ASCE MAGAZINE ISSN: 3073–1178

Revista ASCE Magjjazine, Periodicidad: Trimestral, Volumen: 5, Número: 3, Año: 2026 páginas 4065-4086

Doi: https://doi.org/10.70577/asce.v5i3.805

Recibido: 2026-04-20

Aceptado: 2026-05-04

Publicado: 2026-08-11

Análisis de Vulnerabilidades en Vibe Coding para Aplicaciones Web: Un Estudio de Caso

Vulnerability Analysis in Vibe Coding for Web Applications: A Case Study

Autor(s)

Danilo Marcelo Gallo Colcha[1]
Facultad de Ingenierías
dgallo5@indoamerica.edu.ec
https://orcid.org/0009-0005-9936-5636
Universidad Tecnológica Indoamérica
Ambato – Ecuador

Fátima Avilés Castillo[2]
Facultad de Ingenierías
fatimaaviles@uti.edu.ec
https://orcid.org/0000-0002-0184-8893
Universidad Tecnológica Indoamérica
Ambato - Ecuador

Cómo citar

Gallo Colcha, D. M., & Avilés Castillo, F. (2026). Análisis de Vulnerabilidades en Vibe Coding para Aplicaciones Web: Un Estudio de Caso. *ASCE MAGAZINE*, *5*(3), 4065–4086. https://doi.org/10.70577/asce.v5i3.805

Resumen

El vibe coding, un término emergente que se originó en 2025, describe una forma de programación en la que el desarrollador delega una parte significativa de la producción de código al modelo de lenguaje a través de instrucciones en lenguaje natural. Su impacto en la productividad y accesibilidad es en gran medida el tema de investigaciones recientes, sin embargo, la robustez de estas aplicaciones cuando ya están funcionando en el mundo real sigue siendo un terreno poco explorado. Se describe aquí un estudio de caso único, nuestra auditoría de caja blanca para un SaaS multi-tenant construido completamente con vibe coding y el modelo Claude Sonnet 4.6 basado en un prompt maestro que dejó las decisiones de seguridad al modelo dentro del marco funcional ya definido por el prompt maestro. El procedimiento fue guiado con reconocimiento, análisis de código estático, sondeo en vivo del artefacto y verificación manual de falsos positivos basado en la Guía de Pruebas de Seguridad Web de OWASP. El segundo modelo operó como copiloto de auditoría. Los 41 hallazgos identificados se contrastaron con el estándar OWASP Top 10:2025, y el resultado fue notable: el 39,0 % quedaba concentrado en solo dos categorías, Configuración Incorrecta de Seguridad (A02) y Fallos de Autenticación (A07). Esta distribución refleja una particularidad del caso estudiado y no pretende extrapolarse como un patrón generalizable. La revisión asistida por el modelo arrojó, además, un 10,9 % de falsos positivos, lo que pone de manifiesto la importancia de incluir una fase de verificación humana sistemática en auditorías de este tipo. Existe también un sesgo potencial en la evaluación que conviene reconocer: el modelo auditor utilizado (Claude Opus 4.7) pertenece a la misma familia que el modelo generador (Claude Sonnet 4.6). En cuanto al comportamiento del generador, las instrucciones genéricas de seguridad no fueron suficientes para evitar los hallazgos reportados, incluso conociendo de antemano la existencia de una auditoría posterior. Los resultados quedan circunscritos, en todos los casos, al escenario analizado.

Palabras clave: vibe coding; seguridad de aplicaciones web; auditoría de caja blanca; OWASP Top 10:2025; código generado por IA; estudio de caso.

Abstract

Vibe coding, a term coined in 2025, describes a way of programming in which the developer hands over much of the code production to a language model through natural-language instructions. Recent scholarship has mostly looked at its effects on productivity and accessibility; the security of artifacts actually deployed in production has received far less attention. This article reports on a single case study: a white-box security audit of a multi-tenant SaaS application built entirely through vibe coding with Claude Sonnet 4.6. The master prompt delegated security decisions to the model, within the functional constraints the prompt itself established. The audit combined reconnaissance, static source-code analysis, live probing of the deployed application, and manual false-positive verification, following the OWASP Web Security Testing Guide. A second language model served as an audit copilot throughout. All 41 confirmed findings were mapped to the OWASP Top 10:2025. In this particular case, 39.0% clustered in just two categories Security Misconfiguration (A02) and Authentication Failures (A07) though these results are specific to the case examined and are not intended to generalize beyond it. The AI-assisted inspection also produced a 10.9% false-positive rate, which points to the practical value of including a systematic human-verification step in audits of this kind. A possible bias is acknowledged: the auditor model (Claude Opus 4.7) belongs to the same family as the generator (Claude Sonnet 4.6). In the case analyzed, generic security instructions did not prevent the documented findings even when the model was warned about a subsequent audit. Directions for future research are also proposed.

Keywords: vibe coding; web application security; white-box audit; OWASP Top 10:2025; AI-generated code; case study.

Introducción

El desarrollo de software vive una transformación rápida de la mano del *vibe coding*, una forma de trabajo en la que el programador interactúa con modelos de lenguaje de gran tamaño mediante instrucciones en lenguaje natural y les cede parte de la producción, revisión y ajuste del código (Karpathy, 2025; Sarkar & Drosos, 2025). El término fue acuñado por Andrej Karpathy en febrero de 2025 y, en poco más de un año, ha sido recogido por una decena de publicaciones académicas. Sarkar y Drosos (2025) introducen el concepto de material disengagement, mientras que Ray (2025) propone una taxonomía de cinco modos de interacción que abarca desde la colaboración experta hasta la delegación plena, en la que el modelo opera con autonomía casi total. La adopción industrial avanza a un ritmo acelerado: Meske et al. (2025) reportan, con base en datos de Y Combinator, que el 25 % de las *startups* admitidas en la cohorte de invierno de 2025 declararon bases de código con hasta un 95 % de contenido generado por IA.

La mayor parte de la investigación académica se ha ocupado de los efectos del paradigma sobre la productividad, la accesibilidad y la forma en que se reconfigura el trabajo intelectual (Gadde, 2025; Mahiques Fenollar, 2025; Ray, 2025). Los riesgos que se señalan (black box codebases, responsibility gaps, atrofia de competencias) aparecen, en general, enunciados de forma amplia y no se desagregan en perfiles concretos de vulnerabilidad sobre artefactos desplegados (Meske et al., 2025). La evidencia empírica disponible en torno a la seguridad del código generado por modelos de lenguaje de gran tamaño (LLM, del inglés Large Language Model), que documenta tasas de código inseguro cercanas al 40 % en distintos modelos comerciales (Mohsin et al., 2024; Wang et al., 2024), se ha construido sobre muestras sintéticas: fragmentos aislados para tareas específicas, no sistemas funcionales en producción que integran código de aplicación, configuración de infraestructura, despliegue y operación.

Para asegurar la trazabilidad y la comparabilidad del análisis con la práctica profesional de auditoría de aplicaciones web, el presente estudio se ancla en estándares formales de ciberseguridad: la guía de pruebas de seguridad web de la Open Web Application Security Project (OWASP) Web Security Testing Guide v4.2 (OWASP Foundation, 2020) opera como marco metodológico de las pruebas, y los hallazgos confirmados se clasifican conforme al estándar OWASP Top 10:2025 (OWASP Foundation, 2025), que recoge las diez clases de riesgo más

críticas en aplicaciones web. Esta doble adopción permite que los resultados sean interpretables más allá del caso concreto, aunque los hallazgos no se extiendan al paradigma en general.

El caso adquiere particular relevancia cuando el desarrollador solicita al modelo "el estándar de seguridad más alto" sin traducir esa instrucción en requisitos específicos por vector de ataque: lo que Ray (2025) clasifica como *full delegation*. El modelo arbitra entonces, a la vez, decisiones con respuesta canónica bien documentada (como el *hashing* de contraseñas con Argon2id, recomendación del *Password Hashing Competition* (Biryukov et al., 2016)) y decisiones que solo tienen sentido dentro de un contexto operacional específico (como dónde almacenar un *token* de sesión en el cliente). Conviene matizar que esta delegación nunca es absoluta: el modelo decide dentro del marco que el propio prompt fija (el *stack*, el dominio, las restricciones de arquitectura), de modo que su libertad está acotada aunque el desarrollador no prescriba los mecanismos de seguridad. La pregunta que orienta este estudio es, en ese marco, acotada:

¿Qué tipos de vulnerabilidades de seguridad emergen en un SaaS multi-tenant construido íntegramente mediante vibe coding, cuando las decisiones de seguridad se delegan a un modelo de lenguaje, en un estudio de caso único?

El estudio se articula en torno a un caso concreto: un SaaS de gestión de flujo de caja desarrollado íntegramente mediante vibe coding con Claude Sonnet 4.6 y auditado posteriormente en modalidad de caja blanca con Claude Opus 4.7. La contribución es a la vez metodológica y empírica, y ofrece una tríada reproducible compuesta por (a) el prompt maestro completo que dio origen al sistema, (b) el artefacto desplegado en producción y (c) el informe con los 41 hallazgos confirmados, mapeados al OWASP Top 10:2025 (OWASP Foundation, 2025). Los datos suplementarios el prompt, la tabla de hallazgos y la comparación entre la pila solicitada y la obtenida se publican en un repositorio público cuyo enlace figura al final del artículo, con el propósito de facilitar la replicación con otros modelos o con variaciones del prompt. El artículo se organiza como sigue: Material y métodos describe el diseño de la aplicación y el procedimiento de auditoría; Resultados presenta los 41 hallazgos mapeados al OWASP Top 10:2025 y detalla tres casos críticos; Discusión analiza los patrones observados y las limitaciones del estudio; y Conclusiones sintetiza los hallazgos principales y esboza líneas de investigación futura.

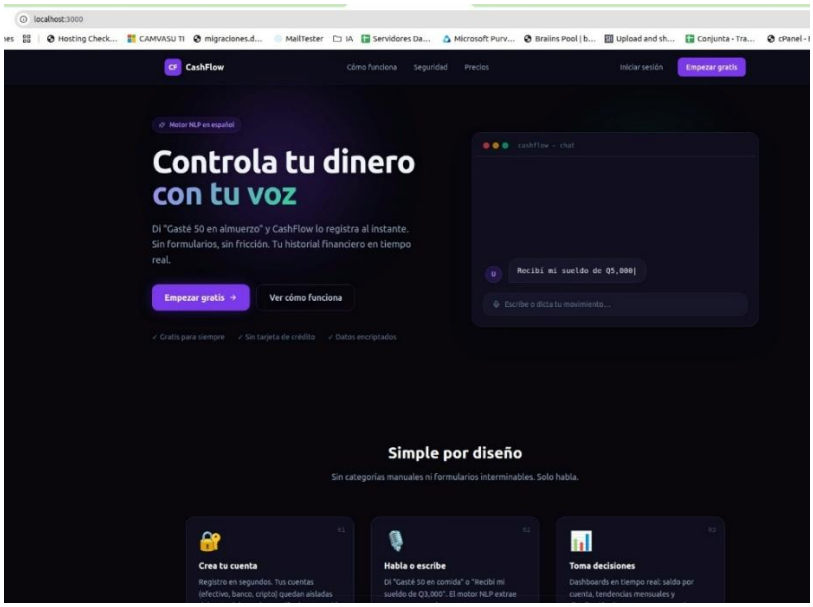
Material y métodos

Diseño de la aplicación

La aplicación auditada es un Software as a Service (SaaS) multi-tenant, pensado para que cada usuario registre ingresos y egresos desde una interfaz de lenguaje natural. El sistema se desplegó en Hetzner Cloud (Ubuntu 22.04) con Cloudflare como red de distribución y cortafuegos perimetral. El desarrollo transcurrió entre marzo y abril de 2026, con Claude Sonnet 4.6 como modelo asistente y un único prompt maestro como punto de partida (disponible en el repositorio de la investigación.). El prompt fijaba requisitos funcionales (gestión de identidad, aislamiento entre usuarios, panel privado, landing pública, entrada en lenguaje natural) y un stack de referencia (FastAPI, SQLAlchemy y PostgreSQL en el backend; Next.js 14 y Tailwind CSS en el frontend; JSON Web Token (JWT) para autorización). Las decisiones sobre los mecanismos de seguridad quedaron en manos del modelo, dentro del marco que el prompt ya establecía, a través de la instrucción: “Implementa lo que tú consideres el estándar de seguridad más alto para producción”. Esa forma de delegar encaja con lo que Sarkar & Drosos (2025) y Meske et al. (2025) describen como material disengagement.

Figura 1

Interfaz de usuario y módulo de entrada en lenguaje natural



Fuente: Elaboración propia.

El artefacto resultante se desvió del stack solicitado en varios puntos, documentados en el repositorio de la investigación: el frontend correspondió a una versión más reciente de la prevista (Next.js 16.2.4 en lugar de 14), y el modelo incorporó por iniciativa propia Redis, Alembic, Argon2id, slowapi y Docker Compose. Al momento de la auditoría, el código sumaba unas 11.200 líneas (4.600 en el backend Python, 6.100 en el frontend TypeScript, 500 en configuración de despliegue), con diez tablas PostgreSQL.

Herramientas empleadas

Para maximizar la replicabilidad se emplearon herramientas de código abierto o de uso libre. La Tabla 1 las agrupa por fase del procedimiento.

Tabla 1

Fase	Herramienta	Función
1	ls, find, tree	Enumeración de archivos
	curl	Sondeo HTTP de cabeceras y endpoints
2	grep, ripgrep (rg)	Búsqueda de patrones de riesgo en código fuente
	Claude Opus 4.7 (Anthropic)	Análisis asistido como copiloto, con verificación humana obligatoria
3	curl, openssl s_client	Pruebas de la interfaz pública
	ss, ufw, docker, nginx -T	Inspección de red y configuración en el servidor virtual
	npm audit	Análisis de dependencias del frontend
	pip-audit (no ejecutado — registrado como limitación)	Análisis de dependencias del backend
4	Lectura manual en editor	Verificación contextual de hallazgos

Herramientas empleadas por fase

Nota. SAST: Static Application Security Testing (análisis estático del código fuente).

Fuente: Elaboración propia.

Procedimiento

Diseño de la investigación

El trabajo adopta un diseño de estudio de caso único con muestreo intencional (Yin, 2018), apropiado cuando el objeto analizado revela propiedades de un fenómeno emergente y cuando el investigador cuenta con acceso privilegiado al artefacto (Ambati, 2023). El caso es el sistema antes descrito, construido íntegramente mediante vibe coding (Sarkar & Drosos, 2025) a partir de un único prompt maestro. La auditoría operó en modalidad de caja blanca autorizada, lo que permitió cotejar directamente el artefacto desplegado con su código fuente, algo inviable en una auditoría externa (Páez Poblete, 2024; Pramudia et al., 2025).

Conviene declarar dos limitaciones metodológicas de esta elección. Al tratarse de un caso único, la generalización de los hallazgos queda limitada a su valor ilustrativo. El estudio tampoco incluye un grupo de control desarrollado bajo un paradigma tradicional, lo que impide extraer inferencias causales sobre la contribución específica del vibe coding al perfil de vulnerabilidades observado. Ambas limitaciones se retoman en la Discusión y se proponen como líneas de investigación futura.

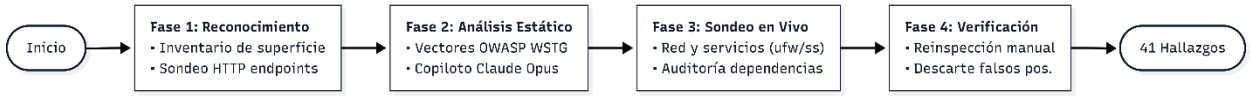
El enfoque metodológico es híbrido y combina tres estrategias complementarias: análisis estático del código fuente (SAST, del inglés Static Application Security Testing) mediante revisión manual asistida, sondeo en vivo del artefacto desplegado en producción y verificación manual orientada al descarte de falsos positivos. La triangulación sigue la OWASP Web Security Testing Guide (OWASP Foundation, 2020) y converge con el marco propuesto por Páez Poblete (2024), que dedica un apartado específico al tratamiento de falsos positivos.

Procedimiento

La auditoría se organizó en cuatro fases secuenciales, ejecutadas durante abril de 2026 y documentadas en un informe técnico interno que sirvió como insumo primario para el apartado de Resultados.

Figura 2

Diagrama de fases del procedimiento de auditoría



Fuente: Elaboración propia.

Criterios de severidad

La clasificación de severidad aplica el marco metodológico OWASP (OWASP Foundation, 2020; Pramudia et al., 2025), que combina dos dimensiones: el impacto potencial sobre los tres pilares de la seguridad de la información (confidencialidad, integridad y disponibilidad) y la complejidad de explotación (conocimiento requerido, acceso previo y precondiciones necesarias para materializar el ataque). Los niveles de severidad se definieron del siguiente modo: Crítico (impacto alto en al menos dos pilares de seguridad y complejidad de explotación baja), Alto (impacto alto en un pilar o medio en varios, con complejidad baja o media), Medio (impacto medio con complejidad media, o impacto alto cuando la complejidad es alta), Bajo (impacto limitado o complejidad alta con precondiciones específicas) e Informativo (señal de diseño subóptimo o divulgación de información sin consecuencias directas). El mapeo al OWASP Top 10:2025 (OWASP Foundation, 2025) se realizó de forma independiente a la asignación de severidad y se recoge en la Tabla 3 del apartado de Resultados, donde los hallazgos se desglosan por categoría OWASP.

Consideraciones éticas

La auditoría siguió tres principios. Primero, divulgación responsable: los hallazgos críticos se mantienen sin pruebas de concepto explotables hasta su remediación; el artículo describe los vectores a nivel conceptual pero omite payloads directamente reutilizables. Segundo, protección de datos de usuarios: la auditoría se realizó antes del lanzamiento público, sin procesar datos personales reales, por lo que el estudio queda fuera del dictamen de un comité de ética, si bien se siguieron los principios del Reglamento General de Protección de Datos (RGPD) y la normativa ecuatoriana aplicable. Tercero, siguiendo la recomendación de Moore & Tatonetti (2025) sobre supervisión metodológica en sistemas generados por vibe coding, la inspección asistida se acompañó siempre de verificación humana, y el modelo no emitió decisiones vinculantes sobre severidad ni descarte.

Resultados

Esta sección presenta los hallazgos empíricos de la auditoría aplicada entre marzo y abril de 2026. Los resultados se organizan en cinco bloques: el inventario general, la distribución por categoría OWASP Top 10:2025, el análisis detallado de hallazgos críticos, el descarte de falsos positivos y las prácticas de seguridad correctamente producidas por el modelo, que matizan la lectura del conjunto.

Inventario general de hallazgos

La auditoría identificó 41 hallazgos confirmados, distribuidos en cinco niveles de severidad conforme a los criterios de severidad definidos en el apartado de Material y métodos. Cinco hallazgos preliminares se descartaron por verificación manual como falsos positivos conforme se detalla en el apartado de resultados. La Tabla 2 resume la distribución.

Tabla 2

Distribución de hallazgos confirmados por severidad

Severidad	Confirmados	Porcentaje
Crítico	5	12,2 %
Alto	11	26,8 %
Medio	10	24,4 %
Bajo	8	19,5 %
Informativo	7	17,1 %
Total	41	100 %

Nota. Elaboración propia a partir de los datos de la auditoría.

Más de un tercio de los hallazgos (16 de 41; 39,0 %) corresponden a severidad crítica o alta, lo que refleja una exposición no trivial pese a que el sistema presenta decisiones de diseño acertadas en capas específicas, detalladas en el apartado de Resultados. La densidad de vulnerabilidades asciende a 3,66 hallazgos por cada 1.000 líneas de código (kLoC), calculada sobre las 11.200 líneas que componen el sistema auditado.

Distribución por categoría OWASP Top 10:2025

La Tabla 3 mapea los 41 hallazgos contra las diez categorías del estándar OWASP Top 10:2025 (OWASP Foundation, 2025). La concentración en A02 (*Security Misconfiguration*) y A07 (*Authentication Failures*), que agrupan 16 de los 41 hallazgos (39,0 %), es el dato más saliente del

caso analizado y sustenta buena parte del análisis desarrollado en la Discusión, donde se contrastan los hallazgos con la teoría del material disengagement.

Tabla 3

Hallazgos por categoría OWASP Top 10:2025

Código	Categoría	Hallazgos
A01	Broken Access Control	2
A02	Security Misconfiguration	9
A03	Software Supply Chain Failures	4
A04	Cryptographic Failures	2
A05	Injection	3
A06	Insecure Design	1
A07	Authentication Failures	7
A08	Software or Data Integrity Failures	6
A09	Security Logging and Alerting Failures	5
A10	Mishandling of Exceptional Conditions	1
Total		41

Nota. A02 y A07 concentran la mayor densidad de hallazgos.

Fuente: Elaboración propia.

A continuación se describe brevemente cada categoría.

A01: Broken Access Control

Dos hallazgos. El primero, de severidad crítica, corresponde a una validación de autorización cosmética en el middleware del frontend: la comprobación se realiza únicamente sobre la existencia de una cookie no protegida (sin atributos HttpOnly ni Secure), sin verificación de la firma del token JWT emitido por el backend. El segundo, de severidad alta, es una exposición del endpoint de exportación en formato de valores separados por comas (CSV, del inglés Comma-Separated Values) que delega la autorización exclusivamente en la cookie de sesión, sin requerir el portador Bearer del JWT, lo que habilita un vector de falsificación de solicitud entre sitios (CSRF) en flujos autenticados.

Security Misconfiguration

Nueve hallazgos, el mayor número del estudio. Dos son críticos: los puertos del frontend (3000/TCP) y del backend (8000/TCP) escuchan en 0.0.0.0 sin cortafuegos activo, lo que permite eludir el proxy inverso Cloudflare y neutralizar sus capas de protección (cortafuegos de aplicación web o WAF, del inglés Web Application Firewall; rate-limiting; y mitigación de ataques de denegación de servicio distribuido o DDoS, del inglés Distributed Denial of Service); y las interfaces de documentación de la interfaz de programación de aplicaciones (API, del inglés Application Programming Interface) (/api/docs, /api/openapi.json, /api/redoc) están expuestas públicamente en producción, divulgando el mapa de endpoints y flujos de autenticación. Los siete restantes abarcan la ausencia total de cabeceras de seguridad HTTP (HyperText Transfer Protocol): HTTP Strict Transport Security (HSTS), Content Security Policy (CSP), X-Frame-Options, X-Content-Type-Options, Referrer-Policy y Permissions-Policy, la divulgación de tecnología vía x-powered-by, falta de validación de longitud mínima para la clave de firma de tokens, una IP privada hardcoded en configuración, divergencia entre Nginx versionado y desplegado, y dos deficiencias en los contenedores (sin directiva USER no privilegiada y sin endurecimiento de Docker Compose: sin cap_drop, read_only, security_opt ni límites de recursos).

Cadena de suministro, criptografía, inyección y diseño

Las cuatro categorías intermedias agrupan diez hallazgos cuyo detalle archivo-línea se consigna en el anexo B del material suplementario. La cadena de suministro (A03, cuatro hallazgos) revela un contraste: npm audit reportó cero vulnerabilidades entre 483 dependencias del frontend, pero el requirements.txt del backend presenta el paquete asyncpg duplicado, psycopg2-binary sin versión fijada, ausencia de lockfile y referencias a imágenes Docker con etiquetas flotantes sin fijación por digest. La categoría de criptografía (A04, dos hallazgos) registra Nginx con TLSv1.0/1.1 (Transport Layer Security, TLS) activos, versiones declaradas obsoletas por la Fuerza de Tarea de Ingeniería de Internet (IETF) desde 2021, junto con el cliente de correo electrónico (SMTP) configurado con VALIDATE_CERTS=False. En cuanto a la inyección (A05, tres hallazgos), destaca un hallazgo de severidad alta: las plantillas HTML de correo electrónico interpolan campos de usuario sin sanitizar mediante f-strings, lo que abre un vector de phishing sobre correos transaccionales. Se suman una inyección de prompt latente en un endpoint no registrado y un parser de lenguaje natural vulnerable a catastrophic backtracking. El diseño inseguro (A06, un hallazgo)

es una discrepancia entre el mensaje de error que promete una política de contraseñas robusta y la expresión regular (^.{8,}$) que no la implementa.

El modelo presenta una distribución sesgada hacia errores de configuración y no de inyección SQL, correctamente mitigada por el uso sistemático del mapeador objeto-relacional (ORM, del inglés Object-Relational Mapper) SQLAlchemy con consultas parametrizadas.

Authentication Failures

Siete hallazgos, la segunda mayor densidad. Dos son críticos: el almacenamiento de ambos tokens (acceso y renovación) en localStorage, que expone las credenciales a cualquier ataque XSS (Cross-Site Scripting); y la política de contraseñas, que acepta cualquier cadena de ocho o más caracteres sin requisito de complejidad. Los cinco restantes abarcan la ausencia de atributos HttpOnly y Secure en la cookie de sesión, la enumeración de usuarios en /auth/register (código 409 diferenciable del 200 de recuperación), la falta de validación de los claims aud e iss en la decodificación JWT, el uso de HS256 simétrico en un contexto multi-inquilino donde RS256 ofrecería mejores propiedades de rotación, y un límite de cinco intentos de inicio de sesión por minuto cuya mitigación recae principalmente en el bloqueo de cuenta tras fallos consecutivos.

Integridad, registro y condiciones excepcionales

La integridad de software y datos (A08, seis hallazgos) se concentra en el módulo de integración con inteligencia artificial: un *router* /ai/chat con código muerto que referencia una variable de configuración inexistente, ausencia de control de costo por usuario, falta de redacción de información personal identificable antes del envío al proveedor, validación ciega del formato de respuesta y *hardcoding* del identificador del modelo. El registro de eventos (A09, cinco hallazgos) revela la ausencia total de logging estructurado para eventos de seguridad: inicios de sesión exitosos y fallidos, uso de tokens de renovación, cierre de sesión, restablecimiento y cambio de contraseña, y operaciones administrativas. A esto se suma la falta de integración con plataformas de agregación o alerta, y logs de contenedor que crecen sin rotación. El tratamiento de condiciones excepcionales (A10, un hallazgo) corresponde al parser de lenguaje natural ya mencionado en A05.

Tres hallazgos críticos representativos

Tres de los cinco hallazgos críticos ilustran el patrón observado en el caso. El primero es la elusión del proxy inverso de protección: los contenedores de frontend y backend escuchan en 0.0.0.0 con el cortafuegos inactivo, lo que permite a cualquier adversario que descubra la IP de origen (trivial

mediante registros históricos del Sistema de Nombres de Dominio (DNS, del inglés Domain Name System) o motores de búsqueda de activos expuestos) saltar Cloudflare y neutralizar la capa de protección perimetral (cortafuegos de aplicación web o WAF, del inglés Web Application Firewall; rate-limiting; y mitigación de ataques de denegación de servicio distribuido o DDoS, del inglés Distributed Denial of Service). El segundo es el almacenamiento de credenciales en localStorage: el módulo contexts/AuthContext.tsx guarda allí tanto el token de acceso como el de renovación, lo que expone ambas credenciales a cualquier vector de cross-site scripting. La práctica canónica consiste en emitirlas mediante cookies marcadas HttpOnly y Secure (Jones et al., 2015), algo que el modelo sí aplicó correctamente en otros flujos descritos en el apartado de Resultados. El tercero es la validación de autorización cosmética en el middleware: el middleware de Next.js se limita a comprobar la existencia de una cookie cf_logged_in sin atributos de seguridad, emitida por el propio cliente. Un adversario que la fije manualmente accede al renderizado del panel privado, aunque el backend sí valida el token JWT en cada solicitud de datos.

Falsos positivos descartados tras verificación

El scaneo asistido por Claude Opus 4.7 generó 46 hallazgos preliminares; la verificación manual línea por línea ejecutada en la etapa de validación descrita en la Metodología, descartó cinco como falsos positivos, con una tasa bruta de error del 10,9 %. La Tabla 4 resume los casos y su justificación.

Tabla 4

Falsos positivos descartados y justificación

#	Hallazgo preliminar	Veredicto tras verificación
1	Server Actions sin validación de sesión	La aplicación no emplea Server Actions; toda la comunicación usa fetch del lado cliente hacia endpoints REST. La búsqueda grep "use server" devolvió cero coincidencias.
2	Asignación masiva mediante payload.model_dump() en cuatro routers	Los schemas Pydantic (SubscriptionCreate, BudgetCreate, CategoryCreate, IncomeSourceCreate) son estrictos y no exponen campos sensibles (user_id, is_admin, is_system). El router inyecta user_id explícitamente fuera del operador de expansión.
3	Verificación de administrador únicamente en el cliente	El backend aplica la dependency require_admin en todos los endpoints administrativos, que valida current_user.is_admin contra la base de datos. El control del cliente es exclusivamente cosmético.
4	Condición de carrera en renovación de tokens	La rotación del token de renovación se ejecuta atómicamente mediante transacción PostgreSQL (rt.revoked = True precede al commit que emite el nuevo hash).
5	AttributeError explotable en /ai/chat	El router no se encuentra registrado en main.py; constituye código muerto sin exposición actual. Reclasificado como riesgo pre-explotable en la categoría A08.

Nota. JWT: JSON Web Token. CSRF: falsificación de solicitud entre sitios (Cross-Site Request *Forgery). CSV: valores separados por comas (Comma-Separated Values).*

Fuente: Elaboración propia.

Esta tasa, que no es despreciable, concuerda con Páez Poblete (2024) sobre la necesidad de reservar una fase dedicada al manejo de falsos positivos en la auditoría asistida por herramientas automatizadas, y se retoma en la Discusión.

Prácticas de seguridad correctamente producidas por el modelo

La lectura de los 41 hallazgos pide un contrapeso: el modelo también produjo, sin que se lo pidiera el prompt, decisiones de seguridad alineadas con el estado del arte. Identificarlas permite caracterizar un *perfil* de fallo asimétrico entre capas, en vez de una generalización indiscriminada sobre el caso. La Tabla 5 enumera las prácticas observadas.

Tabla 5

Prácticas de seguridad emergentes no solicitadas explícitamente

Práctica observada	Ubicación
Hashing de contraseñas con Argon2id (parámetros conservadores)	core/security.py:15-21
Rotación del token de renovación con revocación atómica	auth.py:113-140
Almacenamiento en base de datos del hash SHA-256, no del token	core/security.py:57-65
Bloqueo de cuenta tras cinco intentos fallidos consecutivos	auth.py:153-157
Verificación constante en tiempo para prevenir timing attacks en inicio de sesión	auth.py:77-78
Límite de tasa mediante slowapi en endpoints sensibles	routers/auth.py:23,29,49
Control de acceso administrativo mediante dependency injection	routers/admin.py:20-23
Mensaje genérico en recuperación de contraseña	routers/auth.py:44

Nota. JWT: JSON Web Token. SHA-256: función de hash criptográfico de 256 bits. HttpOnly: atributo de cookie que impide el acceso desde scripts del lado cliente.

Fuente: Elaboración propia.

La asimetría entre las capas donde el modelo adoptó buenas prácticas (hashing, rotación de *tokens*, límite de tasa) y aquellas donde falló (almacenamiento del cliente, configuración de despliegue, registro de eventos) constituye el fenómeno central que se analiza en el apartado de discusión.

Discusión

La evidencia analizada abre cinco líneas interpretativas, que se presentan a continuación antes de declarar las limitaciones del estudio.

Concentración en A02 y A07

El dato más saliente del caso es la concentración del 39,0 % de los hallazgos (16 de 41) en apenas dos categorías del OWASP Top 10:2025: *Security Misconfiguration* (A02, 9 hallazgos) y *Authentication Failures* (A07, 7 hallazgos). Si los errores se distribuyeran uniformemente entre los diez vectores, la expectativa sería cercana al 10 % por categoría; la densidad observada ronda cuatro veces ese valor en A02 y tres veces en A07. La observación se alinea con el marco de Meske et al. (2025) sobre el vibe coding como reconfiguración de la mediación del intento, donde la experticia se desplaza desde la implementación hacia la orquestación y el modelo termina a cargo de capas que el desarrollador deja de auditar. A02 y A07 encajan con lo que los autores denominan *responsibility gaps*: combinan alta especificidad contextual con baja visibilidad para quien programa. El desarrollador presta atención al endpoint con mucho más frecuencia que al Dockerfile, y el modelo carece de un patrón canónico para decidir si el token debe residir en localStorage o en una cookie HttpOnly.

Instrucciones genéricas de seguridad

El prompt instruyó al modelo con la fórmula *"implementa lo que tú consideres el estándar de seguridad más alto para producción"*. Los resultados apuntan a que esa formulación genérica produce una respuesta bimodal: en capas con canon técnico consolidado, el modelo adopta prácticas del estado del arte sin supervisión (Argon2id, rotación atómica del *token* de renovación, *hashing* SHA-256 del *token* en base de datos, *rate-limiting* sobre *endpoints* sensibles); en capas donde la decisión correcta depende del contexto operacional, tiende a reproducir el patrón mayoritario de su corpus de entrenamiento, que no siempre es el adecuado para producción (*tokens* en localStorage, común en tutoriales; documentación OpenAPI expuesta, útil en desarrollo; Nginx con protocolos heredados). La observación converge con la distinción que establece Domínguez Chávez (2025) entre la excelencia en la forma y la profundidad analítica en el contenido de los modelos generativos: el código es sintácticamente impecable y aplica los patrones mejor documentados, pero el modelo no distingue entre el uso apropiado y el uso apropiado en un entorno

de producción multi-tenant desplegado en la nube pública. La implicación práctica es directa: en este caso, los prompts de seguridad debieron ser específicos por vector de ataque, no genéricos.

Meta-advertencia y auto-corrección

Un rasgo del caso merece consideración aparte: el prompt contenía una meta-advertencia sobre la auditoría posterior (material suplementario, Anexo A), de modo que el modelo sabía que su salida sería sometida a pentesting. Aun así, el artefacto presenta cinco hallazgos críticos, lo que sugiere que Claude Sonnet 4.6 no parece activar un régimen de producción más cauto por el hecho de que el usuario le anuncie una auditoría. La observación complementa lo reportado por Geng et al. (2025) y Wang et al. (2024) sobre la ausencia de *security-awareness* en la generación de código por modelos de lenguaje, y deja sin respaldo el supuesto frecuente en la literatura práctica de que meta-instrucciones del tipo “sé cuidadoso” o “esto será auditado” induzcan comportamientos más fiables.

Código de aplicación e infraestructura

La Tabla 3 muestra una diferencia cualitativa entre los hallazgos que residen en el código de aplicación (FastAPI, Next.js) y los que residen en la configuración de infraestructura (Dockerfile, docker-compose.yml, Nginx): el primero adopta decisiones en línea con el estado del arte (Argon2id, bloqueo por intentos fallidos, *rate-limiting*, verificación en tiempo constante), mientras que el segundo concentra hallazgos críticos y altos de carácter estructural (contenedores como root, falta de endurecimiento del orquestador, puertos expuestos, cortafuegos inactivo). Una hipótesis plausible, que este caso único no puede verificar y se deja abierta, es que los corpus de entrenamiento están sesgados hacia código de aplicación curado (repositorios públicos, registros de paquetes, Stack Overflow) y subrepresentan la configuración de infraestructura de producción real, que habitualmente permanece en repositorios privados. La idea es compatible con la caracterización de Mohsin et al. (2024) sobre los modelos como "potential source of new vulnerabilities to the software supply chain": el eslabón débil no estaría en el código que el modelo escribe, sino en el andamiaje que el modelo asume. Contrastar esta hipótesis requiere estudios comparativos con modelos afinados sobre corpus de *DevSecOps*, línea que Ray (2025) señala como prioritaria.

Auditoría asistida y falsos positivos

El 10,9 % de falsos positivos en la primera pasada asistida constituye un resultado metodológico en sí mismo. La auditoría asistida por modelo de lenguaje resulta útil para ampliar el alcance de la inspección, pues cubre rápidamente los diez vectores de la *OWASP Web Security Testing Guide*, pero no sustituye la verificación humana: los cinco falsos positivos descartados encajaban aisladamente con vulnerabilidades conocidas (asignación masiva, condiciones de carrera, control de acceso sólo en el cliente), pero estaban mitigados por mecanismos en capas adyacentes que el modelo no inspeccionó. La lectura concuerda con Páez Poblete (2024) y con su recomendación de reservar una fase específica al manejo de falsos positivos; el diseño aplicado en este estudio, con una fase 4 destinada a la verificación manual, resultó condición necesaria para la fiabilidad del informe. Como circunstancia metodológica adicional conviene registrar que un modelo de la misma familia (Claude Opus 4.7) auditó código generado por otro modelo (Claude Sonnet 4.6); la independencia efectiva entre ambos regímenes y sus implicaciones para el sesgo de la auditoría quedan como pregunta abierta.

Limitaciones del estudio

El presente trabajo es un estudio de caso único, lo que restringe el alcance de cualquier inferencia. Los hallazgos tienen valor ilustrativo para el caso analizado y no permiten generalizar a otros modelos generadores, dominios de aplicación ni grados de especificidad del prompt. El diseño no incluye un grupo de control desarrollado bajo un paradigma tradicional, por lo que no es posible establecer relaciones causales entre el uso de vibe coding y el perfil de vulnerabilidades observado: los datos describen un resultado en este caso, no una causa del paradigma.

Se reconoce, además, un sesgo del investigador: el autor definió el prompt, eligió el stack de referencia, fijó las condiciones de generación y llevó a cabo la auditoría e interpretación de los resultados. Esta concentración de roles puede introducir sesgos involuntarios de familiaridad y de encuadre que la inspección asistida por un segundo modelo no elimina por completo.

Conclusiones

Este trabajo documenta, a partir de un estudio de caso único, el perfil de vulnerabilidades de un SaaS multi-tenant desarrollado íntegramente mediante vibe coding, en el que las decisiones de seguridad fueron delegadas a un modelo de lenguaje dentro de un marco funcional definido por un prompt maestro. En este caso, el 39,0 % de las 41 vulnerabilidades confirmadas evidencia una concentración en dos categorías del OWASP Top 10:2025: Security Misconfiguration (A02) y Authentication Failures (A07). Se observa también un comportamiento diferencial entre capas: el modelo aplicó prácticas alineadas con el estado del arte en componentes con soluciones canónicas, pero mostró debilidades en aspectos que exigen evaluación contextual, como el almacenamiento de tokens, la exposición de interfaces y la configuración de infraestructura. La advertencia anticipada sobre una auditoría posterior no modificó este comportamiento en el caso analizado, y la inspección asistida por un segundo modelo apunta a la conveniencia de incorporar una fase de verificación humana sistemática.

Estos resultados deben interpretarse exclusivamente en el contexto del caso analizado y no permiten inferencias sobre el paradigma en general. La contribución principal del trabajo es la documentación de un caso reproducible que puede servir de base para estudios futuros. En este caso concreto, las instrucciones genéricas de seguridad no resultaron suficientes en un escenario de delegación amplia, lo que sugiere que una especificación más detallada por vector de ataque podría contribuir a reducir el perfil de riesgo. Como líneas de investigación futura, se propone replicar el estudio con otros modelos generadores, variar la especificidad del prompt, extender el análisis a otros dominios y desarrollar diseños comparativos que permitan explorar la contribución del vibe coding al perfil de vulnerabilidades observado.

Referencias bibliográficas

Ambati, S. H. (2023). Security and Authenticity of AI-generated code [Master of Science thesis]. University of Saskatchewan, Department of Computer Science.

Biryukov, A., Dinu, D., & Khovratovich, D. (2016). Argon2: the memory-hard function for password hashing and other applications. 2016 IEEE European Symposium on Security and Privacy (EuroS&P).

Domínguez Chávez, J. (2025). Análisis Crítico de ChatGPT y sus Aplicaciones. [revista por confirmar — disponible en ResearchGate*]*.

Gadde, A. (2025). Democratizing Software Engineering through Generative AI and Vibe Coding: The Evolution of No-Code Development. *Journal of Computer Science and Technology Studies*, *7*(4), 556-572. https://doi.org/10.32996/jcsts.2025.7.4.66

Garcia Berzosa, H. (2024). *Seguridad y privacidad: ¿cómo valorar la seguridad sobre un sistema basado en IA?* [Trabajo Fin de Máster (Máster Universitario en Ciberseguridad y Privacidad, Área M1.886 Privacidad)]. Universitat Oberta de Catalunya.

Geng, F., Shah, A., Li, H., Mulla, N., Swanson, S., Raj, G. S., Zingaro, D., & Porter, L. (2025). Exploring Student-AI Interactions in Vibe Coding. *arXiv preprint arXiv:2507.22614*. https://arxiv.org/abs/2507.22614

Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT)* (RFC 7519). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc7519

Karpathy, A. (2025). *There's a new kind of coding I call «vibe coding»*. Publicación en X/Twitter, 2 de febrero de 2025.

Mahiques Fenollar, M. (2025). Desarrollo de una aplicación web full-stack para pruebas regresivas mediante una metodología de programación asistida por IA: Análisis comparativo con el desarrollo tradicional [Trabajo Fin de Grado (Grado en Ingeniería Informática)]. Universitat Politècnica de València, Escuela Politécnica Superior de Alcoy.

Meske, C., Hermanns, T., Weiden, E. von der, Loser, K.-U., & Berger, T. (2025). Vibe Coding as a Reconfiguration of Intent Mediation in Software Development: Definition, Implications, and Research Agenda. *arXiv preprint arXiv:2507.21928*. https://arxiv.org/abs/2507.21928

Mohsin, A., Janicke, H., Wood, A., Sarker, I. H., Maglaras, L., & Janjua, N. (2024). Can We Trust Large Language Models Generated Code? A Framework for In-Context Learning, Security Patterns, and Code Evaluations Across Diverse LLMs. *arXiv preprint arXiv:2406.12513*. https://arxiv.org/abs/2406.12513

Moore, J. H., & Tatonetti, N. (2025). Vibe coding: a new paradigm for biomedical software development. *BioData Mining*, *18*(46). https://doi.org/10.1186/s13040-025-00462-9

OWASP Foundation. (2020). *Web Security Testing Guide v4.2*. OWASP Foundation.
OWASP Foundation. (2025). *OWASP Top 10:2025*. https://owasp.org/Top10/2025/

Páez Poblete, E. R. (2024). Aplicación de herramientas para el análisis estático y dinámico de código en aplicaciones web [Trabajo Final (Tecnicatura Universitaria en Programación Web)]. Universidad Nacional de San Juan, Facultad de Ciencias Exactas, Físicas y Naturales, Departamento de Informática.

Pramudia, A., Suhartana, M., & Hassan, R. (2025). Simulation Vulnerabilities Web Application using Top

10 OWASP Approach. *Asia-Pacific Journal of Information Technology and Multimedia (APJITM)*, *14*(2), 595-605. https://doi.org/10.17576/apjitm-2025-1402-16

Ray, P. P. (2025). A Review on Vibe Coding: Fundamentals, State-of-the-art, Challenges and Future Directions. *TechRxiv preprint*. https://doi.org/10.36227/techrxiv.175475402.27450434

Sarkar, A., & Drosos, I. (2025). Vibe coding: programming through conversation with artificial intelligence. *arXiv preprint arXiv:2506.23253*. https://arxiv.org/abs/2506.23253

Wang, J., Cao, L., Luo, X., Zhou, Z., Xie, J., Jatowt, A., & Cai, Y. (2024, abril). Enhancing Large Language Models for Secure Code Generation: A Dataset-driven Study on Vulnerability Mitigation. *Proceedings of the 46th International Conference on Software Engineering (ICSE '24)*. https://arxiv.org/abs/2310.16263

Yin, R. K. (2018). *Case Study Research and Applications: Design and Methods* (6.ª ed.). SAGE Publications.

Material suplementario

El material suplementario (prompt maestro, listado completo de hallazgos y comparación de stack) y el código fuente del sistema auditado están disponibles en: https://github.com/danilogalloec/cashflow-nlp

Conflicto de intereses:

El autor declara que no existe conflicto de interés.

Financiamiento:

El artículo no recibió asistencia financiera de partes externas; los costos de infraestructura fueron asumidos por el autor con recursos propios.

Agradecimiento:

Agradezco profundamente a mi esposa e hijos por su apoyo incondicional, su paciencia y por ser el motor que impulsó cada jornada de este proceso de investigación; así como a mis padres, por ser el pilar fundamental de mi formación personal y profesional. De igual manera, extiendo mi gratitud a la Universidad Tecnológica Indoamérica por el respaldo institucional brindado y a la Dra. Fátima Avilés-Castillo por su invaluable orientación técnica y académica durante la fase de auditoría

Nota:

El artículo no es producto de una publicación anterior.