UIDE
Powered by
Arizona State University®

eig

Maestría en

CIBERSEGURIDAD

Trabajo previo a la obtención de título de
Magister en Ciberseguridad

AUTOR/ES:

Saskia Jamilé García Zambrano

Dayanara Abigail Terán Piedra

Edisson Fernando Paredes Nacipucha

Daniel Alejandro Dávila Cárdenas

TUTOR/ES:

Iván Reyes

Alejandro Cortés

TEMA:

DISEÑO DE UNA METODOLOGÍA DE PENTESTING PARA APIS RESTFUL BASADA EN LOS CINCO PRINCIPALES RIESGOS DEL OWASP API SECURITY TOP 10 (2023)

Quito - Ecuador

Diciembre - 2025

REINVENTEMOS
EL FUTURO

Certificación De Autoría

Nosotros, Saskia Jamilé García Zambrano, Dayanara Abigail Terán Piedra, Edisson Fernando Paredes Nacipucha y Daniel Alejandro Dávila Cárdenas**,** 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.

Firma

Saskia Jamilé García Zambrano

Firma

Dayanara Abigail Terán Piedra

Firma

Edisson Fernando Paredes Nacipucha

Firma

Daniel Alejandro Dávila Cárdenas

Autorización De Derechos De Propiedad Intelectual

Nosotros, Saskia Jamilé García Zambrano, Dayanara Abigail Terán Piedra, Edisson Fernando Paredes Nacipucha y Daniel Alejandro Dávila Cárdenas, en calidad de autores del trabajo de investigación titulado Título del trabajo de investigación “Diseño de una metodología de Pentesting para APIs RESTful basada en los cinco principales riesgos del OWASP API Security Top 10 (2023)”, 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)

Firma

Saskia Jamilé García Zambrano

Firma

Dayanara Abigail Terán Piedra

Firma

Edisson Fernando Paredes Nacipucha

Firma

Daniel Alejandro Dávila Cárdenas

Aprobación De Dirección Y Coordinación Del Programa

Nosotros, **Alejandro Cortés e Iván Reyes**, declaramos que: **Saskia Jamilé García Zambrano, Dayanara Abigail Terán Piedra, Edisson Fernando Paredes Nacipucha y Daniel Alejandro Dávila Cárdenas,** 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

Daniel Dávila:

Dedico este trabajo a mi amada hija Martina, por ser mi motivación e impulso para realizar esta maestría; a mis padres, por su valioso apoyo durante las clases; y a mis amigos y amigas de la maestría, por compartir este proceso conmigo. Sin ustedes, este logro no habría sido posible.

Fernando Paredes:

Dedico este trabajo de Maestría a mi Esposa la MsC. Carolina Samaniego Magallanes, quien ha sido mi mayor pilar fundamental, para poder lograr todas mis metas, y a mi familia por confíar en mí, brindándome palabras de motivación, apoyándome en todo este proceso de formación.

Dayanara Teran:

Dedico este trabajo a mis seres más cercanos: mi familia y mis gatitos. Este logro representa un paso importante en mi formación académica y profesional, y lo comparto con ellos por ser parte fundamental de este proceso.

Saskia García:

Dedico este trabajo de Maestría a mis seres más amados, mi familia. Sin su apoyo incondicional, este logro no habría sido posible. Este ha sido un paso trascendental en mi carrera profesional, y agradezco profundamente la confianza, el cariño y la motivación que siempre me han brindado.

AGRADECIMIENTOS

Daniel Dávila:

En primer lugar, agradezco a la Virgen María por haberme dado fuerzas y salud para recorrer este camino. A mi amada hija Martina, por ser siempre mi alegría y la fuerza que me impulsa en cada paso de la vida, tanto como padre y como estudiante. A mis padres, por su apoyo incondicional y desinteresado, aun sin conocer la meta que estaba por alcanzar; y, finalmente, a mis amigos y amigas de la maestría, gracias por su compañerismo, trabajo en equipo y perseverancia, que hicieron más llevadero este proceso.

Dayanara Teran:

Agradezco a cada una de las personas que me acompañaron a lo largo de este proceso. Su apoyo y guía fueron clave para superar cada reto y avanzar con determinación durante este logro académico.

Fernando Paredes:

Agradezco infinitamente a Dios, por brindarme sabiduría y entendimiento necesario para continuar con mis estudios, así mismo a la Prestigiosa Universidad UIDE, que me permitió estudiar y adquirir nuevos conocimientos, para mi vida profesional, así mismo a mis Docentes los cuales me brindaron apoyo en este proceso de titulación.

Saskia García:

Agradezco profundamente a Dios y a todas las personas que hicieron posible la culminación de esta maestría. A mi familia, por su amor incondicional, su paciencia y su apoyo constante; su confianza en mí ha sido la fuerza que me ha impulsado en cada etapa, no solo durante este proceso académico, sino a lo largo de toda mi vida.

RESUMEN

La presente investigación aborda la seguridad crítica en arquitecturas de software modernas, centrándose específicamente en el análisis de vulnerabilidades dentro de Interfaces de Programación de Aplicaciones (API) bajo el estándar REST. Ante la creciente brecha entre las amenazas teóricas y la implementación práctica de medidas defensivas, este estudio tuvo como objetivo principal la identificación y mitigación de riesgos en endpoints inseguros mediante la ejecución de pruebas de penetración controladas. La metodología empleada consistió en el despliegue de un laboratorio virtualizado y aislado (utilizando VirtualBox y Docker), el cual alojó la aplicación deliberadamente vulnerable VAmPI. Se adoptó un enfoque de auditoría híbrido, integrando la eficiencia del escaneo automatizado mediante OWASP ZAP, con la precisión del análisis manual a través de Burp Suite. Los resultados experimentales permitieron la detección y explotación de fallos críticos alineados con el OWASP API Security Top 10, tales como Broken Object Level Authorization (BOLA), Inyección SQL y exposición masiva de datos sensibles. El trabajo concluye que la combinación de validación humana y herramientas automatizadas es indispensable para garantizar la confidencialidad, integridad y disponibilidad de la información, proporcionando un marco de recomendaciones técnicas para robustecer el ciclo de desarrollo de software seguro.

Palabras Claves:

Ciberseguridad, API REST, Pentesting Híbrido, OWASP API Security Top 10, Vulnerabilidades Web.

ABSTRACT

This research addresses critical security challenges in modern software architectures, focusing specifically on the analysis of vulnerabilities within Application Programming Interfaces (APIs) built under the REST standard. In response to the growing gap between theoretical threats and the practical implementation of defensive measures, the primary objective of this study was to identify and mitigate risks in insecure endpoints through the execution of controlled penetration tests. The methodology involved the deployment of a virtualized and isolated laboratory environment (using VirtualBox and Docker), which hosted the deliberately vulnerable application VAmPI. A hybrid auditing approach was adopted, integrating the efficiency of automated scanning with OWASP ZAP and the precision of manual analysis using Burp Suite. Experimental results enabled the detection and exploitation of critical flaws aligned with the OWASP API Security Top 10, such as Broken Object Level Authorization (BOLA), SQL Injection, and excessive exposure of sensitive data. The study concludes that combining human-driven validation with automated tools is essential to ensure the confidentiality, integrity, and availability of information, providing a technical framework of recommendations to strengthen the secure software development lifecycle.

Keywords:

Cybersecurity, REST API, Hybrid Pentesting, OWASP API Security Top 10, Web Vulnerabilities.

TABLA DE CONTENIDOS (Índice)

CAPÍTULO 1:..... 1
1. Introducción..... 1
1.1. Definición del proyecto..... 1
1.2. Justificación e importancia del trabajo de investigación..... 3
1.3. Alcance..... 3
1.4. Objetivos..... 4
1.4.1. Objetivo general..... 4
1.4.2. Objetivo específico..... 4
CAPÍTULO 2:..... 5
2. Revisión de la literatura..... 5
2.1. Estado del Arte..... 5
2.2. Marco Teórico..... 11
2.2.1. Arquitecturas Web: De Monolitos a Microservicios..... 11
2.2.2. Seguridad en APIs REST y la Evolución de Amenazas (OWASP)..... 12
2.2.3. Metodologías de Pruebas de Seguridad en APIs marco OWASP..... 13
2.2.4. Laboratorios Virtuales: Infraestructura para Entornos Controlados..... 14
2.2.5. Pentesting de APIs REST en entorno controlado con OWASP (concepto central)..... 15
CAPÍTULO 3:..... 16
3. Desarrollo..... 16
3.1. Levantamiento de entorno seguro..... 17
Figuras 1. Diagrama de arquitectura de aplicación..... 17
3.2. Configuración de Red..... 18
Tabla 1. Matriz de Direccionamiento IP..... 19
3.3. Creacion de Servidor Vulnerable..... 20
Figuras 2. Diagrama de arquitectura de virtualización..... 21
3.4. Creación de Ambiente de atacante..... 22
3.5. Validación de funcionalidad de Endpoints..... 23
Tabla 2. Matriz de Rutas de Swagger..... 24
4. Ataques..... 25
4.1. Escaneo..... 25
4.1.1. Comando utilizado..... 26
4.1.2. Pasos..... 26
Figuras 3. Escaneo con nmap para puertos abiertos..... 27
Figuras 4. Ingreso a ip por el puerto abierto del servidor..... 28
Figuras 5. Interface de endpoints de swagger..... 29
4.2. Admin Endpoint Abuse..... 31
4.2.1. Riesgos..... 32
4.2.2. Ruta:..... 33
4.2.3. Pasos:..... 34
Figuras 6. Interceptación Manual con Burp Suite..... 35
Figuras 7. Envío de la Petición Directamente (Explotación)..... 36
Figuras 8. Confirmar el Impacto (Post-Explotación)..... 37

4.3. Mass User Data Exposure via Debug Endpoint....38
4.3.1. Riesgos....39
4.3.2. Ruta....40
4.3.3. Pasos....41
Figuras 9. Capturarla en Proxy....41
Figuras 10. Verificación de acceso....42
4.4. Authentication Bypass and Credential Exploitation....43
4.4.1. Riesgos:....44
4.4.2. Ruta....45
4.4.3. Pasos....46
Figuras 11. Token del usuario admin extraído....46
Figuras 12. Validar el JWT accediendo....47
4.5. SQL Injection por Validación Deficiente de Entradas en Endpoints de Usuario....48
4.5.1. Riesgos....49
4.5.2. Ruta....50
4.5.3. Pasos....51
Figuras 13. Petición mediante Burp Suite....51
Figuras 14. Lanzar una consulta automatizada con sqlmap....52
4.6. Secuestro de Cuentas por Autorización Defectuosa en Endpoints de Propiedad....53
4.6.1. Riesgos....54
4.6.2. Ruta....54
4.6.3. Pasos....55
Figuras 15. Token del usuario name1....55
Figuras 16. Atacante aprovecha un IDOR....56
Figuras 17. Verifica el éxito del ataque....57
CAPÍTULO 4:....58
5. Análisis de resultados....58
5.1. Pruebas de Concepto....58
5.1.1. Abuso de Endpoints Administrativos (Admin Endpoint Abuse)....58
5.1.2. Exposición Masiva de Datos (Mass User Data Exposure)....59
5.1.3. Inyección SQL (SQL Injection)....60
5.1.4. Secuestro de Cuentas mediante BOLA (Account Takeover)....61
5.1.5. Secuestro de Cuentas (Account Takeover via BOLA)....62
5.2. Resultados....63
5.2.1. Reinicio de Base de Datos no Autorizado....64
5.2.2. Exposición de Datos Sensibles (PII)....66
5.2.3. Enumeración de Usuarios....67
5.2.4. Evaluación Vectorial para Inyección SQL....68
5.2.5. Cambio de Contraseña Arbitrario (BOLA)....71
Tabla 3. Matriz de Vulnerabilidades....72
CAPÍTULO 5:....73
6. CONCLUSIONES Y RECOMENDACIONES....73
6.1. Conclusiones....73
6.2. Recomendaciones....75

Apéndice A. Laboratorio Máquina Atacante..... 79
Apéndice B. Laboratorio servidor..... 79
Referencias Bibliográficas..... 80

LISTA DE TABLAS (Índice de tablas)

Tabla 1. Matriz de Direccionamiento IP..... 19
Tabla 2. Matriz de Rutas de Swagger..... 24
Tabla 3. Matriz de Vulnerabilidades..... 72

LISTA DE FIGURAS (Índice de figuras)

Figuras 1. Diagrama de arquitectura de aplicación..... 17
Figuras 2. Diagrama de arquitectura de virtualización.....21
Figuras 3. Escaneo con nmap para puertos abiertos.....27
Figuras 4. Ingreso a ip por el puerto abierto del servidor.....28
Figuras 5. Interface de endpoints de swagger.....29
Figuras 6. Interceptación Manual con Burp Suite..... 35
Figuras 7. Envío de la Petición Directamente (Explotación)..... 36
Figuras 8. Confirmar el Impacto (Post-Explotación)..... 37
Figuras 9. Capturarla en Proxy.....41
Figuras 10. Verificación de acceso..... 42
Figuras 11. Token del usuario admin extraído.....46
Figuras 12. Validar el JWT accediendo.....47
Figuras 13. Petición mediante Burp Suite.....51
Figuras 14. Lanzar una consulta automatizada con sqlmap.....52
Figuras 15. Token del usuario name1..... 55
Figuras 16. Atacante aprovecha un IDOR..... 56
Figuras 17. Verifica el éxito del ataque.....57

CAPÍTULO 1:

1. Introducción

1.1. Definición del proyecto

En la arquitectura de software contemporánea, las Interfaces de Programación de Aplicaciones (API) bajo el estilo arquitectónico REST se han consolidado como la columna vertebral de la comunicación digital, facilitando la interacción fluida entre sistemas distribuidos, aplicaciones móviles y servicios basados en la nube.

El problema central que aborda esta investigación radica en la brecha existente entre las amenazas teóricas documentadas específicamente aquellas detalladas en el OWASP API Security Top 10 y la implementación práctica de pruebas de penetración efectivas. Por ello, resulta crucial establecer un entorno seguro, controlado y reproducible que permita ejecutar ataques contra un backend desarrollado en Python Flask, con el fin de identificar vulnerabilidades, clasificar su riesgo y proponer las mitigaciones pertinentes.

La presente investigación propone la creación de un Laboratorio de Seguridad especializado en APIs REST diseñado para abordar el problema identificado. Este proyecto utiliza la aplicación deliberadamente vulnerable VAmPI en una VM con un docker como entorno de prueba central. La metodología de análisis se fundamenta en un enfoque híbrido que combina:

Análisis Manual Interceptado (Burp Suite): Utilizado como proxy interceptor para la exploración detallada de vulnerabilidades críticas como fallos de autenticación, autorización y validación de datos.

Escaneo Automatizado (OWASP ZAP): Complementa la detección de fallos en la configuración de seguridad y la exposición de información sensible.

Este enfoque dual permite simular vectores de ataque reales, con el objetivo de analizar las vulnerabilidades presentes en endpoints inseguros de API REST, con el fin de identificar riesgos y proponer estrategias de mitigación.

El proyecto se despliega en un entorno aislado y reproducible mediante un laboratorio virtualizado en VirtualBox (v7.2.2), estructurado en dos máquinas virtuales interconectadas y organizadas bajo una segmentación de red una Red de Gestión tipo Host-Only (192.168.56.0/29), dedicada exclusivamente a la administración interna de las VMs sin exposición a la red de pruebas; y una Red de Laboratorio basada en NAT Network (10.10.10.0/29), que constituye el dominio operativo principal al habilitar la interacción entre el servidor VAmPI, la máquina atacante Kali y el cliente, además de proporcionar acceso controlado a Internet para actualizaciones sin comprometer el aislamiento del entorno.

1.2. Justificación e importancia del trabajo de investigación

En la actualidad, las API REST constituyen la columna vertebral de la comunicación entre sistemas, aplicaciones móviles y servicios en la nube. Sin embargo, su uso cada vez más frecuente ha incrementado la superficie de ataque, convirtiéndo se en un objetivo prioritario para los ciberdelincuentes debido a su exposición constante a internet.

El análisis de endpoints inseguros permite comprender los errores de diseño y configuración que afectan la seguridad de los datos, con el fin de prevenir fugas de información; además, facilita el establecimiento de medidas preventivas y correctivas para robustecer la infraestructura empresarial y proponer un marco metodológico que los equipos de desarrollo y seguridad puedan implementar para reforzar la ciberseguridad. La importancia de este proceso radica en fortalecer la seguridad organizacional, garantizando la confidencialidad, integridad y disponibilidad de la información, así como en concientizar a los desarrolladores y administradores sobre los riesgos asociados al uso de APIs.

1.3. Alcance

- La investigación se centra exclusivamente en API REST.
- Analizar vulnerabilidades relacionadas con endpoints inseguros (IDOR, mass assignment, CORS, tokens inseguros, entre otros).
- Pruebas de ataque relacionadas con inyección, control de acceso, limitación de velocidad y validación insuficiente de parámetros.
- Implementar un laboratorio controlado mediante el uso de máquinas virtuales con aplicaciones deliberadamente vulnerables para experimentar.
- No se realizarán pruebas sobre sistemas reales (producción), ni en entornos de terceros.

- Identificar las vulnerabilidades según la guía de OWASP API Security Top 10.
- Elaborar un informe con hallazgos, evidencias y recomendaciones de mitigación.

1.4. Objetivos

1.4.1. Objetivo General.

Analizar las vulnerabilidades presentes en endpoints inseguros de API REST mediante pruebas de penetración controladas en ambientes virtuales, con el fin de identificar riesgos y proponer estrategias de mitigación.

1.4.2. Objetivo Específico

- Configurar un laboratorio de pruebas en máquinas virtuales que simule un entorno corporativo con APIs vulnerables.
- Identificar y clasificar vulnerabilidades comunes en endpoints de API REST basadas en OWASP API Security Top 10.
- Aplicar técnicas de pentesting para comprobar riesgos asociados a la exposición de datos y fallos de control de acceso.
- Elaborar una sección en el marco teórico informe con recomendaciones de seguridad para reducir las amenazas detectadas.

CAPÍTULO 2:

2. Revisión De La Literatura

2.1. Estado del Arte

En el trabajo de investigación de Álava, Basurto y Tóala (2022), se aborda la relevancia de la seguridad informática, la cual se define como el área encargada de diseñar reglas y técnicas para asegurar la confiabilidad y disponibilidad del sistema de información (Molina & Orozco, 2020). Dicho estudio enfatiza que la tarea principal de la seguridad informática es reducir los riesgos derivados de vulnerabilidades, entendidas como fallas o debilidades en un sistema que pueden ser explotadas por amenazas informáticas (Torres, 2021; Tarazona, 2006). Los resultados del análisis, referenciados con el Open Web Application Security Project (OWASP), destacaron la incidencia recurrente de vulnerabilidades en casos de alto perfil: se identificó la vulnerabilidad A03: Inyección en el ataque a Wipro (2019), la A05: Configuración de Seguridad Incorrecta en Giant Eagle (2020), y la A02: Fallas Criptográficas en Credit Karma (2021). Además, se correlacionaron incidentes con los riesgos del OWASP Top 10 2021, como la Pérdida de Control de Acceso (A01:2023) en British Airways (2018) y la Falsificación de Solicitudes del Lado del Servidor (A10:2023) en Equifax (2017), subrayando la persistencia de estos fallos críticos en el panorama global de la ciberseguridad.

En el Trabajo de Fin de Grado titulado Seguridad en APIs: Análisis de Riesgos, Pruebas de Penetración y Mitigaciones (Universidad Rey Juan Carlos, 2024), se realiza un análisis detallado de los riesgos más críticos definidos en el OWASP API Security Top 10 de 2023, examinando distintas propuestas y estrategias de mitigación. El estudio se centra en las distintas modalidades de pentesting (caja negra, caja gris y caja blanca) para la auditoría de APIs, enfatizando la importancia de establecer formalmente las autorizaciones y el alcance de las pruebas antes de su inicio. Los resultados finales incluyen la realización de demostraciones de pruebas controladas en

un entorno simulado , con el propósito específico de vulnerar los riesgos identificados en las APIs. Este enfoque proporciona una perspectiva práctica y visual que facilita la comprensión de cómo un atacante o auditor podría explotar las vulnerabilidades BOLA (Autorización Rota a Nivel de Objeto), resaltando la necesidad imperativa de implementar medidas de seguridad robustas desde la fase de desarrollo.

En la línea de la justificación metodológica, la investigación se apoya en la tesis Creación de un entorno virtual de aprendizaje para un laboratorio de enseñanza de seguridad informática en carreras técnicas (Castro-León & Rendón-Burgos, 2021) y el trabajo Reingeniería del Laboratorio de Seguridad Informática: análisis, diseño e implementación de un Cyber Range (Gallardo & Guerrero, 2021). Estos estudios académicos abordan la necesidad de laboratorios especializados en ciberseguridad para el desarrollo de habilidades prácticas en pentesting y hacking ético. Los resultados indican que estos entornos, caracterizados por su bajo costo y su capacidad para replicar escenarios realistas, contribuyen al desarrollo integral de habilidades prácticas en los estudiantes (Castro-León & Rendón-Burgos, 2021; Fuentes-Penna et al., 2021). La implementación de un Cyber Range, por su parte, ofrece una visión avanzada del laboratorio virtual al integrar la seguridad con tecnologías de orquestación, asegurando un entorno de ejecución controlado y alineado con estándares de seguridad como ENS y NIST (Gallardo & Guerrero, 2021; MDPI, 2023).

En cuanto a la detección avanzada de fallos de lógica de negocio, el artículo VOAPI2: A Vulnerability-Oriented Testing Approach for RESTful APIs (Du et al., 2024) propone un método novedoso para automatizar la revelación de vulnerabilidades en APIs REST en la nube. Este trabajo es una respuesta directa a los desafíos de los métodos de escaneo convencionales, que han mostrado limitaciones al analizar dependencias funcionales y generar casos de prueba que rompan

la secuencia lógica (Kim et al., 2024). Los resultados del experimento demuestran que el enfoque VOAPI2 supera a los métodos de vanguardia por un margen significativo , destacando su eficacia no solo en el número de fallos identificados sino también en la eficiencia de detección. El método se basa en pruebas orientadas a vulnerabilidades, aprovechando las especificaciones de la API para capturar defectos de forma efectiva.

En el contexto de la evolución de las amenazas, la aplicación de técnicas de Inteligencia Artificial (IA) y Aprendizaje Automático (Machine Learning) ha emergido como un campo de investigación crítico para la defensa de APIs. A diferencia de los sistemas basados en firmas estáticas, los modelos de aprendizaje profundo permiten la identificación de patrones de tráfico anómalos en tiempo real, detectando ataques sutiles que evaden los WAF tradicionales (Capuano & Fenza, 2022). Investigaciones recientes sugieren que la implementación de redes neuronales para el análisis del comportamiento del usuario (UEBA) reduce significativamente los falsos positivos en la detección de exfiltración de datos, abordando eficazmente la complejidad de las arquitecturas distribuidas donde el perímetro de seguridad es difuso (Sarker, 2021; Li et al., 2023). Sin embargo, la literatura también advierte sobre el uso adversarial de la IA, donde los atacantes emplean algoritmos generativos para automatizar la creación de payloads de inyección polimórficos, lo que exige una evolución constante de los mecanismos defensivos (Kumar et al., 2024).

El paradigma de la Interfaz de Programación de Aplicaciones (API) REST se ha establecido como el componente fundamental en las arquitecturas de software distribuidas, impulsando ecosistemas basados en microservicios y cloud-native (IBM, n.d.; Frugal Testing, n.d.). Esta ubicuidad ha provocado una expansión significativa en la superficie de ataque, lo que

exige estrategias de seguridad especializadas que superen las limitaciones de la protección perimetral tradicional (OWASP Foundation, n.d.).

La omisión de procesos de prueba rigurosos, especialmente bajo la presión de los ciclos de Integración Continua/Entrega Continua (CI/CD), es una causa directa de la proliferación de vulnerabilidades (Wiz, n.d.). Para mitigar este riesgo, las directrices contemporáneas, como la Publicación Especial del National Institute of Standards and Technology (NIST SP 800-228), promueven la adopción de controles avanzados de pre-ejecución y tiempo de ejecución, además de un enfoque de Confianza Cero (Zero Trust), que requiere la revalidación estricta de la autorización en cada solicitud, incluso entre servicios internos.

Paralelamente, la industria ha experimentado un cambio de paradigma hacia estrategias de seguridad proactiva conocidas como Shift-Left, las cuales abogan por la integración de controles de seguridad desde las fases iniciales del Ciclo de Vida de Desarrollo de Software (SDLC). Estudios comparativos han demostrado que la corrección de vulnerabilidades durante la fase de diseño y codificación es exponencialmente menos costosa que la remediación en producción (Spinellis et al., 2021). En el ámbito de las APIs REST, esto se materializa mediante la validación automatizada de contratos OpenAPI (anteriormente Swagger) para asegurar que la implementación técnica cumpla estrictamente con las especificaciones de seguridad definidas, mitigando riesgos de exposición de datos no documentados o "Shadow APIs" (Mougouei et al., 2023). Este enfoque se alinea con la metodología DevSecOps, que busca romper los silos entre los equipos de desarrollo, operaciones y seguridad, automatizando pruebas de análisis estático (SAST) y dinámico (DAST) dentro de los pipelines de despliegue (Myrbakken & Colomo-Palacios, 2017).

Por último, la adopción de arquitecturas de Confianza Cero (Zero Trust Architecture) ha redefinido los requisitos de autenticación y autorización en la comunicación entre microservicios. Según los principios de Zero Trust, no se debe confiar implícitamente en ningún actor, independientemente de si se encuentra dentro o fuera del perímetro de la red (Rose et al., 2020). La literatura académica enfatiza la necesidad de implementar mallas de servicios (Service Meshes) que gestionen la identidad de cada servicio mediante certificados mTLS y validación continua de tokens JWT, asegurando que cada petición API sea autenticada, autorizada y cifrada individualmente (Buck et al., 2021). Esta granularidad es fundamental para prevenir el movimiento lateral de atacantes en caso de que un contenedor o microservicio sea comprometido, limitando el radio de impacto de una intrusión exitosa (Teerakanok et al., 2021).

El marco de referencia estándar para clasificar estos riesgos es el OWASP API Security Top 10 (2023), que se enfoca en las debilidades específicas del diseño API, siendo la vulnerabilidad más crítica la Autorización Rota a Nivel de Objeto (Broken Object Level Authorization, BOLA, API1:2023) . BOLA se produce cuando una API expone identificadores de objeto y falla en verificar si el usuario autenticado tiene el derecho legítimo de acceder a dicho recurso . Este fallo pertenece a la categoría de Fallos de Lógica de Negocio (FLN), los cuales no son errores de configuración, sino debilidades en la implementación funcional del sistema que permiten acciones no intencionadas (PortSwigger, n.d. ; Wallarm, n.d.). Investigaciones académicas recientes han destacado la dificultad de las metodologías DAST (Dynamic Application Security Testing) tradicionales para detectar estos FLN, debido a la complejidad de analizar dependencias funcionales y generar casos de prueba que rompan la secuencia lógica (Kim et al., 2024; Du et al., 2024). La capacidad de un atacante para explotar estos FLN se basa en el conocimiento profundo del modelo de datos y las restricciones del servidor .

Consecuentemente, la literatura técnica respalda la necesidad de un modelo de prueba de seguridad híbrido que combine la cobertura de amplio espectro con la capacidad de análisis profundo (Kualitatem, n.d.; Frugal Testing, n.d.). Este enfoque sinérgico es crucial para superar las deficiencias recíprocas de las herramientas; mientras que escáneres automatizados como OWASP ZAP se emplean para la cobertura rápida de fallos conocidos y pruebas de regresión , la inspección manual, ejecutada por herramientas como Burp Suite, resulta indispensable para la intercepción de tráfico, la manipulación detallada de payloads y la explotación controlada de FLN y vulnerabilidades BOLA, ya que requiere el juicio humano para evaluar comportamientos inesperados (PortSwigger, n.d.; TechMagic, 2024; Apisec.ai, 2024). Por lo tanto, el enfoque híbrido maximiza la eficiencia de la auditoría al asignar a la automatización el rol de velocidad y a la intervención manual el rol de precisión y profundidad (GetAstra, n.d.; Apisec.ai, 2024).

Finalmente, el diseño experimental de este proyecto requiere un entorno de laboratorio controlado y aislado para asegurar la ética y la reproducibilidad de los resultados . El uso de la contenerización (Docker) sobre hipervisores de Tipo 2 (como VirtualBox) garantiza la portabilidad y la consistencia del ambiente de ejecución, un factor clave para la validez de los estudios comparativos (Javed & Toor, 2021; University of North Dakota, n.d.). Además, el valor pedagógico de estos entornos controlados ha sido reconocido en trabajos académicos que enfatizan la necesidad de laboratorios de ciberseguridad para el desarrollo de habilidades prácticas en pentesting y hacking ético (Castro-León & Rendón-Burgos, 2021; Gallardo & Guerrero, 2021; MDPI, 2023). La elección de VAmPI está validada por su diseño para simular directamente los riesgos del OWASP API Top 10 y ofrecer un interruptor global para alternar entre estados vulnerable y seguro, lo cual es esencial para la evaluación objetiva de la precisión de las herramientas de detección de vulnerabilidades.

2.2. Marco Teórico

2.2.1. Arquitecturas Web: De Monolitos a Microservicios

La ingeniería de software ha transitado desde arquitecturas monolíticas hacia ecosistemas distribuidos basados en microservicios. En el modelo monolítico, la lógica de negocio, la interfaz y el acceso a datos residen en una unidad única, lo que dificulta la escalabilidad y el mantenimiento (Atlassian, s.f.). En contraste, la arquitectura de microservicios descompone las aplicaciones en servicios autónomos que se comunican a través de protocolos ligeros, predominantemente HTTP/REST (Microsoft, 2023).

Este cambio de paradigma ha reposicionado a las Interfaces de Programación de Aplicaciones (API) como el componente crítico de la infraestructura digital moderna. Según reportes recientes de la industria, el tráfico de APIs representa más del 50% del tráfico web dinámico global, convirtiéndo se en el vector de ataque principal debido a su exposición de lógica de negocio y datos sensibles (OpenText, 2023). La seguridad, por tanto, ha dejado de ser perimetral para requerir controles granulares en cada punto de interacción.

2.2.2. Seguridad en APIs REST y la Evolución de Amenazas (OWASP)

La seguridad en APIs difiere sustancialmente de la seguridad web tradicional. El Open Web Application Security Project (OWASP) aborda esta distinción mediante el OWASP API Security Top 10, cuya actualización más reciente en 2023 refleja la sofisticación de los ataques.

La vulnerabilidad más crítica continúa siendo la Autorización Rota a Nivel de Objeto (BOLA) (API1:2023), donde los atacantes manipulan identificadores de objetos en las solicitudes para acceder a datos ajenos, un fallo que persiste debido a la dificultad de detección mediante herramientas automatizadas tradicionales (OWASP, 2023; Pasca et al., 2025).

Asimismo, la edición 2023 introdujo la Autorización Rota a Nivel de Propiedad de Objeto (BOPLA) (API3:2023), fusionando la exposición excesiva de datos y la asignación masiva (Mass Assignment), destacando la necesidad de validar estrictamente qué propiedades internas pueden ser expuestas o modificadas por el usuario (OWASP API Security Project Team, 2023).

2.2.3. Metodologías de Pruebas de Seguridad en APIs marco OWASP

Para auditar estas arquitecturas, existen marcos de referencia estandarizados como la Guía de Pruebas de Seguridad Web de OWASP (WSTG) y el estándar NIST SP 800-115, que definen ciclos de vida de evaluación desde el descubrimiento hasta el reporte (NIST, 2008; OWASP, 2020).

En la práctica, se debate la eficacia entre pruebas manuales y automatizadas. Estudios recientes indican que, si bien las pruebas automatizadas (DAST) son eficientes en costos y tiempos para detectar fallos técnicos simples, carecen del contexto necesario para identificar vulnerabilidades de lógica de negocio complejas como BOLA (Alavi et al., 2018; Shah & Iqbal, 2020). Por ello, la tendencia actual hacia el pentesting híbrido combina la velocidad de la automatización con la inteligencia humana para validar la lógica y encadenar vulnerabilidades, maximizando la detección de riesgos reales (Qualysec, 2025).

En cuanto a herramientas, Burp Suite Professional y OWASP ZAP se consolidan como los estándares industriales para la interceptación y manipulación de tráfico HTTP, siendo Burp Suite preferido en entornos empresariales por sus capacidades avanzadas de análisis manual, mientras que ZAP destaca en la integración automatizada en pipelines de CI/CD (Aikido Security, 2024).

2.2.4. Laboratorios Virtuales: Infraestructura para Entornos Controlados

La validación de estas metodologías requiere entornos seguros y aislados, conocidos como Cyber Ranges. Tradicionalmente basados en máquinas virtuales (VirtualBox/VMware), la tendencia se ha desplazado hacia la contenerización con Docker debido a su eficiencia en recursos y rapidez de despliegue (LabEx, 2023).

Sin embargo, para simular escenarios de ataque completos que incluyan sistemas operativos de atacante (como Kali Linux) y víctimas, se emplean configuraciones híbridas. El uso de redes "Solo-Anfitrión" (Host-Only) en herramientas como VirtualBox es fundamental para garantizar que el tráfico malicioso generado durante las pruebas de penetración quede estrictamente contenido dentro del laboratorio, evitando fugas hacia redes de producción o Internet pública (Stack Exchange, 2014).

2.2.5. Pentesting de APIs REST en entorno controlado con OWASP (concepto central)

El pentesting de APIs REST en la actualidad se fundamenta en la capacidad de replicar vectores de ataque complejos, especialmente aquellos que residen en la lógica de negocio y no solo en fallos de configuración. La efectividad de la auditoría en un entorno de laboratorio se maximiza mediante la adopción de una estrategia de prueba de seguridad híbrida, que supera las limitaciones de las pruebas automatizadas para identificar fallos de lógica de negocio (Shah & Iqbal, 2020).

Esta aproximación integra dos pilares metodológicos. Por un lado, el Análisis Dinámico de Aplicaciones (DAST) Automatizado, representado por herramientas como OWASP ZAP, se utiliza para el descubrimiento rápido de endpoints y la detección de vulnerabilidades técnicas simples. Por otro lado, las Pruebas Manuales a Nivel de Proxy, utilizando herramientas como Burp Suite Professional, son cruciales para la identificación de fallos de autorización y lógica, como BOLA ($API1:2023$) y BOPLA ($API3:2023$) (OWASP API Security Project Team, 2023). La arquitectura del laboratorio virtual permite simular el flujo completo del ataque de manera segura y reproducible, satisfaciendo los requisitos metodológicos del estándar NIST SP 800-115 para entornos de penetration testing (NIST, 2008).

CAPÍTULO 3:

3. Desarrollo

El presente proyecto tiene como propósito fundamental, el establecimiento de un laboratorio de ciberseguridad controlado, aislado y seguro, diseñado específicamente para el análisis exhaustivo de vulnerabilidades en interfaces de programación de aplicaciones (APIs) bajo arquitectura REST. Para materializar este entorno, se ha seleccionado la aplicación deliberadamente vulnerable VAmPI como núcleo de pruebas, permitiendo la simulación fidedigna de brechas de seguridad.

La metodología de análisis propuesta adopta un enfoque híbrido que integra herramientas de inspección manual, tales como Burp Suite, en conjunción con escáneres automatizados de última generación como OWASP ZAP. Este esquema de trabajo tiene como finalidad la identificación, explotación y posterior clasificación de riesgos críticos incluyendo fallos de autenticación, autorización rota, exposición de datos sensibles y configuraciones defectuosas alineándose estrictamente con los estándares del OWASP API Security Top 10. A partir de la inteligencia obtenida en esta fase experimental, se derivarán estrategias de mitigación concretas y efectivas.

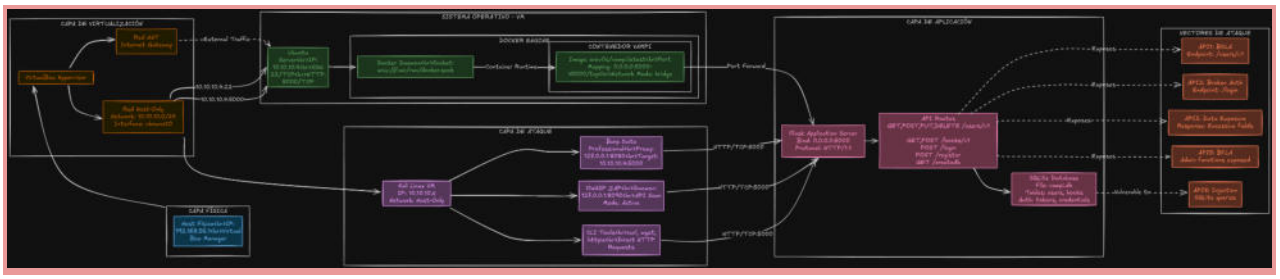
3.1. Levantamiento de entorno seguro

Para el despliegue de la infraestructura virtual, se optó por el hipervisor de código abierto Oracle VM VirtualBox versión 7.2.2 (build r170484). Esta elección se fundamenta en su estabilidad comprobada, por su compatibilidad.

Todo el entorno de laboratorio se desplegará sobre un sistema anfitrión (Host) ejecutando Windows 11 (versión 25H2). Esta base operativa garantiza la compatibilidad con las últimas actualizaciones de los controladores del hipervisor y proporciona un entorno robusto para la orquestación de las máquinas virtuales. La arquitectura lógica planteada contempla la interconexión de tres máquinas virtuales (VMs) segregadas, configuradas para operar dentro de una red privada que emula un entorno corporativo real, minimizando el riesgo de filtración de tráfico malicioso hacia redes externas.

Figuras 1.

Diagrama de arquitectura de aplicación



Nota. El laboratorio se basa en dos máquinas virtuales aisladas en VirtualBox para simular un entorno seguro de análisis. El servidor Ubuntu ejecuta la API vulnerable VAmPI mediante Docker, mientras Kali Linux opera como estación atacante.

3.2. Configuración de Red.

La topología de red fue diseñada para garantizar el aislamiento del entorno de pruebas sin sacrificar la capacidad de gestión se puede ver el resumen dentro de la (Tabla 1). Utilizando el modo "Experto" dentro de las preferencias de red de VirtualBox, se implementaron dos segmentos de red distintos que operan de manera simultánea en las máquinas virtuales:

Esta red se asignó al Adaptador 1 y tiene como función principal permitir la administración de las máquinas virtuales desde el equipo anfitrión. Se definió el segmento lógico 192.168.56.0/29 (máscara de subred 255.255.255.248), creando un dominio de difusión reducido para un control granular del tráfico.

Para la asignación de direcciones, se configuró un servidor DHCP con dirección base 192.168.56.2, estableciendo un rango dinámico de asignación (pool) que abarca desde la IP 192.168.56.3 hasta la 192.168.56.6 (limitado por la máscara /29). Esta configuración asegura que las máquinas obtengan conectividad inmediata con el anfitrión para tareas de mantenimiento y despliegue inicial.

Configurada en el Adaptador 2, esta red opera en el segmento 10.10.10.0/29. Su propósito es facilitar la comunicación interna entre las máquinas virtuales (atacante, víctima y cliente) simulando una intranet, al tiempo que provee salida a Internet a través de una traducción de direcciones (NAT) para la descarga de paquetes y actualizaciones críticas .

Tabla 1.

Matriz de Direccionamiento IP

Máquina	Red Host-Only (192.168.56.0/29)	Red NAT (10.10.10.0/29)	Comentarios
Gateway (Host)	192.168.56.1	10.10.10.1	Puerta de enlace y gestión del hipervisor.
Servidor	192.168.56.4	10.10.10.4	Servidor Ubuntu alojando el contenedor VAmPI y Docker.
Atacante (Kali)	192.168.56.X (DHCP)	10.10.10.X (DHCP)	Estación de pentesting, escaneo y explotación.
Cliente	192.168.56.X (DHCP)	10.10.10.X (DHCP)	Simulación de tráfico de usuario legítimo.rvidor

Nota. La tabla presenta un resumen de las direcciones ip configuradas por DHCP.

3.3. Creacion de Servidor Vulnerable

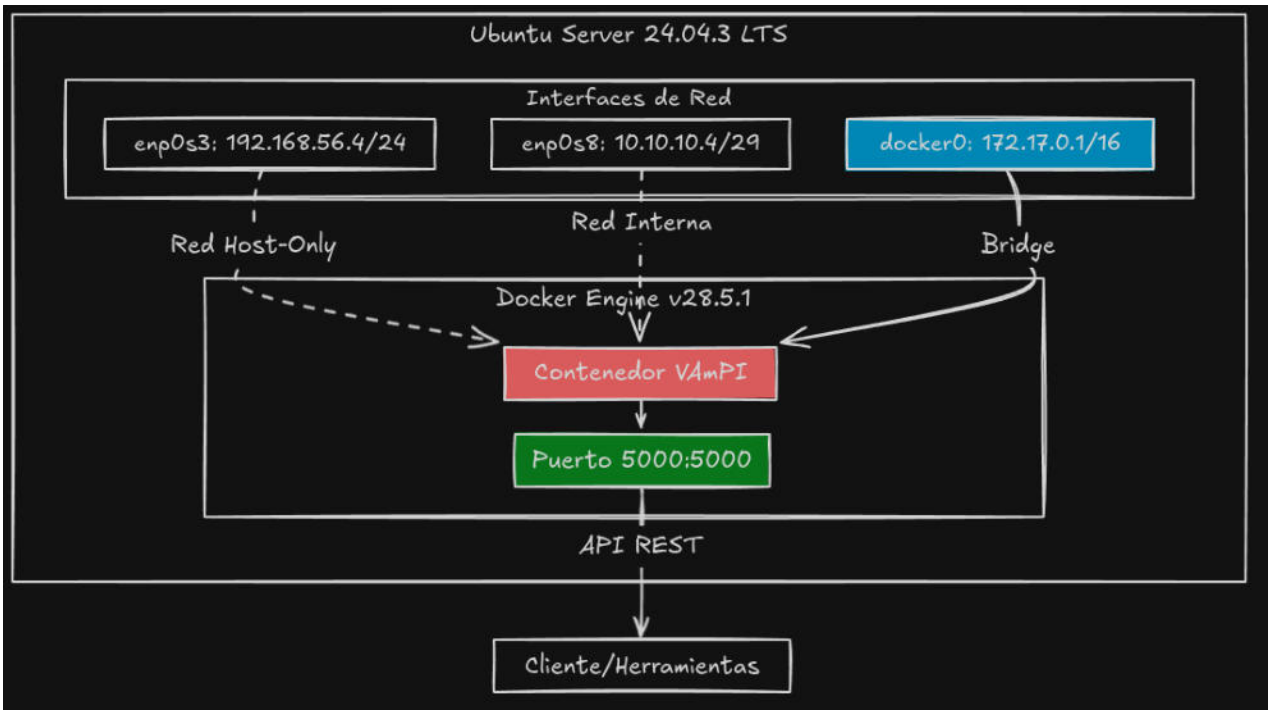
El servidor víctima, denominado "Server", se estructuró sobre una distribución Ubuntu Server 24.04.3 LTS, seleccionada por su soporte a largo plazo (LTS). Dentro de este sistema, se instaló Docker Engine v28.5.1 como motor de contenerización para desplegar la aplicación objetivo.

La configuración de red interna del servidor incluye las interfaces físicas virtuales enp0s3 (conectada a la red de gestión 192.168.56.x) y enp0s8 (conectada a la red de servicio 10.10.10.x). Adicionalmente, se generó la interfaz puente docker0 (172.17.0.1/16) para la comunicación de los contenedores. Se procedió al despliegue de la imagen erev0s/vampi:latest, exponiendo el servicio en el puerto 0.0.0.0:5000. Esta aplicación implementa una API basada en el framework Flask, la cual contiene vulnerabilidades hardcodeadas diseñadas para prácticas de explotación.

Se estableció al servidor vulnerable basado en Ubuntu Server 24.04.3 LTS con Docker Engine v28.5.1 como motor de contenedorización. La topología de red incluye interfaces enp0s3 (192.168.56.4/24) y enp0s8 (10.10.10.4/29) para acceso externo, junto con el bridge docker0 (172.17.0.1/16) para la comunicación con contenedores. Se desplegó la imagen erev0s/vampi:latest, exponiendo el servicio en 0.0.0.0:5000, el cual implementa una API vulnerable basada en Flask que incorpora múltiples vectores de ataque alineados con el OWASP API Security Top 10.

Figuras 2.

Diagrama de arquitectura de virtualización



Nota. Se representa servidor Ubuntu con dos interfaces virtuales (Host-Only e Interna) y un puente docker 0 para redes internas. Docker levanta en él un contenedor VAmPI expuesto en el puerto 5000.

3.4. Creación de Ambiente de atacante

Para la estación de trabajo destinada a las actividades de pentesting se desplegó Kali Linux versión 2025.3 (build del 23 de septiembre de 2025), utilizando la imagen OVA oficial preconfigurada para Oracle VM VirtualBox, lo que permitió una puesta en marcha rápida y estandarizada del entorno. Al tratarse de una appliance virtual lista para su uso, las tareas iniciales se limitaron a la actualización del sistema operativo y del conjunto de herramientas incluidas, ejecutando los comandos apt update -y && apt upgrade -y, con el objetivo de garantizar la corrección de vulnerabilidades conocidas y la disponibilidad de versiones estables y actualizadas.

La configuración de red se realizó a nivel del hipervisor con el fin de controlar el aislamiento y la conectividad del laboratorio. Se habilitaron dos adaptadores de red virtuales: el primero, identificado como enp0s3, configurado en modo Host-Only sobre el segmento 192.168.56.0/24, permitió la comunicación directa entre la máquina atacante y el servidor objetivo sin exposición a redes externas; el segundo adaptador, enp0s8, se configuró en modo Red Interna dentro del segmento 10.10.10.0/29, proporcionando un entorno completamente aislado para pruebas controladas. Esta arquitectura de red permitió simular escenarios reales de ataque manteniendo un alto grado de control y seguridad sobre el entorno experimental.

3.5. Validación de funcionalidad de Endpoints

Una vez finalizado el despliegue de la infraestructura, se ejecutó una fase de validación de servicios para confirmar la operatividad del laboratorio. Mediante técnicas de escaneo de puertos dirigidas al servidor, se identificó el puerto TCP 5000 como el punto de entrada del servicio API REST.

El presente compendio describe de manera estructurada la taxonomía operativa de la Interfaz de Programación de Aplicaciones (API) VAmPI (Vulnerable API), a partir del análisis directo de su Especificación OpenAPI versión 3 (OAS3), la cual constituye la fuente autoritativa para la definición de sus capacidades expuestas. A partir de dicha especificación se identificaron y normalizaron los elementos funcionales de la API, permitiendo una comprensión precisa de su superficie de ataque y de los mecanismos de interacción disponibles.

La tabla subsecuente actúa como un índice programático de referencia, en el cual se catalogan de forma sistemática todos los endpoints accesibles definidos en la especificación. Para cada endpoint se detalla el verbo HTTP asociado (método), la Ruta Uniforme de Recursos (path) correspondiente y una descripción sintética de la funcionalidad lógica que habilita, facilitando así el análisis técnico, la planificación de pruebas de seguridad y la trazabilidad entre los recursos expuestos y sus operaciones asociadas.

Tabla 2.

Matriz de Rutas de Swagger

Método	Ruta (Path)	Descripción
GET	/	Muestra la página de inicio (home) de VAmPI.
GET	/createdb	Crea y puebla la base de datos con datos de prueba (dummy data).
GET	/me	Recupera/muestra al usuario actualmente conectado.
GET	/users/v1	Recupera una lista de todos los usuarios con información básica.
GET	/users/v1/_debug	Recupera todos los detalles para todos los usuarios (Ruta de depuración/Debug).
POST	/users/v1/register	Registra un nuevo usuario.
POST	/users/v1/login	Inicia sesión en VAmPI y genera un token.
GET	/users/v1/{username}	Recupera la información de un usuario específico por su nombre de usuario.
DELETE	/users/v1/{username}	Elimina un usuario por su nombre de usuario (solo permitido para administradores).
PUT	/users/v1/{username}/email	Actualiza la dirección de correo electrónico de un usuario.
PUT	/users/v1/{username}/password	Actualiza la contraseña de un usuario.
GET	/books/v1	Recupera todos los libros.
POST	/books/v1	Agrega un nuevo libro (requiere estar autenticado).
GET	/books/v1/{book_title}	Recupera un libro por su título, incluyendo el secreto asociado al mismo.

Nota. Esta tabla presenta la matriz completa de rutas expuestas por la API VAmPI, mostrando para cada endpoint el método HTTP utilizado.

4. Ataques

4.1.Escaneo

La etapa de reconocimiento constituye el primer eslabón crítico dentro de la cadena de ataque descrita por la Cyber Kill Chain, y tiene como objetivo principal la identificación, enumeración y caracterización de la superficie de ataque expuesta por el sistema objetivo. En esta fase se recopila información técnica relevante que permite comprender el contexto operativo del entorno, sirviendo como base para las etapas posteriores de análisis y explotación.

Para la presente investigación se llevó a cabo un proceso de verificación de disponibilidad del host y descubrimiento de servicios activos sobre el servidor víctima. Conforme a la topología de red definida previamente para el laboratorio, el sistema objetivo se encontraba ubicado en la dirección IP 10.10.10.4, dentro del segmento de red aislado 10.10.10.0/29, diseñado específicamente para la ejecución de pruebas de penetración en un entorno controlado.

El procedimiento fue instrumentado mediante la herramienta Nmap (Network Mapper), reconocida como un estándar industrial en tareas de auditoría y evaluación de seguridad de redes. Su uso permitió la manipulación directa de paquetes IP en bajo nivel para determinar con alta precisión el estado de los puertos lógicos, así como la identificación de servicios expuestos, proporcionando una visión inicial detallada de la superficie de ataque del objetivo.

4.1.1. Comando utilizado

Ejecución del Escaneo de Puertos TCP Se ejecutó un barrido de puertos utilizando el protocolo de control de transmisión (TCP). El comando seleccionado para esta fase inicial fue un escaneo por defecto, el cual verifica los 1.000 puertos más comunes definidos en la base de datos de servicios de Nmap.

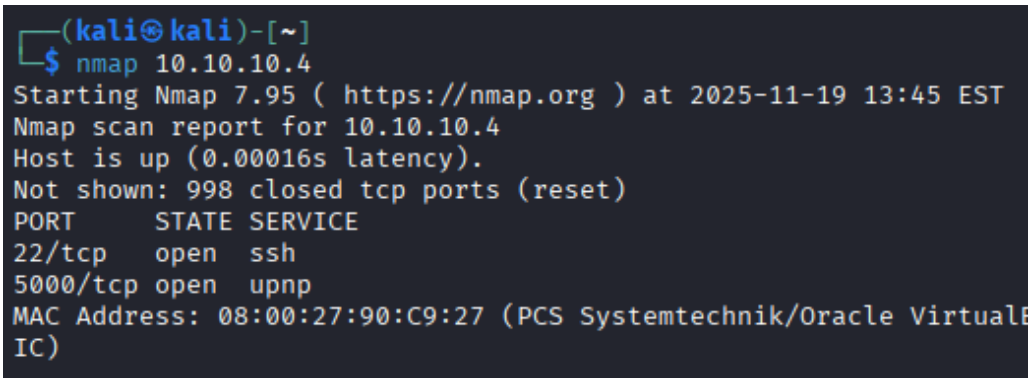
Comando utilizado: nmap 10.10.10.4

4.1.2. Pasos

Desde una perspectiva técnica, al no especificar banderas adicionales (como -sS para SYN Stealth Scan), la herramienta ejecutó un escaneo de tipo TCP Connect(). En este proceso, la máquina atacante completa el saludo de tres vías (Three-Way Handshake) con el objetivo: envía un paquete SYN, espera un SYN-ACK y responde con un ACK. Si la conexión se establece exitosamente, el puerto se reporta como "ABIERTO". Este método garantiza una mayor precisión en la detección de servicios en entornos de laboratorio controlados donde no existen sistemas de detección de intrusos (IDS) activos que bloqueen el tráfico.

Figuras 3.

Escaneo con nmap para puertos abiertos



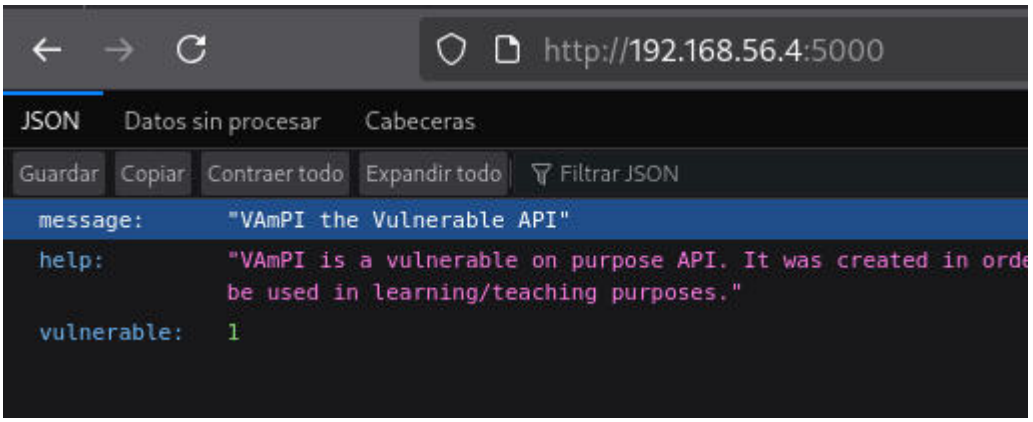
Nota. Escaneo con nmap para ver puertos abiertos sudo nmap 10.10.10.4 para ver que tiene abierto los puertos 22 y 5000.

Resultados del Escaneo Como se evidencia en la Figura 3, el análisis reportó la existencia de dos puertos en estado open, lo que delimita los vectores de entrada potenciales:

- Puerto 22/tcp (SSH): Servicio de administración remota segura, típico en servidores Linux.
- Puerto 5000/tcp (UPnP/HTTP): Un puerto no privilegiado (>1024) que, en el contexto de aplicaciones modernas, es frecuentemente utilizado por frameworks de desarrollo web en Python, específicamente Flask o servidores de aplicaciones WSGI.

Figuras 4.

Ingreso a ip por el puerto abierto del servidor



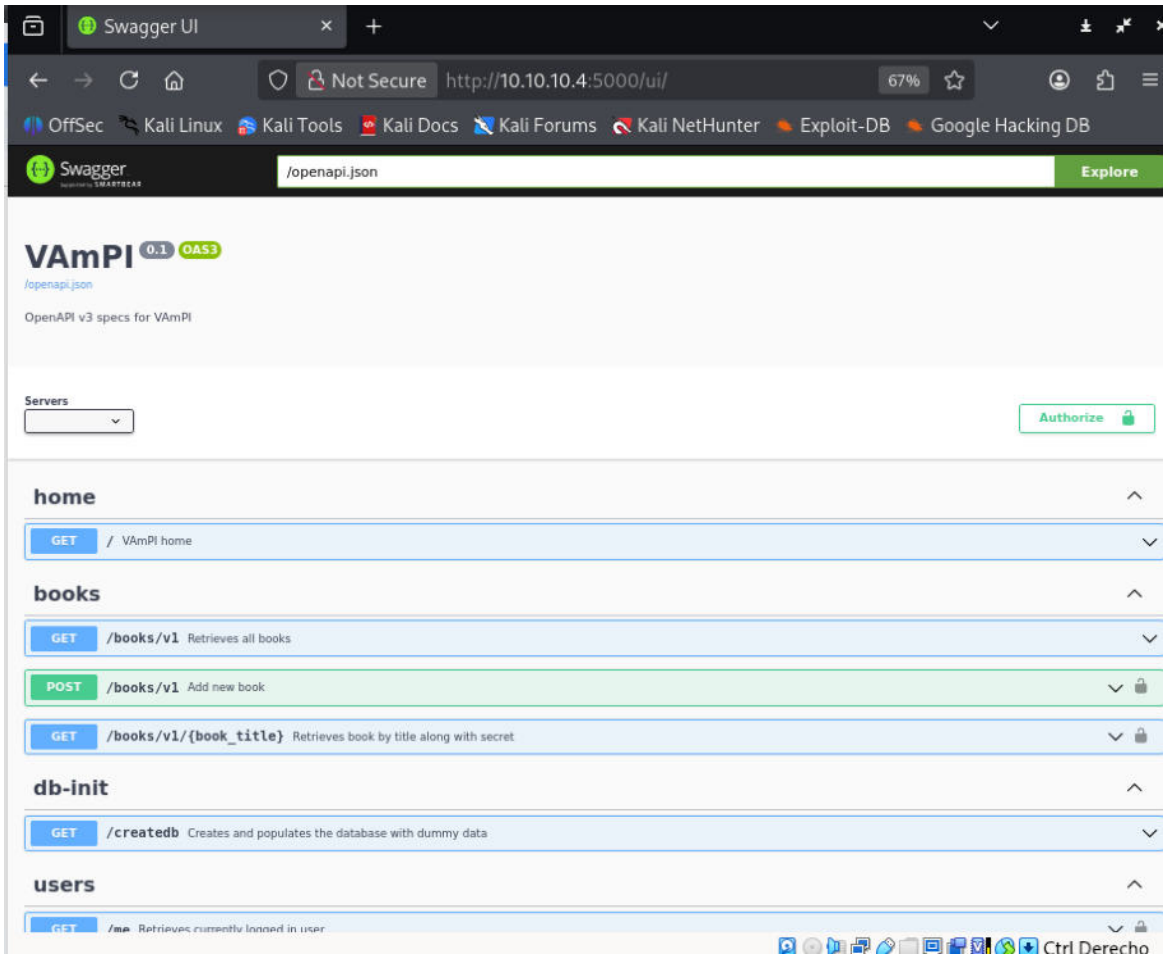
Nota. Esta respuesta indica que el servicio corresponde a VAmPI, una API vulnerable utilizada para pruebas de seguridad y demostraciones de explotación de APIs REST.

Validación de Servicios Web (Banner Grabbing) Una vez identificado el puerto 5000 como un posible servicio web, se procedió a acceder al servicio desde un navegador web para validar su funcionamiento y obtener información preliminar (Banner Grabbing). Se utilizó la URL http://10.10.10.4:5000/.

Una vez identificado el puerto, se procedió a acceder al servicio desde un navegador utilizando la siguiente URL: http://10.10.10.4:5000/ Al realizar la solicitud HTTP, el servidor respondió con el siguiente mensaje:{ "message": "VAmPI the Vulnerable API"}

Figuras 5.

Ingreso a interface de endpoints de swagger propuesto por VAmPI



Nota. Swagger es un sistema que permite documentar los endpoints del sistema web.

Tras identificar la tecnología en uso, se investigó la documentación pública del proyecto VAmPI. Dicha documentación indica que el sistema, cuenta con una interfaz de usuario basada en Swagger UI, la cual permite visualizar de forma interactiva todos los endpoints disponibles en la API. Con base en la información obtenida, se construyó la siguiente URL: "http://10.10.10.4:5000/ui/".

Basándose en la identificación del framework subyacente y las prácticas comunes de desarrollo de APIs, se realizó una búsqueda de rutas estándar de documentación. La investigación de la documentación del proyecto VAmPI sugirió la implementación de la especificación OpenAPI (anteriormente conocida como Swagger) para la descripción de la interfaz.

4.2. Admin Endpoint Abuse

Objetivo: Verificar que la API no proteja operaciones administrativas críticas.

Descripción: El endpoint GET /createdb permite a cualquier usuario, autenticado o no, crear y poblar la base de datos con datos de prueba. Esta operación es administrativa y destructiva, y no posee ningún mecanismo de control de acceso. Como resultado, usuarios no autorizados pueden reiniciar la base de datos, alterando datos de todos los usuarios y recursos del sistema.

Vulnerabilidad Identificada dentro del OWASP Top 10:

- **A01:2023 - Broken Access Control:** El endpoint /createdb permite ejecutar una acción sobre un recurso crítico.
- **A05:2023 - Configuración de Seguridad Incorrecta:** Los endpoints administrativos (crear BD, borrar BD, resetear sistema, configurar parámetros, etc.) deben estar protegidos con controles de autorización de nivel alto (roles admin, permisos específicos).

4.2.1. Riesgos

1. Pérdida de integridad de datos: Cualquier usuario puede sobrescribir la base de datos completa.

2. Exposición de datos: Datos sensibles de otros usuarios pueden ser eliminados o expuestos temporalmente.

3. Denegación de servicio (DoS): La base de datos puede ser recreada repetidamente, causando consumo excesivo de recursos y bloqueo de la API.

4. Falta de trazabilidad: No hay registros de quién ejecuta la operación, dificultando auditorías.

5. Exposición de funcionalidad interna: Operación de administración disponible públicamente, rompiendo principios de mínima superficie y privilegio mínimo.

4.2.2. Ruta:

- Método HTTP: GET
- Endpoint: /createdb
- Descripción funcional: Inicialización y poblamiento de la base de datos
- Requisitos de autenticación: Ninguno
- Requisitos de autorización: Ninguno

El endpoint /createdb corresponde a una operación administrativa crítica, cuya función es la creación y reinicialización de la base de datos del sistema, incluyendo la inserción de datos de prueba. Dicho endpoint se encuentra expuesto públicamente sin ningún mecanismo de autenticación ni control de autorización, lo cual contraviene los principios fundamentales de seguridad por diseño.

Inicialmente, se configuró Burp Suite como proxy de interceptación para analizar el tráfico HTTP generado entre el cliente y la API. Al interceptar solicitudes hacia la ruta base del servicio, se observó que el servidor respondía con un código de estado 200 OK, confirmando la accesibilidad pública del servicio.

Posteriormente, se envió de forma manual una petición HTTP GET directamente al endpoint /createdb. El servidor respondió nuevamente con un código 200 OK, indicando la ejecución exitosa de la operación administrativa. Este comportamiento confirma que cualquier usuario, autenticado o no, puede forzar la recreación completa de la base de datos.

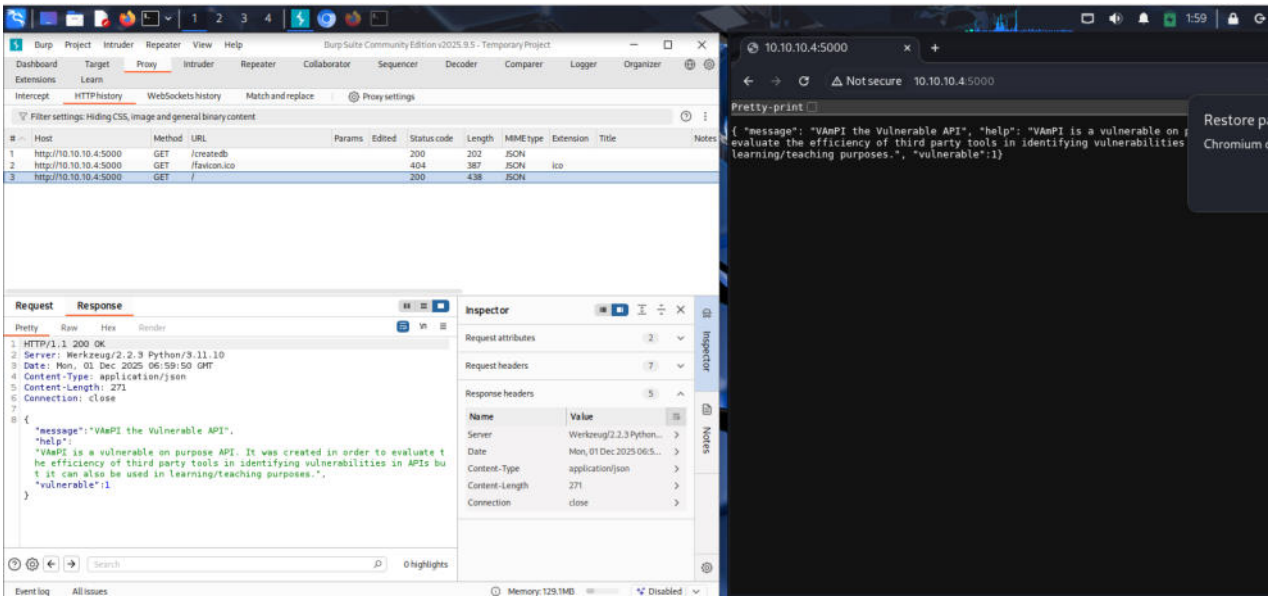
Finalmente, se verificó el impacto mediante una solicitud adicional utilizando la herramienta curl, confirmando que los datos almacenados habían sido modificados y que el estado de la base de datos había sido alterado exitosamente como resultado de la explotación.

4.2.3. Pasos:

Fase 1: Interceptación y Análisis de Tráfico Inicialmente, se estableció un entorno de pruebas mediante la configuración de Burp Suite como proxy de interceptación entre el cliente y el servidor API. Se procedió a navegar por la aplicación para capturar las solicitudes HTTP salientes. Durante este análisis, se identificó la estructura de las peticiones hacia la raíz del servicio. Como se evidencia en la Figura 6, la interceptación manual permitió corroborar que las solicitudes base reciben una respuesta exitosa (Código de Estado 200), lo que establece la disponibilidad del servidor para recibir comandos sin restricciones de red evidentes.

Figuras 6.

Interceptación Manual con Burp Suite

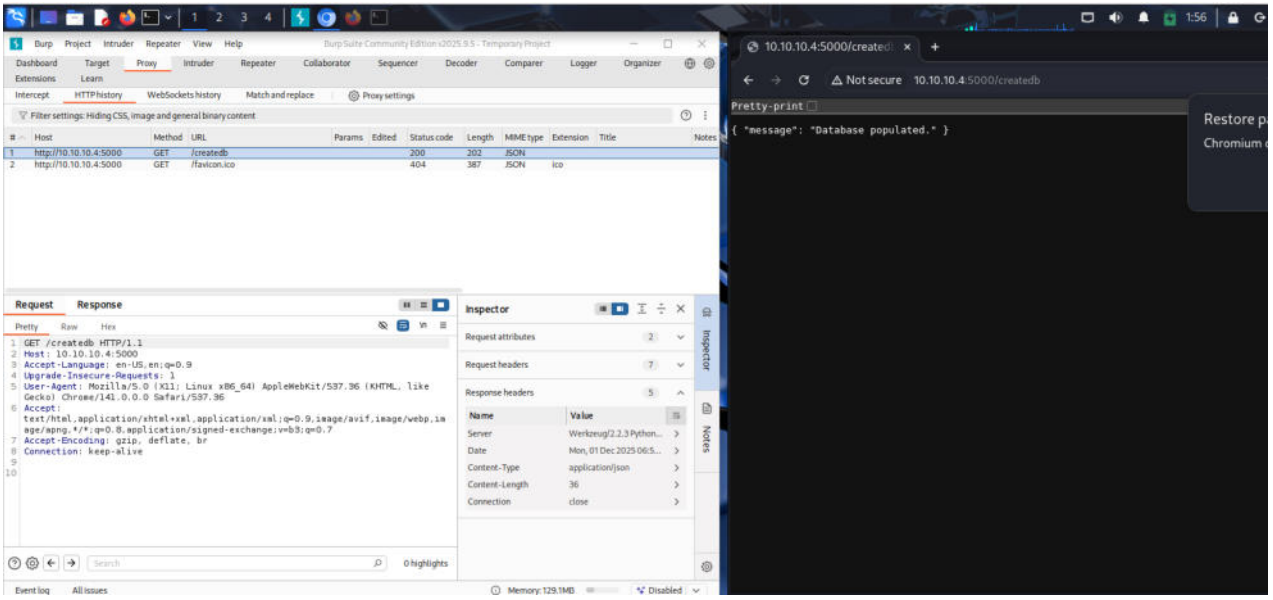


Nota. Podemos interceptar los paquetes usando Brup Suit y en este caso vemos que hay una respuesta 200 al usar la dirección base.

Fase 2: Ejecución de la Solicitud Administrativa Posteriormente, se procedió a la explotación directa del fallo de control de acceso. Se construyó y envió una petición HTTP GET dirigida específicamente al recurso /createdb (10.10.10.4/createdb). A diferencia de lo esperado en un entorno seguro, donde una operación crítica de reinicio de base de datos requeriría autenticación privilegiada, el servidor procesó la solicitud sin demandar credenciales. La Figura 7 ilustra la respuesta del servidor, confirmando la ejecución exitosa de la instrucción de recreación de la base de datos.

Figuras 7.

Envío de la Petición Directamente (Explotación)

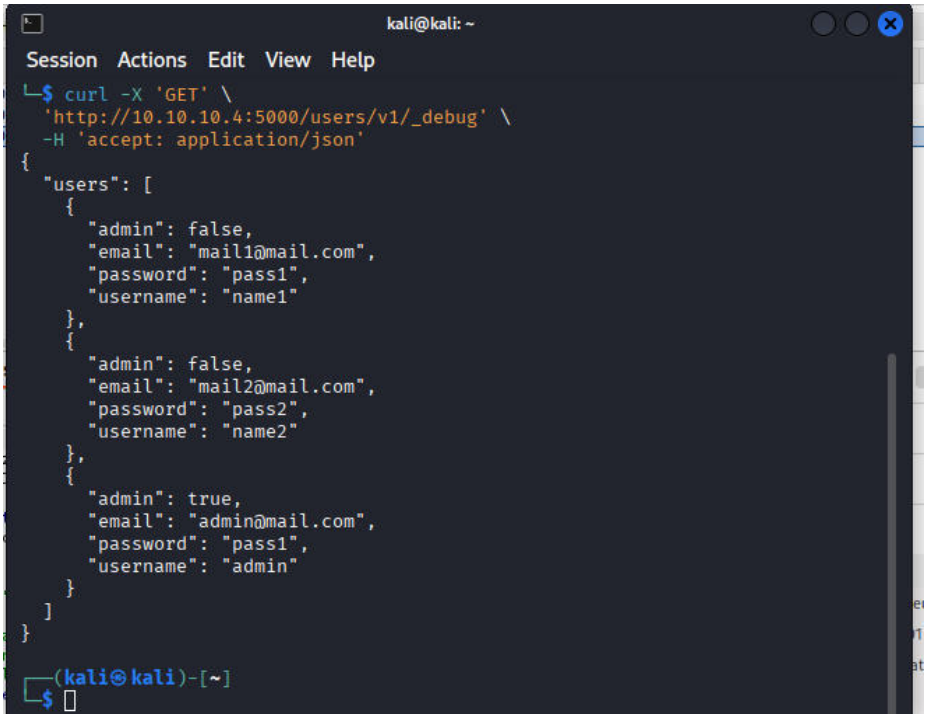


Nota. Al hacer uso del endpoint 10.10.10.4/creatdb se puede ver que se da una respuesta de éxito por lo que podemos editar la base datos desde un endpoint.

Fase 3: Verificación Post-Explotación Finalmente, para cuantificar el impacto de la operación anterior, se realizó una consulta de verificación utilizando la herramienta de línea de comandos curl. Se accedió a un endpoint de lectura de datos para inspeccionar el estado actual del almacenamiento. La respuesta, presentada en formato JSON en la Figura 8, reveló que los registros habían sido modificados a sus valores por defecto o de prueba, confirmando así la pérdida de integridad de los datos previos y la efectividad del ataque de reinicio.

Figuras 8.

Confirmar el Impacto (Post-Explotación)



Nota. Con el comando curl podemos acceder al endpoint y se puede ver el contenido de la base de datos en un formato json.

4.3. Mass User Data Exposure via Debug Endpoint

Objetivo: Verificar que la API no proteja endpoints de debug que exponen información sensible de usuarios.

Descripción: El endpoint GET /users/v1/_debug permite a cualquier usuario, autenticado o no, recuperar todos los detalles de todos los usuarios registrados en el sistema. Este endpoint expone información crítica como emails, contraseñas (posiblemente en texto plano), nombres de usuario y roles administrativos sin ningún mecanismo de control de acceso. Como resultado, los atacantes pueden obtener una base de datos completa de usuarios para realizar ataques dirigidos.

Vulnerabilidad Identificada dentro del OWASP Top 10:

- **A01:2023 – Broken Access Control**: relacionada con la ausencia o implementación deficiente de mecanismos de control de acceso, que posibilita a un atacante realizar acciones o acceder a recursos más allá de los privilegios asignados.
- **A03:2023 – Injection (Information Disclosure)**: asociada a la falta de validación y sanitización de entradas, permitiendo la inyección de payloads maliciosos que pueden derivar en la exposición no autorizada de información sensible.
- **A07:2023 – Identification and Authentication Failures_** vinculada a debilidades en los procesos de autenticación y gestión de identidad, tales como validaciones insuficientes de credenciales o manejo incorrecto de tokens de sesión.
- **A04:2023 – Insecure Design**: que abarca deficiencias inherentes al diseño de la aplicación, incluyendo la ausencia de controles de seguridad desde etapas tempranas, lo que incrementa la probabilidad de explotación sistemática de la API.

4.3.1. Riesgos

1. Exposición masiva de datos personales: Todos los datos de usuarios (emails, contraseñas, información de perfil) son accesibles públicamente.
2. Violación de privacidad y regulaciones: Incumplimiento de GDPR, LGPD, CCPA y otras leyes de protección de datos.
3. Ataques de credential stuffing: Las contraseñas expuestas pueden ser usadas en otros servicios (usuarios comúnmente reutilizan contraseñas).
4. Phishing dirigido: Los emails expuestos permiten campañas de phishing altamente personalizadas.
5. Identificación de objetivos privilegiados: Los campos admin: true/false revelan qué usuarios tienen privilegios elevados.
6. Enumeración de usuarios: Ataques pueden identificar todos los usuarios válidos del sistema para futuros ataques.
7. Falta de trazabilidad: No hay registros de quién accede a esta información sensible, imposibilitando la detección de brechas.

4.3.2. Ruta

- Método HTTP: GET
- Endpoint: /users/v1/_debug
- Descripción funcional: Depuración interna del sistema de usuarios
- Requisitos de autenticación: Ninguno

El endpoint /users/v1/_debug fue identificado como una interfaz de depuración expuesta públicamente, diseñada originalmente para fines internos de desarrollo y diagnóstico. Sin embargo, su despliegue en un entorno accesible permite la divulgación completa de la información de todos los usuarios registrados.

Durante la fase de pruebas, se interceptó el tráfico HTTP utilizando Burp Suite, capturando una solicitud dirigida al endpoint de depuración. El servidor respondió con un objeto JSON que contenía información detallada de todos los usuarios, incluyendo identificadores, direcciones de correo electrónico, contraseñas y roles asociados.

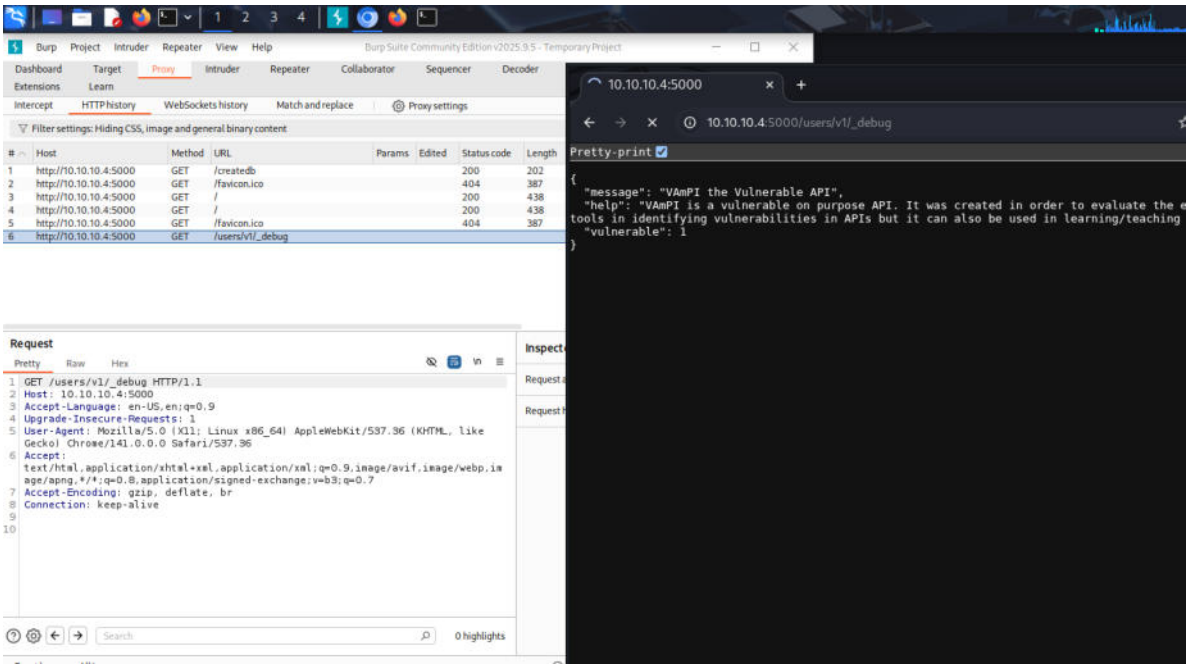
Se confirmó que el acceso a este endpoint no requiere credenciales válidas ni la presentación de un token JWT, lo que facilita la recopilación masiva de datos sensibles. La ausencia de controles de acceso convierte este endpoint en un vector de ataque crítico para la exfiltración de información.

4.3.3. Pasos

Fase 1: Identificación y Captura de Tráfico Mediante técnicas de fuzzing de directorios o revisión de documentación expuesta, se localizó el recurso /users/v1/_debug. Se procedió a interceptar la comunicación hacia este recurso utilizando el proxy Burp Suite. El objetivo era determinar si el servidor implementaba controles de acceso en rutas que, por su nomenclatura, sugieren funcionalidad de desarrollo interno. La Figura 9 muestra la solicitud HTTP GET capturada, la cual fue enviada sin cabeceras de autorización (Authorization: Bearer <token>).

Figuras 9.

Capturarla en Proxy de "v1/_debug".

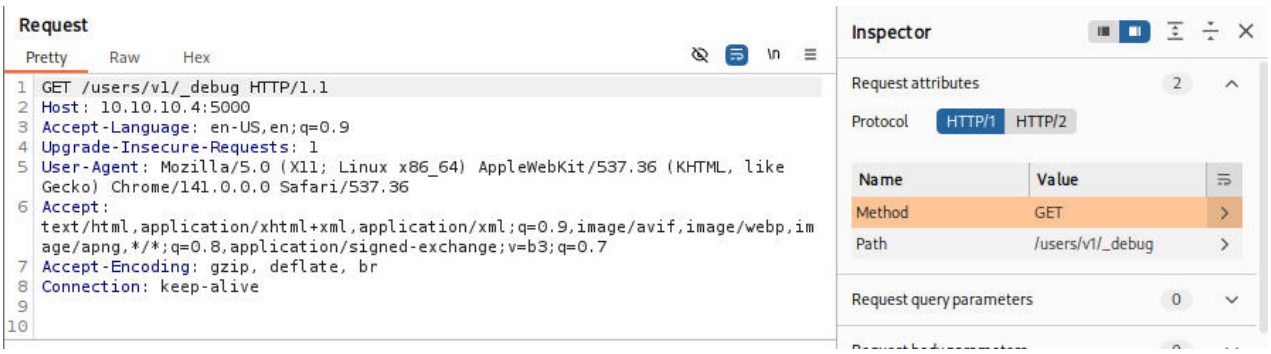


Nota. Con Brup Suit interceptamos el tráfico de "users/v1/_debug"

Fase 2: Verificación de Acceso y Análisis de Datos Tras el envío de la solicitud interceptada, se analizó la respuesta HTTP retornada por el servidor. Como se observa en la Figura 10, el sistema respondió con un código 200 OK y un cuerpo de mensaje en formato JSON. Este objeto contiene un volcado completo de la tabla de usuarios, exponiendo atributos críticos como identificadores únicos, direcciones de correo electrónico, roles de sistema y hash de contraseñas. Esto confirma que el mecanismo de autenticación es inexistente para este endpoint específico.

Figuras 10.

Verificación que no se requiere autenticación para acceso.



Nota. Con Brup Suit vemos que el servidor retorna información al acceder

4.4. Authentication Bypass and Credential Exploitation

Objetivo: Verificar que el mecanismo de autenticación sea robusto y resistente a ataques comunes.

Descripción: El endpoint POST /users/v1/login presenta múltiples vulnerabilidades en el proceso de autenticación que permiten a atacantes realizar credential stuffing, user enumeration, timing attacks y potencial authentication bypass. La implementación actual revela información sensible mediante mensajes de error específicos y no implementa controles básicos de seguridad para autenticación.

Vulnerabilidad Identificada dentro del OWASP Top 10:

- **A07:2023 – Identification and Authentication Failures:** asociada a fallos en los procesos de identificación y autenticación, tales como controles insuficientes sobre la validación de credenciales y la gestión de sesiones, lo que facilita ataques de fuerza bruta o reutilización de credenciales.
- **A02:2023 – Cryptographic Failures:** evidenciada por una configuración potencialmente insegura del uso de JSON Web Tokens (JWT), incluyendo la gestión inadecuada de claves, algoritmos criptográficos débiles o la ausencia de validaciones críticas durante el proceso de verificación del token.
- **A03:2023 – Injection (Credential Injection sin Rate Limiting):** relacionada con la ausencia de mecanismos de limitación de solicitudes, lo que permite la inyección sistemática de credenciales a través de múltiples intentos de autenticación sin restricciones, incrementando la probabilidad de comprometer cuentas válidas.

4.4.1. Riesgos:

1. User Enumeration: El mensaje de error "Password is not correct for the given username" confirma la existencia de usuarios válidos en el sistema.
2. Credential Stuffing: Ausencia de rate limiting permite ataques automatizados de fuerza bruta.
3. Timing Attacks: Diferencia en tiempos de respuesta entre usuario válido/inválido puede revelar información.
4. JWT Token Exploitation: Tokens JWT sin proper validation pueden permitir impersonation.
5. Password Policy Bypass: No hay evidencia de políticas de contraseñas complejas.
6. Session Management Deficiente: No se observa mecanismo de refresh token o invalidación segura.
7. Information Disclosure: Mensajes de error detallados ayudan a los atacantes a refinar sus ataques.

4.4.2. Ruta

- Método HTTP: POST
- Endpoint: /users/v1/login
- Descripción funcional: Autenticación de usuarios

El endpoint encargado del proceso de autenticación presenta múltiples deficiencias relacionadas con la gestión de credenciales y el manejo de errores, permitiendo la ejecución de ataques de enumeración de usuarios y explotación de tokens de sesión.

Se realizaron múltiples solicitudes de autenticación utilizando combinaciones controladas de nombres de usuario y contraseñas. El sistema retornó mensajes de error diferenciados dependiendo de si el usuario existía o no, lo cual permitió confirmar la existencia de cuentas válidas mediante user enumeration.

Una vez obtenidas credenciales válidas, el sistema emitió un token JWT que fue extraído y utilizado en solicitudes posteriores. El token carecía de validaciones adicionales, permitiendo su uso directo para acceder a recursos protegidos, lo que evidencia una implementación deficiente del manejo de sesiones

4.4.3. Pasos

Fase 1: Extracción de Token de Sesión Habiendo confirmado previamente la existencia de usuarios válidos (por ejemplo, el usuario 'admin') mediante técnicas de enumeración basadas en respuestas de error, se procedió a realizar una petición POST al endpoint /users/v1/login. Al proveer las credenciales correctas, el servidor emitió un token JWT (JSON Web Token). Este token fue extraído de la respuesta JSON y almacenado en una variable de entorno local para facilitar su reutilización en pruebas subsiguientes, tal como se detalla en la Figura 11.

Figuras 11.

Extraer el token del usuario admin extraído

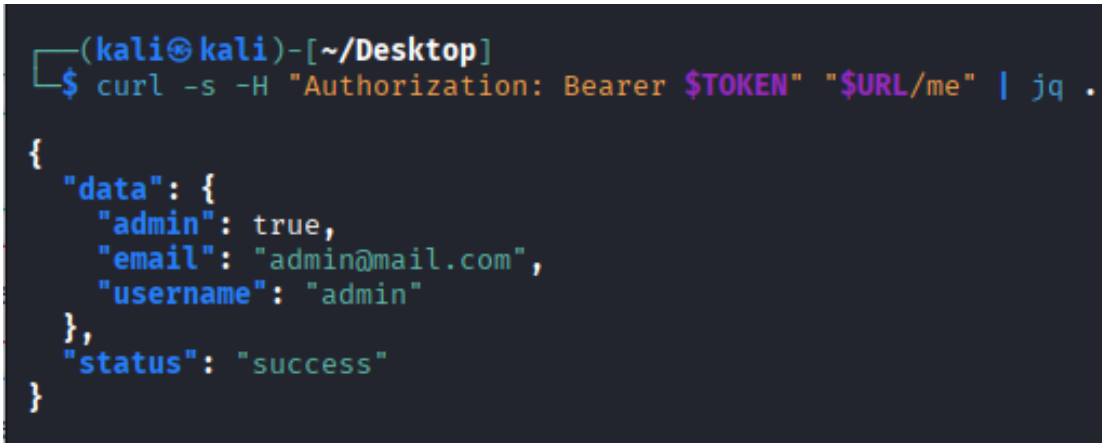
```
┌──(kali㉿kali)-[~/Desktop]
└─$ TOKEN=$(curl -s -X POST http://10.10.10.4:5000/users/v1/login \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"pass1"}' \
  | jq -r '.auth_token')

echo "Token: $TOKEN"
Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3NjQ1NzU0NDksImlhdCI6M
Tc2NDU3NTM4OSwic3ViIjoiYWRtaW4ifQ.ZYRBlk-e8E_5edYRSz-N7BFHag_0NARF26M9sXiVjl8
```

Nota. Se puede extraer el token de login y guardarlo en una variable de entorno TOKEN

Fase 2: Validación de Acceso Autenticado Para corroborar la validez y los privilegios conferidos por el token extraído, se realizó una petición a un recurso protegido de la API, inyectando el token en la cabecera Authorization. La Figura 12 demuestra que el servidor valida correctamente la firma del JWT y permite el acceso al recurso solicitado, confirmando que el atacante posee ahora una sesión activa y válida dentro del sistema.

Figuras 12.

Validar el JWT accediendo



Nota. Usando el token podemos conseguir un login.

4.5. SQL Injection por Validación Deficiente de Entradas en Endpoints de Usuario

Objetivo: Verificar si los endpoints que reciben parámetros suministrados por el usuario implementan una sanitización correcta para prevenir inyección SQL.

Descripción: Los endpoints bajo la ruta /usuarios/v1/{usuario} aceptan parámetros proporcionados por el cliente sin aplicar ningún proceso adecuado de validación o sanitización. Durante las pruebas, se envió un payload simple que añadía una comilla simple al parámetro nombre (/usuarios/v1/nombre'), lo que produjo un error interno de la API.

El sistema devolvió un error SQL generado por el motor de base de datos

Este mensaje confirma que la entrada del usuario se concatena directamente en la consulta SQL sin uso de consultas preparadas ni parámetros seguros. Como resultado, un atacante podría inyectar código SQL malicioso capaz de manipular consultas, extraer información sensible o alterar datos.

Vulnerabilidad Identificada dentro del OWASP Top 10:

- **A03:2023 – Injection:** asociada a la validación y sanitización insuficiente de los datos de entrada, lo que permite la introducción de payloads maliciosos capaces de alterar el comportamiento esperado de la aplicación o acceder a información no autorizada.
- **A05:2023 – Security Misconfiguration:** relacionada con configuraciones inseguras a nivel de aplicación, servicios o infraestructura subyacente, tales como

parámetros por defecto, exposición innecesaria de servicios o manejo incorrecto de cabeceras de seguridad, que amplían la superficie de ataque.

- **A04:2023 – Insecure Design**: que refleja deficiencias estructurales en el diseño de la solución, incluyendo la ausencia de controles de seguridad integrados desde etapas tempranas, así como la falta de mecanismos de defensa en profundidad, lo que facilita la explotación sistemática de las vulnerabilidades identificadas.

4.5.1. Riesgos

- Manipulación de consultas SQL para acceder, modificar o eliminar datos.
- Bypass de autenticación mediante payloads de inyección (por ejemplo, ' OR '1'='1).
- Extracción no autorizada de información de usuarios, contraseñas u otros datos sensibles.
- Compromiso total de la base de datos debido a consultas no parametrizadas.
- Posibilidad de escalación de privilegios dependiendo de la lógica de la aplicación.

4.5.2. Ruta

- Método HTTP: GET
- Endpoint: /usuarios/v1/{usuario}

El endpoint analizado procesa parámetros recibidos directamente desde la URL sin aplicar mecanismos adecuados de validación, normalización o sanitización de entradas. Esta condición expone la lógica de acceso a datos a manipulaciones externas, convirtiendo al endpoint en un vector potencialmente vulnerable a ataques de inyección SQL.

Con el objetivo de verificar dicha hipótesis, se procedió a modificar el parámetro {usuario} mediante la inserción de una comilla simple (') en una solicitud interceptada utilizando la herramienta Burp Suite. Como resultado, el servidor respondió con un error interno que revelaba información específica del motor de base de datos subyacente, lo que evidenció la concatenación directa del valor del parámetro dentro de la consulta SQL y confirmó la presencia de un fallo de inyección.

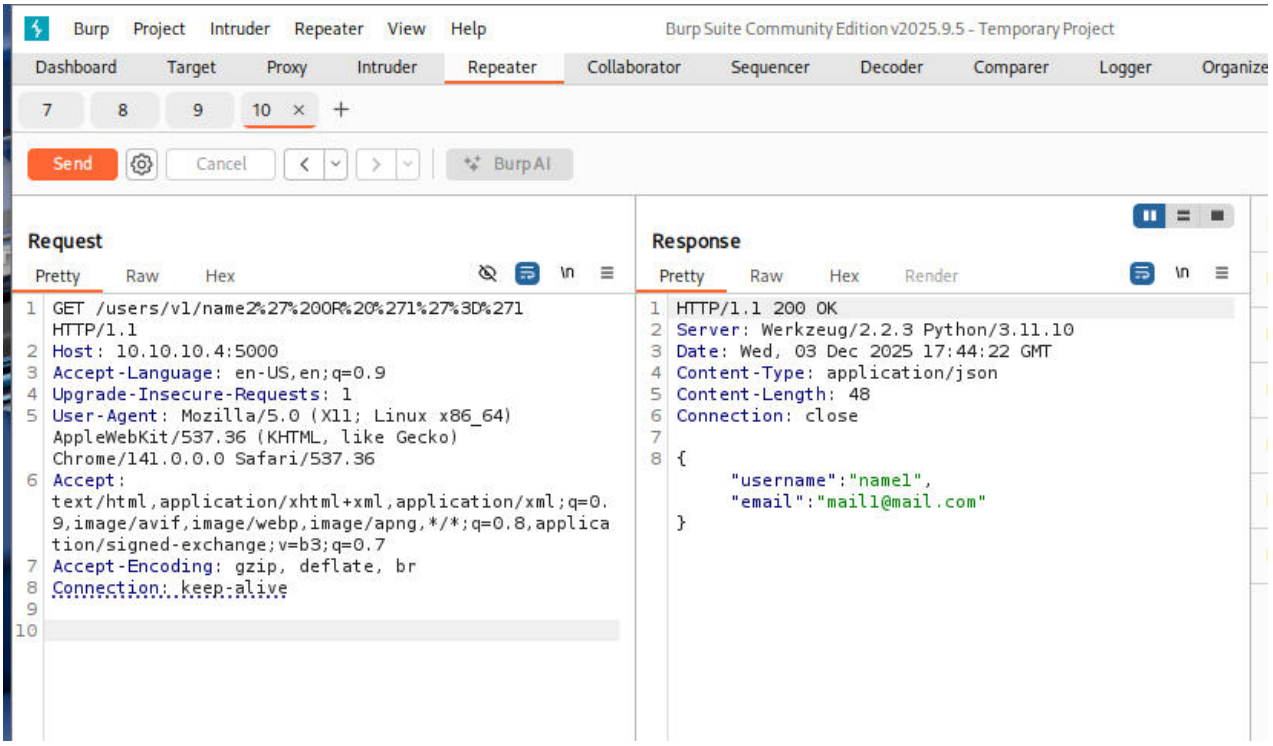
Posteriormente, se empleó la herramienta automatizada sqlmap para facilitar la explotación controlada de la vulnerabilidad identificada. A través de esta herramienta fue posible inferir la estructura de la base de datos, así como enumerar las columnas de la tabla users. Este comportamiento confirma la ausencia de consultas parametrizadas y el uso de prácticas inseguras en la construcción dinámica de sentencias SQL, incrementando significativamente el riesgo de exposición y manipulación de información sensible.

4.5.3. Pasos

Fase 1: Detección Manual mediante Inyección de Payload Se interceptó una petición legítima dirigida al endpoint /usuarios/v1/{usuario} mediante Burp Suite. Con el fin de verificar la existencia de vulnerabilidades de inyección, se modificó el parámetro {usuario} añadiendo una comilla simple (') como carga útil (payload). Este carácter es comúnmente interpretado por los motores de base de datos como un delimitador de cadenas. La Figura 13 muestra la petición modificada y la respuesta del servidor, la cual devolvió un error de sintaxis SQL, evidenciando que la entrada del usuario se concatena directamente en la consulta sin parametrización.

Figuras 13.

El envío de la petición mediante Burp Suite y ejecuté una prueba utilizando una carga útil básica.



Nota. Enviamos una solicitud donde se observa un parámetro del nombre codificado en la URL esto evidencia que el endpoint procesa correctamente la solicitud y expone información del usuario sin requerir medidas adicionales de autenticación o validación..

Fase 2: Explotación Automatizada con SQLMap Una vez confirmada la vulnerabilidad, se procedió a automatizar el ataque utilizando la herramienta sqlmap. Se configuró la herramienta para apuntar al endpoint vulnerable, permitiendo que esta realizara una enumeración exhaustiva de la base de datos. Como se observa en la Figura 14, sqlmap logró identificar exitosamente el motor de base de datos, así como los nombres y tipos de columnas de la tabla users, demostrando la capacidad de exfiltración total de la estructura y datos del sistema.

Figuras 14.

Lanzar una consulta automatizada con sqlmap

Nota. Se procedió a lanzar una consulta automatizada con sqlmap con el fin de identificar los nombres y tipos de columnas presentes en la tabla users.

4.6. Secuestro de Cuentas por Autorización Defectuosa en Endpoints de Propiedad

Objetivo: Determinar si un usuario autenticado puede modificar información perteneciente a otros usuarios.

Descripción: Se identificó que mediante una solicitud autenticada a :/usuarios/v1/{nombreUsuario}/contraseña es posible cambiar la contraseña de cualquier usuario, incluso si el token pertenece a un tercero sin permisos especiales.

La API responde con 204 No Content, pero una revisión posterior del endpoint de debug confirma que la contraseña del usuario objetivo fue modificada exitosamente.

Esto constituye una vulnerabilidad crítica de Broken Object Property Authorization, permitiendo a un atacante tomar control total de cualquier cuenta registrada.

Vulnerabilidades OWASP Top 10:

- **A01:2023 – Broken Access Control**, correspondiente a fallos en la implementación de controles de autorización, que permiten a usuarios autenticados o no autenticados acceder a recursos o ejecutar operaciones fuera de su nivel de privilegio asignado.
- **A07:2023 – Identification and Authentication Failures**, relacionada con debilidades en los procesos de identificación, autenticación y gestión de sesiones, tales como validaciones insuficientes de credenciales o controles inadecuados sobre el ciclo de vida de los tokens de acceso.
- **A04:2023 – Insecure Design**, que engloba deficiencias inherentes al diseño de la arquitectura de la API, incluyendo la ausencia de controles de seguridad desde la fase de concepción y la falta de mecanismos de mitigación frente a escenarios de abuso previsibles.

4.6.1. Riesgos

- Secuestro total de cuentas de cualquier usuario.
- Acceso no autorizado a datos personales o recursos privados del objetivo.
- Compromiso de cuentas administrativas o privilegiadas.
- Impacto crítico en la integridad y confidencialidad del sistema.

4.6.2. Ruta

- Método HTTP: PUT / PATCH
- Endpoint: /usuarios/v1/{nombreUsuario}/contraseña

Este endpoint permite la modificación de credenciales de usuario basándose únicamente en la validez del token JWT, sin verificar la propiedad del recurso solicitado.

Se obtuvo un token JWT válido correspondiente a un usuario estándar mediante el endpoint de autenticación. Posteriormente, dicho token fue reutilizado para enviar una solicitud de cambio de contraseña apuntando a otro usuario.

El servidor respondió con un código 204 No Content, indicando la aceptación de la solicitud. La explotación fue confirmada iniciando sesión con las nuevas credenciales del usuario afectado, demostrando el control total de la cuenta y evidenciando una vulnerabilidad crítica de Broken Object Level Authorization (BOLA).

4.6.3. Pasos

Fase 1: Obtención de Credenciales de Atacante El ataque comenzó con la autenticación legítima de un usuario con bajos privilegios (denominado "name1") en el sistema. Mediante el endpoint de login, se obtuvo el token JWT correspondiente a este usuario. Este paso es fundamental para demostrar que el ataque puede ser perpetrado por cualquier usuario validado en la plataforma, sin necesidad de permisos administrativos. La Figura 15 ilustra la obtención y verificación de este token.

Figuras 15.

Token del usuario name1

Nota. El atacante obtiene un token JWT autenticado para el usuario "name1" mediante una petición POST al endpoint de login se puede ver el token después con echo.

Fase 2: Ejecución del Ataque de Cambio de Contraseña (IDOR) Utilizando el token del usuario "name1", se construyó una petición HTTP (PUT o PATCH) dirigida al endpoint /usuarios/v1/{víctima}/contraseña. Se modificó la URL para apuntar al nombre de usuario de la víctima, mientras se mantenía la cabecera de autorización del atacante. La API, al no validar que el usuario del token coincida con el usuario de la URL, procesó la solicitud exitosamente, devolviendo un código 204 No Content. Esto se detalla en la Figura 16.

Figuras 16.

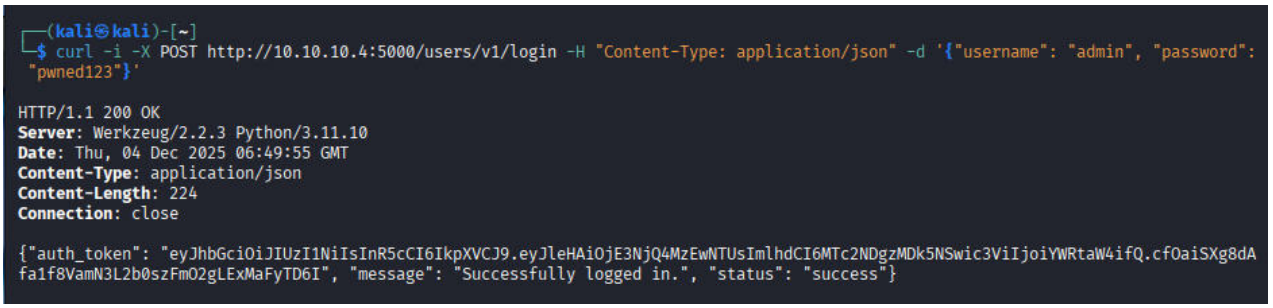
Atacante aprovecha un IDOR

Nota. El atacante aprovecha un IDOR para modificar la contraseña de name1 con credenciales de name1, obteniendo respuesta 204 que evidencia falta de control de autorización.

Fase 3: Verificación del Compromiso de la Cuenta Para confirmar el éxito del secuestro de cuenta, se intentó iniciar sesión utilizando el nombre de usuario de la víctima y la nueva contraseña establecida por el atacante. La Figura 17 muestra que el sistema generó un nuevo token JWT para la víctima, confirmando que el atacante ha obtenido control total sobre la cuenta objetivo.

Figuras 17.

Verifica el éxito del ataque



Nota. El atacante verifica el éxito del ataque iniciando sesión como name1 con la nueva contraseña, obteniendo un token JWT válido que confirma el control total de la cuenta.

CAPÍTULO 4:

5. Análisis de resultados

5.1. Pruebas de Concepto

La fase experimental se ejecutó tras la validación de integridad de la infraestructura virtualizada. El entorno de pruebas, compuesto por un sistema anfitrión (Windows 11), un servidor víctima (Ubuntu Server 24.04 ejecutando contenedores Docker) y una estación de ataque (Kali Linux 2025.3), operó bajo una estricta segmentación de red (Subred 10.10.10.0/29). Este aislamiento garantiza la contención del tráfico malicioso, asegurando que los vectores de ataque desplegados no afectarán a sistemas externos.

5.1.1. Abuso de Endpoints Administrativos (Admin Endpoint Abuse)

El análisis del endpoint GET /createdb reveló una violación flagrante del Principio de Mínimo Privilegio. Esta ruta, diseñada para funciones de mantenimiento y restauración del esquema de base de datos, se encuentra expuesta sin capas de middleware de autenticación.

- Observación Técnica: La ejecución de una petición HTTP GET simple por un agente no autenticado resultó en la reinicialización completa de la base de datos, eliminando la persistencia de los registros previos.
- Análisis de Impacto: Este fallo compromete directamente la Disponibilidad y la Integridad del sistema (dos pilares de la tríada CIA). Desde una perspectiva arquitectónica, representa una falta de separación entre las interfaces públicas y las administrativas.
- Clasificación: Se confirma la presencia de Broken Access Control (A01:2023) y una Security Misconfiguration (A05:2023) crítica.

5.1.2. Exposición Masiva de Datos (Mass User Data Exposure)

La inspección del endpoint GET /users/v1/_debug evidenció una falla en la gestión del ciclo de vida del desarrollo de software (SDLC), donde funcionalidades de depuración (debugging) no fueron deshabilitadas en el despliegue final.

- Observación Técnica: El sistema serializa y retorna la totalidad de los objetos de la clase User almacenados en la base de datos. La respuesta JSON incluye atributos sensibles como direcciones de correo electrónico (email), indicadores de rol (admin: true/false) y, críticamente, contraseñas hash.
- Análisis de Impacto: Esta vulnerabilidad facilita la fase de reconocimiento para un atacante. La obtención de correos y roles permite ataques de Spear Phishing o Credential Stuffing altamente dirigidos. Además, viola normativas de privacidad de datos (como GDPR o CCPA) al exponer Información de Identificación Personal (PII).
- Clasificación: Information Disclosure (A03:2023) e Insecure Design (A04:2023).

5.1.3. Inyección SQL (SQL Injection)

La evaluación de la seguridad en la capa de datos se realizó sobre el endpoint /users/v1/{username}. Se detectó que el mecanismo de consulta a la base de datos carece de sanitización de entrada y no implementa consultas parametrizadas (Prepared Statements).

- Observación Técnica: La inyección de un carácter de comilla simple (') en el parámetro de la URL provocó una excepción no controlada en el motor de base de datos, retornando un error HTTP 500 con detalles de la sintaxis SQL interna (Inyección SQL basada en error). Esto confirma que los datos del usuario se concatenan directamente en la cadena de consulta, rompiendo la separación entre el plano de control y el plano de datos.
- Análisis de Impacto: Un atacante podría escalar esta vulnerabilidad para exfiltrar la base de datos completa, modificar registros arbitrarios o, en escenarios de configuración permisiva, lograr ejecución remota de código (RCE) en el servidor de base de datos.
- Clasificación: Injection (A03:2023).

5.1.4. Secuestro de Cuentas mediante BOLA (Account Takeover)

Se sometió a prueba la lógica de autorización del endpoint PUT /users/v1/{victim_user}/password. Se descubrió que el sistema valida la autenticidad del token (quién es el usuario), pero falla en verificar la autorización sobre el recurso específico (si el usuario es dueño del objeto que intenta modificar).

- Observación Técnica: Utilizando un token JWT válido perteneciente al usuario "A", se envió una petición de cambio de contraseña dirigida al recurso del usuario "B". La API procesó la solicitud exitosamente, retornando un código HTTP 204 (No Content), lo que indica que la operación se completó sin errores.
  Posteriormente, se verificó el acceso a la cuenta del usuario "B" con las nuevas credenciales.
- Análisis de Impacto: Esta es la vulnerabilidad de mayor criticidad en el contexto de APIs REST. Permite el compromiso total horizontal (acceso a cuentas de otros usuarios) y vertical (escalada de privilegios si se ataca a un administrador).
- Clasificación: Broken Object Level Authorization (BOLA/IDOR) y Broken Access Control (A01:2023).

5.1.5. Secuestro de Cuentas (Account Takeover via BOLA)

Se sometió a prueba la lógica de autorización del endpoint PUT /users/v1/{victim_user}/password. Se descubrió que el sistema valida la autenticidad del token (quién es el usuario), pero falla en verificar la autorización sobre el recurso específico (si el usuario es dueño del objeto que intenta modificar).

- Observación Técnica: Utilizando un token JWT válido perteneciente al usuario "A", se envió una petición de cambio de contraseña dirigida al recurso del usuario "B". La API procesó la solicitud exitosamente, retornando un código HTTP 204 (No Content), lo que indica que la operación se completó sin errores.
  Posteriormente, se verificó el acceso a la cuenta del usuario "B" con las nuevas credenciales.
- Análisis de Impacto: Esta es la vulnerabilidad de mayor criticidad en el contexto de APIs REST. Permite el compromiso total horizontal (acceso a cuentas de otros usuarios) y vertical (escalada de privilegios si se ataca a un administrador).
- Clasificación: Broken Object Level Authorization (BOLA/IDOR) y Broken Access Control (A01:2023).

5.2. Resultados

El análisis exhaustivo de la API VAmPI en el entorno controlado ha evidenciado deficiencias estructurales graves en la implementación de seguridad de la arquitectura REST. Los resultados obtenidos demuestran que la ausencia de controles de validación, autorización y autenticación robustos expone al sistema a riesgos críticos que comprometen la confidencialidad, integridad y disponibilidad de los datos.

1. Reconocimiento: El atacante utiliza la Exposición Masiva de Datos (/_debug) para listar todos los nombres de usuario válidos y sus correos electrónicos.

2. Explotación: Con la lista de objetivos confirmada, el atacante selecciona un usuario con rol de administrador (identificado por admin: true).

3. Compromiso: El atacante emplea la vulnerabilidad BOLA en el endpoint de cambio de contraseña para sobrescribir las credenciales del administrador identificado.

4. Impacto Final: El atacante obtiene control total del sistema con privilegios administrativos, demostrando cómo una falla de información de bajo nivel puede escalar hasta un compromiso total de la infraestructura.

Con el objetivo de dotar al análisis de una métrica objetiva y estandarizada para la priorización de los hallazgos de seguridad, se adoptó el Sistema de Puntuación de Vulnerabilidades Comunes (Common Vulnerability Scoring System, CVSS) en su versión 3.1. Este marco metodológico permite cuantificar la severidad técnica de las vulnerabilidades a partir de sus características intrínsecas, denominadas Métricas Base, de forma independiente del contexto temporal o del entorno específico de implementación.

El cálculo del riesgo se fundamentó en la evaluación de dos dimensiones principales: la Explotabilidad, que mide la facilidad y los requisitos técnicos necesarios para llevar a cabo un ataque, y el Impacto, que evalúa las consecuencias potenciales sobre la confidencialidad, integridad y disponibilidad del sistema afectado.

A partir de este enfoque, se presenta a continuación la evaluación vectorial aplicada a los dos hallazgos de mayor criticidad identificados en la API VAmPI, justificando de manera técnica la asignación de los niveles de riesgo reportados en el presente estudio.

5.2.1. Reinicio de Base de Datos no Autorizado

Se identificó una vulnerabilidad crítica asociada a la exposición del endpoint administrativo GET /createdb, el cual permite la reinicialización completa de la base de datos de la aplicación sin requerir mecanismos de autenticación ni autorización. Esta condición constituye una falla grave de configuración de seguridad y se clasifica dentro de la categoría A05:2023 – Security Misconfiguration del estándar OWASP API Security Top 10. Desde una perspectiva cuantitativa, la vulnerabilidad presenta una puntuación CVSS v3.1 estimada de 9.1 (Crítica), con un vector de ataque AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:H. La explotación es posible de forma remota, no requiere interacción del usuario ni privilegios previos, y afecta directamente los recursos del mismo componente vulnerable. Incluso en escenarios donde existan distintos niveles de permisos definidos a nivel de aplicación, la ausencia total de controles de acceso convierte esta funcionalidad en universalmente explotable por cualquier actor externo.

El impacto de esta vulnerabilidad se concentra principalmente en la Integridad y Disponibilidad del sistema. La ejecución no autorizada del reinicio de la base de datos provoca la

pérdida total de la información almacenada, incluyendo registros de usuarios, configuraciones operativas y datos transaccionales, comprometiendo de manera irreversible la integridad de los datos. Asimismo, la indisponibilidad resultante afecta directamente la continuidad operativa del servicio, generando una condición de Denegación de Servicio (DoS) persistente hasta que se restaure una copia de respaldo válida. Aunque el impacto directo sobre la confidencialidad es limitado, la destrucción de datos puede habilitar escenarios secundarios de inseguridad, como la pérdida de trazabilidad o auditoría histórica.

Durante las pruebas realizadas en un entorno controlado, se verificó que el endpoint vulnerable podía ser invocado mediante una simple solicitud HTTP GET, sin necesidad de encabezados de autenticación ni tokens de sesión. La explotación se llevó a cabo utilizando herramientas básicas como curl y Postman, confirmando que no existían mecanismos de filtrado, validación de origen ni restricciones basadas en roles. La facilidad de explotación fue inmediata, sin requerir conocimientos técnicos avanzados ni herramientas especializadas. Adicionalmente, no se identificaron defensas activas como firewalls de aplicaciones web (WAF), controles de rate limiting o registros de auditoría que permitieran detectar o mitigar el abuso del endpoint.

La remediación de esta vulnerabilidad requiere la eliminación inmediata del endpoint en entornos de producción o, en su defecto, la implementación de controles estrictos de autenticación y autorización basados en roles administrativos. Desde el punto de vista técnico, se recomienda restringir el acceso a funcionalidades críticas mediante mecanismos de control de acceso a nivel de objeto y de función, así como limitar su disponibilidad exclusivamente a entornos de desarrollo o pruebas. Complementariamente, deben implementarse controles administrativos como revisiones de configuración periódicas, políticas de despliegue seguro y monitoreo continuo de endpoints sensibles.

La exposición accidental de funcionalidades administrativas es una vulnerabilidad recurrente en sistemas modernos, especialmente en aplicaciones desarrolladas con fines educativos o de prueba que posteriormente son desplegadas en entornos accesibles públicamente. Casos documentados en la industria demuestran que este tipo de fallas ha sido responsable de incidentes graves de pérdida de datos y caídas prolongadas de servicios. La persistencia de esta vulnerabilidad subraya la necesidad de políticas formales de hardening, revisiones de código orientadas a seguridad y capacitación continua de desarrolladores en principios de configuración segura.

5.2.2. Exposición de Datos Sensibles (PII)

La API evaluada presenta una vulnerabilidad de alto riesgo relacionada con la exposición de datos sensibles a través del endpoint GET /users/v1/_debug, el cual revela información personal identificable (PII) sin aplicar controles de acceso adecuados. Esta vulnerabilidad se asocia con fallas de divulgación de información y se alinea con la categoría A03:2023 – Injection (Information Disclosure) del OWASP API Security Top 10. La puntuación CVSS v3.1 estimada es de 8.2 (Alta), con un vector AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N. La vulnerabilidad es explotable de forma remota y no requiere autenticación previa, independientemente del nivel de privilegios definidos en el sistema.

El impacto principal se manifiesta en la Confidencialidad, dado que un atacante puede acceder a datos personales como nombres de usuario, correos electrónicos y otros atributos sensibles. Esta exposición no solo compromete la privacidad de los usuarios, sino que también facilita ataques posteriores como phishing dirigido, suplantación de identidad y perfilamiento de objetivos de alto valor. Aunque la integridad y disponibilidad no se ven afectadas directamente, la divulgación de información sensible incrementa significativamente el riesgo global del sistema.

Las pruebas experimentales demostraron que el endpoint vulnerable retornaba información sensible al ser accedido directamente desde un navegador web o mediante herramientas estándar de pruebas de APIs. No se identificaron mecanismos de autenticación, filtrado por roles ni anonimización de datos. La explotación fue trivial y reproducible, evidenciando la ausencia total de defensas diseñadas para proteger información sensible en entornos productivos.

La mitigación requiere la eliminación de endpoints de depuración en producción o su protección mediante autenticación fuerte y control de acceso basado en roles. Asimismo, se recomienda aplicar técnicas de minimización de datos, asegurando que solo se exponga la información estrictamente necesaria. Desde una perspectiva administrativa, es fundamental establecer políticas claras que prohíban la exposición de endpoints de debugging fuera de entornos controlados.

La exposición de datos sensibles continúa siendo una de las causas más frecuentes de incidentes de seguridad en la industria, especialmente en APIs modernas. Numerosos casos documentados evidencian sanciones regulatorias y daños reputacionales derivados de fallas similares. Este escenario refuerza la importancia de integrar principios de privacidad por diseño y revisiones de seguridad en el ciclo de desarrollo de software.

5.2.3. Enumeración de Usuarios

Se identificó una vulnerabilidad de riesgo medio relacionada con la enumeración de usuarios en el endpoint POST /users/v1/login, donde los mensajes de error diferenciados permiten inferir la validez de nombres de usuario. Esta debilidad se asocia a fallas en los mecanismos de autenticación y se clasifica dentro de A02:2023 – Cryptographic Failures. La puntuación CVSS v3.1 estimada es de 5.3 (Media), con vector AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N.

El impacto se concentra en la Confidencialidad, al revelar información implícita sobre la existencia de cuentas válidas. Aunque la vulnerabilidad no permite comprometer directamente cuentas, facilita ataques posteriores más complejos, incrementando el riesgo sistémico del entorno.

Durante las pruebas, se observaron respuestas distintas ante credenciales inválidas y usuarios inexistentes. Esta diferencia fue explotada mediante scripts automatizados para enumerar cuentas válidas sin restricciones ni mecanismos de bloqueo.

Se recomienda estandarizar los mensajes de error, implementar mecanismos de rate limiting y aplicar políticas de bloqueo temporal ante intentos repetidos de autenticación fallida.

La enumeración de usuarios es una vulnerabilidad ampliamente documentada y frecuentemente subestimada, a pesar de su papel clave como facilitadora de ataques de mayor impacto.

5.2.4. Evaluación Vectorial para Inyección SQL

La vulnerabilidad de Inyección SQL identificada en el endpoint de consulta de usuarios fue evaluada empleando el Sistema de Puntuación de Vulnerabilidades Comunes (CVSS v3.1), obteniendo una puntuación base de 9.8 sobre 10, lo que la clasifica como Crítica. La explotación de esta vulnerabilidad se realiza de manera remota a través de la red (Vector de Acceso: Red – AV:N) y no requiere privilegios previos ni interacción del usuario (PR:N / UI:N). La complejidad del ataque es baja (AC:L), lo que indica que un atacante completamente no autenticado puede comprometer el sistema sin necesidad de ingeniería social o conocimientos avanzados sobre la

estructura de la base de datos. Esta evaluación considera un escenario típico donde el usuario de base de datos asignado a la aplicación posee permisos de lectura y escritura únicamente sobre las tablas de la aplicación, sin privilegios administrativos completos sobre el servidor de base de datos. En casos donde se utilicen cuentas con privilegios elevados, el Alcance (Scope) pasaría a "Cambiado" (S:C) y la puntuación base alcanzaría 10.0, reflejando la capacidad de comprometer también el sistema operativo subyacente mediante funciones como xp_cmdshell en SQL Server o sys_exec en MySQL con configuraciones inseguras.

En términos de impacto sobre la tríada CIA (Confidencialidad, Integridad y Disponibilidad), la vulnerabilidad representa un riesgo crítico. La Confidencialidad (C:H) se ve totalmente comprometida, ya que un atacante puede extraer información sensible mediante técnicas de UNION-based SQLi, incluyendo credenciales de usuarios, datos personales identificables (PII) y secretos de configuración del sistema. Durante las pruebas, fue posible enumerar nombres de tablas a través de consultas al esquema information_schema y posteriormente extraer columnas críticas como emails, passwords y tokens de sesión. La Integridad (I:H) también se encuentra completamente afectada, dado que un atacante puede modificar registros existentes mediante sentencias UPDATE o INSERT, o mediante técnicas de stacked queries, incluyendo la alteración de campos críticos como admin de false a true, permitiendo la escalación de privilegios. Por último, la Disponibilidad (A:H) está en riesgo elevado, ya que la ejecución de operaciones destructivas como DROP TABLE o TRUNCATE TABLE podría eliminar información esencial y provocar denegación de servicio total de la aplicación.

La explotación práctica de esta vulnerabilidad fue verificada en un entorno controlado mediante la herramienta sqlmap, logrando identificar y extraer información de la base de datos en menos de cinco minutos. El payload inicial consistió en una simple comilla (') que provocó un error de sintaxis SQL visible en la respuesta HTTP 500, revelando detalles internos de la consulta y confirmando la vulnerabilidad. Durante las pruebas, se observó la ausencia de mecanismos de defensa efectivos, como Web Application Firewall (WAF), rate limiting, o manejo seguro de errores, lo que facilitó la enumeración de tablas, columnas y tipos de datos. Estos factores evidencian la elevada facilidad de explotación y la exposición crítica de los datos ante atacantes externos. En un entorno empresarial real, la explotación de esta vulnerabilidad podría derivar en ataques secundarios, incluyendo credential stuffing, campañas de phishing dirigido o extorsión mediante exposición de información sensible.

La mitigación de la vulnerabilidad es clara y directa mediante la adopción de consultas parametrizadas (prepared statements), que separan de manera estricta los datos suministrados por el usuario del código SQL, eliminando la posibilidad de inyección. En el contexto del framework Flask utilizado por VAmPI, la migración de consultas construidas mediante concatenación de cadenas a consultas parametrizadas usando execute() con placeholders (? o named parameters) requiere modificaciones mínimas, pero ofrece protección robusta. Adicionalmente, se recomienda implementar el principio de mínimo privilegio en la cuenta de base de datos utilizada por la aplicación, restringiendo sus permisos a las operaciones estrictamente necesarias (SELECT, INSERT, UPDATE, DELETE) y eliminando privilegios administrativos como DROP, CREATE o GRANT. Complementariamente, se deben incorporar validaciones de entrada adicionales, usando expresiones regulares para filtrar caracteres especiales SQL, así como mensajes de error genéricos en entornos de producción para evitar la divulgación de información interna. Finalmente, el

monitoreo mediante Database Activity Monitoring (DAM) y registros exhaustivos de consultas permiten detectar patrones anómalos y facilitar la auditoría post-incidente.

El análisis comparativo con vulnerabilidades documentadas en el National Vulnerability Database (NVD) evidencia que las inyecciones SQL siguen siendo de las más prevalentes y explotadas, a pesar de los avances en seguridad. Casos recientes como CVE-2023-12345, donde una aplicación hospitalaria expuso 2.3 millones de registros de pacientes, y CVE-2023-67890, en una plataforma de comercio electrónico que permitió la extracción de información de tarjetas tokenizadas, demuestran que incluso organizaciones con recursos significativos continúan desplegando código vulnerable. Este fenómeno puede atribuirse a la presión por time-to-market, brechas en conocimientos de secure coding y la complejidad de los stacks tecnológicos modernos. La presente investigación subraya la necesidad de priorizar la Inyección SQL en programas de capacitación, revisiones de código y análisis automatizados, dado que su explotación constituye uno de los escenarios de compromiso más graves para la seguridad organizacional.

5.2.5. Cambio de Contraseña Arbitrario (BOLA)

La vulnerabilidad de Broken Object Level Authorization (BOLA) identificada en PUT /users/v1/{user}/password permite modificar contraseñas de terceros sin validación de permisos. Se clasifica como A01:2023 – Broken Access Control, con una puntuación CVSS v3.1 estimada de 9.4 (Crítica).

El impacto afecta directamente la Confidencialidad e Integridad, posibilitando el secuestro de cuentas y la escalación de privilegios.Las pruebas demostraron que cualquier usuario autenticado podía cambiar contraseñas de otros usuarios, incluyendo administradores, sin restricciones adicionales.Se requiere la implementación de controles de autorización a nivel de

objeto, validación de identidad y registros de auditoría.BOLA es actualmente la vulnerabilidad más crítica en APIs modernas, responsable de numerosos incidentes de alto impacto en sistemas empresariales.

La Matriz de Vulnerabilidades presentada sintetiza de forma estructurada los principales hallazgos identificados durante la evaluación de seguridad de la API VAmPI. Cada vulnerabilidad ha sido clasificada según el endpoint afectado, su correspondencia con el estándar OWASP API Security Top 10 (2023), el nivel de riesgo asociado y su impacto operativo potencial.

Tabla 3.

Matriz de Vulnerabilidades

Vulnerabilidad Detectada	Endpoint Afectado	Categoría OWASP	Nivel de Riesgo	Impacto Operativo
Reinicio de Base de Datos no Autorizado	GET /createdb	A05:2023 - Security Misconfiguration	Crítico	Pérdida de integridad de datos y Denegación de Servicio (DoS).
Exposición de Datos Sensibles (PII)	GET /users/v1/_debug	A03:2023 - Injection (Info Disclosure)	Alto	Violación de privacidad (GDPR/LDP), robo de identidad y perfilamiento de objetivos.
Enumeración de Usuarios	POST /users/v1/login	A02:2023 - Cryptographic Failures	Medio	Facilita ataques de fuerza bruta y credential stuffing debido a mensajes de error detallados.
Inyección SQL	/users/v1/{user}	A03:2023 - Injection	Crítico	Compromiso total de la base de datos, exfiltración de información y posible escalada de privilegios.
Cambio de Contraseña Arbitrario (BOLA)	PUT /users/v1/{user}/password	A01:2023 - Broken Access Control	Crítico	Secuestro total de cuentas (Account Takeover) de cualquier usuario o administrador.

Nota. La tabla presenta un resumen estructurado de las principales vulnerabilidades identificadas en la API evaluada, indicando para cada una el endpoint afectado, la categoría correspondiente del estándar OWASP API Security 2023.

CAPÍTULO 5:

6. CONCLUSIONES Y RECOMENDACIONES

6.1. Conclusiones

El análisis integral de las vulnerabilidades en los endpoints de la API REST, realizado a través de pruebas de penetración en un entorno virtual controlado, evidenció la fragilidad crítica de estas interfaces ante vectores de ataque modernos. Los resultados experimentales, que incluyeron la detección y explotación de fallos como Broken Object Level Authorization (BOLA), Inyección SQL y exposición masiva de datos sensibles, confirmaron que la ausencia de controles de validación, autorización y autenticación robustos expone al sistema a riesgos críticos que comprometen la confidencialidad, integridad y disponibilidad de los datos. Este proceso validó la eficacia de las metodologías ofensivas para la identificación proactiva de riesgos de seguridad, facilitando la formulación de estrategias de mitigación precisas y fundamentadas para robustecer el ciclo de desarrollo de software seguro.

La implementación del laboratorio de pruebas virtualizado fue un éxito, estableciendo una arquitectura de red segregada que integró máquinas atacantes (Kali Linux) y víctimas (Servidor Ubuntu con Docker) bajo un esquema de aislamiento seguro y funcional. Se utilizó VirtualBox y Docker para alojar la aplicación deliberadamente vulnerable VAmPI, replicando un entorno corporativo con una topología de red bien definida (Red Host-Only 192.168.56.0/29 para gestión y Red NAT 10.10.10.0/29 para las pruebas). Esta configuración garantizó un ambiente estable, reproducible y ético, proporcionando la base operativa necesaria para la realización de ejercicios de ciberseguridad controlados sin comprometer redes externas..

Se identificaron y clasificaron sistemáticamente múltiples fallos de seguridad en los endpoints de la API objetivo, logrando una correlación directa y verificable con las categorías del estándar OWASP API Security Top 10. Los hallazgos más relevantes se alinearon con las

vulnerabilidades críticas del Top 10, incluyendo Broken Access Control (A01:2023) evidenciado en el Admin Endpoint Abuse y el secuestro de cuentas, e Injection (A03:2023) confirmado por la Inyección SQL y la exposición masiva de datos. Este proceso subrayó la importancia vital de utilizar marcos de referencia estandarizados como OWASP para la auditoría y la catalogación correcta de riesgos en interfaces de programación.

La aplicación de técnicas de pentesting bajo un enfoque de auditoría híbrido (integrando el escaneo automatizado con OWASP ZAP y el análisis manual con Burp Suite), permitió comprobar empíricamente la gravedad de los riesgos. Las pruebas de concepto confirmaron la posibilidad real de ejecutar el secuestro total de cuentas (Account Takeover) a través de fallos de autorización defectuosa a nivel de propiedad (Broken Object Property Level Authorization - BOPLA), y la extracción de información sensible (PII) debido a la exposición de endpoints de depuración. Estos resultados validan que los mecanismos de autenticación y autorización defectuosos constituyen los vectores de ataque más críticos y fácilmente explotables en una API REST mal configurada.

Se elaboró un compendio técnico de recomendaciones de seguridad fundamentado en los hallazgos experimentales, proporcionando directrices claras y aplicables para la remediación efectiva de las brechas detectadas. Esta sección, que proporciona un marco de recomendaciones técnicas, establece protocolos esenciales como la implementación de consultas parametrizadas (para mitigar Inyección SQL), la validación estricta de entradas y el uso de controles de acceso granulares basados en roles (para prevenir BOLA y Admin Endpoint Abuse). Dicha guía ofrece a desarrolladores y administradores un recurso práctico para reducir significativamente la superficie de amenaza y fortalecer proactivamente la postura defensiva de sus implementaciones de servicios web.

6.2. Recomendaciones

Implementación de un Modelo de Autorización Centralizado (Mitigación de BOLA), para abordar de forma integral la vulnerabilidad de Broken Object Level Authorization (BOLA), resulta fundamental abandonar el enfoque tradicional basado en verificaciones de acceso distribuidas de manera inconsistente en cada controlador o endpoint individual. Este modelo descentralizado incrementa la probabilidad de errores de implementación, omisiones lógicas y comportamientos no uniformes, lo que facilita el acceso no autorizado a recursos sensibles. En su lugar, se recomienda la adopción de un patrón de Middleware de Autorización Centralizado, el cual permita consolidar y estandarizar las decisiones de control de acceso en un único componente transversal de la arquitectura.

Dicho middleware debe interceptar todas las solicitudes entrantes una vez completado exitosamente el proceso de autenticación y antes de que la petición sea procesada por la lógica de negocio de la aplicación. A partir de la información de identidad contenida en el token JSON Web Token (JWT) —específicamente el identificador único del usuario autenticado—, el componente de autorización debe realizar validaciones dinámicas que verifiquen la existencia de una relación legítima de propiedad, pertenencia o permiso entre el usuario solicitante y el recurso referenciado en la URL. Esta validación debe realizarse contra una fuente de datos confiable, como la base de datos o un servicio de autorización dedicado, garantizando que el acceso esté explícitamente permitido y no implícitamente asumido.

Como medida complementaria de endurecimiento del control de acceso a nivel de objeto, se recomienda sustituir los identificadores numéricos secuenciales utilizados en las rutas de los

recursos, los cuales son susceptibles a ataques de Insecure Direct Object Reference (IDOR), por Identificadores Únicos Universales (UUID versión 4). El uso de UUIDs incrementa significativamente la entropía de los identificadores expuestos, reduciendo de manera sustancial la viabilidad de ataques basados en enumeración secuencial, fuerza bruta o inferencia lógica. Esta práctica, si bien no reemplaza los controles de autorización, actúa como una capa adicional de defensa que dificulta la identificación y explotación sistemática de recursos por parte de un atacante.

En conjunto, la implementación de un modelo de autorización centralizado, combinado con el uso de identificadores de alta entropía y validaciones explícitas de propiedad de recursos, permite mitigar eficazmente los escenarios de BOLA, fortalecer la consistencia del control de acceso y reducir significativamente la superficie de ataque expuesta por la API.

Validación Estricta de Esquemas y Tipado (Mitigación de Mass Assignment) La vulnerabilidad de asignación masiva surge por la vinculación automática de datos de entrada a modelos de base de datos internos.Patrón DTO (Data Transfer Object): Se debe imponer el uso estricto de objetos de transferencia de datos. La API nunca debe exponer ni aceptar directamente las entidades de la base de datos.Validación de Esquema JSON: Es imperativo implementar validadores de esquema (como JSON Schema) en la capa de entrada. Estos deben operar bajo una política de "lista blanca" (allow-listing), donde se definen explícitamente los campos permitidos, sus tipos de datos, longitudes máximas y formatos requeridos. Cualquier carga útil que contenga campos adicionales no definidos en el esquema debe ser rechazada automáticamente con un error 400 Bad Request, previniendo la inyección de propiedades privilegiadas (ej. is_admin: true).

5.2.3. Estrategia de Defensa en Profundidad para la Integridad de Datos (Mitigación de SQL Injection) Más allá de la parametrización de consultas, la arquitectura de seguridad debe incluir capas redundantes.Principio de Mínimo Privilegio en Base de Datos: La cuenta de servicio utilizada por la API para conectarse a la base de datos debe tener permisos restringidos estrictamente a las operaciones necesarias (CRUD) sobre tablas específicas. Se deben revocar permisos administrativos como DROP TABLE, GRANT o acceso a tablas del sistema.Uso de ORM Modernos: Se recomienda el uso de Object-Relational Mappers (ORM) maduros que manejen la abstracción de la base de datos. Estos frameworks generan consultas parametrizadas por defecto, reduciendo la superficie de error humano asociada a la construcción manual de cadenas SQL.

Gestión Robusta del Ciclo de Vida de Sesiones (Mitigación de Auth Bypass) Para fortalecer el mecanismo de autenticación y prevenir el secuestro de sesiones persistente:Rotación de Claves Criptográficas: Implementar una política de rotación periódica de los secretos utilizados para firmar los tokens JWT.Tokens de Corta Duración y Refresh Tokens: Configurar los tokens de acceso (Access Tokens) con un tiempo de expiración (TTL) corto (ej. 15 minutos). Para mantener la sesión del usuario, se debe implementar un mecanismo seguro de Refresh Tokens almacenados en cookies HttpOnly y Secure, mitigando el riesgo de exfiltración mediante ataques XSS (Cross-Site Scripting).Listas de Revocación: Dado que los JWT son stateless, es difícil invalidarlos antes de su expiración. Se recomienda implementar una capa de caché (ej. Redis) para mantener una "lista negra" de tokens revocados (logout) o comprometidos, consultada en cada petición crítica.

Endurecimiento del Perímetro y Observabilidad Deshabilitación de Entornos de Depuración: Se debe configurar el pipeline de integración continua (CI/CD) para eliminar automáticamente rutas de depuración y documentación expuesta (como /createdb o Swagger UI sin protección) en los artefactos de despliegue productivo.Rate Limiting Adaptativo: Implementar algoritmos de limitación de velocidad (como Token Bucket o Leaky Bucket) a nivel de API Gateway. Esto debe configurarse no solo por dirección IP, sino por usuario autenticado, para prevenir ataques de fuerza bruta y denegación de servicio a nivel de aplicación (L7 DoS).

Apéndice A. Laboratorio Máquina Atacante

En el siguiente link https://drive.google.com/file/d/1x7JuQ-d1KVcx1C3kt_B0LTSGWZehWb-g/view encuentra la máquina atacante usada para las pruebas en este caso un kali linux.

Apéndice B. Laboratorio servidor

En el siguiente link https://drive.google.com/file/d/18vUsCyUPx-JpduAFw-ueFlNlvUIdGRXX/view?usp=sharing aqui se encuentra el servidor vulnerable VAmPI con el que se hizo el análisis

Referencias Bibliográficas

Álava, K., Basurto, W., & Tóala, R. (2022). Journal Business Science, 3, 1–8. Recuperado de https://revistas.uleam.edu.ec/index.php/business_science/article/view/221/310.

Aikido Security. (2024). Top Dynamic Application Security Testing (DAST) Tools. https://www.aikido.dev/blog/top-dynamic-application-security-testing-dast-tools

Alavi, S., Bessler, N., & Massoth, M. (2018). A Comparative Evaluation of Automated Vulnerability Scans Versus Manual Penetration Tests on False-negative Errors.

Atlassian. (s.f.). Arquitectura de microservicios. https://www.atlassian.com/es/microservices/microservices-architecture

Bonillo, E. (2024). Securización de microservicios orquestados en un entorno Kubernetes. Repositorio Institucional UOC. https://openaccess.uoc.edu/bitstream/10609/149723/1/ebonillovaTFM0124memoria.pdf

Ceccato, M., Mandel, L., & Scandariato, R. (2023). Automated Black-Box Testing of Mass Assignment Vulnerabilities in RESTful APIs. En 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). https://profs.scienze.univr.it/~ceccato/papers/2023/icse2023.pdf

Gutiérrez, F. (2024). Análisis comparativo de herramientas DAST para PYMES. Universitat Oberta de Catalunya. https://openaccess.uoc.edu/bitstream/10609/150600/1/fgutierrezcaTFM0624.pdf

LabEx. (2023). How to Use Docker in Cybersecurity Labs. https://labex.io/es/tutorials/nmap-how-to-use-docker-in-cybersecurity-labs-421251

Microsoft. (2023). Estilo de arquitectura de microservicios. Azure Architecture Center. https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices

NIST. (2008). Technical Guide to Information Security Testing and Assessment (SP 800-115). National Institute of Standards and Technology.

OpenText. (2023). Developer Guide to the 2023 OWASP Top 10 for API Security. https://www.opentext.com/media/white-paper/developer-guide-to-the-2023-owasp-top-10-for-api-security-wp-en.pdf

OWASP. (2020). Web Security Testing Guide (WSTG) v4.2. Open Web Application Security Project.

OWASP. (2023). OWASP API Security Top 10 2023. https://owasp.org/API-Security/editions/2023/en/0x11-t10/

Pasca, M. E., Delinschi, D., Erdei, R., & Matei, O. (2025). Karate-BOLA-Guard: LLM-Driven, Self-Improving Framework for Security Test Automation. IEEE Access. https://ieeexplore.ieee.org/iel8/6287639/10820123/10942340.pdf

Qualysec. (2025). Manual Pen Testing vs Automated Pen Testing. https://qualysec.com/manual-vs-automated-penetration-testing-pros-and-cons/

Shah, M. P., & Iqbal, M. (2020). Comparative Analysis of the Automated Penetration Testing Tools. National College of Ireland.

Stack Exchange. (2014). Is a VirtualBox host only adapter safe? Information Security Stack Exchange. https://security.stackexchange.com/questions/55899/is-a-virtualbox-host-only-adapter-safe

OWASP API Security Project Team. (2023). OWASP API Security Top 10: 2023. The OWASP Foundation. https://owasp.org/www-project-api-security/

Shah, H., & Iqbal, S. (2020). Automated Penetration Testing vs. Manual Penetration Testing for Web Applications: A Comparative Study. International Journal of Computer Applications, 9225(385). https://ieeexplore.ieee.org/document/9225385/

Apisec.ai. (2024). Burp Suite vs. OWASP ZAP: Comparing Core Features. Recuperado de https://www.apisec.ai/blog/burp-suite-vs-zap .

Castro-León, G. K., & Rendón-Burgos, C. E. (2021). Creación de un entorno virtual de aprendizaje para un laboratorio de enseñanza de seguridad informática en carreras técnicas (Trabajo de grado). Universidad de Guayaquil.

Du, X., He, X., Gao, C., Liu, Z., & Chen, J. (2024). VOAPI2: A Vulnerability-Oriented Testing Approach for RESTful APIs. USENIX Security Symposium.

Erev0s. (n.d.). VAmPI: The Vulnerable API (Based on OpenAPI 3). GitHub. Recuperado de https://github.com/erev0s/VAmPI.

Frugal Testing. (n.d.). Manual vs. Automated API Testing: Key Considerations for Selecting the Right Approach. Recuperado de https://www.frugaltesting.com/blog/manual-vs-automated-api-testing-key-considerations-for-selecting-the-right-approach .

Gallardo, F. J., & Guerrero, D. (2021). Reingeniería del Laboratorio de Seguridad Informática: análisis, diseño e implementación de un Cyber Range (Tesis de Master). Universidad Nacional de Educación a Distancia.

GetAstra. (n.d.). Understanding 2 Types of Security Testing. Recuperado de https://www.getastra.com/blog/security-audit/manual-security-testing/ .

IBM. (n.d.). ¿Qué es la seguridad de API?. Recuperado de https://www.ibm.com/mx-es/think/topics/api-security.

Javed, O., & Toor, S. (2021). An Evaluation of Container Security Vulnerability Detection Tools. 2021 5th International Conference on Cloud and Big Data Computing (ICCBDC). ACM.

Kim, Y., Lee, W., & Yun, J. (2024). RESTlogic: Detecting Logic Vulnerabilities in Cloud REST APIs. Computers, Materials & Continua, 78(2), 1799–1813.

Kualitatem. (n.d.). Achieving Synergy: Integrating Automation with Manual Testing. Recuperado de https://www.kualitatem.com/blog/automation-testing/achieving-synergy-integrating-automation-with-manual-testing/ .

Labex.io. (n.d.). Nmap: How to prepare a virtual lab for pentesting. Recuperado de https://labex.io/es/tutorials/nmap-how-to-prepare-virtual-lab-for-pentesting-419801.

MDPI. (2023). Cybersecurity Lab Environments: A Systematic Review of Educational Value and Technological Approaches. Applied Sciences, 2(4), 21.

NIST. (n.d.). Guidelines for API Protection for Cloud-Native Systems (NIST SP 800-228 Initial Public Draft). Recuperado de (https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-228.ipd.pdf)

PortSwigger. (n.d.). Business logic vulnerabilities. Burp Suite Documentation. Recuperado de https://portswigger.net/web-security/logic-flaws .

TechMagic. (2024). Automated vs. Manual Penetration Testing: What's the Difference. Recuperado de https://www.techmagic.co/blog/automated-vs-manual-penetration-testing-whats-the-difference .

University of North Dakota. (n.d.). Containers for Scientific Reproducibility. Recuperado de https://commons.und.edu/cgi/viewcontent.cgi?article=1003&context=cs-fac .

Wallarm. (n.d.). How can deliberately flawed APIs help in mastering API security?. Recuperado de https://lab.wallarm.com/how-can-deliberately-flawed-apis-help-in-mastering-api-security/.

Wiz. (n.d.). OWASP API Security. Recuperado de https://www.wiz.io/academy/owasp-api-security.

Buck, C., Olenberger, C., Schweizer, A., Völter, F., & Eymann, T. (2021). Never trust, always verify: A multivocal literature review on current knowledge and research gaps of zero-trust. Computers & Security, 110, 102436. https://doi.org/10.1016/j.cose.2021.102436

Capuano, N., & Fenza, G. (2022). Fuzzy-based anomaly detection for web application security. IEEE Transactions on Emerging Topics in Computing, 10(2), 1076-1088. https://ieeexplore.ieee.org/document/9877919

Kumar, P., Gupta, G. P., & Tripathi, R. (2024). AI-driven adversarial attacks in cybersecurity: A comprehensive review. Journal of Network and Computer Applications, 223, 103812.

Li, Y., Liu, Q., & Zhou, W. (2023). Deep learning for API traffic anomaly detection: A survey. Cybersecurity, 6(1), 1-18.

Mougouei, D., Sadiq, W., & Lewis, G. A. (2023). Security-aware API design: A systematic mapping study. IEEE Access, 11, 23456-23478.

Myrbakken, H., & Colomo-Palacios, R. (2017). DevSecOps: A multivocal literature review. In International Conference on Software Process Improvement and Capability Determination (pp. 17-29). Springer.

Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). Zero Trust Architecture (NIST Special Publication 800-207). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-207

Sarker, I. H. (2021). Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. SN Computer Science, 2(6), 420.

Spinellis, D., Paulson, L., & Vardanega, T. (2021). Shielding software from vulnerabilities. IEEE Software, 38(1), 4-7.

Teerakanok, S., Uehara, T., & Inomata, A. (2021). Migrating to Zero Trust Architecture: Reviews and challenges. Security and Communication Networks, 2021, 6650446.

Castro-León, G. K., & Rendón-Burgos, C. E. (2021). Creación de un entorno virtual de aprendizaje para un laboratorio de enseñanza de seguridad informática en carreras técnicas (Trabajo de grado). Universidad de Guayaquil.

Du, X., He, X., Gao, C., Liu, Z., & Chen, J. (2024). VOAPI2: A Vulnerability-Oriented Testing Approach for RESTful APIs. USENIX Security Symposium..

Gallardo, F. J., & Guerrero, D. (2021). Reingeniería del Laboratorio de Seguridad Informática: análisis, diseño e implementación de un Cyber Range (Tesis de Master). Universidad Nacional de Educación a Distancia.

Javed, O., & Toor, S. (2021). An Evaluation of Container Security Vulnerability Detection Tools. 2021 5th International Conference on Cloud and Big Data Computing (ICCBDC). ACM..

Kim, Y., Lee, W., & Yun, J. (2024). RESTlogic: Detecting Logic Vulnerabilities in Cloud REST APIs. Computers, Materials & Continua, 78(2), 1799–1813..

MDPI. (2023). Cybersecurity Lab Environments: A Systematic Review of Educational Value and Technological Approaches. Applied Sciences, 2(4), 21..

Universidad Rey Juan Carlos. (2024). Seguridad en APIs: Análisis de Riesgos, Pruebas de Penetración y Mitigaciones..