

# Integración y configuración de análisis estático con SonarCloud, Semgrep y Snyk

Haessler Joan Ortiz Moncada

Universidad Distrital Francisco José de Caldas

Facultad de Ingeniería

Curso: Construcción de Pruebas de Software

1 de septiembre de 2025

## 1. Introducción

En este informe documento el proceso de integración y configuración de tres plataformas para realizar análisis estático de código en el repositorio de mi autoría: **confi\_seg**.

El objetivo principal es consolidar los resultados de seguridad y calidad del código, automatizando su ejecución mediante un único workflow en **GitHub Actions**.

### Nota:

El archivo unificado **security-ci.yml**, que permite ejecutar las tres pruebas de forma coordinada, fue generado con la ayuda de una **Inteligencia Artificial**, en este caso ChatGPT, para optimizar la integración y reducir errores en la configuración.

## 2. Metodología

Primero exploré cada plataforma para comprender sus características, los tipos de análisis que realizan y su integración con GitHub. Posteriormente, configuré un único workflow llamado **security-ci.yml** para ejecutar las tres pruebas automáticamente en cada *push* o *pull request*.

## 3. Resultados

### 3.1. Resultados de Semgrep

El análisis realizado con **Semgrep** detectó múltiples vulnerabilidades relacionadas con el manejo inseguro de datos, consultas SQL y configuración de CORS.

Categoría	Lenguaje	Severidad	Archivos afectados
SQL concatenado sin sanitización	Python	Baja	projects.py (líneas 140, 163, 179, 188, 196)
Uso de métodos inseguros en DOM	JavaScript	Baja	principal.js (línea 216)
Consultas SQL formateadas sin parámetros	Python	Baja	projects.py (mismas líneas que anteriores)
Etiqueta sin atributo de integridad	HTML	Baja	principal.html (línea 67)
Política CORS insegura	Python	Media	main.py (línea 14)

## Resumen general de hallazgos

### Principales vulnerabilidades detectadas

- **SQL concatenado sin sanitización** (projects.py): Varias funciones concatenan datos directamente en consultas SQL, lo que podría permitir ataques de **inyección SQL**.
- **Métodos inseguros en el DOM** (principal.js): Uso de innerHTML con datos controlados por el usuario, potencialmente expuesto a ataques **XSS**.
- **Política CORS insegura** (main.py): Configuración de CORS que permite solicitudes desde cualquier origen, lo que supone un riesgo de seguridad.

## 3.2. Resultados de Snyk

El análisis con **Snyk** me reportó hallazgos principalmente en el backend (Python), con foco en construcción insegura de SQL, manejo de rutas y uso de parámetros no validados en conexiones. A continuación dejo el resumen:

Categoría	Lenguaje	Severidad	Archivos afectados (ejemplos)
SQL Injection (CWE-89)	Python	Alta	backend/routers/projects.py (múltiples líneas: 136, 175, 184, 192, 304, 311, 328, 336, 362, 471, 480, 486, 502, 508, 512, 525, 554, 571, 577, 585, 599, 623, 676, 743, etc.)
Path Traversal (CWE-23)	Python	Alta	backend/routers/projects.py (725, 859); qgis_tools/generar_proyecto_qgis.py (35)
SSRF (CWE-918)	Python	Media	backend/routers/projects.py (155, 169, 218, 288)
Revocación/roles con SQL dinámico	Python	Media	backend/routers/projects.py (806, 813)

## Resumen general de hallazgos

## Principales vulnerabilidades detectadas

- **SQL Injection:** Encontré múltiples consultas construidas con `f-strings` que interpolan datos controlados por el usuario. Se deben generar consultas parametrizadas (`cur.execute(sql, params)`), validación estricta y, si es posible, ORM.
- **Path Traversal:** Se generan rutas a partir de entrada externa (lectura/escritura/borrado). Se deben generar listas blancas de rutas, normalización con `os.path.abspath`, validaciones de nombre y extensión, y `Pathlib`.
- **SSRF:** Parámetros externos terminan en funciones de conexión; debo validar host, puerto, nombre de DB y bloquear destinos internos.
- **Permisos y roles vía SQL dinámico:** Evitar interpolación directa al gestionar `GRANT/REVOKE/DROP ROLE`; usar plantillas parametrizadas y catálogos del motor.

## 3.3. Resultados de SonarCloud

Con **SonarCloud** obtuve una radiografía amplia de *code smells*, bugs y vulnerabilidades en varias áreas del repositorio. Este es el resumen de los hallazgos:

Categoría	Lenguaje	Severidad	Archivos afectados (ejemplos)
Construcción de SQL desde datos del usuario	Python	Bloqueante/Crítica	<code>backend/routers/projects.py</code> (140, 196, 308, 315, 516, 575, 581, 615, 626, 661, 680, 747, etc.)
Complejidad cognitiva excesiva	Python/JS	Alta	<code>backend/routers/projects.py</code> (28, 30, 20, 26, 78); <code>frontend/assets/js/principal.js</code> (funciones con complejidad elevada)
Código inalcanzable / comentado / variables sin uso	Python/JS/HTML	Media/Baja	<code>projects.py</code> (unreachable y vars sin uso); <code>principal.js</code> (vars sin uso); <code>principal.html</code> (detalles de integridad/escape)
Convenciones de nombres y estilo	Python	Baja	<code>qgis_tools/qgis_core.py</code> (parámetros/campos que no cumplen regex recomendada)

## Resumen general de hallazgos

### Principales vulnerabilidades y problemas

- **SQL inseguro:** Igual que en Snyk, SonarCloud refuerza que debo eliminar la interpolación directa en `execute(...)`, usar parámetros y/o ORM.
- **Complejidad cognitiva:** Varias funciones superan los umbrales; debo dividir responsabilidades, extraer helpers y reducir condiciones anidadas.
- **Mantenibilidad:** Código inalcanzable, comentado y variables sin uso; limpieza y linters automáticos (`ruff/flake8/eslint`) ayudarán.
- **Consistencia y convenciones:** Ajustar nombres a las guías de estilo (`snake_case` en Python, etc.) para mejorar legibilidad.

## 4. Análisis comparativo

Para comparar las tres plataformas, analicé sus principales características:

Criterio	SonarCloud	Semgrep	Snyk
Tipo de análisis	Calidad, bugs y <i>code smells</i>	SAST basado en reglas personalizadas	SCA (dependencias) + SAST
Profundidad	Alta (métricas y deuda técnica)	Media-Alta (reglas flexibles)	Alta en vulnerabilidades conocidas
Lenguajes soportados	+25	+30	+20
Integración con GitHub Actions	Automática, genera .yaml	Automática, genera .yaml	Manual, requiere configuración propia
Reportes	Visuales y detallados	Consolidados y exportables	Incluye SARIF y dashboard propio
Costo	Gratis con límites	Gratis con límites	Gratis con límites

En general:

- **SonarCloud** es ideal para evaluar calidad, métricas y deuda técnica.
- **Semgrep** es más flexible y permite crear reglas personalizadas adaptadas al proyecto.
- **Snyk** sobresale en la detección de vulnerabilidades en dependencias externas y propone soluciones rápidas.

## 5. Conclusiones

1. La combinación de **SonarCloud**, **Semgrep** y **Snyk** proporciona una cobertura completa del análisis estático de código, otorgando un enfoque diferente y complementario.
2. La automatización mediante un único workflow simplifica la gestión de resultados.