Universitat Oberta de Catalunya

# Trabajo Final de Máster: Automatización de la seguridad en el desarrollo del Software

Máster en Ciberseguridad y Privacidad

Área de Análisis de Datos

Jordi Nogués López

Tutor: Joan Caparrós Ramírez

Profesor: Pau Perea Paños

Enero de 2025

# Contexto

En la era digital actual, la seguridad del software es una prioridad crítica para las organizaciones de cualquier tamaño, desde diferentes puntos de vista:

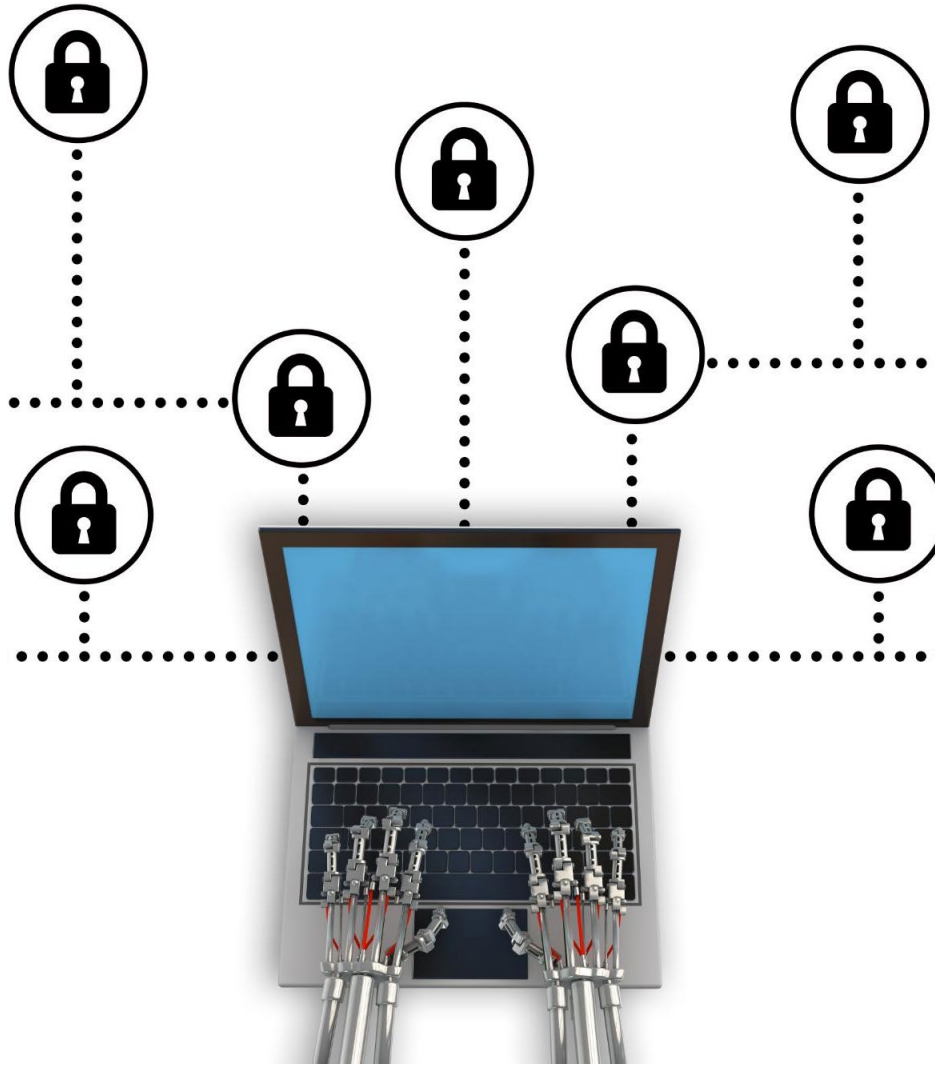
- **Perspectiva empresarial**: Las brechas de seguridad pueden provocar pérdidas financieras significativas, afectar la **continuidad del negocio** y dañar la **reputación** de la organización.
- **Perspectiva legal y regulatoria**: Las organizaciones deben cumplir con leyes y regulaciones que protegen 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.

# Desafíos

Para muchas organizaciones resulta complejo integrar prácticas de verificación del software de manera efectiva.

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.



# Resumen ejecutivo

**OBJETIVOS DEL PROYECTO**

- Ofrecer un análisis del **panorama actual de herramientas** de Análisis de Seguridad de Aplicaciones
- Demostrar cómo se pueden integrar esas herramientas en un **flujo DevSecOps automatizado**

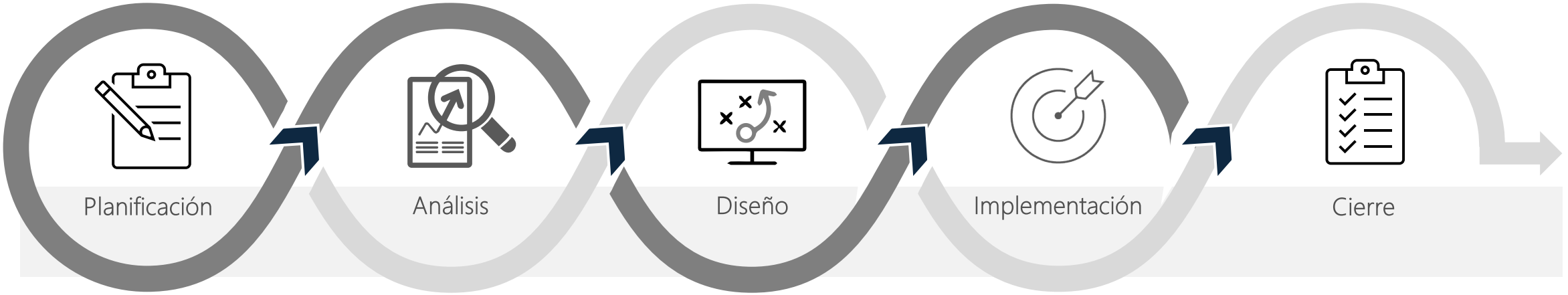
**ETAPAS CLAVE**

Las etapas clave del proyecto incluyen la **investigación** y **selección de herramientas**, la **implementación del entorno** de laboratorio y la **configuración del pipeline** para lograr una integración eficiente de las herramientas de análisis estático de seguridad **SAST** y **SCA**.

**LECCIONES APRENDIDAS**

Entre las lecciones aprendidas, destacan la relevancia de la **automatización**, impulsada aún más por la **Inteligencia Artificial**, la importancia de **adaptar las herramientas** al flujo de trabajo del desarrollador y la necesidad de **priorizar las vulnerabilidades según el riesgo**.

# Planificación y Metodología




| 3 semanas | 3 semanas | 3 semanas | 3 semanas | 3 semanas |
|---|---|---|---|---|
| Definición de objetivos, enfoque, alcance y entregables.<br>Elaboración del calendario de trabajo. | Introducción a los fundamentos de seguridad del SW.<br>Estado del arte de las herramientas, según OWASP y los analistas de mercado. | Evaluación de herramientas para el ámbito profesional.<br>Selección de herramientas para el proyecto, y diseño del entorno de laboratorio. | Instalación del entorno de laboratorio.<br>Configuración del pipeline.<br>Pruebas SAST.<br>Pruebas SCA. | Elaboración de las conclusiones.<br>Documentación de la memoria.<br>Realización del vídeo de presentación. |

# Fundamentos de seguridad en el desarrollo del software

- La **seguridad por diseño** favorece que las aplicaciones sean concebidas desde su origen con medidas de protección integradas.
- **Modelado de amenazas**
  - ¿Qué es lo que se está construyendo?
  - ¿Qué es lo que puede ir mal una vez esté construido?
  - ¿Qué debe hacerse al respecto de las cosas que pueden ir mal?
- **Buenas prácticas**
  - ✓ Autenticación y autorización con protocolos seguros
  - ✓ Cifrado en reposo y en tránsito
  - ✓ Validación de entradas para evitar inyecciones de código
  - ✓ Control del manejo de errores
  - ✓ Uso de librerías confiables


6

# Investigación - OWASP y Gartner

- La ***OWASP DevSecOps Guideline*** ofrece una visión completa de las herramientas disponibles para la seguridad del software, tanto comerciales como de código abierto.
- La guía muestra en qué fase del ciclo de vida del software se aplican las herramientas, cómo se integran y qué problemas solucionan.
- Los informes de **Gartner** permiten identificar a los fabricantes mejor valorados en el mercado y entender los criterios objetivos que se emplean para evaluar y posicionar cada herramienta.
- Esto aporta una perspectiva sólida para la selección de las soluciones más adecuadas en cada escenario.

# Herramientas seleccionadas

## Entorno Empresarial

SAST: Checkmarx SAST

SCA: Synopsys Black Duck

IAST: Synopsys Seeker IAST

DAST: HCL AppScan

## Laboratorio del Proyecto

SAST: CodeQL y Semgrep

SCA: Dependabot

*SAST – Static Application Security Testing*

- *Analiza el código fuente para detectar vulnerabilidades antes de la compilación.*

*SCA – Software Composition Analysis*

- *Identifica vulnerabilidades y problemas de licenciamiento en los componentes de terceros utilizados en la aplicación.*

*DAST – Dynamic Application Security Testing*

- *Analiza la aplicación en tiempo de ejecución, sin acceso al código fuente.*

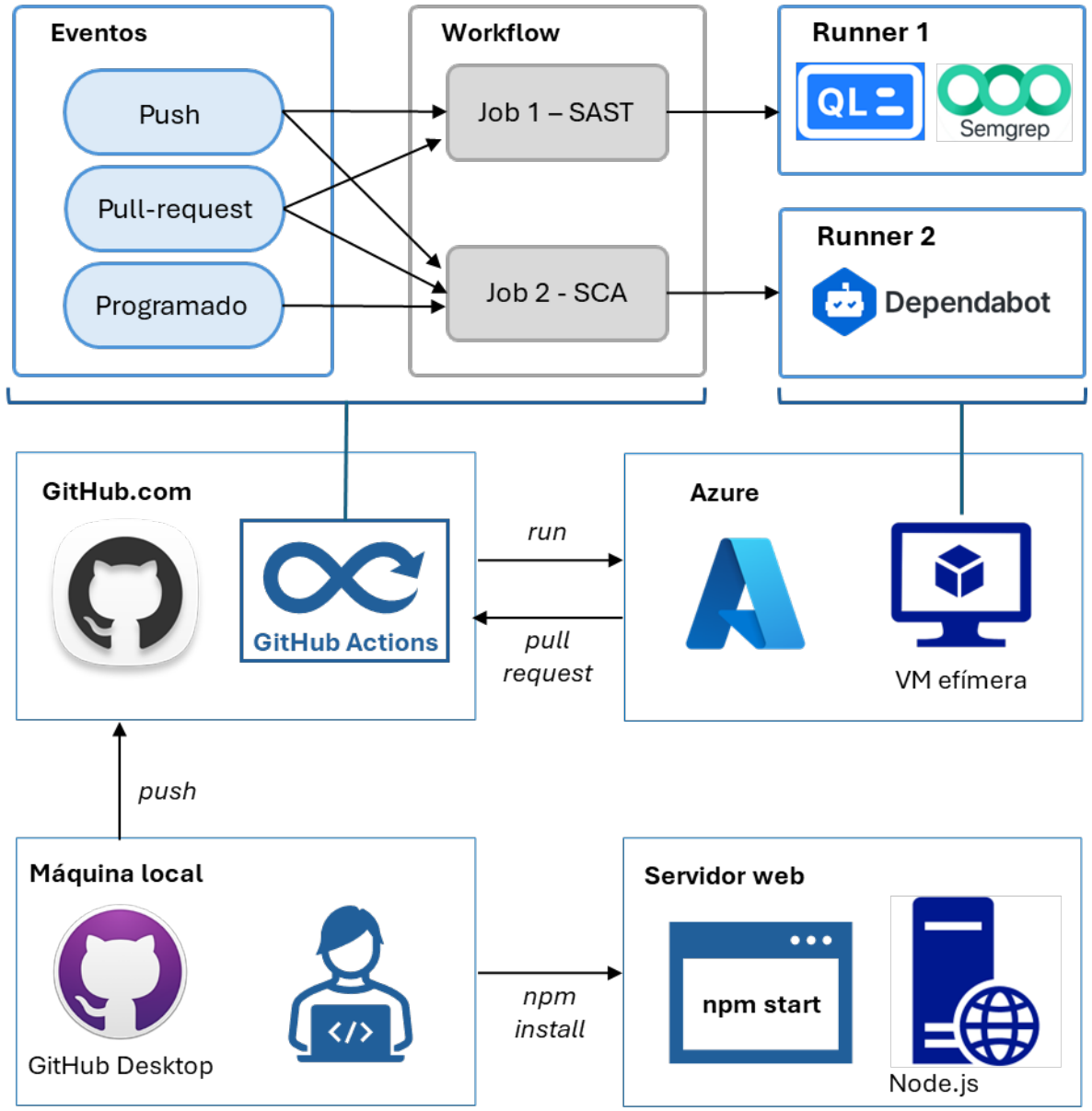
*IAST – Interactive Application Security Testing*

- *Combina elementos de SAST y DAST proporcionando una visión más completa.*

# Diseño del laboratorio

El entorno de laboratorio consta de:

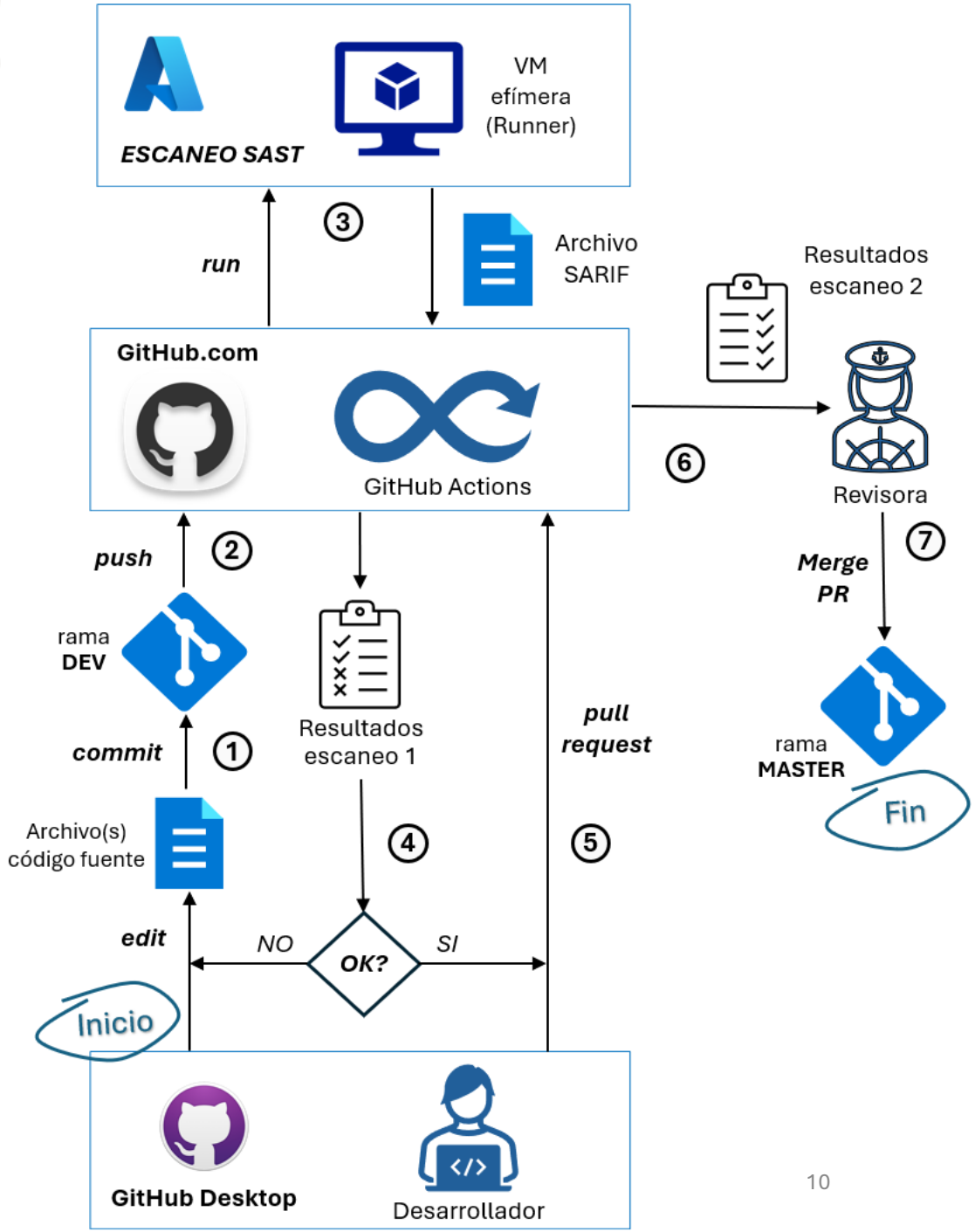
- Repositorio de código: **GitHub** (gestionado con **GitHub Desktop**)
- Plataforma *DevSecOps*: **GitHub Actions**
- Herramientas SAST: **CodeQL** y **Semgrep**
- Herramienta SCA: **Dependabot**
- Infraestructura de ejecución de las pruebas: **Azure IaaS** (proporcionada de forma transparente mediante GitHub Actions)
- Aplicación objetivo: **OWASP Juice Shop**
- Infraestructura de ejecución de la aplicación: **Node.js** (servidor web integrado)



# Diseño del flujo SAST

Este flujo debe validar la seguridad de la aplicación de forma continua, a medida que el desarrollador progresa en la construcción del **código fuente**:

1. El desarrollador realiza cambios en la rama de desarrollo de su repositorio local, y los incorpora al historial de versiones de su entorno local mediante "*commit*".
2. Posteriormente, integra los cambios en el repositorio central mediante el mandato "*push*".
3. La plataforma *DevSecOps* (GitHub Actions) dispara el proceso de escaneo del código, clonándolo previamente en una Máquina Virtual temporal (*runner*) en Azure.
4. El resultado del escaneo se publica en GitHub y se le notifica al desarrollador para que lo verifique y depure los posibles errores detectados.
5. Mediante "*pull-request*", el desarrollador solicita la integración de los cambios en la rama principal del proyecto.
6. GitHub Actions dispara nuevamente el escaneo para verificar si está todo correcto, y genera el informe correspondiente.
7. Tras completar las verificaciones, la jefatura del proyecto fusiona los cambios del "*pull-request*" en la rama principal del repositorio.



# Diseño del flujo SCA

El análisis SCA se centra en verificar la seguridad de las **librerías de terceros y open-source** utilizadas en el proyecto.

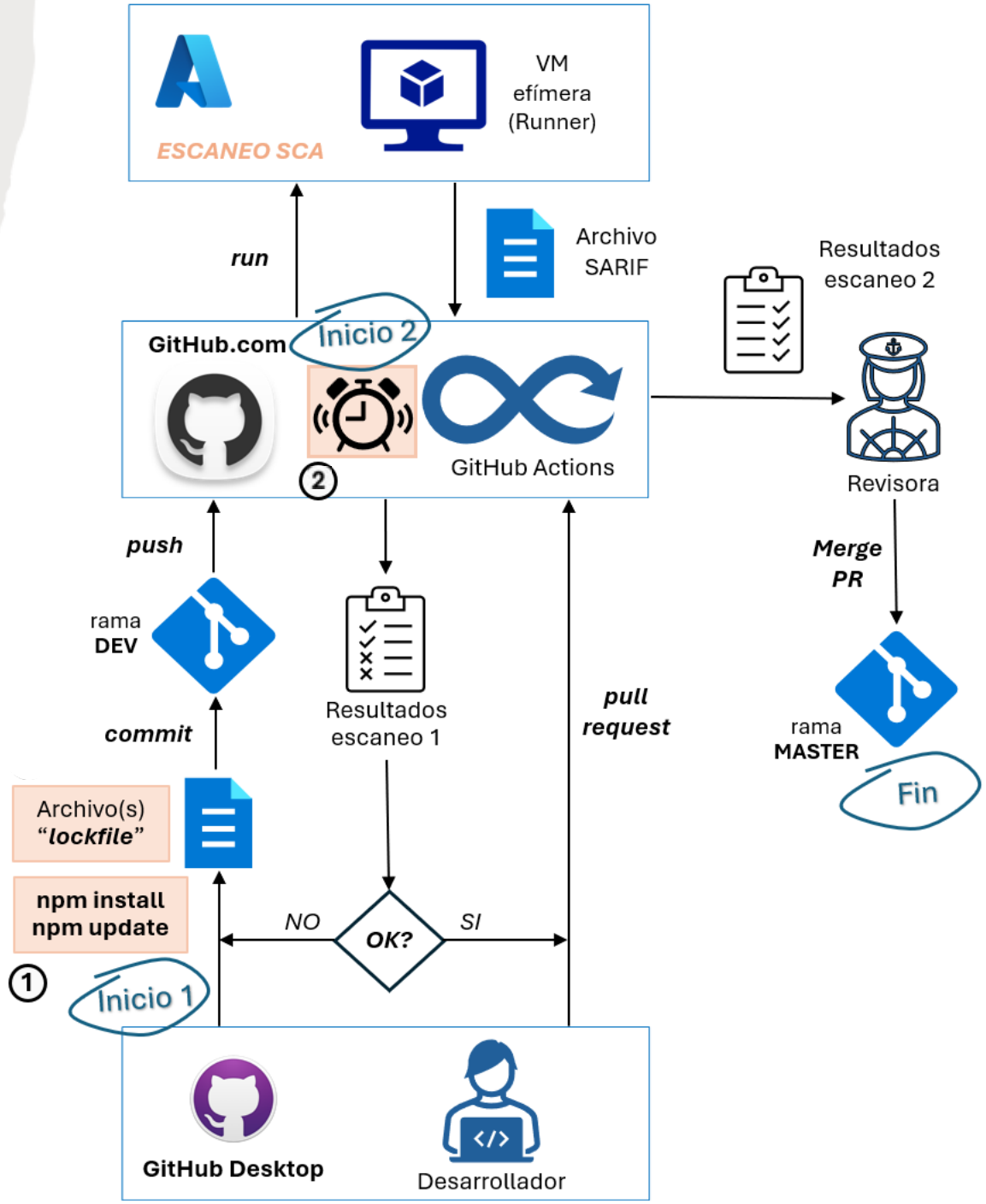
Por consiguiente, no es necesario ejecutar el escaneo ante cualquier cambio en el código, sino solamente cuando se instala o actualiza alguna dependencia.

Además, también hay que verificar regularmente que las librerías y sus licencias de uso siguen estando vigentes.

Por consiguiente, el flujo debe activarse en respuesta a 2 eventos distintos:

1. Cuando el desarrollador **añada o actualice** la versión de **una librería** ya instalada en el proyecto.

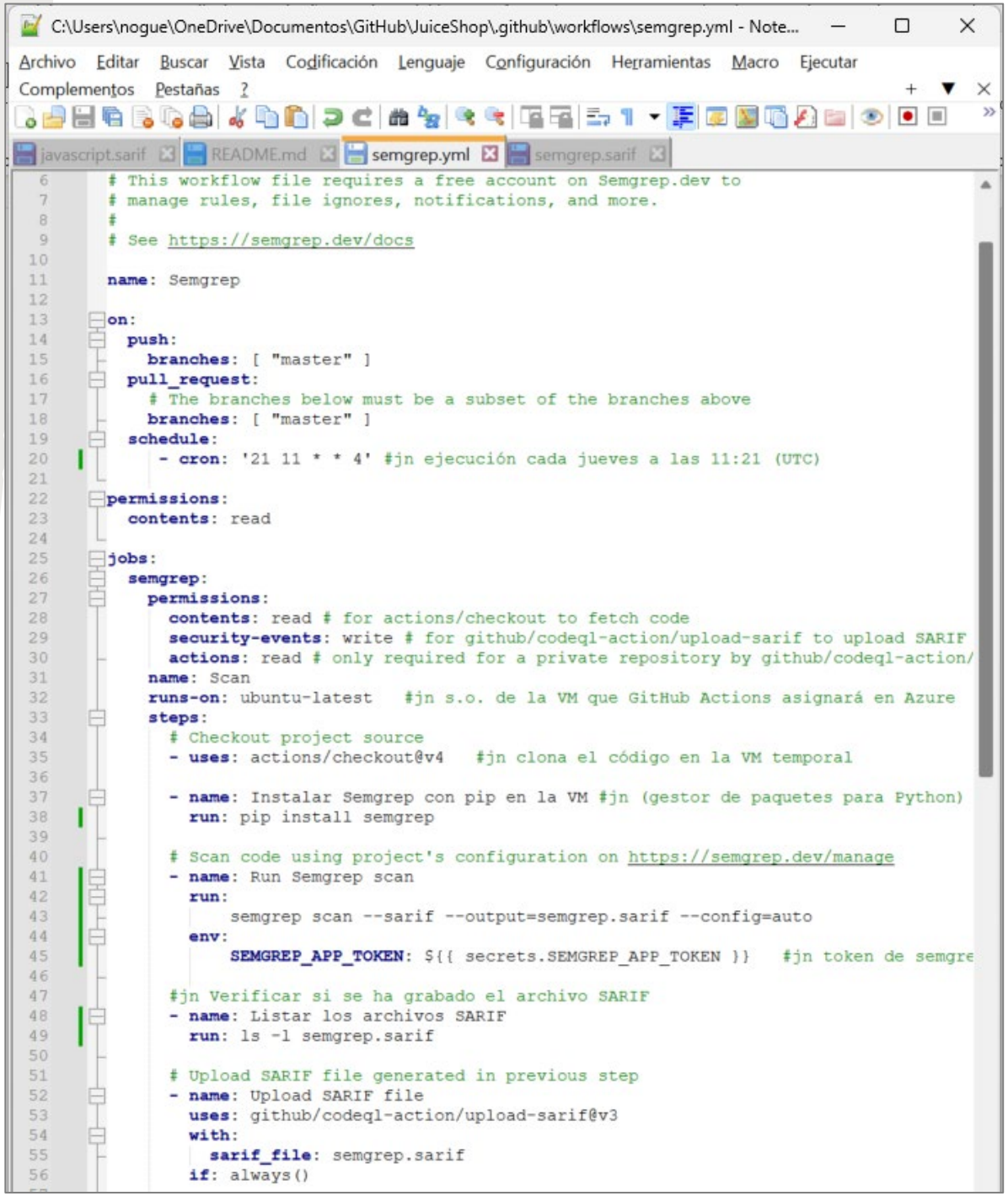
2. De forma automática **a una hora programada**, independientemente de si ha habido cambios en las librerías.



# Configuración del flujo

**Parámetros principales**

- Eventos de activación: *push*, *pull-request*, *schedule*
- Permisos: De lectura para acceder al repositorio de código, y de escritura sobre el registro de eventos de seguridad
- Sistema operativo del *runner*: *Ubuntu-latest*
- Clonado del código del repositorio en la VM de Azure:
    - actions/checkout@v4
- Instalación y ejecución de la herramienta SAST en la VM:
    - run: pip install semgrep
    - run: semgrep scan --sarif –output=…
- Asignación del token de autenticación mediante variable de entorno:
    - Env: SEMGREP_APP_TOKEN: ${{ secrets.SEMGREP_APP_… }}
- Carga en GitHub del archivo SARIF con las alertas detectadas:
    - github/codeql-action/upload-sarif@v3

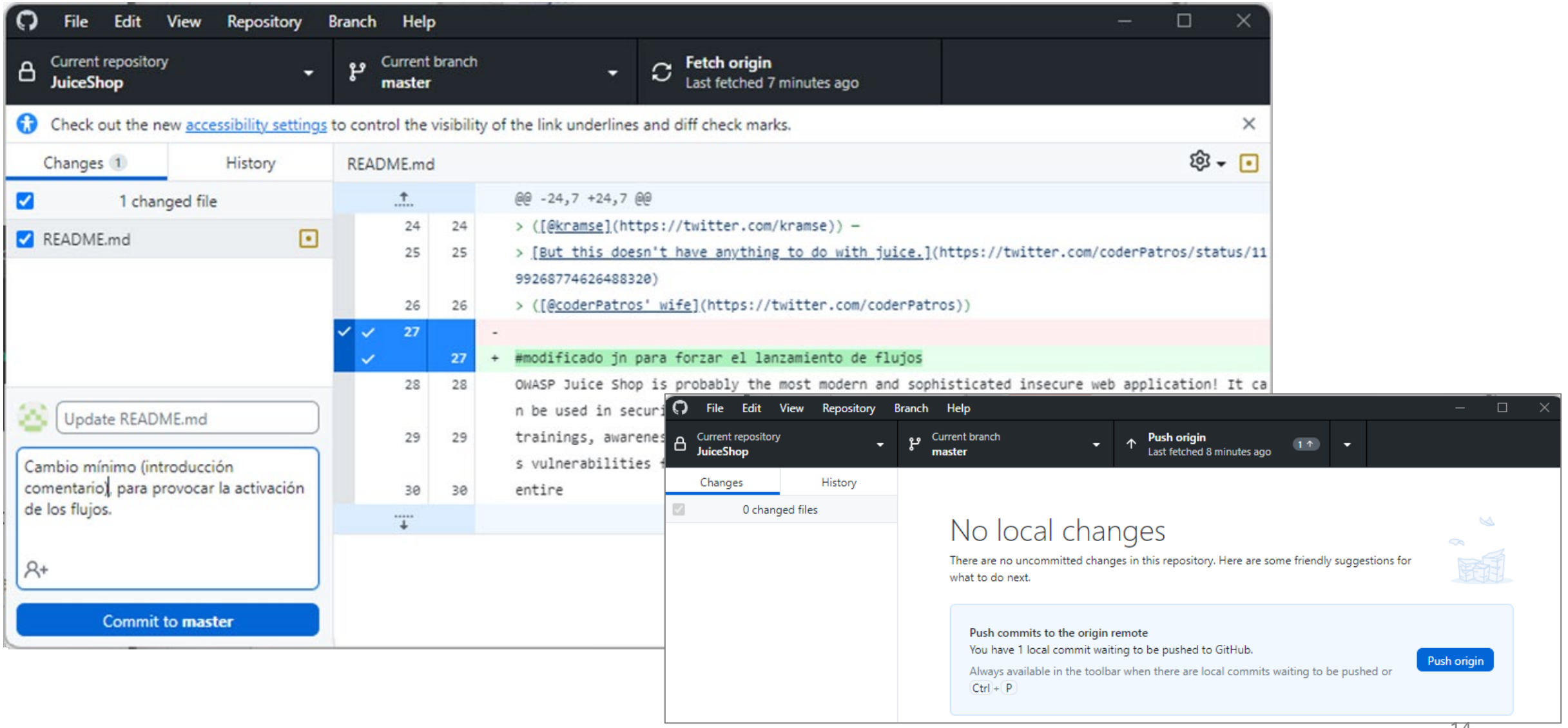


# Implementación

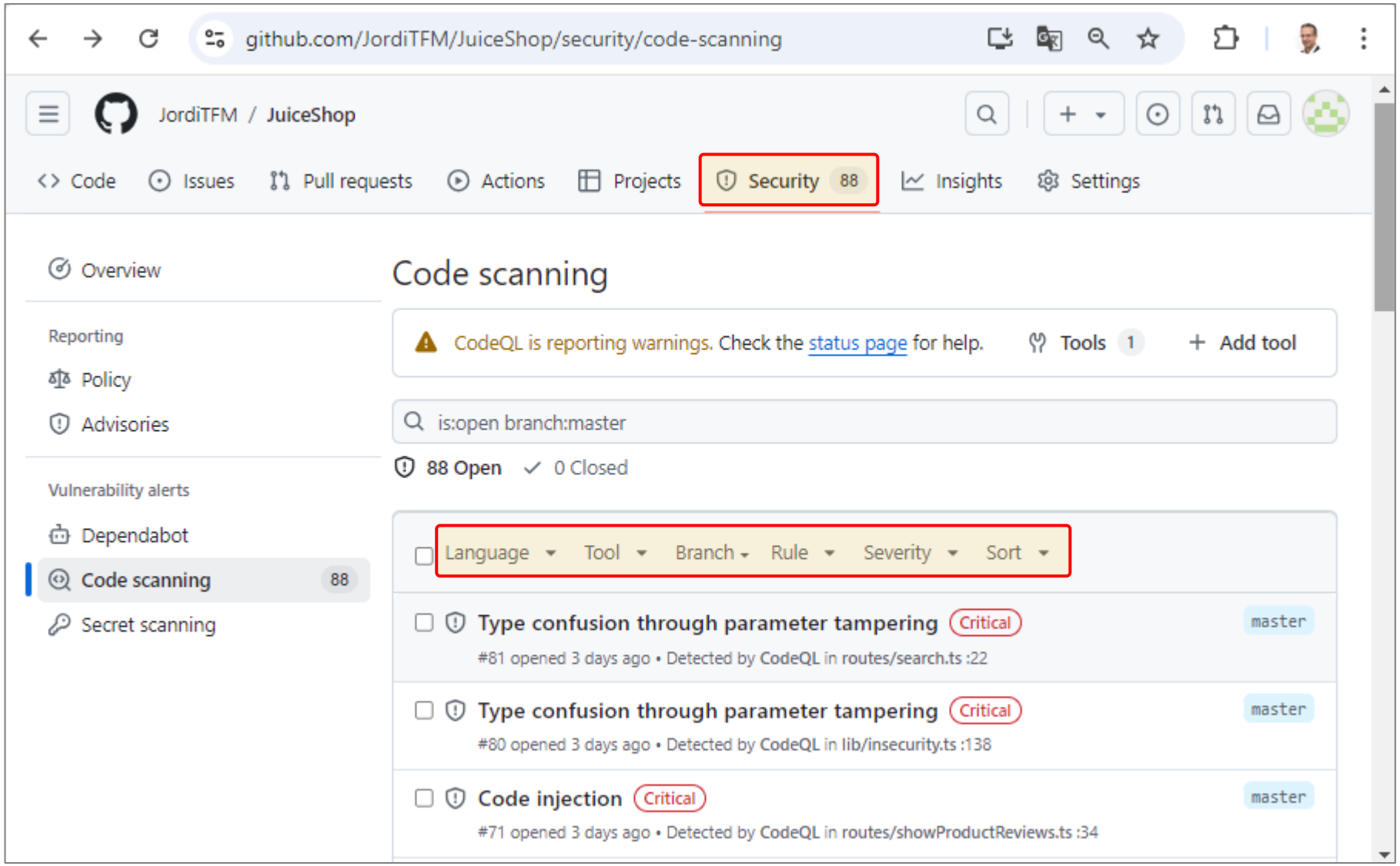
A continuación, se demuestra la implementación de 2 casos de uso en el entorno de laboratorio:

1. SAST – Detección de vulnerabilidades en el código con **CodeQL** y remediación con **Copilot Autofix**
2. SCA – Detección y resolución automática de librerías no seguras u obsoletas con **Dependabot**.

# 1. SAST – Publicación de los cambios en el código



# 1. SAST – Consulta de alertas



# 1. SAST – Gestión de la incidencia

# 1. SAST – Remediación automática con COPILOT

## Type confusion through parameter tampering

Dismiss

Open in `master` 4 days ago

**Copilot Autofix** AI generated a fix less than a minute ago

To fix the problem, we need to ensure that the `req.query.q` parameter is a string before using it. This can be done by adding a type check for `req.query.q` and handling cases where it is not a string. If the parameter is not a string, we should treat it as an invalid input and handle it accordingly.

1. Add a type check for `req.query.q` to ensure it is a string.
2. If `req.query.q` is not a string, set `criteria` to an empty string or handle it as an invalid input.
3. This change should be made in the `routes/search.ts` file, specifically around lines 21-22.

`routes/search.ts:22` Autofix

```
...   @@ -20,3 +20,3 @@
20      return (req: Request, res: Response, next: NextFunction) => {
21        let criteria: any = req.query.q === 'undefined' ? '' : req.query.q ?? ''
22        criteria = (criteria.length <= 200) ? criteria : criteria.substring(0, 200)
```

```
20      return (req: Request, res: Response, next: NextFunction) => {
21        let criteria: any = (typeof req.query.q === 'string') ? req.query.q : ''
22        criteria = (criteria.length <= 200) ? criteria : criteria.substring(0, 200)
```

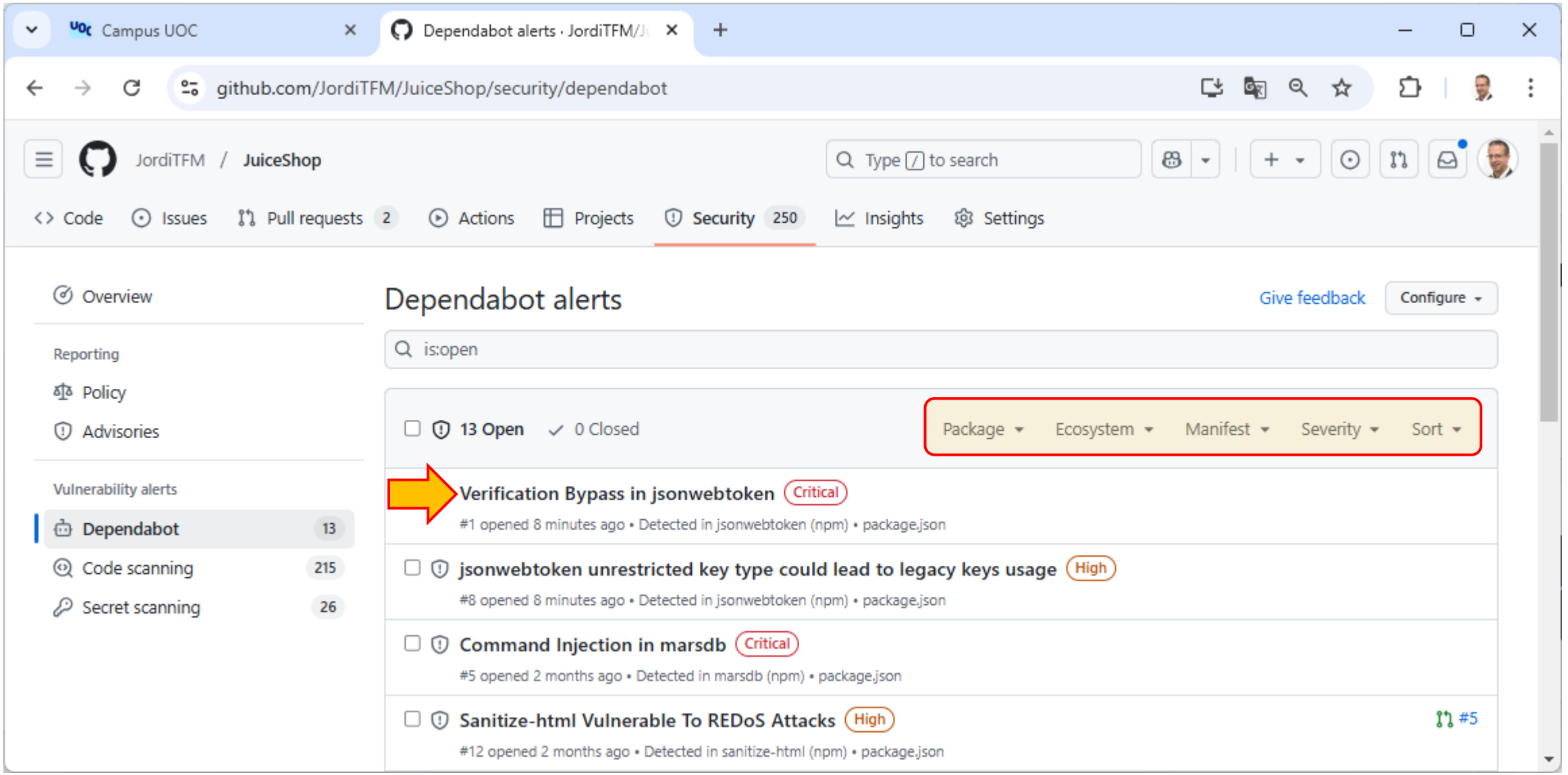
Severity: Critical

Affected branches: `master`

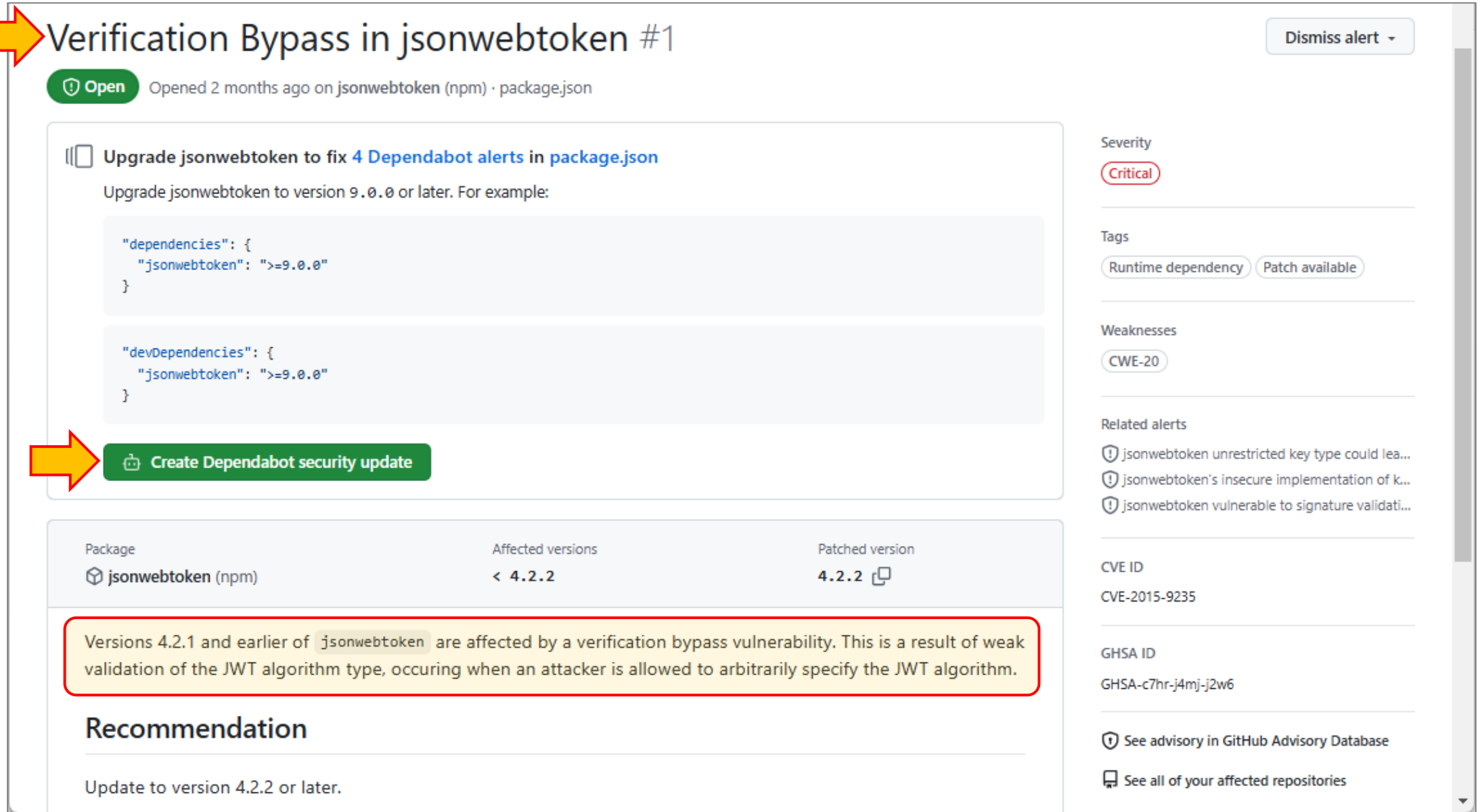
Tags: security

Weaknesses: CWE-843

# 2. SCA – Consulta de alertas



# 2. SCA – Gestión de la incidencia



## Verification Bypass in jsonwebtoken #1

Dismiss alert

Open · Opened 2 months ago on **jsonwebtoken** (npm) · package.json

**Upgrade jsonwebtoken to fix 4 Dependabot alerts in package.json**

Upgrade jsonwebtoken to version `9.0.0` or later. For example:

```
"dependencies": {
  "jsonwebtoken": ">=9.0.0"
}
```

```
"devDependencies": {
  "jsonwebtoken": ">=9.0.0"
}
```

Create Dependabot security update

| Package | Affected versions | Patched version |
|---|---|---|
| jsonwebtoken (npm) | < 4.2.2 | 4.2.2 |

Versions 4.2.1 and earlier of `jsonwebtoken` are affected by a verification bypass vulnerability. This is a result of weak validation of the JWT algorithm type, occuring when an attacker is allowed to arbitrarily specify the JWT algorithm.

### Recommendation

Update to version 4.2.2 or later.

Severity: Critical

Tags: Runtime dependency, Patch available

Weaknesses: CWE-20

Related alerts:
- jsonwebtoken unrestricted key type could lea...
- jsonwebtoken's insecure implementation of k...
- jsonwebtoken vulnerable to signature validati...

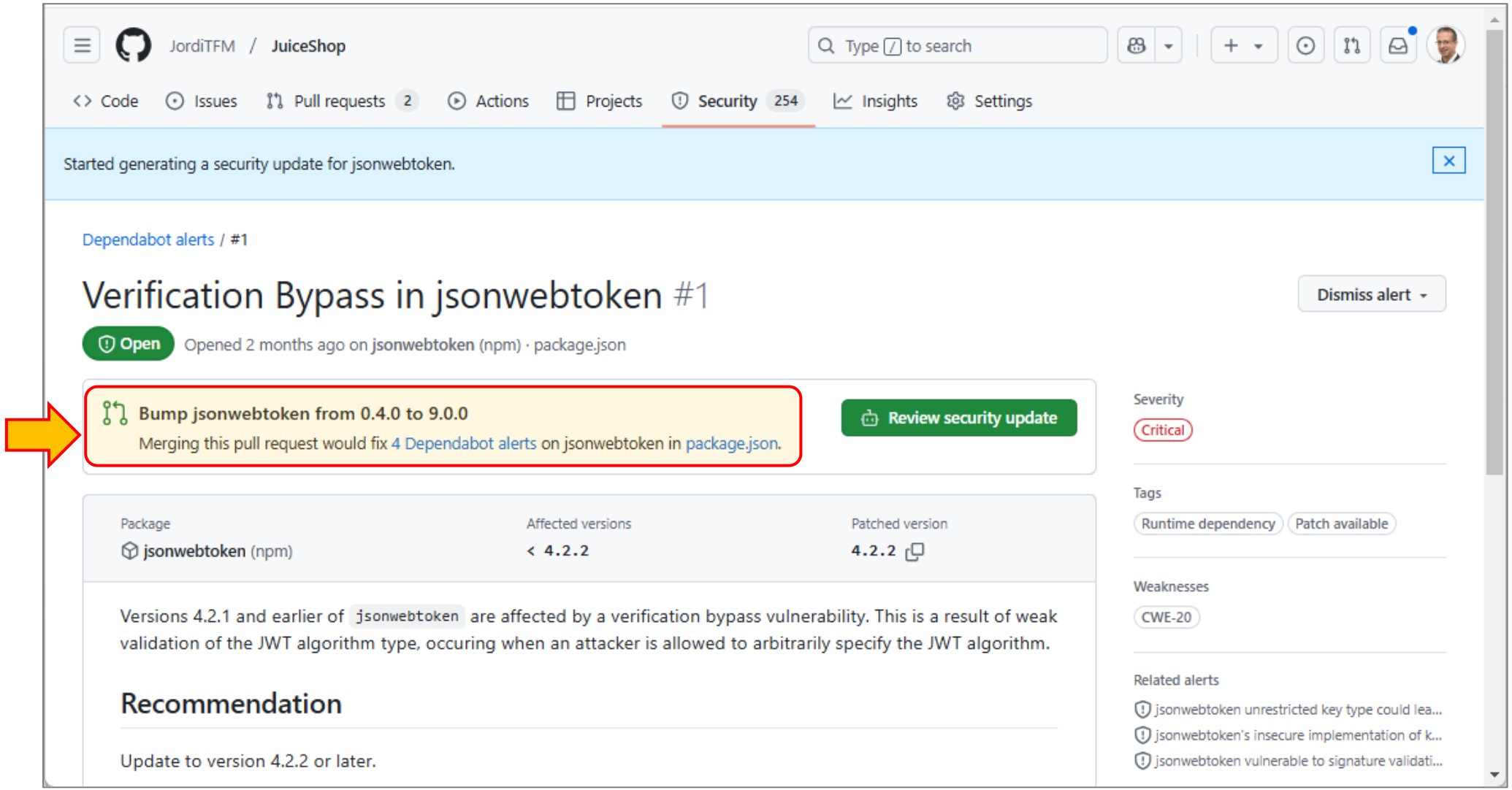
CVE ID: CVE-2015-9235

GHSA ID: GHSA-c7hr-j4mj-j2w6

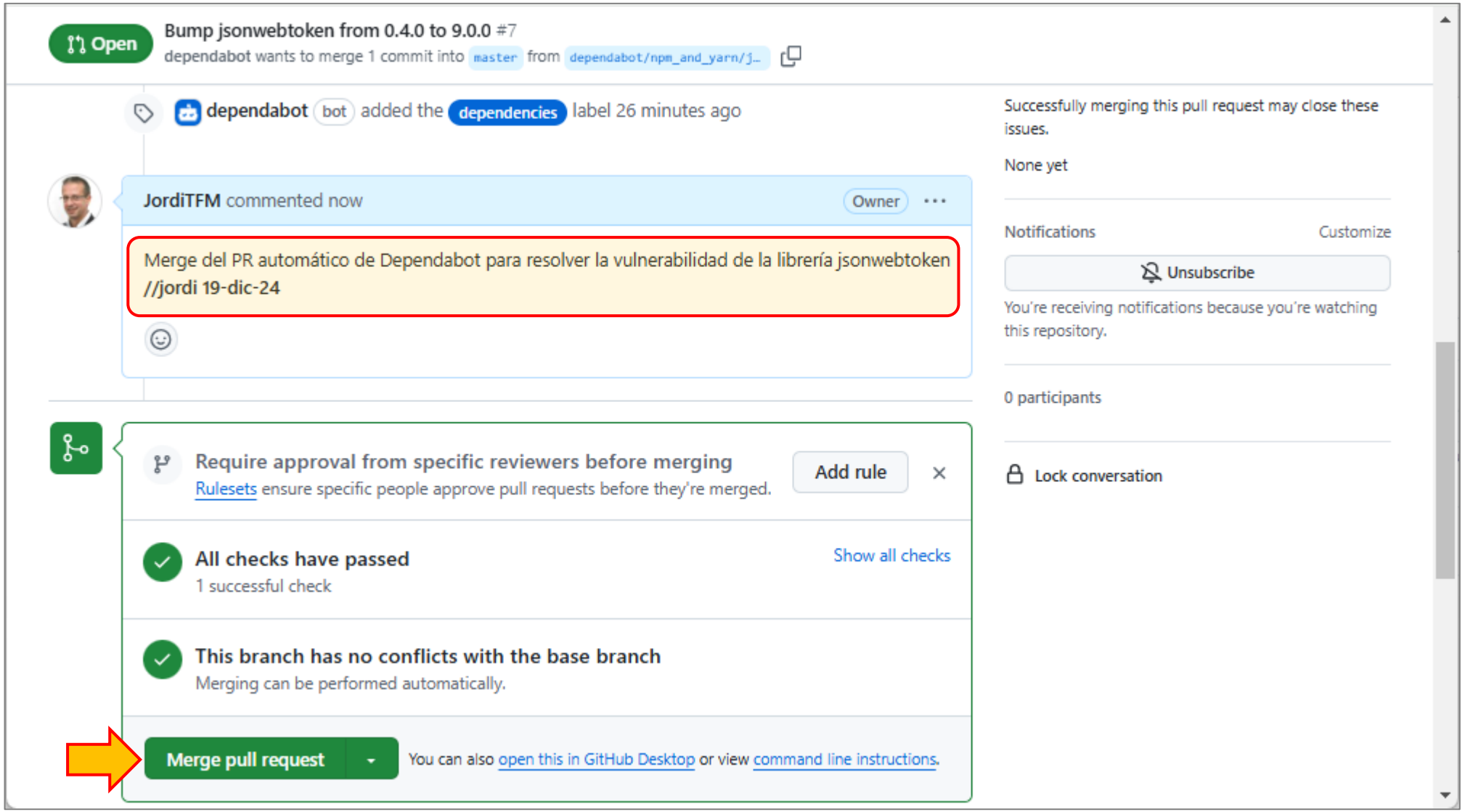
See advisory in GitHub Advisory Database

See all of your affected repositories

# 2. SCA – Actualización automática de la librería



# 2. SCA – Fusión del cambio



# CONCLUSIONES

**La automatización es posible**, y permite producir un código seguro sin penalizar el ritmo de desarrollo que exige el negocio digital.

**Impulsada por la Inteligencia Artificial**, acelera el proceso de remediación y democratiza el acceso a las buenas prácticas de ciberseguridad.

# Posibles trabajos futuros

**Integración de herramientas DAST e IAST**

El uso de herramientas DAST e IAST puede ofrecer las máximas garantías de eficacia en los controles de seguridad del software. El reto principal será transformar las alertas generadas a un formato compatible que permita integrarlas en el entorno *DevSecOps*.

**Benchmarking de herramientas**

La evolución de plataformas como el *OWASP Benchmark Project*, actualmente limitada a aplicaciones desarrolladas con el SDK de Java 11, abriría la puerta a una evaluación objetiva de la eficiencia de herramientas SAST sobre aplicaciones modernas.

Gracias!