# UNIVERSIDAD ESTATAL DE BOLÍVAR

## FACULTAD DE CIENCIAS ADMINISTRATIVAS, GESTIÓN EMPRESARIAL E INFORMÁTICA

CARRERA DE SOFTWARE

**TRABAJO DE INTEGRACIÓN CURRICULAR**
**PREVIO A LA OBTENCIÓN DEL TÍTULO DE**
**INGENIERO EN SOFTWARE**

**FORMA:** PROYECTO DE INVESTIGACIÓN

**TEMA:**

HERRAMIENTAS DE ANÁLISIS DE CÓDIGO SEGURO PARA EL USO DE DESARROLLO DE SOFTWARE, AÑO 2024

**AUTOR:**

AXEL ISMAEL HUERA QUINATOA

**DIRECTOR:**

ING. CHRISTIAN FERNANDO BARRAGÁN QUIZHPE

**GUARANDA – ECUADOR**

**2024**

**TEMA DEL PROYECTO DE INVESTIGACIÓN**

HERRAMIENTAS DE ANÁLISIS DE CÓDIGO SEGURO PARA EL USO DE DESARROLLO DE SOFTWARE, AÑO 2024

**AGRADECIMIENTO**

Agradezco profundamente a todas las personas que han hecho posible la realización de este proyecto.

En primer lugar, extiendo mi gratitud a mi director, Ing. Christian Barragán, por su guía, paciencia y valiosas sugerencias que enriquecieron mi trabajo e impulso mi alcance a nuevos horizontes académicos.

Asimismo, deseo expresar mi más sincero agradecimiento a mi familia, cuyo apoyo incondicional, comprensión y aliento fueron la base que permitió superar cada obstáculo. Su fe en mí y en mis capacidades me inspiró a dar siempre lo mejor de mí.

Por último, deseo agradecer a mis pares académicos, quienes, con sus críticas constructivas y constante guía, generaron un ambiente de colaboración y crecimiento intelectual que fue esencial para cada etapa del proyecto.

**DEDICATORIA**

A mi madre, Silvia Quinatoa, por ser el pilar de mi vida, por su amor incondicional, sacrificio y esfuerzo incansable. Gracias por cada consejo, por creer en mí incluso cuando yo dudaba, y por ser mi mayor inspiración para seguir adelante.

A mi padre, Antonio Huera, por su apoyo constante, por enseñarme con su ejemplo el valor del trabajo, la perseverancia y la honestidad. Gracias por cada palabra de aliento y por brindarme las herramientas necesarias para alcanzar mis metas.

A mis hermanos, por estar siempre a mi lado, por su compañía y aliento en los momentos difíciles, y por celebrar conmigo cada pequeño logro. Su apoyo ha sido fundamental en este proceso.

A todas aquellas personas, de una u otra manera, han estado presentes, brindándome palabras de aliento y respaldo incondicional. Su confianza en mí ha sido una motivación constante.

A mis profesores, quienes con su guía, conocimientos y exigencia, me han ayudado a crecer académica y profesionalmente. Gracias por compartir su sabiduría y por fomentar en mí el deseo de aprender y mejorar cada día.

A todos ustedes, con profunda gratitud, les dedico este esfuerzo y logro.

**Axel I. Huera**

CERTIFICADO DE VALIDACIÓN

UNIVERSIDAD ESTATAL DE BOLIVAR

FACULTAD DE CIENCIAS ADMINISTRATIVAS, GESTIÓN EMPRESARIAL E INFORMÁTICA

## CERTIFICADO DE VALIDACIÓN

Ing. Christian Barragán, Fis. Rafael Medina e Ing. Danilo Barreno, en su orden Director y Pares Académicos del Trabajo de Integración Curricular "HERRAMIENTAS DE ANÁLISIS DE CÓDIGO SEGURO PARA EL USO DE DESARROLLO DE SOFTWARE, AÑO 2024" desarrollado por el Señor Axel Ismael Huera Quinatoa.

**CERTIFICAN**

Que, luego de revisado el Trabajo de Integración Curricular en su totalidad, cumple con las exigencias académicas de la carrera SOFTWARE, por lo tanto, autorizamos su presentación y defensa.

Guaranda, 26 de marzo del 2025




Ing. Christian Barragan
**Director**




Fis. Rafael Medina
**Par Académico**




Ing. Danilo Barreno
**Par Académico**

**DERECHOS DE AUTOR**

UEB | UNIVERSIDAD ESTATAL DE BOLIVAR

BIBLIOTECA GENERAL

**DERECHOS DE AUTOR**

Yo **Axel Ismael Huera Quinatoa** portador de la Cédula de Identidad No **1756044259** en calidad de autor y titular de los derechos morales y patrimoniales del Trabajo de Titulación: **HERRAMIENTAS DE ANÁLISIS DE CÓDIGO SEGURO PARA EL USO DE DESARROLLO DE SOFTWARE, AÑO 2024**, modalidad **Proyecto de Investigación**, de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN, concedemos a favor de la Universidad Estatal de Bolívar, una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos. Conservamos a mi favor todos los derechos de autor sobre la obra, establecidos en la normativa citada.

Así mismo, autorizo a la Universidad Estatal de Bolívar, para que realice la digitalización y publicación de este trabajo de titulación en el Repositorio Digital, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

El autor declara que la obra objeto de la presente autorización es original en su forma de expresión y no infringe el derecho de autor de terceros, asumiendo la responsabilidad por cualquier reclamación que pudiera presentarse por esta causa y liberando a la Universidad de toda responsabilidad.

Axel Ismael Huera Quinatoa
C.I. 1756044259

# ÍNDICE DE CONTENIDO

TEMA DEL PROYECTO DE INVESTIGACIÓN ..... i
AGRADECIMIENTO ..... ii
DEDICATORIA ..... iii
CERTIFICADO DE VALIDACIÓN ..... iv
DERECHOS DE AUTOR ..... v
ÍNDICE DE CONTENIDO ..... vi
ÍND ICE DE TABLAS ..... ix
ÍNDICE DE FIGURAS ..... xi
INTRODUCCIÓN ..... 1
RESUMEN ..... 3
ABSTRACT ..... 4
CAPÍTULO I ..... 5
FORMULACIÓN GENERAL DEL PROYECTO ..... 5
1.1. Descripción del Problema ..... 5
1.2. Formulación del Problema ..... 6
1.3. Preguntas de Investigación ..... 6
1.4. Justificación ..... 7
1.5. Objetivos: ..... 8
1.5.1. General ..... 8
1.5.2. Específicos ..... 8
1.6. Idea a defender ..... 9
1.7. Variables ..... 9
1.7.1. Operacionalización de Variables ..... 9
CAPÍTULO II ..... 11
MARCO TEÓRICO ..... 11

2.1. Antecedentes ..... 11
2.2. Científico ..... 13
2.3. Conceptual ..... 16
2.4. Legal ..... 21
CAPÍTULO III ..... 25
METODOLOGÍA ..... 25
3.1. Tipo de Investigación ..... 25
3.2. Enfoque de la investigación ..... 26
3.3. Métodos de Investigación ..... 27
3.4. Metodología General para la Evaluación ..... 27
3.5. Técnicas e Instrumentos de Recopilación de Datos ..... 28
3.6. Procesamiento de la Información ..... 30
CAPÍTULO IV ..... 31
RESULTADOS Y DISCUSIÓN ..... 31
4.1. Criterios de selección ..... 31
4.2. Software para el análisis ..... 34
4.3. Análisis, Interpretación y Discusión de Resultados ..... 35
4.4. Aplicación de las Herramientas de Análisis ..... 36
4.4.1. SonarQube ..... 36
4.4.2. Semgrep ..... 42
4.4.3. OWASP ASST ..... 46
4.4.4. HCL AppScan CodeSweep ..... 50
4.5. Análisis comparativo ..... 55
4.5.1. Análisis de Precisión ..... 55
4.5.2. Análisis de Recall ..... 56
4.5.3. Análisis de efectividad ..... 57

4.6. Resultado del análisis comparativo ..... 59
4.7. Discusión de resultados ..... 59
CAPÍTULO V ..... 62
PROPUESTA: ..... 62
5.1. Guía de uso para la herramienta OWASP ASST. ..... 62
5.2. Introducción ..... 62
5.3. Objetivo ..... 62
5.4. Descarga e Instalación ..... 63
5.4.1. Descarga e instalación de complementos. ..... 63
5.5. Guía de Configuración de OWASP ASST ..... 70
5.6. Ejecución de OWASP ASST ..... 73
CONCLUSIONES ..... 79
RECOMENDACIONES ..... 81
BIBLIOGRAFÍA ..... 82
ANEXOS ..... 87

# ÍNDICE DE TABLAS

**Tabla 1** Operacionalización de la Variable independiente ..... 9
**Tabla 2** Herramientas SAST ..... 31
**Tabla 3** Criterios de Selección de Herramientas de Análisis de Seguridad ..... 33
**Tabla 4** Descripción de los criterios de selección con cada herramienta. ..... 33
**Tabla 5** Requisitos hardware para SonarQube ..... 37
**Tabla 6** Requisitos Software para SonarQube ..... 37
**Tabla 7** Recursos utilizados por SonarQube en el análisis del Proyecto 1 ..... 39
**Tabla 8** Análisis de los resultados del proyecto 1 con SonarQube ..... 40
**Tabla 9** Recursos utilizados por SonarQube en el análisis del Proyecto 2 ..... 40
**Tabla 10** Análisis de los resultados del proyecto 2 con SonarQube ..... 41
**Tabla 11** Requisitos hardware para Semgrep ..... 42
**Tabla 12** Requisitos Software para Semgrep ..... 43
**Tabla 13** Recursos utilizados por Semgrep en el análisis del Proyecto 1 ..... 43
**Tabla 14** Análisis de los resultados Análisis de los resultados del proyecto 1 con Semgrep ..... 44
**Tabla 15** Recursos utilizados por Semgrep en el análisis del Proyecto 2 ..... 45
**Tabla 16** Análisis de los resultados del proyecto 2 con Semgrep ..... 46
**Tabla 17** Requisitos hardware para OWASP ASST ..... 47
**Tabla 18** Recursos utilizados por OWASP ASST en el análisis del Proyecto 1.. 47
**Tabla 19** Análisis de los resultados del proyecto 1 con OWASP ASST ..... 48
**Tabla 20** Recursos utilizados por OWASP ASST en el análisis del Proyecto 2.. 49
**Tabla 21** Análisis de los resultados del proyecto 2 con OWASP ASST ..... 50
**Tabla 22** Requisitos hardware para HCL AppScan CodeSweep ..... 51
**Tabla 23** Requisitos software para HCL AppScan CodeSweep ..... 51
**Tabla 24** Recursos utilizados por HCL AppScan CodeSweep en el análisis del Proyecto 1 ..... 52
**Tabla 25** Análisis de los resultados del proyecto 1 con HCL AppScan CodeSweep ..... 53
**Tabla 26** Recursos utilizados por HCL AppScan CodeSweep en el análisis del Proyecto 2 ..... 53

**Tabla 27** Análisis de los resultados del proyecto 2 con HCL AppScan CodeSweep ..... 54
**Tabla 28** Análisis comparativo de las herramientas (%)..... 55
**Tabla 29** Promedio global de los proyectos por métrica..... 57

# ÍNDICE DE FIGURAS

**Figura 1** Resultado del Análisis del Proyecto 1 con SonarQube ..... 39
**Figura 2** Resultado del Análisis del Proyecto 2 con SonarQube ..... 41
**Figura 3** Resultado del Análisis del Proyecto 1 con Semgrep ..... 44
**Figura 4** Resultado del Análisis del Proyecto 2 con Semgrep ..... 45
**Figura 5** Resultado del Análisis del Proyecto 1 con OWASP ASST ..... 48
**Figura 6** Resultado del Análisis del Proyecto 2 con OWASP ASST ..... 49
**Figura 7** Resultado del Análisis del Proyecto 1 HCL AppScan CodeSweep ..... 52
**Figura 8** Resultado del Análisis del Proyecto 2 HCL AppScan CodeSweep ..... 54
**Figura 9** Representación del promedio de los resultados de cada Herramienta... 57
**Figura 10** Página de descarga de XAMPP ..... 63
**Figura 11** Archivo de Instalación XAMPP ..... 64
**Figura 12** Inicio de la instalación XAMPP ..... 64
**Figura 13** Instalación completa de XAMPP ..... 65
**Figura 14** Panel de control XAMPP ..... 65
**Figura 15** Página de descarga de Node.js ..... 66
**Figura 16** Archivo de Instalación Node.js ..... 66
**Figura 17** Inicio de la instalación de Node.js ..... 67
**Figura 18** Términos y condiciones de Node.js ..... 67
**Figura 19** Herramienta OWASP ASST ..... 68
**Figura 20** Archivo de la herramienta OWASP ASST ..... 69
**Figura 21** Extracción de la herramienta ..... 69
**Figura 22** OWASP ASST y los proyectos a escanear ..... 70
**Figura 23** Archivo config.js ..... 71
**Figura 24** Archivo config_php_lang.js ..... 71
**Figura 25** Modificación de las variables de la base de datos ..... 72
**Figura 26** Configuración de los apartados user y admin ..... 72
**Figura 27** Configuración de la columna contraseña del user y admin ..... 73
**Figura 28** Archivo ASST.bat ..... 74
**Figura 29** Ejecución de OWASP ASST ..... 74
**Figura 30** Finalización del proceso de análisis ..... 75
**Figura 31** Archivo report.html ..... 76

**Figura 32** Reporte del análisis con OWASP ASST ..... 77

## INTRODUCCIÓN

La seguridad del código fuente en el desarrollo de software presenta un aspecto crítico para garantizar la integridad, disponibilidad y confidencialidad en las aplicaciones, dado que, en este tipo de entornos, donde las amenazas evolucionan constantemente es imprescindible que los desarrolladores implementen herramientas y prácticas que permitan detectar y mitigar los fallos, errores y vulnerabilidades en etapas tempranas del ciclo de vida del software.

En este contexto, la revisión de código fuente es una estrategia fundamental para mejorar la calidad y seguridad de los proyectos de desarrollo web, el interés de esta investigación se presenta gracias a la creciente necesidad de fortalecer estos aspectos, teniendo en cuenta que la proliferación de herramientas de análisis de código seguro ofrece múltiples opciones para los desarrolladores.

Debido a esto, el estudio presenta una metodología basada en un análisis comparativo de diversas herramientas de análisis de código estático (SAST), evaluando su desempeño en entornos de desarrollo web.

El propósito de esta investigación es presentar un análisis comparativo, facilitando a los desarrolladores la selección de la herramienta de análisis de código seguro adecuada a sus necesidades. Proporcionada una evaluación detallada basada en pruebas controladas, contribuyendo a la toma de decisiones informadas en el ámbito de la seguridad del software.

El contenido del documento se presenta en cinco capítulos.

En el **Capítulo I**, se define el problema de la investigación, se justifica la importancia del estudio del mismo modo se enuncia tanto el objetivo principal como los objetivos específicos.

El **Capítulo II**, presenta el marco teórico, abordando antecedentes, conceptos clave y normativas de seguridad relevantes.

En el **Capítulo III**, se describe la metodología empleada en el estudio donde se detalla el tipo de investigación, el enfoque de la misma, así como las técnicas de recolección de datos y el procesamiento de la investigación.

El **Capítulo IV** presenta los criterios para seleccionar las herramientas, al igual que, expone los resultados obtenidos de las pruebas, comparando el desempeño de cada herramienta con su respectivo análisis.

Finalmente, en el **Capítulo V** se muestra una propuesta práctica en formato de guía de uso para la herramienta seleccionada como la opción más equilibrada en términos de rendimiento y fiabilidad.

**RESUMEN**

En el presente estudio se comparó las principales herramientas de análisis de código estático disponibles en el mercado en 2024 que se utilizó para evaluar proyectos de desarrollo web, con el objetivo de determinar su utilidad y eficacia en la detección de vulnerabilidades reales, así como identificar diferencias en cuanto a efectividad, tiempo de análisis y consumo de recursos. Se empleó un método experimental complementado con un método comparativo para realizar pruebas controladas en dos proyectos software representativos. El enfoque adoptado para la investigación corresponde al cuantitativo y cualitativo, con técnicas de medición como el análisis de precisión, recall pertenecientes a las métricas de la curva ROC, complementadas con observación y revisión documental. Aunque todas las tecnologías utilizadas las cuales incluye a: SonarQube, Semgrep, OWASP ASST y HCL AppScan CodeSweep; fueron eficaces a la hora de detectar vulnerabilidades se encontraron diferencias significativas, con lo cual se creó una guía de uso con la herramienta que tiene el mayor equilibro ente las métricas para medir la fiabilidad de las herramientas evaluadas. Esto indica que la selección de herramientas adecuadas es crítica y debe ajustarse a las características específicas de cada proyecto, destacando la importancia de incorporar estas tecnologías en las primeras fases del ciclo de desarrollo para mitigar los riesgos de seguridad.

**ABSTRACT**

This study compared the main static code analysis tools available on the market in 2024 that were used to evaluate web development projects, in order to determine their usefulness and efficiency in detecting real vulnerabilities, as well as to identify differences in terms of effectiveness, analysis time and resource consumption. An experimental method complemented with a comparative method was used to perform controlled tests on two representative software projects. The approach adopted for the research corresponds to the quantitative and qualitative, with measurement techniques such as precision analysis, recall belonging to the metrics of the ROC curve, complemented with observation and documentary review. Although all the technologies used which include: SonarQube, Semgrep, OWASP ASST and HCL AppScan CodeSweep; were effective in detecting vulnerabilities significant differences were found, with which a usage guide was created with the tool that has the greatest balance between the metrics to measure the reliability of the tools evaluated. This indicates that the selection of appropriate tools is critical and must be adjusted to the specific characteristics of each project, highlighting the importance of incorporating these technologies early in the development cycle to mitigate security risks.

# CAPÍTULO I

# FORMULACIÓN GENERAL DEL PROYECTO

## 1.1. Descripción del Problema

Para un mundo globalizado donde se comparte y recibe información de diferentes tipos, la seguridad de los sistemas informáticos por donde se transmiten estos datos es de vital importancia, con el avance vertiginosos de la tecnología, aumenta la posibilidad de que el software presente fallos o en su defecto sea violentado de manera maliciosa y deliberada, teniendo en cuenta que no existe un sistema cien por ciento invulnerable.

Entre los problemas que pueden afectar al código de software desarrollado, se encuentran las inyecciones, las cuales son un tipo de vulnerabilidad que ocurre cuando los datos proporcionados por el usuario no son validados, filtrados o sanitizados correctamente, permitiendo que consultas dinámicas, comandos o procedimientos almacenados sean manipulados de forma maliciosa (OWASP, 2021).

El Cross-Site Scripting (XSS) es otro problema común donde los atacantes inyectan scripts maliciosos en páginas web vistas por otros usuarios, comprometiendo sus datos y credenciales. Los ataques XSS pueden introducirse en cualquier lugar donde un programa web incorpore la entrada de datos proporcionado por el usuario en su contenido generado sin validarlos o codificarlos correctamente (KirstenS, 2024).

La exposición de datos sensibles describe como la mayoría de los desarrolladores web dejan información relevante para la organización en texto plano mientras crean aplicaciones en línea (Zambrano et al., 2019, p. 303).

La falta de experiencia de los desarrolladores que provocan incorrectas configuraciones de seguridad, como parámetros predeterminados inseguros o inapropiados, pueden ser explotadas por aquellos con interés de atacar a un sistema con la finalidad de acceder a datos sensibles o funciones administrativas. La

deserialización insegura de datos no confiables puede resultar en la ejecución de código arbitrario o la escalada de privilegios. Además, el uso de componentes con vulnerabilidades conocidas, como bibliotecas y frameworks, puede comprometer la seguridad del software. De igual manera la insuficiente capacidad de registro y monitoreo puede impedir la detección y respuesta a incidentes de seguridad a tiempo.

Los desarrolladores a menudo no conocen las herramientas de seguridad, lo que resulta en costos elevados y problemas para integrarlas. La presión por tiempos cortos de desarrollo y entrega prioriza la funcionalidad sobre la seguridad, lo que resulta en la creación de software con vulnerabilidades. Sin el uso de herramientas de análisis de código seguro aumenta el riesgo de fallos y vulnerabilidades, que podrían ser evitadas con su implementación adecuada. Esto pue

de causar pérdidas económicas y de reputación para las empresas, además de poner en peligro la privacidad y la seguridad de los usuarios finales.

En la actualidad, la inseguridad en el desarrollo de software es un problema importante debido a las vulnerabilidades en el código fuente. La complejidad de los sistemas informáticos, que incluyen una variedad de tecnologías y plataformas, dificulta la gestión y la seguridad del código. Además, la falta de conciencia y capacitación en seguridad entre los desarrolladores, junto con el uso inadecuado o inexistente de herramientas de análisis de seguridad, agrava la situación.

## 1.2. Formulación del Problema

¿Qué herramientas de análisis de código seguro son fiables para evaluar de manera adecuada los proyectos de desarrollo de software?

## 1.3. Preguntas de Investigación

- ¿Cuáles son las principales herramientas de análisis de código seguro disponibles en el mercado para el año 2024?
- ¿Qué tan eficientes son estas herramientas de análisis de código en la detección de vulnerabilidades de seguridad reales en proyectos de software?

- ¿Cómo comparar el rendimiento de las diferentes herramientas de análisis de código seguro en términos de eficiencia, tiempos de análisis y consumo de recursos?

## 1.4. Justificación

En la actualidad la sociedad se caracteriza por su alta dependencia de la tecnología y el software, gran parte de las actividades diarias están relacionadas con sistemas informáticos y aplicaciones, desde las interacciones diarias hasta el manejo de infraestructuras críticas, la introducción regular de nuevas aplicaciones y sistemas en nuestra vida cotidiana es el resultado de la evolución continua de nuevas tecnologías e innovaciones de aplicaciones, lo que consolida aún más el papel de las aplicaciones y los sistemas informáticos en todos los aspectos de nuestra existencia.

El análisis de código con el apoyo de herramientas tales como las SAST resulta de gran ayuda para los equipos de desarrollo que operan bajo presión para entregar producto de calidad a tiempo, cumpliendo con los estándares de codificación y normativas de cumplimiento, donde los errores no son una opción (Bellairs, 2019).

Sin embargo, es importante reconocer que actualmente no existe una herramienta SAST infalible, cuyos resultados sean cien por ciento confiables sin un proceso de selección adecuado en base a resultados. Esto se debe a que el software es un ente vivo, sujeto a cambios y actualizaciones constantes, así como a la detección continua de errores de seguridad en sus funciones (García, 2019).

Dada la importancia crítica de la seguridad del software y la necesidad de herramientas fiables y eficientes que permitan analizar código, esta investigación es pertinente, debido a que, se centrará en la evaluación de dichos softwares disponibles en el mercado. Teniendo como meta identificar las fortalezas y debilidades, su capacidad para integrarse en procesos de desarrollo ágiles y su efectividad en la detección y mitigación de vulnerabilidades, la investigación no solo intentará dar una guía actualizada para los equipos de desarrollo, sino que

también contribuirá al avance del campo de la seguridad del software al destacar áreas donde se necesita más innovación y mejora.

El uso eficaz de SAST implica integrar estas herramientas en los procesos de desarrollo y entrega establecidos, permitiendo a los desarrolladores identificar y solucionar vulnerabilidades desde las primeras etapas del desarrollo del proyecto (Li, 2020). Esta integración ayuda a descubrir y mitigar problemas de seguridad mucho antes de que el software llegue a la fase de pruebas, lo cual es esencial en un entorno donde la complejidad de los sistemas informáticos está en constante aumento y donde la presión por reducir los tiempos de desarrollo puede comprometer la seguridad del producto final.

Por tanto, es factible el desarrollo del presente trabajo de investigación ya que se seleccionará y evaluará las herramientas adecuadas de análisis de código seguro, ya que las organizaciones deben fomentar una cultura de seguridad que valore tanto la rapidez en la entrega como la integridad del código, y los desarrolladores deben estar bien informados y capacitados para usar estas herramientas.

## 1.5. Objetivos:

### 1.5.1. General

Evaluar las herramientas de análisis de código seguro enfocadas en proyectos de desarrollo web disponibles en el mercado para el año 2024

### 1.5.2. Específicos

- Identificar las principales herramientas de análisis de código seguro disponibles en el mercado para el año 2024.
- Evaluar la eficacia de cada herramienta en la detección de vulnerabilidades de seguridad reales en proyectos de software focalizando el desarrollo web.
- Comparar el rendimiento de las diferentes herramientas en términos de: tiempos de análisis y consumo de recursos.
- Seleccionar la o las herramientas más adecuadas para su implementación en entornos de desarrollo de software.

## 1.6. Idea a defender

Las principales herramientas de análisis de código seguro disponibles en el mercado en 2024 serán útiles y eficaces en la detección de vulnerabilidades de seguridad reales en proyectos web, pero presentarán diferencias significativas en términos de efectividad, tiempos de análisis y consumo de recursos

## 1.7. Variables

**Variable independiente:**

Rendimiento de las herramientas de análisis de código seguro.

### 1.7.1. Operacionalización de Variables

**Tabla 1**

*Operacionalización de la Variable independiente*

| Variable | Definición conceptual | Dimensión | Indicadores | Métricas |
|---|---|---|---|---|
| Rendimiento de las herramientas de análisis de código seguro | Efectividad general de las herramientas para detectar vulnerabilidades, eficiencia operativa, facilidad de uso, mejora de la seguridad sin afectar la productividad. | Eficacia en detección<br>Eficiencia<br>Usabilidad<br>Impacto en seguridad | Eficacia en la detección de vulnerabilidades<br>Eficiencia (tiempos de análisis y consumo) | Tasa de detección de vulnerabilidades reales (Precisión)<br>Tasa de falsos positivos y negativos (Recall)<br>Tiempos de análisis |

| | |
|---|---|
| | Consumo de recursos |
| | Facilidad de uso |

*Nota*. La tabla presenta la descripción de la variable dependiente para el estudio.

*Fuente:* Elaboración propia.

# CAPÍTULO II

# MARCO TEÓRICO

## 2.1. Antecedentes

A lo largo de las últimas décadas, con el incremento de la complejidad de los sistemas tecnológicos, las amenazas cibernéticas se han tornado paulatinamente más sofisticadas, en respuesta a dichas amenazas, distintas organizaciones han adoptado progresivamente diversas técnicas y herramientas, entre las que sobresalen los sistemas de análisis de código seguro, las cuales tienen el propósito de evaluar y garantizar la integridad del código fuente desde las fases iniciales del ciclo de desarrollo.

Cabe destacar que los esfuerzos por mejorar la seguridad del software no constituyen un fenómeno reciente, sino que representan el resultado de una evolución sostenida, impulsada por la necesidad de hacer frente a los desafíos emergentes en el ámbito de la ciberseguridad.

De acuerdo a López Álvarez (2020) la ingeniería en software busca ordenar y maximizar la eficiencia de la creación de software dentro de las empresas. Por su parte, García-Peñalvo (2018), menciona que la ingeniería en software nace por: “La incapacidad de las organizaciones para predecir tiempo, esfuerzos y costes en el desarrollo de software producido son dos de las principales bases sobre las que surge la Ingeniería del Software como una disciplina científica”, lo que dio lugar a la ingeniería de software como una diciplina científica formal.

La seguridad del software se convierte en un elemento esencial en este contexto, ya que afecta directamente nuestra dependencia de la tecnología, la protección contra amenazas cibernéticas, la privacidad de los datos y la implementación de nuevos avances tecnológicos (De Vega Martín, 2024). Es un aspecto crítico para mejorar el funcionamiento seguro y efectivo de muchas de las actividades y servicios que damos por sentado en la vida cotidiana.

El uso de herramientas de análisis de código, específicamente las herramientas de análisis estático de código (SAST), ha evolucionado significativamente en los últimos años. Según Eterovic (2023) en su publicación “Análisis de las herramientas para realizar pruebas estáticas de seguridad de las aplicaciones” Las herramientas SAST permiten automatizar la detección de vulnerabilidades y se pueden integrar al sistema de integración y la distribución o implementación continuas para que detecten vulnerabilidades en etapas tempranas del ciclo de vida del desarrollo del software. Esto ayuda al equipo de Seguridad de las Aplicaciones a implementar un ciclo de vida seguro. Es decir que, ayuda a los desarrolladores a corregir el código realizado de manera oportuna y viable permitiendo un despliegue seguro de dicha aplicación.

Para Caño Quintero (2019) integrar la seguridad en todas las etapas del proceso de desarrollo de manera activa requiere hacer uso de herramientas que permitan la automatización de los controles de seguridad. Estos controles automáticos, deberán comprobar que no existan vulnerabilidades cuando se produzca un cambio en el código o en la infraestructura antes de que entren en producción.

Jurn et al. (2018) menciona que, las vulnerabilidades de software se refieren a errores de codificación en el programa que causan acciones no intencionadas en el software. Estos errores de codificación pueden resultar en un fallo del sistema, una pérdida de conectividad, la incapacidad de los usuarios para iniciar sesiones, la actualización de los privilegios del usuario, la concesión de acceso a los usuarios no aprobados, y la creación de errores de impresión. Todo el software es susceptible, pero los sistemas operativos, navegadores web, procesadores de texto, calculadoras, reproductores de vídeo y audio, sistemas integrados y firmware son particularmente vulnerables.

## 2.2. Científico

### Norma ISO 27001

La ISO/IEC 27001 es la norma más conocida del mundo para sistemas de gestión de la seguridad de la información (SGSI). Esta norma define los requisitos que debe cumplir un SGSI (Organización Internacional de Normalización (ISO), 2013)

El objetivo de la norma ISO 27001 es apoyar la gestión de la seguridad de la información dentro de las empresas. A cada tipo de empresa, independientemente de su tamaño y actividad, le interesa.

El principal determinante para decidir si implementar un sistema de gestión de seguridad de la información es la importancia de las actividades de información dentro de una organización como componentes esenciales para lograr sus objetivos.

### Herramientas de análisis estático de código (SAST)

Cuando se habla de las herramientas SAST (Static Application Security Testing) es necesario saber que estas se basan en técnicas de revisión de código fuente, cuyo principal objetivo es identificar los posibles fallos de seguridad.

Así pues, OWASP define a este análisis de código estático como:

El análisis de código estático (también conocido como análisis de código fuente) se realiza generalmente como parte de una revisión de código (también conocida como prueba de caja blanca) y se lleva a cabo en la fase de implementación de un ciclo de vida de desarrollo de seguridad (SDL). El análisis de código estático se refiere comúnmente a la ejecución de herramientas de análisis de código estático que intentan destacar posibles vulnerabilidades dentro del código fuente "estático" (no en ejecución) mediante el uso de técnicas como el análisis de contaminación y el análisis de flujo de datos. (Dewhurst, 2024, párr. 1)

Gartner (2024) propone que Static Application Security Testing (SAST) es una colección de herramientas creadas para examinar binarios, código de bytes y código fuente de aplicaciones en busca de fallos de codificación y diseño que pueden

indicar fallos de seguridad. Cuando un sistema no esté operativo, las soluciones SAST lo examinan por dentro.

Estas herramientas de seguridad son prácticas al momento de la detección de vulnerabilidades, sin embargo, hay que tener en cuenta que no existe una herramienta absolutamente fiable, por ende, como en OWASP se menciona la existencia de los falsos positivos y falsos negativos son una limitación de este tipo de software.

**Falsos positivos**

Cuando una herramienta de análisis estático de código sugiere una posible vulnerabilidad que en realidad no existe, con frecuencia genera falsos positivos. Esto ocurre con frecuencia como resultado de la incapacidad de la herramienta para garantizar la seguridad y la integridad de los datos a medida que pasan de la entrada a la salida dentro de la aplicación. (Dewhurst, 2024, párr. 13).

**Falsos negativos**

El uso de herramientas de análisis estático de código puede producir resultados falsos negativos, en los que la herramienta encuentra vulnerabilidades, pero no informa de ellas. Esto puede ocurrir si se encuentra un componente externo con una nueva vulnerabilidad o si la herramienta de análisis desconoce el entorno de ejecución y su configuración de seguridad. (Dewhurst, 2024, párr. 14)

**Herramientas SAST utilizadas para el Estudio**

**SonarQube**

Según la documentación de su página web oficial SonarQube es una herramienta autónoma de revisión de código que facilita la entrega continua de código limpio. Se integra en los flujos de trabajo existentes para identificar problemas en el código, apoyando inspecciones continuas de los proyectos. (SonarQube, 2024, párr. 1)

Es una plataforma de código abierto para la inspección continua de la calidad del código, que realiza un análisis estático para detectar vulnerabilidades, bugs y malas prácticas en código. SonarQube fue seleccionada por su amplia cobertura de

vulnerabilidades, facilidad de integración con herramientas de desarrollo comunes, y su fuerte soporte comunitario.

**Semgrep**

Semgrep es una herramienta de seguridad de aplicaciones para encontrar errores, detectar vulnerabilidades de dependencia y aplicar estándares de código. Sus reglas se parecen al código que ya escribes: sin árboles de sintaxis abstracta, luchas con expresiones regulares ni DSL engorrosos. (Semgrep.dev, 2024)

Es una herramienta de análisis estático ligera y de código abierto que permite realizar análisis rápidos y precisos del código. Semgrep fue seleccionada por su flexibilidad, capacidad de personalización y su enfoque en proporcionar reglas específicas y contextuales para la detección de vulnerabilidades.

**OWASP ASST**

ASST es una herramienta de escaneo de código fuente de código abierto, es una aplicación CLI (Interfaz de línea de comandos), desarrollada con JavaScript (framework Node.js).

Actualmente se concentra en los lenguajes de programación PHP y MySQL, pero como sus funcionalidades básicas están listas y disponibles para todos, los programadores pueden contribuir y agregar complementos o extensiones, para añadir características y hacer que escanee otros lenguajes de programación como Java, C#, Python, etc., y sus frameworks. Por lo que su infraestructura está diseñada para que otros programadores contribuyan con ella para hacerla mejor y más novedosa.  Seyceri (2024)

**HCL AppScan CodeSweep**

HCL AppScan CodeSweep es una herramienta de pruebas estáticas de seguridad de aplicaciones (SAST) centrada en el desarrollador y diseñada para principiantes (y profesionales) que necesitan un programa rápido, sencillo y compatible con la plataforma. CodeSweep proporciona pruebas de seguridad sobre la marcha, funciones de corrección automática, compatibilidad con más de 30 lenguajes/frameworks (HCL Software, 2024).

Para el presente proyecto se optó por utilizar la extensión HCL AppScan para Visual Studio Code la cual proporciona la funcionalidad de pruebas de seguridad de aplicaciones estáticas (SAST) para detectar vulnerabilidades en las primeras etapas del ciclo de vida del desarrollo.

## 2.3. Conceptual

### Ataque informático

Un ataque informático es una actividad que pretende explotar una vulnerabilidad de un sistema informático para influir en él o incluso hacerse con su control. Puede incluir tanto actividades intencionadas como no intencionadas que pongan en peligro el sistema. (Coronel Suárez & Quirumbay Yagual, 2022)

### Aplicaciones web

Una aplicación web, o "web app", es una pieza de software que se ejecuta en un servidor web. A diferencia de las aplicaciones de escritorio tradicionales, que se ejecutan a través del sistema operativo, a las aplicaciones web se accede a través de un navegador web, ya que son aplicaciones basadas en la web. Como resultado, una aplicación web debe programarse en un lenguaje que sea comprensible para un navegador web. Los navegadores web entienden un número finito de lenguajes, lo que significa que las aplicaciones web deben estar programadas en uno de ellos para ser entendidas (Sarhan et al., 2018).

Los entornos operativos en los que funcionan las aplicaciones web son accesibles a través de Internet; el acceso aquí no se refiere necesariamente al uso del programa, sino a la interacción con el ordenador que lo aloja. La forma en que se utilizan e incluso se descargan las aplicaciones web es similar, pero difieren entre sí en cuanto a la tecnología, el número de servicios que ofrecen y, con frecuencia, los protocolos que las sustenta, si las comparamos con sus homólogas de escritorio, podemos observar incluso que operan en entornos diferentes. (Coronel Suárez & Quirumbay Yagual, 2022)

**Buffer Overflow**

Mediante el método de ensayo y error, estos ataques pretenden recabar información hasta que encuentra la combinación correcta, el atacante prueba metódicamente todas las combinaciones posibles. Debido a la naturaleza del ataque mediante el cual se intenta utilizar todos los medios posibles para obtener información específica, la cual, podría ser una clave de cifrado o una contraseña, se conoce como "ataque de fuerza bruta".

Es importante señalar que, aunque este tipo de ataque no es nuevo, los hackers siguen utilizándolo ya que, dependiendo de su longitud y complejidad, el valor puede recuperarse tras un número predeterminado de intentos. Por ejemplo, puede llevar hasta 16 intentos averiguar el valor correcto de una clave de cifrado de 4 bits intentando todas las combinaciones posibles (Sánchez & Ramírez, 2022).

**Ciclo de vida de desarrollo de software (SDLC)**

La normativa ISO/IEC/IEEE 12207:2017 establece, un marco uniforme para los procesos del ciclo de vida del software con una nomenclatura bien definida que puede utilizar la industria del software. Describe los procesos, acciones y deberes implicados en la adquisición, suministro, desarrollo, operación, mantenimiento y eliminación de sistemas, bienes y servicios de software. Estas actividades del ciclo de vida se llevan a cabo con la participación de las partes interesadas, con el objetivo último de satisfacer las expectativas del cliente (Organización Internacional de Normalización (ISO), 2017).

**Cross-Site Scripting (XSS)**

Cross-Site Scripting (XSS) son un tipo de inyección, en los que se introducen scripts maliciosos en sitios web de confianza y por lo demás inocuos. Los ataques XSS se producen cuando un hacker envía código malicioso, normalmente en forma de script del lado del navegador, a otro usuario final a través de una aplicación web. Dondequiera que un programa web incorpore la entrada del usuario en la salida que produce sin verificarla o codificarla, existen fallos que hacen posibles estos ataques. Un atacante puede enviar un script malicioso a un usuario desprevenido utilizando

XSS. El navegador del usuario final ejecutará el script ya que no tiene medios para saber que no debería ser de confianza. (KirstenS, 2024)

**Denial of Service (DoS)**

Dejar el sistema inutilizable es una característica distintiva de un ataque de denegación de servicio. Esto ocurre porque los servidores que gestionan las peticiones de los usuarios están sobrecargados, lo que provoca un alto volumen de tráfico en la red. Esto sobrecarga los recursos del sistema, haciendo que funcione mal y deje de responder a las peticiones válidas de los usuarios. El modo de denegación de servicio distribuido (DDoS) es otra forma en que pueden producirse los ataques DoS, en los que varias máquinas asaltan el mismo objetivo simultáneamente. A menudo se utiliza una red de bots conocida como botnet para llevar a cabo este asalto, en el que cada bot envía múltiples solicitudes de conexión al servidor objetivo (Sánchez & Ramírez, 2022).

**Inyección de código SQL**

Un ataque de inyección SQL es la inserción o "inyección" de una consulta SQL en el programa a través de los datos de entrada del cliente. Un exploit de inyección SQL exitoso puede leer datos sensibles de la base de datos, modificar sus registros (Insertar/Actualizar/Borrar), realizar operaciones de administración de la base de datos (como apagar el DBMS), recuperar el contenido de un archivo específico en el sistema de archivos del DBMS y, en algunos casos, emitir comandos al sistema operativo (Sarhan et al., 2018).

**Phishing**

El phishings una forma de robo de identidad Para acceder a sus cuentas o cometer delitos utilizando sus identificadores de usuario y contraseñas, los atacantes intentan robar sus datos e información personal. Los atacantes suelen intentar obtener información como números de tarjetas bancarias, números de la seguridad social o números de empleados.  Los enlaces a sitios web que se hacen pasar por fuentes fiables pueden utilizarse como ataques. Tras hacer clic en ese enlace, se pide al usuario que introduzca su nombre de usuario y contraseña en una página ficticia (Sánchez & Ramírez, 2022).

**Receiver Operating Characteristic (ROC)**

De acuerdo con Ward Powers Receiver (2008) Receiver Operating Characteristic (ROC) es una técnica para visualizar, organizar y seleccionar clasificadores en función de su rendimiento. Los gráficos ROC se han utilizado durante mucho tiempo en la teoría de detección de señales para representar la compensación entre las tasas de aciertos y las tasas de falsas alarmas de los clasificadores.

**Seguridad de las aplicaciones web**

La seguridad de las aplicaciones web es la práctica de salvaguardar los datos sensibles que se mantienen en línea de accesos y cambios no autorizados, esto se consigue aplicando políticas estrictas. Las personas malintencionadas pueden intentar acceder a información sensible y poner en peligro los datos de una organización. El objetivo de la seguridad de las aplicaciones web es identificar los activos críticos de la organización, los usuarios reales que pueden acceder a los datos, el nivel de acceso concedido a cada usuario, las diversas vulnerabilidades que pueden existir en la aplicación, la criticidad de los datos, el análisis de riesgos sobre la exposición de los datos y las medidas correctivas apropiadas (Sarhan et al., 2018).

**Seguridad del software**

Dentro del ciclo de vida del software se puede entender por seguridad los requerimientos del sistema y poder cumplir con sus funciones, pero realmente no es así, se estima que hay una gran confusión con el tema seguridad el cual es un concepto delicado con respecto al manejo de datos. Es importante aplicar la seguridad al ciclo de vida del software con el fin de tener calidad en los productos desarrollados y cumplir con los tres objetivos de la seguridad que son la integridad, confidencialidad y disponibilidad (López Álvarez, 2020)

**Seguridad de la información**

La seguridad de la información se refiere a proteger los datos y los sistemas de información contra el acceso ilegal, el uso, la divulgación, la alteración o la destrucción es el objetivo de la seguridad de la información, una actividad estratégica fundamental para toda empresa. Para preservar el valor de la

información y garantizar la continuidad de las operaciones incluso frente a posibles amenazas, se toman decisiones y medidas para garantizar su disponibilidad, confidencialidad e integridad. (Arpi-Saquipay & Cajamarca-Criollo, 2023)

**Software libre**

De acuerdo con Stallman "con software libre nos referimos a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software."(2020, p. 152).

Stallman defina 4 clases de libertad para los usuarios de software:

- Libertad 0: la libertad para ejecutar el programa sea cual sea nuestro propósito.
- Libertad 1: la libertad para estudiar el funcionamiento del programa y adaptarlo a tus necesidades - el acceso al código fuente es condición indispensable para esto.
- Libertad 2: la libertad para redistribuir copias y ayudar así a tu vecino.
- Libertad 3: la libertad para mejorar el programa y luego publicarlo para el bien de toda la comunidad - el acceso al código fuente es condición indispensable para esto.

Cualquier aplicación que conceda a sus usuarios estas libertades se considera software libre. Por lo tanto, se le debe permitir distribuir copias libremente o por un precio a todo el mundo, donde sea, con o sin cambios. Entre otras cosas, disfrutar de esta libertad implica no tener que pagar por ella ni obtener permiso. (Stallman, 2020)

**Vulnerabilidades**

Las vulnerabilidades son defectos en los sistemas, no puertas abiertas creadas intencionadamente, sino errores de diseño, configuración o implementación que crean posibilidades de ataque, haciendo así plausible una amenaza (Veloz & Veloz, 2024).

### Vulnerabilidades en aplicaciones web

Las vulnerabilidades en las aplicaciones web son puntos débiles de seguridad que los atacantes pueden explotar para poner en peligro la integridad, confidencialidad y disponibilidad de la aplicación. Estas vulnerabilidades pueden producirse en muchas fases de la aplicación y deben abordarse utilizando métodos de seguridad adecuados (Veloz & Veloz, 2024).

## 2.4. Legal

### Ley Orgánica de Protección de Datos Personales

En Ecuador, la Ley Orgánica de Protección de Datos Personales, 2021 (LOPDP) regula el manejo de datos de esta característica. Es fundamental obtener el consentimiento informado de los participantes de encuestas o entrevistas y proteger la privacidad de sus datos.

**Art. 8.-**Consentimiento. - Se podrán tratar y comunicar datos personales cuando se cuente con la manifestación de la voluntad del titular para hacerlo. El consentimiento será válido, cuando la manifestación de la voluntad sea:

1) Libre, es decir, cuando se encuentre exenta de vicios del consentimiento;

2) Específica, en cuanto a la determinación concreta de los medios y fines del tratamiento;

3) Informada, de modo que cumpla con el principio de transparencia y efectivice el derecho a la transparencia,

4) Inequívoca, de manera que no presente dudas sobre el alcance de la autorización otorgada por el titular.

El consentimiento podrá revocarse en cualquier momento sin que sea necesaria una justificación, para lo cual el responsable del tratamiento de datos personales establecerá mecanismos que garanticen celeridad, eficiencia, eficacia y gratuidad, así como un procedimiento sencillo, similar al proceder con el cual recabó el consentimiento.

El tratamiento realizado antes de revocar el consentimiento es lícito, en virtud de que este no tiene efectos retroactivos. Cuando se pretenda fundar el tratamiento de los datos en el consentimiento del afectado para una pluralidad de finalidades será preciso que conste que dicho consentimiento se otorga para todas ellas.

**Código Orgánico de la Economía de los Conocimientos, Creatividad e Innovación**

Código Orgánico Economía Social Conocimientos Creatividad Innovación, 2016 marca un paso dentro del sistema de Propiedad Intelectual debido a que promueve el uso de Software libre, tecnología que no sólo permite conocer el código fuente de sus programas sino también posibilita la modificación de las herramientas informáticas de acuerdo a las necesidades de cada persona.

**Apartado Segundo:** De las tecnologías libres y formatos abiertos

**Artículo 142.-** Tecnologías libres. - Se entiende por tecnologías libres al software de código abierto, los estándares abiertos, los contenidos libres y el hardware libre. Los tres primeros son considerados como Tecnologías Digitales Libres.

Se entiende por software de código abierto al software en cuya licencia el titular garantiza al usuario el acceso al código fuente y lo faculta a usar dicho software con cualquier propósito. Especialmente otorga a los usuarios, entre otras, las siguientes libertades esenciales:

- La libertad de ejecutar el software para cualquier propósito;
- La libertad de estudiar cómo funciona el software, y modificarlo para adaptarlo a cualquier necesidad. El acceso al código fuente es una condición imprescindible para ello;
- La libertad de redistribuir copias; y,
- La libertad de distribuir copias de sus versiones modificadas a terceros.

Se entiende por código fuente, al conjunto de instrucciones escritas en algún lenguaje de programación, diseñadas con el fin de ser leídas y transformadas por alguna herramienta de software en lenguaje de máquina o instrucciones ejecutables en la máquina.

**Decreto 1014 Software libre en Ecuador**

Decreto N° 1014, Política pública para las entidades de la Administración Pública Central la utilización del software libre en sus sistemas y equipamientos informáticos, 2008

**Artículo 1.-** Establecer como política pública para las entidades de la Administración Pública Central la utilización de software libre en sus sistemas y equipamientos informáticos.

**Artículo 2.-** Se entiende por software libre, a los programas de computación que se pueden utilizar y distribuir sin restricción alguna, que permitan su acceso a los códigos fuentes y que sus aplicaciones puedan ser mejoradas. Estos programas de computación tienen las siguientes libertades:

a) Utilización del programa con cualquier propósito de uso común;

b) Distribución de copias sin restricción alguna;

c) Estudio y modificación del programa (Requisito: código fuente disponible); Y.

d) Publicación del programa mejorado (Requisito: código fuente disponible).

**Artículo 3.-** Las entidades de la Administración Pública Central previa a la instalación del software libre en sus equipos, deberán verificar la existencia de capacidad técnica que brinde el soporte necesario para el uso de este tipo de software.

**Artículo 4.-** Se faculta la utilización de software propietario (no libre) únicamente cuando no exista una solución de software libre que supla las necesidades requeridas, o cuando esté en riesgo la seguridad nacional, o cuando el proyecto informático se encuentre en un punto de no retorno. Para efectos de este decreto se comprende como seguridad nacional, las garantías para la supervivencia de la colectividad y la defensa de patrimonio nacional. Para efectos de este decreto se entiende por un punto de no retorno. cuando el sistema o proyecto informático se encuentre en cualquiera de estas condiciones:

a) Sistema en producción funcionando satisfactoriamente que un análisis de costo beneficio muestre que no es razonable ni conveniente una migración a software libre; y,

b) Proyecto en estado de desarrollo y que un análisis de costo- beneficio muestre que no es conveniente modificar el proyecto y utilizar software libre.

Periódicamente se evaluarán los sistemas informáticos que utilizan software propietario con la finalidad de migrarlos a software libre

# CAPÍTULO III

# METODOLOGÍA

## 3.1. Tipo de Investigación

### Investigación descriptiva

Según Guevara Alban et al. (2020), la investigación descriptiva busca identificar las características de la población estudiada. Se describe como el proceso de documentar, analizar y comprender el estado actual de los fenómenos, incluyendo su composición y procesos, su objetivo es descubrir las tendencias predominantes y comprender cómo se comporta una persona, grupo o entidad en el presente. Este tipo de investigación pretende caracterizar las propiedades esenciales de colecciones homogéneas de fenómenos utilizando criterios sistemáticos, lo que permite determinar su estructura o comportamiento, así como proporcionar información ordenada y comparable a otras fuentes

La investigación descriptiva demostró ser adecuada para evaluar herramientas de análisis de código seguro ya que facilita la catalogación y comparación de las características y capacidades de distintas herramientas. Este enfoque proporcionó una imagen clara y detallada de lo que ofrecen estas herramientas y cómo se utilizan en diferentes contextos.

### Investigación exploratoria

La investigación exploratoria consiste en proveer una referencia general de la temática, a menudo desconocida, presente en la investigación a realizar. Entre sus propósitos podemos citar la posibilidad de formular el problema de investigación, para extraer datos y términos que nos permitan generar las preguntas necesarias. Asimismo, proporciona la formulación de hipótesis sobre el tema a explorar, sirviendo de apoyo a la investigación descriptiva (Morales, 2015).

Este tipo de investigación fue valiosa para la evaluación de herramientas de análisis de código seguro, ya que permite descubrir nuevas vulnerabilidades y tendencias

en el ámbito de la seguridad. Además, ayudó a identificar los retos y necesidades específicos que enfrentan los desarrolladores al utilizar estas herramientas, sirviendo como base para investigaciones más exhaustivas.

## 3.2. Enfoque de la investigación

### Enfoque cualitativo

El enfoque cualitativo permite la investigación sistemática de fenómenos. A diferencia del enfoque cuantitativo, que parte de una teoría y luego la contrasta con la realidad empírica, la investigación cualitativa comienza con la observación de los hechos y la revisión concurrente de estudios anteriores, con el objetivo de elaborar una teoría que sea coherente con lo observado. (Hernández Sampieri & Mendoza Torres, 2018).

Del mismo modo, Hernández Sampieri & Mendoza Torres afirman que en esta forma de investigación las preguntas pueden surgir antes, durante o después de la recogida y el análisis de los datos. La indagación evoluciona dinámicamente, moviéndose entre los hechos y su interpretación en un proceso circular en el que la secuencia metodológica no es rígida y puede variar entre estudios.

El presente estudio utilizó un enfoque cualitativo para analizar las herramientas de análisis de código seguro seleccionadas, lo cual permitió explorar a fondo las capacidades, limitaciones y desafíos relacionados con el uso de estas herramientas en proyectos de software.

### Enfoque cuantitativo

Es un proceso analítico en el que el todo se divide en sus elementos básicos; es decir, de lo general a lo específico; también puede percibirse como un camino que va del fenómeno a la ley, es decir, del efecto a la causa. Este enfoque se caracteriza por las predicciones la cual son interpretaciones de los hechos observados; por lo tanto, su dirección es predecible. (Faneite, 2023)

Aunque el estudio parte de una exploración cualitativa de los puntos de vista y los obstáculos que plantea el empleo de herramientas de seguridad, se requiere un

método cuantitativo para evaluar objetivamente el rendimiento de las herramientas. Esto justifica su uso, ya que proporciona datos precisos y mensurables para complementar el análisis cualitativo.

## 3.3. Métodos de Investigación

### Método experimental

"Un experimento es una investigación en la cual el investigador manipula y controla una o más variables independientes y observa la o las variables dependientes para medir las variaciones concomitantes" (Agudelo Viana & Aigneren Aburto, 2008).

El estudio adoptó el método experimental ya que la variable independiente se manipuló intencionadamente para observar y medir su impacto. Las pruebas se realizaron en entornos controlados, aplicando cada herramienta a distintos proyectos software para evaluar su desempeño en condiciones reproducibles, alineados con los principios del método experimental.

### Método comparativo

"La metodología comparativa es el procedimiento de la comparación sistemática de casos de análisis que en su mayoría se aplica con fines de generalización empírica y de la verificación de hipótesis" (Sánchez Molina & Murillo Garza, 2022).

El método comparativo resultó adecuado para este estudio, ya que permitió evaluar y contrastar tecnologías de análisis de código seguro basándose en criterios clave como la precisión y el recall.

## 3.4. Metodología General para la Evaluación

El análisis de rendimiento de las herramientas de análisis de código seguro se centró en dos métricas clave: Precisión y Recall utilizando la metodología ROC (Receiver Operating Characteristic). Estas métricas fueron relevantes para evaluar la capacidad de las herramientas en la detección de vulnerabilidades dentro de proyectos de desarrollo web (Fawcett, 2006).

Para poder llevar a cabo este análisis se tomará en consideración los siguientes conceptos clave:

- Verdaderos positivos (TP): Vulnerabilidades detectadas correctamente por la herramienta.
- Falsos positivos (FP): Alertas incorrectas sobre vulnerabilidades inexistentes.
- Falsos negativos (FN): Vulnerabilidades reales que la herramienta no detecta.
- Verdaderos negativos (TN): Ausencia de vulnerabilidades correctamente identificadas.

A partir de estas métricas, se calcularán las siguientes dos métricas clave (Ward Powers, 2008):

- **Precisión**: Proporción de resultados relevantes entre todos los resultados positivos reportados por la herramienta. $Precision = \frac{TP}{TP+FP}$
- **Recall**: Proporción de vulnerabilidades reales detectadas entre todas las vulnerabilidades existentes. $Recall = \frac{TP}{P}$

  Teniendo en cuenta que el valor de P es $P = TP + FN$.

Este enfoque permitió comparar la efectividad de las herramientas en términos de precisión en la detección y la capacidad de cobertura de vulnerabilidades, lo que facilitará una interpretación clara y directa de los resultados.

Asegurando que las diferencias en la forma de reportar los resultados no afecten la capacidad de análisis, proporcionando una base uniforme para evaluar el rendimiento de cada herramienta en la detección de vulnerabilidades de seguridad.

## 3.5. Técnicas e Instrumentos de Recopilación de Datos

### Medición Directa en Escenarios Controlados

La medición directa en escenarios controlados implicó ejecutar las herramientas de análisis de código seguro en entornos de prueba diseñados específicamente para evaluar su rendimiento y precisión, al utilizar conjuntos de código con

vulnerabilidades conocidas, se puede determinar la capacidad de las herramientas para detectar fallos de seguridad.

Esta técnica proporcionó datos empíricos sobre el tiempo de análisis, la cantidad de falsos positivos y negativos, y la eficacia general de las herramientas en condiciones controladas que simulan situaciones reales de desarrollo.

**Monitoreo del Uso de Recursos del Sistema**

El monitoreo del uso de recursos del sistema durante el análisis de código fue crucial para evaluar la eficiencia de las herramientas de análisis de código seguro.

Al observar y registrar el consumo de CPU, memoria y almacenamiento mientras las herramientas SAST realizan sus análisis, es posible identificar el impacto en el rendimiento del sistema. Esta información fue vital para determinar si una herramienta es adecuada para integrarse en entornos de desarrollo intensivos, donde el uso eficiente de recursos es esencial para mantener la productividad y estabilidad del sistema.

**Análisis Documental**

La revisión exhaustiva de documentación técnica, informes de seguridad, manuales de usuario y estudios previos relacionados con las herramientas de análisis de código seguro proporcionó una comprensión detallada de sus funcionalidades, limitaciones y metodologías subyacentes.

Al ampliar el análisis para incluir revisiones de investigaciones anteriores, informes de vulnerabilidad y brechas de seguridad relevantes, así como bases de conocimiento y foros de usuarios, se obtuvo una visión completa y actualizada del panorama de las herramientas de seguridad de código.

**Observación Directa**

La observación directa de prácticas de desarrollo de software y el uso de herramientas de análisis de código seguro en entornos reales proporciono información contextual invaluable sobre su implementación y efectividad.

Al realizar observaciones en tiempo de entornos de desarrollo y sus interacciones con las herramientas, se pudo identificar patrones de uso, desafíos comunes y áreas de mejora, además, la observación participante, donde el investigador se involucra activamente en el proceso de desarrollo, permite una comprensión más profunda de las dinámicas y decisiones detrás del uso de estas herramientas. Integrar la observación en el estudio brinda una perspectiva práctica y enriquecedora que complementa los datos recopilados a través de otros métodos.

## 3.6. Procesamiento de la Información

### Organización y limpieza de datos

Los datos obtenidos de las pruebas prácticas fueron organizados y limpiados para eliminar duplicados e inconsistencias, incluyendo la recopilación y estructuración de los resultados generados por las herramientas, así como el registro de monitoreo del sistema durante las pruebas.

### Análisis descriptivo

Se evaluó las características principales de los datos recopilados:

- Número de vulnerabilidades detectadas por cada herramienta
- El tiempo promedio que tomó cada herramienta para completar los análisis.
- Número de falsos positivos y negativos reportados.

### Análisis comparativo

Los resultados obtenidos fueron comparados entre las herramientas evaluadas, permitiendo que el análisis identifique las fortalezas y debilidades, considerando aspectos como la precisión de la detección de vulnerabilidades, la eficiencia en términos de tiempo y recursos utilizados, y la facilidad de uso.

# CAPÍTULO IV

# RESULTADOS Y DISCUSIÓN

## 4.1. Criterios de selección

### Selección de Herramientas

Para una selección adecuada de herramientas SAST en esta investigación, se tomó como referencia la "Metodología de benchmark de herramientas SAST" (De Vega Martín, 2024, p.26). Esta metodología proporciona una tabla que compara diversas herramientas SAST disponibles en el mercado, tanto libres como de pago.

La siguiente imagen se utilizó para seleccionar las herramientas libres que se analizarán en profundidad en esta investigación, asegurando una comparación objetiva basada en características y funcionalidades relevantes para la detección de vulnerabilidades de seguridad.

### Tabla de las diferentes herramientas del SAST

La siguiente tabla muestra distintas herramientas de SAST donde las marcas X refieren a las tecnologías que admite cada herramienta.

**Tabla 2**

*Herramientas SAST*

| | SAST | | | | | |
|---|---|---|---|---|---|---|
| | **CodeQL** | **SonarQube** | **Semgrep** | **Fortify** | **Checkmarx** | **SpotBugs** |
| Soportye SARIF | X | X | X | X | X | X |
| Tipo de licencia | Privativa | LGPL (Software libre) | LGPL (Software libre) | Privativa | Privativa | LGPL (Software libre) |
| .NET | | | | X | X | |
| ABAP | | | | X | X | |
| Apex | | | | | X | |

| | | | | | | |
|---|---|---|---|---|---|---|
| Aplicaciones web | | | | | | |
| C# | X | X | | X | | |
| C++ | X | | X | X | | |
| Cobol | | | | X | X | |
| CSS | | X | | | | |
| Docker | | | X | | | |
| Go | X | | X | X | X | |
| Groovy | | | | X | X | |
| HTML | | X | X | | X | |
| Java | X | X | X | X | X | X |
| JavaScript | X | X | X | X | X | |
| JSP | | | | | X | |
| Kotlin | | X | X | X | X | |
| Objective-C | | | | X | X | |
| PHP | | X | X | X | X | |
| PL/SQL | | | | X | X | |
| Python | X | X | X | X | X | |
| Ruby | | X | X | X | X | |
| Rust | | | | | | |
| Scala | | | | X | X | |
| Swift | | X | X | X | X | |
| Terraform | | | | | | |
| T-SQL | | | | X | X | |
| TypeScript | X | X | X | X | X | |
| VB6 | | | | X | X | |
| XML | | X | | | X | |
| YMAL | | | X | | | |

*Nota*. La tabla presenta de forma reducida las herramientas SAST y los diferentes lenguajes que soportan.

*Fuente*. Adaptado del trabajo de De Vega Martín (2024) *Metodología de benchmark de herramientas SAST*

## Criterios de Selección de Herramientas de Análisis de Seguridad

Para garantizar que las herramientas de análisis de seguridad seleccionadas sean efectivas y adecuadas para los propósitos del proyecto de investigación, es crucial considerar varios criterios, los mismos aseguran que las herramientas proporcionen una cobertura completa y se integren eficientemente en el flujo de trabajo de desarrollo.

Los criterios de selección de las herramientas se presentan en la siguiente tabla:

**Tabla 3**

*Criterios de Selección de Herramientas de Análisis de Seguridad*

| Criterio | Descripción |
|---|---|
| **Cobertura de vulnerabilidades** | La capacidad de la herramienta para detectar una amplia gama de vulnerabilidades conocidas. |
| **Facilidad de uso** | La simplicidad y claridad de la interfaz de usuario y la facilidad de integración en el flujo de trabajo del desarrollo. |
| **Compatibilidad** | Capacidad de la herramienta para integrarse con otras herramientas y plataformas de desarrollo ya existentes. |
| **Tipo de licencia** | Todas las herramientas con las que se realizó el trabajo investigativo son de código abierto. Open Souce |

*Nota*. En la tabla se presenta información en la que se basó la selección de las herramientas para el estudio.

*Fuente.* Elaboración propia.

Para tener un contexto claro de la selección de las herramientas se realizó una tabla que muestra la información de la herramienta con respecto a cada criterio

**Tabla 4**

*Descripción de los criterios de selección con cada herramienta.*

| Herramienta | Cobertura de Vulnerabilidades | Facilidad de Uso | Compatibilidad | Licencia |
|---|---|---|---|---|
| **SonarQube** | Alta cobertura de vulnerabilidades conocidas, incluyendo bugs y code smells. | Interfaz de usuario intuitiva, fácil integración con CI/CD y diversos IDEs. | Amplia compatibilidad con múltiples lenguajes de programación y herramientas de desarrollo. | Open Souce |
| **Semgrep** | Detecta una amplia gama de vulnerabilidades | Fácil de usar con reglas personalizables, sin necesidad de | Compatible con varios lenguajes de programación y fácil | Open Souce |

| | | | | |
|---|---|---|---|---|
| | específicas y contextuales. | configuraciones complejas. | integración en flujos de trabajo existentes. | |
| **OWASP ASST** | Alta cobertura de vulnerabilidades en aplicaciones web. | Instalación y configuración fácil, presentación clara de resultados. | Compatible con múltiples herramientas de desarrollo y CI/CD. | Open Souce |
| **HCL AppScan CodeSweep** | Detección de una amplia gama de vulnerabilidades en varios lenguajes. | Al ser una extensión de Visual Studio Code, esta herramienta se integra de manera adecuada en el entorno de desarrollo, proporcionando un alto nivel de usabilidad. | La herramienta es compatible con una amplia variedad de lenguajes de programación, facilitando su integración a entornos de desarrollo. | Open Souce |

*Nota*. Descripción de los criterios seleccionados para el estudio para cada herramienta

*Fuente*. Elaboración propia. Tomado de la documentación oficial de cada herramienta (HCL Software, 2024; Semgrep.dev, 2024; Seyceri, 2024; SonarQube, 2024)

## 4.2. Software para el análisis

El software seleccionado para la investigación consta de dos aplicaciones web funcionales que simulan una tienda en línea.

Estos proyectos intentan cubrir casos de uso comunes en aplicaciones web comerciales, proporcionando un entorno adecuado para identificar y analizar posibles vulnerabilidades de seguridad.

El primer sistema incluye las siguientes características principales:

- Inicio de sesión y registro de usuarios.

- Gestión de un carrito de compras.
- Un módulo de administración para la gestión de productos y pedidos.
- Perfiles de usuario.

En cuanto a las tecnologías empleadas, el backend fue desarrollado en PHP, mientras que el frontend utiliza HTML, CSS y JavaScript para la interfaz de usuario. El almacenamiento de datos se gestiona a través de una base de datos MySQL.

Este proyecto fue diseñado con un enfoque realista, incorporando funcionalidades básicas y esenciales, lo que lo hace ideal para someterlo a pruebas de análisis estático (SAST).

Adicional al primer proyecto, para complementar el análisis se obtuvo un proyecto público disponible en GitHub que también corresponde a una tienda en línea. Este proyecto, desarrollado con tecnologías similares (PHP, HTML, CSS, JavaScript y MySQL), incorpora un conjunto distinto de funcionalidades y estilos de codificación.

Cachirulo´s: https://github.com/Je-Real/CACHIRULOS-web-store,

La evaluación de este software busca explorar cómo las herramientas de análisis de código seguro se desempeñan al enfrentarse a código externo y desconocido, ampliando el alcance de la investigación.

### 4.3. Análisis, Interpretación y Discusión de Resultados

Se abordan los resultados obtenidos a partir de la evaluación de las herramientas de análisis de código seguro empleadas en el proyecto de investigación.

El enfoque de este apartado es desglosar y examinar en detalle los datos recolectados durante el proceso de evaluación, para proporcionar un contexto claro y comprensible sobre los resultados presentados, se comenzará con una introducción a las herramientas utilizadas: SonarQube, Semgrep, OWASP ASST y HCL AppSacan CodeSweep.

Esta introducción permitirá una mejor comprensión de los criterios y metodologías aplicados en el análisis, facilitando así una discusión más informada y relevante sobre los hallazgos y su impacto en el ámbito del desarrollo de software seguro.

## 4.4. Aplicación de las Herramientas de Análisis

Para la evaluación de las herramientas de análisis de seguridad, se aplicaron a un conjunto de software ya desarrollado, con el objetivo de analizar cómo cada herramienta se desempeña en contextos reales y existentes. A continuación, se detalla el proceso de utilización de cada herramienta en el análisis del software seleccionado.

Para el desarrollo de la investigación en la obtención de resultados mediante pruebas de cada herramienta se ha empleado un equipo informático (laptop) de uso doméstico y que presenta especificaciones moderadas para los estándares habituales para este tipo de actividades.

Es importante detallar el dispositivo empleado, ya que sus características de hardware representaron una limitación durante el desarrollo de la investigación.

MSI Katana GF76 11UD con las siguientes características:

- Procesador: Intel Core i7-11800H (8 núcleos físicos / 16 procesadores lógicos)
- RAM: 16 GB (DDR4)
- Almacenamiento:
    - 477 GB KINGSTON NVMe
- Sistemas Operativos:
    - Windows 11 Home (x64)

### 4.4.1. SonarQube

Esta herramienta se utilizó para realizar análisis estáticos en el código fuente de las aplicaciones seleccionadas. La herramienta se integró en el entorno de desarrollo del software para realizar revisiones continuas del código.

Para llevar a cabo el proceso, se inicia con la revisión de los requisitos necesarios del sistema para utilizar esta herramienta, los mismos que se encuentran en su página oficial, la cual presenta una tabla que muestra los requerimientos de hardware y software.

**Requisitos del sistema:**

**Hardware**

**Tabla 5**

*Requisitos hardware para SonarQube*

| Categoría | Requisitos |
|---|---|
| RAM | Instalación a pequeña escala:<br>• 4 GB de RAM<br>Instalación a gran escala:<br>• 16 GB de RAM<br>Sistema de 64 bits |
| Procesador | Instalación a pequeña escala:<br>• 2 núcleos<br>Instalación a gran escala:<br>• 8 núcleos<br>Una instalación de servidor desde una imagen de Docker:<br>• Arquitectura amd64 o Apple Silicon basada en arm64 |
| Espacio en el disco | Depende de la cantidad de código que analices con SonarQube.<br>Para una instalación a pequeña escala:<br>• 30 GB |

*Nota.* Especificaciones técnicas para el correcto funcionamiento de la herramienta SonarQube

*Fuente. T*oma de la documentación oficiala de SonarQube, (SonarQube, 2024)

**Software**

**Tabla 6**

*Requisitos Software para SonarQube*

| Categoría | Requisitos |
|---|---|

| | |
|---|---|
| Navegador web | • Microsoft Edge: última versión<br>• Mozilla Firefox: última versión<br>• Google Chrome: última versión<br>• Safari: última versión |
| Java | Se aplica únicamente a una instalación de servidor desde el archivo ZIP.<br>• Oracle JRE o OpenJDK<br>• Versión 17 de Java<br>• Recomendación: utilice versiones de CPU de Java (actualización de parches críticos).<br>Nota: SonarQube puede analizar cualquier tipo de archivos fuente de Java independientemente de la versión de Java con la que cumplan. |

*Nota*. Especificaciones técnicas para el funcionamiento de la herramienta SonarQube

*Fuente*. Toma de la documentación oficiala de SonarQube, (SonarQube, 2024)

La documentación de SonarQube menciona los requisitos mínimos para que el sistema se ejecute teniendo en cuenta lo siguiente:

- Una instalación a pequeña escala suele ser una instalación de Community Edition o Developer Edition que admite hasta 1 millón de líneas de código.
- Una instalación a gran escala suele ser una instalación de un solo nodo de Enterprise Edition que admite hasta 50 millones de líneas de código.

Para el presente trabajo se ha optado por realizar la instalación de Community Edition, ya que cumple con los criterios de selección mencionados anteriormente.

**Configuración inicial:**

Para realizar la instalación de SonarQube Community Edition es necesario descargarlo de su página oficial https://www.sonarsource.com/products/sonarqube/

Adicional a eso es necesario descargar e instalara SonarScanner una herramienta que posibilita el escaneo del código, para ello hay revisar la documentación oficial de SonarQube https://docs.sonarsource.com/sonarqube-server/latest/

Se configuraron perfiles de análisis para evaluar distintos aspectos del código, incluyendo vulnerabilidades, errores y malas prácticas. Los resultados proporcionaron un informe detallado sobre la calidad del código, permitiendo identificar y corregir problemas antes de que el software se despliegue.

**Proyecto 1**

## A. Consumo de recursos

**Tabla 7**

*Recursos utilizados por SonarQube en el análisis del Proyecto 1*

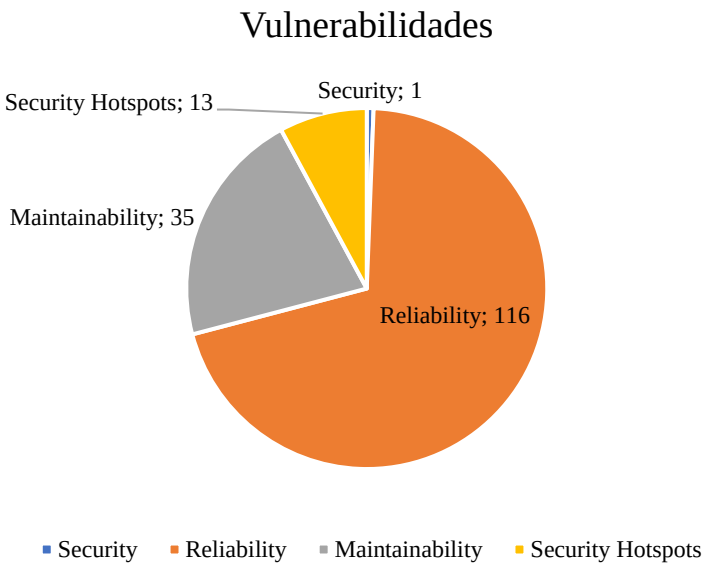
| Rendimiento | |
|---|---|
| **Consumo de recursos** | |
| **Tiempo de análisis** | 36.880 segundos |
| **CPU** | 0,51% |
| **RAM** | 33,5 MB |

*Nota*. La tabla presenta la cantidad de recursos y tiempo de análisis que utilizó SonarQube en la prueba para el "Proyecto 1". *Fuente.* Elaboración propia.

## B. Resultado del análisis "Proyecto 1"

**Figura 1**

*Resultado del Análisis del Proyecto 1 con SonarQube*




*Nota*. Representación gráfica de las categorías que la herramienta SonarQube presenta en tras el análisis del "Proyecto 1". Elaboración propia.

**Vulnerabilidades en el código**

Total, de problemas en el código: 165

**Tabla 8**

*Análisis de los resultados del proyecto 1 con SonarQube*

| Vulnerabilidades | | Métricas de Fiabilidad | |
|---|---|---|---|
| **FP** | 151 | **Precisión** | 0.084848 |
| **TP** | 14 | **Recall** | 0.102190 |
| **FN** | 123 | | |

*Nota*. La tabla 8 presenta las vulnerabilidades del código y las métricas de fiabilidad del análisis de SonarQube en el Proyecto 1

*Fuente.* Elaboración propia.

Tras el análisis de los resultados obtenidas del "Proyecto 1" obtenemos los siguientes resultados en porcentajes:

Vulnerabilidades reales (precisión): 8.4848%

Vulnerabilidades presentes (recall): 10.2190%

**Proyecto 2**

**A. Consumo de recursos**

**Tabla 9**

*Recursos utilizados por SonarQube en el análisis del Proyecto 2*

| Rendimiento | |
|---|---|
| **Consumo de recursos** | |
| **Tiempo de análisis** | 26,405 segundos |
| **CPU** | 1,20% |
| **RAM** | 33,25 MB |

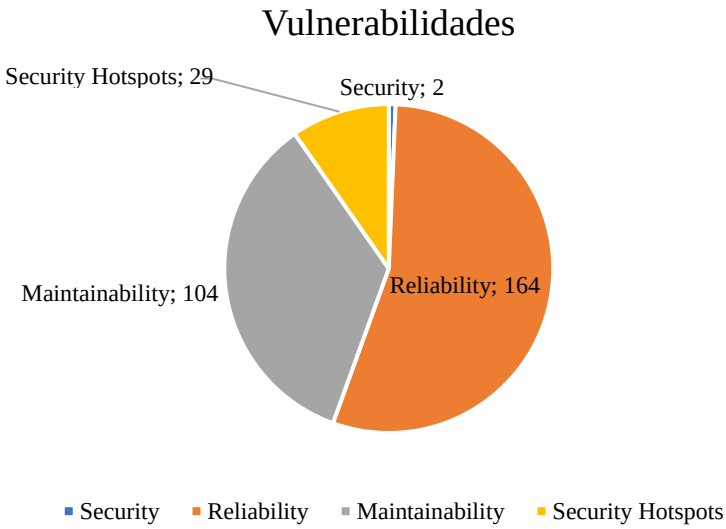
*Nota*. En la tabla se muestra la cantidad de recursos y tiempo de análisis que utilizo SonarQube en la prueba para el "Proyecto 2".

*Fuente.* Elaboración propia

## B. Resultado del análisis “Proyecto 2”

**Figura 2**

*Resultado del Análisis del Proyecto 2 con SonarQube*




*Nota*. Representación gráfica de las categorías que la herramienta SonarQube presenta en tras el análisis del “Proyecto 2”. Elaboración propia.

### Vulnerabilidades en el código

Total, de problemas en el código: 299

**Tabla 10**

*Análisis de los resultados del proyecto 2 con SonarQube*

| Vulnerabilidades | | Métricas de Fiabilidad | |
|---|---|---|---|
| **FP** | 268 | **Precisión** | 0.103679 |
| **TP** | 31 | **Recall** | 0.197452 |
| **FN** | 126 | | |

*Nota*. La tabla 10 presenta las vulnerabilidades del código y las métricas de fiabilidad del análisis de SonarQube en el Proyecto 2

*Fuente*. Elaboración propia.

Tras el análisis de los resultados obtenidas en la “Proyecto 2” obtenemos los siguientes resultados en porcentajes:

Vulnerabilidades reales (Precisión): 10.3679%

Vulnerabilidades presentes (recall): 19.7452%

### 4.4.2. Semgrep

En esta sección se realizaron pruebas de análisis de seguridad utilizando Semgrep, una herramienta estática de análisis de código (SAST) que permitió identificar vulnerabilidades y errores específicos en el código fuente.

El objetivo de estas pruebas fue evaluar la capacidad de Semgrep para detectar fallos de seguridad contextuales en el software, proporcionando resultados claros y precisos que facilitaron la identificación y corrección de las debilidades presentes en las aplicaciones evaluadas.

Para utilizar Semgrep, se deben considerar los siguientes requisitos en cuanto a hardware y software, que permiten un análisis eficiente del código fuente.

**Requisitos del sistema:**

**Hardware**

Para el estudio se utilizará la versión Semgrep Community Edition (CE), debido a que la herramienta trabaja en el navegador y su documentación oficial solo menciona las características software necesario, se toma como requisitos hardware necesario los especificados para cualquier navegador.

**Tabla 11**

*Requisitos hardware para Semgrep*

| Categoría | Requisitos |
|---|---|
| RAM | Recomendado: 4 GB o más |
| Procesador | AMD Athlon 64 o posterior compatible con SSE2<br>Procesador Intel Pentium 4 o posterior compatible con SSE3 |
| Espacio en el disco | Depende de la cantidad de reglas y código analizado |

*Nota.* Los requisitos del hardware que especifican la tabla corresponden al navegador Opera.

*Fuente.* Toma de la página oficial de Opera, (Semgrep.dev, 2024)

**Software**

**Tabla 12**

*Requisitos Software para Semgrep*

| Categoría | Requisitos |
|---|---|
| Sistema Operativo | Windows, macOS, Linux |
| Python | Versión 3.6 o superior en el equipo en el que está ejecutando Semgrep. |
| Herramienta de línea de comandos. | |

*Nota.* Especificaciones Software recomendadas para el funcionamiento de Semgrep

*Fuente.* Tomado de la documentación oficial de Semgrep, (Semgrep.dev, 2024)

Semgrep es una herramienta de análisis estático de código fuente que se distingue por su simplicidad y enfoque directo en la detección de patrones específicos dentro del código.

A diferencia de otras herramientas más complejas, como SonarQube, Semgrep no requiere una configuración extensa ni un análisis profundo del proyecto para comenzar a operar.

Su funcionamiento se basa en la búsqueda y coincidencia de patrones en el código fuente, lo que permite identificar vulnerabilidades y malas prácticas de manera eficiente y rápida. Dado su diseño enfocado en la agilidad y la personalización, este análisis no incluye una contextualización extensa de la herramienta, sino que se concentra en ofrecer soluciones rápidas y configurables a problemas específicos de seguridad y calidad del código.

**Proyecto 1**

**A. Consumo de recursos**

**Tabla 13**

*Recursos utilizados por Semgrep en el análisis del Proyecto 1*

| Rendimiento |
|---|
| **Consumo de recursos** |

| | |
|---|---|
| **Tiempo de análisis** | 2:00 minutos |
| **CPU** | 1,36% |
| **RAM** | 285.36 MB |

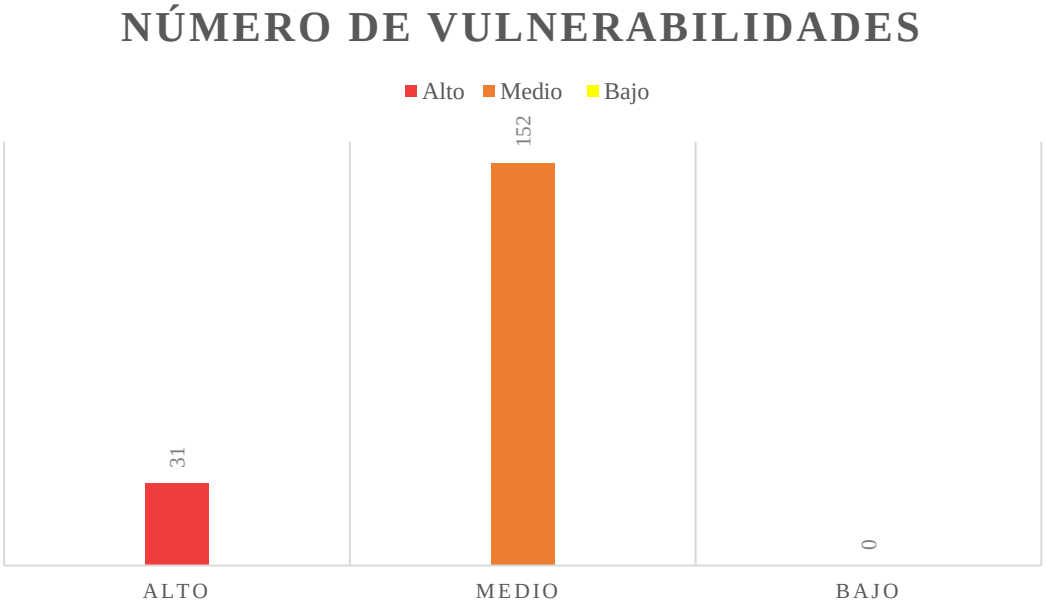
*Nota*. Esta tabla presenta los recursos y tiempo necesarios para completar el análisis de en el "Proyecto 1" con Semgrep.

*Fuente*. Elaboración propia,

## B. Resultado del análisis "Proyecto 2"

**Figura 3**

*Resultado del Análisis del Proyecto 1 con Semgrep*




*Nota*. Representación gráfica de los resultados del análisis realizado por Semgrep al "Proyecto 1". Elaboración propia.

### Vulnerabilidades en el código

Total, de problemas en el código: 183

**Tabla 14**

*Análisis de los resultados Análisis de los resultados del proyecto 1 con Semgrep*

| **Vulnerabilidades** | | **Métricas de Fiabilidad** | |
|---|---|---|---|
| FP | 31 | Precisión | 0,830601 |
| TP | 152 | Recall | 0,655172 |
| FN | 80 | | |

*Nota*. La tabla 14 presenta las vulnerabilidades del código y las métricas de fiabilidad del análisis de Semgrep en el Proyecto 1. *Fuente.* Elaboración propia.

Tras el análisis de los resultados obtenidas en la “Proyecto 1” Con Semgrep obtenemos los siguientes resultados en porcentajes:

Vulnerabilidades reales (precisión): 83.0601%

Vulnerabilidades presentes (recall): 65.5172%

**Proyecto 2**

**A. Consumo de recurso**

**Tabla 15**

*Recursos utilizados por Semgrep en el análisis del Proyecto 2*

| **Rendimiento** | |
|---|---|
| **Consumo de recursos** | |
| **Tiempo de análisis** | 3:09 minutos |
| **CPU** | 1,30% |
| **RAM** | 256 MB |

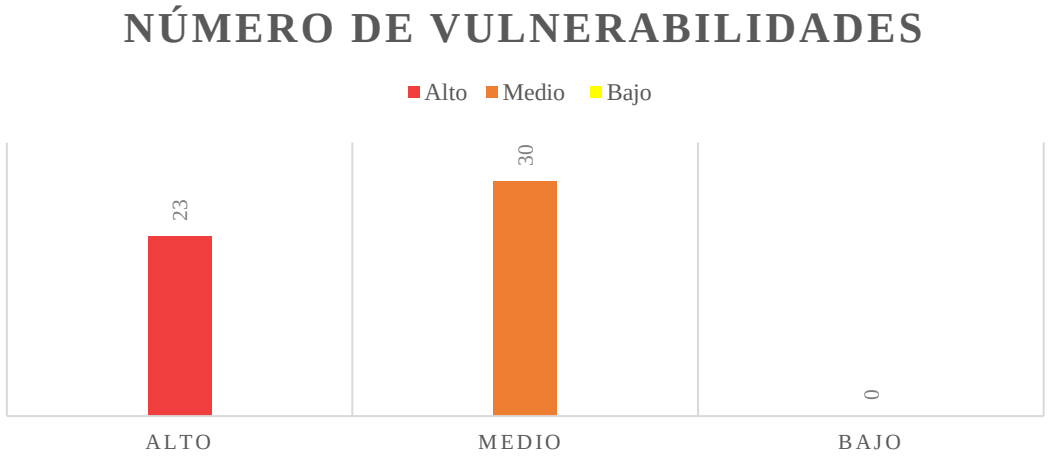
*Nota*. Esta tabla presenta los recursos y tiempo necesarios para completar el análisis de en el “Proyecto 2” con Semgrep.

*Fuente.* Elaboración propia.

**B. Resultado del análisis “Proyecto 2”**

**Figura 4**

*Resultado del Análisis del Proyecto 2 con Semgrep*




*Nota*. Representación gráfica de los resultados del análisis realizado por Semgrep al “Proyecto 2”. Elaboración propia.

**Vulnerabilidades en el código**

Total, de problemas en el código: 53

**Tabla 16**

*Análisis de los resultados del proyecto 2 con Semgrep*

| **Vulnerabilidades** | | **Métricas de Fiabilidad** | |
|---|---|---|---|
| FP | 14 | Precisión | 0.730769 |
| TP | 39 | Recall | 0.275362 |
| FN | 100 | | |

*Nota*. La tabla 16 presenta las vulnerabilidades del código y las métricas de fiabilidad del análisis de Semgrep en el Proyecto 2.

*Fuente*. Elaboración propia.

Tras el análisis de los resultados obtenidas en la "Proyecto 2" Con Semgrep obtenemos los siguientes resultados en porcentajes:

Vulnerabilidades reales (precisión): 73.0769%

Vulnerabilidades presentes (recall): 27.5362%

### 4.4.3. OWASP ASST

OWASP ASST se utilizó para llevar a cabo análisis estáticos de las aplicaciones web ya desarrolladas.

Se configuró para realizar escaneos de seguridad en el código fuente, identificando vulnerabilidades antes de que las aplicaciones estuvieran en producción. OWASP ASST proporcionó un análisis exhaustivo de la seguridad del código, destacando áreas críticas que requerían atención y corrección.

Para utilizar OWASP ASST de manera efectiva, es importante considerar ciertos requisitos, los cuales aseguran que la herramienta pueda operar de forma fluida y eficiente durante las pruebas de seguridad.

**Hardware**

El hardware necesario para OWASP ASST debe cumplir con los requisitos mínimos del sistema operativo en el que se instalará (Windows, macOS o Linux), así como con los del software necesario.

**Software**

**Tabla 17**

*Requisitos hardware para OWASP ASST*

| Categoría | Requisitos |
|---|---|
| Sistema Operativo | Compatible con: Windows, macOS , Linux |
| Requisitos adicionales | • localhost en su PC, le recomendamos (XAMPP).<br>• Motor Node.js v12.13.0 |
| Nota: El mejor uso de ASST es ejecutarlo directamente en el servidor de producción en línea y escanear los proyectos que contiene, ya que ASST también verifica las versiones de PHP y MySQL | |

*Nota.* Especificaciones técnicas para el correcto funcionamiento de la herramienta OWASP ASST.
*Fuente.* Tomado de la documentación digital de la herramienta OWASP, (Seyceri, 2024)

**Proyecto 1**

**A. Consumo de recursos**

**Tabla 18**

*Recursos utilizados por OWASP ASST en el análisis del Proyecto 1*

| **Rendimiento** | |
|---|---|
| **Consumo de recursos** | |
| **Tiempo de análisis** | 5 segundos |
| **CPU** | 0,1% |
| **RAM** | 38.6 MB |

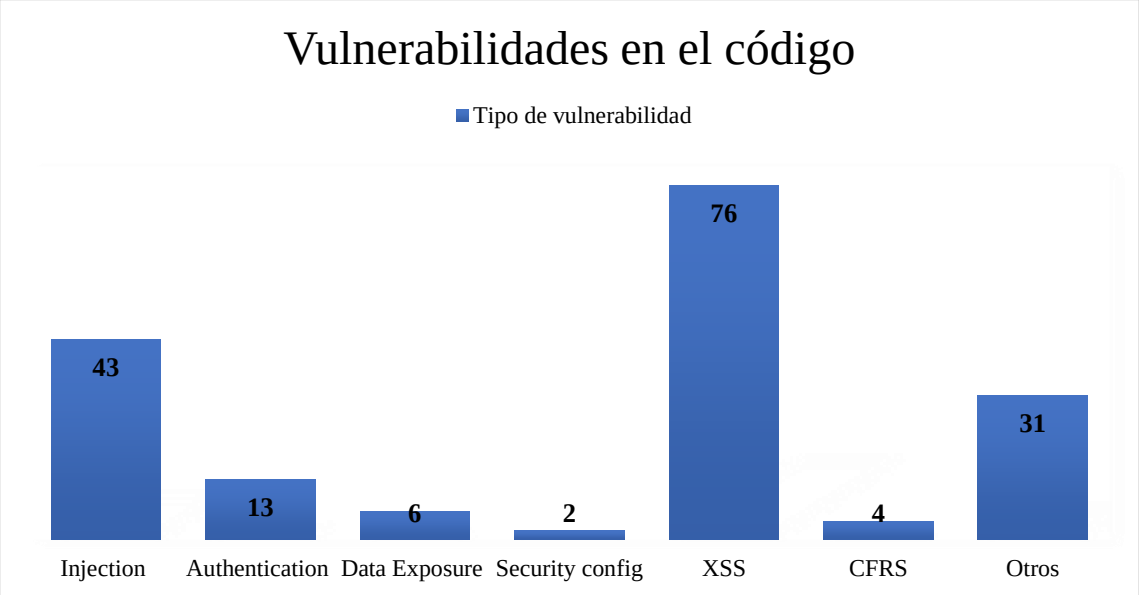
*Nota. R*ecursos y tiempo necesarios para completar el análisis de en el "Proyecto 1" con OWASP ASST.

*Fuente.* Elaboración propia. Vulnerabilidades en el código

## B. Resultado del análisis “Proyecto 1”

**Figura 5**

*Resultado del Análisis del Proyecto 1 con OWASP ASST*




*Nota:* Representación gráfica de los tipos de vulnerabilidades que muestra OWASP ASST después del análisis del Proyecto 1. Elaboración propia.

**Vulnerabilidades en el código**

Total, de problemas en el código: 175

**Tabla 19**

*Análisis de los resultados del proyecto 1 con OWASP ASST*

| **Vulnerabilidades** | | **Métricas de Fiabilidad** | |
|---|---|---|---|
| FP | 56 | Precisión | 0,68 |
| TP | 119 | Recall | 0,639784 |
| FN | 67 | | |

*Nota*. La tabla 19 presenta las vulnerabilidades del código y las métricas de fiabilidad del análisis

*Fuente.* Elaboración propia

Tras el análisis de los resultados obtenidas en la “Proyecto 1” Con OWASP ASST obtenemos los siguientes resultados en porcentajes:

Vulnerabilidades reales (precisión): 68%

Vulnerabilidades presentes (recall): 63.9784%

**Proyecto 2**

## A. Consumo de recursos

**Tabla 20**

*Recursos utilizados por OWASP ASST en el análisis del Proyecto 2*

| Rendimiento | |
|---|---|
| **Consumo de recursos** | |
| **Tiempo de análisis** | 6 segundos |
| **CPU** | 0,2% |
| **RAM** | 48.1 MB |

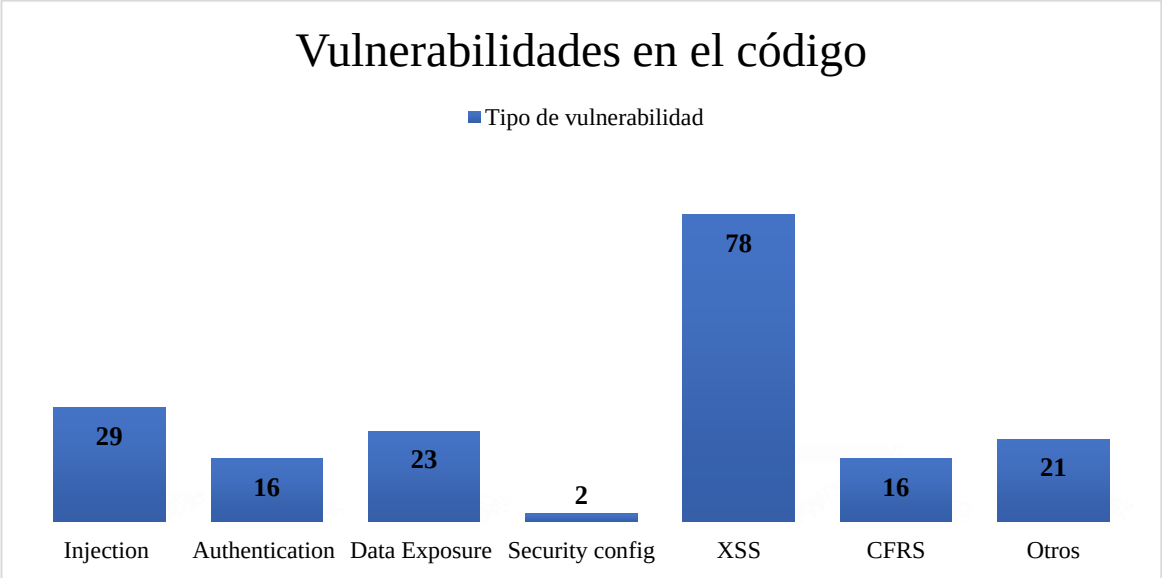
*Nota*. Recursos y tiempo necesarios para completar el análisis de en el "Proyecto 1" con OWASP ASST.

*Fuente*. Elaboración propia

## B. Resultado del análisis "Proyecto 2"

**Figura 6**

*Resultado del Análisis del Proyecto 2 con OWASP ASST*




*Nota:* Representación gráfica de los tipos de vulnerabilidades que muestra OWASP ASST después del análisis del Proyecto 2. Elaboración propia.

**Vulnerabilidades en el código**

Total, de problemas en el código: 185

**Tabla 21**

*Análisis de los resultados del proyecto 2 con OWASP ASST*

| **Vulnerabilidades** | | **Métricas de Fiabilidad** | |
|---|---|---|---|
| **FP** | 73 | Precisión | 0,60540541 |
| **TP** | 112 | Recall | 0,67878788 |
| **FN** | 53 | | |

*Nota*. La tabla 21 presenta las vulnerabilidades del código y las métricas de fiabilidad del análisis

*Fuente.* Elaboración propia

Tras el análisis de los resultados obtenidas en la "Proyecto 2" Con OWASP ASST obtenemos los siguientes resultados en porcentajes:

Vulnerabilidades reales (precisión): 60,5405%

Vulnerabilidades presentes (recall): 67,8787%

### 4.4.4. HCL AppScan CodeSweep

La extensión HCL AppScan para Visual Studio Code proporciona la funcionalidad de pruebas de seguridad de aplicaciones estáticas (SAST) para detectar vulnerabilidades en las primeras etapas del ciclo de vida del desarrollo. (HCL Software, 2024)

De acuerdo con HCL Software esta herramienta SAST permite analizar más de 30 tipos de archivos, dada la naturaleza de la investigación sólo se analizarán los siguientes tipos:

- PHP
- JavaScript
- CSS
- JQuery

Los requisitos mínimos para el correcto funcionamiento de esta herramienta es necesario tener en cuenta lo siguiente:

- Versión de VSCode (Visual Studio Code) 1.45.0 o superior

Dado que la herramienta es una extensión de VSCode hay que tener en consideración los requisitos de funcionamiento del editor de texto.

**Hardware**

**Tabla 22**

*Requisitos hardware para HCL AppScan CodeSweep*

| Categoría | Requisitos |
|---|---|
| RAM | Recomendado: 1 GB o más |
| Procesador | Procesador de 1,6 GHz o más rápido |
| Espacio en el disco | Visual Studio Code es una descarga pequeña (menos de 200 MB) y ocupa menos de 500 MB en disco. |

*Nota*. Especificaciones técnicas para el correcto funcionamiento de la herramienta HCL AppScan CodeSweep como extensión de VSCode

*Fuente*. Tomado de la documentación oficial de Visual Studio Code, (HCL Software, 2024)

**Software**

**Tabla 23**

*Requisitos software para HCL AppScan CodeSweep*

| Categoría | Requisitos |
|---|---|
| Sistema Operativo | Windows 10 y 11 (64 bits)<br>Versiones de macOS compatibles con actualizaciones de seguridad de Apple. Normalmente, se trata de la última versión y las dos versiones anteriores.<br>Linux (Debian): escritorio Ubuntu 20.04, Debian 10<br>Linux (Red Hat): Red Hat Enterprise Linux 8, Fedora 36 |
| Java | Tener instalado Java Runtime (JRE) 8 o superior. |

*Nota*. Requisitos Software necesarios para el funcionamiento de HCL AppScan CodeSweep

*Fuente*. Tomado de la documentación oficial de Visual Studio Code,(HCL Software, 2024)

**Proyecto 1**

## A. Consumo de recursos

**Tabla 24**

*Recursos utilizados por HCL AppScan CodeSweep en el análisis del Proyecto 1*

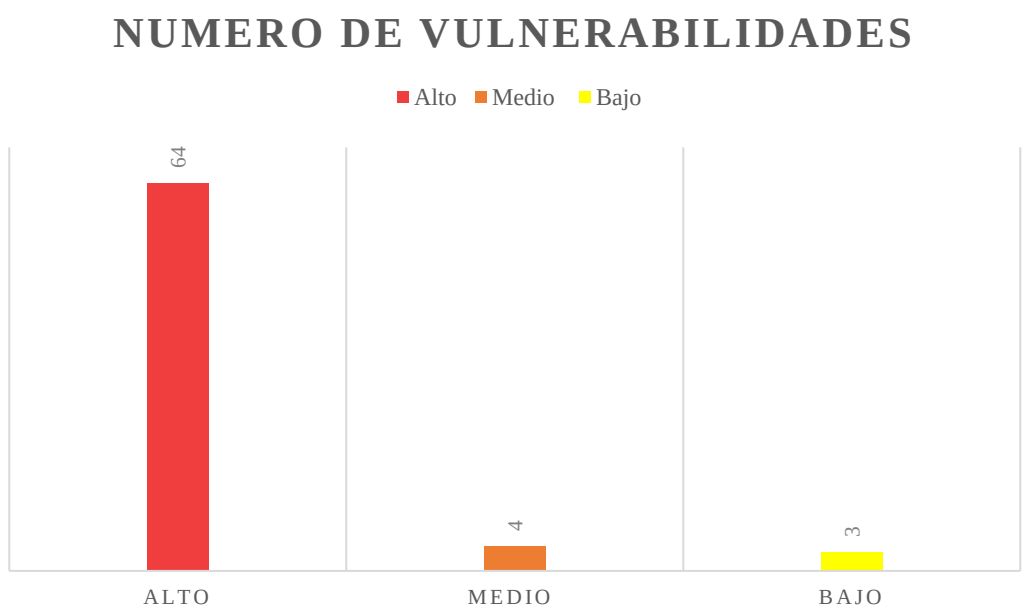
| Rendimiento | |
|---|---|
| **Consumo de recursos** | |
| **Tiempo de análisis** | 5 minutos |
| **CPU** | 0.19% |
| **RAM** | 48.6 MB |

*Nota*. La tabla presenta la cantidad de recursos y tiempo de análisis que utilizo HCL AppScan CodeSweep en el análisis del "Proyecto 1". *Fuente*. Elaboración propia.

## B. Resultado del análisis "Proyecto 1"

**Figura 7**

*Resultado del Análisis del Proyecto 1 HCL AppScan CodeSweep*




*Nota*. Representación gráfica de las categorías que la herramienta HCL AppScan CodeSweep presenta en tras el análisis del "Proyecto 1". Elaboración propia.

## Vulnerabilidades en el código

Total, de problemas en el código: 71

**Tabla 25**

*Análisis de los resultados del proyecto 1 con HCL AppScan CodeSweep*

| **Vulnerabilidades** | | **Métricas de Fiabilidad** | |
|---|---|---|---|
| FP | 32 | Precisión | 0.549296 |
| TP | 39 | Recall | 0.317073 |
| FN | 84 | | |

*Nota*. La tabla 25 presenta las vulnerabilidades del código y las métricas de fiabilidad del análisis de SonarQube en el Proyecto 1.

*Fuente.* Elaboración propia.

Tras el análisis de los resultados obtenidas en la "Proyecto 1" Con CL AppScan CodeSweep obtenemos los siguientes resultados en porcentajes:

Vulnerabilidades reales (precisión): 54.9296%

Vulnerabilidades presentes (recall): 31.7073%

**Proyecto 2**

**A. Consumo de recursos**

**Tabla 26**

*Recursos utilizados por HCL AppScan CodeSweep en el análisis del Proyecto 2*

| **Rendimiento** | |
|---|---|
| **Consumo de recursos** | |
| **Tiempo de análisis** | 8 minutos |
| **CPU** | 0.1% |
| **RAM** | 38.6 MB |

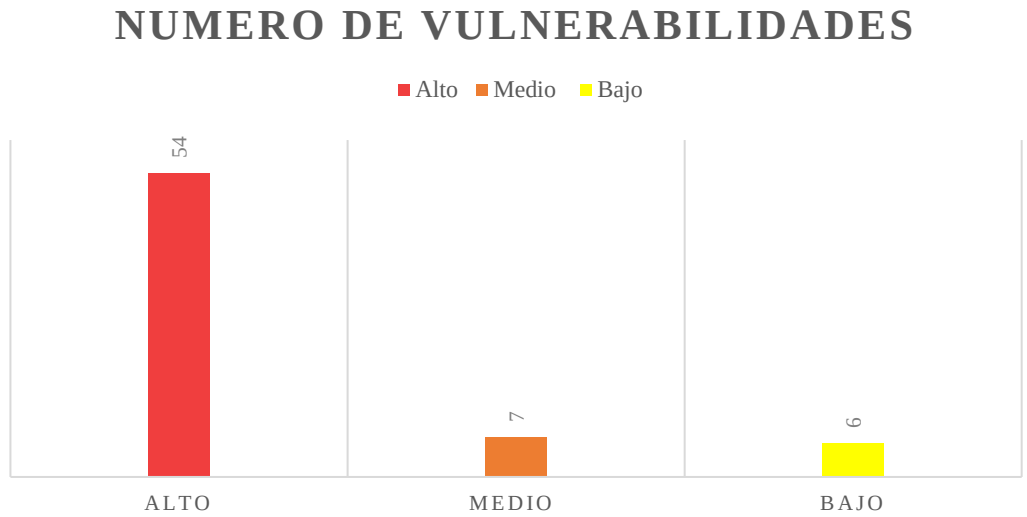
*Nota*. La tabla presenta la cantidad de recursos y tiempo de análisis que utilizo HCL AppScan CodeSweep en el análisis del "Proyecto 2".

*Fuente.* Elaboración propia.

## B. Resultado del análisis “Proyecto 2”

**Figura 8**

*Resultado del Análisis del Proyecto 2 HCL AppScan CodeSweep*




*Nota*. Representación gráfica de las categorías que la herramienta HCL AppScan CodeSweep presenta en tras el análisis del “Proyecto 1”. Elaboración propia.

### Vulnerabilidades del código

Total, de problemas en el código: 81

**Tabla 27**

*Análisis de los resultados del proyecto 2 con HCL AppScan CodeSweep*

| **Vulnerabilidades** | | **Métricas de Fiabilidad** | |
|---|---|---|---|
| FP | 51 | Precisión | 0.370370 |
| TP | 30 | Recall | 0.238095 |
| FN | 96 | | |

*Nota*. La tabla 27 presenta las vulnerabilidades del código y las métricas de fiabilidad del análisis de SonarQube en el Proyecto 2.

*Fuente*. Elaboración propia.

Tras el análisis de los resultados obtenidas en la “Proyecto 2” Con CL AppScan CodeSweep obtenemos los siguientes resultados en porcentajes:

Vulnerabilidades reales (precisión): 37.0370%

Vulnerabilidades presentes (recall): 23.8095%

## 4.5. Análisis comparativo

En este apartado se presentará un análisis comparativo de las herramientas utilizadas en el estudio: SonarQube, Semgrep, OWASP ASST

Para ello, se ha elaborado un cuadro comparativo en el que se evalúan métricas clave, tales como la precisión y el recall de cada herramienta con él con el objetivo de determinar su eficacia en la detección de vulnerabilidades en entornos de desarrollo web.

Este análisis proporcionó una visión integral de las fortalezas y debilidades de cada herramienta, ofreciendo una base sólida para la selección de la solución más adecuada para las necesidades específicas del desarrollo web.

A continuación, se presenta los cuadros comparativos de la fiabilidad de las herramientas utilizadas para el estudio por cada uno de los proyectos analizados:

**Presentación de los resultados en porcentajes:**

**Tabla 28**

*Análisis comparativo de las herramientas (%)*

| Herramienta | Proyecto 1 | | Proyecto 2 | |
|---|---|---|---|---|
| | **Precisión (%)** | **Recall (%)** | **Precisión (%)** | **Recall (%)** |
| **SonarQube** | 8,48% | 10,21% | 10,36% | 19,74% |
| **Semgrep** | 83.06% | 65.51% | 73.46% | 28.26% |
| **OWASP ASST** | 68% | 63.97% | 60.54% | 67.87% |
| **HCL AppScan** | 54.92% | 31.70% | 37.03% | 23.80% |

*Nota*. La tabla presenta una comparación entre las herramientas del estudio con respecto a la precisión y el recall.

*Fuente.* Elaboración propia.

### 4.5.1. Análisis de Precisión

**Proyecto 1**

En el análisis del Proyecto 1, la herramienta con mayor precisión fue Semgrep, demostrando que la mayoría de las vulnerabilidades identificadas son correctas, reduciendo significativamente los falsos positivos. Le sigue OWASP ASST,

mostrando una alta precisión aunque ligeramente inferior a Semgrep. HCL AppScan CodeSweep presenta una precisión aceptable, lo cual indica que, identifica correctamente un buen número de vulnerabilidades. Finalmente, SonarQube muestra una baja precisión, sugiriendo un alto número de falsos positivos.

**Proyecto 2**

De la misma manera, en el Proyecto 2, Semgrep se posiciona como la herramienta con mayor precisión, corroborando su alto nivel en la identificación de vulnerabilidades. A pesar de ello OWASP ASST sigue, demostrando una buena precisión aunque inferior a Semgrep. HCL AppScan CodeSweep muestra una precisión significativamente menor en comparación con las herramientas anteriores. Finalmente, SonarQube nuevamente tiene la precisión más baja con un alto número de falsos positivos.

### 4.5.2. Análisis de Recall

**Proyecto 1**

Respecto a la recall, OWASP ASST destaca, reflejando una mejor capacidad para detectar la mayoría de las vulnerabilidades en el código. Para HCL AppScan CodeSweep la puntuación en esta métrica es relativamente alta pero inferior a OWASP ASST. Semgrep presenta una recall inferior a las herramientas anteriormente mencionadas, indicando que, aunque es precisa, no detecta todas las amenazas presentes. Por último, SonarQube muestra el menor porcentaje de detección total de vulnerabilidades, lo cual resulta preocupante ya que muchas de estas pueden pasar por alto.

**Proyecto 2**

Para el segundo proyecto en términos de recall, OWASP ASST continúo mostrando una alta capacidad para detectar vulnerabilidades en el código. En este caso le sigue Semgrep lo que indica que también es efectiva, aunque ligeramente inferior. HCL AppScan CodeSweep muestra un recall considerablemente bajo en comparación

con las otras herramientas. Por último, SonarQube presenta la menor medida, lo cual sugiere que muchas vulnerabilidades no son detectadas.

### 4.5.3. Análisis de efectividad

Para tener un análisis global de la efectividad de cada herramienta, se incluye uno cuadro que muestra el promedio de los resultados de los dos proyectos en las respectivas métricas utilizadas en el estudio:

**Tabla 29**

*Promedio global de los proyectos por métrica*

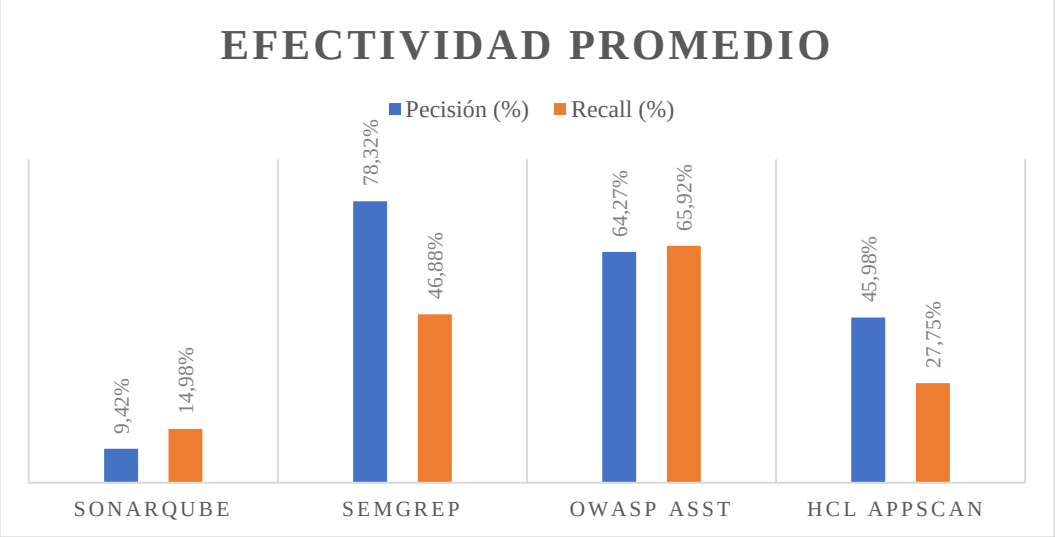
| Herramienta | Precisión (%) | Recall (%) |
|---|---|---|
| **SonarQube** | 9,42% | 14,98% |
| **Semgrep** | 78.32% | 46.88% |
| **OWASP ASST** | 64.27% | 65.92% |
| **HCL AppScan** | 45.98% | 27.75% |

*Nota.* La tabla 29 presenta un promedio por herramienta del análisis de los proyectos con respecto a las métricas precisión y el recall.

*Fuente.* Elaboración propia.

**Figura 9**

*Representación del promedio de los resultados de cada Herramienta*




*Nota.* Representación gráfica de la precisión media y el recall media de cada herramienta. *Fuente.* Elaboración propia

**Precisión:**

La precisión de SonarQube es baja, lo que indica que la herramienta identifica correctamente muy pocas vulnerabilidades en comparación con el total de vulnerabilidades marcadas, sugiriendo un alto número de falsos positivos.

Semgrep muestra una alta precisión, lo que indica que la mayoría de las vulnerabilidades identificadas por la herramienta son correctas. Esto sugiere un bajo número de falsos positivos, lo cual es beneficioso para los desarrolladores al reducir el tiempo dedicado a la revisión de vulnerabilidades innecesarias.

OWASP ASST muestra una buena precisión, indicando que la mayoría de las vulnerabilidades identificadas son correctas, aunque aún podría haber falsos positivos.

HCL AppScan CodeSweep presenta una precisión intermedia, lo que sugiere que una parte considerable de las vulnerabilidades identificadas son correctas, aunque aún existe un número notable de falsos positivos.

**Recall**

El recall también es bajo, lo que significa que SonarQube no logra detectar la mayoría de las vulnerabilidades presentes. Esto es preocupante, ya que muchas vulnerabilidades pueden pasar desapercibidas.

Sin embargo, el recall es relativamente moderado, lo que significa que la herramienta no logra detectar una proporción significativa de las vulnerabilidades presentes. Esto podría indicar que, aunque es precisa, necesita mejoras en su capacidad para descubrir todas las amenazas.

El recall de OWASP ASST es significativamente alto, demostrando una mejor capacidad para detectar la mayoría de las vulnerabilidades presentes en el código. Esto la convierte en una herramienta fiable en términos de cobertura de seguridad.

El recall es moderado, lo que indica que la herramienta tiene una capacidad decente para detectar vulnerabilidades, pero aún puede mejorar en la detección de todas las amenazas presentes.

## 4.6. Resultado del análisis comparativo

OWASP ASST se destaca como la herramienta más equilibrada en términos de efectividad, con buena precisión y alta recall, lo que la convierte en una opción confiable. Su capacidad para detectar la mayoría de las vulnerabilidades y producir un bajo número de falsos positivos hace que sea una elección sólida para la protección de código. Esta herramienta es especialmente valiosa para proyectos que requieren una cobertura de seguridad exhaustiva y precisa.

Semgrep ofrece una excelente precisión, reduciendo falsos positivos, pero su moderado recall sugiere la necesidad de mejorar en la detección de vulnerabilidades. A pesar de esto, sigue siendo una herramienta muy útil para situaciones donde la precisión es crítica, y los desarrolladores pueden complementar su uso con revisiones manuales o herramientas adicionales para asegurar una cobertura completa.

HCL AppScan CodeSweep presenta un rendimiento intermedio en ambas métricas, adecuado pero con espacio para mejoras. Aunque no lidera en ninguna de las categorías, su desempeño es suficientemente robusto para ser considerado en entornos donde se requiere un buen balance entre precisión y recall, con la posibilidad de ajustes y mejoras adicionales para aumentar su eficacia.

SonarQube muestra las métricas más bajas, lo que sugiere que podría no ser la herramienta más adecuada para la identificación y detección de vulnerabilidades en comparación con las otras herramientas evaluadas. Sin embargo, puede seguir siendo útil en contextos específicos donde se requieren capacidades adicionales de análisis de código, aunque debería complementarse con otras herramientas de seguridad para asegurar una protección más completa.

## 4.7. Discusión de resultados

Los resultados de la evolución de herramientas de código para el desarrollo web muestran notables variaciones en términos de uso de recursos, tiempo de análisis y eficiencia. Para llegar a estos resultados se utilizó un minucioso proceso experimental, que consiste en aplicar cada herramienta a proyectos representativos

y utilizar el enfoque ROC para medir métricas importantes como la precisión y el recall.

Durante el desarrollo de la investigación las etapas del proceso se definen claramente, desde la configuración inicial de cada herramienta, hasta el análisis en entornos controlados, al igual que el registro de tiempo, uso de recursos, y número de vulnerabilidades encontradas, tanto falsos positivos, verdaderos positivos y falsos negativos, útiles para esta investigación. La metodología empleada se basa en el análisis descriptivo y el cálculo de métricas de efectividad. (Fawcett, 2006)

Del mismo modo en investigaciones como el de De Vega Martín (2024) menciona que, herramientas como Semgrep y su conjunto de reglas tienen buenos resultados en cuanto a verdaderos positivos y falsos positivos en los lenguajes Java y PHP, siendo de gran utilidad en análisis de tipo SAST.

Esto reafirma que es crucial seleccionar de manera adecuada tanto la tecnología como la herramienta para un análisis de código estático. Como menciona Eterovic et al. (2023) los aspectos a tener en cuenta a la hora de seleccionar una herramienta SAST en lugar de otra es la velocidad con la que deseamos ejecutar el análisis y la profundidad del mismo, ya que, cuanto más profundo el estudio más tiempo conlleva, y viceversa, sin olvidar la relevancia del lenguaje de programación y tener en cuenta los falsos positivos detectados.

De esta manera, el análisis de los datos demuestra que, si bien todas las herramientas utilizadas en el estudio son funcionales, existen variaciones importantes, como la alta precisión de Semgrep, que contribuye con su capacidad de detectar correctamente vulnerabilidades, o SonarQube y OWASP ASST, que ofrecen un bajo tiempo de análisis y consumo de recursos mínimos, sin dejar de lado a HCL AppScan, con su rápida y fácil implementación en entornos de desarrollo.

Estos resultados evidencian que la selección de la herramienta de análisis de código seguro debe estar alineada con las necesidades específicas del proyecto y los recursos disponible en el entorno de desarrollo.

En conclusión, los resultados ponen en evidencia la importancia de contextualizar el despliegue de las herramientas SAST, considerando tanto la precisión de la

detección como el rendimiento operativo. Este método permite la toma de decisiones informadas, y subraya la importancia de integrar diversas estrategias de seguridad para gestionar eficazmente los riesgos en el desarrollo de software.

# CAPÍTULO V

# PROPUESTA:

## 5.1. Guía de uso para la herramienta OWASP ASST.

## 5.2. Introducción

La presente guía de uso, centra sus esfuerzos en guiar a los usuarios para el correcto manejo y funcionamiento de la herramienta OWASP ASST

Esta herramienta fue seleccionada debido a la capacidad que presenta para identificar y mitigar vulnerabilidades en aplicaciones web de manera eficiente y precisa. La creciente importancia de la seguridad en el desarrollo de software exige la adopción de prácticas y herramientas que permitan a los desarrolladores detectar y corregir posibles fallos antes de que estos sean explotados.

Los criterios para esta selección están relacionados directamente con el análisis de resultados, ya que permitió establecer cuál de las herramientas SAST presenta un alto nivel de confiabilidad en base a las métricas establecidas en el estudio.

Por este motivo OWASP ASST (Application Security Static Testing) ha sido seleccionada como la herramienta preferida para la creación de esta guía de uso debido a su notable equilibrio entre precisión y recall en la detección de vulnerabilidades.

## 5.3. Objetivo

Proporcionar una guía práctica y detallada para la instalación, configuración y uso de OWASP ASST, permitiendo a los desarrolladores realizar análisis de seguridad estático en proyectos web.

## 5.4. Descarga e Instalación

Para la instalación de este software hay que tener en cuenta las siguientes consideraciones para garantizar un funcionamiento cien por ciento:

Localhost en su PC (recomendado XAMMP)

Node.js Engine v12.13.0

### 5.4.1. Descarga e instalación de complementos.

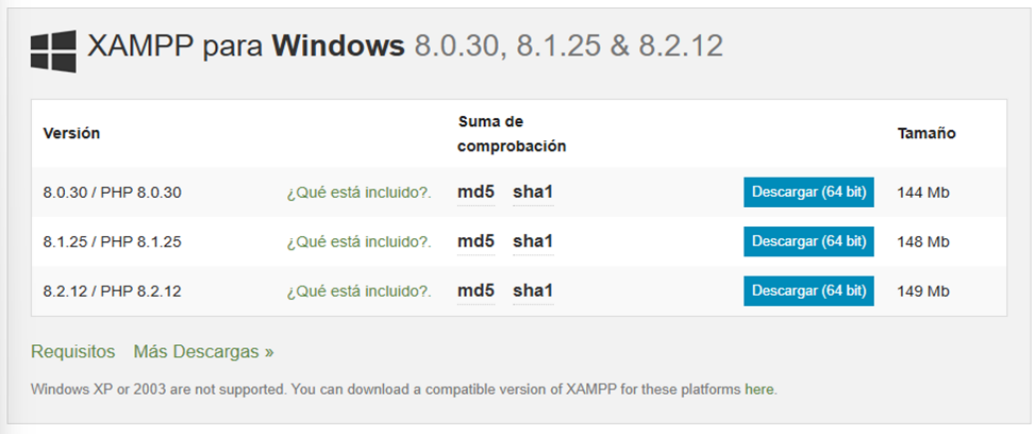
**Xampp:**

**Paso 1:** abrir el sitio web de XAMPP https://www.apachefriends.org/es/download.html en el navegador.

**Paso 2:** dirigirse al apartado de XAMPP para Windows y presionar en descargar.

**Figura 10**

*Página de descarga de XAMPP*



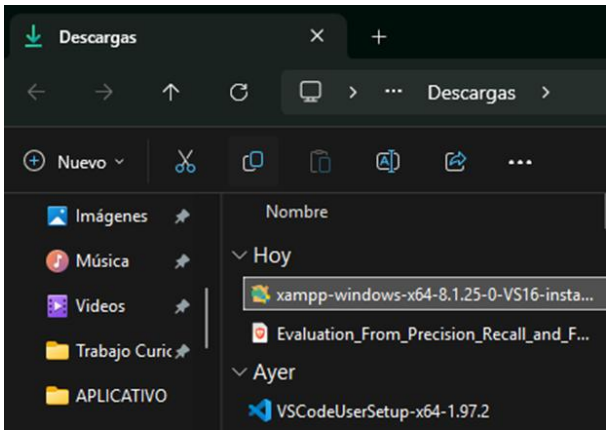

*Nota*. Captura de pantalla de la página oficial de XAMPP

*Fuente*. Tomado de Apache Friends (2024)

**Paso 3:** hacer doble clic en el archivo descargado y aceptar los permisos de Windows.

**Figura 11**

*Archivo de Instalación XAMPP*



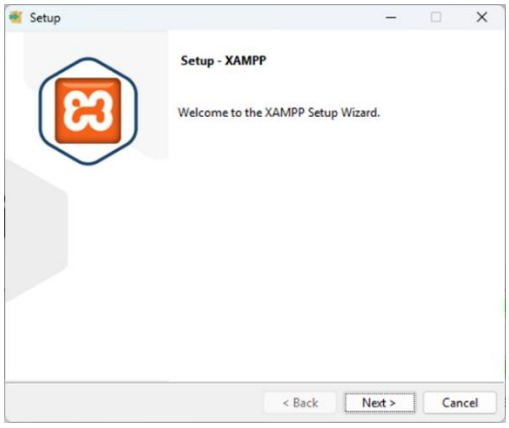

*Nota*. Captura de pantalla del archivo descargado en el sitio web de XAMPP.

*Fuente*. Elaboración propia.

**Paso 4:** una vez aparezca la ventana de diálogo procedemos a seleccionar siguiente hasta que empiece la instalación, esto hará que XAMPP establezca los valores predeterminados. Hay que tener en cuenta que para el funcionamiento de la herramienta OWASP ASST la instalación predeterminada de XAMPP es adecuada. ESTRUCTURA DE LA GUÍA

**Figura 12**

*Inicio de la instalación XAMPP*



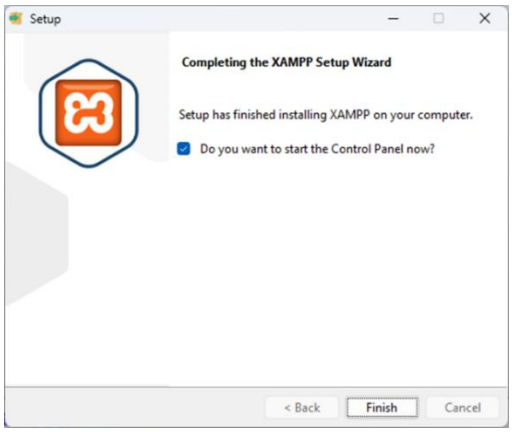

*Nota*. Captura de pantalla del proceso de instalación de XAMPP

*Fuente*. Elaboración propia

**Paso 5:** una vez termine la instalación nos mostrará una ventana donde podemos iniciar el panel de control de XAMPP o nos dirigimos a la ruta donde se instaló el software y buscamos xampp-control.exe para ejecutar el panel de control XAMPP. (ruta predeterminada: C:\xampp)

**Figura 13**

*Instalación completa de XAMPP*



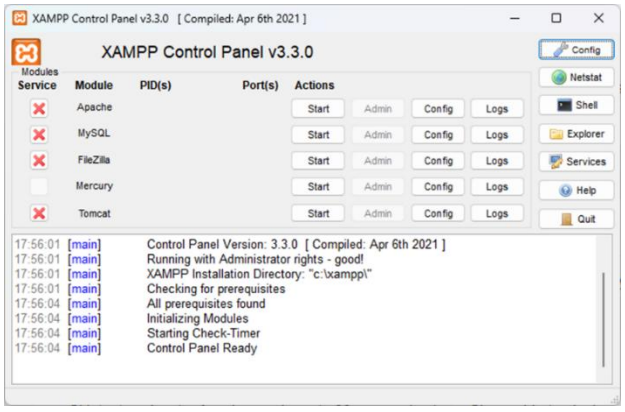

*Nota.* Captura de pantalla de la finalización del proceso de instalación XAMPP.

*Fuente.* Elaboración propia.

**Paso 6:** una vez hechos los pasos anteriores ya se puede empezar a utilizar XAMPP

**Figura 14**

*Panel de control XAMPP*




*Fuente.* Elaboración propia

**Node.js:**

**Paso 1:** abrir el sitio web de Node.js https://nodejs.org/en/download/ en el navegador.

**Paso 2:** seleccionar el sistema operativo y la arquitectura después presionar en descargar.

**Figura 15**

*Página de descarga de Node.js*



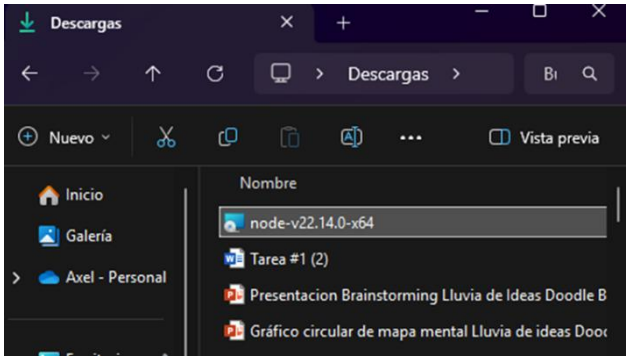

*Note.* Captura de pantalla del sitio oficial de descarga de la herramienta Node.js

*Fuente.* Tomado de OpenJS Foundation (2024)

**Paso 3:** hacer doble clic en el archivo descargado y aceptar los permisos de Windows.

**Figura 16**

*Archivo de Instalación Node.js*

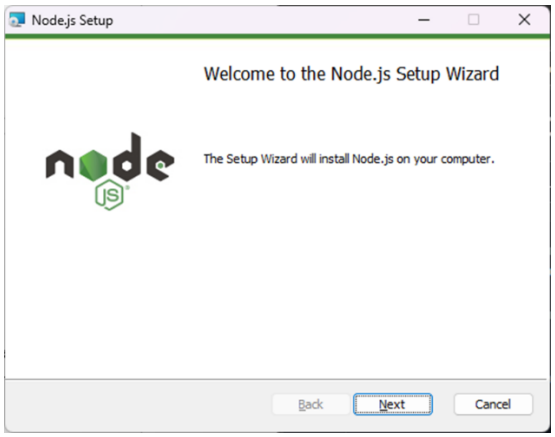



*Nota.* Captura de pantalla del archivo de instalación para Node.js. *Fuente. Elaboración propia*

**Paso 4:** una vez aparezca la ventana de diálogo procedemos a seleccionar siguiente hasta que empiece la instalación, esto hará que Note.js establezca los valores predeterminados. Debemos tener en cuenta que hay que aceptar los términos y condiciones del software.

**Figura 17**

*Inicio de la instalación de Node.js*




*Nota.* Captura de pantalla de la ventana inicial del proceso de instalación de Node.js

*Fuente.* Elaboración propia.

**Figura 18**

*Términos y condiciones de Node.js*




*Nota.* Captura de pantalla de los términos y condiciones de Node.js

*Fuente.* Elaboración propia.

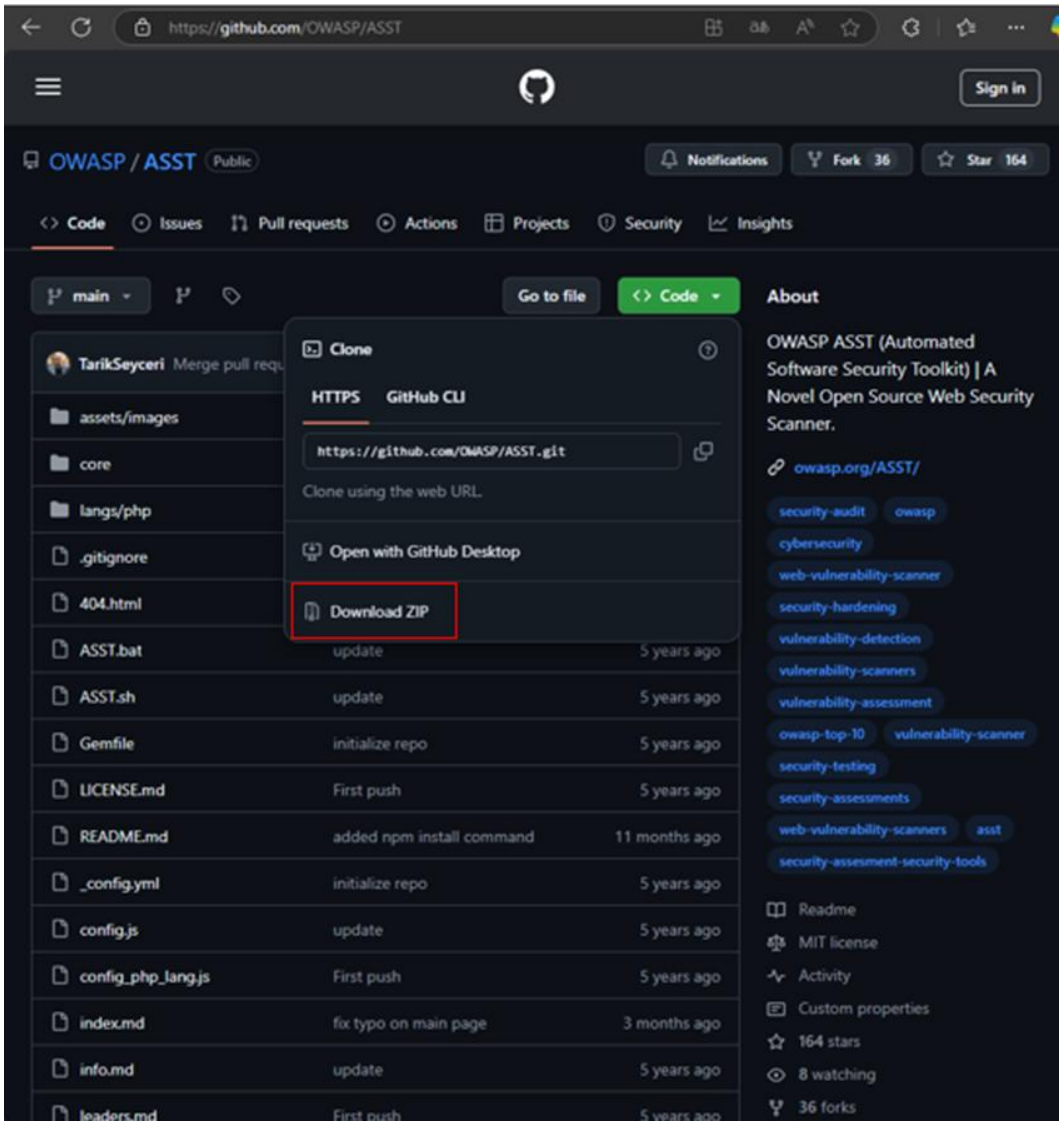
**Paso 5:** una vez completada la instalación hacer clic en el botón para finalizar con el proceso.

**Descarga de OWAP ASST**

**Paso 1:** descargar el proyecto de ASST de la página de GitHub https://github.com/OWASP/ASST

**Figura 19**

*Herramienta OWASP ASST*



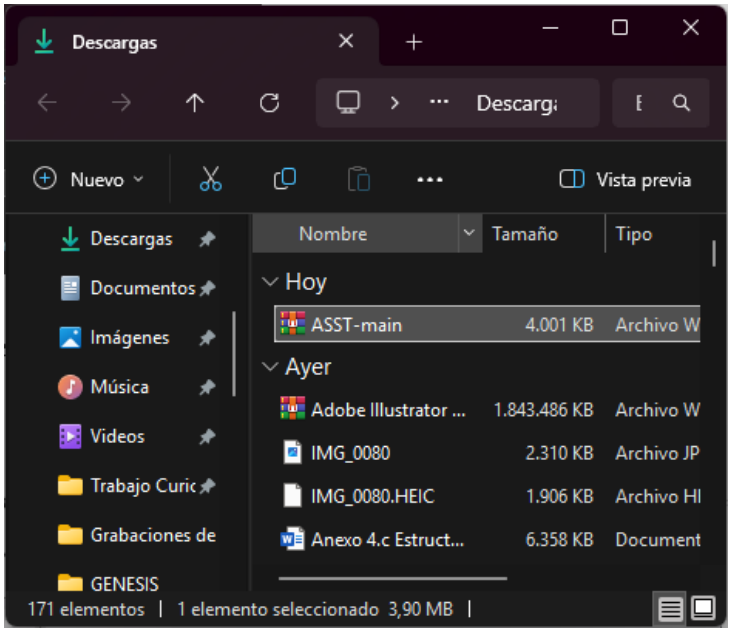

*Nota*. Captura de pantalla del proyecto OWASP ASST en el repositorio digital GitHub

*Fuente*. Tomado de Seyceri (2024)

**Paso 2:** Extraer el proyecto del archivo zip descargado

**Figura 20**

*Archivo de la herramienta OWASP ASST*



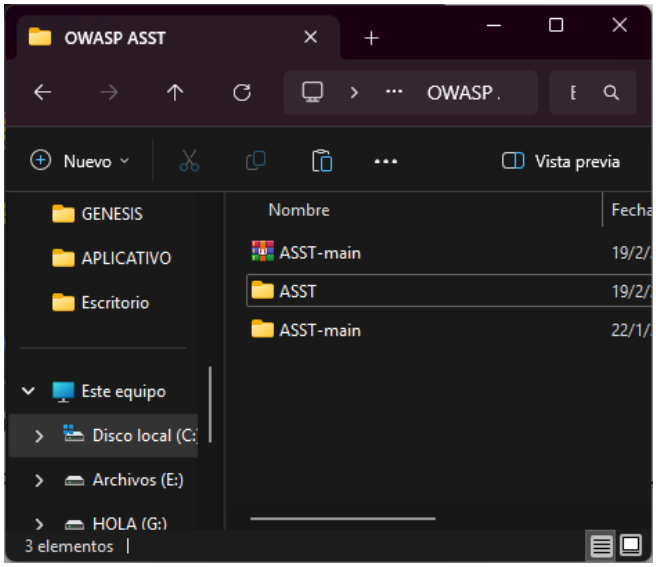

*Nota.* Captura de pantalla del archivo zip descargado de GitHub de la herramienta OWASP ASST

*Fuente.* Elaboración propia

**Paso 3:** una vez extraída la carpeta de la herramienta cambiar el nombre de "ASST-main" a "ASST", se recomienda hacer una copia antes de cambiar el nombre

**Figura 21**

*Extracción de la herramienta*

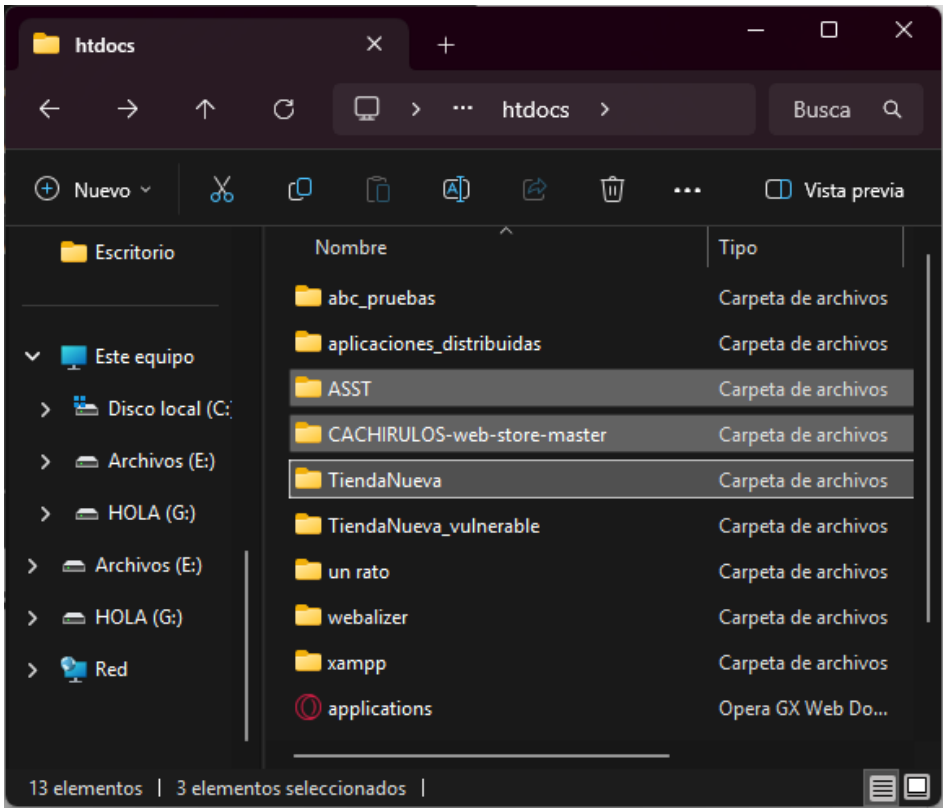



*Nota.* Captura de pantalla de cambio de nombre de la carpeta ASST-main para poder utilizar la herramienta. *Fuente.* Elaboración propia.

**Paso 4:** Mueva la carpeta “ASST” junto al proyecto para escanearlo, si se utiliza el servidor local XAMPP la ruta predeterminada debe quedar de la siguiente manera: C:\xampp\htdocs\ASST

**Figura 22**

*OWASP ASST y los proyectos a escanear*




*Nota.* Captura de pantalla de cambio de ubicación de la carpeta ASST.

*Fuente.* Elaboración propia.

## 5.5. Guía de Configuración de OWASP ASST

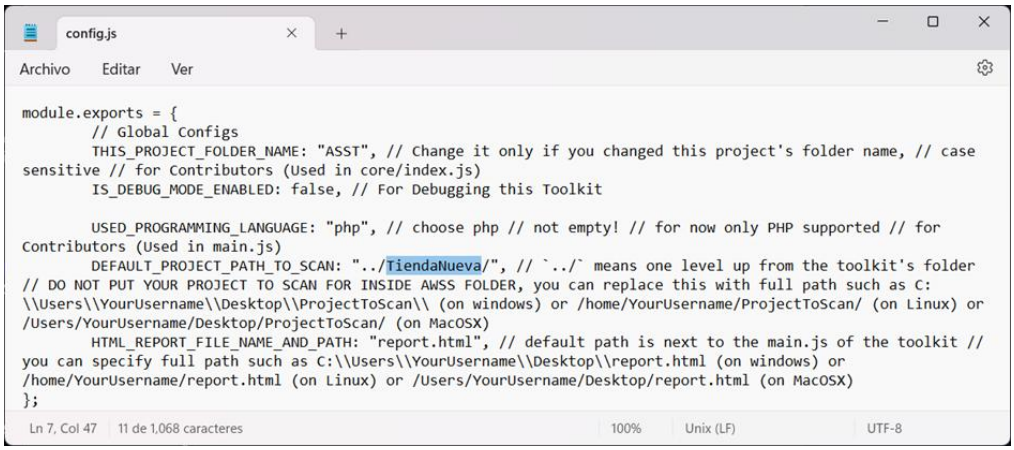
### Configuración Archivo config.js

**Paso 1:**

Abrir el archivo config.js dentro de la carpeta de ASST y configure el nombre de la carpeta de su proyecto web que se escaneará en la variable DEFAULT_PROJECT_PATH_TO_SCAN.

**Figura 23**

*Archivo config.js*




*Nota*. Captura de pantalla de la configuración del archivo config.js. *Fuente.* Elaboración propia.

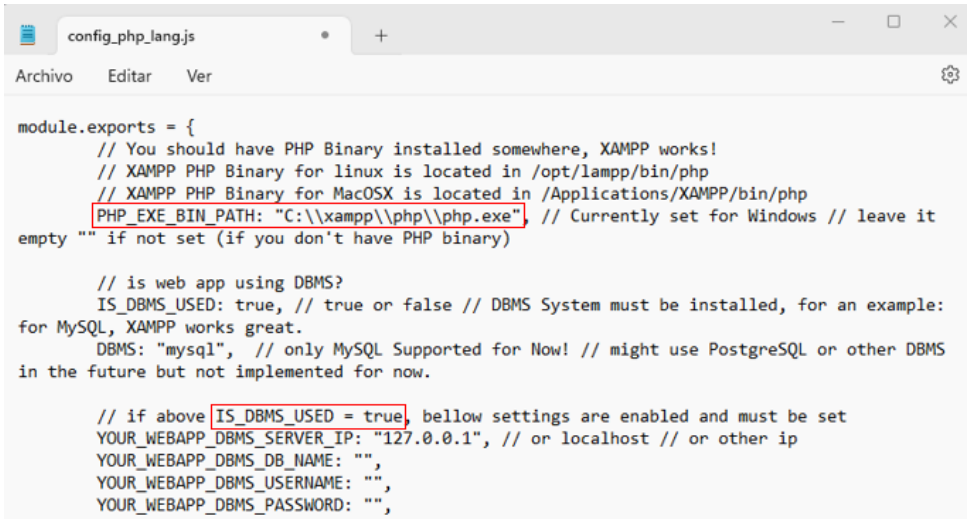
## Configuración Archivo config_php_lang.js

### Paso 1:

Abrir el archivo config_php_lang.js dentro de la carpeta ASST: si se utiliza MySQL (base de datos predeterminada en XAMPP) seguir al paso 3, caso contrario simplemente configure la variable IS_DBMS_USED en falso e ignore el resto, tener en cuenta que PHP_EXE_BIN_PATH está configurado en la ubicación predeterminada de XAMPP, así que cámbielo si está usando una ubicación diferente o un binario PHP diferente.

**Figura 24**

*Archivo config_php_lang.js*




*Nota*. Captura de pantalla del archivo config_php_lang.js resultandos los puntos que se mencionan en el paso 1. *Fuente* elaboración propia.

**Paso 2:**

Para hacer la configuración del archivo config_php_lang.js utilizando MySQL debe configurar la información de la base de datos del proyecto correspondiente

**Figura 25**

*Modificación de las variables de la base de datos*

```
// if above IS_DBMS_USED = true, bellow settings are enabled and must be set
YOUR_WEBAPP_DBMS_SERVER_IP: "127.0.0.1", // or localhost // or other ip
YOUR_WEBAPP_DBMS_DB_NAME: "bd",
YOUR_WEBAPP_DBMS_USERNAME: "root",
YOUR_WEBAPP_DBMS_PASSWORD: "",
```

*Nota.* Configuración de las credenciales de la base de datos del proyecto.

*Fuente.* Elaboración propia

**Paso 3:**

Configurar la información de las tablas necesarias “user” y “admin” de ser necesario con la información de su base de datos

Nota: para el proyecto a analizar estas tablas contiene información sensible

**Figura 26**

*Configuración de los apartados user y admin*

```
YOUR_WEBAPP_DBMS_USER_TABLE_NAME: "clientes",
YOUR_WEBAPP_DBMS_USER_TABLE_PASSWORD_COLUMN_NAME: "contasena",
YOUR_WEBAPP_DBMS_USER_TABLE_SALT_COLUMN_NAME: "", // if not existed or used bcrypt
or don't know what it is, leave it empty text: ""

YOUR_WEBAPP_DBMS_ADMIN_TABLE_NAME: "administradores", // maybe same of user table,
so write same table name // if hardcoded admin credentials or doesn't exist, leave
it empty text: ""
YOUR_WEBAPP_DBMS_ADMIN_TABLE_PASSWORD_COLUMN_NAME: "contasena", // maybe same of
user table, so write same column name
YOUR_WEBAPP_DBMS_ADMIN_TABLE_SALT_COLUMN_NAME: "", // if not existed or used
bcrypt or don't know what it is, leave it empty text: "" // maybe same of user
table, so write same column name
};
```

24, Col 2 | 1,647 caracteres. | 100% | Unix (LF) | UTF-8

*Nota.* Captura de pantalla de la información de los campos user y admin con la información de la base de datos del proyecto al que se hará el análisis.

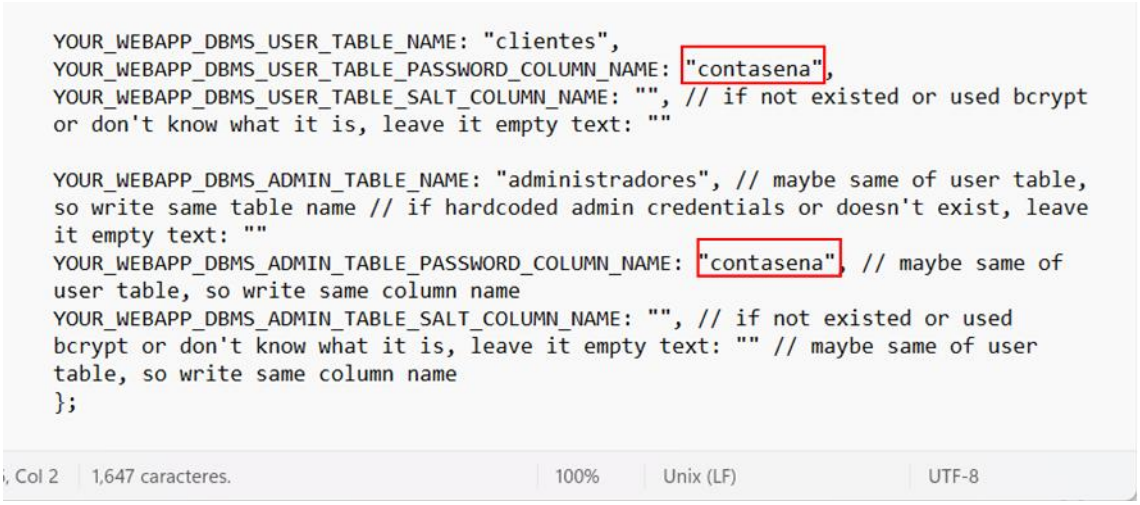
*Fuente.* Elaboración propia.

**Paso 4:**

Cambiar el nombre de la columna correspondiente a la contraseña del user y el admin

**Figura 27**

*Configuración de la columna contraseña del user y admin*




*Nota*. Captura de pantalla del cambio de nombre de la columna contraseña correspondiente a la información del proyecto a analizar. *Fuente.* Elaboración propia.

## 5.6. Ejecución de OWASP ASST

Tras completar la instalación y configuración de OWAST ASST, su ejecución es un proceso simple, debido a que la herramienta está diseñada para facilitar el análisis de seguridad con comandos sencillos. Este apartado, explica de manera rápida cómo iniciar la herramienta.

**Paso 1**

Haga doble clic en ASST.bat para ejecutarlo. Si Windows Defender Smart Screen lo bloquea, permítalo, haciendo clic en Más información y luego en Ejecutar o Ejecutar de todos modos, o simplemente puede ejecutarlo usando el comando CMD.

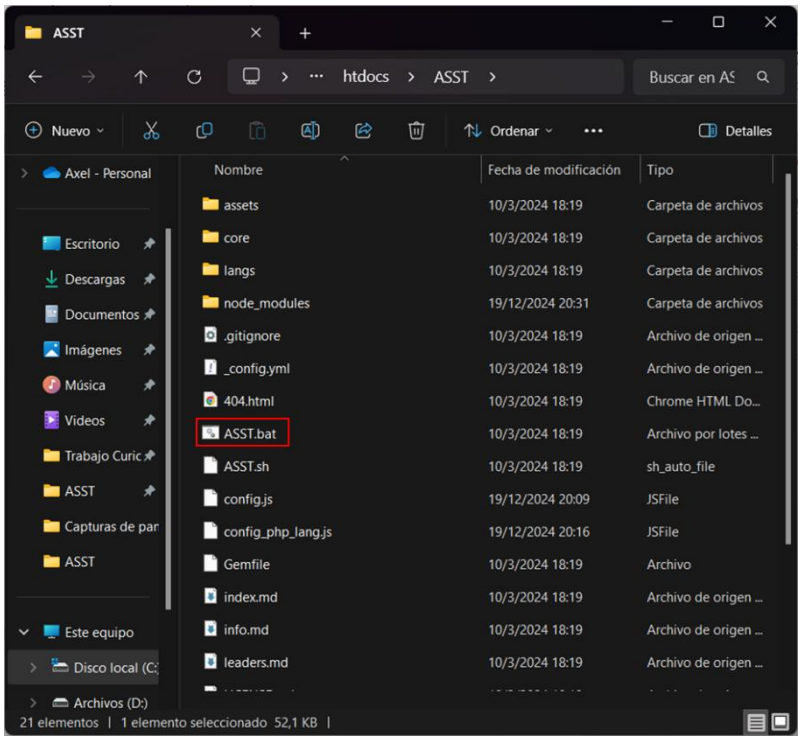
Comandos cmd:

Comando CMD predeterminado para ejecutar ASST:

$ nodo C:\xampp\htdocs\ASST\main.js

**Figura 28**

*Archivo ASST.bat*



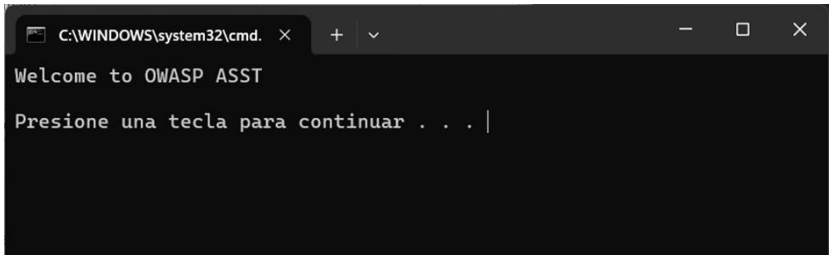

*Nota*. Captura de pantalla que muestra el archivo ASST.bat el cual se debe ejecutar. *Fuente.* Elaboración propia.

**Paso 2:**

Una vez ejecutado se abrirá una ventana donde se presionará una tecla para iniciar el análisis.

**Figura 29**

*Ejecución de OWASP ASST*

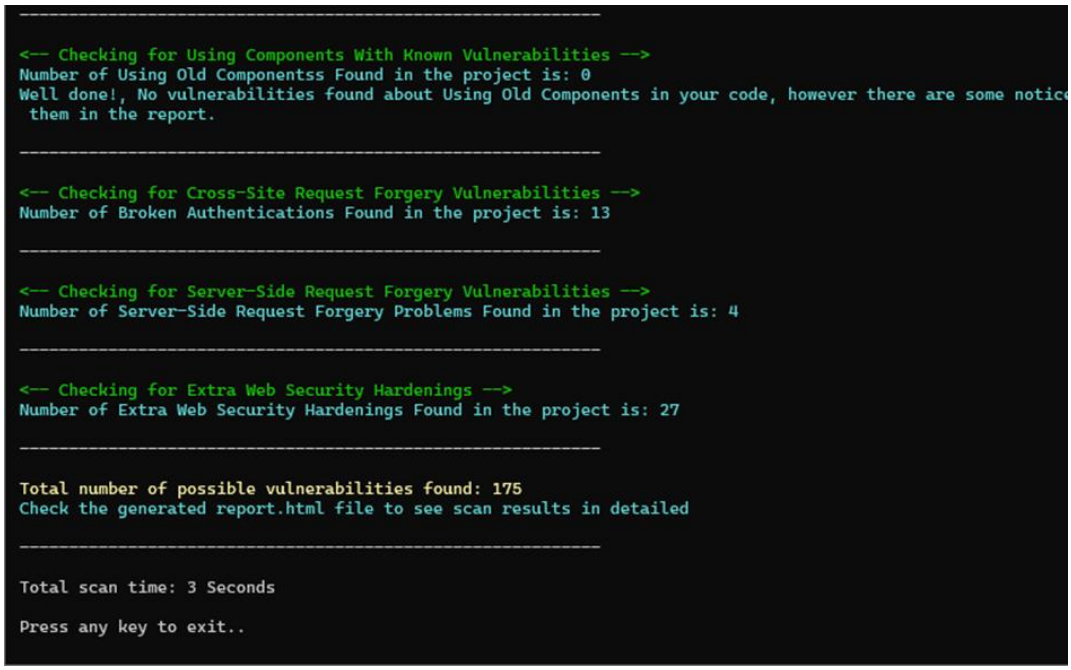



*Nota*. Captura de pantalla de la ventana de diálogo para iniciar el análisis del proyecto configurado.

*Fuente*. Elaboración propia.

**Figura 30**

*Finalización del proceso de análisis*




*Nota*. Captura de pantalla que muestra la finalización del análisis al proyecto seleccionado.

*Fuente.* Elaboración propia.

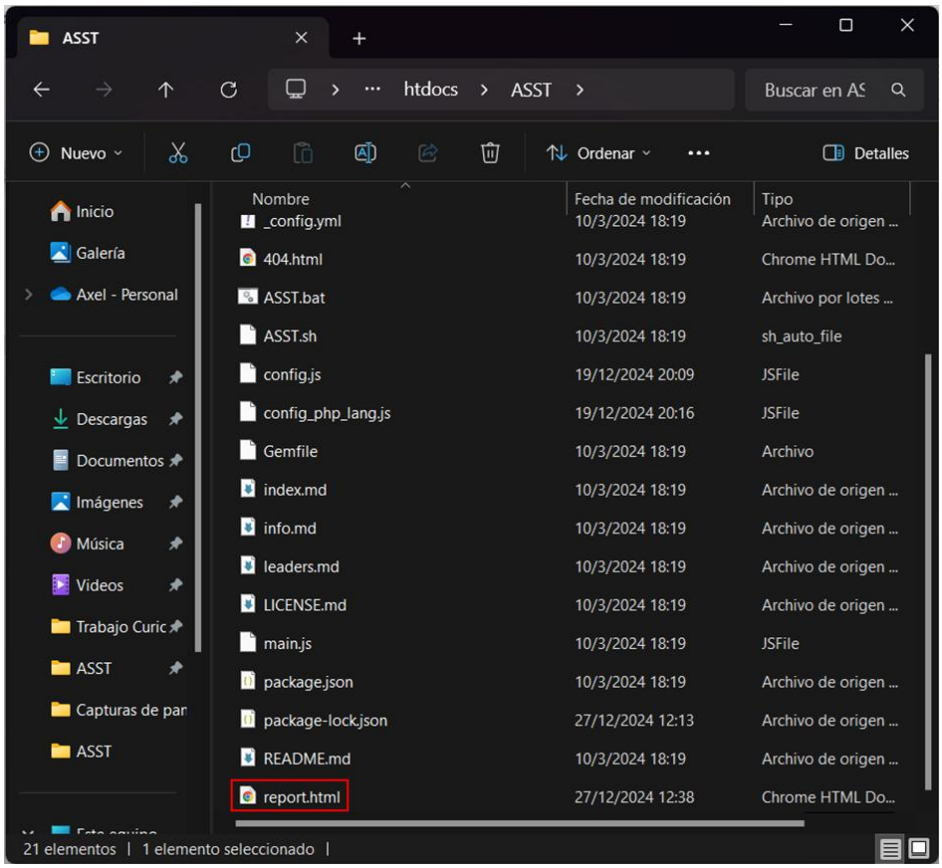
**Paso 3:**

Una vez terminado el análisis se solicitará que se presione una tecla para dar por finalizado el proceso. Del mismo modo se generará un reporte en la carpeta ASST.

El reporte del análisis realizado con esta herramienta se presenta en el archivo nombrado report.html que con la siguiente ruta predeterminada C:\xampp\htdocs\ASST

**Figura 31**

*Archivo report.html*




*Nota.* Captura de pantalla del reporte generado después del análisis correspondiente.

*Fuente.* Creación propia.

Una de las ventajas de utilizar esta herramienta es que los resultados se presentan con más detalle mediante un reporte que ayuda en la interpretación de los resultados obtenidos como se puede ver a continuación.

**Figura 32**

*Reporte del análisis con OWASP ASST*




*Nota*. Captura de pantalla del reporte generado por OWASP ASST

*Fuente.* Elaboración propia

Una vez obtenido el reporte de la aplicación analizada, esta información resulta de gran ayuda para el equipo de desarrollo detrás del proyecto, debido a que facilita la corrección oportuna de aquellas vulnerabilidades que no impiden la puesta en producción del software, evitando las futuras fallas en seguridad, provocando pérdidas de información, accesos no autorizados y otro tipo de ataques.

La estructura precisa y ordenada del informe permite priorizar la corrección de vulnerabilidades en función de su criticidad, eliminando riesgos innecesarios en el ciclo de vida del desarrollo.

De la misma forma, una vez obtenido el reporte, esta guía de uso propone el siguiente curso de acción:

- Las vulnerabilidades se clasifican y priorizan, haciendo hincapié en las de mayor impacto.
- Los errores se corrigen siguiendo las mejores prácticas de seguridad.
- Revolución del código tras la rectificación para garantizar que los problemas se han corregido correctamente.
- Formación del equipo de desarrollo, con las conclusiones del análisis como referencia para mejorar la seguridad en futuras implementaciones.
- Integración del análisis SAST en el proceso, automatizando las revisiones periódicamente para detectar vulnerabilidades de forma continua

## CONCLUSIONES

- Los resultados revelan que, si bien las herramientas disponibles en 2024 logran su objetivo de aumentar la seguridad del software, difieren significativamente en términos de eficacia, tiempos de análisis y uso de recursos. Esto pone en evidencia la necesidad de seleccionar la herramienta adecuada en función de las características y los requisitos del proyecto, mejorando así la seguridad y preservando el rendimiento del desarrollo.
- El proyecto de investigación ha identificado las principales herramientas de análisis de código seguro del mercado para el año 2024, basándose en criterios como la cobertura de vulnerabilidades, la facilidad de uso, la compatibilidad y el tipo de licencia. De las cuales destacan herramientas como SonarQube, Semgrep, OWASP ASST y HCL AppScan CodeSweep, cada una de las cuales presenta diferentes características y enfoques en cuanto a detección de vulnerabilidades, tiempos de análisis y consumo de recursos. Esto ofrece a los desarrolladores un abanico de opciones que pueden ser evaluadas en función de las necesidades específicas de cada proyecto.
- La evaluación comparativa de la eficacia de detección de vulnerabilidades en entornos de desarrollo web reveló variaciones considerables. Aunque todas encuentran vulnerabilidades reales, algunas como OWASP ASST y Semgrep tienen mayores tasas de detección y menos falsos positivos, y otras, tal como SonarQube proporcionan escaneos exhaustivos pero pueden generar más falsos negativos y positivos. Estos resultados subrayan la necesidad de elegir la herramienta adecuada no sólo por su capacidad de detección, sino también por su compatibilidad con las características y requisitos únicos de cada proyecto web.
- La comparación del rendimiento demuestra que las herramientas presentan diferencias significativas en los tiempos de análisis y la utilización de recursos. Ya que, algunas, como OWASP ASST, completan los análisis rápidamente y con poco uso de CPU y memoria, mientras que otras, como HCL AppScan o SonarQube, requieren más potencia de procesamiento y duraciones de ejecución más largas para ofrecer análisis más extensos. Estos

resultados subrayan la importancia de tener en cuenta tanto los límites del hardware como la influencia en la productividad a la hora de elegir la herramienta óptima para el entorno de desarrollo.

- La mejor herramienta para su implantación en entornos de desarrollo de software debe encontrar un equilibrio entre eficacia, rendimiento y compatibilidad con los procesos existentes. A partir de la evaluación, se concluye que la herramienta elegida no sólo debe detectar con precisión las vulnerabilidades, sino también aprovechando al máximo la infraestructura disponible y adaptándose a las características únicas de cada proyecto, este movimiento estratégico permite optimizar la seguridad del código sin perder eficacia operativa.

## RECOMENDACIONES

- Es fundamental instar a la comunidad académica a seguir avanzando en el campo de la seguridad de software, animando a universidades, facultades y programas de ingeniería a incorporar estos temas en los planes de estudio y a las iniciativas de investigación en colaboración, no solo ampliando el conocimiento científico, sino que también formará a los futuros profesionales para afrontar los retos de la ciberseguridad en el desarrollo del software.
- Se recomienda investigar más casos de estudio con diferentes condiciones al presente análisis, evaluando herramientas en múltiples entornos y lenguajes de programación. Esto mejorará nuestra comprensión del rendimiento de estas tecnologías y contribuirá al desarrollo de soluciones de protección de software más eficaces.
- Se recomienda el uso temprano de herramientas de análisis seguros de código a lo largo del ciclo de vida del software para ayudar a detectar y rectificar vulnerabilidades antes de que se convierten en problemas graves. Además se aconseja que la selección de estas herramientas para su implementación sea acorde a las características específicas del proyecto con que se trabaja.
- Para ofrecer una protección continua del sistema, se recomienda evaluaciones y actualizaciones periódicas de las prácticas de seguridad en respuesta al desarrollo de amenazas y avances técnicos.

## BIBLIOGRAFÍA


Agudelo Viana, L. G., & Aigneren Aburto, J. M. (2008). Diseños de investigación experimental y no-experimental. *Universidad de Antioquia, Facultad de Ciencias Sociales y Humanas*. https://bibliotecadigital.udea.edu.co/bitstream/10495/2622/1/AgudeloGabriel_2008_DisenosInvestigacionExperimental.pdf

Apache Friends. (2024). *Descargar XAMPP*. https://www.apachefriends.org/es/download.html

Arpi-Saquipay, W. A., & Cajamarca-Criollo, O. A. (2023). Análisis de riesgos de seguridad de la información en el proceso de contratación de personal en una Institución de Educación Superior en Ecuador, basado en la Norma ISO 27002 Anexo A dominio 7. *MQRInvestigar*, *7*(3), 2793–2808. https://doi.org/10.56048/mqr20225.7.3.2023.2793-2808

Bellairs, R. (2019). *What Is Static Analysis (Static Code Analysis)?* Perforce.

Caño Quintero, J. J. (2019). *DevSecOps: integración de herramientas SAST, DAST y de análisis de Dockers en un sistema de integración continua.* https://openaccess.uoc.edu/bitstream/10609/95987/6/jjcquinteroTFM0619memoria.pdf

Código Orgánico Economía Social Conocimientos Creatividad Innovación, Pub. L. No. f/n, Registro Oficial Suplemento 899 (2016). https://www.gob.ec/sites/default/files/regulations/2019-02/Documento_C%C3%B3digo-Org%C3%A1nico-Econom%C3%ADa-Social-Conocimientos-Creatividad-Innovaci%C3%B3n.pdf

Coronel Suárez, I., & Quirumbay Yagual, D. (2022). Seguridad informática, metodologías, estándares y marco de gestión en un enfoque hacia las aplicaciones web. *Revista Científica y Tecnológica UPSE*, *9*(2), 97–108. https://doi.org/10.26423/rctu.v9i2.672

De Vega Martín, M. E. (2024). *Metodología de benchmark de herramientas SAST*.

Decreto N° 1014, Política pública para las entidades de la Administración Pública Central la utilización del software libre en sus sistemas y equipamientos informáticos, Pub. L. No. Decreto 1014, Registro Oficial (2008). https://www.gobiernoelectronico.gob.ec/wp-content/uploads/2018/09/Decreto-Ejecutivo-N%C2%B0-1014-del-10-de-abril-del-2008.pdf

Dewhurst, R. (2024). *Static Code Analysis*. OWASP. https://owasp.org/www-community/controls/Static_Code_Analysis

Eterovic, J., Silvestri, V., Vera, A., Zeballos, M., & Sinopoli, A. E. (2023). *Análisis de las Herramientas para Realizar Pruebas Estáticas de Seguridad de las Aplicaciones*.

Faneite, S. F. (2023). Los enfoques de investigación en las Ciencias Sociales. *REVISTA LATINOAMERICANA OGMIOS*, *3*. https://doi.org/10.53595/rlo.v3.i8.084

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

García Pérez Javier. (2019). *Análisis de fiabilidad de las herramientas SAST. Privativo vs Libre*. https://reunir.unir.net/handle/123456789/9554

García-Peñalvo, F. J. (2018). *Ingeniería del Software*. https://repositorio.grial.eu/handle/grial/1228

Gartner. (2024). *Static Application Security Testing (SAST)*. https://www.gartner.com/en/information-technology/glossary/static-application-security-testing-sast

Guevara Alban, G. P., Verdesoto Arguello, A. E., & Castro Molina, N. E. (2020). Metodologías de investigación educativa (descriptivas, experimentales, participativas, y de investigación-acción). *RECIMUNDO*, *4*(3), 163–173. https://doi.org/10.26820/recimundo/4.(3).julio.2020.163-173

HCL Software. (2024). *HCL AppScan CodeSweep*. Tomado de https://marketplace.visualstudio.com/items?itemName=HCLTechnologies.hclappscancodesweep

Hernández Sampieri, Roberto., & Mendoza Torres, C. Paulina. (2018). *Metodología de la investigación: las rutas cuantitativa, cualitativa y mixta* (Mcgraw-Hill Educación, Ed.; Primera).

Jurn, J., Kim, T., & Kim, H. (2018). An Automated Vulnerability Detection and Remediation Method for Software Security. *Sustainability*, *10*(5), 1652. https://doi.org/10.3390/su10051652

KirstenS. (2024). *Cross Site Scripting (XSS)*. OWASP. https://owasp.org/www-community/attacks/xss/

Ley Orgánica de Protección de Datos Personales, Pub. L. No. 0, Registro Oficial Suplemento 459 (2021). https://www.finanzaspopulares.gob.ec/wp-content/uploads/2021/07/ley_organica_de_proteccion_de_datos_personales.pdf

Li, J. (2020). Vulnerabilities mapping based on OWASP-SANS: A survey for static application security testing (SAST). *Annals of Emerging Technologies in Computing*, *4*(3). https://doi.org/10.33166/AETiC.2020.03.001

López Álvarez, D. M. (2020). Método para el desarrollo de software seguro basado en la ingeniería de software y ciberseguridad. *INNOVA Research Journal*, *5*(3.1), 263–280. https://doi.org/10.33890/innova.v5.n3.1.2020.1440

Morales, N. (2015). *Investigación Exploratoria: Tipos, Metodología y Ejemplos*. https://d1wqtxts1xzle7.cloudfront.net/64537756/Investigaci%C3%B3n_Exploratoria-libre.pdf?1601263412=&response-content-disposition=inline%3B+filename%3DInvestigacion_Exploratoria_Tipos_Metodol.pdf&Expires=1741794708&Signature=Ywxv5m9HmQZc8z-q~LtR8aQisVlAiBopfH3RXfby6pOLcrrYvomuG0hAuN9fMeUcRGIFbkxLxlRBX6-MLrNiVx0xUsU10NM8nI0sJDlr6-dmzcOv-uOgpq5ard4Qx-3UWGc5R0Hi3rkipVTn8Q~ffT8wqEsh4H2kAsO8FZy6R6lpsIUlBuLrvEOtfNAjyd2TuCsx6K9g~p-l7Xq002f-

yOEeYOkvTYNjOjwhRpWnivu5MYDY4vi0clSoRC9ccH3C8yP8hbWBnFom3t~iRtuWm5qHBXO-pE4MII0G9KqzeYTr7HBx9aa9htDBcEEnQEJ-xaBAWD74BE1f60KZV171rQ__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

OpenJS Foundation. (2024). *Download Node.js*.

Organización Internacional de Normalización (ISO). (2013). *ISO/IEC 27001:2013 - Tecnología de la información - Técnicas de seguridad - Sistemas de gestión de seguridad de la información - Requisitos*. ISO/IEC 27001:2013. https://www.iso.org/es/contents/data/standard/08/28/82875.html?tid=331662889277

Organización Internacional de Normalización (ISO). (2017). *ISO/IEC/IEEE 12207:2017(es) Ingeniería de sistemas y software: procesos del ciclo de vida del software*. ISO/IEC/IEEE 12207:2017. https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:12207:ed-1:v1:en

OWASP. (2021). *A03:2021 – Inyección*. https://owasp.org/Top10/A03_2021-Injection/

Sánchez, G., & Ramírez, L. (2022). Amenazas de seguridada considerar en el desarrollo de softwareSecurity treats to consider in software development. *Publicación semestral*, *10*(19), 31–37. https://repository.uaeh.edu.mx/revistas/index.php/xikua/issue/archive

Sánchez Molina, A. A., & Murillo Garza, A. (2022). Enfoques metodológicos en la investigación histórica: cuantitativa, cualitativa y comparativa. *Debates por la historia*, *9 núm.2*. https://www.scielo.org.mx/scielo.php?pid=S2594-29562021000200147&script=sci_arttext

Sarhan, A. A., Farhan, S. A., & Al-Harby, F. M. (2018). Understanding and discovering sql injection vulnerabilities. *Advances in Intelligent Systems and Computing*, *593*, 45–51. https://doi.org/10.1007/978-3-319-60585-2_5

Semgrep.dev. (2024). *Semgrep docs*. https://semgrep.dev/docs/

Seyceri, T. (2024). *OWASP ASST*. OWASP. https://owasp.org/ASST/

SonarQube. (2024). *SonarQube Community Build Documentation*. https://docs.sonarsource.com/sonarqube-community-build/

Stallman, R. (2020). Definición de Software Libre. *Communiars. Revista de Imagen, Artes y Educación Crítica y Social*, *3*, 151–154. https://doi.org/10.12795/communiars.2020.i03.09

Veloz, E. A., & Veloz, V. T. (2024). Vulnerabilidades en aplicaciones web, amenazas y ataques. *Revista Científica Multidisciplinar G-ner@ndo*, *5*, 384–393. https://orcid.org/0000-0003-

Ward Powers, D. M. (2008). *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation*.

Zambrano, A., Guarda, T., Haro, E., & Ninahualpa, G. (2019). Técnicas de mitigación para principales vulnerabilidades de seguridad en aplicaciones web. *Revista Ibérica de Sistemas e Tecnologias de Informação*, *E17*, 299–308. https://www.researchgate.net/publication/331178479

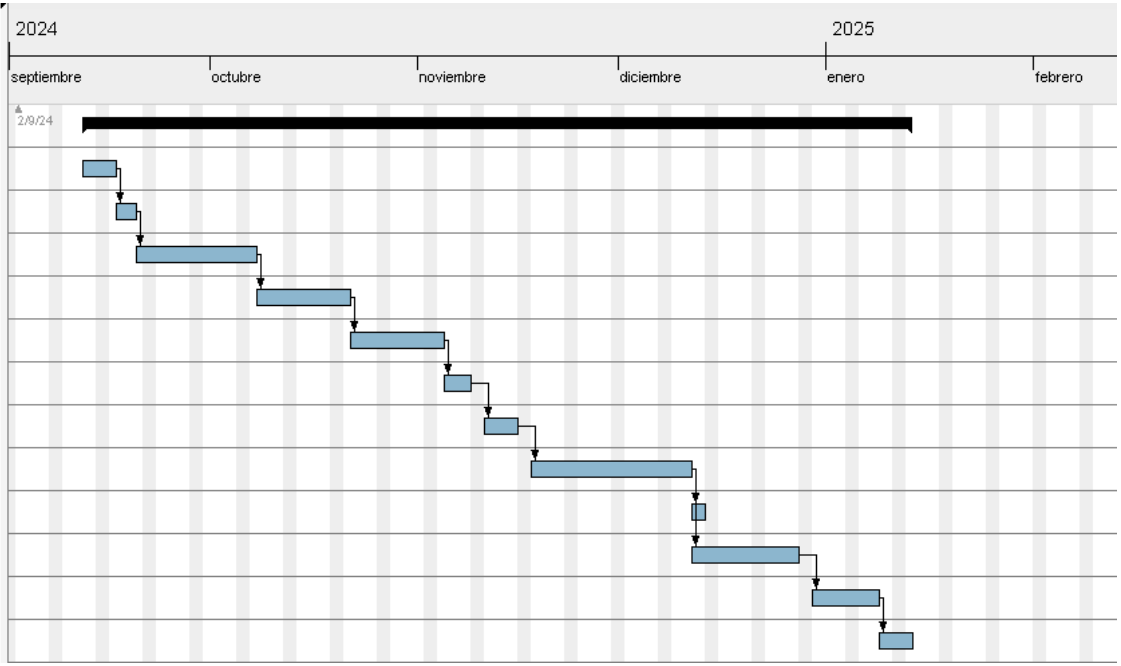
# ANEXOS

# ANEXOS 1:

## Cronograma (Gantt)

## Cronograma Aproximado

| Nombre | Fecha de inicio | Fecha de fin |
|---|---|---|
| Proyecto de Investigasion | 12/9/24 | 13/1/25 |
| Presentación Tema | 12/9/24 | 16/9/24 |
| Formulación general del Proyecto | 17/9/24 | 19/9/24 |
| Revisión bibliográfica del tema | 20/9/24 | 7/10/24 |
| Planteamiento del primer capitulo | 8/10/24 | 21/10/24 |
| Marco Teorico | 22/10/24 | 4/11/24 |
| Seleccion de Herramientas | 5/11/24 | 8/11/24 |
| Revision de Herramientas SAST | 11/11/24 | 15/11/24 |
| Evaluasion de las Herramientas Seleccionadas | 18/11/24 | 11/12/24 |
| Formulación general de la Propuesta | 12/12/24 | 13/12/24 |
| Planteamiento del 4to Cap | 12/12/24 | 27/12/24 |
| Propuesta | 30/12/24 | 8/1/25 |
| Revision general de Proyecto | 9/1/25 | 13/1/25 |



# ANEXOS 2:

## Presupuesto ejecutado

**Tabla de presupuesto**

| Concepto | Cantidad | Tiempo | Descripción | Costo |
|---|---|---|---|---|
| **Tecnología hardware y software** | 1 | 4 meses | Portátil, internet, programas varios | 3000 |
| **Costos del investigador** | 1 | 4 meses | Costos varios necesarios para el investigador. | 3200 |
| **Costos operativos** | 1 | 4 meses | Transporte | 100 |
| **Costos de utilería** | 1 | 4 meses | Impresiones, anillado y empastado | 150 |
| **Contingente** | 1 | 4 meses | Imprevistos surgidos durante el desarrollo del trabajo de titulación | 400 |
| | **Total** | | | 6850 |

Fuente. Elaboración propia.

# ANEXOS 3:

## Instrumentos de recolección de datos (ficha de observación)

**Anexo 3.1: Fichas de observación de la herramienta SonarQube**

**Ficha de la herramienta Proyecto 1**

| FICHA DE OBSERVACION | |
|---|---|
| Fecha: 1-10-2024 | N° 1 |
| **Información General** | |
| **Observador** | Axel Huera |
| **Contexto de la Observación** | Evaluación de herramientas de análisis de seguridad de código en un proyecto de desarrollo de software. |
| **Objetivo de la Observación** | Determinar la eficiencia y rendimiento de las herramientas de análisis de seguridad en la detección de vulnerabilidades en la primera prueba. |
| **Características Técnicas** | |
| **Herramienta de Análisis** | SonarQube |
| **Versión** | 10.8 |
| **Tipo de análisis** | SAST |
| **Plataformas compatibles** | Windows, Linux, MAC |
| **Lenguajes de programación a analizar** | php, js, html, css |
| **Eficiencia en la Detección de Vulnerabilidades** | |
| **Número de vulnerabilidades detectadas** | 165 |
| **Tipo de vulnerabilidades detectadas** | Security, 1<br>Reliability, 116<br>Maintainability, 35<br>Security Hotspots, 13 |
| **Rendimiento** | |
| **Consumo de recursos** | |
| **Tiempo de análisis** | 26,405 segundos |
| **CPU** | 1,20% |
| **RAM** | 33,25 MB |

**Ficha de observación Proyecto 2**

| FICHA DE OBSERVACION | |
|---|---|
| Fecha: 8-10-2024 | N° 2 |
| **Información General** | |
| **Observador** | Axel Huera |
| **Contexto de la Observación** | Evaluación de herramientas de análisis de seguridad de código en un proyecto de desarrollo de software. |
| **Objetivo de la Observación** | Determinar la eficiencia y rendimiento de las herramientas de análisis de seguridad en la detección de vulnerabilidades en la primera prueba. |
| **Características Técnicas** | |
| **Herramienta de Análisis** | SonarQube |
| **Versión** | 10.8 |
| **Tipo de análisis** | SAST |
| **Plataformas compatibles** | Windows, Linux, MAC |
| **Lenguajes de programación a analizar** | php, js, html, css |
| **Eficiencia en la Detección de Vulnerabilidades** | |
| **Número de vulnerabilidades detectadas** | 299 |
| **Tipo de vulnerabilidades detectadas** | Security, 2<br>Reliability, 164<br>Maintainability, 104<br>Security Hotspots, 29 |
| **Rendimiento** | |
| **Consumo de recursos** | |
| **Tiempo de análisis** | 36.880 segundos |
| **CPU** | 0,51% |
| **RAM** | 33,5 MB |

**Anexo 3.2: Ficha de observación de la herramienta Semgrep**

**Ficha de observación Proyecto 1**

| FICHA DE OBSERVACION | |
|---|---|
| Fecha: 30-07-2024 | N° 3 |
| **Información General** | |
| **Observador** | Axel Huera |
| **Contexto de la Observación** | Evaluación de herramientas de análisis de seguridad de código en un proyecto de desarrollo de software. |
| **Objetivo de la Observación** | Determinar la eficiencia y rendimiento de las herramientas de análisis de seguridad en la detección de vulnerabilidades en la primera prueba. |
| **Características Técnicas** | |
| **Herramienta de Análisis** | Semgre |
| **Versión** | 10.6 |
| **Tipo de análisis** | SAST |
| **Plataformas compatibles** | Windows, Linux, MAC |
| **Lenguajes de programación a analizar** | php, js, html, css |
| **Eficiencia en la Detección de Vulnerabilidades** | |
| **Número de vulnerabilidades detectadas** | 183 |
| **Categoría de vulnerabilidades detectadas** | cadena-sql-contaminada<br>etiqueta-eco-insegura-contaminada<br>nombre-archivo-contaminado<br>uso-desvinculación<br>Rendimiento |
| **Consumo de recursos** | |
| **Tiempo de análisis** | 2:00 minutos |
| **CPU** | 1,36% |
| **RAM** | 282,36 MB |

**Ficha de observación Proyecto 2**

| FICHA DE OBSERVACION | |
|---|---|
| Fecha: 30-07-2024 | N° 4 |
| **Información General** | |
| **Observador** | Axel Huera |
| **Contexto de la Observación** | Evaluación de herramientas de análisis de seguridad de código en un proyecto de desarrollo de software. |
| **Objetivo de la Observación** | Determinar la eficiencia y rendimiento de las herramientas de análisis de seguridad en la detección de vulnerabilidades en la primera prueba. |
| **Características Técnicas** | |
| **Herramienta de Análisis** | Semgre |
| **Versión** | 10.6 |
| **Tipo de análisis** | SAST |
| **Plataformas compatibles** | Windows, Linux, MAC |
| **Lenguajes de programación a analizar** | php, js, html, css |
| **Eficiencia en la Detección de Vulnerabilidades** | |
| **Número de vulnerabilidades detectadas** | 52 |
| **Categoría de vulnerabilidades detectadas** | método de documento inseguro<br>cadena sql contaminada<br>inyección de propiedad remota<br>inyección sql nativa de laravel<br>etiqueta de echo contaminada e insegura<br>bucle de contaminación de prototipo<br>enlace http de texto sin formato<br>detectar expresión regular no literal<br>eval detectado |
| **Rendimiento** | |
| **Consumo de recursos** | |
| **Tiempo de análisis** | 3:09 minutos |
| **CPU** | 1,30% |
| **RAM** | 256 MB |

**Anexo 3.3: Ficha de observación de la herramienta OWASP ASST**

**Ficha de observación Proyecto 1**

| **FICHA DE OBSERVACION** | |
|---|---|
| **Fecha:** | **N° 5** |
| **Información General** | |
| **Observador** | Axel Huera |
| **Contexto de la Observación** | Evaluación de herramientas de análisis de seguridad de código en un proyecto de desarrollo de software. |
| **Objetivo de la Observación** | Determinar la eficiencia y rendimiento de las herramientas de análisis de seguridad en la detección de vulnerabilidades en la primera prueba. |
| **Características Técnicas** | |
| **Herramienta de Análisis** | OWASP ASST |
| **Versión** | 1.0 |
| **Tipo de análisis** | SAST |
| **Plataformas compatibles** | Windows, Linux, MAC |
| **Lenguajes de programación a analizar** | php, js, html, css |
| **Eficiencia en la Detección de Vulnerabilidades** | |
| **Número de vulnerabilidades detectadas** | 175 |
| **Categoría de vulnerabilidades detectadas** | Injection=43<br>Authentication=13<br>Data Exposure=6<br>Security config=2<br>XSS=76<br>CFRS=4<br>Otros=31 |
| **Rendimiento** | |
| **Consumo de recursos** | |
| **Tiempo de análisis** | 5 segundos |
| **CPU** | 0,10% |
| **RAM** | 38,6 MB |

**Ficha de observación Proyecto 2**

| FICHA DE OBSERVACION | |
|---|---|
| **Fecha:** | **N° 6** |
| **Información General** | |
| **Observador** | Axel Huera |
| **Contexto de la Observación** | Evaluación de herramientas de análisis de seguridad de código en un proyecto de desarrollo de software. |
| **Objetivo de la Observación** | Determinar la eficiencia y rendimiento de las herramientas de análisis de seguridad en la detección de vulnerabilidades en la primera prueba. |
| **Características Técnicas** | |
| **Herramienta de Análisis** | OWASP ASST |
| **Versión** | 1.0 |
| **Tipo de análisis** | SAST |
| **Plataformas compatibles** | Windows, Linux, MAC |
| **Lenguajes de programación a analizar** | php, js, html, css |
| **Eficiencia en la Detección de Vulnerabilidades** | |
| **Número de vulnerabilidades detectadas** | 185 |
| **Categoría de vulnerabilidades detectadas** | Injection =29<br>Authentication=16<br>Data Exposure=23<br>Security config=2<br>XSS=78<br>CFRS=16<br>Otros=21 |
| **Rendimiento** | |
| **Consumo de recursos** | |
| **Tiempo de análisis** | 6 segundos |
| **CPU** | 0,20% |
| **RAM** | 48,1MB |

**Anexo 3.4: Fichas de observación de la herramienta HCL AppScan CodeSweep**

**Anexo Ficha de observación proyecto 1**

| FICHA DE OBSERVACION | |
|---|---|
| Fecha: 12-12-2024 | N° 7 |
| **Información General** | |
| **Observador** | Axel Huera |
| **Contexto de la Observación** | Evaluación de herramientas de análisis de seguridad de código en un proyecto de desarrollo de software. |
| **Objetivo de la Observación** | Determinar la eficiencia y rendimiento de las herramientas de análisis de seguridad en la detección de vulnerabilidades en la primera prueba. |
| **Características Técnicas** | |
| **Herramienta de Análisis** | HCL AppScan CodeSweep |
| **Versión** | Extensión VSC |
| **Tipo de análisis** | SAST |
| **Plataformas compatibles** | Windows, Linux, MAC |
| **Lenguajes de programación a analizar** | php, js, html, css |
| **Eficiencia en la Detección de Vulnerabilidades** | |
| **Número de vulnerabilidades detectadas** | 74 |
| **Categoría de vulnerabilidades detectadas** | JavaScript - Insecure Use of InnerHTML or OuterHTML<br>JQuery - JQuery HTML Function IJse<br>JQuery - JQuery Potentially Setting An Attribute With Tainted Data<br>PHP- Anchor generated with potential user-controlled data<br>PHP- Potential path traversal through variable argument<br>PHP- Potential user-controlled data within PHP converted to HTML<br>PHP- Potentially hazardous query detected: variable substitution in-line with query text<br>PHP- Potential header injection discovered<br>PHP- Open communications scheme detected |
| **Rendimiento** | |
| **Consumo de recursos** | |
| **Tiempo de análisis** | 5 min |
| **CPU** | 0,19% |
| **RAM** | 48,6 MB |

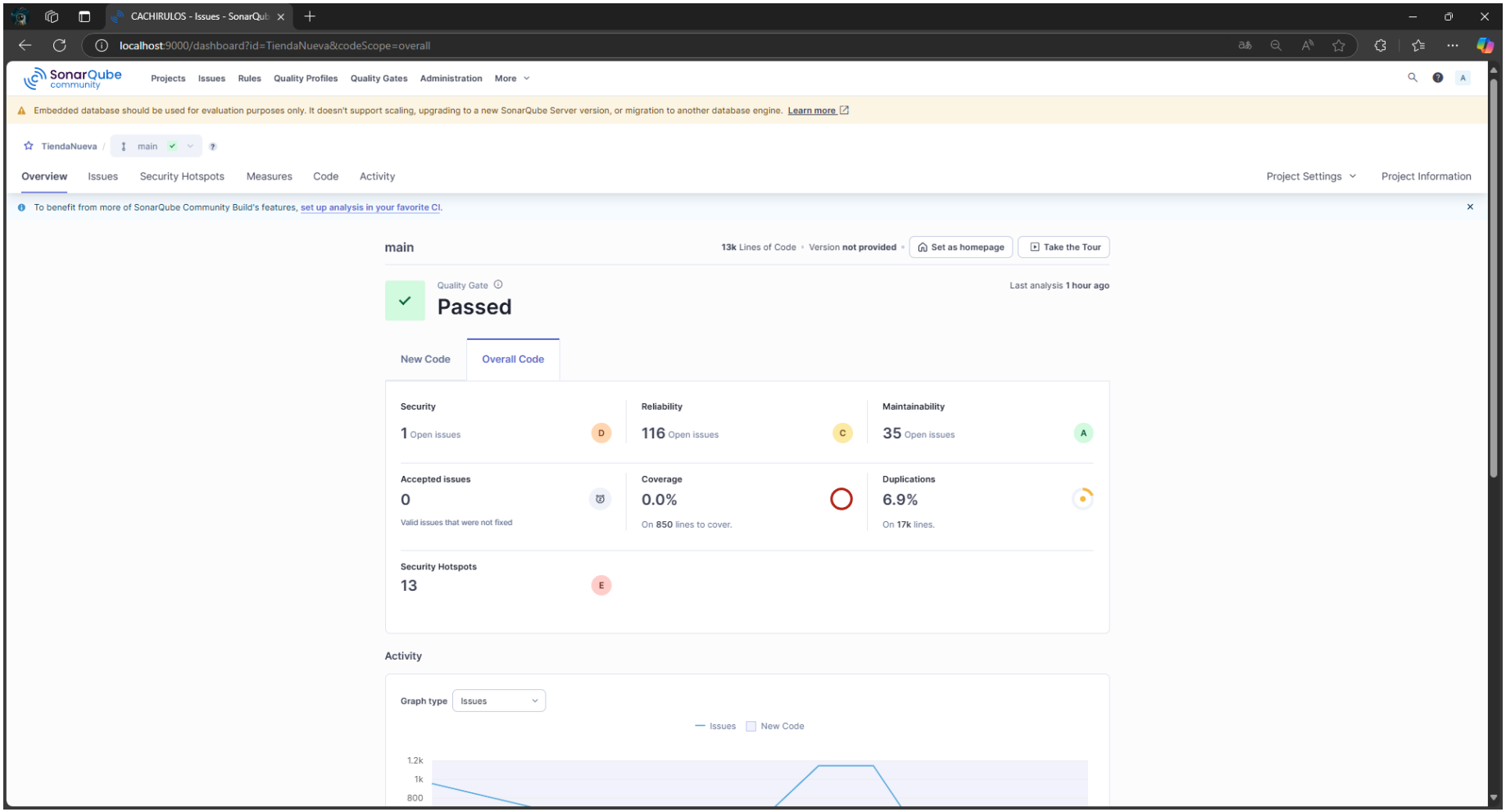
**Ficha de observación Proyecto 2**

| FICHA DE OBSERVACION | |
|---|---|
| Fecha: 23-12-2024 | N° 8 |
| **Información General** | |
| **Observador** | Axel Huera |
| **Contexto de la Observación** | Evaluación de herramientas de análisis de seguridad de código en un proyecto de desarrollo de software. |
| **Objetivo de la Observación** | Determinar la eficiencia y rendimiento de las herramientas de análisis de seguridad en la detección de vulnerabilidades en la primera prueba. |
| **Características Técnicas** | |
| **Herramienta de Análisis** | HCL AppScan CodeSweep |
| **Versión** | Extensión VSC |
| **Tipo de análisis** | SAST |
| **Plataformas compatibles** | Windows, Linux, MAC |
| **Lenguajes de programación a analizar** | php, js, html, css |
| **Eficiencia en la Detección de Vulnerabilidades** | |
| **Número de vulnerabilidades detectadas** | 84 |
| **Categoría de vulnerabilidades detectadas** | JavaScript - Insecure Use of InnerHTML or OuterHTML<br>JQuery - JQuery HTML Function Use<br>JQuery - JQuery Potentially Setting An Attribute With Tainted Data<br>JQuery - Potential XSS vulnerability detected in jQuery.appendO method<br>PHP - Potential user-controlled data within PHP converted to HTML<br>PHP - Potentially hazardous query detected: variable substitution in-line with query text<br>VueJS - HTML Injection using InnerHTML in VueJS<br>Angular - Direct DOM Manipulation Through (element)<br>JavaScript - Allowing untrusted Site by passing user controlled input<br>JQuery - JQuery AJAX Error Handler Could Leak Information<br>PHP - Open communications scheme detected |
| **Rendimiento** | |
| **Consumo de recursos** | |
| **Tiempo de análisis** | 8 min |
| **CPU** | 0,35% |
| **RAM** | 72,4 MB |

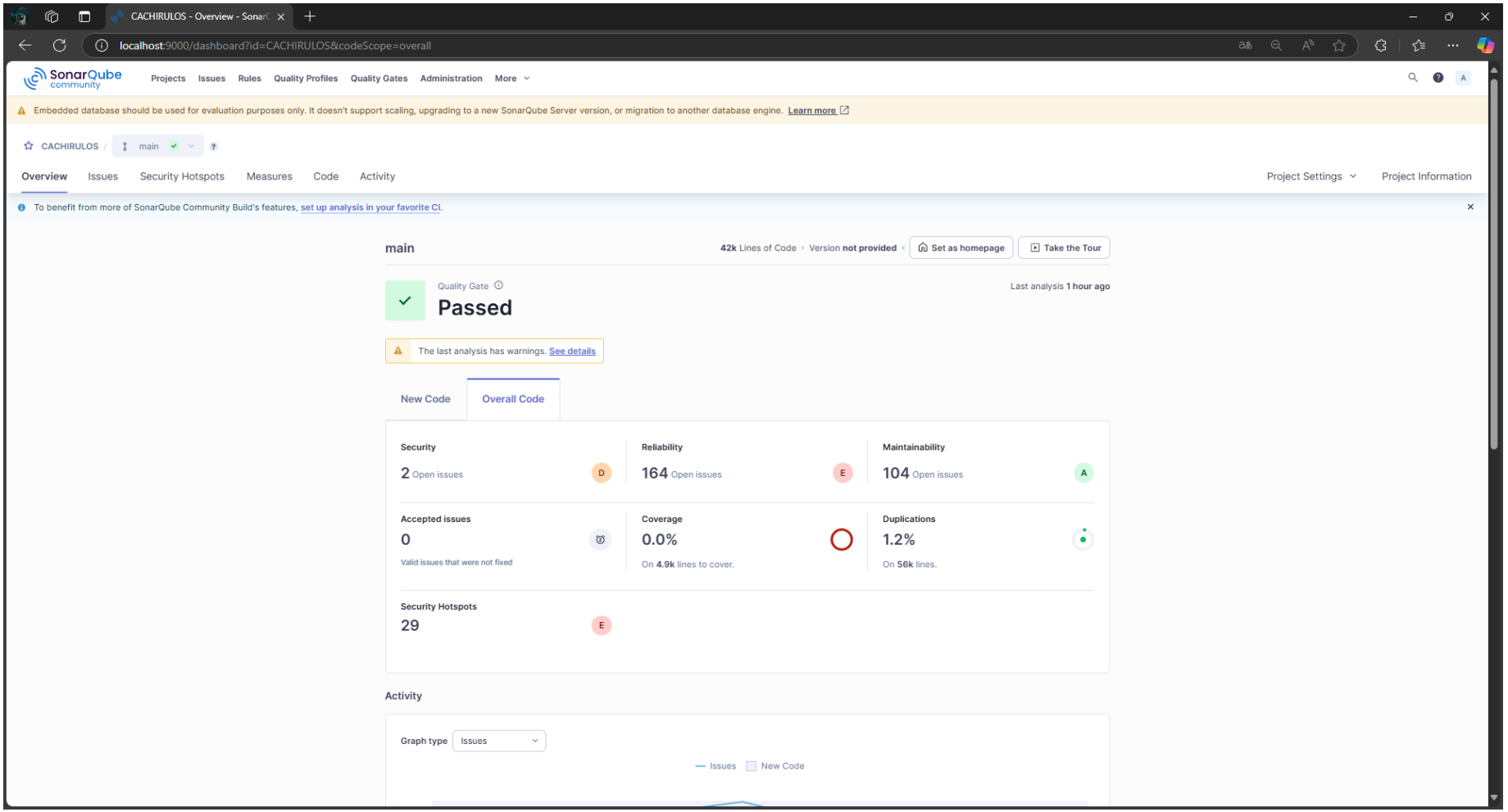
# ANEXOS 4:

## Capturas de pantalla de los reportes de cada herramienta

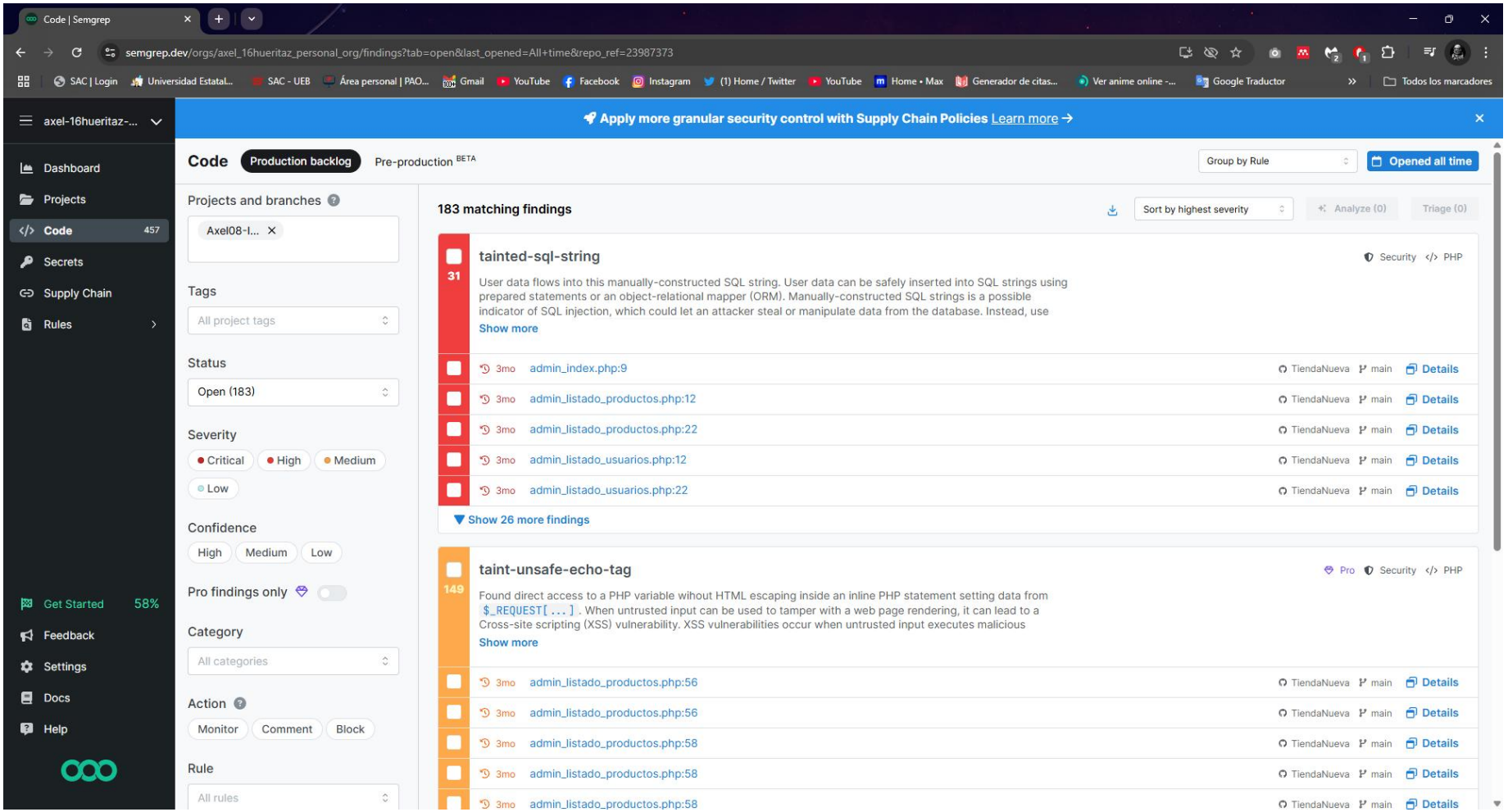
**Anexo 4.1:** Capturas de pantalla SonarQube: proyecto 1



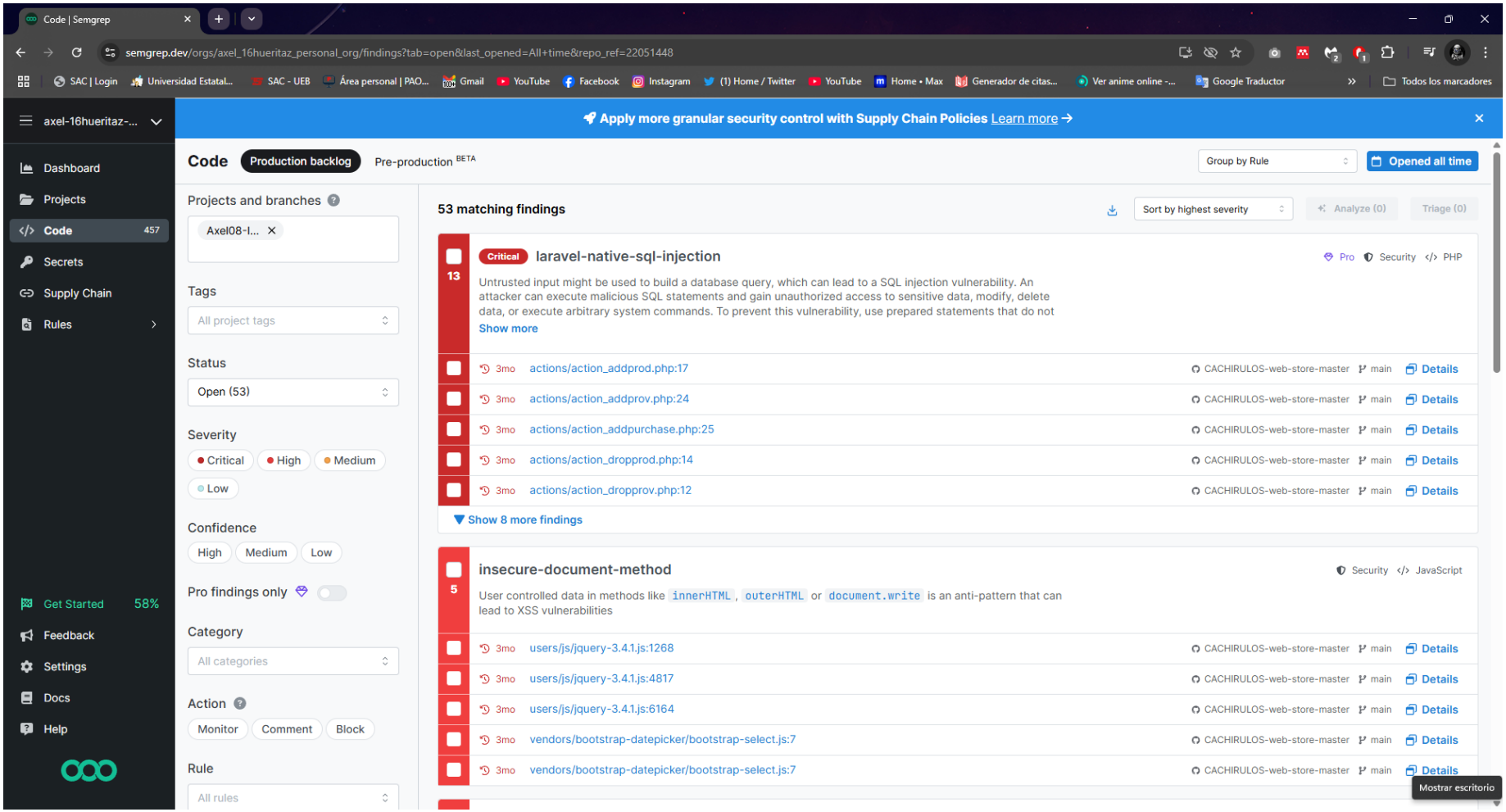
**Anexo 4.2:** Capturas de pantalla SonarQube: proyecto 2



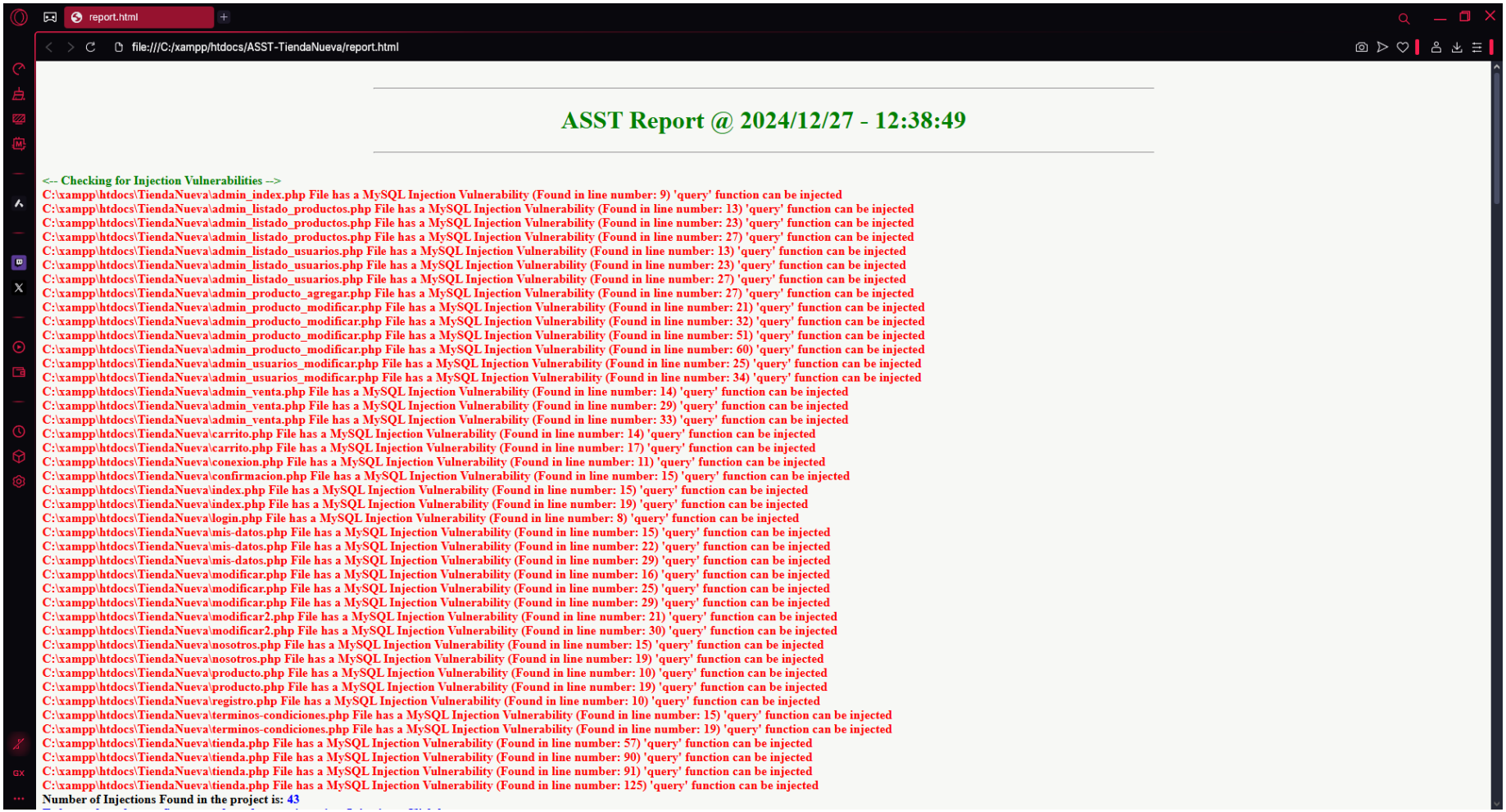
**Anexo 4.3:** Capturas de pantalla Semgrep: proyecto 1



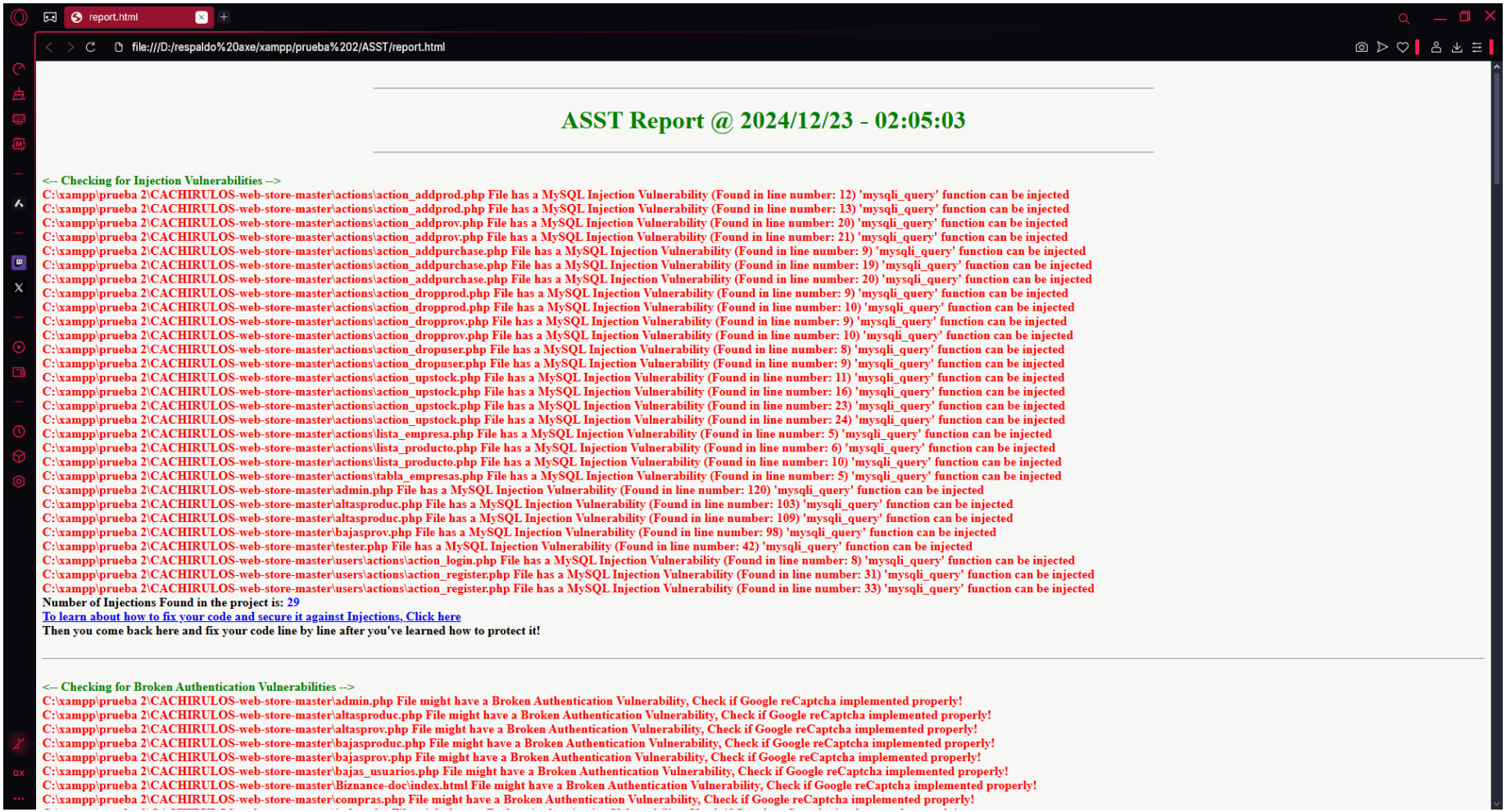
**Anexo 4.4:** Capturas de pantalla Semgrep: proyecto 2



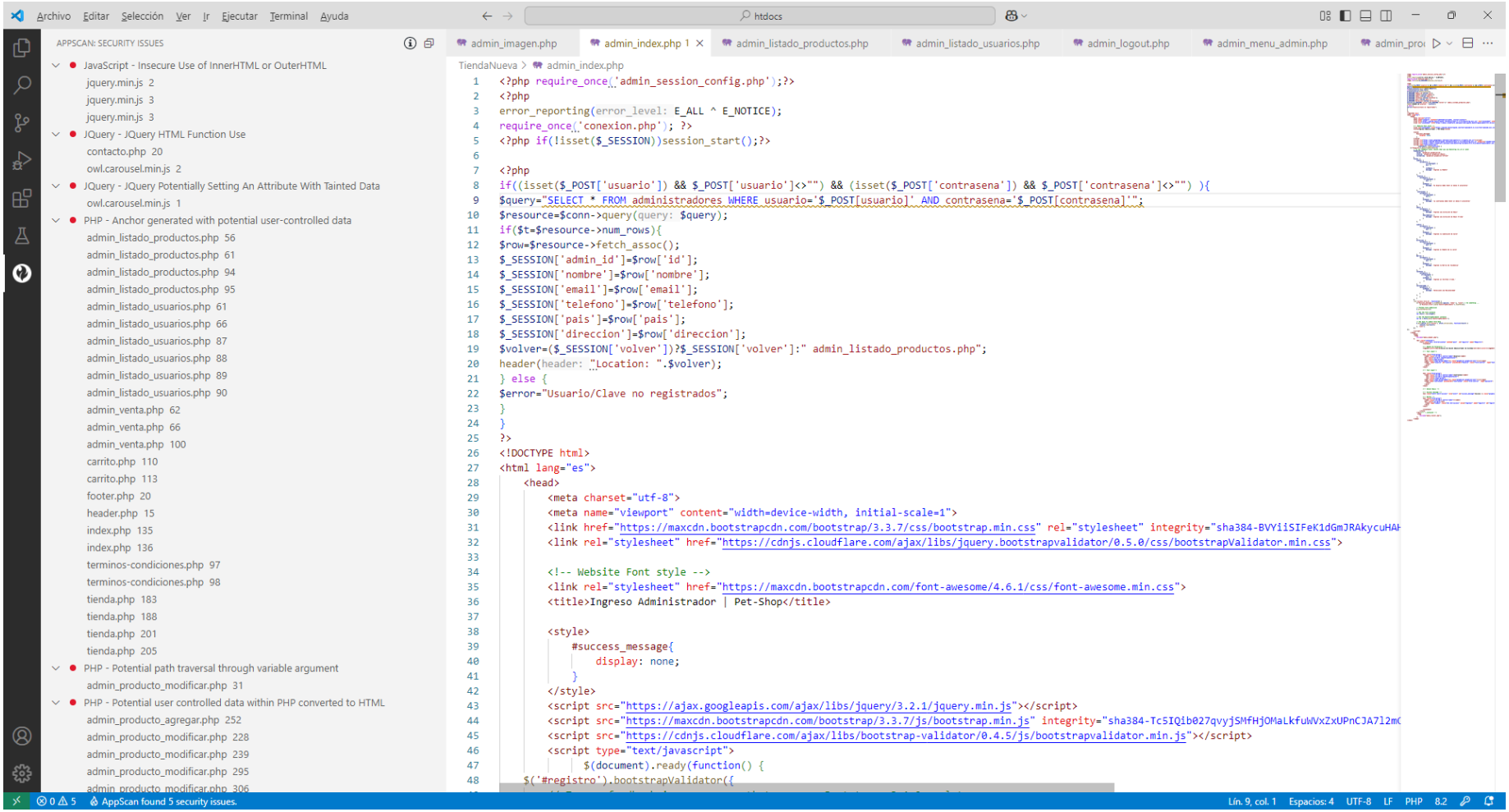
**Anexo 4.5:** Capturas de pantalla OWASP ASST: proyecto 1



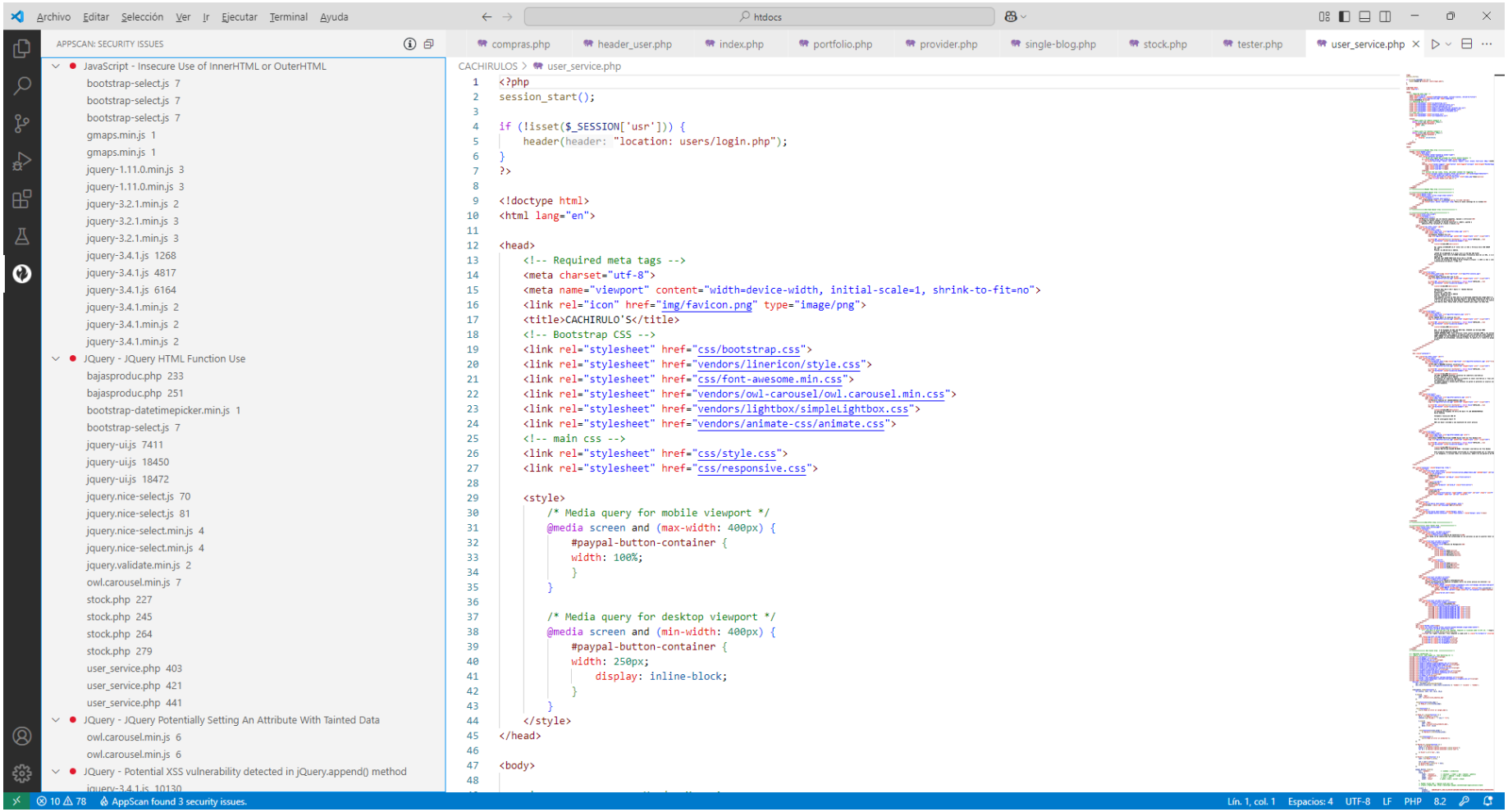
**Anexo 4.6:** Capturas de pantalla OWASP ASST: proyecto 2



**Anexo 4.7:** Captura de pantalla del reporte generado por HCL AppScan: proyecto 1



**Anexo 4.8:** Captura de pantalla del reporte generado por HCL AppScan: proyecto 2



# ANEXOS 4:

## Certificado Antiplagio

**ING. CHRISTIAN BARRAGÁN EN CALIDAD DE DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR,**

# CERTIFICA

Que el trabajo de integración curricular denominado "H**erramientas de análisis de código seguro para el uso de desarrollo de software, año 2024**", presentado por Axel Ismael Huera Quinatoa estudiante de la **carrera de Software** pasó el análisis de coincidencia no accidental en la herramienta TURNITIN, reflejando un **porcentaje de similitud del 5%**, como se puede evidenciar en el documento adjunto.

Guaranda, 21 de marzo de 2025

Atentamente,




Ing. Christian Barragán
**Director**

# Axel Huera

## TesisAxelcorregido.docx

My Files

My Files

Universidad Estatal de Bolivar

### Detalles del documento

Identificador de la entrega
**trn:oid:::3117:441366369**

Fecha de entrega
**21 mar 2025, 11:39 a.m. GMT-5**

Fecha de descarga
**21 mar 2025, 11:48 a.m. GMT-5**

Nombre de archivo
**TesisAxelcorregido.docx**

Tamaño de archivo
**11.3 MB**

**132 Páginas**

**18.250 Palabras**

**106.468 Caracteres**

# 5% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

## Filtered from the Report

- Bibliography
- Quoted Text
- Small Matches (less than 15 words)

## Exclusions

- 5 Excluded Sources
- 15 Excluded Matches

## Top Sources

5% Internet sources
0% Publications
4% Submitted works (Student Papers)

## Integrity Flags

**0 Integrity Flags for Review**

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

# ANEXOS 5: Link del repositorio digital de biblioteca donde fue subido el proyecto