elaboración propia

Flujos de automatización

Navegando por el repositorio, en la carpeta “.github/workflows” se encuentran los flujos de automatización que han sido importados junto con la aplicación, tal como se aprecia en la imagen siguiente:

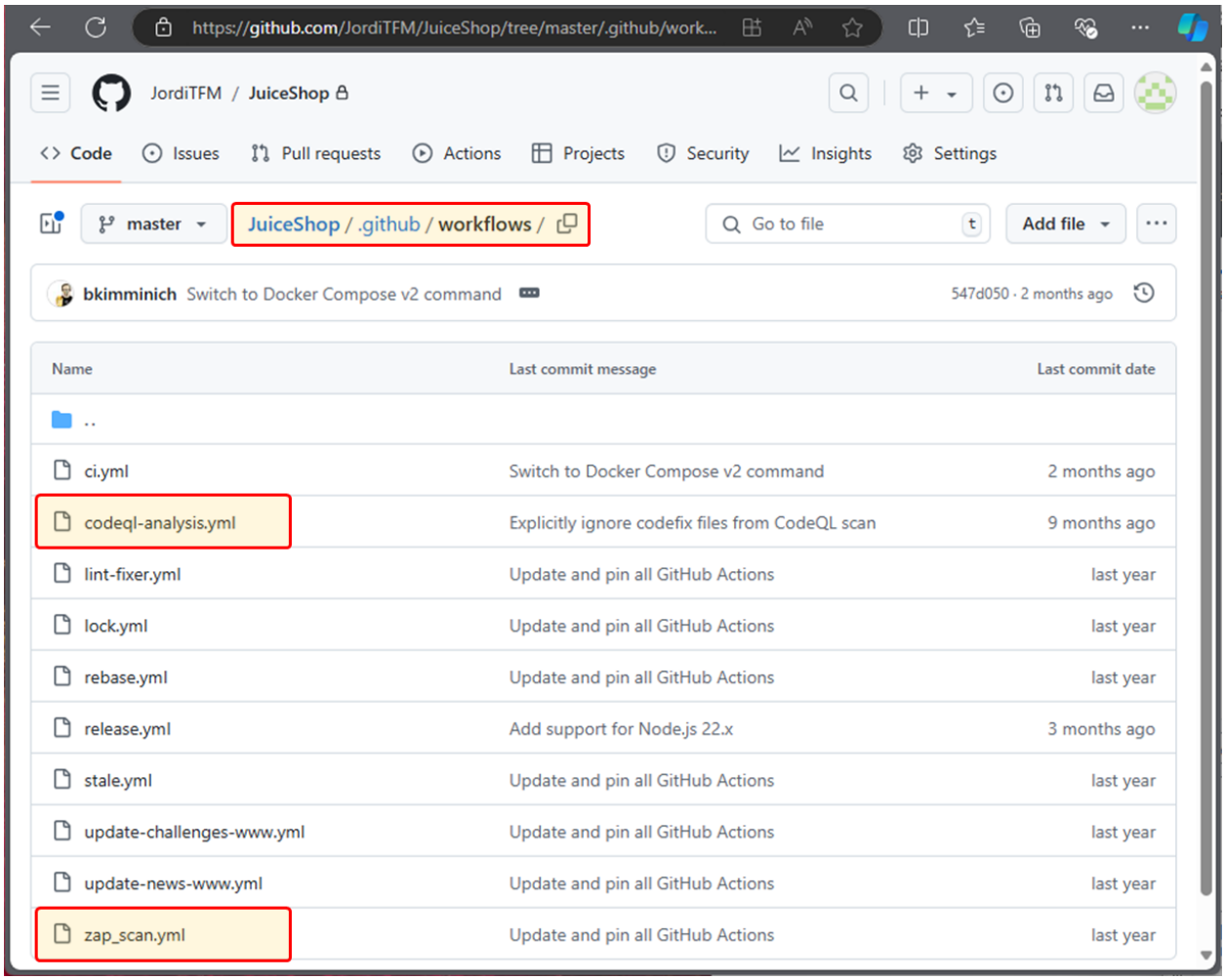


Ilustración 18. Archivos de workflow en GitHub Actions. Fuente: Elaboración propia

En la imagen de arriba se destacan dos flujos en particular, por estar relacionados con las pruebas de seguridad de la aplicación:

- **codeql-analysis.yml**: Configura el análisis de seguridad con CodeQL.
- **zap_scan.yml**: Configura un escaneo DAST con OWASP ZAP.

7.6. Configuración del entorno local de desarrollo y ejecución

Este anexo detalla los pasos necesarios para configurar un entorno local de desarrollo y ejecución, proporcionando instrucciones claras y visuales para instalar y desplegar OWASP Juice Shop. Permite a los desarrolladores clonar el repositorio, instalar las dependencias y verificar la aplicación en su máquina local. Se explica cómo configurar GitHub Desktop, instalar Node.js y npm, y la importancia de los archivos de dependencias para las pruebas SCA. Esta guía asegura que el entorno local esté correctamente configurado y operativo, ofreciendo una base sólida para el desarrollo y análisis de seguridad del proyecto.

7.6.1.Configuración del entorno local de desarrollo

GitHub Desktop se puede descargar gratuitamente desde la siguiente URL: https://desktop.github.com/download/.

Una vez descargado, es necesario autorizar su acceso a repositorios públicos y privados, así como a datos personales del usuario y a los archivos de workflows de GitHub Actions.

Después de la autorización, aparecerá la pantalla de configuración de Git en GitHub Desktop, donde el usuario puede definir el nombre y la dirección de correo que quedarán registrados en sus *commits*.

La imagen siguiente ilustra el proceso:

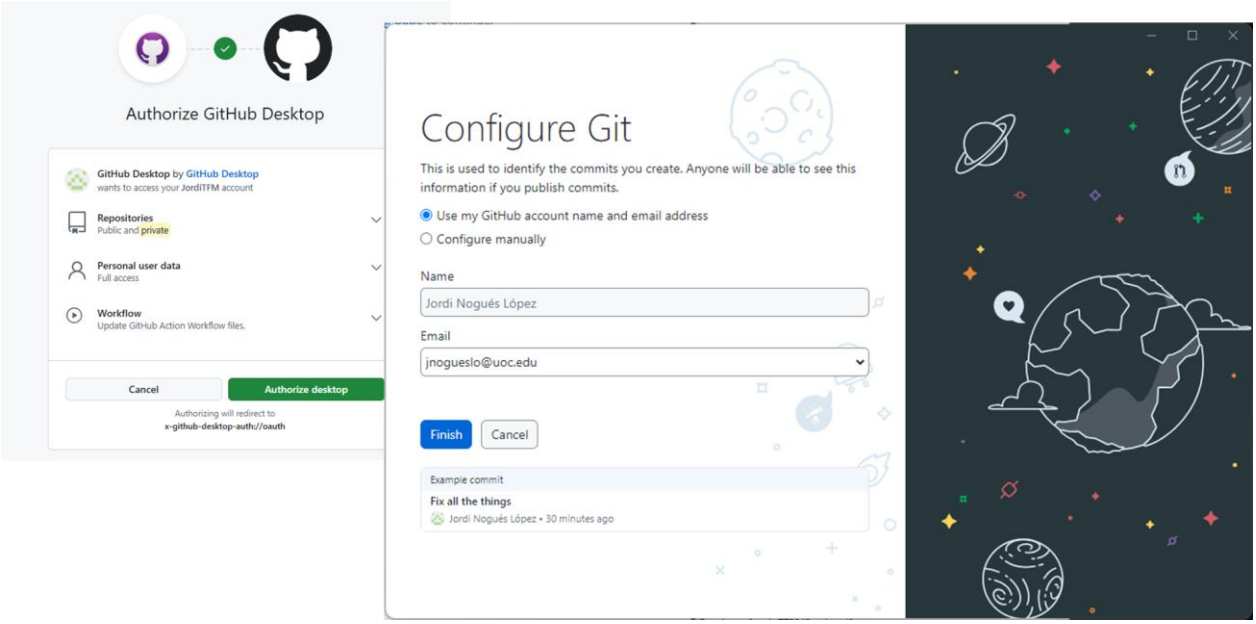


Ilustración 19. Configuración de GitHub Desktop. Fuente: Elaboración propia

Una vez instalado GitHub Desktop e iniciada la sesión con la cuenta de GitHub, se puede clonar el repositorio del proyecto en el entorno local mediante la opción “Clonar repositorio” que se encuentra en el menú “Archivo”.

7.6.2.Instalación del entorno de ejecución

Para ejecutar la aplicación es necesario instalar previamente **Node.js**, que es el entorno de ejecución para aplicaciones JavaScript en el lado del servidor, como OWASP Juice Shop.

Node.js no solo permite la ejecución del código, sino que también proporciona un **servidor web** integrado que permite acceder a la aplicación directamente desde un navegador sin necesidad de configurar servidores adicionales. Este servidor, que se activa automáticamente al ejecutar OWASP Juice Shop, facilita el despliegue y la prueba de la aplicación en entornos de desarrollo.

Al instalar Node.js, también se instala **npm** (Node Package Manager), que se utilizará para instalar las dependencias del proyecto listadas en el archivo **package.json**.

Para realizar la instalación de Node.js y npm, se deben seguir las instrucciones de la página de descarga: https://nodejs.org/en/download/package-manager.

Luego, dentro de la carpeta raíz del repositorio (donde se encuentra el archivo package.json), se deben **instalar las dependencias** del proyecto ejecutando el siguiente comando en la consola:

- ⇨ **npm install**

Este proceso generará archivos de dependencias, tales como **package-lock.json** y otros archivos **.lock** específicos de las dependencias instaladas. Estos archivos documentan las versiones exactas de cada paquete que la aplicación necesita para su correcto funcionamiento y tienen una función importante en el *test* **SCA**, ya que permiten identificar posibles vulnerabilidades en las versiones de las librerías instaladas.

Ejecución de la aplicación

Para iniciar la aplicación, debe abrirse una consola en la carpeta raíz del proyecto y ejecutar el comando:

- ⇨ **npm start**

Como se muestra en la imagen siguiente, este comando iniciará el servidor web proporcionado por Node.js, que permanecerá a la escucha en el puerto 3000:

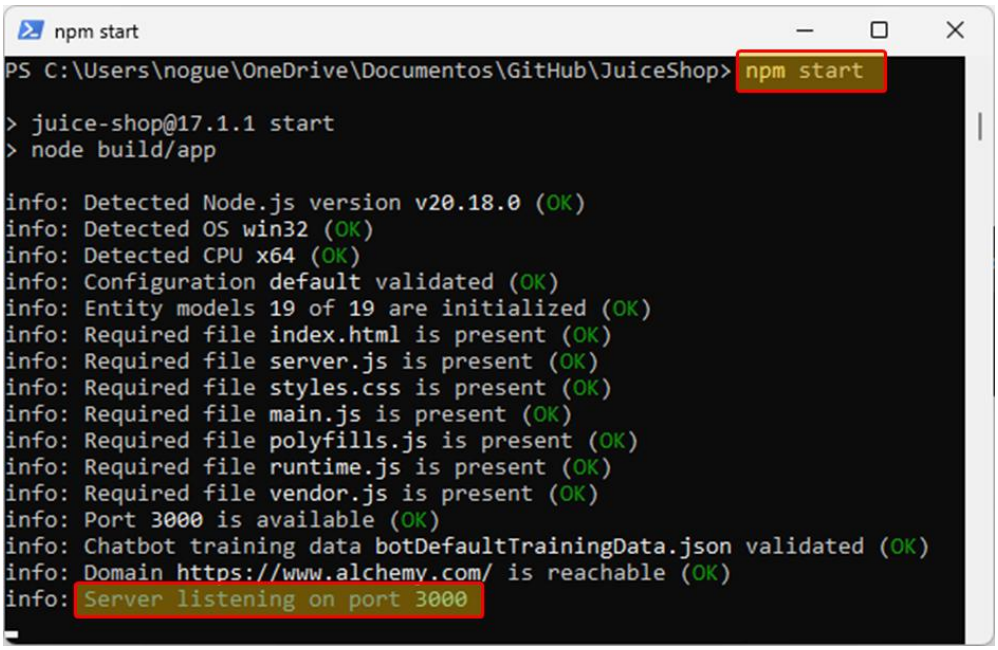


Ilustración 20. Arranque de la aplicación OWASP Juice Shop. Fuente: Elaboración propia

Verificación del funcionamiento de la aplicación

Para verificar que la aplicación funciona correctamente, se debe acceder a la URL http://localhost:3000 en un navegador. Allí se podrá navegar por la página inicial y comprobar su funcionamiento.

La imagen siguiente muestra la página inicial de la aplicación OWASP Juice Shop:

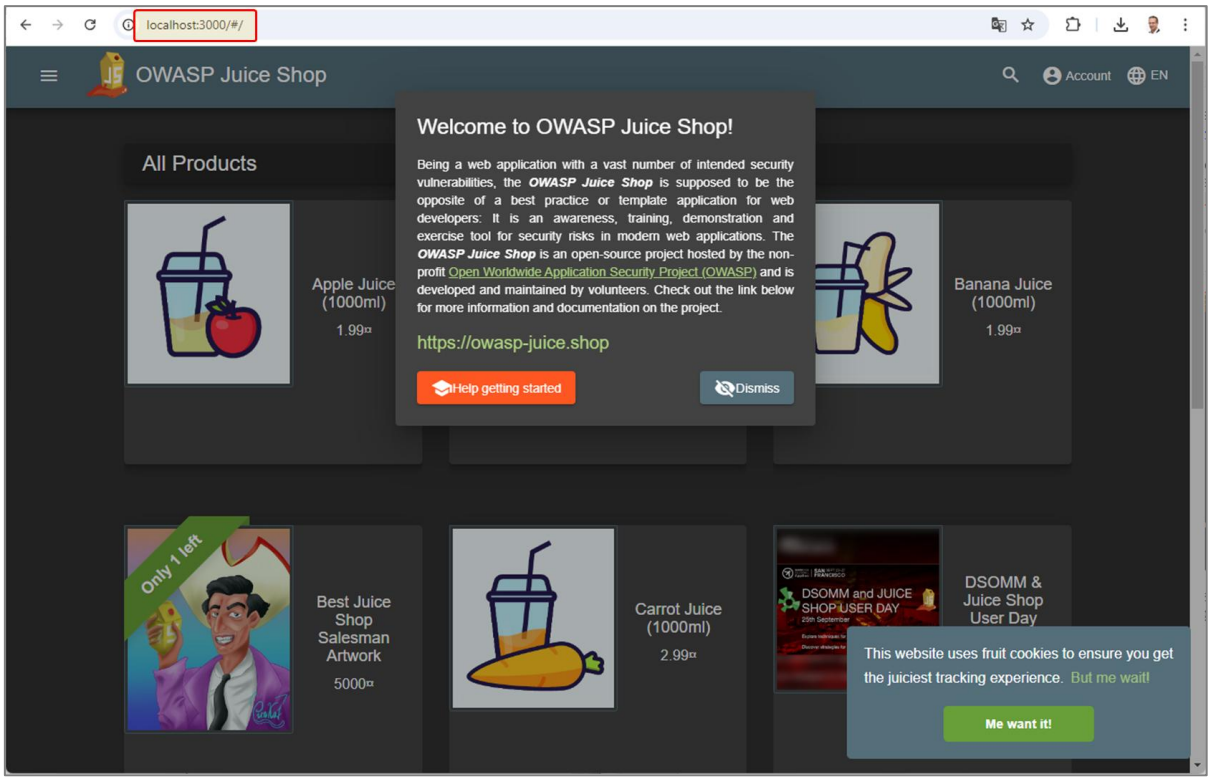


Ilustración 21. Página de inicio de la aplicación OWASP Juice Shop. Fuente: Elaboración propia

7.7. Configuración de CodeQL

Este anexo describe el procedimiento para habilitar y configurar CodeQL en un repositorio GitHub, destacando los pasos necesarios para ajustar la visibilidad del repositorio y activar esta funcionalidad; y para definir los lenguajes a analizar, los eventos que disparan los escaneos y el conjunto de consultas a utilizar (críticas o extendidas).

En primer lugar, hay que tener en cuenta que CodeQL requiere licencia Enterprise de GitHub, o bien que el repositorio sea **público**. En este caso, al no disponer de licencia Enterprise, para habilitar la funcionalidad de CodeQL se optará por hacer público el repositorio.

La imagen siguiente muestra cómo realizar el cambio de visibilidad de privado a público en la sección "**Danger Zone**" de la pestaña de configuración del repositorio:

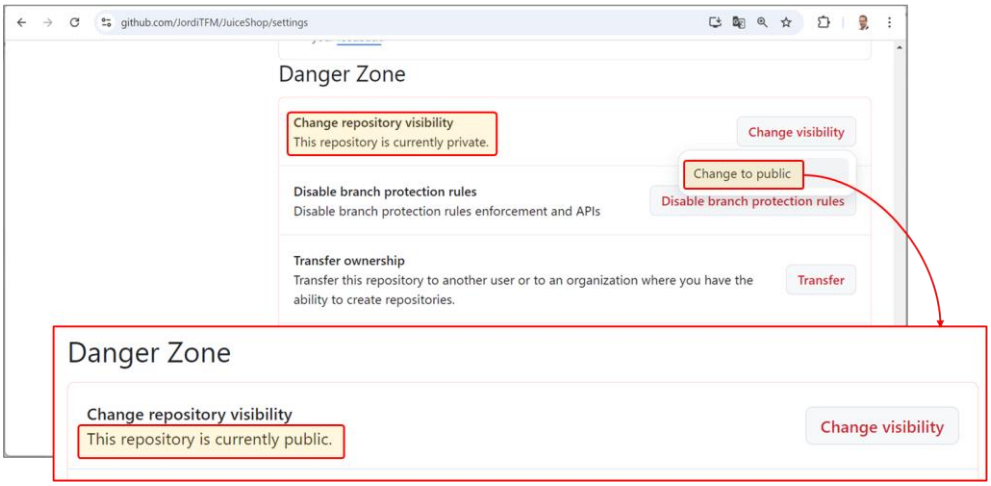


Ilustración 22. Cambio de la visibilidad del repositorio. Fuente: Elaboración propia

Una vez que el repositorio es público, puede accederse a la configuración de CodeQL para habilitarlo, ajustándola si es preciso, tal como se muestra en la imagen siguiente:

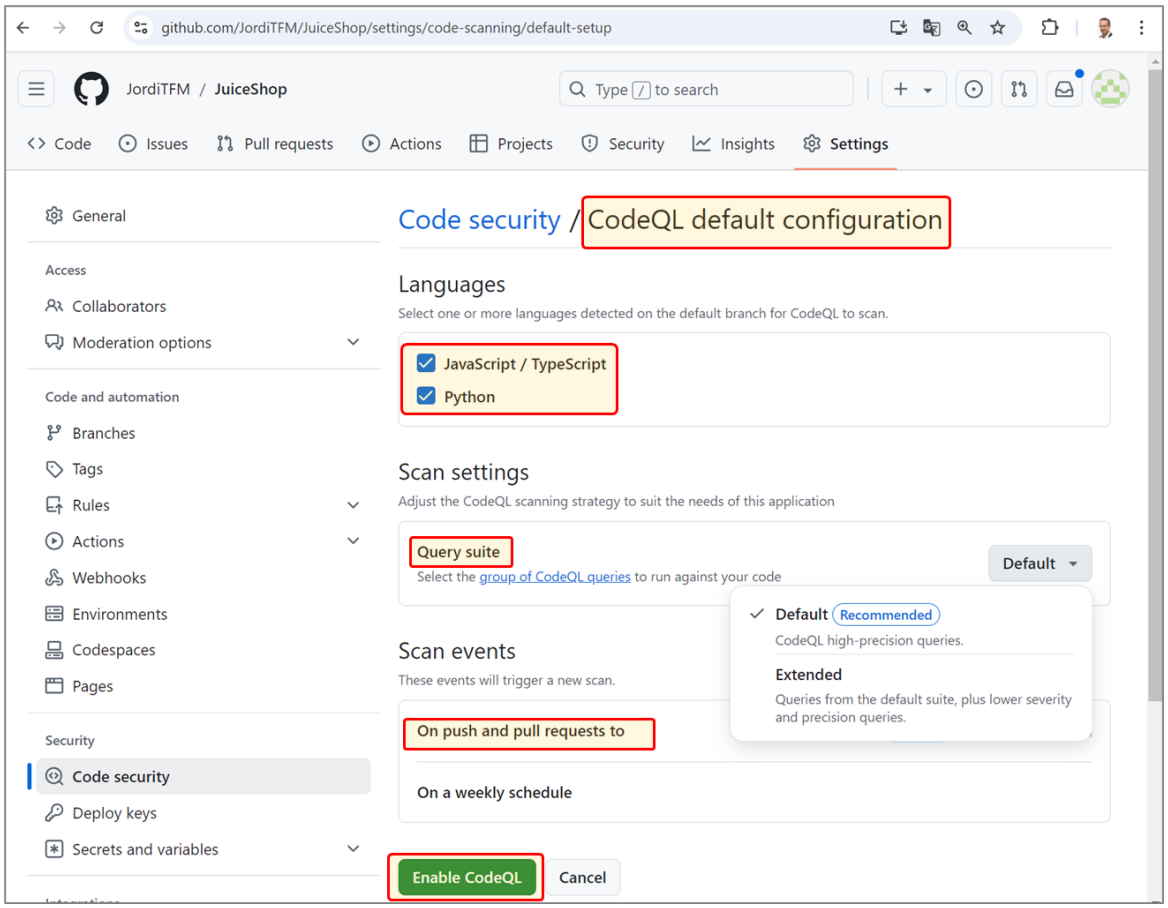


Ilustración 23. Configuración de CodeQL. Fuente: Elaboración propia

7.8. Experiencia de usuario en el escaneo SAST

Este anexo muestra de manera práctica la experiencia del desarrollador con la interfaz gráfica de GitHub Desktop y GitHub Actions en el contexto de un escaneo SAST con CodeQL, desde que realiza y publica una modificación en el código fuente de la aplicación, hasta la ejecución del escaneo SAST y la correspondiente generación de alertas.

Dicha secuencia, de forma simplificada, transcurre de la siguiente manera:

1- El desarrollador modifica un archivo cualquiera en su repositorio local.
2- Utilizando GitHub Desktop, solicita el *commit* para aceptar la modificación, y el *push* para sincronizarla en la rama principal del repositorio remoto.
3- Se lanza automáticamente el escaneo de CodeQL, y se generan las alertas que describen las vulnerabilidades de seguridad detectadas en el código fuente.

Para provocar el lanzamiento automático del flujo, bastará por ejemplo con editar cualquier archivo del repositorio, para después hacer un “push”.

En este caso, se fuerza la prueba modificando el archivo readme.md, y, como se puede observar en la imagen siguiente, GitHub Desktop detecta la modificación al instante, para que de una forma muy cómoda se pueda incorporar el cambio a la rama principal del repositorio local mediante ***commit***, según se ilustra en la siguiente imagen:

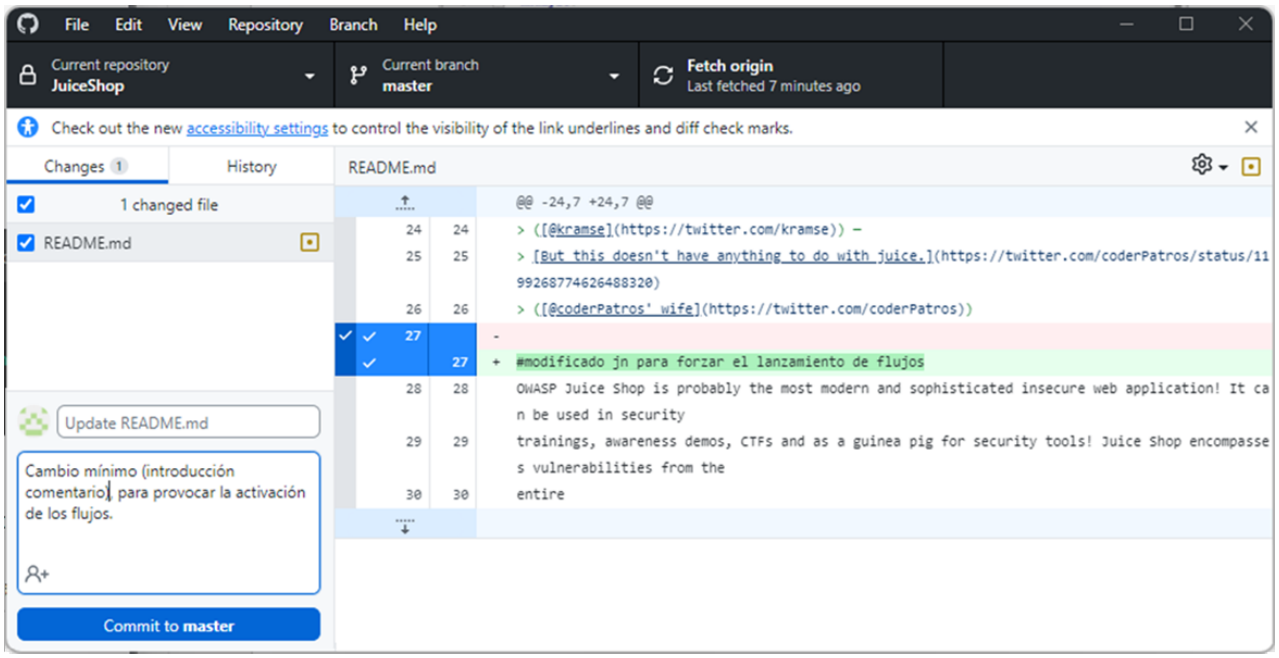


Ilustración 24. Commit en el repositorio local. Fuente: Elaboración propia

Tras pulsar “commit”, aparecerá en pantalla la opción ***push*** para que el cambio que se ha hecho en local se propague a la rama principal del repositorio remoto. La imagen siguiente muestra cómo debe realizarse la acción desde la pantalla de GitHub Desktop:

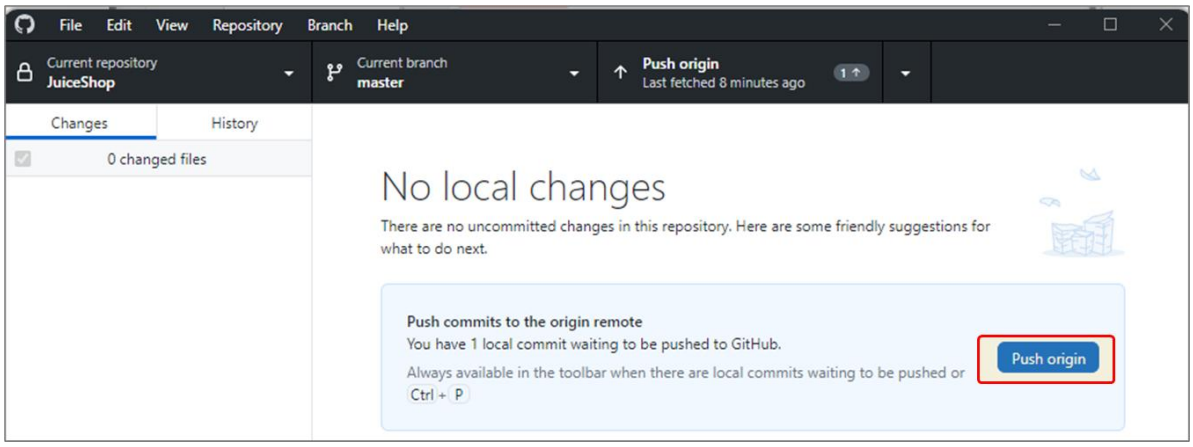


Ilustración 25. Push de los cambios al repositorio remoto. Fuente: Elaboración propia

Tras hacer el ***push***, el flujo de CodeQL se disparará automáticamente. Una vez finalice el escaneo, que en este caso será al cabo de unos 3 o 4 minutos de duración, a través de la pestaña "Actions" del repositorio GitHub se podrá consultar el resultado de la ejecución, tal como ilustra la imagen siguiente:

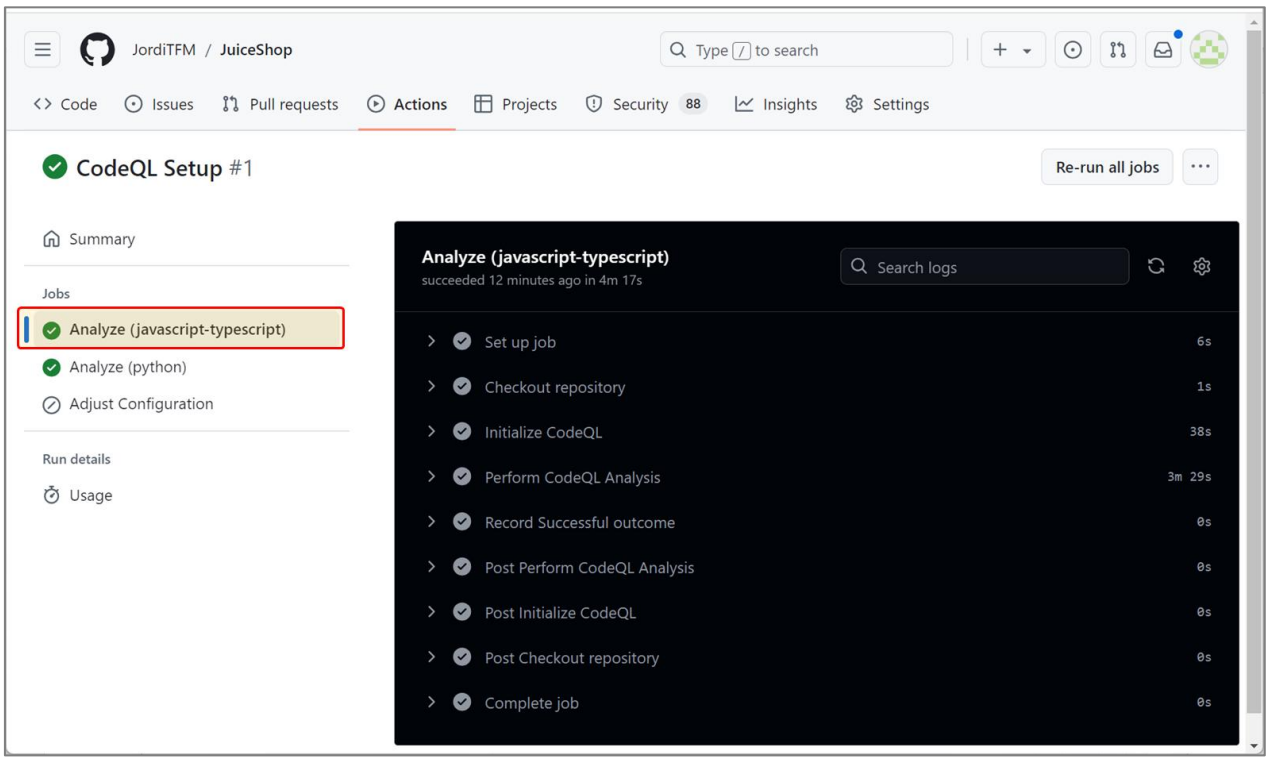


Ilustración 26. Consulta de los resultados del trabajo de CodeQL. Fuente: Elaboración propia

En la página puede consultarse si cada uno de los pasos ejecutados en el trabajo ha finalizado correctamente, o si se han producido errores.

7.9. Archivo SARIF: Función y estructura

Este anexo describe el resultado del escaneo del código realizado por CodeQL, generando un archivo con formato SARIF (Static Analysis Results Interchange Format), que contiene información detallada sobre cada una de las vulnerabilidades identificadas.

El formato SARIF permite que los resultados puedan integrarse en otras herramientas de análisis, compararlos con análisis anteriores, o generar informes personalizados.

Puede accederse a dicho archivo a través de la pestaña Actions del repositorio GitHub, como muestra la imagen de abajo, seleccionando a continuación el trabajo correspondiente al escaneo realizado:

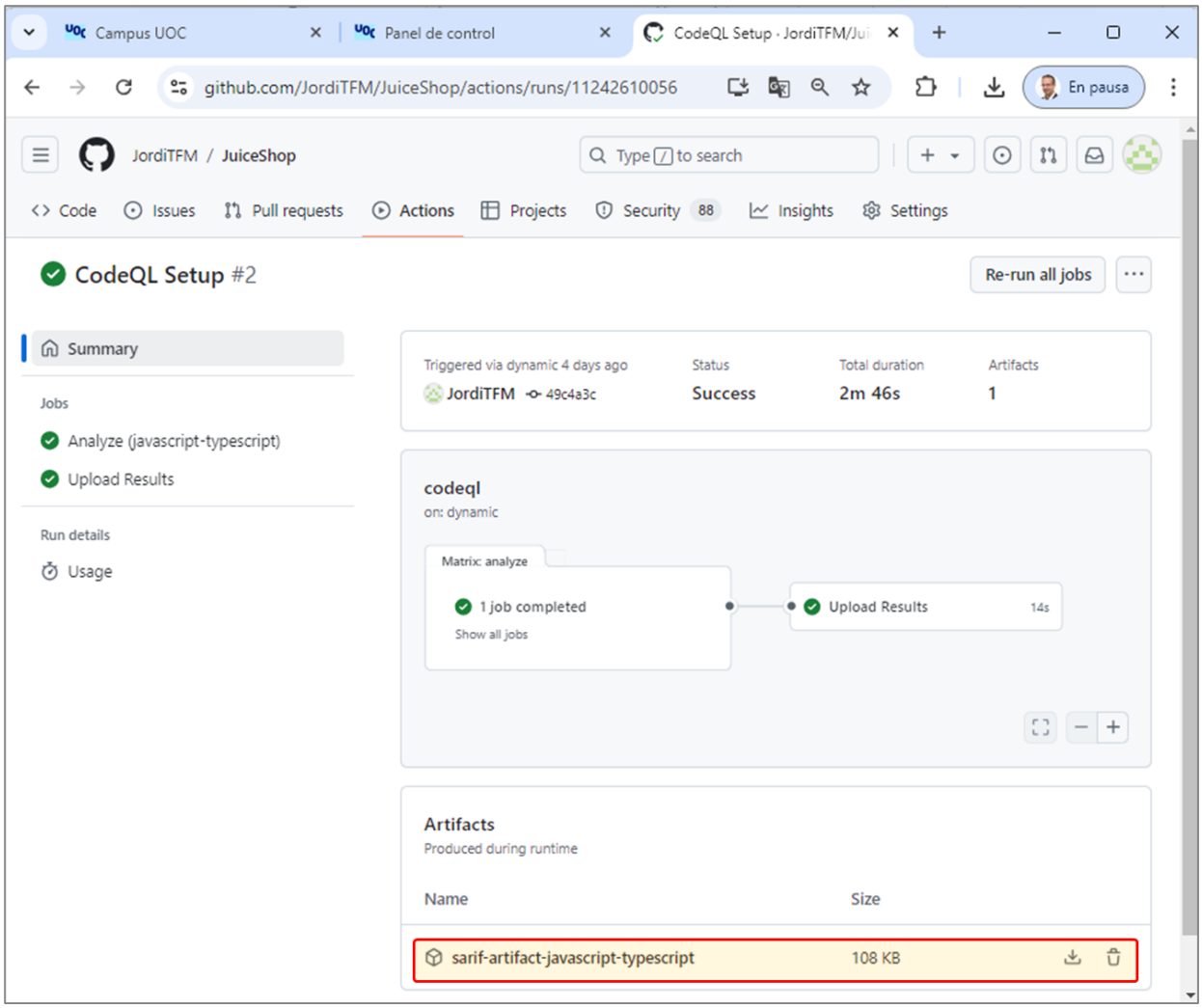


Ilustración 27. Exportación del archivo SARIF. Fuente: Elaboración propia

El archivo SARIF puede exportarse pinchando en el icono de descarga que aparece dentro del rectángulo resaltado en amarillo en la imagen de arriba.

A continuación se muestra, a título ilustrativo, un extracto de este archivo: *(Para una adecuada visualización en Notepad++, debe utilizarse el visor de formatos JSON)*

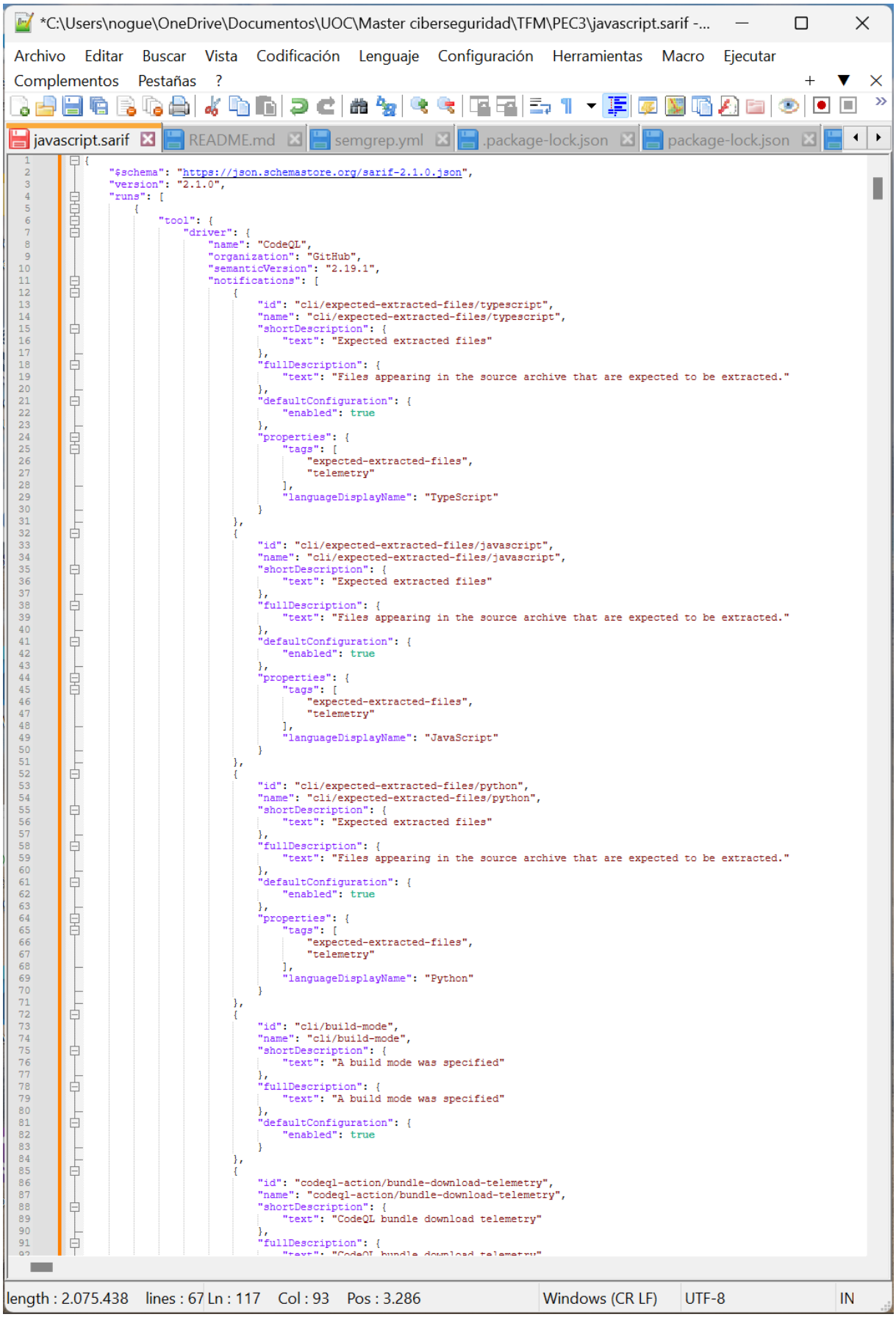


Ilustración 28. Estructura del archivo SARIF. Fuente: Elaboración propia

7.10. Gestión de vulnerabilidades con GitHub

Este anexo explica cómo gestionar las vulnerabilidades detectadas por CodeQL a través de la pestaña “Security” del repositorio GitHub, que proporciona una interfaz avanzada para visualizar, priorizar y remediar alertas de seguridad. Se indica cómo navegar por las vulnerabilidades identificadas, consultar su contexto específico y emplear las herramientas integradas para gestionarlas. También se explican opciones clave como descartar alertas irrelevantes, abrir incidencias asignables, aplicar correcciones automáticas con Copilot Autofix y consultar información adicional en la base de datos MITRE mediante el identificador CWE. A través de un ejemplo práctico, se ilustra cómo identificar y gestionar una vulnerabilidad concreta, destacando la utilidad de estas funciones para mantener un código seguro y eficiente.

En la imagen siguiente se observa cómo el escaneo realizado por CodeQL ha identificado **88 vulnerabilidades** en el código fuente de la aplicación:

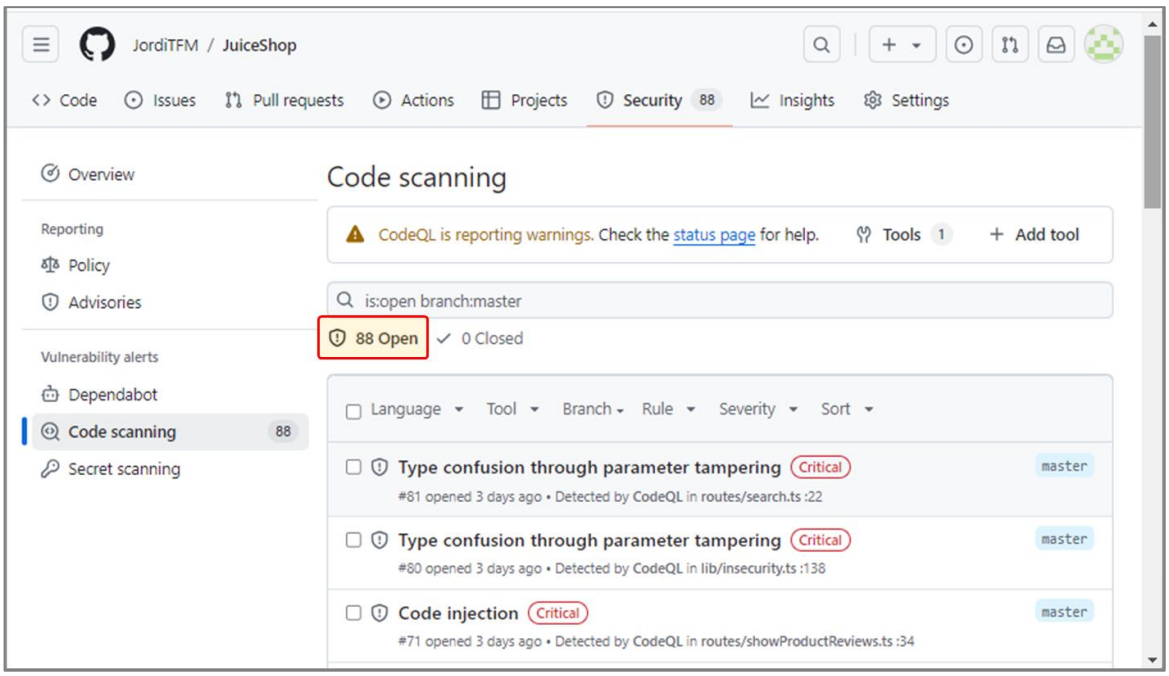


Ilustración 29. Listado de vulnerabilidades detectadas por CodeQL. Fuente: Elaboración propia

Pinchando sobre cualquier alerta en el listado, se carga una página con los detalles específicos de la vulnerabilidad. Esta página muestra el fragmento de código donde se ha detectado el problema, junto con una descripción detallada de la vulnerabilidad y recomendaciones para corregirla. Además, GitHub ofrece un conjunto de opciones para que el desarrollador o analista de seguridad gestione de manera efectiva cada alerta:

1- **Descartar la alerta**: Permite marcar una alerta como falso positivo o indicar que, por algún motivo, no se va a corregir. Esto ayuda a reducir el ruido y centrarse solo en las alertas críticas.

2- **Abrir una incidencia**: Facilita la creación de una tarea asignable, de manera que el problema pueda ser revisado y resuelto por el desarrollador o el equipo de seguridad correspondiente.

3- **Utilizar Copilot Autofix:** En aquellos casos en los que sea posible, Copilot puede sugerir y aplicar automáticamente una solución a la vulnerabilidad detectada, agilizando el proceso de remediación.

4- **Navegar a la página de MITRE** Cada alerta incluye un identificador CWE (Common Weakness Enumeration), que permite acceder a información oficial detallada sobre la vulnerabilidad en la página de MITRE. Esta página proporciona una explicación exhaustiva del problema, posibles vectores de ataque y recomendaciones de seguridad.

La imagen siguiente muestra la página de detalle de una alerta de seguridad detectada por CodeQL en GitHub. En esta vista se resaltan las cuatro opciones de gestión mencionadas anteriormente, que permiten al usuario interactuar con la alerta, ya sea para descartarla, crear una incidencia, aplicar una corrección automática con Copilot o consultar información detallada en la base de datos de MITRE a través del identificador CWE.

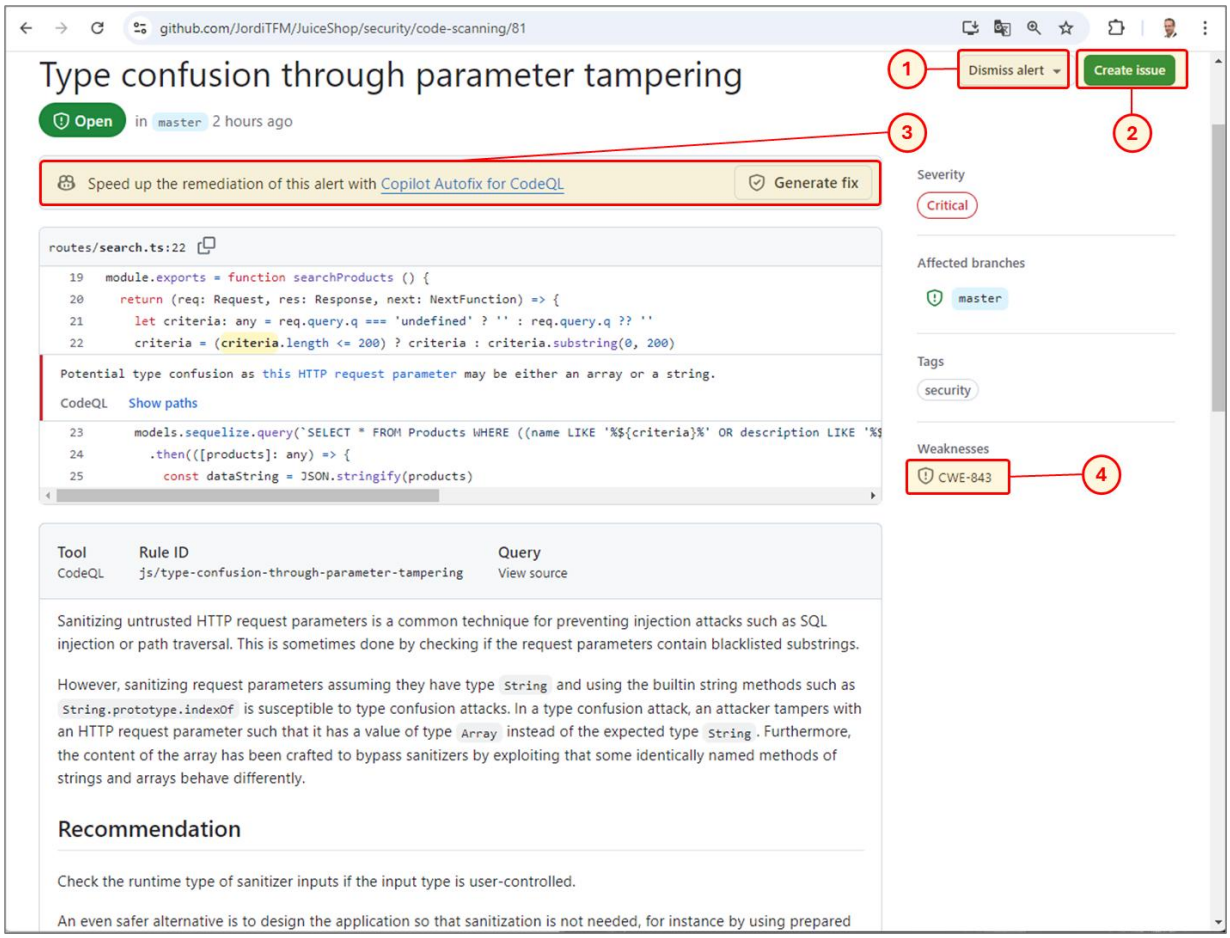


Ilustración 30. Gestión de vulnerabilidades en GitHub. Fuente: Elaboración propia

En la imagen anterior se observa un ejemplo específico de vulnerabilidad detectada por CodeQL: **CWE-843** (Confusión de tipos a través de la manipulación de parámetros). Esta vulnerabilidad consiste en que el código aplica un método de sanitización que asume que el parámetro recibido será siempre de tipo cadena (string), lo cual no se verifica adecuadamente.

Esta suposición es insegura, ya que un atacante podría manipular el parámetro en la solicitud HTTP y asignarle un valor de tipo array en lugar de string. Esto podría permitir al atacante explotar diferencias en el comportamiento de métodos con el mismo nombre, según si tratan cadenas o arrays, para evadir controles de seguridad o modificar la ejecución del código.

Al hacer click en el identificador CWE-843 de la alerta en GitHub, se abre la página oficial de MITRE con una descripción completa de la vulnerabilidad. Esta página proporciona detalles técnicos y posibles mitigaciones, como se muestra en la imagen siguiente:

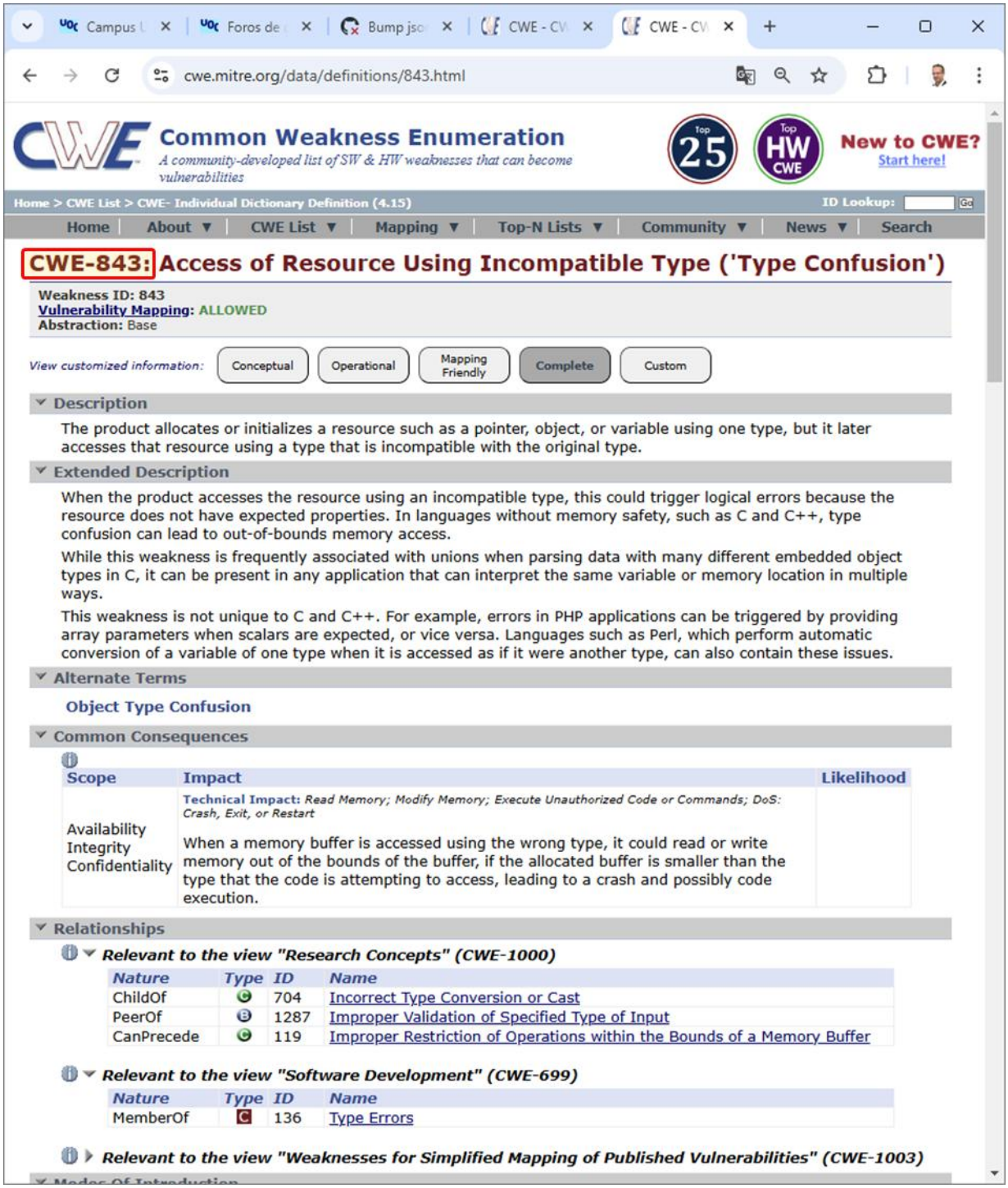


Ilustración 31. CWE-843: Type Confusion (MITRE, 2024)

7.11. Caso de uso de Copilot Autofix

Este anexo presenta un caso práctico del uso de **Copilot Autofix**, una funcionalidad avanzada de GitHub que utiliza la Inteligencia Artificial para proponer y aplicar correcciones automáticas a las vulnerabilidades detectadas en el código fuente.

Al estar integrada en GitHub, esta herramienta puede analizar el contexto de las vulnerabilidades detectadas por las herramientas AST y proponer correcciones automáticas en base a buenas prácticas de desarrollo seguro.

El proceso de remediación es rápido y sencillo: una vez que se identifica una vulnerabilidad, Copilot Autofix ofrece una propuesta de corrección específica que el desarrollador puede aceptar y aplicar automáticamente. Esta capacidad facilita una remediación rápida de las vulnerabilidades, reduciendo el tiempo de exposición a posibles ataques y favoreciendo el ritmo de desarrollo sin comprometer la seguridad.

La imagen siguiente muestra un ejemplo concreto de esta funcionalidad aplicada a una vulnerabilidad detectada en la aplicación OWASP Juice Shop. En este caso, la vulnerabilidad identificada corresponde a CWE-843 (*Type Confusion*), un problema que ocurre debido a la falta de verificación del tipo de dato en el parámetro *req.query.q* antes de utilizarlo en una consulta. Copilot Autofix sugiere una corrección basada en la inclusión de una verificación del tipo de dato, asegurando que el parámetro es una cadena (string) antes de que se procese en el código:

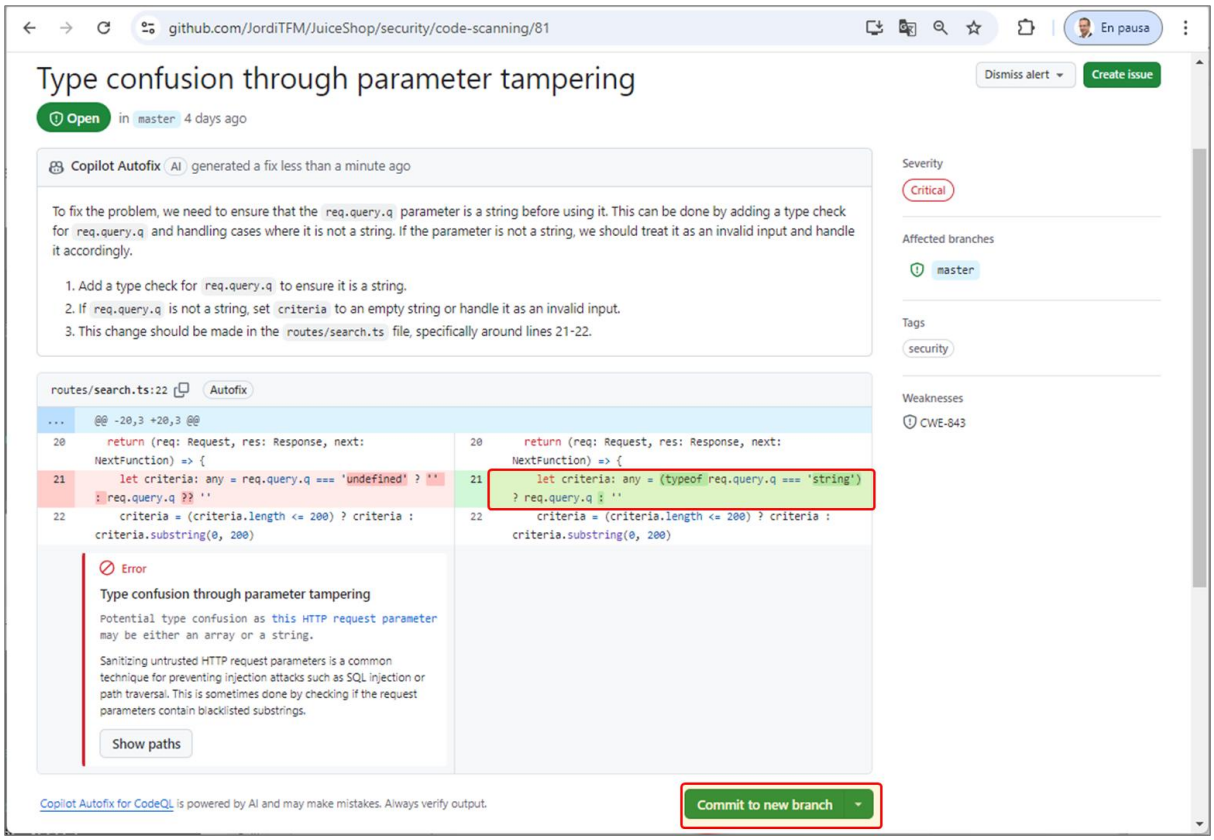


Ilustración 32. Autofix de Copilot para CWE-843. Fuente: Elaboración propia

En la imagen se observa la propuesta de Copilot Autofix y las opciones para aplicarla directamente. El desarrollador puede optar por realizar el cambio en la rama principal o en una nueva rama de revisión. Esta flexibilidad permite al desarrollador elegir la mejor estrategia según las necesidades y las políticas del proyecto.

7.12. Configuración de Dependabot

Este anexo detalla el proceso de configuración de Dependabot en GitHub, destacando las opciones disponibles para personalizar la gestión de alertas y las actualizaciones automáticas de dependencias. Desde la sección "Code security" del repositorio, se puede activar fácilmente la funcionalidad "Dependabot alerts" y acceder a herramientas avanzadas como Dependabot rules, que permiten definir criterios específicos para gestionar vulnerabilidades basados en severidad, ecosistemas, paquetes, o identificadores como CWE, CVE y GHSA.

En la imagen siguiente se muestra cómo habilitar Dependabot desde el menú de configuración del repositorio, resaltando las opciones principales para gestionar alertas y crear reglas personalizadas:

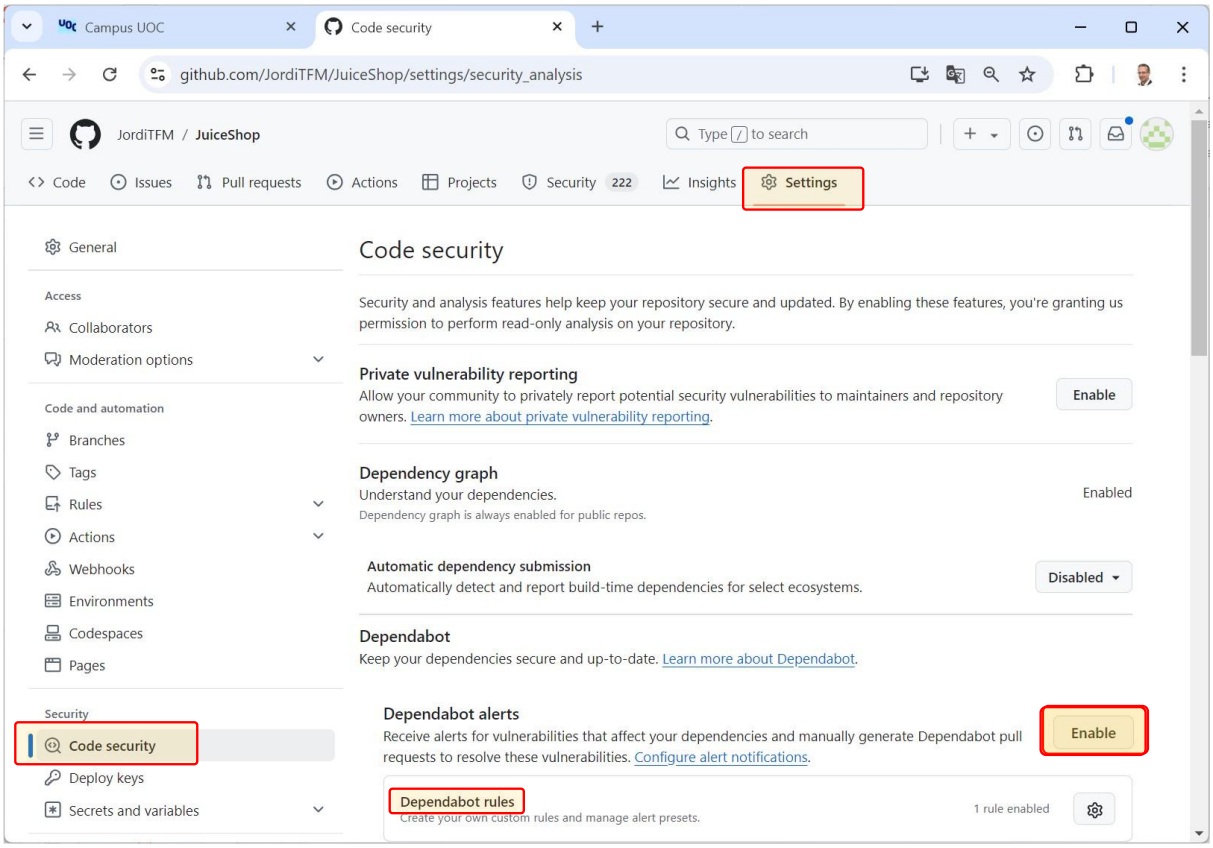


Ilustración 33. Activación del escaneo de Dependabot. Fuente: Elaboración propia

En la parte inferior de la misma imagen se puede observar la etiqueta “Dependabot rules”, una funcionalidad que permite crear reglas personalizadas para la gestión de las alertas. A continuación, se detallan las distintas funcionalidades de personalización que Dependabot ofrece a través de este interfaz gráfico:

Dependabot rules

Esta funcionalidad permite personalizar los criterios para aplicar acciones automatizadas sobre las alertas de seguridad, basándose en diversos parámetros:

- Severidad: Permite especificar la severidad de las alertas que activarán la regla. Las opciones incluyen valores como critical, high, moderate, y low. Este filtro ayuda a centrarse en alertas de acuerdo con su nivel de gravedad.
- Paquete: Se puede indicar el nombre del paquete específico en el que se desean aplicar las reglas. Esto permite que las reglas se enfoquen en dependencias específicas dentro del proyecto.
- Ecosistema: Permite seleccionar el ecosistema o gestor de paquetes (como npm, pip, maven, entre otros). Esto facilita la aplicación de reglas a los paquetes manejados por un ecosistema particular, especialmente útil en proyectos que dependen de múltiples gestores.
- Alcance: Define si la regla se aplicará a dependencias en entornos de ejecución o de desarrollo. Esto permite tratar de forma diferenciada las dependencias que se utilizan solo en el desarrollo y aquellas que están presentes en el entorno de ejecución.
- Archivo de manifiesto: Permite apuntar a archivos de manifiesto específicos en el proyecto, como package.json o requirements.txt, para aplicar las reglas a las dependencias documentadas en esos archivos.
- CWE: Filtra las alertas según el tipo de vulnerabilidad (CWE).
- CVE ID: Filtra por una vulnerabilidad en concreto (CVE).
- GHSA ID (GitHub Security Advisory ID): Filtra las alertas basándose en el identificador de asesoría de seguridad de GitHub. Esto es útil para aplicar reglas a vulnerabilidades reportadas específicamente por GitHub.

Dependabot ofrece dos opciones de acción para las alertas que cumplen con estos criterios:

- Ignorar las alertas temporalmente (hasta que haya un parche) o de forma indefinida.
- Crear un *pull-request* para solucionar las vulnerabilidades que coincidan con los criterios configurados en la regla.

Dependabot security updates

Al habilitar esta opción, Dependabot intenta automáticamente abrir *pull-requests* para actualizar las dependencias vulnerables con parches disponibles. Si se necesitan configuraciones más avanzadas para controlar estas actualizaciones automáticas, se pueden gestionar mediante las reglas de Dependabot descritas en el apartado anterior.

Grouped security updates

Agrupa todas las actualizaciones de seguridad disponibles en un único *pull-request* por cada gestor de paquetes y directorio de archivos de dependencias. Esta opción simplifica la revisión y fusión de cambios de seguridad, aunque se puede personalizar mediante reglas en dependabot.yml para crear agrupaciones diferentes.

Dependabot version updates

Permite que Dependabot abra automáticamente *pull-requests* para mantener las dependencias actualizadas cuando se liberan nuevas versiones. Esta opción ayuda a asegurar que el proyecto utiliza las versiones más recientes de sus dependencias. Para una configuración avanzada, se puede editar el archivo dependabot.yml.

Dependabot on Actions runners

Activa las actualizaciones de seguridad y de versiones de Dependabot en los runners de GitHub Actions. Esto asegura que las dependencias de GitHub Actions también estén actualizadas y sean seguras, lo que es especialmente útil para flujos de trabajo automatizados.

7.13. Gestión del *pull-request* en GitHub

Este anexo muestra cómo se gestiona un *pull-request* en GitHub, destacando los pasos necesarios para verificar la seguridad y la calidad del código antes de integrarlo en la rama principal.

A continuación, se presenta una imagen que ilustra el proceso completo, desde la descripción de los cambios realizados hasta la validación de las reglas de aprobación configuradas. Estas reglas aseguran que los cambios sean revisados por usuarios autorizados y que pasen todas las verificaciones automáticas antes de proceder con el merge del *pull-request*.

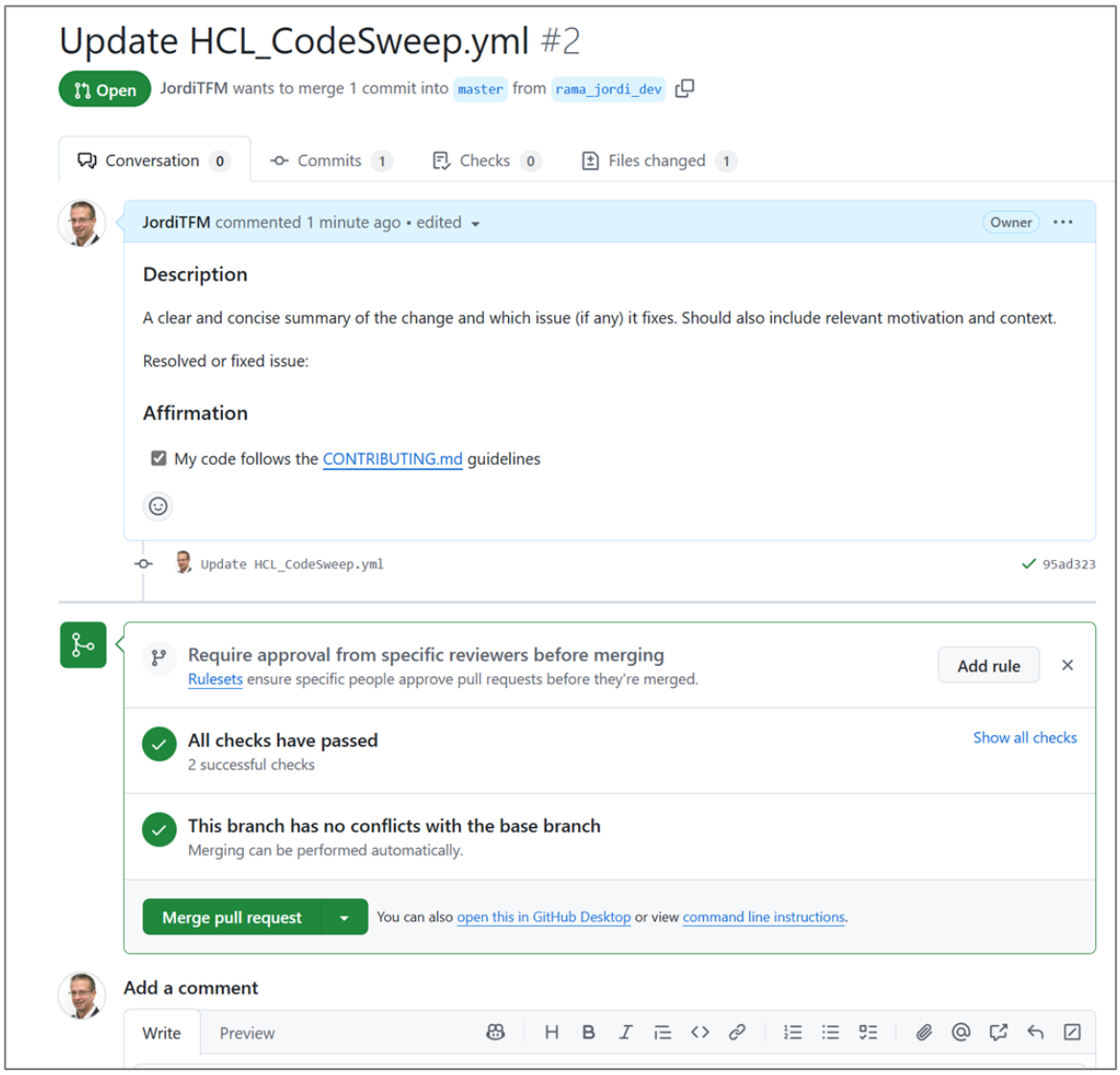


Ilustración 34. Gestión del pull-request en GitHub. Fuente: Elaboración propia

En la sección "Description", el autor ha proporcionado un resumen de los cambios realizados y ha confirmado que el código cumple con las buenas prácticas definidas para el proyecto en el archivo CONTRIBUTING.md.

En la sección de aprobación de revisores y reglas de aprobación, se muestra que la herramienta permite definir reglas para garantizar que los cambios pasan por un

revisor autorizado antes de proceder con el merge del *pull-request* que consolidará los cambios desde la rama de desarrollo a la rama principal.

En este caso, se muestra que el *pull-request* ha superado todos los checks automáticos configurados en GitHub. Esto significa que las verificaciones automáticas, que podrían incluir pruebas de integración, escaneos SCA, escaneos SAST u otros checks de seguridad, han sido exitosas.

7.14. Listado de vulnerabilidades SCA

En este anexo se muestra el listado de vulnerabilidades detectadas en el escaneo SCA realizado por Dependabot. Desde la sección "Dependabot Alerts" de la pestaña "Security", GitHub permite visualizar y gestionar las alertas de seguridad relacionadas con las dependencias del proyecto.

En la imagen siguiente se presentan las 13 vulnerabilidades identificadas, clasificadas según su severidad y los paquetes afectados:

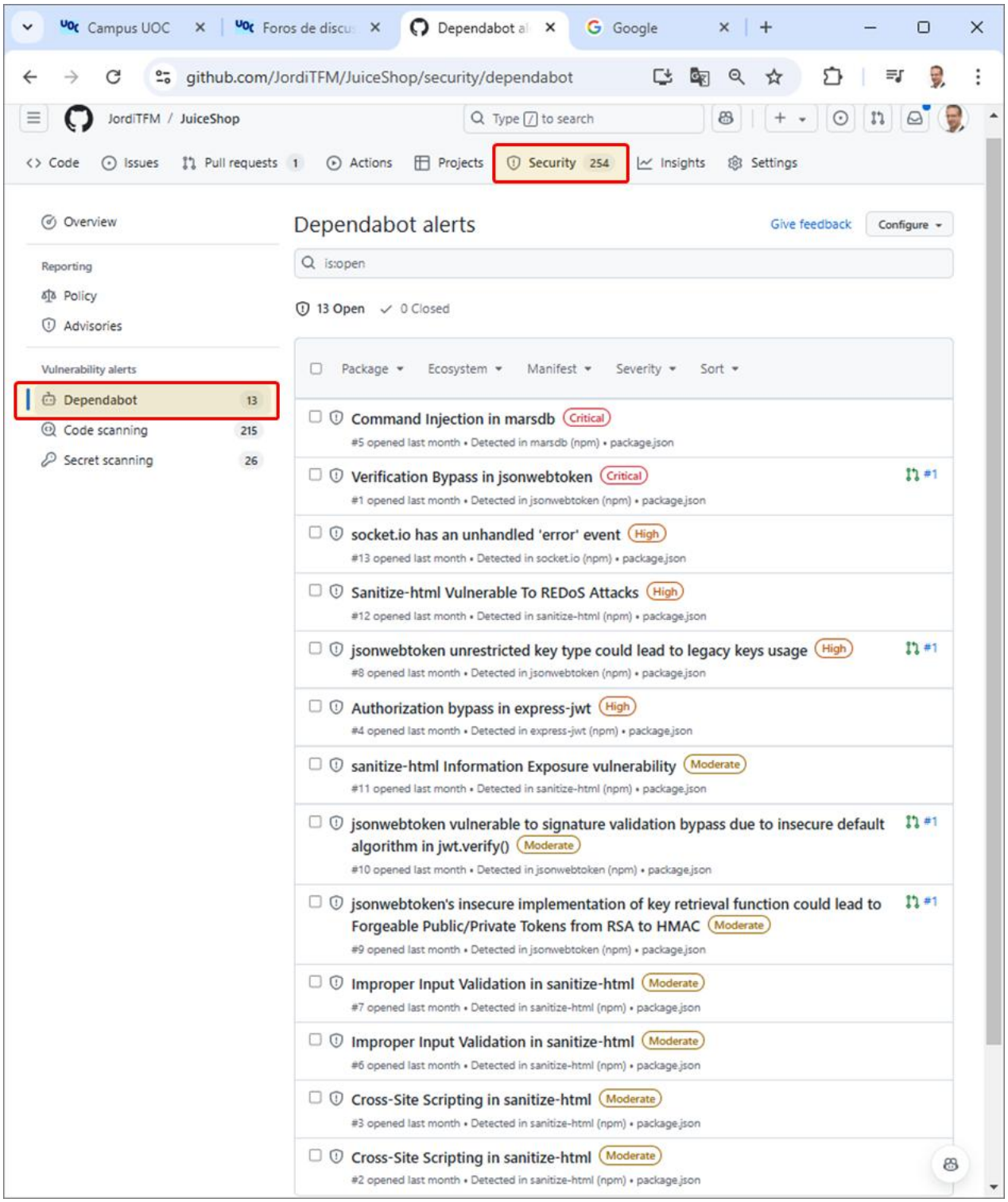


Ilustración 35. Listado de alertas detectadas por Dependabot. Fuente: Elaboración propia

7.15. Remediación automática con Dependabot

Este anexo describe el proceso de remediación automática que Dependabot facilita para resolver vulnerabilidades en las dependencias del proyecto, ejemplificado mediante la actualización de la librería **jsonwebtoken** para solucionar varias alertas de seguridad críticas.

En imagen siguiente se ilustra cómo Dependabot propone un pull-request que puede ser fusionado en la rama principal del repositorio, tras verificar los cambios sugeridos:

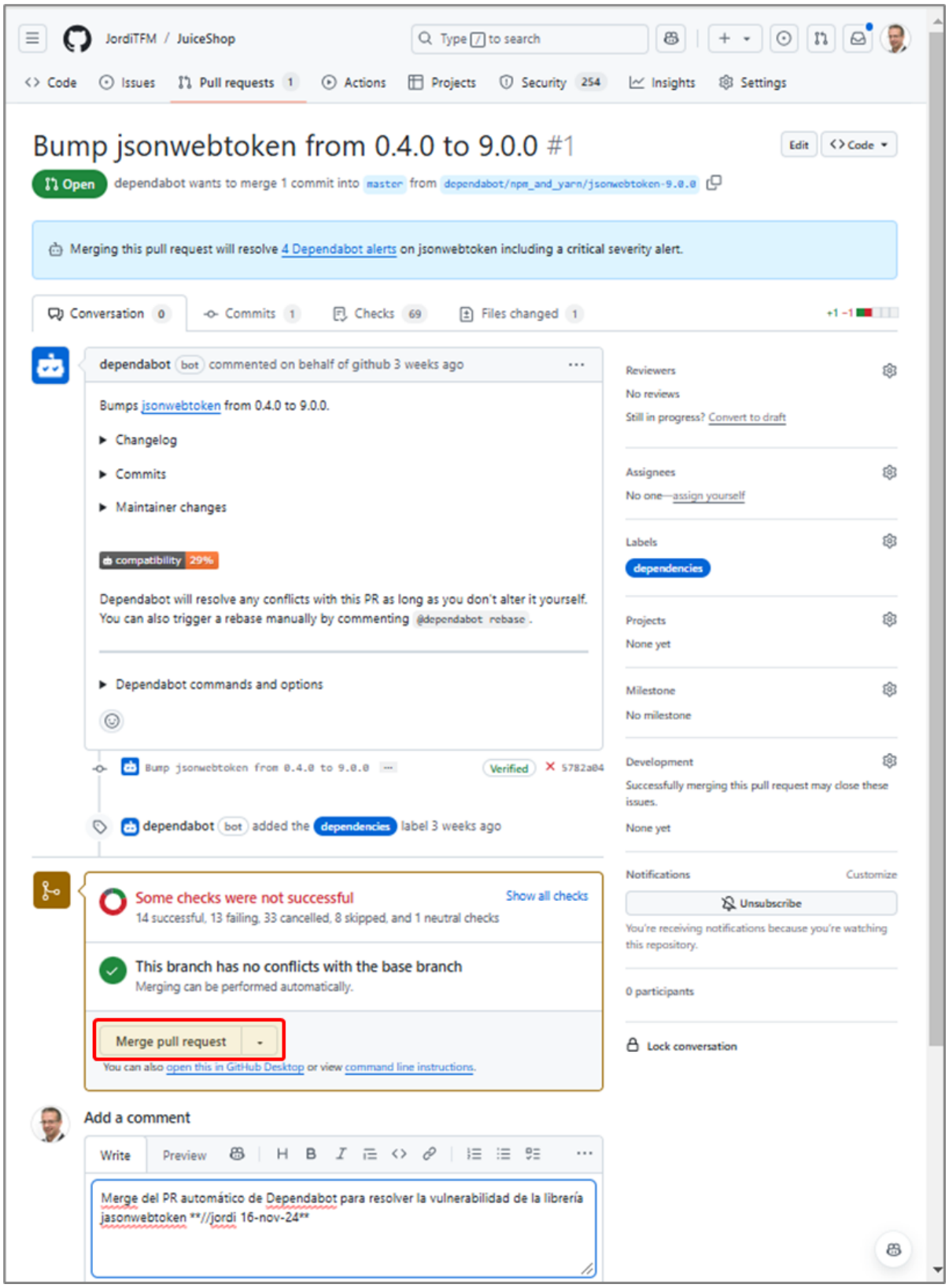


Ilustración 36. Actualización automática de la librería jsonwebtoken. Fuente: Elaboración propia

Al pinchar el botón “Merge pull request” que aparece resaltado en amarillo en la imagen anterior, se despliegan las 3 posibles fórmulas de fusionado que nos permite realizar la herramienta, ilustradas en la siguiente captura:

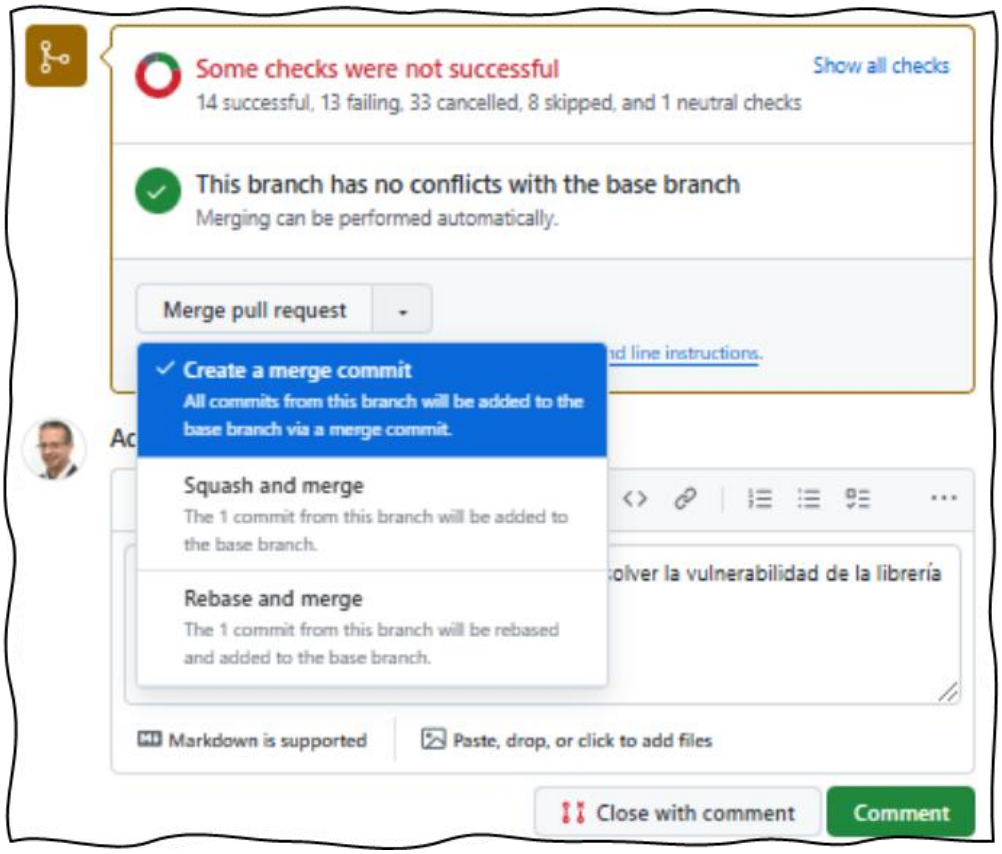


Ilustración 37. Opciones de fusionado en GitHub. Fuente: Elaboración propia

Estas opciones permiten decidir cómo se integrarán los cambios en la rama principal del repositorio:

- ***Create a merge commit:*** Esta opción crea un *commit* adicional que combina los cambios de la rama origen en la rama principal, preservando el historial completo de los commits realizados en la rama origen. Es útil si se desea mantener un registro detallado de todos los cambios.
- ***Squash and merge*** (Alisar y fusionar): Combina todos los *commits* de la rama en un único *commit* antes de fusionarlos en la rama principal. Esta opción simplifica el historial al eliminar los *commits* intermedios, dejando solo un resumen de los cambios.
- ***Rebase and merge****:* Reaplica los *commits* de la rama origen en la parte superior de la rama principal, creando un historial lineal. Esto evita los *commits* de *merge* pero conserva todos los *commits* individuales de la rama origen.

En este caso, se seleccionará la opción “Squash and merge” para mantener un historial de cambios lo más claro posible, sin información superflua.

Una vez seleccionada la opción “Squash and merge”, la página mostrará un resumen de toda la información necesaria para entender los detalles del cambio, y presentará, tal como se destaca en el recuadro amarillo de la siguiente imagen, el botón para confirmar la acción:

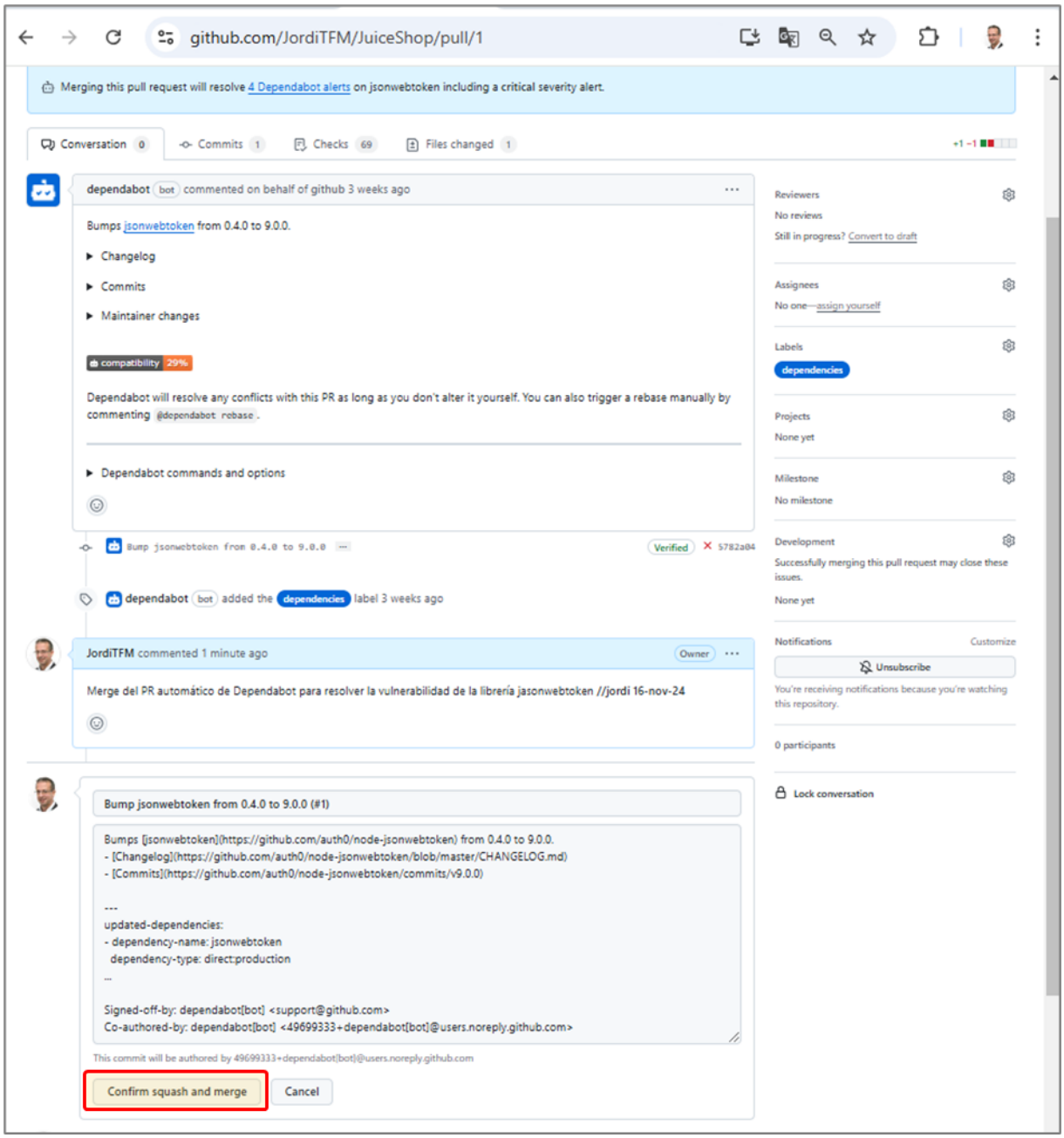


Ilustración 38. Página de confirmación de la acción de fusionado. Fuente: Elaboración propia

Una vez confirmada la acción de fusionado y, por consiguiente, actualizada la versión de la librería “jsonwebtoken” en el archivo “package.json”, habrán quedado resueltas las cuatro vulnerabilidades identificadas por Dependabot, asociadas al mismo componente.

7.16. Escaneo del repositorio GitHub desde la aplicación Semgrep

Este anexo describe el proceso de configuración y ejecución de un escaneo SAST utilizando la versión de evaluación de la herramienta **Semgrep**, disponible en su sitio web https://semgrep.dev, mediante la opción “Try for free”.

Semgrep se conecta a GitHub mediante OpenID Connect y OAuth 2.0, solicitando los permisos necesarios para interactuar con los repositorios.

Las siguientes imágenes muestran el proceso de autenticación en GitHub, la creación de una organización en Semgrep, y la configuración del propósito del uso, en este caso indicando un uso académico:

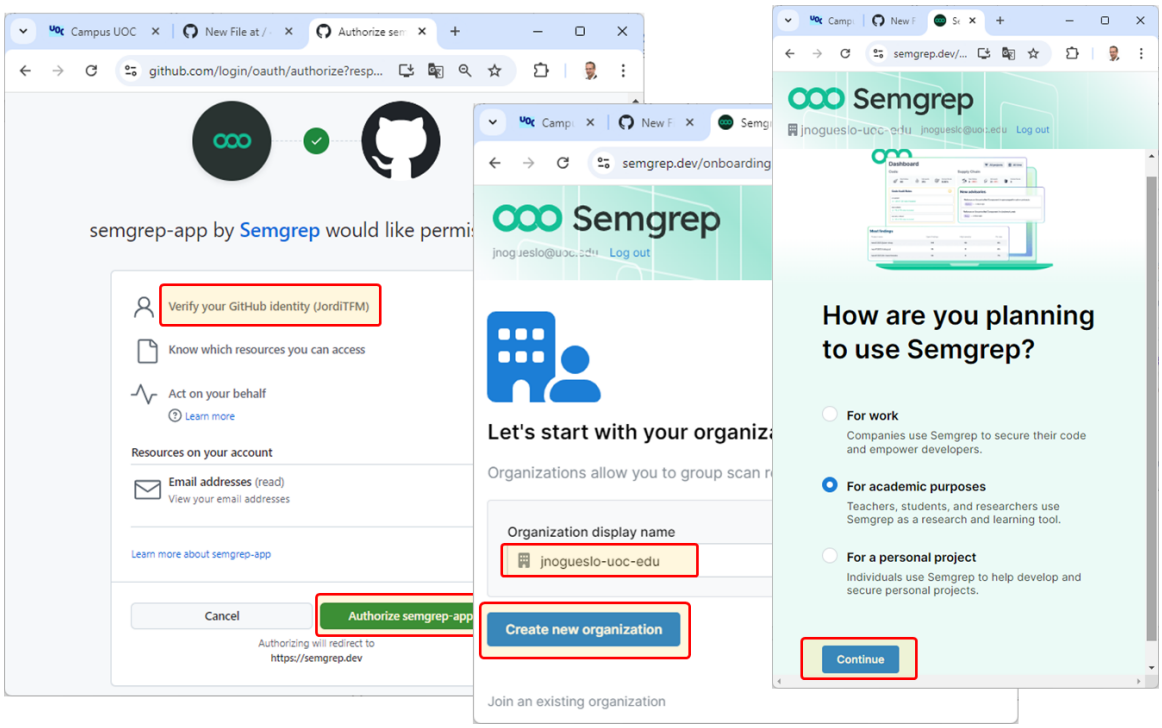


Ilustración 39. Consentimiento de acceso desde Semgrep. Fuente: Elaboración propia

Una vez completada la configuración, el asistente de Semgrep guiará el proceso hasta la activación del proceso de escaneo, según se muestra en la imagen siguiente:

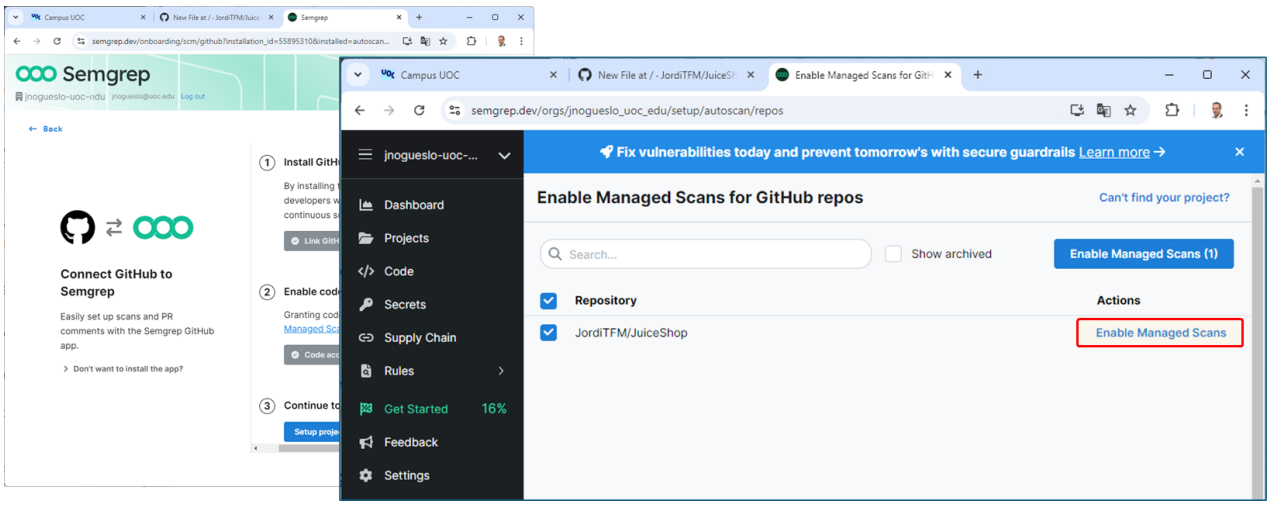


Ilustración 40. Activación del escaneo gestionado desde Semgrep. Fuente: Elaboración propia

Transcurridos unos minutos, los resultados del escaneo aparecerán en la consola de Semgrep, como se observa en la siguiente imagen:

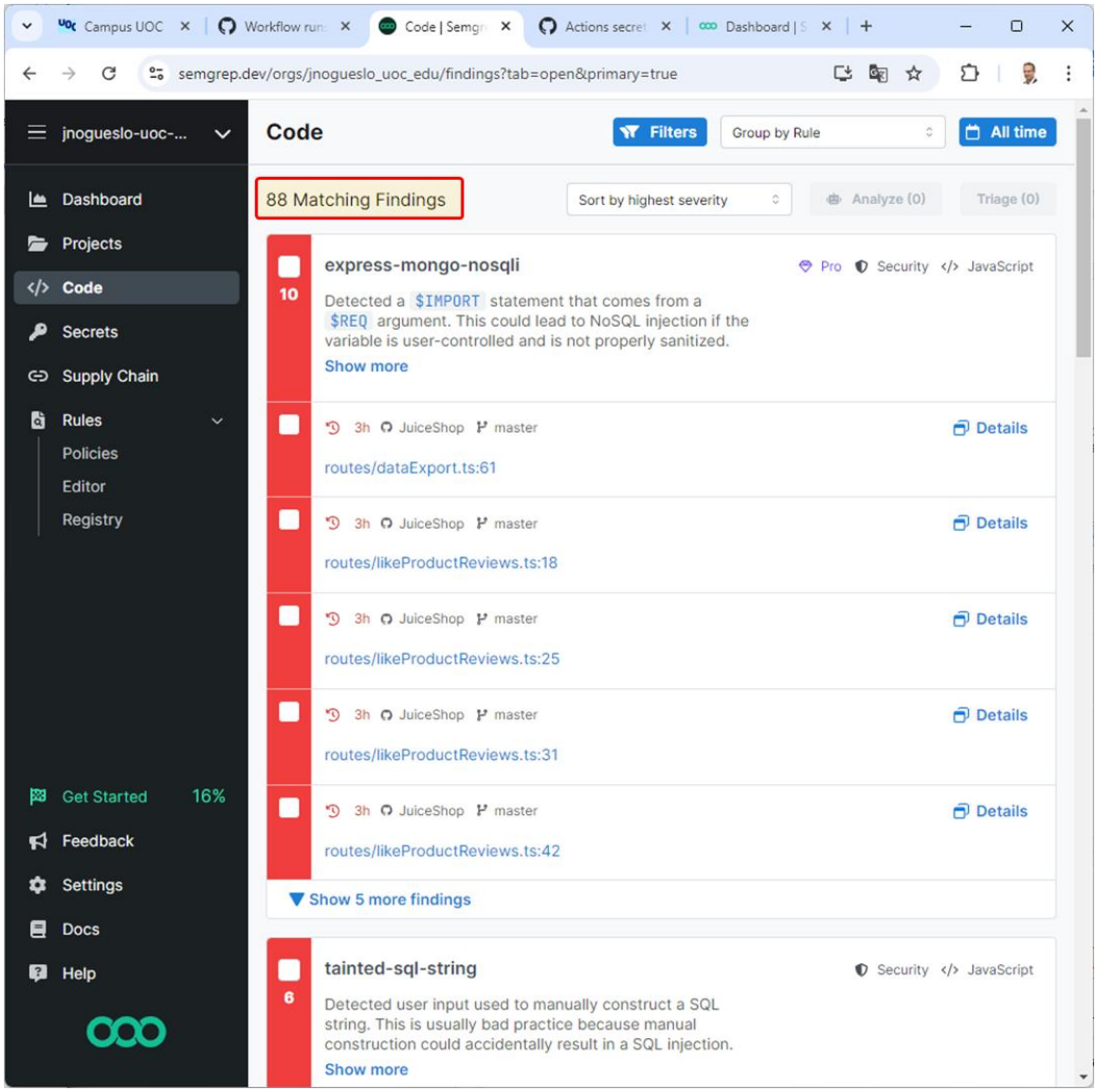


Ilustración 41. Listado de alertas en la consola de Semgrep. Fuente: Elaboración propia

En la consola se observa que el escaneo de Semgrep ha identificado 88 alertas de seguridad en el código del repositorio GitHub.

Aunque coincide en número con el de vulnerabilidades detectadas por CodeQL, no se ha verificado si las alertas corresponden exactamente a las mismas vulnerabilidades en ambos análisis.

7.17. Integración de la plantilla YAML predefinida

Este anexo describe el proceso de integración de una plantilla YAML predefinida para Semgrep en GitHub.

Esta plantilla, disponible desde la sección "**Security > Code scanning > Add Tool**", facilita la configuración automatizada de escaneos de seguridad mediante GitHub Actions. Una vez seleccionada, la plantilla puede añadirse directamente a la carpeta de workflows del repositorio, lo que permite gestionar y ejecutar los escaneos de manera integrada.

La imagen siguiente ilustra este procedimiento, desde la configuración inicial hasta la incorporación de la plantilla en el repositorio:

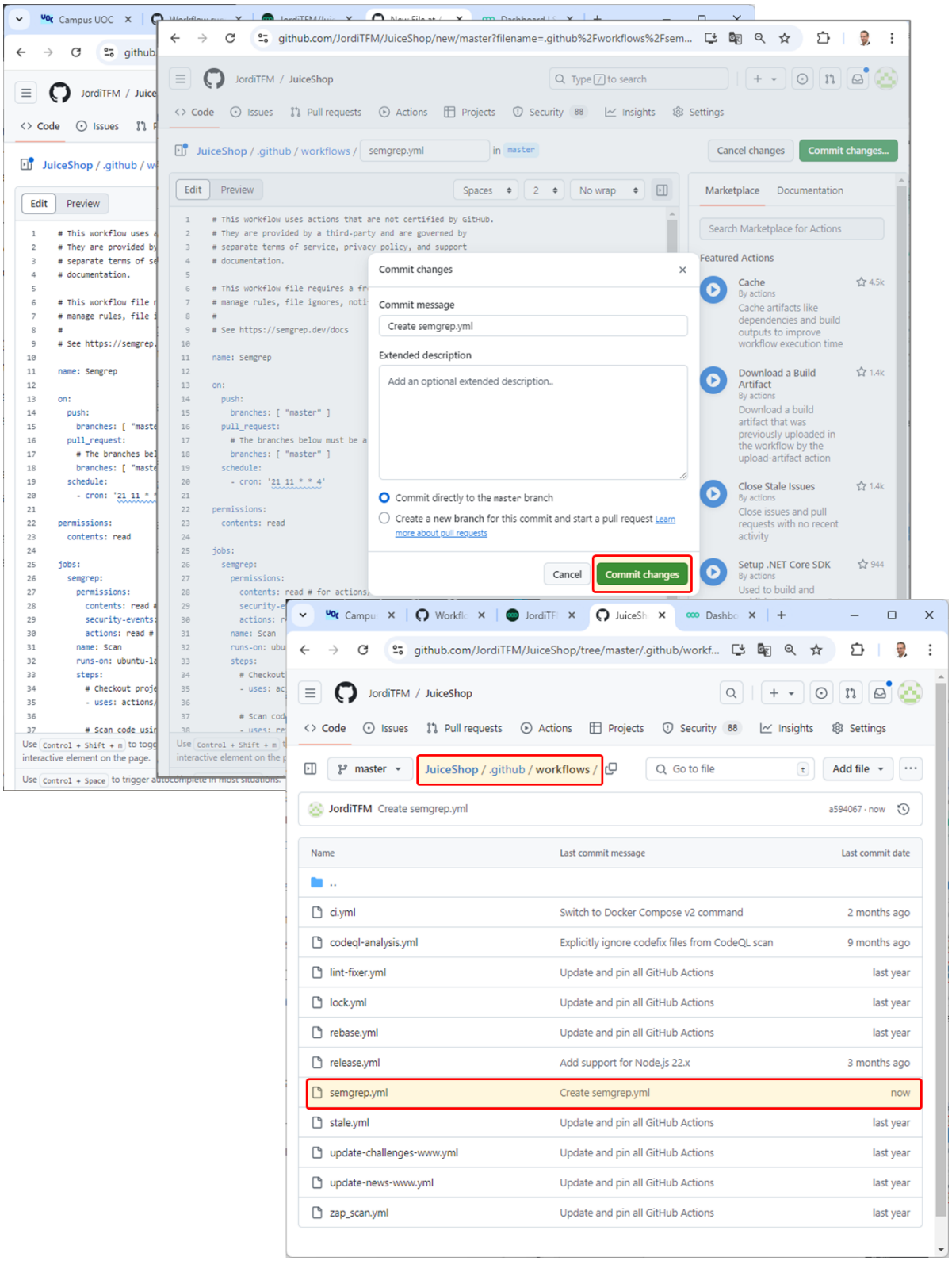


Ilustración 42. Integración de una plantilla YAML predefinida. Fuente: Elaboración propia

7.18. Creación del token de acceso para la conexión con Semgrep

Este anexo detalla el proceso de creación y configuración de un token de acceso para habilitar la conexión entre GitHub y Semgrep, necesaria para la ejecución automática de los escaneos de seguridad mediante workflows de GitHub Actions.

Este token se genera desde la pestaña “Tokens” del menú “Settings” en la aplicación Semgrep, como se ilustra en la siguiente imagen:

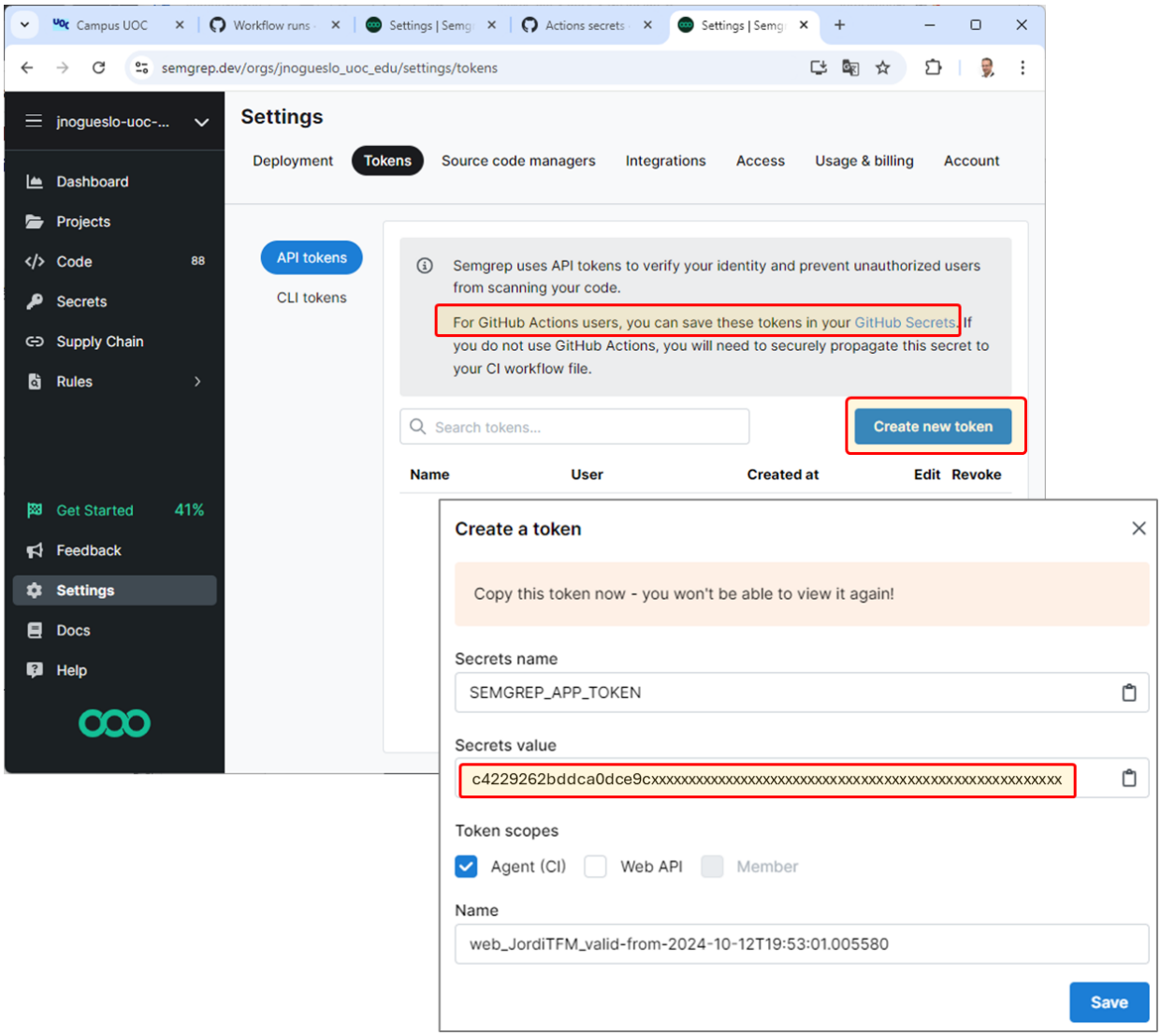


Ilustración 43. Generación del token de acceso en Semgrep. Fuente: Elaboración propia

Una vez generado el token, debe salvaguardarse de forma segura -puesto que Semgrep no lo volverá a mostrar-, y debe ser configurado como un secreto en el entorno GitHub.

La configuración del secreto en GitHub se realiza en la sección “Security” del menú de configuración del repositorio, mediante la opción “**Secrets and variables/Actions**”.

La imagen siguiente muestra cómo se da de alta el token “SEMGREP_APP_TOKEN” con el valor proporcionado por la aplicación Semgrep:

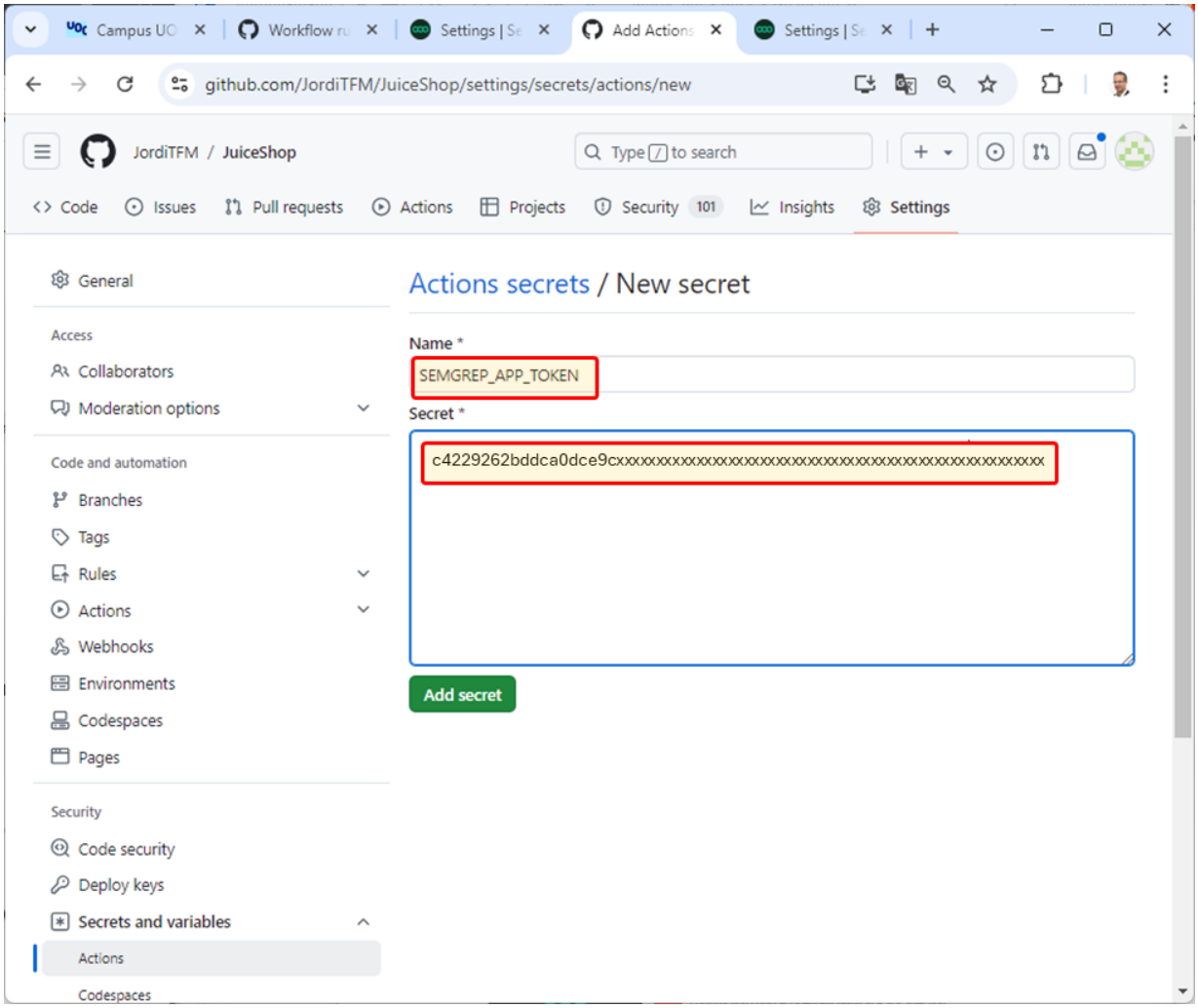


Ilustración 44. Configuración del secreto en GitHub. Fuente: Elaboración propia

Posteriormente, este token se utilizará para configurar el acceso en la plantilla YAML de GitHub Actions.

7.19. Configuración del archivo de workflow (.yml)

Este anexo detalla la configuración del archivo de workflow **semgrep.yml**, diseñado para automatizar el escaneo de seguridad con Semgrep mediante GitHub Actions.

La imagen a continuación muestra el contenido completo del archivo, incluyendo los parámetros esenciales como los eventos de activación, permisos, y pasos del job principal. Entre estos pasos, se muestra destacado mediante un recuadro amarillo el uso del token SEMGREP_APP_TOKEN, configurado previamente como secreto en GitHub, para autenticar las ejecuciones automatizadas:

Ilustración 45. Configuración del archivo de workflow .yml. Fuente: Elaboración propia

A continuación, se detalla el significado de cada parámetro:

Eventos de Activación:

- **push**: El workflow se ejecutará automáticamente cada vez que se haga un push a la rama master
- **pull_request**: También se activaría en cada solicitud de pull hacia la rama master
- **schedule**: La ejecución programada se define mediante una expresión **cron** (21 11 * * 4), configurada para ejecutarse los jueves a las 11:21 UTC

Permisos:

- **contents: read**: Permiso necesario para que la acción pueda acceder al contenido del repositorio
- **security-events: write**: Permiso para escribir en el registro de eventos de seguridad, necesario para subir los resultados de los análisis de seguridad como eventos de GitHub Security

Configuración del Job Principal (semgrep):

- **permissions**: Se establecen permisos específicos, similares a los definidos a nivel global.
- **runs-on: ubuntu-latest** Indica que el escaneo se ejecutará en una máquina virtual (temporal) con la distribución de Ubuntu más reciente

Pasos del Job:

- **actions/checkout@v4**: Clona el código del repositorio en la máquina virtual
- **run: pip install semgrep**: Instala Semgrep usando el gestor de paquetes de Python
- **semgrep scan** --sarif --output=semgrep.sarif --config=auto: Ejecuta Semgrep con las siguientes opciones
  - **--sarif**: Genera los resultados en formato SARIF
  - --**output=semgrep.sarif**: Define el nombre del archivo de salida
  - --**config=auto**: El escaneo utiliza la configuración automática, es decir, las reglas definidas en Semgrep
  - **env**: define la variable de entorno **SEMGREP_APP_TOKEN** para autenticar la ejecución de Semgrep, y le asigna el valor almacenado en el secreto que se ha configurado en GitHub Actions.
- **uses: github/codeql-action/upload-sarif@v3**: Utiliza la acción de GitHub para subir el archivo SARIF a GitHub Security.
- **with: sarif_file: semgrep.sarif**: indica que el archivo semgrep.sarif se subirá para que las vulnerabilidades detectadas aparezcan en la pestaña "Security" del repositorio
- **if: always()**: asegura que este paso se ejecutará independientemente de si los pasos anteriores fallan o no, garantizando la subida del archivo SARIF
  - **uses: actions/upload-artifact@v4**: Utiliza la acción de GitHub para subir el archivo SARIF como un artefacto.
  - **with: name: semgrep-sarif**: Nombre con el que se mostrará en la sección de artefactos de GitHub
    path: semgrep.sarif: Nombre del archivo físico