RAG-based architecture for cybersecurity vulnerabilities remediation in a specific technology stack

Victoria Elizabeth Meichtry Regner[1], Diego Ignacio Maldonado[1], Karim Jair Servin[1]

[1] Facultad Regional Santa Fe, UTN, Lavaisse 610, Santa Fe, Argentina
{meichtryv4@gmail.com, imaldonado@frsf.utn.edu.ar, kservin@frsf.utn.edu.ar}

Abstract. The surge in cyber threats and the increasing complexity of software architectures have made information security a priority. While Static Analysis (SAST) and Dynamic Analysis (DAST) tools are effective at detecting flaws, they often lack the necessary context to propose direct solutions within the source code. This limitation forces developers to spend significant time adapting generic fixes to their specific technology stack. To address this inefficiency, this paper proposes the development of an intelligent assistant based on a Retrieval-Augmented Generation (RAG) architecture. This system leverages locally executed Large Language Models (LLMs) combined with a vector database populated with valeted security standards to generate precise, contextualized code mitigations. The proposed architecture utilizes a Spring Boot orchestrator, MongoDB for vector search and Ollama for local inference. By decoupling knowledge retrieval from language generation, the RAG approach significantly mitigates the risk of "hallucinations" inherent in generic LLMs. Furthermore, local processing ensures the privacy of proprietary source code. Future work will focus on data sanitization, LLM optimization and performance metric validation.

Keywords: cybersecurity, RAG, vulnerability remediation.

Arquitectura basada en RAG para remediación de vulnerabilidades de ciberseguridad en un stack tecnológico determinado

Resumen. El incremento de las amenazas cibernéticas y la complejidad de las arquitecturas de software hacen de la seguridad informática una prioridad. Si bien herramientas de análisis estático (SAST) y dinámico (DAST) son eficientes para detectar fallos, carecen de contexto necesario para proponer soluciones directas en el código fuente. Esta limitación obliga a los desarrolladores a invertir tiempo adaptando soluciones genéricas a su tecnología particular. Para resolver esta ineficiencia, el presente trabajo propone el desarrollo de un asistente inteligente basado en una arquitectura Retrieval-Augmeted Generation (RAG). Este sistema utiliza modelos de lenguaje grande (LLMs) de ejecución local combinados con una base de datos vectorial compuesta por estándares de seguridad validados para generar mitigaciones de código precisas y contextualizadas. La arquitectura propuesta emplea un orquestador en Spring Boot, MongoDB para la búsqueda

vectorial y Ollama para la inferencia local. Al separar la recuperación de conocimiento de la generación de lenguaje, el enfoque RAG mitiga significativamente el riesgo de "alucinación" inherente a los LLMs genéricos. Además, el procesamiento local garantiza la privacidad del código propietario. El trabajo futuro se centra en la sanitización de los datos, optimización del LLM y validación de métricas de rendimiento.

Palabras clave: ciberseguridad, RAG, remediación de vulnerabilidades.

1 Introducción

En la actualidad, el incremento sostenido de las amenazas cibernéticas y la creciente complejidad de las arquitecturas de software han convertido a la seguridad informática en una prioridad para las organizaciones. A pesar de que las herramientas de análisis estático de seguridad (SAST) y análisis dinámico de seguridad (DAST) son importantes para la detección temprana de vulnerabilidades, éstas presentan una brecha operativa en la industria: son altamente eficaces para identificar fallos, pero carecen del contexto necesario para proponer una solución especifica y directa en el código fuente. Esta limitación obliga a los equipos de desarrollo a invertir tiempo y recursos en la investigación y adaptación de soluciones genéricas a la tecnología particular que están utilizando para desarrollar en ese momento, lo cual aumenta la ventana de exposición al riesgo.

Ante este escenario, los modelos de lenguaje grande (LLMs) han surgido como asistentes para automatizar tareas de desarrollo. Sin embargo, la utilización de modelos genéricos en el ámbito de la ciberseguridad presenta riesgos que limitan su adopción profesional. Por un lado, estos modelos son propensos a la "alucinación", generando sugerencias que pueden incluir librerías obsoletas o, en el peor de los casos, introducir nuevas vulnerabilidades. Por otro lado, el envío de fragmentos de código propietario hacia interfaces de programación de aplicaciones (APIs) de terceros representa una potencial fuga de datos sensibles incumpliendo las políticas de confidencialidad corporativas.

Para resolver estas deficiencias, el presente trabajo propone el desarrollo de un asistente inteligente basado en una arquitectura Retrieval-Augmented Generation (RAG). Este enfoque permite que el sistema genere soluciones para mitigar o prevenir problemas de ciberseguridad precisas y contextualizadas al stack tecnológico del usuario mediante la consulta en tiempo real a bases de conocimiento curadas. Al procesar la información de manera local o controlada, la arquitectura garantiza tanto la fiabilidad técnica de las soluciones como la privacidad del código fuente, reduciendo el esfuerzo requerido para solucionar fallas de seguridad.

2 Marco conceptual y trabajos relacionados

2.1 Estándares de seguridad: OWASP y CWE.

La estandarización en ciberseguridad es fundamental para establecer un lenguaje común entre desarrolladores y expertos. El Open Worldwide Application Security Project (OWASP) es una fundación sin fines de lucro, se utiliza su proyecto más

destacado, el OWASP Top 10, el cual identifica los 10 riesgos de seguridad más críticos y pautas de mitigación técnica (OWASP Foundation, 2021), asegurando que la base de conocimiento del sistema posea respaldo experto.

Complementariamente, la base de datos Common Weakness Enumeration (CWE) actúa como un diccionario formal de debilidades de software y hardware. Esto facilita un marco de referencia preciso que permite entender las fallas desde sus conceptos más abstractos hasta sus defectos de implementación específicos.

2.2 Justificación técnica: RAG frente a Fine-tuning.

La especialización de un modelo de lenguaje grande (LLM) para la remediación especifica de código puede abordarse mediante dos estrategias principales: el ajuste fino (Fine-tuning) o la generación aumentada por recuperación (RAG). Se descartó el Fine-tuning debido a su alto costo computacional, rápida obsolescencia ante nuevas vulnerabilidades y nula trazabilidad de las fuentes.

Por el contrario, se optó por la arquitectura RAG (Lewis et al., 2020) que separa la capacidad de procesamiento del lenguaje del almacenamiento de conocimiento. Este esquema consulta una base de datos externa en tiempo real, lo que ofrece tres ventajas determinantes:

- Elimina la necesidad de ciclos de reentrenamiento costosos, permitiendo que el sistema funcione en hardware más modesto.
- Se mitigan significativamente las "alucinaciones" al obligar al modelo a basarse únicamente en documentación valida.
- Permite actualizaciones instantáneas del conocimiento y facilita el despliegue local, garantizando la privacidad del código fuente del usuario.

2.3 Trabajos relacionados.

En los últimos años, la aplicación de la inteligencia artificial para la gestión y mitigación de vulnerabilidades ha sido objeto de intensa investigación. Un estudio reciente publicado en la revista Tono (Baloira Reyes et al., 2025) propone la integración de modelos de lenguaje de gran tamaño (LLMs) y aprendizaje por refuerzo profundo para la ciberseguridad. Sin embargo, dicha investigación concluye con una advertencia crítica: permitir que estos modelos operen de forma autónoma o generen respuestas sin un marco de supervisión estricto representa un riesgo inaceptable para las organizaciones. Esto fundamenta la necesidad de acortar la libertad de los LLMs genéricos mediante mecanismos de control de contexto, como el que provee la arquitectura RAG.

Para adaptar, estos modelos al dominio de la seguridad se han explorado diversas estrategias. Un proyecto reciente abordó la creación de un asistente de IA especializado en la resolución de entornos vulnerables (maquinas VulnHub) mediante el ajuste fino (Fine-tuning) de un modelo fundacional. Si bien el trabajo demostró la viabilidad de entrenar modelos para asistir en ciberseguridad, también evidenció las barreras logísticas de este enfoque: la exigencia de crear y curar datasets formativos estáticos, la dificultad técnica de ejecutar modelos pesados en entornos locales comunes y la falta de actualización dinámica. Estas limitaciones de la industria respaldan directamente la elección del diseño propuesto en el presente artículo, donde la recuperación semántica (RAG) reemplaza al costoso reentrenamiento.

Complementariamente, investigaciones coinciden en la pertinencia de evaluar críticamente el impacto y la integración de la IA generativa en entornos de desarrollo software seguro. La convergencia de estos trabajos académicos refuerza el modelo arquitectónico aquí propuesto: para que un LLM sea una herramienta de remediación verdaderamente confiable para un desarrollador, no basta con su capacidad generativa pura. El modelo debe actuar como una interfaz de razonamiento estructurado sobre una base de datos externa validada (como los lineamientos de OWASP), garantizando así la privacidad de ejecución local y mitigando los riesgos.

3 Arquitectura propuesta

La arquitectura modular propuesta desacopla la lógica de orquestación de la ejecución del modelo, integrando el procesamiento de lenguaje natural con persistencia de datos. Este enfoque facilita la escalabilidad eficiente y garantiza la soberanía de los datos al operar en entornos controlados, cuyos componentes y flujo se detallan a continuación.

3.1 Esquema del sistema.

El núcleo del sistema es un orquestador desarrollado en Spring Boot, el cual gestiona la comunicación entre la interfaz de usuario, el motor de búsqueda semántica y el modelo de lenguaje. Para facilitar el entendimiento de la Fig.1 se explica cada parte del asistente a desarrollar:

- Capa de aplicación (Frontend/API). El punto de entrada donde el usuario describe la tecnología y la vulnerabilidad detectada.
- Orquestador (Backend – Spring Boot). Actúa como el centro lógico que transforma la consulta en un vector numérico y coordina las llamadas a la base datos y al modelo.
- Base de datos vectorial (MongoDB). Almacena los fragmentos de conocimiento.
- Motor de inferencia (LLM Local – Ollama). El modelo que recibe el contexto recuperado y genera la solución técnica.

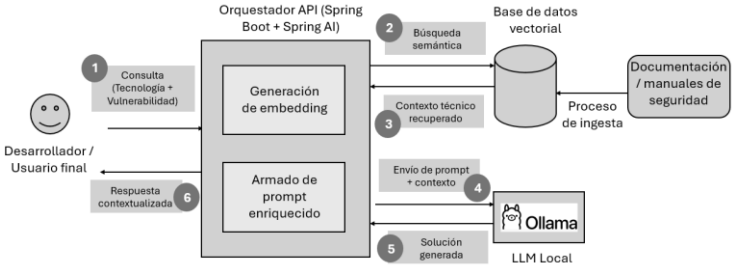

Fig. 1. Arquitectura del sistema propuesto basado en RAG para brindar soluciones de código específico a vulnerabilidades detectadas en un stack tecnológico específico.

3.2 Funcionamiento arquitectónico y flujo de datos.

El funcionamiento de la arquitectura se divide en dos procesos: la preparación del conocimiento y la generación de la respuesta. El primer proceso, denominado vectorización, consiste en trasformar la documentación técnica de seguridad informática en una representación matemática. Este proceso implica fragmentar los documentos en unidades de información manejables (chunking), convirtiéndolos en vectores mediante un modelo de embeddings y almacenarlos en MongoDB utilizando sus capacidades de búsqueda vectorial.

El segundo proceso ocurre durante la inferencia del usuario. Cuando se recibe una consulta, el orquestador genera un vector para dicha entrada y realiza una búsqueda de similitud de coseno (para determinar la similitud semántica) en la base de datos para recuperar los fragmentos de mitigación más relevantes. Posteriormente, se construye un "prompt enriquecido" que incluye tanto la consulta original como el contexto técnico recuperado. Este paquete de información es enviado al LLM local, el cual, bajo instrucciones, sintetiza una respuesta que incluye la explicación de la vulnerabilidad y un ejemplo de código corregido adaptado al stack tecnológico especificado por el desarrollador.

4 Prueba de concepto teórica y resultados esperados

Al encontrarse el proyecto en una fase de diseño arquitectónico y formulación, los resultados preliminares se centrarán en la validación teórica del flujo de datos. Para ilustrar el comportamiento esperado del asistente, se propone un hilo de ejecución típico basado en un escenario de uso real. Consideramos el caso en el que un desarrollador ingresa la siguiente petición al sistema: "Identifique una vulnerabilidad de Inyección SQL en mi backend desarrollado con Node.js". El flujo de procesamiento propuesto seguiría una secuencia lógica de cinco pasos:

1. El controlador REST recibe la consulta en texto plano y solicita al servicio de embeddings su conversión a un formato vectorial.
2. El vector generado se utiliza para realizar una búsqueda de los k documentos de seguridad más cercano a MongoDB.
3. El sistema extrae los lineamientos importantes de los documentos recuperados.
4. El orquestador ensambla un mensaje que concatena la consulta original del usuario con el contexto técnico curado y recuperados en el paso anterior.
5. El modelo de lenguaje procesa el contexto y formula la solución final.
6. El sistema entrega una explicación estructurada junto con el código corregido listo para usar en el stack especificado.

Como resultado de este flujo, el sistema emite una respuesta estructurada que no se limita a la teoría, sino que entrega el código listo para usar.

5 Conclusiones y trabajos futuros

La implementación de una arquitectura RAG reduce la fricción entre la detección de vulnerabilidades y el desarrollo de su solución. Al combinar bases de conocimiento

normativas con LLMs locales, el sistema transforma reportes de seguridad estáticos en soluciones de código adecuadas y aplicables, garantizando a su vez confidencialidad del software propietario de la organización.

A nivel técnico, este enfoque mitiga el riesgo de "alucinación" propio de los LLMs genéricos, ya que ancla cada sugerencia a una base documental validada. Además, la elección de un stack estándar para el desarrollo (Spring Boot, MongoDB) asegura que el sistema sea escalable y facilite su despliegue en entornos corporativos.

Como trabajo futuro, la investigación se centrará en tres ejes. Primero, se profundizará en la búsqueda y sanitización de nuevas fuentes de datos y guías de seguridad para ampliar el soporte tecnológico del asistente. Segundo, se profundizará en los aspectos técnicos del procesamiento de datos, evaluando distintas estrategias de chunking semántico y comparando el rendimiento de diversos modelos de embeddings para optimizar la recuperación del contexto. En segundo lugar, se abordará el diseño de las interfaces de usuario (UI) y la implementación técnica de la memoria conversacional para asegurar la usabilidad de la herramienta. Finalmente, se ejecutará una validación incorporando métricas concretas para evaluar la arquitectura; esto incluirá métricas de recuperación de información para el motor vectorial (tales como Precision, Recall, F1-Score).

References

Baloira Reyes, A., Baluja García, W., & Anías Calderón, C. E. (2025). Solución autonómica para la gestión integral de la ciberseguridad basado en aprendizaje por refuerzo. Tono, Revista Técnica de la Empresa de Telecomunicaciones de Cuba S.A., 22(2), 7-20. Recuperado de https://www.revistatono.etecsa.cu/tono/article/view/445

Cordón Muñoz, J. A. (2024). TransformerGuard: Desarrollo de un asistente de ciberseguridad proactivo mediante IA generativa. Repositorio Documental Gredos, Universidad de Salamanca (USAL). Recuperado de https://gredos.usal.es/handle/10366/164899

Del Campo Gragera, A., Gálvez Gómez, J. M., Redondo Navarro, I., Montero Gómez, J., Trenado Martin, A. (2025). Ciberseguridad en la era de la IA generativa y la IA explicable. Repositorio Institucional Docta Complutense, Universidad Complutense de Madrid (UCM). Recuperado de https://docta.ucm.es/entities/publication/d432b964-db2f-49af-ae8a-61f135848161

Lewis, P., et. Al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems.

MITRE (2023). Common Weakness Enumeration (CWE). https://cwe.mitre.org/

MongoDB, Inc. (2024). MongoDB Atlas Vector Search Documentation. https://www.mongodb.com/docs/atlas/atlas-vector-search/

Ollama. (2024). Ollama: Get up and running with large language models locally. https://ollama.com/

OWASP Foundation (2021). OWASP Top 10:2021 – The ten most critical web application security risks. https://owasp.org/Top10/