Articulo de revisión

Modelos de Pruebas de Seguridad Estática en Reducción de Ineficiencia identificación de Inyección SQL en Aplicaciones Web

Static Security Testing Models in Inefficiency Reduction Identification of SQL Injection in Web Applications

Armando Tipacti Garcia[1] https://orcid.org/0009-0006-3067-0445

[1]Facultad de Ingeniería de Sistemas e Informática - Unidad de Postgrado, Universidad Nacional Mayor de San Marcos UNMSM | Lima – Perú | CP 15730

armando.tipacti@unmsm.edu.pe

http://doi.org/10.26423/rctu.v11i2.800
Páginas: 130- 144

Resumen

La detección temprana de vulnerabilidades es crucial en el desarrollo de software para garantizar la seguridad de las aplicaciones Web, especialmente frente a ataques de inyección SQL. Las pruebas de Seguridad Estática (SAST) permiten identificar vulnerabilidades desde las primeras etapas del ciclo de vida del desarrollo. Este artículo revisa sistemáticamente la literatura para identificar y analizar los modelos de SAST más eficaces en reducir ineficiencias en la detección de inyecciones SQL. Siguiendo las guías PRISMA 2020 y el enfoque de Kitchenham, se realizaron búsquedas exhaustivas en bases de datos como EBSCO y Scopus. Los resultados muestran que la integración temprana de SAST y el uso de inteligencia artificial mejoran la detección de vulnerabilidades, reduciendo falsos positivos y negativos. La implementación de modelos avanzados de SAST es esencial para mejorar la seguridad de las aplicaciones Web, sugiriéndose investigaciones futuras en metodologías más integradas y nuevas herramientas.

Palabras clave: Pruebas Seguridad Estática, Desarrollo de Software Seguro, DevSecOps, Inyección SQL.

Abstract

Early detection of vulnerabilities is crucial in software development to ensure the security of Web applications, especially against SQL injection attacks. Static Application Security Testing (SAST) allows for the identification of vulnerabilities from the early stages of the development lifecycle. This article systematically reviews the literature to identify and analyze the most effective SAST models in reducing inefficiencies in detecting SQL injections. Following PRISMA 2020 guidelines and Kitchenham's approach, exhaustive searches were conducted in databases like EBSCO and Scopus. The results show that early integration of SAST and the use of artificial intelligence significantly improve vulnerability detection, reducing false positives and negatives. The implementation of advanced SAST models is essential for enhancing the security of Web applications, with future research suggested to explore more integrated methodologies and new tools.

Keywords: Static application security testing, Secure software development, DevSecOps, SQL Inyection.

Recepción: 25/06/2024 | Aprobación: 06/11/2024 | Publicación: 26/12/2024

1. Introducción

En la última década, el desarrollo de aplicaciones Web ha experimentado un crecimiento exponencial, convirtiéndose en un pilar esencial de la economía digital y de la interacción social [1]. A medida que más empresas y organizaciones dependen de estas aplicaciones para realizar operaciones críticas, la seguridad de las mismas se ha convertido en una preocupación primordial [2]. Una de las amenazas más prevalentes en este ámbito es la inyección SQL, una vulnerabilidad que permite a los atacantes manipular consultas en bases de datos para obtener acceso no autorizado, comprometiendo así la integridad, disponibilidad y confidencialidad de la información [3].

La importancia de proteger las aplicaciones Web de estas amenazas ha impulsado a la comunidad científica y a la industria a desarrollar diversas herramientas y métodos de pruebas de seguridad [4]. Entre ellas, las Pruebas de Seguridad Estática (SAST) se destacan por su capacidad para identificar vulnerabilidades en las primeras etapas del ciclo de vida del desarrollo de software [5]. A diferencia de las pruebas dinámicas, SAST analiza el código fuente sin ejecutarlo, lo que permite a los desarrolladores detectar y corregir errores de seguridad antes de que el software sea desplegado en un entorno de producción [3].

El desarrollo de métodos avanzados, como el uso de inteligencia artificial y aprendizaje automático en SAST, ha demostrado ser fundamental para mejorar la detección de vulnerabilidades [6]. Estudios recientes destacan cómo la integración de estos enfoques en SAST no solo aumenta la precisión de la detección de fallas de seguridad, sino que también reduce significativamente los falsos positivos y negativos [7]. Esta capacidad para mejorar la eficiencia en la identificación de vulnerabilidades es crucial en un entorno donde la seguridad del software es cada vez más amenazada por ataques sofisticados [2].

Sin embargo, aunque la combinación de estas técnicas ha mostrado mejoras, persisten desafíos significativos en la implementación práctica de SAST. La literatura destaca que, a pesar de las innovaciones, muchos equipos de desarrollo todavía luchan con la integración efectiva de SAST en sus flujos de trabajo, particularmente en entornos ágiles y DevSecOps [8]. Esto se debe en parte a la necesidad de una mayor personalización de las herramientas SAST para adaptarse a las especificidades de cada proyecto y al hecho de que las tecnologías emergentes a menudo superan la capacidad de las herramientas existentes para mantenerse al día con las nuevas amenazas y vulnerabilidades [1].

A pesar de estos avances, la literatura actual señala que las herramientas SAST aún enfrentan desafíos importantes [8]. Estos desafíos incluyen la falta de integración con otros procesos de desarrollo seguro y la adaptación a nuevas tecnologías y lenguajes de programación [5]. Además, existe un debate sobre la eficiencia y precisión de estas herramientas en comparación con otros enfoques, como las pruebas de seguridad dinámica (DAST) y las pruebas interactivas (IAST) [1].

Un aspecto no completamente abordado en la literatura es la identificación de los modelos de pruebas de seguridad estática más efectivos para la detección de inyecciones SQL en aplicaciones Web [2]. Algunos estudios sugieren que la efectividad de SAST depende en gran medida de la configuración de la herramienta y del contexto en el que se aplique, lo que subraya la necesidad de investigaciones más detalladas y contextualizadas [3]. Además, es crucial entender los factores determinantes que influyen en la eficiencia de estas pruebas, como la calidad del código fuente, la experiencia del equipo de desarrollo y la capacidad de las herramientas SAST para adaptarse a metodologías ágiles o DevSecOps [6].

La reducción de falsos positivos y negativos es una de las áreas más investigadas en la mejora de SAST [5]. Varios estudios han demostrado cómo la aplicación de técnicas de aprendizaje automático y procesamiento del lenguaje natural puede mejorar la precisión de la detección de vulnerabilidades, reduciendo significativamente la tasa de falsos positivos [3]. Además, se ha identificado que la combinación de SAST con otros métodos, como el análisis dinámico y el modelado de amenazas, puede mejorar la cobertura y precisión de las pruebas, optimizando los recursos y el tiempo invertido en la seguridad del software [5].

Este estudio tiene como objetivo realizar una revisión sistemática de la literatura para identificar y analizar los modelos de pruebas de seguridad estática más adecuados para reducir la ineficiencia en la detección de vulnerabilidades de inyección SQL en aplicaciones Web. A través de esta revisión, se busca:

1. Evaluar la evidencia disponible que respalda la efectividad de los modelos de pruebas de seguridad estática en la reducción de la ineficiencia en la detección de inyecciones SQL.
2. Identificar los factores determinantes que influyen en la efectividad de las pruebas SAST, incluyendo tanto factores técnicos como humanos.
3. Determinar cuáles modelos específicos de pruebas de seguridad estática han sido desarrollados y evaluados en la literatura para abordar estas ineficiencias.

El resultado de esta revisión no solo pretende llenar los vacíos de conocimiento existentes, sino también proporcionar una base sólida para futuras investigaciones y aplicaciones prácticas en el desarrollo seguro de software. Las conclusiones ofrecerán una orientación crítica sobre cómo mejorar la eficiencia y efectividad de las pruebas de seguridad estática, fortaleciendo la seguridad en el desarrollo de aplicaciones Web en un entorno cada vez más amenazado por ataques sofisticados.

2. Metodología

En esta revisión sistemática, se empleó la guía PRISMA (2020) [9] junto con el enfoque sugerido por Barbara Kitchenham[10]. Este enfoque se centra en la minuciosidad y transparencia del proceso, asegurando una revisión imparcial y confiable al definir un protocolo detallado que establece los objetivos, las preguntas de investigación y los criterios de inclusión y exclusión de estudios. Después de esta etapa inicial, se procede con una exploración de la literatura en una amplia variedad de fuentes, seguida de una selección meticulosa de estudios pertinentes de acuerdo con los parámetros establecidos. Además, una vez que se eligen los estudios, se recopilan datos relevantes mediante un formulario estructurado. Se lleva a cabo una evaluación

de la calidad metodológica de los estudios incluidos para garantizar la fiabilidad de la evidencia. Posteriormente, se sintetizan y presentan de manera coherente los resultados de los estudios, lo que facilita una comprensión profunda del tema de investigación.

La aplicación del enfoque de Kitchenham (2004) [10] proporciona un fundamento robusto para tomar decisiones informadas y para investigaciones posteriores en el ámbito de la ingeniería de software. En esta revisión sistemática, seguiremos este método para garantizar la imparcialidad y la fiabilidad de nuestros descubrimientos, lo que contribuirá al progreso del conocimiento en esta área. Las etapas detalladas se presentan en la Tabla 1.

Tabla 1: Método del proceso de revisión sistemática de Barbara Kitchenham.

	Fases	Actividad
I.	Realización de la Revisión	▪ Identificación de la necesidad para la revisión ▪ Desarrollo del protocolo de revisión
II.	Realización de la Revisión	▪ Identificación de los estudios ▪ Selección de los estudios ▪ Evaluación de la calidad de los estudios ▪ Extracción de los datos ▪ Síntesis de los datos
III.	Reporte de la Revisión	▪ Elaboración del reporte

Fase I. Planificación de la Revisión
Identificación de la necesidad para la revisión:

La identificación de la necesidad marca el inicio en el enfoque presentado por Kitchenham [10], que implica la comprensión de la relevancia de llevar a cabo una revisión sistemática sobre un tema de investigación específico.

En esta fase, el objetivo es aclarar y fundamentar la importancia y extensión de la revisión. Esto implica entender los motivos que respaldan la revisión de la literatura disponible, identificar las preguntas de investigación a tratar y evaluar cómo la revisión contribuirá al avance del conocimiento en el campo temático en cuestión. Como parte de la identificación de la necesidad, se formularon las siguientes interrogantes:

1. ¿Cuál es la evidencia disponible que respalda la efectividad de los modelos de pruebas de seguridad estática en la reducción de la ineficiencia en la detección de Inyecciones SQL en aplicaciones Web?
2. ¿Cuáles son los factores determinantes que influyen en la reducción de la ineficiencia en la identificación de Inyecciones SQL en aplicaciones Web?
3. ¿Qué modelos específicos de pruebas de seguridad estática se han desarrollado para abordar la ineficiencia en la detección de Inyecciones SQL en aplicaciones Web?

Desarrollo del protocolo de revisión:

A continuación, se detallarán los pasos definidos como parte del protocolo de la revisión sistemática, divididos en: identificación de palabras clave, bases de datos electrónicos y criterios de selección (inclusión y exclusión).

Para la identificación de palabras clave, se utilizó las frases: Static application security testing, Static code analysis, Source code security analysis, Static analysis tools, Static vulnerability analysis.

Asimismo, las bases de datos electrónicos elegidas para esta revisión son EBSCO y Scopus. Para los criterios de selección, se dividió en dos grupos: criterios de inclusión y exclusión. Para los criterios de inclusión se consideró que los artículos tengan como máximo 5 años de antigüedad, que estén en idioma inglés y español, que sean estudios primarios y revisiones publicados en revistas científicas e investigaciones que se centren específicamente en modelos de pruebas de seguridad estática, así como estar realizados en diferentes países y contextos geográficos. Por otro lado, para los criterios de exclusión se consideró que los artículos no estén disponibles como Acceso Abierto o en forma de texto completo, investigaciones duplicadas o redundantes.

Fase II. Realización de la Revisión
A continuación, se describirá a detalle los pasos realizados como parte del desarrollo de la revisión sistemática.

Identificación de los estudios:

Luego de aplicar los criterios de inclusión y exclusión, se obtuvieron 228 artículos, tal como se muestra en la Tabla 2:

Tabla 2: Primera selección de estudios.

Base de datos	Cadena de búsqueda	Aplicando inclusiones	Aplicando exclusiones
EBSCO	("Static application security testing.ᴼR SAST OR "Static code analysis.ᴼR "Source code security analysis.ᴼR "Static security testing.ᴼR "Static analysis tools.ᴼR "Secure code review.ᴼR "Static vulnerability analysis.ᴼR "Static security scanning.ᴼR DevSecOps) AND (application OR software OR "web applications.ᴼR "web apps")	227	85
SCOPUS	("Static application security testing.ᴼR SAST OR "Static code analysis.ᴼR "Source code security analysis.ᴼR "Static security testing.ᴼR "Static analysis tools.ᴼR "Secure code review.ᴼR "Static vulnerability analysis.ᴼR "Static security scanning.ᴼR DevSecOps) AND (application OR software OR "web applications.ᴼR "web apps")	236	143
TOTAL		463	228

Selección de los estudios:

Luego de identificar los estudios, se realizó el procedimiento de selección (ver Figura 1) considerando como guía el que se encuentra en el artículo "Gamificación en la enseñanza de las matemáticas: una revisión sistemática" de Holguín [11] y se obtuvo la lista final de 41 artículos, tal como se muestra en la Tabla 3.

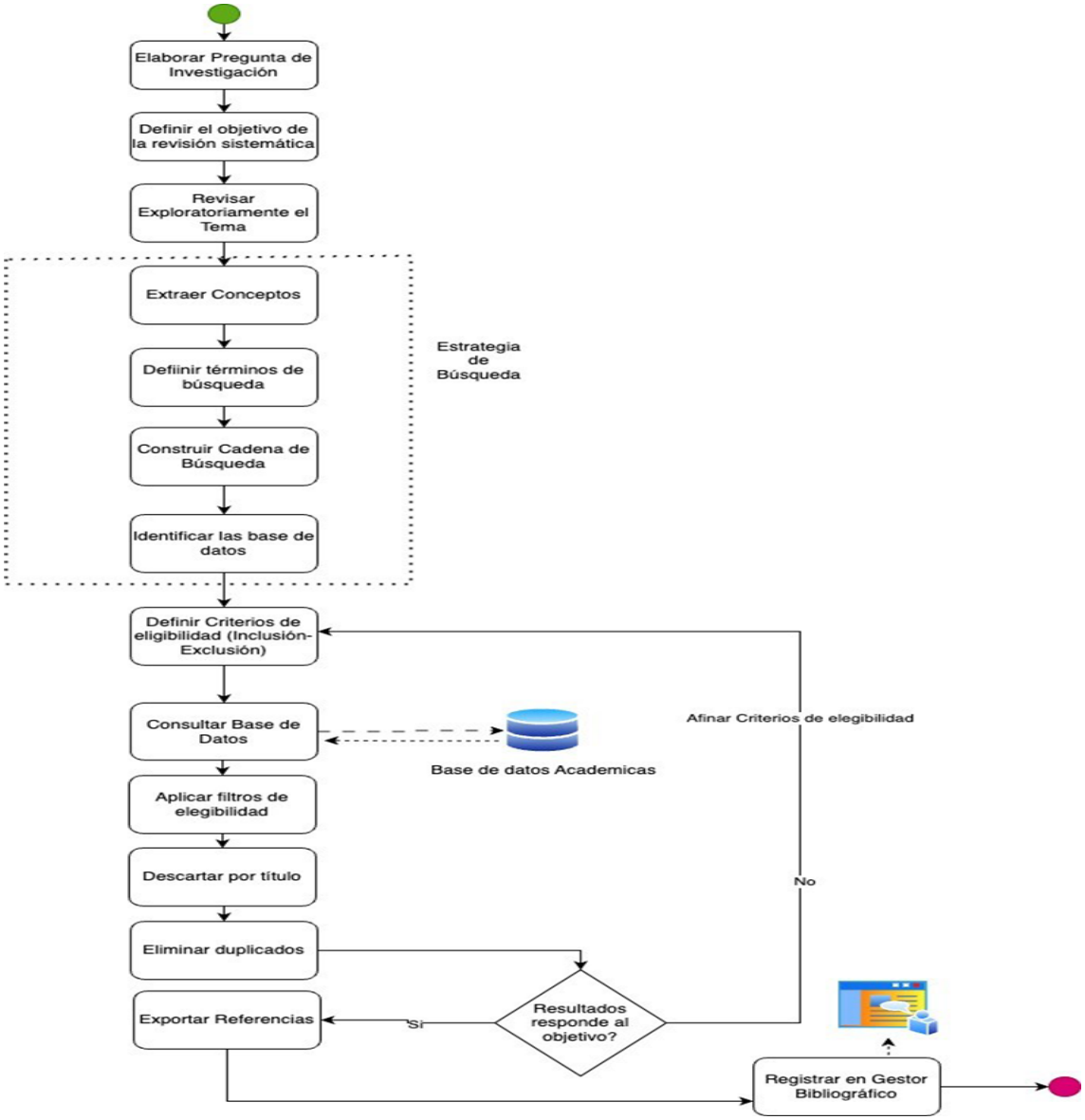

Figura 1. Procedimiento de selección para la revisión sistemática.

Tabla 3: Selección final de estudios .

Base de datos	Pregunta resuelta
EBSCO	14
Scopus	27
Total	41

Evaluación de la calidad de los estudios:

Para obtener la lista final de artículos para la revisión mencionada que hace referencia a la tabla 3, se definió el instrumento de evaluación de calidad compuesto de 10.

1. ¿El estudio tiene un objetivo claro y bien definido?
2. ¿La metodología utilizada está claramente explicada y justificada?
3. ¿El estudio proporciona datos empíricos que respaldan la efectividad de los modelos de pruebas de seguridad estática?
4. ¿Los resultados muestran una mejora significativa en la detección de inyecciones SQL mediante el uso de modelos de pruebas de seguridad estática?
5. ¿El estudio identifica y discute factores específicos que influyen en la efectividad de las pruebas de seguridad estática?
6. ¿Se mencionan y evalúan factores técnicos y humanos que afectan la eficiencia de la detección de inyecciones SQL?

7. ¿El estudio incluye comparaciones con otros métodos de detección de vulnerabilidades?
8. ¿Los modelos específicos de pruebas de seguridad estática están claramente descritos y evaluados?
9. ¿El estudio incluye casos de estudio o aplicaciones prácticas que demuestran la efectividad de los modelos propuestos?
10. ¿Las conclusiones del estudio están bien fundamentadas en los datos y resultados presentados?

Los documentos que pasaron la evaluación de calidad proporcionaron una base sólida para explorar y comprender la efectividad de los modelos de desarrollo de software para reducir la ineficiencia identificación de Inyección SQL en Aplicaciones Web.

Extracción de los datos:

Para la selección de los estudios y extracción de sus datos se utilizó el aplicativo Excel de la suite Microsoft Office. Adicionalmente, se usó esta herramienta para la gestión de las referencias y de los criterios de elegibilidad y exclusión. Luego de seleccionar los estudios para esta investigación y evaluar la calidad de la información, se utilizó el formato de extracción de datos (ver Tabla 4) para desarrollar todas las ideas posibles para la revisión sistemática. Finalmente, se procedió a categorizar y ordenar los datos extraídos para su posterior análisis.

Tabla 4: Formato de extracción de datos.

Sección	Información asociada
Información básica	▪ Título ▪ Autor ▪ País ▪ Tipo de artículo ▪ Resumen del artículo
Análisis de la investigación	▪ Objeto de estudio
Análisis de los resultados de la investigación	▪ Ubicación geográfica de la investigación ▪ Modelos ▪ actores determinantes ▪ Evidenciass

Síntesis de los datos:

Para el proceso de síntesis de datos se realizó lo siguiente: Primero, posterior a la extracción de datos, se identificaron las ideas principales y secundarias de cada una de las fuentes seleccionadas. Segundo, se realizó la integración y relacionamiento de las ideas antes mencionadas para obtener la base necesaria para las fichas de síntesis. Para esto último, se hizo una revisión exhaustiva identificando la relación entre las ideas encontradas para generar nuevas ideas que permitan responder la pregunta de investigación. Tercero, se redactó un resumen breve y conciso con los aspectos realmente puntuales y se procuró el uso de oraciones cortas evitando redundar o dar información que no sea relevante. Por último, se completó la ficha de síntesis con toda la información recolectada y resumida de los puntos anteriores.

Fase III. Reporte de la Revisión
Elaboración del reporte:

La elaboraciópn del reporte se centra en la presentación detallada de los hallazgos y la evaluación de los estudios seleccionados. En esta etapa, se sintetizan los resultados obtenidos durante la revisión, identificando patrones, tendencias y brechas en la literatura. Los resultados de esta fase se reflejan directamente en las discusiones de la revisión sistemática, donde se analizan y contextualizan los modelos específicos de pruebas de seguridad estática, su efectividad en la detección de inyecciones SQL y los factores determinantes que influyen en la reducción de ineficiencias en aplicaciones Web. Este análisis permite validar la relevancia de los hallazgos y proporciona una base sólida para futuras investigaciones y mejoras en la seguridad del software.

3. Resultados y Discusión

Q1. ¿Cuál es la evidencia disponible que respalda la efectividad de los modelos de pruebas de seguridad estática en la reducción de la ineficiencia en la detección de Inyecciones SQL en aplicaciones Web?

La revisión sistemática realizada, abarca 41 artículos que evidencia la efectividad de los modelos de pruebas de seguridad estática (SAST). Para responder a la pregunta 1, se han clasificado las evidencias como se ve en la tabla 5.

Tabla 5: Artículos clasificados por tipos de evidencias.

Tipos de Evidencias	Cantidad de Artículos	Artículos
Reducción del tiempo de identificación de vulnerabilidades	12	[12] [7] [13] [4] [8] [14] [3] [15] [16] [17] [18] [2]
Reducción de falsos positivos y falsos negativos	9	[6] [19] [20] [21] [22] [23] [24] [25] [26]
Mejora en la precisión de la Detección de Vulnerabilidades	8	[27] [28] [29] [30] [31] [32] [33] [34]
Reducción de falsos positivos	5	[1] [35]
Mejora en la cobertura de detección de vulnerabilidades	2	[36] [5]
Reducción del costo de revisión de código	1	[37]
Mayor capacidad para integrar y gestionar	1	[38]
Reducción de falsos negativos	1	[39]
Adopción y escalabilidad en la Industria	1	[40]
Precisión en la identificación de deudas técnicas	1	[41]
Total	41	

Doce estudios se centraron en la "Reducción del tiempo de identificación de vulnerabilidades". Los estudios analizados de Schiewe *et al*. (2022) [3], Abeyrathna *et al*. (2020) [4], Bermejo *et al*. (2021) [8] y Alqaradaghi *et al*. (2024) [17], muestran que el uso de análisis estático y modelado de amenazas permite detectar y correlacionar fallas de seguridad arquitectónicas con errores a nivel de código, mejorando la eficiencia de la identificación de vulnerabilidades mediante automatización y conocimiento semántico. Asimismo, Lombardi y Fanton (2023) [12] y Lee y Liu (2023) [13] demuestran que la integración de prácticas de seguridad continua en pipelines DevSecOps, utilizando herramientas de análisis estático y dinámico, mejora significativamente la eficiencia del proceso, reduciendo el tiempo de detección de vulnerabilidades. Otro estudio como el de Piskachev *et al*. (2023) [16] destaca que la conFiguración adecuada de las herramientas SAST puede aumentar la efectividad en la resolución de vulnerabilidades al personalizar las opciones según el contexto del proyecto. En el artículo de Saurabh y Kumar (2024) [18] subrayan que el uso temprano de SAST en las fases iniciales de DevSecOps puede reducir significativamente el tiempo de ejecución del pipeline y mejorar la calidad del código.

En el estudio de Wijesiriwardana *et al*. (2020) [14] analiza la efectividad de las herramientas de análisis estático en la detección de vulnerabilidades de seguridad, destacando mejoras en la identificación de vulnerabilidades mediante la visualización tridimensional de la seguridad del código de software. El artículo de Schubert *et al*. (2022) [15] refuerza la importancia de implementar modelos SAST desde las primeras etapas del ciclo de vida del software para optimizar recursos y tiempo en la gestión de vulnerabilidades críticas como la Inyección SQL. Por último, estudios como el de Casola *et al*. (2024) [2] y Wijesiriwardana *et al*. (2023) [7] subrayan la necesidad de un enfoque basado en el conocimiento para modelar y detectar vulnerabilidades de seguridad en todas las fases del ciclo de vida del software. Estos hallazgos subrayan la importancia de adoptar un enfoque integrado y automatizado en la seguridad del software para mejorar su calidad y seguridad.

La "Reducción de falsos positivos y falsos negativos", es evidenciado por nueve estudios, Bakhshandeh *et al*. (2023) [6] demuestra que el uso de inteligencia artificial, como ChatGPT, puede reducir significativamente las tasas de falsos positivos y falsos negativos en la detección de vulnerabilidades de código. Yuan *et al*. (2023) [19] propone un método de detección estática para vulnerabilidades de inyección SQL basado en la transformación de programas, logrando una mejora notable en la precisión de la detección. Kuszczyński y Walkowski (2023) [20] destacan que la combinación de análisis estático y métodos de aprendizaje automático puede mejorar la exactitud de la detección de vulnerabilidades. Correa *et al*. (2021) [21] propone metodologías híbridas de evaluación de seguridad que combinan análisis estático y dinámico para reducir los falsos positivos y negativos en aplicaciones Web. Siewruk y Mazurczyk (2021) [22] destacan el uso de análisis de flujo de datos en contextos específicos para mejorar la detección de vulnerabilidades al tener en cuenta el contexto del software. Lomio *et al*. (2022) [23] realiza un análisis comparativo de reglas de SonarQube mediante aprendizaje automático y profundo, mostrando mejoras en la precisión de la detección de vulnerabilidades. Al-Johany *et al*. (2023) [24] investiga la predicción y corrección de defectos de software utilizando análisis estático y métodos de aprendizaje automático, destacando la reducción de falsos positivos y negativos.

Szabó y Bilicki (2023) [25] presentan un nuevo enfoque para la seguridad de aplicaciones Web mediante la integración de SAST y análisis dinámico para mejorar la precisión. Por último, Alqaradaghi *et al*. (2023) [26] desarrolla un verificador automático de análisis estático en su artículo, que muestra resultados óptimos en la detección de usos inseguros del SecurityManager en Java, reduciendo falsos positivos y negativos. Estos estudios confirman la efectividad de SAST en mejorar la precisión y eficiencia de la detección de vulnerabilidades en el software.

En cuanto a "Mejora en la precisión de la detección de vulnerabilidades", ocho estudios muestran cómo SAST ha optimizado la identificación de vulnerabilidades en el software. Sheneamer (2024) [27] analiza las vulnerabilidades en aplicaciones JavaScript utilizando redes neuronales convolucionales, demostrando una mejora significativa en la detección precisa de funciones vulnerables. Además, Filus y Domańska (2023) [28] investigan las vulnerabilidades en aplicaciones de aprendizaje profundo basadas en TensorFlow, destacando la efectividad del análisis estático en identificar y mitigar riesgos. Amankwah et al. (2023) [29] realiza una evaluación extensa de herramientas de análisis estático, mostrando cómo estas herramientas pueden mejorar la precisión de la detección de errores en código Java. Kaya *et al*. (2019) [30] subraya la importancia de las técnicas de análisis estático en la mejora de la precisión. Otro estudio significativo es de Pujar *et al*. (2024) [31] que analiza las vulnerabilidades en el código fuente utilizando el conjunto de datos D2A, demostrando mejoras en la precisión de la detección.

Shahoor *et al*. (2020) [32] presenta una herramienta de análisis de código estático para sistemas críticos de misión, que mejora significativamente la precisión en la identificación de errores y vulnerabilidades en código C#. Li (2020) [35] propone un mapeo de vulnerabilidades basado en OWASP-SANS, mejorando la precisión de las pruebas de seguridad de aplicaciones estáticas al abordar específicamente las vulnerabilidades comunes en aplicaciones Web y móviles. Por último, Brito *et al*. (2023) [34] estudia diversas herramientas de análisis estático para la detección de vulnerabilidades en JavaScript, mostrando mejoras sustanciales en la precisión de la identificación de riesgos de seguridad en el código. Estos estudios reafirman el papel crítico de SAST en la mejora de la precisión de la detección de vulnerabilidades, contribuyendo a un software más seguro y confiable.

Además, en términos de Reducción de falsos positivos", cinco estudios proporcionan evidencia sobre la efectividad de SAST en la disminución de falsos positivos en la detección de vulnerabilidades. Park *et al*. (2023) [42] realiza un estudio comparativo sobre la reducción de falsos positivos para errores en tiempo de ejecución en software C/C++, destacando cómo el uso de técnicas de aprendizaje automático y profundo puede mejorar significativamente la precisión de la detección. En el estudio de Nguyen *et al*. (2023) [1] utilizan el modelo BERT para reducir las alarmas falsas en el análisis estático de debilidades, mostrando resultados prometedores en la disminución de

falsos positivos. Zhang *et al*. (2020) [35] presenta un modelo automatizado de identificación de defectos a nivel de variable basado en aprendizaje automático, logrando una reducción notable en los falsos positivos en la detección de defectos.

La investigación de Scull *et al*. (2022) [43] demuestra la derivación de pruebas de seguridad estática a partir de la protección de seguridad en tiempo de ejecución también muestra mejoras significativas en la reducción de falsos positivos. Por último, Hegedus y Ferenc (2022) [44] sostienen la filtración de alarmas de análisis de código estático presenta una metodología que logra disminuir considerablemente los falsos positivos en la detección de errores de tiempo de ejecución. Estos estudios confirman que la integración de técnicas avanzadas de análisis estático y aprendizaje automático no solo mejora la precisión de la detección de vulnerabilidades, sino que también reduce de manera efectiva los falsos positivos, contribuyendo a una mayor eficiencia en la gestión de seguridad del software.

También, en términos de ”Mejora en la cobertura de detección de vulnerabilidades”, dos estudios destacan la efectividad de SAST en la ampliación de la cobertura de detección. Nunes *et al*. (2019) [36] examina la combinación de diversas herramientas de análisis estático para mejorar la detección de vulnerabilidades en aplicaciones, demostrando que la combinación de herramientas puede aumentar significativamente la cobertura de detección al aprovechar las fortalezas individuales de cada herramienta. Mateo *et al*. (2020) [5] investiga la combinación de análisis estático, dinámico e interactivo para mejorar la detección de las principales vulnerabilidades de seguridad en aplicaciones web según el OWASP Top Ten, mostrando que una combinación de estos métodos puede mejorar la cobertura de detección de manera notable. Estos estudios subrayan que una estrategia combinada de diversas técnicas de análisis puede proporcionar una cobertura más amplia y efectiva en la detección de vulnerabilidades de seguridad en aplicaciones Web.

Finalmente, la revisión sistemática también revela otros aspectos de la efectividad de los modelos de pruebas de seguridad estática (SAST) en diferentes áreas. En términos de Reducción del costo de revisión de código”, Gunawardena *et al*. (2023) [37] demuestra cómo la implementación efectiva de SAST puede reducir los costos asociados a la revisión de código al detectar automáticamente una cantidad significativa de defectos antes de que lleguen a esta etapa. Además, en relación con la ”Mayor capacidad para integrar y gestionar”, Abdel-Kader *et al*. (2020) [38] presenta un modelo automatizado que mejora la capacidad de integración y gestión de vulnerabilidades de seguridad en lenguajes de scripting, lo que facilita la identificación y mitigación de riesgos en aplicaciones Web.

En cuanto a la “Reducción de falsos negativos”, Ochodek *et al*. (2020) [39] destaca cómo el uso de aprendizaje automático en herramientas SAST puede reducir los falsos negativos al mejorar la precisión en la identificación de violaciones de las directrices de codificación específicas de la empresa. En términos de “Adopción y escalabilidad en la Industria”, Nguyen-Duc *et al*. (2021) [40] explora la adopción de análisis estático en la industria del software, subrayando cómo estas herramientas son escalables y pueden ser integradas efectivamente en diversos entornos de desarrollo. Por último, en ”Precisión en la identificación de deudas técnicas”, Rantala *et al*. (2023) [41] demuestra cómo el uso de análisis estático puede mejorar la precisión en la identificación de deudas técnicas auto admitidas, permitiendo a los desarrolladores gestionar mejor las cargas técnicas a lo largo del ciclo de vida del software.

Q2. ¿Cuáles son los factores determinantes que influyen en la reducción de la ineficiencia en la identificación de Inyecciones SQL en aplicaciones Web?,

Para responder a la pregunta Q2. se han clasificado los factores determinantes como se ve en la Figura 2.

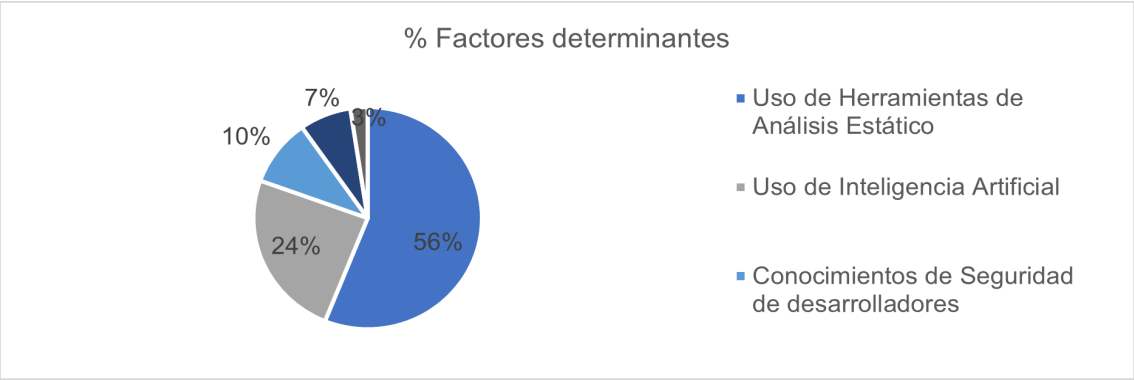

Figura 2. Porcentaje de factores determinantes que influyen en la reducción de la ineficiencia en la identificación de Inyecciones SQL en aplicaciones Web.

Uso de Herramientas de Análisis Estático

El 56 % de los artículos considerados para esta revisión sistemática, indican que un factor determinante para la reducción de la ineficiencia en la identificación de Inyecciones SQL en aplicaciones Web es el “Uso de herramientas de análisis estático”.

El uso de herramientas de análisis estático es un factor determinante en la reducción de la ineficiencia en la identificación de inyecciones SQL en aplicaciones Web. Varios estudios demuestran cómo estas herramientas mejoran la precisión y reducen los falsos positivos y negativos. En los artículos de Bakhshandeh *et al*. (2023) [6] y Sheneamer (2024) [27] destacan la efectividad de la combinación

de las herramientas de análisis estático como Semgrep, Bandit y SonarQube con técnicas de inteligencia artificial y aprendizaje profundo para identificar vulnerabilidades. Yuan *et al*. (2023) [19] propone la herramienta con extensiones de base de datos que transforman el código para mejorar la precisión en la detección de inyecciones SQL. En la evaluación de diversas herramientas de análisis estático, artículos como el de Kuszczyński y Walkowski (2023) [20] y Nunes *et al*. (2019) [36] muestran que la combinación de múltiples herramientas como Semgrep, Bandit, SonarQube y Coverity puede mitigar las limitaciones individuales, aumentando la efectividad del análisis. Subrayan Amankwah *et al*. (2023) [29] y Gunawardena *et al*. (2023) [37] la importancia de las herramientas del análisis estático en la identificación temprana de vulnerabilidades y la automatización de la revisión de código. La integración de estas herramientas en el ciclo de desarrollo es crucial para mejorar la seguridad desde las primeras etapas, como lo discuten Lombardi y Fanton (2023) [12] y Lee y Liu (2023) [13]. Artículos como el de Shahoor *et al*. (2020) [32] y Siewruk y Mazurczyk (2021) [22] resaltan la necesidad de herramientas precisas y contextuales para mejorar la seguridad del software. Bermejo *et al*. (2021) [8] y Schiewe *et al*. (2022) [3] demuestran cómo la identificación de componentes independientes del lenguaje puede mejorar la efectividad del análisis estático. Nguyen-Duc *et al*. (2021) [40] discuten los desafíos y beneficios de la adopción del análisis estático, mientras que Scull *et al*. (2022) [43] argumenta la derivación de pruebas estáticas desde protecciones en tiempo de ejecución para mejorar la precisión y efectividad. Hegedus y Ferenc (2022) [44] proponen métodos para filtrar alarmas, reduciendo falsos positivos y mejorando la eficiencia del análisis.

Además, Saurabh y Kumar (2024) [18] muestran cómo la integración de análisis estático en las primeras fases del ciclo DevOps puede reducir significativamente el tiempo de ejecución del pipeline y mejorar la seguridad del software. Destaca Alqaradaghi *et al*. (2023) [26] la importancia de herramientas precisas de análisis estático para detectar el uso inseguro de SecurityManager en Java, mejorando la detección de vulnerabilidades en código real y casos de prueba personalizados. Schubert *et al*. (2022) [15] aborda cómo el análisis de flujo de datos estático puede identificar vulnerabilidades en líneas de productos de software, mejorando la seguridad general del código. Rantala *et al*. (2023) [41] explora cómo la deuda técnica admitida por los desarrolladores puede ser detectada y gestionada mediante análisis estático, reduciendo riesgos de seguridad en el software. Piskachev *et al*. (2023) [16] estudia cómo la conFiguración de análisis estático puede optimizar la detección y resolución de vulnerabilidades de seguridad, mejorando la efectividad de estas herramientas. Brito *et al*. (2023) [34] compara diversas herramientas de análisis estático para la detección de vulnerabilidades en JavaScript, destacando sus fortalezas y debilidades. Alqaradaghi y Kozsik (2024) [17] proporcionan una evaluación exhaustiva de diversas herramientas de análisis estático, resaltando su importancia en la detección temprana de vulnerabilidades y la mejora de la seguridad del software. Estos estudios respaldan que la implementación de herramientas de análisis estático no solo mejora la calidad del software, sino que también reduce significativamente la ineficiencia en la identificación de inyecciones SQL, asegurando un desarrollo más seguro y robusto.

Uso de Inteligencia Artificial

La revisión sistemática sobre los modelos de pruebas de seguridad estática en la reducción de la ineficiencia en la identificación de inyecciones SQL en aplicaciones Web revela que el uso de la inteligencia artificial es un factor determinante clave. Los artículos de Nguyen *et al*. (2023) [1] y Park *et al*. (2023) [42] destacan la eficacia del modelo BERT en la reducción de falsos positivos y en la mejora de la precisión de la detección de vulnerabilidades. Enfatizan Kaya *et al*. (2019) [30] y Brito *et al*. (2023) [35] la importancia de la selección de características y el uso de técnicas de balanceo de datos para optimizar la predicción de vulnerabilidades. Los estudios de Filus y Domańska (2023) [28] y Pujar *et al*. (2024) [31] analizan la aplicación de técnicas de aprendizaje profundo para identificar vulnerabilidades en diferentes contextos, demostrando su efectividad y adaptabilidad. Por otro lado, Lomio *et al*. (2022) [23] y Ochodek *et al*. (2020) [39] muestran cómo las reglas específicas y las directrices de codificación pueden ser integradas en modelos de aprendizaje automático para mejorar la detección de defectos. Finalmente, Al-Johany *et al*. (2023) [24] y Szabó y Bilicki (2023) [25] abordan la reducción de errores en tiempo de ejecución mediante el uso de técnicas avanzadas de aprendizaje automático, confirmando que la inteligencia artificial no solo mejora la identificación de vulnerabilidades, sino que también minimiza los falsos positivos, aumentando la eficiencia de los análisis estáticos. Estos hallazgos subrayan que la implementación de la inteligencia artificial es esencial para mejorar la precisión y eficiencia de los modelos de pruebas de seguridad estática.

Conocimientos de Seguridad de los desarrolladores

En la revisión sistemática también se identificó que el Çonocimiento de Seguridad de los Desarrolladores.[es] un factor crucial en la reducción de la ineficiencia en la identificación de inyecciones SQL en aplicaciones web. El artículo de Wijesiriwardana *et al*. (2023) [7] destaca la importancia de visualizar y comprender las facetas de seguridad del software para mejorar las prácticas de codificación seguras entre los desarrolladores. Por otro lado, Abeyrathna *et al*. (2020) [4] y Wijesiriwardana *et al*. (2020) [14] subrayan la necesidad de un enfoque basado en el modelado del conocimiento para inferir las asociaciones entre los artefactos de diseño y el código fuente, lo cual facilita la identificación de fallos de seguridad desde las primeras etapas del desarrollo del software. Estos artículos también resaltan la eficacia del modelado de amenazas y el análisis estático del código para detectar errores de seguridad. Finalmente, Casola *et al*. (2024) [2] aborda la implementación de prácticas seguras en el ciclo de vida del desarrollo de software, destacando que la formación y el conocimiento en seguridad de los desarrolladores son esenciales para identificar y mitigar vulnerabilidades. Estos estudios en conjunto evidencian que un conocimiento sólido y específico en seguridad entre los desarrolladores es fundamental para reducir la ineficiencia en la identificación de inyecciones SQL, ya que permite una mejor integración de prácticas de seguridad en todas las fases del desarrollo del software.

Uso de enfoques combinados de Análisis estático y dinámico

La combinación de análisis estático y dinámico es un enfoque crucial para mejorar la detección de vulnerabilidades en

aplicaciones Web, y es respaldado por varios estudios. El artículo de Mateo *et al*. (2020) [3] enfatiza la efectividad de combinar diferentes herramientas de análisis para mejorar la detección de las vulnerabilidades más críticas según OWASP, resaltando cómo la integración de estos métodos puede reducir los falsos negativos y mejorar la cobertura de pruebas. Abdel-Kader *et al*. (2020) [38] presenta un modelo de análisis estático que se complementa con técnicas dinámicas para detectar vulnerabilidades en lenguajes de scripting como PHP, demostrando que la combinación de estos enfoques aumenta la precisión y robustez en la identificación de problemas de seguridad. Por último, Correa *et al*. (2021) [21] propone una metodología híbrida que integra análisis estático y dinámico, demostrando a través de estudios de caso que esta combinación mejora significativamente la detección de vulnerabilidades y la eficiencia de los procesos de seguridad. Estos estudios subrayan que el uso de enfoques combinados no solo mejora la detección y mitigación de vulnerabilidades, sino que también optimiza el tiempo y los recursos invertidos en el aseguramiento de la calidad del software.

Mapeo de Vulnerabilidades principales de OWASP Top 10 y CWE/SANS

El mapeo de las principales vulnerabilidades de OWASP Top 10 y CWE/SANS es fundamental para mejorar la seguridad en el desarrollo de aplicaciones. El artículo de Li (2020) [33] destaca la importancia de integrar los estándares de seguridad OWASP y CWE/SANS en un marco unificado para la prueba de seguridad estática de aplicaciones. Este estudio presenta un mapeo detallado de las vulnerabilidades más críticas según OWASP y CWE/SANS, proporcionando una matriz que sincroniza estas listas con las consultas de vulnerabilidad de Checkmarx. Esta integración permite a los equipos de desarrollo identificar y abordar eficientemente las vulnerabilidades de código, minimizando los falsos positivos y mejorando la precisión de las pruebas de seguridad. Un estudio de caso con una aplicación de detección de malware para Android demuestra cómo este enfoque mejora la mitigación de vulnerabilidades, reduciendo significativamente las amenazas de seguridad. Este mapeo no solo facilita la identificación de vulnerabilidades críticas como inyecciones SQL y fallos de autenticación, sino que también guía a los desarrolladores en la implementación de soluciones más seguras a lo largo del ciclo de vida del desarrollo de software.

Q3. ¿Qué modelos específicos de pruebas de seguridad estática se han desarrollado para abordar la ineficiencia en la detección de Inyecciones SQL en aplicaciones Web?,

Para responder a la pregunta Q3 se han clasificado los modelos como se ve en la Figura 3.

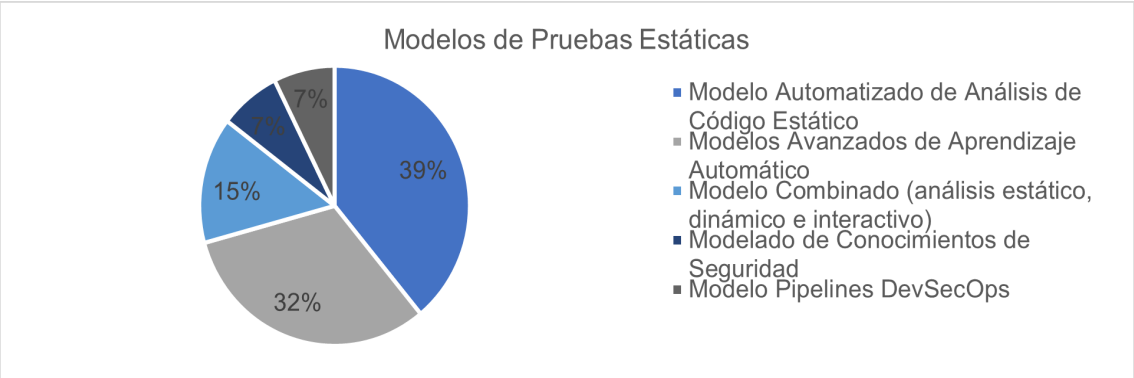

Figura 3. Modelos de Pruebas Estáticas que influyen en la reducción de la ineficiencia en la identificación de Inyecciones SQL en aplicaciones Web.

La revisión sistemática revela que el 39 % de los artículos hacen referencia sobre el ”Modelo Automatizado de Análisis de Código Estático”que demuestra ser eficaz en la reducción de la ineficiencia en la detección de inyecciones SQL en aplicaciones Web, como se evidencia en varios estudios. Los artículos de Amankwah *et al*. (2023) [29] y Shahoor *et al*. (2020) [32] destacan la precisión de estas herramientas en la detección de vulnerabilidades críticas, incluyendo inyecciones SQL, en sistemas de misión crítica y código Java. Además, Li (2020) [33] subraya la importancia de mapear las vulnerabilidades de OWASP y CWE/SANS, y comparar diversas herramientas para mejorar la precisión y reducir los falsos positivos en la detección de vulnerabilidades. En el contexto de la adopción y métodos específicos, Schubert *et al*. (2022) [15], Yuan *et al*. (2023) [19] y Nguyen-Duc *et al*. (2021) [40] discuten enfoques innovadores como la transformación de programas y el análisis de flujo de datos, los cuales son esenciales para identificar patrones de vulnerabilidad específicos de inyecciones SQL. Por otro lado, Wijesiriwardana *et al*. (2023) [7] introduce un enfoque visual tridimensional que facilita la identificación de clases y métodos vulnerables en proyectos de software, mejorando la comprensión y mitigación de riesgos. Schiewe *et al*. (2022) [3] y Abdel-Kader *et al*. (2020) [38] presentan modelos que son agnósticos al lenguaje, permitiendo la detección de vulnerabilidades en diversos lenguajes de programación, lo cual es crucial para entornos de desarrollo heterogéneos. Alqaradaghi *et al*. (2023) [26] demuestran cómo la implementación precisa de verificadores automatizados puede detectar usos inseguros del código, mejorando la robustez del análisis estático. Además, estudios de Hegedus y Ferenc (2022) [44] y Rantala *et al*. (2023) [41] se enfocan en mejorar la eficacia de los análisis estáticos mediante la filtración de falsas alarmas y la etiquetación adecuada de la deuda técnica, mientras que Piskachev *et al*. (2023) [16] resalta la importancia de la configuración óptima de las herramientas de análisis para maximizar su efectividad.

Por último, Alqaradaghi y Kozsik (2024) [17] y Brito *et al*. (2023) [34] evalúan exhaustivamente diversas herramientas de análisis estático, confirmando su eficacia en la detección de inyecciones SQL y otras vulnerabilidades en código JavaScript. Estos estudios demuestran que los modelos automatizados de análisis de código estático son esenciales para mejorar la precisión y eficiencia en la identificación y mitigación de vulnerabilidades de inyección SQL en aplicaciones Web, optimizando así la seguridad en el desarrollo de software.

El 39 % de los artículos hacen referencia sobre ”Modelos Avanzados de Aprendizaje Automático”, este tipo de modelo ha demostrado ser altamente efectivo en la detección de inyecciones SQL en aplicaciones Web, mejorando significativamente la eficiencia y precisión de las pruebas de seguridad estática. El artículo de Zhang *et al*. (2020) [35] destaca la importancia del aprendizaje automático a nivel de variable para identificar defectos de manera precisa. Bakhshandeh *et al*. (2023) [6] muestra que las herramientas basadas en inteligencia artificial, como ChatGPT, pueden reducir las tasas de falsos positivos y negativos en la detección de vulnerabilidades, comparándose favorablemente con herramientas tradicionales de pruebas de seguridad estática. En cuanto a la detección de vulnerabilidades específicas, Sheneamer (2024) [27] y Ochodek *et al*. (2020) [39] enfatizan el uso de redes neuronales convolucionales y aprendizaje automático para detectar funciones vulnerables y violaciones de directrices de codificación específicas de la empresa. Estos modelos aprenden patrones complejos de vulnerabilidades a partir de grandes conjuntos de datos, mejorando la precisión de la detección. Estudios de Al-Johany *et al*. (2023) [24] y Filus y Domańska (2023) [28] abordan la aplicación de técnicas de aprendizaje profundo y redes neuronales para identificar y corregir defectos de software en aplicaciones de aprendizaje profundo y de paso de mensajes. Estos estudios subrayan la adaptabilidad y efectividad de los modelos de aprendizaje profundo en diversos contextos de desarrollo de software. Artículos de Lomio *et al*. (2022) [23] y Gunawardena *et al*. (2023) [37] muestran que los análisis basados en aprendizaje automático y profundo pueden integrarse con reglas de SonarQube para mejorar la identificación de vulnerabilidades durante las revisiones de código. Park *et al*. (2023) [42] resalta la capacidad predictiva de los modelos de aprendizaje automático para reducir los falsos positivos, mejorando significativamente la precisión en la detección de errores en tiempo de ejecución. Por último, Nguyen *et al*. (2023) [1] y Pujar *et al*. (2024) [31] demuestran la efectividad del modelo BERT en la detección de vulnerabilidades. Estos modelos utilizan técnicas avanzadas de procesamiento de lenguaje natural para identificar patrones de vulnerabilidades con alta precisión y reducir la tasa de falsos positivos y negativos. Refuerzan estos hallazgos Szabó y Bilicki *et al*. (2023) [25] y Pujar *et al*. (2024) [31] al aplicar técnicas de aprendizaje automático para el análisis de vulnerabilidades en aplicaciones Web y conjuntos de datos específicos.

Estos estudios en conjunto demuestran que los modelos avanzados de aprendizaje automático son esenciales para mejorar la precisión y eficiencia en la identificación de inyecciones SQL y otras vulnerabilidades en aplicaciones Web. La integración de técnicas de aprendizaje automático en las herramientas de pruebas de seguridad estática permite una detección más rápida y precisa de vulnerabilidades, optimizando así la seguridad del software.

El 15 % de los artículos revisados, hacen referencia sobre ”Modelos Combinados (análisis estático, dinámico e interactivo)”que se ha desarrollado para abordar la ineficiencia en la detección de inyecciones SQL en aplicaciones Web, aprovechando las fortalezas de cada tipo de análisis para mejorar la precisión y cobertura de las pruebas de seguridad. El artículo de Mateo *et al*. (2020) [5] destaca la eficacia de combinar diferentes herramientas de análisis para mejorar la detección de las vulnerabilidades más críticas según OWASP, reduciendo tanto los falsos positivos como los negativos. En una línea similar, Correa *et al*. (2021) [21] propone una metodología híbrida que integra análisis estático y dinámico, demostrando a través de estudios de caso que esta combinación mejora significativamente la detección de vulnerabilidades y la eficiencia de los procesos de seguridad. El artículo de Bermejo *et al*. (2021) [8] presenta un método combinatorio que utiliza análisis estático para evaluar la seguridad del código fuente, complementado con técnicas dinámicas para validar los resultados, aumentando la robustez del proceso de detección. Siewruk y Mazurczyk (2021) [22] introducen un sistema de clasificación de vulnerabilidades basado en el contexto que utiliza técnicas de aprendizaje automático para automatizar la priorización y gestión de vulnerabilidades detectadas por herramientas de análisis estático y dinámico permitiendo una clasificación más precisa y relevante de las vulnerabilidades en función del contexto en el que se detectan. Nunes *et al*. (2019) [36] examina la efectividad de combinar diversas herramientas de análisis estático en diferentes escenarios de desarrollo, demostrando que la diversidad de herramientas mejora la detección de vulnerabilidades y reduce los falsos positivos.

Por último, Scull *et al*. (2022) [43] aborda la derivación de pruebas de seguridad estática a partir de la protección de seguridad en tiempo de ejecución, mostrando cómo los análisis dinámicos pueden informar y mejorar las pruebas estáticas para una mayor precisión en la detección de vulnerabilidades. Estos estudios en conjunto demuestran que los modelos combinados de análisis estático, dinámico e interactivo son esenciales para mejorar la precisión y eficiencia en la identificación de inyecciones SQL y otras vulnerabilidades en aplicaciones Web. La integración de múltiples enfoques de análisis permite una detección más completa y precisa de vulnerabilidades, optimizando así la seguridad del software.

El 15 % de los artículos hacen mención sobre “Modelado de Conocimientos de Seguridad” que es un enfoque crucial para abordar la ineficiencia en la detección de inyecciones SQL en aplicaciones Web. Este enfoque integra la seguridad en todas las fases del ciclo de vida del desarrollo de software, proporcionando una visión holística de las vulnerabilidades y sus soluciones. El artículo de Abeyrathna *et al*. (2020) [4] destaca un enfoque de modelado de conocimiento que relaciona los fallos de seguridad a nivel arquitectónico con los errores de seguridad a nivel de código. Utilizando el modelado de amenazas y el análisis estático, este enfoque permite inferir asociaciones que de otro modo serían tediosas de identificar manualmente, mejorando la identificación y mitigación de vulnerabilidades desde la fase de diseño hasta la implementación. Por otro lado, Wijesiriwardana *et al*. (2020) [14] presenta un marco conceptual y una implementación de prueba de concepto para interrelacionar fallos de seguridad en el diseño con errores en el código

fuente. Este enfoque se basa en el modelo de categorización de amenazas STRIDE y las vulnerabilidades OWASP Top 10, proporcionando un método estructurado para identificar las causas raíz de los errores de seguridad durante el desarrollo del software. Los resultados experimentales confirman que las fallas de seguridad a nivel arquitectónico tienen un impacto significativo en la aparición de errores de seguridad a nivel de código, demostrando la eficacia del modelado de conocimientos en la mejora de la seguridad del software. Por último, el artículo de Casola *et al*. (2024) [2] subraya la importancia de implementar prácticas seguras en todo el ciclo de vida del desarrollo de software. Este estudio resalta que la educación y el conocimiento en seguridad de los desarrolladores son esenciales para la identificación y mitigación de vulnerabilidades. El enfoque de modelado de conocimientos facilita la integración de técnicas de seguridad en cada fase del desarrollo, asegurando que las vulnerabilidades sean tratadas de manera efectiva desde el diseño hasta la implementación. Estos estudios demuestran que el modelado de conocimientos de seguridad es esencial para mejorar la precisión y eficiencia en la identificación y mitigación de inyecciones SQL y otras vulnerabilidades en aplicaciones Web. La integración de este enfoque permite una comprensión más profunda y completa de las vulnerabilidades, optimizando así la seguridad del software a lo largo de todo su ciclo de vida.

Los hallazgos más relevantes de este estudio se recapitulan de la siguiente manera:

1. **Efectividad de SAST en la detección de inyecciones SQL:** La revisión de 41 estudios confirma que los modelos de SAST son altamente efectivos para mejorar la detección de inyecciones SQL en aplicaciones web. Los estudios más relevantes destacan que la integración temprana de SAST en pipelines DevSecOps, optimiza la eficiencia en la identificación de vulnerabilidades y reduce significativamente los tiempos de detección [3], [4], [12], [13]. Además, la personalización de SAST según el contexto del proyecto es crucial para mejorar la calidad del código y gestionar vulnerabilidades críticas [16], [18]. Estos hallazgos subrayan que un enfoque integrado y automatizado en la seguridad del software es esencial para mitigar las amenazas y garantizar un desarrollo más seguro y eficiente.
2. **Factores determinantes en la eficiencia de SAST:** La revisión identificó varios factores clave que influyen en la reducción de la ineficiencia en la identificación de inyecciones SQL en aplicaciones Web. El uso de herramientas de análisis estático es el factor más determinante, citado por el 56 % de los estudios revisados, que demuestran su eficacia en mejorar la precisión y reducir falsos positivos [6], [27], [19]. La inteligencia artificial también es esencial, como se destaca en [1], [42], para optimizar la detección de vulnerabilidades. El conocimiento en seguridad de los desarrolladores se muestra crucial para la implementación de prácticas seguras desde las primeras etapas [7], [14], [2]. Además, la combinación de enfoques de análisis estático y dinámico mejora la detección y mitigación de vulnerabilidades, según estudios como [5], [21]. Por último, el mapeo de vulnerabilidades de OWASP y CWE/SANS es fundamental para guiar a los desarrolladores en la identificación y resolución de vulnerabilidades críticas [33].
3. **Modelos específicos de SAST para inyecciones SQL:** La revisión de estudios destaca varios modelos específicos desarrollados para abordar la ineficiencia en la detección de inyecciones SQL en aplicaciones Web. Los Modelos Automatizados de Análisis de Código Estático, mencionados en el 39 % de los estudios, son efectivos para mejorar la precisión y reducir falsos positivos en la detección de vulnerabilidades [29], [32], [33]. Asimismo, los Modelos Avanzados de Aprendizaje Automático, también referidos por el 39 % de los estudios, han demostrado aumentar la eficiencia y exactitud en la identificación de amenazas utilizando redes neuronales y técnicas de inteligencia artificial [35], [6], [27]. Por otro lado, los Modelos Combinados de Análisis Estático, Dinámico e Interactivo, citados por el 15 % de los estudios, integran múltiples enfoques para maximizar la cobertura y la precisión de las pruebas de seguridad [5], [21], [8]. Finalmente, el Modelado de Conocimientos de Seguridad, igualmente referido por el 15 % de los estudios, se enfoca en integrar la seguridad en todas las fases del desarrollo del software, ofreciendo una visión completa y estructurada de las vulnerabilidades [4], [14], [2]. Estos modelos son esenciales para mejorar la eficiencia y precisión en la detección de inyecciones SQL, contribuyendo a la optimización de la seguridad en el desarrollo de software.

4. Conclusiones

La revisión sistemática realizada sobre los modelos de pruebas de seguridad estática confirma su efectividad en la reducción de ineficiencias en la detección de inyecciones SQL en aplicaciones Web, como lo reflejan los estudios de Nguyen *et al*. (2023) [1] y Schiewe *et al*. (2022) [3]. La implementación de estas pruebas permite la identificación temprana de vulnerabilidades, lo cual es crucial para evitar ataques antes de que el software sea desplegado en un entorno de producción. Es fundamental integrar herramientas de análisis estático desde las primeras etapas del desarrollo de software, como lo destacan Nguyen *et al*. (2023) [1], Bakhshandeh *et al*. (2023) [6] y Bermejo *et al*. (2021) [8]. Estas herramientas mejoran la precisión y reducen los falsos positivos y negativos, contribuyendo a una gestión más eficiente de las vulnerabilidades. Además, el uso de técnicas de aprendizaje automático e inteligencia artificial potencia la efectividad de estas herramientas, permitiendo una detección más precisa y oportuna de fallas de seguridad

Un enfoque combinado de análisis estático y dinámico se demuestra esencial para mejorar la cobertura y precisión de las pruebas de seguridad, optimizando el tiempo y los recursos necesarios para asegurar la calidad del software, como lo indica Mateo *et al*. (2020) [5]. Asimismo, la formación continua y la sensibilización sobre las prácticas seguras de desarrollo entre los desarrolladores son vitales para minimizar las vulnerabilidades y fortalecer la seguridad del software desde sus fases iniciales [7].

Futuras investigaciones podrían centrarse en el desarrollo de metodologías más avanzadas que integren de manera efectiva estas prácticas en el ciclo de vida del desarrollo de software, así como en la evaluación de nuevas herramientas y técnicas

que puedan mejorar aún más la detección y mitigación de vulnerabilidades de inyección SQL.

Fuentes de financiamiento:

No fue necesario financiamiento.

Conflicto de intereses:

No existen conflictos de intereses

Contribución de autor/es:

La escritura, gestión, investigación y recursos tecnológicos fueron realizados totalmente por el autor.

5. Referencias

1. NGUYEN, DINH; SEO, ARIA; NNAMDI, NNUBIA; SON, YUNSIK. False Alarm Reduction Method for Weakness Static Analysis Using BERT Mode. *Applied Sciences, MDPI* [online]. 2023, vol. 13, n.º 6, págs. 1-3. ISSN 2076-3417. Disponible en: https://doi.org/10.3390/app13063502.
2. CASOLA, VALENTINA; DE BENEDICTIS, ALESSANDRA; MAZZOCCA, CARLO; ORBINATO, VITTORIO. Secure software development and testing: A model-based methodology. *Computers Security, Elsevier* [online]. 2024, vol. 137, págs. 1-16. ISSN 0167-4048. Disponible en: https://doi.org/10.1016/j.cose.2023.103639.
3. SCHIEWE, MICAH; CURTIS, JACOB; BUSHONG, VINCENT; CERNY, TOMAS. Advancing Static Code Analysis with Language-Agnostic Component Identification. *IEEE Access* [online]. 2022, vol. 10, págs. 30743-30761. ISSN 2169-3536. Disponible en: https://doi.org/10.1109/ACCESS.2022.3160485.
4. ABEYRATHNA, ASHANTHI; SAMARAGE, CHAMAL; DAHANAYAKE, BUDDIKA; WIJESIRIWARDANA, CHAMAN; WIMALARATNE, PRASAD. A security specific knowledge modelling approach for secure software engineering. *Journal of the National Science Foundation of Sri Lanka* [online]. 2020, vol. 48, n.º 1, págs. 93-98. ISSN 1391-4588. Disponible en: https://doi.org/10.4038/jnsfsr.v48i1.8950.
5. MATEO, FRANCESC; BERMEJO, JUAN; BERMEJO, JAVIER; SICILIA, JUAN; ARGYROS, MICHAEL. On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications. *Applied Sciences* [online]. 2020, vol. 10, n.º 24, págs. 1-26. ISSN 2076-3417. Disponible en: https://doi.org/10.4038/jnsfsr.v48i1.8950.
6. BAKHSHANDEH, ATIEH; KERAMATFAR, ABDALSAMAD; NOROUZI, AMIR; CHEKIDEHKHOUN, MOHAMMAD. Using ChatGPT as a Static Application Security Testing Tool. *The ISC International Journal of Information Security* [online]. 2023, vol. 15, n.º 3, págs. 1-8. Disponible en: https://doi.org/10.22042/isecure.2023.182082.
7. WIJESIRIWARDANA, C.; WIMALARATNE, P.; ABEYSINGHE, T.; SHALIKA S., AHMED, N.; MUFARRIJ, M. Software Engineering Secure CodeCity: 3-dimensional visualization of software security facets. *Journal of the National Science Foundation of Sri Lanka* [online]. 2023, vol. 51, n.º 3, págs. 423-436. ISSN 2362-0161. Disponible en: https://doi.org/10.4038/jnsfsr.v51i3.11201.
8. BERMEJO, JUAN; BERMEJO, JAVIER; SICILIA, JUAN; SUREDA, TOMÁS; ARGYROS, CHRISTOPHER; MAGREÑÁN, ALBERTO. Combinatorial Method with Static Analysis for Source Code Security in Web Applications. En: Computer Modeling in Engineering Sciences. *Tech Science Press* [online]. 2021, vol. 129, n.º 2, págs. 541-565. Disponible en: https://doi.org/10.32604/cmes.2021.017213.
9. PAGE, MATTHEW; MCKENZIE, JOANNE; BOSSUYT, PATRICK; BOUTRON, ISABELLE; HOFFMANN, TAMMY; MULROW, CYNTHIA; ET AL. The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ Journals* [online]. 2021, vol. 372, n.º 71, págs. 1-9. Disponible en: https://doi.org/10.1136/bmj.n71.
10. KITCHENHAM, BARBARA. Procedures for Performing Systematic Reviews. *Keele University Technical Report* [online]. 2004, págs. 1-28. ISSN 1353-7776. Disponible en: https://www.researchgate.net/publication/228756057.
11. HOLGUIN, FRESIA; HOLGUIN, EDYS; GARCIA, NELLY. Gamificación en la enseñanza de las matemáticas: una revisión sistemática. *Revista de Estudios Interdisciplinarios en Ciencias Sociales* [online]. 2020, vol. 22, n.º 1, págs. 62-75. ISSN 1317-0570. Disponible en: https://doi.org/10.36390/telos221.05.
12. LOMBARDI, FRANCISCO; FANTON, ALBERTO. From DevOps to DevSecOps is not enough. CyberDevOps: an extreme shifting-left architecture to bring cybersecurity within software security lifecycle pipeline. *Software Quality Journal, Springer* [online]. 2023, vol. 31, n.º 2, págs. 619-654. ISSN

1317-0570. Disponible en: https://doi.org/10.1007/s11219-023-09619-3.

13. LEE, WEN; LIU, ZHUN. Microservices-based DevSecOps Platform using Pipeline and Open Source Software. *Journal of Information Science and Engineering, Airiti Library* [online]. 2023, vol. 39, n.º 5, págs. 1117-1128. Disponible en: https://doi.org/10.6688/JISE.202309_39(5).0007.

14. WIJESIRIWARDANA, CHAMAN; ABEYRATNE, ASHANTHI; SAMARAGE, CHAMAL; DAHANAYAKE, BUDDIKA; WIMALARATNE, PRASAD. Secure Software Engineering: A Knowledge Modeling based Approach for Inferring Association between Source Code and Design Artifacts. *International Journal of Advanced Computer Science and Applications* [online]. 2020, vol. 11, n.º 12, págs. 708-716. Disponible en: https://doi.org/10.14569/IJACSA.2020.0111282.

15. SCHUBERT, PHILIPP; GAZZILLO, PAUL; PATTERSON, ZACH; BRAHA, JULIAN; SCHIEBEL, FABIAN; HERMANN, BEN; WEI, SHIYI; BODDEN, ERIC. Static data-flow analysis for software product lines in C. *Automated Software Engineering, Springer* [online]. 2022, vol. 29, n.º 35, págs. 1-37. Disponible en: https://doi.org/10.1007/s10515-022-00333-1.

16. PISKACHEV, GORAN; BECKER, MATTHIAS; BODDEN, ERICK. Can the conFiguration of static analyses make resolving security vulnerabilities more effective? - A user study. *Empirical Software Engineering, Springer* [online]. 2022, vol. 28, n.º 118, págs. 1-28. Disponible en: https://doi.org/10.1007/s10664-023-10354-3.

17. ALQARADAGHI, MIDYA; KOZSIK, TAMÁS. Comprehensive Evaluation of Static Analysis Tools for Their Performance in Finding Vulnerabilities in Java Code. *IEEE Access* [online]. 2024, vol. 12, n.º 0, págs. 55824-55842. ISSN 2169-3536. Disponible en: https://doi.org/10.1109/ACCESS.2024.3389955.

18. SAURABH, SHOBHIT; KUMAR, DEEPAK. Model to reduce DevOps Pipeline execution time using SAST. *International Journal of System Assurance Engineering and Management* [online]. 2024, vol. 15, n.º 0, págs. 1999-2009. ISSN 2693-5015. Disponible en: https://doi.org/10.1007/s13198-024-02262-6.

19. YUAN, YE; LU, YULIANG; ZHU, KAILONG; HUANG, HUI; YU, LU; ZHAO, JIAZHEN. A Static Detection Method for SQL Injection Vulnerability Based on Program Transformation. *Applied Sciences MDPI* [online]. 2023, vol. 13, n.º 21, págs. 1-18. ISSN 2076-3417. Disponible en: https://doi.org/10.3390/app132111763.

20. KUSZCZYŃSKI, KAJETAN; WALKOWSKI, MICHAL. Comparative Analysis of Open-Source Tools for Conducting Static Code Analysis. *Sensors MDPI* [online]. 2023, vol. 23, n.º 18, págs. 2-33. ISSN 2076-3417. Disponible en: https://doi.org/10.3390/s23187978.

21. CORREA, RODDY; BERMEJO, BERMEJO, JAVIER; SICILIA, JUAN; SANCHEZ, MANUEL; MAGREÑÁN, ALBERTO. Hybrid security assessment methodology for web applications. *Computer Modeling in Engineering and Sciences, Tech Science Press* [online]. 2021, vol. 126, n.º 1, págs. 89-124. ISSN 2076-3417. Disponible en: https://doi.org/10.32604/CMES.2021.010700.

22. SIEWRUK, GRZEGORZ; MAZURCZYK, WOJCIECH. Context-Aware Software Vulnerability Classification Using Machine Learning. *IEEE Access* [online]. 2021, vol. 9, n.º 0, págs. 89-124. ISSN 88852-88867. Disponible en: https://doi.org/10.1109/ACCESS.2021.3075385.

23. LOMIO, FRANCESCO; MORESCHINI, SERGIO; LENARDUZZI, VALENTINA. A machine and deep learning analysis among SonarQube rules, product, and process metrics for fault prediction. *Empirical Software Engineering, Springer* [online]. 2022, vol. 27, n.º 189, págs. 1-57. Disponible en: https://doi.org/10.1007/s10664-022-10164-z.

24. AL-JOHANY, NORAH; EASSA, FATHY; SHARAF, SANAA; NOAMAN, AMIN; AHMED, ASSAD. Prediction and Correction of Software Defects in Message-Passing Interfaces Using a Static Analysis Tool and Machine Learning. *IEEE Access* [online]. 2023, vol. 11, n.º 0, págs. 60668-60680. ISSN 2169-3536. Disponible en: https://doi.org/10.1109/ACCESS.2023.3285598.

25. SZABÓ, ZOLTÁN;BILICKI, VILMOS. A New Approach to Web Application Security: Utilizing GPT Language Models for Source Code Inspection. *Future Internet MDPI* [online]. 2023, vol. 15, n.º 326, págs. 1-27. ISSN 1999-5903. Disponible en: https://doi.org/10.3390/fi15100326.

26. ALQARADAGHI, MIDYA; NAZIR, MUHAMMAD; KOZSIK, TAMÁS. Design and Implement an Accurate Automated Static Analysis Checker to Detect Insecure Use of SecurityManager. *Computers MDPI* [online].

2023, vol. 12, n.º 247, págs. 2-12. ISSN 2073-431. Disponible en: https://doi.org/10.3390/computers12120247.

27. SHENEAMER, ABDULLAH. Vulnerable JavaScript functions detection using stacking of convolutional neural networks. *PeerJ Computer Science* [online]. 2024, vol. 10, n.º 0, págs. 2-38. Disponible en: https://doi.org/10.7717/peerj-cs.1838.

28. FILUS, KATARZYNA; DOMAŃSKA, JOANNA. Software vulnerabilities in TensorFlow-based deep learning applications. *Computers Security, Elsevier* [online]. 2023, vol. 124, n.º 0, págs. 1-13. ISSN 0167-4048. Disponible en: https://doi.org/10.1016/j.cose.2022.102948.

29. AMANKWAH, RICHARD; CHEN, JINFU; SONG, HEPING; KUDJO, PATRICK. Bug detection in Java code: An extensive evaluation of static analysis tools using Juliet Test Suites. *Journal of Software Practice and Experience* [online]. 2023, vol. 53, n.º 5, págs. 1125-1143. Disponible en: https://doi.org/10.1002/spe.3181.

30. KAYA, AYDIN; KECELI, ALI; CATAL, CAGATAY; TEKINERDOGAN, BEDIR. The impact of feature types, classifiers, and data balancing techniques on software vulnerability prediction models. *Journal of Software Evolution and Process* [online]. 2019, vol. 31, n.º 9, págs. 1-25. Disponible en: https://doi.org/10.1002/smr.2164.

31. PUJAR, SAURAB; ZHENG, YUNHUI; BURATTI, LUCA; LEWIS, BURN; CHEN, YUNCHUNG; LAREDO, JIM; MORARI, ALESSANDRO; EPSTEIN, EDWARD; LIN, TSUNGNAN; YANG, BO; SU, ZHONG. Analyzing source code vulnerabilities in the D2A dataset with ML ensembles and C-BERT. *Empirical Software Engineering* [online]. 2024, vol. 29, n.º 48. Disponible en: https://doi.org/10.1007/s10664-023-10405-9.

32. SHAHOOR, AROOBA; SHAUKAT, RIDA; MINHAS, SUMAIRA; AWAN, HINA; SAGHAR, KASHIF. A C static code analysis tool for mission critical systems. *Advances in Science Technology and Engineering Systems* [online]. 2020, vol. 5, n.º 6, págs. 561-570. Disponible en: https://doi.org/10.25046/aj050668.

33. LI, JINFENG. Vulnerabilities mapping based on OWASP-SANS: A survey for static application security testing (SAST). *Annals of Emerging Technologies in Computing* [online]. 2020, vol. 4, n.º 3, págs. 1-8. Disponible en: https://doi.org/10.33166/AETiC.2020.03.001.

34. BRITO, TIAGO; FERREIRA, MAFALDA; MONTEIRO, MIGUEL; LOPES, PEDRO; BARROS, MIGUEL; FRAGOSO, JOSE; SANTOS, NUNO. Study of JavaScript Static Analysis Tools for Vulnerability Detection in Node.js Packages. *IEEE Transactions on Reliability* [online]. 2023, vol. 72, n.º 4, págs. 1324-1339. ISSN 1558-1721. Disponible en: https://doi.org/10.1109/TR.2023.3286301.

35. ZHANG, YUWEI; XING, YING; GONG, YUNZHAN; JIN, DAHAI; LI, HONGHUI; LIU, FENG. A variable-level automated defect identification model based on machine learning. *Soft Computing* [online]. 2020, vol. 24, n.º 2, págs. 1045-1061. Disponible en: https://doi.org/doi:%2010.1007/s00500-019-03942-3.

36. NUNES, PAULO; MEDEIROS, IBÉRIA; FONSECA, JOSÉ; NEVES, NUNO; CORREIA, MIGUEL; VIEIRA, MARCO. An empirical study on combining diverse static analysis tools for web security vulnerabilities based on development scenarios. *Computing* [online]. 2019, vol. 101, n.º 2, págs. 161-185. Disponible en: https://doi.org/10.1007/s00607-018-0664-z.

37. GUNAWARDENA, SANURI; TEMPERO, EWAN; BLINCOE, KELLY. Concerns identified in code review: A fine-grained, faceted classification. *Information and Software Technology* [online]. 2023, vol. 153, n.º 0, págs. 1-14. ISSN 0950-5849. Disponible en: https://doi.org/10.1016/j.infsof.2022.107054.

38. ABDEL-KADER, RABAB; NASHAAT, MONA; HABIB, MOHAMED; MAHDI, HANI. Automated server-side model for recognition of security vulnerabilities in scripting languages. *International Journal of Electrical and Computer Engineering* [online]. 2020, vol. 10, n.º 6, págs. 6061-6070. ISSN 2088-8708. Disponible en: https://doi.org/10.11591/ijece.v10i6.pp6061-6070.

39. OCHODEK, MIROSLAW; HEBIG, REGINA; MEDING, WILHELM; FROST, GERT; STARON, MIROSLAW. Recognizing lines of code violating company-specific coding guidelines using machine learning: A Method and Its Evaluation. *Empirical Software Engineering* [online]. 2020, vol. 25, n.º 1, págs. 220-265. Disponible en: https://doi.org/10.1007/s10664-019-09769-8.

40. NGUYEN-DUC, ANH; VIET, MANH; LUONG, QUAN; NGUYEN, KIEM; NGUYEN, ANH. On the adoption of static analysis for software security assessment–A case study of an open-source e-government project. *Computers Security* [online]. 2021, vol. 111, n.º 0, págs. 1-14. Disponible en: https://doi.org/10.1016/j.cose.2021.102470.

41. RANTALA, LEEVI; MÄNTYLÄ, MIKA; LENARDUZZI, VALENTINA. Keyword-labeled self-admitted technical debt and static code analysis have significant relationship but limited overlap. *Software Quality Journal* [online]. 2023, vol. 32, n.º 0, págs. 91-429. Disponible en: https://doi.org/10.1007/s11219-023-09655-z.

42. PARK, JIHYUN; SHIN, JAEYOUNG; CHOI, BYOUNGJU. Reduction of False Positives for Runtime Errors in C/C++ Software: A Comparative Study. *Electronics, MDPI* [online]. 2023, vol. 12, n.º 3518, págs. 1-12. ISSN 2079-9292. Disponible en: https://doi.org/10.3390/electronics12163518.

43. SCULL, ANGEL; NICOLAY, JENS; GONZALEZ, ELISA. Deriving Static Security Testing from Runtime Security Protection for Web Applications. *Art, Science, and Engineering of Programming* [online]. 2022, vol. 6, n.º 1, págs. 1-12. ISSN 2473-7321. Disponible en: https://doi.org/10.22152/programming-journal.org/2022/6/1.

44. HEGEDUS, PÉTER; FERENC, RUDOLF. Static Code Analysis Alarms Filtering Reloaded: A New Real-World Dataset and its ML-Based Utilization. *IEEE,* [online]. 2022, vol. 10, págs. 1-12. ISSN 55090–55101. Disponible en: https://doi.org/10.1109/ACCESS.2022.3176865.