Maestría en

CIBERSEGURIDAD

Trabajo previo a la obtención de título de
Magister en Ciberseguridad

AUTOR/ES:

Barreno Gómez Henry

Guerrero Garrido Saskia

Romero Cedeño Nilson

Sandoval Haro Sebastián

TUTORES:

Iván Reyes Chacón

Juan Fernando López

TEMA

Análisis de vulnerabilidades y fortalecimiento de la seguridad en implementaciones de APIs RESTful mediante el estándar OAuth 2.0 y validación criptográfica de token JWT.

Quito - Ecuador

Junio – 20206

REINVENTEMOS EL FUTURO

Certificación de autoría

Nosotros, **Henry Jeanpiere Barreno Gómez, Saskia Rebeca Guerrero Garrido, Nilson Michael Romero Cedeño, Sebastián Josué Sandoval Haro,** 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.

Henry Jeanpiere Barreno Gomez
Time Stamping Security Data

Firma
Henry Jeanpiere Barreno Gómez

Saskia Rebeca Guerrero Garrido
Time Stamping Security Data

Firma
Saskia Rebeca Guerrero Garrido

Nilson Michael Romero Cedeno
Time Stamping Security Data

Firma
Nilson Michael Romero Cedeño

Sebastian Josue Sandoval Haro
Time Stamping Security Data

Firma
Sebastián Josué Sandoval Haro

Autorización de Derechos de Propiedad Intelectual

Nosotros, Henry Jeanpiere Barreno Gómez, Saskia Rebeca Guerrero Garrido, Nilson Michael Romero Cedeño, Sebastián Josué Sandoval Haro, en calidad de autores del trabajo de investigación titulado Análisis de vulnerabilidades y fortalecimiento de la seguridad en implementaciones de APIs RESTful mediante el estándar OAuth 2.0 y validación criptográfica de token JWT, 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.

Quito, (julio, 2026)

Henry Jeanpiere Barreno Gomez
Time Stamping Security Data

Firma
Henry Jeanpiere Barreno Gómez

Saskia Rebeca Guerrero Garrido
Time Stamping Security Data

Firma
Saskia Rebeca Guerrero Garrido

Nilson Michael Romero Cedeno
Time Stamping Security Data

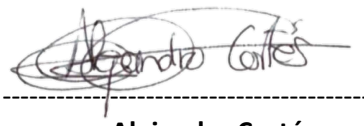
Firma
Nilson Michael Romero Cedeño

Sebastian Josue Sandoval Haro
Time Stamping Security Data

Firma
Sebastián Josué Sandoval Haro

Aprobación de dirección y coordinación del programa

Nosotros, **Alejandro Cortés EIG e Iván Reyes UIDE,** declaramos que: **Henry Jeanpiere Barreno Gómez, Saskia Rebeca Guerrero Garrido, Nilson Michael Romero Cedeño, Sebastián Josué Sandoval Haro** son los autores exclusivos de la presente investigación y que ésta es original, auténtica y personal de ellos.



Alejandro Cortés
Director/a de la
Maestría en Ciberseguridad

Iván Reyes
Coordinador/a de la
Maestría en Ciberseguridad

Dedicatoria

Dedico este trabajo a mis padres, Beatriz Gómez y Henry Barreno, por su amor incondicional, esfuerzo y apoyo constante a lo largo de mi vida. Gracias por ser mi ejemplo de perseverancia y por brindarme la fortaleza necesaria para alcanzar cada una de mis metas. A mis hermanos, Nathaly y Josué, por su compañía, motivación y confianza en cada etapa de este camino. Este logro también les pertenece a ustedes, porque han sido una parte fundamental de mi crecimiento personal y profesional.

Henry Barreno

Dedico este proyecto de titulación a mis padres, hermanos y familia, quienes durante este año me han brindado su apoyo incondicional. Gracias por darme las fuerzas para continuar con mis estudios y permitirme llegar a donde estoy en este momento; ha sido un largo camino, pero hoy veo con orgullo cuánto he avanzado.

Saskia Guerrero

Dedico este proyecto de titulación a mi familia quienes me brindaron su apoyo incondicional y motivación para cumplir este logro. Gracias por haberme acompañado en este camino. Esta meta cumplida me permite comprender que soy capaz de lograr lo que me propongo y motiva a cumplir objetivos más grandes.

Nilson Romero

Dedico este logro a mi familia, pilar fundamental y fuente incondicional de fortaleza y aliento. A Dios, por otorgarme la salud, las habilidades y la guía para alcanzar mis metas. A mí mismo, en reconocimiento a la constancia, el esfuerzo nocturno y el compromiso personal. A todos los que formaron parte de este viaje, gracias por su apoyo en la culminación de esta meta.

Sebastián Sandoval

Agradecimientos

Agradezco a Dios por brindarme la fortaleza y la perseverancia para culminar esta importante etapa. A mis padres, hermanos y familiares, por su apoyo incondicional y constante motivación. A la Universidad Internacional del Ecuador y a sus docentes, por los conocimientos y herramientas aportados a mi formación profesional. Asimismo, agradezco a mis compañeros y a todas las personas que contribuyeron con su apoyo y confianza para la realización de este trabajo.

Henry Barreno

Expreso mi sincero agradecimiento, en primer lugar, a Dios por darme la fortaleza para culminar esta etapa. A la Universidad Internacional del Ecuador (UIDE) por abrirme sus puertas y brindarme las herramientas necesarias en mi formación académica. Asimismo, agradezco a los docentes de este proceso por compartir sus conocimientos, y a mis compañeros quienes formaron parte de este proyecto de titulación por el compromiso y la colaboración técnica demostrada en cada fase del desarrollo.

Saskia Guerrero

Agradezco profundamente a mi familia por su apoyo incondicional, comprensión y constante motivación a lo largo de este proceso académico. Su confianza y acompañamiento fueron fundamentales para superar los desafíos y alcanzar esta importante meta. Gracias por estar presentes en cada etapa del camino y por impulsarme a seguir adelante incluso en los momentos más difíciles. Este logro representa no solo la culminación de un esfuerzo académico, sino también la confirmación de que la perseverancia y el compromiso permiten alcanzar los objetivos propuestos, motivándome a asumir nuevos retos y metas aún más ambiciosas en el futuro.

Nilson Romero

Agradezco profundamente a Dios por guiar mi camino con fe, y a mi amada familia, el pilar más sagrado de mi vida, por sostener mi alma con su amor y aliento en los días más difíciles. Mi reconocimiento a los docentes por su enseñanza y fortalecimiento en mi desarrollo profesional, y a mis compañeros de titulación, con quienes compartí desvelos donde se transformó en un sacrificio para llegar a un triunfo compartido. Finalmente, abrazo con orgullo a mi propio ser; me agradezco infinitamente por creer en mí, por el arduo trabajo sin descanso y jamás renunciar a este gran sueño.

Sebastián Sandoval

RESUMEN

En este proyecto se evaluará la solidez técnica de los marcos de autenticación y autorización basados en OAuth 2.0 y JSON Web Tokens (JWT) en entornos de microservicios, ya que existen crecientes riesgos que enfrentan las API RESTful. La investigación se centra en identificar vulnerabilidades en la autenticación criptográfica y en el manejo de solicitudes que permiten la suplantación de identidad y la elusión de controles.

Utilizando un laboratorio de pruebas de Docker, construido en un entorno de tres servicios en Node.js y Express con almacenamiento local, se llevan a cabo revisiones basadas en el OWASP API Security Top 10 (2023) para evaluar los riesgos de BOLA, Broken Authentication, BFLA y Security Misconfiguration y también se implementa PKCE mediante la herramienta Keycloak. Finalmente, se utiliza Kong API Gateway y middleware de aplicaciones, se compara el estado de referencia con el estado mejorado para medir la eficacia de la mitigación de riesgos, y se establece un modelo de interacción reproducible para fortalecer los sistemas distribuidos.

Palabras clave: Owasp, API RESTful, Kong, Keycloack, ataques, vulnerabilidades, Docker, JWT

ABSTRACT

This project will assess the technical robustness of authentication and authorisation frameworks based on OAuth 2.0 and JSON Web Tokens (JWT) in microservices environments, given the growing risks faced by RESTful APIs. The research focuses on identifying vulnerabilities in cryptographic authentication and request handling that enable impersonation and the circumvention of controls.

Using a Docker test lab, built in a three-service environment using Node.js and Express with local storage, reviews based on the OWASP API Security Top 10 (2023) are carried out to assess the risks of BOLA, Broken Authentication, BFLA and Security Misconfiguration, and PKCE is also implemented using the Keycloak tool. Finally, Kong API Gateway and application middleware are utilised; the baseline state is compared with the improved state to measure the effectiveness of risk mitigation; and a reproducible interaction model is established to strengthen distributed systems.

Keywords: OWASP, RESTful API, Kong, Keycloak, attacks, vulnerabilities, Docker, JWT

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE FÓRMULAS

CAPÍTULO 1

1. Introducción

En este capítulo se explicará el motivo del por qué es importante realizar un análisis de vulnerabilidades en APIs RESTful. Según Tanveer (2025), “La creciente adopción de las API RESTful ha expuesto simultáneamente estas interfaces a importantes amenazas de seguridad que ponen en riesgo la disponibilidad, la confidencialidad y la integridad de los servicios web.” En ese sentido, este proyecto hace énfasis en los ataques relacionados con fallos de autorización, autenticación y seguridad en APIs RESTful, que son:

API1:2023 Broken Object Level Authorization – BOLA

Tipo de ataque: Insecure Direct Object Reference (IDOR)

Justificación: Dado que el desarrollo de la interfaz de usuario no está dentro del alcance de este estudio, la evaluación de seguridad se centra exclusivamente en el nivel de la aplicación. Este enfoque metodológico permite identificar el comportamiento del backend en relación con la vulnerabilidad BOLA – API01:2023, clasificada como la amenaza más grave en el OWASP API Security Top 10. OWASP (2023), donde OWASP es reconocida como el estándar referencial global para la identificación y clasificación de vulnerabilidades críticas en el software. OWASP (2023), y demostrar directamente las consecuencias de este fallo lógico dentro de una arquitectura distribuida con microservicios.

API2:2023 - Broken Authentication (Autenticación Rota)

Tipo de ataque: Ataque de firma criptográfica.

Justificación: Dado que en este sistema se utiliza Keycloak para la emisión de tokens JWT, la autenticación se lleva a cabo de forma descentralizada (stateless). Por este motivo, durante las pruebas es necesario simular el procesamiento de estos tokens. Este procedimiento permite evaluar la seguridad del perímetro y verificar si la puerta de enlace API Gateway es capaz de interceptar las

solicitudes y verificar correctamente la firma criptográfica, utilizando la clave pública de Keycloak, antes de que el tráfico llegue a la API, previniendo así los ataques de suplantación de identidad.

API5:2023 - Broken Function Level Authorization (BFLA)

Tipo de ataque: escalamiento de privilegios

Justificación: La arquitectura del proyecto brinda acceso por roles estructurados dentro de los tokens generados por Keycloak (JWT), los cuales se definen como un estándar abierto (RFC 7519) utilizado para transmitir información de forma segura entre las partes como un objeto JSON. Jones et al., (2015). Se realizará la ejecución de solicitudes HTTP de naturaleza destructiva mediante el método DELETE. Donde HTTP es entendido como el protocolo de transferencia de hipertexto base para el intercambio de datos en la web que utiliza métodos específicos para indicar las acciones deseadas. Fielding et al., (2022). Esta prueba evalúa si el microservicio restringe el acceso de forma nativa o si permite el escalamiento vertical de privilegios a actores no autorizados.

API8:2023 - Security Misconfiguration

Tipo de ataque: bypass de control perimetral por exposición de puertos

Justificación: El proyecto no incluye medidas destinadas a mejorar la seguridad de la red, el objetivo de esta prueba es demostrar que es posible acceder directamente a la API a través de los puertos expuestos por Docker. Permitiendo presentar la evasión de la API Gateway y hacer un bypass de la lógica de seguridad y los controles en los límites de acceso.

En la actualidad en el desarrollo de aplicaciones o servicios web la mayoría utiliza una o varias APIs RESTful, en el cual se emplean los protocolos HTTP y HTTPS los cuales permiten generar, transportar y comunicar los datos. Sin embargo, el hecho de que exista la confidencialidad e integridad de los datos no significa que no estén expuestos a ataques críticos, lo que genera amenazas en la seguridad, como escalamiento de privilegios, acceso no autorizado, configuraciones

inseguras, entre otras. Por tal motivo este proyecto busca analizar el fortalecimiento de la seguridad en las APIs.

1.1. Definición del proyecto

En la actualidad las APIs son parte crucial de las comunicaciones de las diferentes aplicaciones tanto web como móviles y el uso de estas pueden abarcar diferentes servicios críticos. Por lo que es crucial entender el comportamiento de estas para asegurar el consumo. En la actualidad existen diversas implementaciones de APIs con configuraciones predeterminadas o que únicamente mantienen la seguridad en sus seguridades de frontera. OWASP (2023). Debido a esto, se prescinde de la estrategia de defensa en profundidad; por lo tanto, los diferentes microservicios quedan expuestos a ser consumidos sin una autorización previa.

El presente proyecto propone el análisis y la evaluación comparativa de la seguridad en arquitecturas de microservicios, mediante la construcción de dos entornos: uno configurado con parámetros predeterminados y otro robustecido bajo los principios de Zero Trust y el framework de autorización OAuth 2.0.

Con este propósito, se diseñarán dos infraestructuras lógicas que gobernarán un conjunto de APIs REST desarrolladas en Node.js con Express, donde una librería (o biblioteca de software) se define como una colección de funciones, módulos y componentes de código reutilizables que facilitan el desarrollo de aplicaciones al resolver tareas específicas de programación sin necesidad de escribirlas desde cero. La primera reproducirá un escenario de confianza implícita, en el que la seguridad recae exclusivamente en el perímetro de red. Bajo este enfoque, se asume que el tráfico interno entre contenedores es confiable y se omiten las validaciones explícitas de los middlewares. La segunda arquitectura eliminará dicha confianza implícita mediante un modelo de defensa en profundidad estructurado en dos niveles de control. En el primer nivel, Keycloak actuará como servidor de autorización centralizado, responsable de autenticar a los sujetos y de propagar

atributos de identidad y privilegio a través de claims criptográficos contenidos en los tokens JWT. En el segundo nivel, el cumplimiento normativo se distribuirá entre dos componentes especializados. Por una parte, Kong (API Gateway) operará como punto de control perimetral encargado de verificar la autenticidad criptográfica de la firma y la vigencia temporal de cada token mediante consultas al endpoint JWKS de Keycloak. Por otra parte, las APIs en Node.js ejecutarán middlewares internos de aplicación orientados a la auditoría dinámica de roles y al análisis contextual del recurso solicitado.

A fin de garantizar la fidelidad, el aislamiento y la replicabilidad de los escenarios de auditoría, ambas infraestructuras serán desplegadas mediante contenedores Docker. El núcleo técnico de la investigación aborda la problemática derivada de implementaciones en las que la verificación de tokens JWT es parcial o inexistente, condición que origina vulnerabilidades críticas de suplantación de identidad y escalada de privilegios. A través de un análisis dinámico ejecutado en un entorno controlado y autogestionado, el proyecto busca identificar las debilidades estructurales presentes en el flujo de OAuth 2.0. Asimismo, se pretende cuantificar el impacto de las mitigaciones propuestas sobre métricas clave, tales como la tasa de rechazo de tokens inválidos, la tasa de accesos no autorizados bloqueados y la latencia en el tiempo medio de validación criptográfica. Toda esta evaluación se realiza en contraste directo con las amenazas definidas en el estándar OWASP API Security Top 10.

1.2. Justificación e importancia del trabajo de investigación

La presente investigación adopta como referencia el marco OWASP API Security Top 10 (OWASP, 2023). Este modelo es ampliamente reconocido como uno de los principales estándares para la identificación y mitigación de riesgos en APIs. No obstante, en función de los objetivos y del alcance del estudio, se seleccionaron cuatro vulnerabilidades críticas considerando tres criterios. Mitchell et al., (2024):

1. **Alta prevalencia:** su recurrencia en evaluaciones de seguridad recientes.

2. **Impacto crítico:** su efecto directo sobre los mecanismos de autenticación y autorización en arquitecturas de microservicios.
3. **Enfoque holístico:** la necesidad de evaluar una estrategia de Defensa en Profundidad que integre controles tanto en la capa de aplicación como en la infraestructura.

A partir de los criterios señalados, se estructuró la selección de las amenazas específicas que formarán parte de la evaluación experimental, las cuales se describen de manera detallada a continuación:

Broken Object Level Authorization (BOLA). Esta vulnerabilidad fue seleccionada debido a que se ha mantenido como uno de los principales riesgos identificados por OWASP para las APIs desde la publicación de la primera versión del ranking en 2019 (OWASP, 2023). BOLA se produce cuando una API no valida adecuadamente si el usuario autenticado posee autorización para acceder a un recurso específico solicitado mediante identificadores expuestos en la petición. Este escenario resulta especialmente crítico en arquitecturas de microservicios, donde es común que los recursos sean consultados a través de parámetros incluidos en las URI, las cuales se definen como Identificadores de Recursos Uniformes que consisten en una cadena de caracteres estandarizada utilizada para identificar de forma única y unívoca un recurso lógico o físico en una red. Berners-Lee et al., (2005). La ausencia de verificaciones de propiedad o pertenencia puede derivar en accesos no autorizados a información sensible, comprometiendo la confidencialidad e integridad de los datos gestionados por la aplicación.

Broken Authentication. La autenticación constituye el primer mecanismo de protección para restringir el acceso a los servicios expuestos por una API. Las debilidades en la validación de credenciales, la gestión inadecuada de sesiones o la aceptación de tokens manipulados pueden permitir la suplantación de identidades legítimas y el acceso indebido a recursos protegidos. En arquitecturas que emplean OAuth 2.0 y JWT, la correcta verificación de la firma criptográfica, la

vigencia temporal y la integridad de los tokens adquiere una importancia crítica, ya que cualquier deficiencia en estos controles puede comprometer la confianza depositada en el proveedor de identidad y en el proceso de autenticación distribuida. (OWASP, 2023)

Broken Function Level Authorization (BFLA). Esta vulnerabilidad se relaciona con la ausencia o implementación deficiente de controles que restrinjan el acceso a funcionalidades de acuerdo con los privilegios asignados a cada usuario. Su inclusión en esta investigación se justifica porque las arquitecturas basadas en microservicios suelen diferenciar funciones administrativas y operativas mediante roles o atributos contenidos en los tokens de acceso. Cuando estos mecanismos no son validados correctamente, usuarios con privilegios limitados pueden ejecutar acciones reservadas para perfiles de mayor jerarquía, generando escenarios de escalamiento de privilegios y acceso no autorizado a operaciones críticas del sistema. (OWASP, 2023).

Security Misconfiguration. A diferencia de las vulnerabilidades anteriores, centradas principalmente en la lógica de autenticación y autorización, esta categoría aborda deficiencias derivadas de configuraciones inseguras en los componentes tecnológicos que soportan la aplicación. Entre ellas se incluyen la exposición innecesaria de servicios, el uso de configuraciones predeterminadas, la divulgación de información sensible mediante mensajes de error y la ausencia de restricciones adecuadas en los puntos de acceso. Su evaluación resulta pertinente debido a que la seguridad de una arquitectura de microservicios no depende exclusivamente de la robustez del código, sino también de la correcta configuración de los contenedores, el API Gateway y los mecanismos de protección perimetral empleados durante el despliegue. (OWASP, 2023).

A partir de estas vulnerabilidades, la propuesta plantea una estrategia basada en el principio de defensa en profundidad, integrando controles en diferentes niveles de la arquitectura. En particular, se analizará la interacción entre el proveedor de identidad (IdP), las políticas implementadas en el API Gateway, los mecanismos de autenticación y autorización incorporados en

los microservicios y la configuración segura del entorno contenerizado. De esta manera, se busca demostrar que la mitigación efectiva de los riesgos identificados por OWASP requiere la aplicación coordinada de controles técnicos en las distintas capas de una arquitectura basada en microservicios.

1.3. Relevancia Académica y Técnica

Este trabajo de investigación parte de la necesidad de mejorar la seguridad de las API mediante la implementación de mecanismos de autenticación y autorización basados en OAuth 2.0 y JSON Web Token (JWT). Actualmente, OAuth 2.0 es uno de los estándares más utilizados para la gestión de la autorización. Sin embargo, su adopción no garantiza la seguridad de una API, ya que una implementación incorrecta puede provocar vulnerabilidades relacionadas con la autenticación, autorización y configuración de seguridad; por lo tanto, este estudio tiene como objetivo corregir las brechas existentes entre el detalle teórico de los estándares y su implementación práctica, centrándose en aspectos críticos como la validación de firmas digitales, la verificación de requisitos de autorización, el control de privilegios basado en roles, la configuración correcta del Api Gateway y la validación de la caducidad de los tokens. Estos mecanismos se evalúan teniendo en cuenta las vulnerabilidades identificadas por OWASP API Security 2023, de las cuales se trabajarán sobre 4 de las 10 vulnerabilidades que existen.

1.4. Impacto en la Viabilidad y Seguridad

La solidez de la arquitectura se garantizará mediante la integración de mecanismos de seguridad diseñados para la autenticación y autorización de API, incluyendo OAuth 2.0, tokens web JSON (JWT) firmados con RS256, validación de reclamaciones, control de acceso basado en roles, protección a través de API Gateway y la aplicación de las mejores prácticas recomendadas por OWASP API Security 2023. La eficacia de estos mecanismos se verificará mediante pruebas de seguridad diseñadas para evaluar la resistencia frente a vulnerabilidades como broken

authentication, BOLA, BFLA y security misconfiguration. De este modo, la solidez de la arquitectura se medirá por la capacidad del sistema para detectar y bloquear intentos de manipulación de tokens, escalada de privilegios, acceso no autorizado y configuraciones inseguras.

1.5. Métricas de evaluación

A continuación, se presentan los indicadores con los que se evaluarán las vulnerabilidades:

- ***Índice de Eficacia Perimetral (IEP):*** Porcentaje global de éxito del API Gateway (Kong) en la contención de amenazas en la frontera de la arquitectura. Donde la siguiente fórmula resume los ataques: Broken Object Level Authorization (BOLA), Broken Authentication, Broken Function Level Authorization (BFLA). Adicional se suma los ataques de Security Misconfiguration, de esta forma unificamos aquellos errores 401 y 429, a continuación, se explica detalladamente los parámetros de dicha ecuación.

Fórmula 1 *Índice de Eficacia Perimetral (IEP)*

$$\mathbf{IEP} = \left(\frac{\mathbf{SB401} + \mathbf{SB429}}{\mathbf{TS}}\right) * \mathbf{100}\%$$

(1)

Donde *IEP* representa el *Índice de Eficacia Perimetral*, *SB 401 (Solicitudes Bloqueadas con código de error 401)* Cantidad de peticiones anómalas que Kong interceptó y denegó devolviendo un código de estado HTTP 401 (Unauthorized). Esto agrupa los intentos de bypass con tokens malformados, firmas alteradas o tokens con el atributo de expiración (exp) vencido. *SB 429 (Solicitudes Bloqueadas con código de error 429)* Rate Limiting cantidad de peticiones concurrentes que superaron el umbral permitido y que Kong bloqueó devolviendo un código de estado HTTP 429 (Too Many Requests). *TS (Total de Solicitudes de Ataques Perimetrales)* es el total de peticiones

maliciosas enviadas por las herramientas de automatización durante la fase de pruebas físicas contra Kong.

- ***Índice de Eficacia de la Autorización Lógica (IEA):*** Este indicador evalúa la capacidad de los middlewares internos para interceptar y abortar solicitudes que intentan explotar fallas lógicas de acceso. La siguiente fórmula agrupa las respuestas de error HTTP 403. Estas respuestas se generan por dos motivos específicos: cuando el usuario intenta realizar acciones que no corresponden a su rol, y cuando intenta consultar información que no es de su propiedad.

  Fórmula 2 *Índice de Eficacia de la Autorización Lógica (IEA)*

$$\boldsymbol{IEA = \left(\frac{SB403}{TS}\right) X100\%}$$

(2)

Donde *IEA* representa la eficacia de autorización lógica, *SB403):* Número de peticiones anómalas que superaron la seguridad perimetral de Kong, pero que fueron interceptadas y denegadas por los middlewares de las APIs emitiendo un código de estado HTTP 403 (Forbidden) y *TS* son todas las solicitudes maliciosas dirigidas a evadir el control de acceso basado en roles o la propiedad de los recursos.

1.6. Alcance

El proyecto se centra en el análisis y fortalecimiento de la seguridad en una arquitectura de microservicios, donde la comunicación entre componentes está autenticada mediante el estándar OAuth 2.0 y el uso de JSON Web Tokens (JWT). El estudio se desarrollará bajo los siguientes parámetros:

1.6.1. Entorno Tecnológico y Viabilidad.

- **Infraestructura de Microservicios:** Se desplegará un entorno controlado basado en contenedores (Docker), evitando el uso de servicios de nube.
- **APIs de Evaluación:** Se desarrollarán seis APIs RESTful, tres con configuraciones vulnerables y tres con configuraciones robustas con el objetivo de evaluar los mecanismos de autenticación, autorización y seguridad. Estas APIs simularán un entorno de comercio electrónico dedicado a la venta de productos tecnológicos, como ordenadores portátiles, teclados, monitores y otros dispositivos informáticos. En esta arquitectura, se implementarán y evaluarán los escenarios de seguridad relacionados con las vulnerabilidades mencionadas más adelante. En el capítulo 3 se encuentra detallado la función que cumple cada API.
- **Servidor de Identidad (IdP):** Keycloak se utilizará como el servidor de identidad para gestionar los procesos de autenticación y autorización de los usuarios. Esta herramienta permitirá la gestión centralizada de identidades, roles y autorizaciones, así como la generación y validación de tokens. Además, Keycloak simplificará la configuración de elementos de seguridad como los algoritmos de firma, el tiempo de caducidad de los tokens y las reglas de acceso, se constituye como el componente central de la gestión de identidades dentro de la arquitectura propuesta.

1.6.2. Validación y Pruebas Ofensivas (OWASP API Security)

- **Análisis de Vulnerabilidades:** Se evaluará la resistencia del sistema ante fallas frecuentes como Broken Authentication, Broken Object Level Authorization (BOLA), Broken Function Level Authorization (BFLA) y Security Misconfiguration.
- **Simulación de Ataques:** El alcance incluye pruebas dirigidas a:

- **Manipulación de Claims:** Intentos de modificar roles o privilegios dentro del JWT.
- **Bypass de Firma:** Pruebas de cambio de algoritmo (ataque alg: none) para evadir la validación del Gateway.
- **Replay Attacks:** Evaluación de la gestión de tiempos de expiración (exp) y audiencia (aud).

1.7. Fuera de alcance

Seguridad física y perimetral de la infraestructura: Este trabajo no se detendrá en la seguridad física de los servidores, como el acceso a centros de datos o el resguardo del hardware. El proyecto se enfoca totalmente en la seguridad lógica.

Hardening de red y sistemas operativos: Este trabajo no incluye la configuración de dispositivos físicos de red como firewalls, sistemas de detección de intrusos (IDS/IPS), redes privadas virtuales (VPN) ni el endurecimiento del sistema operativo donde residen las APIs. El análisis se centra en el protocolo de autenticación y la estructura de los tokens.

Desarrollo de interfaces de usuario (Frontend): No se diseñará ni implementará ninguna interfaz gráfica o aplicación cliente para el usuario final en este proyecto. La validación técnica del modelo se realizará mediante herramientas de pruebas de APIs y scripts de automatización de seguridad en el backend.

Gestión de identidades a nivel organizacional: En este trabajo no abarca la integración con directorios activos (LDAP/AD) ni la gestión de gobernanza de identidades complejas, limitándose al flujo técnico de autorización entre el cliente, el servidor de autorización y las APIs analizadas.

Bases de datos externas: El presente trabajo no contempla la implementación ni administración de motores de bases de datos relacionales o no relacionales. Con el fin de mantener el enfoque de la investigación en la evaluación de mecanismos de autenticación, autorización y

seguridad en APIs, la persistencia de la información se realizará mediante archivos estructurados almacenados localmente en cada microservicio.

1.8. Objetivos

1.8.1. Objetivo general.

Desarrollar y evaluar un modelo de fortalecimiento de la seguridad para arquitecturas basadas en microservicios mediante la implementación de OAuth 2.0, PKCE y mecanismos de validación criptográfica de JSON Web Tokens (JWT), utilizando una Prueba de Concepto (PdC) integrada por tres APIs y un API Gateway.

1.8.2. Objetivos específicos.

- Diseñar e implementar la arquitectura de la prueba de concepto (PdC) integrada por el API Gateway, las tres APIs y el proveedor de identidad (IdP), configurando los mecanismos de OAuth 2.0, PKCE y validación criptográfica de tokens para establecer el entorno de control base.
- Definir indicadores y métricas de efectividad cuantificables, tales como la tasa de rechazo de tokens malformados y el índice de bloqueo de accesos no autorizados.
- Ejecutar simulaciones de ataques dirigidos a la manipulación de claims y elusión de algoritmos de firma (alg: none) para determinar la vulnerabilidad de una arquitectura estándar frente a una que contenga buenas prácticas de seguridad.
- Comparar el nivel de seguridad y el desempeño técnico entre una implementación de API con configuraciones por defecto y una optimizada con un esquema de validación estricta.

CAPÍTULO 2

2. Revisión de literatura

2.1. Estado del Arte

2.1.1. Estudios recientes sobre los mecanismos de seguridad y vulnerabilidades en APIs RESTful.

En los últimos años, se evidencia en investigaciones el uso de mecanismos en tokens que fortalecen los procesos de autenticación y autorización en APIs RESTful. Con el propósito de contextualizar académicamente este escenario, este apartado expone un análisis crítico y comparativo de los antecedentes científicos más recientes a nivel internacional. A continuación, se examinan las metodologías, propuestas tecnológicas y herramientas automatizadas desarrolladas por diversos autores para la detección y mitigación de fallos lógicos de control de acceso, centrándose especialmente en las amenazas críticas catalogadas en el estándar OWASP. Este recorrido por el estado del arte permitirá identificar las tendencias actuales de la industria y justificar el enfoque técnico adoptado en la presente investigación, iniciando con las siguientes contribuciones:

Para Santos Filho et al. (2025), la propuesta de este artículo científico fue que crearon un método automatizado que detecta ataques en este caso de Broken Object-Level Authorization (BOLA) en una APIs RESTful llamada ‘running example’ esta API definida formalmente como una interfaz de programación de aplicaciones que establece el conjunto de reglas, funciones y protocolos para permitir la comunicación y el intercambio de datos seguro entre componentes de software fue tomada de OWASP Juice Shop, la cual constituye una aplicación web desarrollada en Node.js deliberadamente insegura y de código abierto, diseñada por la organización para el entrenamiento práctico, la demostración y la simulación de las vulnerabilidades del Top 10. Para detectar el ataque utilizaron Colored Petri Nets (CPN), este representó flujos, estados y relaciones entre usuarios y recursos es decir los fallos en la

autorización. El limitante de esta investigación es la falta de mecanismos para mitigar la vulnerabilidad una vez detectada.

En el artículo científico titulado BACFuzz: Exposing the Silence on Broken Access Control Vulnerabilities in Web Applications, Dharmaadi et al. (2025) presentan una herramienta llamada BACFuzz cuyo enfoque se orienta a la identificación de vulnerabilidades de tipo BOLA y BFLA debido a la dificultad que representa la identificación automática de este tipo de ataques. Los autores realizaron las pruebas en 20 aplicaciones web reales y evaluaron las vulnerabilidades en 15 casos de CVE. La herramienta utiliza el modelo de lenguaje largo (LLM) para identificar parámetros importantes relacionados con la autorización como el ID de usuario, roles, tokens e identificadores de recursos, además las consultas basadas en SQL , el cual es el Lenguaje de Consulta Estructurado estandarizado (ISO/IEC 9075) utilizado para la definición, manipulación y control de acceso a datos dentro de sistemas de gestión de bases de datos relacionales, determinaron si usuarios sin permisos obtuvieron algún acceso no autorizado. El limitante de la investigación es la ausencia de pruebas en arquitecturas de microservicios.

En el artículo científico titulado Automated Testing of Broken Authentication Vulnerabilities in Web APIs with AuthREST, Corradini et al. (2025) crearon una herramienta de código abierto llamada AuthREST que evalúa automáticamente vulnerabilidades de tipo Broken Authentication en APIs web. Para ello, los autores evaluaron escenarios asociados a ataques de fuerza bruta, credential stuffing y validación incorrecta de tokens. Los resultados fueron errores en la implementación de mecanismos de autenticación en el cual la seguridad de los servicios se vio expuesta. El limitante en la investigación es la falta de un esquema integrado que combine autenticación y autorización segura para APIs RESTful.

Para Mousavi et al. (2023), en este artículo científico realizaron una revisión enfocada en detectar errores frecuentes en la implementación de APIs de seguridad dentro de aplicaciones

modernas. La investigación se enfoca en analizar cómo las configuraciones incorrectas, validaciones inadecuadas y el uso indebido de mecanismos criptográficos pueden introducir vulnerabilidades críticas en los sistemas. Al respecto, el artículo evidenció que la mayoría de los desarrolladores tienen problemas al implementar APIs seguras. Esto se debe principalmente a la falta de conocimiento y a que se percibe como una alta complejidad técnica, lo cual da como resultado fallas que generan vulnerabilidades relacionadas con la autenticación, autorización y protección de datos. El limitante que existe es que la investigación realizada no profundiza en la implementación de esquemas integrales de autenticación y autorización segura aplicados específicamente a APIs RESTful.

2.1.2. Uso de criptografía asimétrica en una arquitectura de microservicios.

2.1.2.1. Algoritmo criptográfico RS256.

En los mecanismos de autenticación basados en JWT, se utiliza el algoritmo RS256, el cual se define en el estándar RFC 7518 como un esquema de firma digital que combina el criptosistema asimétrico RSA con la función de hash seguro SHA-256 para la generación de digestos criptográficos. Jones (2015). Para firmar digitalmente los tokens mediante cifrado asimétrico. El servidor genera el token con una clave privada y verifica su autenticidad con una clave pública, garantizando que el contenido no haya sido alterado. Este enfoque refuerza la seguridad del proceso de autenticación y reduce las vulnerabilidades asociadas a la manipulación de la firma o a la verificación incorrecta del token. Artículo científico ligado a un estándar técnico. Mahfud et al. (2025)

2.1.3. Enfoque basado en zero trust.

Según la investigación titulada NIST Special Publication 800-207 (Zero Trust Architecture) Borchert et al. (2020) en el cuál es un estándar técnico. Los modelos Zero Trust se basan en el principio “never trust, always verify” (nunca confiar, siempre verificar); según este principio, ninguna solicitud debe considerarse automáticamente fiable. En este proyecto, dicho principio se aplica mediante el uso de JWT, ya que cada solicitud realizada a las API requiere la verificación continua del

token, la verificación criptográfica de su firma digital mediante RS256 y la verificación de los atributos de autorización, tales como roles y período de validez. De este modo, el sistema evita confiar implícitamente en el usuario autenticado y exige una verificación continua de la identidad y las autorizaciones antes de conceder el acceso a los recursos protegidos.

Para Borchert (2020). En la sección stolen credentials/insider threat de este estándar técnico. La implementación adecuada de políticas de zero trust, de seguridad de la información y de resiliencia, así como de las mejores prácticas, reduce el riesgo de que un atacante obtenga un acceso amplio mediante credenciales robadas o a través de un ataque interno. El principio de ZT de no otorgar confianza implícita basada en la ubicación de red implica que los atacantes deben comprometer una cuenta o un dispositivo existente para lograr establecer un punto de apoyo.

En la investigación titulada Enterprise with Public- or Customer-Facing Services, Borchert et al. (2020) donde es un estándar técnico. Explican que los principios de Zero Trust no se aplican de la misma manera a los recursos totalmente públicos que no requieren autenticación, como las páginas web públicas. Sin embargo, cuando los servicios están dirigidos a usuarios registrados o clientes, las organizaciones pueden establecer políticas de control de acceso basadas en la identidad. En el contexto de este proyecto, dado que el acceso a los recursos requiere una autenticación previa y la validación continua de un token de acceso, este enfoque es directamente relevante para las API protegidas por OAuth 2.0 y JWT. Mientras gestiona el proceso de autorización de OAuth 2.0, JWT transporta de forma segura la identidad y los privilegios del usuario autenticado. De este modo, cada solicitud enviada a las API se verifica antes de que se conceda el acceso a los recursos protegidos, y se aplican los principios de Zero Trust, que se centran en la autenticación y la autorización continua.

Para el escenario experimental que se desarrollara se implementa la filosofía de Zero Trust no como un componente aislado, sino como el modelo de arquitectura que rige el control de acceso. Bajo este enfoque, Keycloak opera como el Servidor de Autorización centralizado que, mediante el

framework OAuth 2.0, asigna atributos y privilegios específicos a los usuarios a través de claims criptográficos incrustados en tokens JWT. Estos atributos serán evaluados de forma explícita y obligatoria en cada petición HTTP por el API Gateway (Kong) y los middlewares de las APIs, determinando dinámicamente la autorización sobre los endpoints de negocio según el principio de menor privilegio.

2.1.4. Modelo OWASP API security project.

En la actualidad, la evaluación de la seguridad de las interfaces de programación de aplicaciones (API) se basa esencialmente en el modelo OWASP Top 10 para la seguridad de las API. Diversas investigaciones y análisis del sector coinciden en que el paso de la versión de 2019 a la actualización de 2023 ofrece un enfoque más preciso de los riesgos asociados a las arquitecturas modernas basadas en microservicios y API RESTful (Griffeth, 2025; Salt Security, 2024) siendo un estándar técnico respaldado por fuentes complementarias. Esta actualización incluye amenazas relacionadas con la autenticación insegura, controles de autorización débiles y configuraciones de seguridad incorrectas, que afectan directamente a los mecanismos modernos de autenticación y autorización implementados a través de OAuth 2.0 y JWT.

Según la actualización OWASP API Security Top 10 (2023), una de las vulnerabilidades más críticas y persistentes desde la versión de 2019 es BOLA, clasificada como API1:2023. Esta vulnerabilidad se produce cuando una API no valida correctamente si un usuario autenticado tiene permisos suficientes para acceder a un recurso específico, lo que permite el acceso no autorizado a información confidencial (OWASP, 2023). Sin embargo, OWASP API Security 2023 destaca no solo la relevancia de BOLA, sino también otras amenazas críticas asociadas con los mecanismos modernos de autenticación y autorización en las API RESTful, como Broken Authentication, Broken Function BFLA y Security Misconfiguration.

Estas vulnerabilidades afectan directamente a las arquitecturas basadas en OAuth 2.0, JSON Web Token (JWT) y API Gateway, ya que comprometen los procesos relacionados con la validación de identidad, la comprobación de privilegios, la verificación de firmas digitales y las configuraciones de acceso seguro. En este proyecto, se utilizará JWT para transportar de forma segura la identidad y los privilegios del usuario autenticado, mientras que OAuth 2.0 gestionará el proceso de autorización para los recursos protegidos. Asimismo, el API Gateway actuará como componente central de seguridad, responsable de validar tokens, verificar firmas criptográficas, comprobar autorizaciones y aplicar políticas de acceso antes de permitir la comunicación con las API internas.

Por lo tanto, las vulnerabilidades BOLA, BFLA, Autenticación Incompleta y Configuración Incorrecta de Seguridad representan elementos clave en el análisis y la evaluación de seguridad desarrollados en esta investigación. OWASP (2023). Teniendo en cuenta que OWASP es un estándar técnico.

2.1.5. Virtualización y contenedores en entornos de ciberseguridad.

En el ámbito académico y profesional, herramientas como VirtualBox. Oracle Corporation (2024) y VMware Workstation. VMware (2024). Siguen siendo ampliamente utilizadas debido a su flexibilidad para simular redes complejas y entornos distribuidos. Investigaciones recientes señalan que la virtualización sigue siendo preferida en laboratorios de ciberseguridad cuando se requiere aislamiento fuerte y control total del sistema operativo. Respaldado por el artículo científico de (Singh & Chana, 2021)

Por otro lado; según el artículo científico, escrito por Sharma et al. (2021). la adopción de contenedores ha crecido significativamente con tecnologías como Docker, que permiten desplegar aplicaciones de forma ligera y eficiente. Aunque dentro del alcance de este proyecto no se aplicaran configuraciones de red a nivel de Dockers es importante mencionar que diversos estudios recientes

han evidenciado limitaciones en términos de seguridad, particularmente relacionadas con el aislamiento a nivel de kernel.

Para Alqahtani et al. (2022) en dicho artículo científico. Destacan que los contenedores pueden ser más vulnerables a ataques de escape de contenedor si no se configuran adecuadamente. Asimismo, investigaciones más recientes indican que, aunque Docker mejora la portabilidad y escalabilidad, introduce nuevos vectores de ataque asociados a la gestión de imágenes y permisos.

2.1.6. Comparación entre Máquinas Virtuales y Contenedores.

Mientras que las máquinas virtuales operan con sistemas operativos independientes, los contenedores comparten el kernel del host, lo que puede representar un riesgo en escenarios de seguridad. Basado en artículos científicos por (Pahl, 2015) (Zhang, 2020).

Estudios actuales muestran que las máquinas virtuales ofrecen mayor seguridad en entornos donde se ejecutan pruebas ofensivas controladas. Por otro lado, los contenedores son más eficientes, aunque requieren configuraciones adicionales para alcanzar niveles adecuados de seguridad; por lo tanto, la elección entre ambas tecnologías depende estrictamente del contexto de uso, balanceando el aislamiento frente a la eficiencia (Alqahtani et al., 2022).

En este proyecto se utiliza Docker como plataforma de contenedores para aplicar y evaluar las APIs desarrolladas. Se ha elegido este entorno porque se necesita un entorno aislado, reproducible y fácilmente escalable para realizar pruebas de seguridad. El uso de Docker no solo permite implementar servicios relacionados con los procesos de autenticación y autorización, sino que también permite realizar pruebas de validación de tokens JWT, control de acceso y simulaciones de vulnerabilidades de seguridad relacionadas con BOLA, BFLA, Broken Authentication y Security Misconfiguration sin afectar al sistema principal ni a otros entornos operativos.

El uso de contenedores permite analizar de forma controlada la implementación de estándares como OAuth 2.0 (Hardt, 2012) y JSON Web Token Jones et al. (2015), evaluando tanto

configuraciones seguras como inseguras. Además, facilita la replicabilidad del experimento, un aspecto clave en investigaciones académicas recientes (Singh & Chana, 2021).

2.2. Marco Teórico

2.2.1. Fundamentos criptográficos para la validación de JWT.

El cifrado es un componente fundamental de los mecanismos modernos de autenticación y autorización utilizados en las APIs. Los conceptos más importantes en el contexto de los JSON Web Tokens (JWT) son las funciones hash, el cifrado asimétrico y las firmas digitales. Las funciones hash, como SHA-256, generan un resumen único de los datos, lo que facilita la detección de modificaciones no autorizadas. Por otro lado, el cifrado asimétrico utiliza un par de claves compuesto por una clave privada y una clave pública; SHA-256 se utiliza para firmar digitalmente el token, mientras que el cifrado asimétrico se utiliza para verificar la autenticidad del token. Este proceso garantiza la integridad de la información contenida en el JWT; así, es posible verificar si el token fue emitido por una entidad legítima y si fue manipulado durante la transmisión. Estos principios son fundamentales para comprender cómo funcionan algoritmos como SHA-256, comúnmente utilizados para la verificación de tokens en arquitecturas basadas en OAuth 2.0 y API RESTful. (Torres, 2024)

2.2.2.Tipos de criptografías.

Para este estudio se considerarán la criptografía simétrica y la criptografía asimétrica (Torres, 2024). En la criptografía simétrica se utiliza una única clave para cifrar y descifrar la información; por lo tanto, el remitente como el destinatario deben conocer esta clave y protegerla de posibles exfiltraciones. Por el contrario, el cifrado asimétrico utiliza dos claves matemáticamente relacionadas, es decir, una clave pública y una clave privada. La clave pública puede distribuirse libremente, mientras que la privada debe ser gestionada exclusivamente por su propietario. Este modelo permite realizar operaciones de cifrado, autenticación y firma digital sin tener que revelar la clave privada a terceros. (Torres, 2024).

En el contexto de los JSON Web Token (JWT), el cifrado asimétrico se utiliza mediante algoritmos como RS256. En este sistema, la clave privada sirve para la firma digital del token, mientras que la clave pública permite verificar la autenticidad e integridad del token. De este modo, el destinatario puede asegurarse de que el JWT ha sido emitido por una fuente legítima y de que su contenido no ha sido modificado durante la transmisión. (Torres, 2024).

2.2.3. Algoritmos de cifrado.

El proyecto se enfoca en los algoritmos HS256 y RS256 mismos que son fundamentales, ya que forman parte de las pruebas de seguridad relacionadas con la autenticación mediante JSON Web Tokens (JWT). HS256 utiliza una clave compartida para generar y verificar la firma de los tokens, mientras que RS256 emplea cifrado asimétrico, es decir, utiliza una clave privada para firmar y una clave pública para verificar. Esta distinción es especialmente importante en la arquitectura de microservicios y en las API RESTful, ya que la separación de claves permite un control de seguridad más estricto.

Las pruebas desarrolladas en este estudio se centran en verificar si el sistema solo acepta certificados firmados con el algoritmo autorizado (RS256) y rechaza aquellos que hayan sido alterados o que intenten utilizar algoritmos no válidos, como HS256 o “none”. Para ello, el API Gateway actúa como punto central de autenticación y, antes de conceder acceso a los recursos protegidos, verifica la integridad de la firma, el algoritmo especificado en el encabezado JWT y la autenticidad del certificado. (Mahindrakar y Pujeri, 2020)

2.2.4. JSON Web Token (JWT).

Según el estándar oficial RFC 7519, el JSON Web Token (JWT) se define oficialmente como un mecanismo compacto y seguro, adecuado para entornos URL, que sirve para representar y transmitir afirmaciones (claims) entre dos partes. En esta especificación, las afirmaciones se estructuran como un objeto JSON, que puede actuar como datos útiles de una estructura JSON

Web Signature (JWS) o como texto sin cifrar de una estructura JSON Web Encryption (JWE). Esta arquitectura estandarizada garantiza la protección de la integridad digital de la información transmitida mediante el uso de códigos de autenticación de mensajes (MAC) o firmas digitales asimétricas, y se ha consolidado como un pilar fundamental para la autenticación, la autorización y el intercambio seguro de datos en microservicios y API RESTful. (Mahindrakar y Pujeri, 2020)

Componentes de JWT:

Los siguientes componentes serán modificados y evaluados dinámicamente a lo largo de las pruebas mencionadas en la introducción:

- **Encabezado (Header):** El primer componente de un token web JSON es el encabezado. Contiene información como el algoritmo, el tipo de token, etc. El tipo de algoritmo varía desde HS256 simétrico hasta RS256 asimétrico, según el uso. Este conjunto de datos se codifica en formato Base64 y se coloca como primer componente del token JWT. (Mahindrakar y Pujeri, 2020)
- **Datos / Claims (Payload):** Consiste en información como el ID del cliente, el ID de la empresa, los derechos de acceso del token y la fecha de caducidad del token. Este paquete también se codifica en formato Base64 y se coloca como segundo componente del token JWT. (Mahindrakar y Pujeri, 2020)
- **Firma (Signature):** Tanto el encabezado como la carga útil están vinculados mediante una clave secreta. Esta firma mantiene la integridad del token. (Mahindrakar y Pujeri, 2020)

Las declaraciones registradas se definen en el estándar JWT de la IETF (RFC 7519). No son obligatorias, pero se recomiendan ampliamente para promover la interoperabilidad. Estas declaraciones se abrevian convencionalmente a tres letras para mayor brevedad. Jones et al. (2015). Algunos ejemplos comunes son:

- **sub (subject):** Identifica al principal que es el sujeto del JWT

- **iat (issued at):** Hora de emisión del JWT; puede utilizarse para determinar la antigüedad del JWT.
- **exp (expiration time):** Identifica la hora de caducidad a partir de la cual el JWT no debe aceptarse para su procesamiento.

El control de acceso basado en roles (RBAC) es un modelo para autorizar el acceso de los usuarios finales a sistemas, aplicaciones y datos según el rol predefinido de cada usuario. (Lindemulder y Kosinski, s.f.)

En este proyecto, el claim (rol) se utiliza para almacenar el nivel de permiso del usuario dentro de la carga útil del JWT en un formato (sin estado). Se definen dos roles principales: admin y customer. El rol customer solo tiene acceso a funciones básicas, como ver y gestionar los recursos asociados a su propia cuenta, mientras que el rol admin cuenta con permisos adicionales para realizar operaciones administrativas, como la visualización masiva de productos y la eliminación de registros del sistema.

La autorización se verifica mediante comprobaciones de autenticación implementadas en el backend y la pasarela de API, que confirman que el valor de la reclamación role coincide con los permisos requeridos por cada punto final. Además, se desarrollarán pruebas para detectar vulnerabilidades de autorización a nivel de función rota (BFLA). Estas pruebas intentarán cambiar el valor de la reclamación role en el JWT de customer a admin. El sistema debe rechazar estas solicitudes si detecta que la firma digital del token ha sido manipulada o que el usuario carece de los permisos necesarios para acceder a funciones sensibles.

2.2.5.Tipos de arquitectura API.

La arquitectura de la API proporciona una estructura que define los protocolos para la comunicación entre sistemas y las normas de interoperabilidad. Este marco no solo especifica el tipo de información que puede intercambiarse con terceros, sino que también define los mecanismos

técnicos y los procedimientos operativos permitidos, sentando así las bases para la gestión de datos y la interoperabilidad (Joshi, 2024)

- **Representational State Transfer (REST)**

Se define como un estilo arquitectónico concebido para un intercambio de datos eficiente y con un bajo consumo de recursos en sistemas distribuidos, y constituye el estándar predominante para la comunicación entre microservicios. Su eficiencia radica en la optimización del procesamiento de grandes volúmenes de datos y en la implementación de mecanismos como las funciones de almacenamiento en caché, que reducen los tiempos de latencia y la carga del servidor. El pilar fundamental de REST es su ausencia de estado; dado que el contexto del cliente no se almacena entre las solicitudes, el servidor procesa cada solicitud de forma independiente. Aunque esta característica requiere una configuración rigurosa de los mecanismos de autenticación para cada interacción normalmente mediante tokens, ofrece una arquitectura más escalable y facilita la implementación de conceptos de seguridad robustos y desacoplados. (IBM, 2024).

La naturaleza *stateless* de REST está directamente relacionada con el uso de los JSON Web Token (JWT) como mecanismo de autenticación. Dado que el servidor no almacena información relativa a la sesión entre una solicitud y otra, cada solicitud debe contener toda la información necesaria para la identificación y autorización del usuario. En este contexto, el JWT actúa como un contenedor de información autónomo que integra directamente en el propio token atributos como la identidad del usuario, su rol y sus autorizaciones. De este modo, cada solicitud enviada a la API contiene un JWT correspondiente, lo que permite al servidor verificar su firma digital, confirmar su periodo de validez y comparar los permisos correspondientes sin tener que recurrir constantemente a una memoria de sesión. Esta característica contribuye a mejorar la escalabilidad de las API RESTful y simplifica la implementación de mecanismos de seguridad y autorización basados en OAuth 2.0 y JWT.

2.2.6.Contenedores y Docker.

Los contenedores representan una evolución de la virtualización tradicional orientada a la eficiencia y portabilidad de aplicaciones. A diferencia de las máquinas virtuales, los contenedores no requieren un sistema operativo completo para cada instancia, sino que comparten el kernel del sistema anfitrión, reduciendo significativamente el consumo de recursos (Merkel, 2014).

Docker es actualmente una de las plataformas de contenedores más utilizadas debido a su facilidad de despliegue, portabilidad y compatibilidad con arquitecturas modernas basadas en microservicios. Docker permite encapsular aplicaciones y dependencias dentro de imágenes reutilizables, facilitando la implementación consistente de servicios en distintos entornos.

En ciberseguridad, Docker se utiliza ampliamente para desplegar laboratorios controlados, aplicaciones vulnerables y entornos de prueba aislados. Su capacidad para crear redes internas entre contenedores permite simular arquitecturas distribuidas similares a las utilizadas en sistemas reales.

En el marco de este proyecto, los distintos componentes de la arquitectura de pruebas se encapsulan en contenedores mediante Docker. Esto incluye la API de evaluación de vulnerabilidades y medidas de protección, el API Gateway (Kong) encargada de la validación de la aplicación de las políticas de seguridad y el Servidor de autenticación (Keycloack) centralizada de credenciales, así como los JSON que almacenan la información necesaria para cada servicio. Esta arquitectura implementa los distintos componentes a través de la red interna de Docker de manera que estén aislados entre sí, pero interconectados, lo que facilita la realización de pruebas relacionadas con la autenticación, la autorización y la verificación de certificados JWT. Además, el entorno en contenedores permite la reproducción controlada de escenarios relacionados con Broken Authentication, BOLA, BFLA y Security Misconfiguration.

CAPÍTULO 3

3. Desarrollo

Este capítulo se centra en la arquitectura descrita en los capítulos anteriores. En él se explica la técnica del desarrollo de la arquitectura, implementación de controles y ejecución de pruebas de seguridad. Este capítulo presenta una visión general en un entorno alojado en Docker, la autenticación y autorización basadas en los estándares OAuth 2.0 y JSON Web Tokens (JWT) son los puntos de partida para la investigación y el análisis. Dado que el enfoque principal de la investigación del OWASP API Security Top 10 se centra en vulnerabilidades graves, el objetivo principal es evaluar estas vulnerabilidades que se mencionaron a lo largo de esta investigación. Por lo tanto, se presentan cuatro escenarios específicos de evaluación. Cada uno de ellos ha sido diseñado para demostrar cómo se explota una vulnerabilidad particular del *OWASP API Security Top 10*, y cómo se logra su mitigación aplicando una estrategia de defensa por capas:

- **Escenario 1:** *Manipulación de identificadores (Vulnerabilidad BOLA).* Consiste en pruebas de acceso no autorizado a datos, donde se evalúa si un usuario puede ver información que no le pertenece al modificar los identificadores lógicos en las rutas de la API.
- **Escenario 2:** *Inyección de pases de acceso inválidos (Vulnerabilidad Broken Authentication).* Corresponde a la ejecución de pruebas dinámicas (como la prueba SB401) donde se envían tokens malformados, vacíos o expirados para verificar si el API Gateway bloquea correctamente los accesos en la frontera del sistema.
- **Escenario 3:** *Escalamiento vertical de privilegios (Vulnerabilidad BFLA).* Se enfoca en evaluar el control de roles lógicos, midiendo la capacidad del sistema para generar respuestas de error HTTP 403 cuando un usuario con permisos limitados intenta ejecutar funciones administrativas.

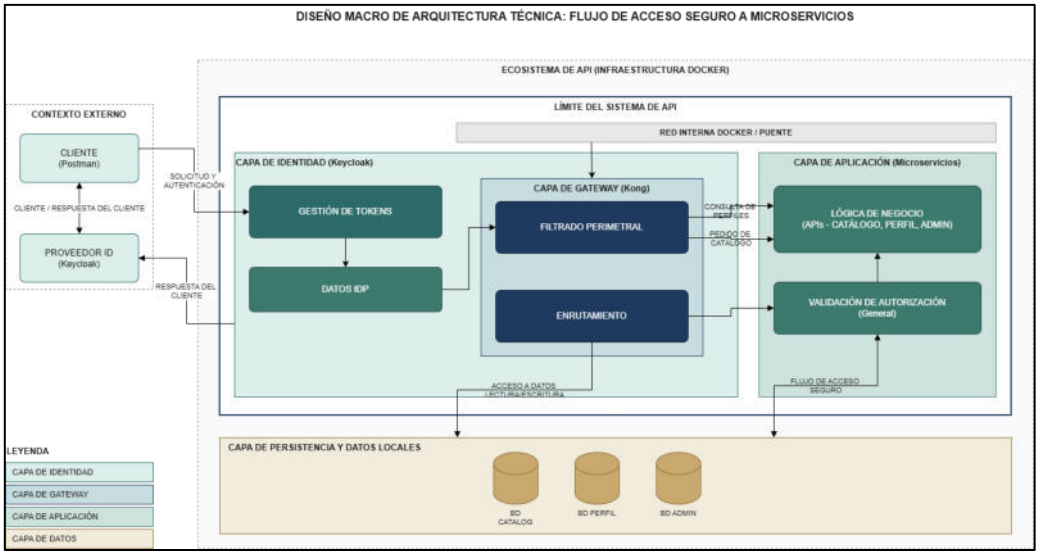
- **Escenario 4:** *Evasión perimetral y abuso de recursos (Vulnerabilidad Security Misconfiguration).* Comprende las pruebas de acceso directo a los puertos de los microservicios (*Gateway Bypass*) y las ráfagas automatizadas de tráfico masivo (Prueba SB429), validando la efectividad del cortafuegos (iptables) y las políticas de limitación de tasa (*Rate Limiting*).

3.1. Levantamiento de la arquitectura

3.1.1. Diagrama de la arquitectura.

Figura 1.

Arquitectura integral del ecosistema de microservicios basado en el modelo de defensa por capas.



Nota. Arquitectura local en Docker que ilustra la defensa en profundidad mediante Keycloak, Kong Gateway, tres microservicios con middlewares de seguridad y persistencia aislada. Elaboración propia.

3.1.2. Estructura del árbol de directorios y archivos

Se ha diseñado una estructura de directorios uniforme para las APIs con el objetivo de mantener un estándar de archivos en el ecosistema y garantizar su replicabilidad mediante Docker. A continuación, se detalla la función de cada uno de sus componentes:

Tabla 1.

Estructura de los directorios (APIs)

Componente / Ruta	Tipo	Propósito en el Entorno Experimental
config/	Directorio	Aloja scripts como db.js para inicializar el acceso o lectura de los recursos locales.
data/	Directorio	Contiene archivos estructurados (.json) que emulan el mecanismo de persistencia sin requerir bases de datos relacionales externas.
routes/	Directorio	Define los puntos de acceso (endpoints) expuestos y es donde se inyectan secuencialmente los middlewares de seguridad.
Dockerfile	Archivo	Define las instrucciones de empaquetamiento, dependencias de la imagen base (Alpine) y asignación de usuarios sin privilegios. La configuración del dockerfile se puede visualizar en el github. **Apéndices**
package.json	Archivo	Registra los metadatos del servicio y las dependencias criptográficas requeridas (ej. jsonwebtoken, jwks-rsa).
package-lock.json	Archivo	Garantiza el bloqueo de las versiones exactas de las librerías instaladas, asegurando la reproducibilidad del entorno.
server.js	Archivo	Inicializa el servidor web Express, procesa el middleware global de lectura JSON y monta el enrutador correspondiente.

Nota. Descripción de los componentes que conforman la estructura de directorios de los microservicios. Elaboración propia.

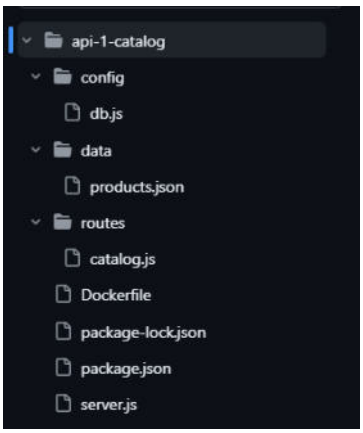
La organización de los componentes descrita en la *Tabla 1* no responde únicamente a una disposición técnica de archivos y directorios, sino a una decisión arquitectónica definida de acuerdo con las necesidades metodológicas de la investigación. Esta estructura fue diseñada para cumplir con tres requerimientos fundamentales del proyecto, garantizando la modularidad de los servicios, la reproducibilidad del entorno experimental y la adecuada ejecución de los escenarios de evaluación de seguridad:

- **Separación de responsabilidades:** La arquitectura implementa un desacoplamiento estricto de componentes para evitar el acoplamiento difuso. El punto de entrada (server.js) delega de forma exclusiva el enrutamiento y la lógica de control a la carpeta routes/, lugar donde se inyectan secuencialmente los *middlewares* de seguridad bajo un esquema de filtrado interceptor. A su vez, los mecanismos de configuración de recursos (config/) y la persistencia simulada de datos estáticos (data/) operan de manera independiente. Esta segregación de funciones garantiza que las fallas lógicas o de seguridad queden confinadas a un módulo específico, simplificando el aislamiento de errores y el mantenimiento del código fuente.

- **Replicabilidad del entorno:** Con el objetivo de cumplir el principio científico de reproducibilidad experimental, la infraestructura se apoya en el archivo de empaquetamiento package-lock.json y en las directivas de automatización del Dockerfile. El primero mitiga el riesgo de deriva de dependencias de software al fijar de forma determinista las versiones exactas de las librerías criptográficas utilizadas (jsonwebtoken, jwks-rsa). El segundo encapsula el microservicio completo sobre una imagen base ligera e inmutable (Alpine Linux), lo que permite instanciar el mismo entorno idéntico y predecible en cualquier nodo físico o virtual, anulando inconsistencias por variables del sistema operativo anfitrión.

- **Facilitación de pruebas de seguridad:** La administración de las pruebas de seguridad se facilita gracias al aislamiento por capas con Docker. Este enfoque permite definir todas las dependencias necesarias en archivos de configuración específicos que se descargan automáticamente al iniciar el sistema. De esta manera, es posible ejecutar el ambiente seguro y el inseguro de forma simultánea, lo que permite comparar ambos escenarios en tiempo real e identificar el impacto exacto que tienen las configuraciones de seguridad sobre el rendimiento y el funcionamiento de los servicios.

Figura 2. *Estructura de las carpetas API*



Nota. Estructura general de directorios de los microservicios. Elaboración propia.

3.1.3.Creación de las APIs

3.1.3.1. Levantamiento de la arquitectura insegura.

API 1 (Catalog). Este microservicio es el encargado de gestionar y exponer los datos públicos dentro de la plataforma. Para su desarrollo se ha utilizado Node.js en conjunto con el framework Express. Dentro del entorno de pruebas, este componente fue configurado deliberadamente con la vulnerabilidad “API8:2023 - Security Misconfiguration (Configuración Incorrecta de Seguridad)” del marco OWASP API Security Top 10. Este escenario constituye un riesgo de seguridad debido a que conservar las configuraciones por defecto, sin aplicar medidas de protección adicionales, convierte al

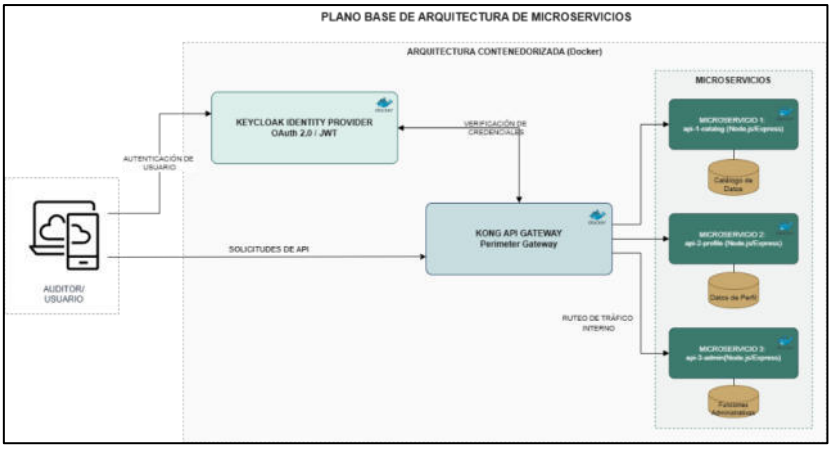
componente en un blanco fácil para posibles ataques. De esta forma, se permite evaluar el comportamiento del sistema ante la ausencia de controles de mitigación perimetrales.

API 2 (Profile). Este microservicio está encargado de simular la consulta de los pedidos realizados por un usuario específico dentro de la plataforma. Para el entorno experimental, este componente fue configurado intencionalmente con la vulnerabilidad “API1:2023 – BOLA” y “API2:2023 - Broken Authentication”. El objetivo de este diseño es permitir la manipulación de los identificadores de recursos en los puntos de acceso, lo que facilita la ejecución de pruebas de seguridad y el análisis del comportamiento del sistema frente a solicitudes de acceso no autorizado.

API 3 (Admin). Este microservicio es el responsable de gestionar las funcionalidades de carácter administrativo dentro de la plataforma. El componente fue desarrollado para evaluar la robustez de los controles de autorización y las políticas de configuración segura. Su implementación permite analizar el impacto de la ausencia o deficiencia en los mecanismos de control de acceso basados en roles (RBAC), recreando escenarios asociados a la vulnerabilidad «API5:2023 - Autorización Rota a Nivel de Función (BFLA)», donde un usuario común intenta ejecutar funciones restringidas a perfiles con mayores privilegios. Asimismo, el servicio incorpora fallos orientados a la categoría «API8:2023 - Configuración Incorrecta de Seguridad», con el fin de examinar los efectos de la exposición de información sensible y la aplicación deficiente de controles de seguridad en el entorno.

Figura 3.

Plano de la arquitectura de microservicios distribuida y control de acceso centralizado con configuraciones por defecto.



Nota. El diagrama ilustra el flujo de interacción y el desacoplamiento estructural de la Prueba de Concepto (PdC) implementada con configuraciones por defecto.

El usuario realiza la autenticación inicial ante el proveedor de identidad (Keycloak) bajo el estándar OAuth 2.0 para obtener un token criptográfico estructurado (JWT). Posteriormente, las solicitudes dirigidas a la infraestructura son interceptadas de forma centralizada por el API Gateway (Kong) en el perímetro, el cual actúa como proxy inverso resolviendo el ruteo lógico del tráfico interno hacia los tres microservicios independientes (api-1-catalog, api-2-profile y api-3-admin) dentro del entorno de red virtualizado de Docker.

3.1.3.2. Levantamiento de la arquitectura segura.

API 1 (Catalog). Este microservicio representa la versión robustecida encargada de la gestión y exposición de datos públicos. En esta fase, la mitigación de la vulnerabilidad *API8:2023 – (Security Misconfiguration)* configurada deliberadamente mediante el uso de parámetros y valores por defecto del componente para evaluar el sistema ante la ausencia de protecciones perimetrales. Por un lado, se implementa el API Gateway Kong como único punto de entrada, el cual oculta los puertos internos de los contenedores y aplica políticas globales de limitación de tráfico (Rate Limiting). Esto evita la saturación del servicio y previene ataques de denegación de servicio (DoS). Por otro lado, a nivel de aplicación se inyectan middlewares de cabeceras de seguridad (como

Helmet) encargados de deshabilitar la divulgación del entorno tecnológico (X-Powered-By) y prevenir el escaneo automatizado de la infraestructura.

API 2 (Profile). Este microservicio incorpora mecanismos de autenticación y autorización diseñados para reducir las vulnerabilidades de "autenticación defectuosa" y "autorización a nivel de objeto defectuosa" (BOLA). En esta API se ha implementado un proceso de verificación criptográfica de JSON Web Tokens (JWT), que permite comprobar la integridad de la firma digital, la validez del token y la autenticidad de la información contenida en las reclamaciones. Gracias a esta verificación, solo las solicitudes enviadas con un identificador válido obtienen acceso a los recursos protegidos del sistema.

Además, para evitar el acceso no autorizado, se ha implementado un mecanismo de autorización basado en la comparación del identificador autenticado con el recurso solicitado. De este modo, el identificador de usuario contenido en el token se compara con el identificador asociado al recurso solicitado, y solo se autorizan aquellas operaciones que corresponden al legítimo propietario de la información. Esto evita el acceso no autorizado a la información mediante la falsificación del identificador en la solicitud.

API 3 (Admin). La versión segura de este microservicio cuenta con mecanismos de seguridad para mitigar las vulnerabilidades BFLA y Security Misconfiguration. Para ello, se aplica un esquema de control de acceso basado en roles (RBAC) que gestiona los permisos de los usuarios autenticados en función de la información contenida en el token JWT. De esta manera, solo los usuarios con los permisos necesarios pueden acceder a funciones de administración o sensibles, y se bloquean los intentos de escalada de privilegios.

Además, este servicio cuenta con funciones de control para gestionar de forma segura los errores y las excepciones, y evita la filtración de información confidencial relacionada con la infraestructura interna de la aplicación. En lugar de revelar detalles técnicos, registros internos o

rastros de ejecución, el sistema ofrece respuestas controladas adecuadas para el entorno de producción. Estas medidas ayudan a reducir la superficie de exposición de la aplicación y refuerzan su postura de seguridad frente a configuraciones de protección incorrectas o insuficientes.

A continuación, se indica un resumen de los fallos activos en el entorno inseguro y los mecanismos técnicos específicos implementados para su mitigación en la arquitectura segura.

Tabla 2.

Resumen de vulnerabilidades evaluadas y componentes técnicos de mitigación por capas.

Vulnerabilidad OWASP	**Componente Afectado**	**Capa / Mecanismo de Mitigación**	**Comportamiento (Ambiente Inseguro)**	**Configuración Mitigante (Ambiente Seguro)**
API1:2023 - BOLA *(Autorización Rota a Nivel de Objeto)*	API 2 (Profile)	Middleware de la Aplicación (Validación de *Claims* de identidad)	Permite modificar los identificadores (IDs) directamente en la URI para consultar datos de otros usuarios sin restricción.	Inyección de lógicas en Express.js que extraen el ID del usuario desde el JWT y validan la propiedad del recurso antes de entregar los datos.
API2:2023 - Broken Authentication *(Autenticación Rota)*	API 2 (Profile)	Proveedor de Identidad (IdP) y API Gateway	Acepta solicitudes sin validación de identidad rigurosa (tokens vacíos, malformados, firmas corruptas o	Centralización de identidad con Keycloak bajo OAuth 2.0 y PKCE. Validación asimétrica de

			ataques de tipo *alg: none*).	firmas criptográficas en los JWT y control de expiración.
API5:2023 - BFLA ***(Autorización Rota a Nivel de Función)***	API 3 (Admin)	Middleware de la Aplicación (Control de Roles RBAC)	Control de acceso basado en roles deficiente, permitiendo a usuarios comunes ejecutar funciones exclusivas de administradores.	Filtros en las rutas de Express.js que leen los roles del JWT e interceptan la petición con un error HTTP 403 si el usuario no tiene el rango requerido.
API8:2023 - Security Misconfiguration ***(Configuración Incorrecta)***	API 1 (Catalog) API 3 (Admin)	Perímetro (API Gateway - Rate Limiting)	Uso de parámetros por defecto, exposición abierta de puertos, mensajes de error detallados (*verbose*) y evasión perimetral (*Gateway Bypass*)	Políticas de *Rate Limiting* en Kong y configuración de puertos frente a consultas directas sin pasar con Kong.

Nota. Se presenta un resumen en el cual compara el comportamiento de las Apis ante los vectores de ataque del OWASP API Security Top 10. Presentando cómo el entorno inseguro procesa las solicitudes anómalas por la falta de controles y cómo la arquitectura segura las neutraliza usando políticas.

3.2. Arquitectura general del entorno

Se diseñó una arquitectura distribuida basada en contenedores Docker, con el propósito de simular un entorno de APIs moderno donde se puedan evaluar mecanismos de autenticación y autorización basados en OAuth 2.0 y JSON Web Token (JWT). La infraestructura busca reproducir una arquitectura de microservicios con separación lógica de funciones, control de acceso centralizado y comunicación segura entre componentes.

La arquitectura está compuesta por 4 nodos principales:

- Máquina atacante (VM1)
- API Gateway (VM2)
- Servicio de proveedor de identidad (VM3)
- Backend de Microservicios (VM4)

Tabla 3.

Especificaciones técnicas de la infraestructura implementada

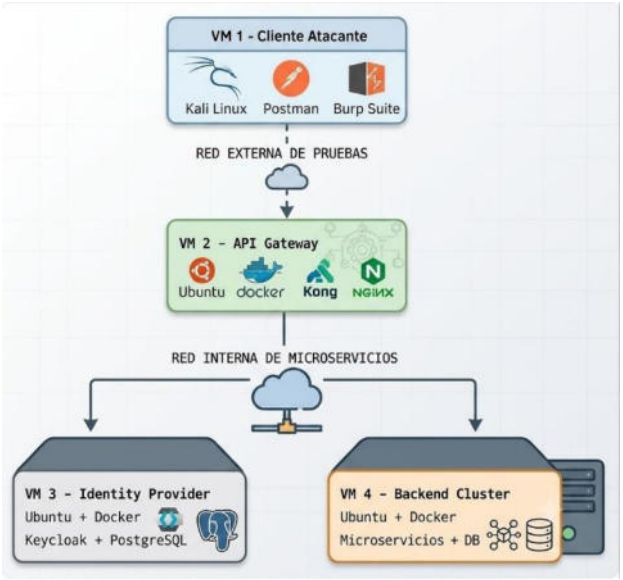
VM	Dirección IP	Rol	Sistema Operativo	Recursos	Servicios principales	Puertos expuestos	Contenedores Docker
VM1	192.168.100.122	Cliente atacante	Kali Linux	2 vCPU, 4 GB RAM, 40 GB SSD	Reconocimiento, pruebas de penetración y validación de APIs	22	No aplica
VM2	192.168.100.169	API Gateway	Ubuntu Server 26.04	2 vCPU, 4 GB RAM, 40 GB SSD	Kong Gateway, NGINX	8000, 8001, 8443, 8444	Kong
VM3	192.168.100.121	Identity Provider	Ubuntu Server 26.04	2 vCPU, 4 GB RAM, 40 GB SSD	Keycloak, PostgreSQL	8080, 8443, 5.432	Keycloak 24.0, PostgreSQL 15
VM4	192.168.100.182	Backend de microservicios	Ubuntu Server 26.04	4 vCPU, 4 GB RAM, 60 GB SSD	APIs protegidas y almacenamiento de datos	3001, 3002, 3003	api-1-catalog, api-2-profile, api-3-admin

Fuente: Elaboración propia.

La segmentación de funciones en máquinas virtuales independientes permitió reproducir una arquitectura similar a la utilizada en entornos corporativos, donde los componentes encargados del acceso, autenticación, gestión de identidades y lógica de negocio operan de forma desacoplada. Esta separación facilita la evaluación de controles de seguridad específicos, el análisis del flujo OAuth 2.0/JWT y la comparación entre escenarios con y sin mecanismos de protección, garantizando además la reproducibilidad del laboratorio propuesto.

La Figura 3 representa la arquitectura implementada en un entorno on-premise, donde se evidencia la segmentación de redes, el flujo de comunicación entre componentes y la separación de responsabilidades entre capas, además de las tecnologías utilizadas en cada máquina para realizar su despliegue.

Figura 4. *Arquitectura de laboratorio para evaluación de seguridad en APIs basadas en OAuth 2.0 y JWT.*



Nota. Arquitectura del laboratorio compuesta por cuatro máquinas virtuales para la evaluación de mecanismos de autenticación y autorización basados en OAuth 2.0 y JWT. **Fuente:** Elaboración propia

3.2.1. **Descripción de la arquitectura**

El diagrama de la Figura 3 representa una arquitectura de microservicios distribuida en 4 máquinas virtuales, organizada en tres capas principales:

- Capa externa: representada por la VM 1, se utiliza como cliente de pruebas para el consumo de las API's y simulador de un atacante externo
- Capa de exposición: representada por la VM 2, es donde se encuentra el API Gateway encargado de centralizar, filtrar y enrutar todas las solicitudes.
- Capa interna de servicios: representada por la VM 3 y VM 4, es la capa donde se tiene expuestos los microservicios y el proveedor de identidad.
    - VM 3: encargada de la autenticación y gestión de identidades mediante keycloak.
    - VM 4: encargada de la lógica de negocio mediante microservicios desplegados en contenedores Docker.

El flujo de comunicación inicia en el cliente, pasa por el API Gateway, que valida la autenticación con el proveedor de identidad y finalmente enruta las solicitudes hacia los microservicios del backend.

3.2.2. Infraestructura de virtualización

La infraestructura desplegada para las capas 2 y 3 se implementó sobre máquinas virtuales con sistema operativo Ubuntu Server 26.04, todas con soporte para Docker. Cada VM cumple un rol específico dentro del ecosistema distribuido, permitiendo una separación clara de responsabilidades y facilitando la escalabilidad del sistema.

3.2.3. Despliegue del API Gateway

El API Gateway (VM 2) actúa como punto único de entrada al sistema. Se implementó utilizando la tecnología Kong, desplegada en un contenedor Docker.

Las características de la máquina virtual desplegada son las siguientes:

- **Sistema Operativo:** Ubuntu Server 26.04
- **Recursos:**
    - 2 vCPU
    - 4 GB RAM
    - 30 GB almacenamiento
- **Servicio:**
    - Kong

Sus funciones principales incluyen:

- Enrutamiento de solicitudes hacia microservicios internos.
- Validación de tokens JWT emitidos por Keycloak.
- Aplicaciones de políticas de seguridad (rate limiting, control de acceso).

Figura 5. *Despliegue y verificación operativa de Kong API Gateway.*

Nota. Inicialización de la infraestructura y verificación del estado de ejecución del API Gateway en Docker.

Ejecución del archivo docker-compose.yml utilizado para desplegar Kong API Gateway en un contenedor Docker. La evidencia muestra la creación y puesta en marcha exitosa del servicio, así como la verificación de su estado mediante el comando docker ps, confirmando que el contenedor se encuentra operativo y escuchando en los puertos configurados para la administración y publicación de APIs. Este componente actúa como punto central de acceso, aplicando políticas de autenticación, autorización y enrutamiento hacia los microservicios protegidos.

3.2.4. Despliegue del Identity Provider

En la VM 3 se implementó Keycloak junto con PostgreSQL en contenedores Docker. Como parte de la arquitectura propuesta, se implementó un proveedor de identidad basado en Keycloak desplegado sobre contenedores Docker, utilizando PostgreSQL como base de datos para el almacenamiento de configuraciones, usuarios, roles y sesiones. Este componente constituye el núcleo del sistema de autenticación y autorización, siendo responsable de la emisión y validación de tokens utilizados por los servicios protegidos.

Las características de la máquina virtual desplegada son las siguientes:

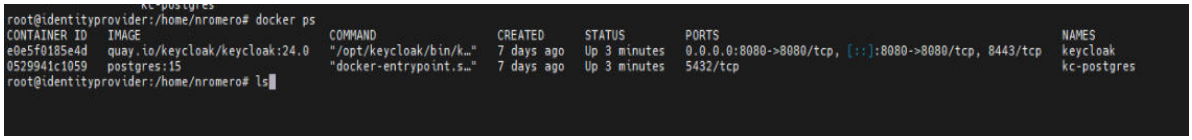
- **Sistema Operativo:** Ubuntu Server 26.04
- **Recursos:**
    - 2 vCPU
    - 4 GB RAM
    - 40 GB almacenamiento
- **Servicio:**
    - Keycloak
    - PostgreSQL

La implementación se realizó mediante contenedores Docker independientes para cada servicio, facilitando la administración, portabilidad y reproducibilidad del entorno de pruebas.

Este componente gestiona:

- Autenticación de usuarios.
- Autorización basada en roles.
- Emisión de tokens OAuth2 / OpenID Connect.
- Integración con el API Gateway para validación de identidad.

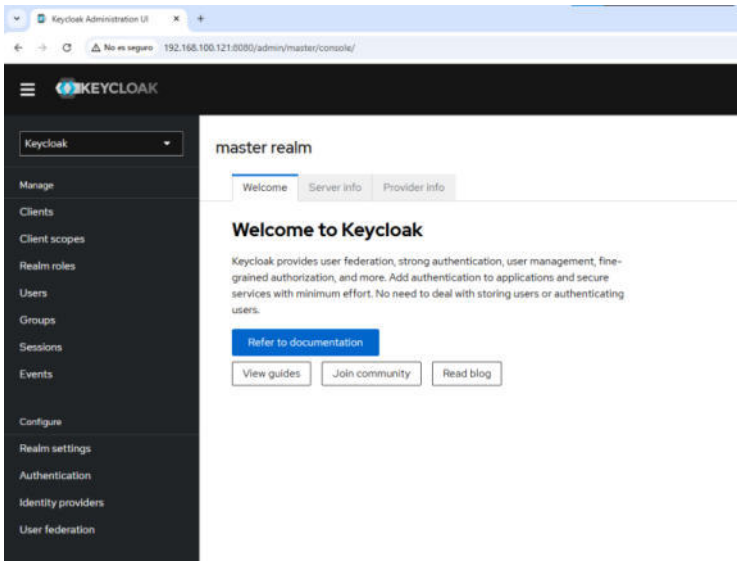
Figura 6. *Despliegue del proveedor de identidad Keycloak y base de datos PostgreSQL en contenedores Docker.*



Nota. Despliegue y verificación del estado operativo del servidor de identidades Keycloak y su base de datos persistente en Docker.

La interfaz administrativa de Keycloak permitió la configuración de clientes, usuarios, roles y flujos de autenticación necesarios para la ejecución de las pruebas definidas en el proyecto. Asimismo, esta plataforma servirá como punto central para evaluar mecanismos de seguridad relacionados con la emisión, validación, expiración y revocación de tokens dentro de la arquitectura propuesta.

Figura 7. *Interfaz de administración del proveedor de identidad Keycloak.*



Nota. Interfaz gráfica de administración de Keycloak.

A través de esta interfaz se administran usuarios, roles, clientes OAuth 2.0/OpenID Connect, sesiones activas y políticas de autenticación. Asimismo, permite configurar los mecanismos de

autorización, la emisión de tokens JWT y la integración con los microservicios protegidos mediante el API Gateway.

3.2.5. Despliegue del Backend de microservicios

El backend de la arquitectura fue implementado mediante un conjunto de microservicios independientes desplegados en contenedores Docker. Cada servicio fue diseñado para gestionar una funcionalidad específica dentro del entorno de pruebas, siguiendo los principios de desacoplamiento y separación de responsabilidades característicos de las arquitecturas basadas en microservicios.

Las características de la máquina virtual desplegada son las siguientes:

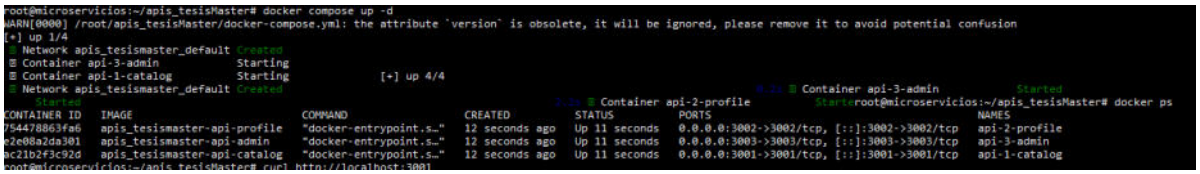
- **Sistema Operativo:** Ubuntu Server 26.04
- **Recursos:**
    - 4 vCPU
    - 4 GB RAM
    - 60 GB almacenamiento
- **Servicio:**
    - Api-1-catalog
    - Api-2-profile
    - Api-3-admin

Cada microservicio se ejecuta en un contenedor independiente, permitiendo una administración modular y facilitando la simulación de escenarios distribuidos. La comunicación entre servicios se realiza mediante redes internas de Docker, mientras que el acceso externo se encuentra restringido y controlado a través del API Gateway.

- Cada microservicio incluye:
    - Lógica de negocio específica.
    - Configuración independiente.

- Comunicación interna mediante red Docker.
- Exposición únicamente a través del API Gateway.

Figura 8. *Despliegue y verificación de los microservicios mediante Docker Compose.*



Nota. Orquestación, despliegue simultáneo y mapeo de puertos del ecosistema de microservicios mediante Docker Compose.

La figura 7 muestra la creación de la red interna Docker y la ejecución satisfactoria de los contenedores api-1-catalog, api-2-profile y api-3-admin. Asimismo, se observa la asignación de los puertos 3001, 3002 y 3003 respectivamente, utilizados para la exposición controlada de los servicios dentro del entorno de pruebas.

3.3. Pruebas ofensivas a las Apis desarrolladas

3.3.1. Escenario de ataque sin controles de seguridad

Con el propósito de evaluar las debilidades presentes en una configuración insegura de autenticación y autorización, se ejecutarán pruebas controladas sobre las APIs desplegadas en el entorno de laboratorio. La evaluación seguirá una metodología de pentesting adaptada al contexto del proyecto, enfocada en la identificación y validación de fallos asociados al uso de OAuth 2.0 y JSON Web Tokens (JWT).

Las pruebas de explotación de vulnerabilidades de las APIS expuestas se realiza mediante una metodología de pentesting real, donde se incluyen las siguientes etapas:

- **Identificación o reconocimiento:** se recopiló información sobre la superficie de ataque expuesta, identificando direcciones IP, servicios disponibles y tecnologías utilizadas por las APIs. Para esta fase se emplearon herramientas como navegador web, cURL y

Postman. Como evidencias se obtuvieron rutas accesibles, mensajes de respuesta del servidor y documentación Swagger expuesta.

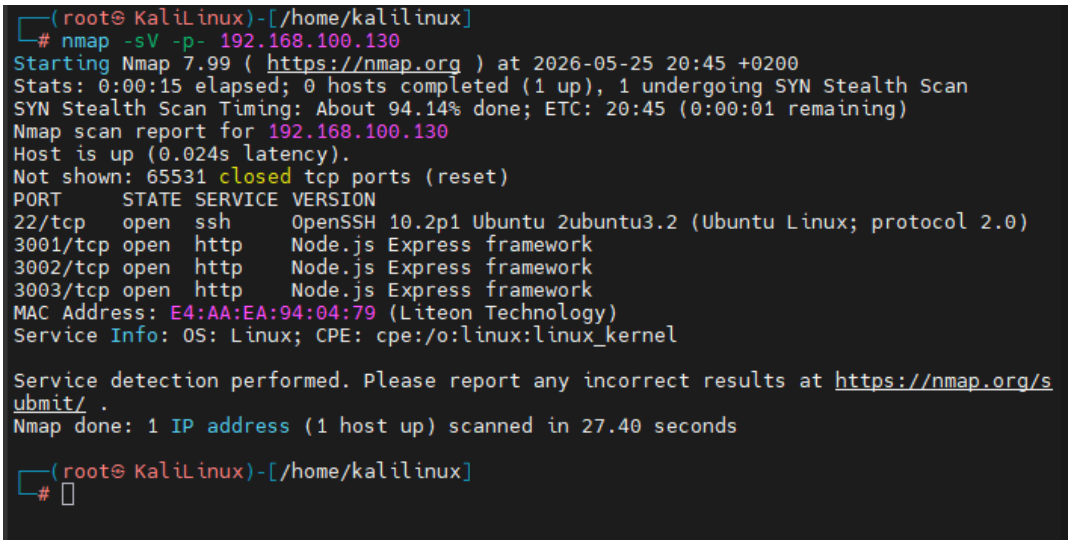
- **Escaneo:** se realizó la enumeración de puertos y servicios activos con el objetivo de identificar componentes accesibles desde la red. Se utilizó la herramienta Nmap para detectar servicios HTTP y SSH, así como las versiones de las tecnologías implementadas. Las evidencias consideradas incluyeron puertos abiertos, banners de servicios y la identificación de frameworks utilizados por las APIs.
- **Evaluación de vulnerabilidades:** se analizaron los recursos descubiertos para determinar la ausencia de mecanismos de autenticación y autorización. Se revisó el comportamiento de los endpoints frente a solicitudes sin credenciales, la exposición de documentación técnica y la posibilidad de interactuar con recursos identificados mediante parámetros. Como criterio de evaluación se consideró vulnerable cualquier endpoint que procesara solicitudes sin validar la identidad o los privilegios del usuario.
- **Explotación:** se ejecutaron pruebas dirigidas a confirmar la existencia de vulnerabilidades mediante el acceso a recursos sin autenticación, la manipulación de identificadores y el intento de acceso a funcionalidades administrativas. Se utilizaron cURL, navegador web, Postman y Burp Suite para generar y modificar solicitudes HTTP. Las evidencias obtenidas incluyeron respuestas HTTP 200 OK, mensajes de error, información sensible expuesta y la confirmación de escenarios asociados a vulnerabilidades BOLA y BFLA.
- **Informe:** se documentaron los hallazgos identificados, describiendo la vulnerabilidad detectada, el procedimiento realizado, las evidencias obtenidas y el impacto potencial sobre la confidencialidad, integridad y disponibilidad de la información. Asimismo, los resultados sirvieron como línea base para compararlos posteriormente con el escenario protegido mediante OAuth 2.0, JWT, Keycloak y Kong API Gateway.

3.3.1.1. ATAQUE 1: Acceso sin autenticación

3.3.1.1.1. Fase de reconocimiento.

La primera actividad realizada consistió en identificar los servicios expuestos dentro del entorno de pruebas. Mediante técnicas de reconocimiento se detectaron múltiples APIs REST accesibles a través de diferentes puertos, lo que permitió determinar los puntos de entrada disponibles para la evaluación de los mecanismos de autenticación implementados.

Figura 9. *Resultados obtenidos durante el proceso de reconocimiento mediante Nmap.*



Nota. Fase de reconocimiento técnico y descubrimiento de puertos y servicios expuestos mediante el uso de Nmap desde el nodo auditor.

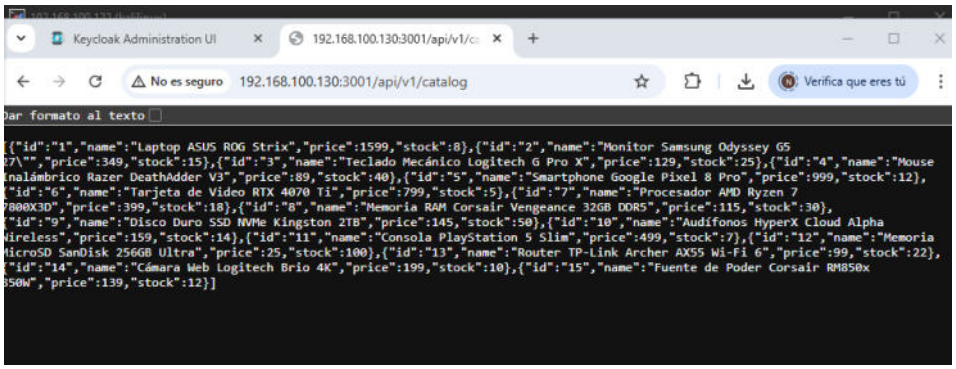
Se identifican los puertos 3001, 3002 y 3003 asociados a los microservicios desplegados mediante Node.js y Express, además del servicio SSH utilizado para la administración del servidor. La información obtenida permitió determinar los puntos de entrada que serían evaluados durante las pruebas de autenticación y control de acceso.

Posteriormente, se realizó una exploración manual de los servicios identificados durante la fase de reconocimiento. Para ello, se accedió a los endpoints expuestos mediante un navegador web y herramientas de prueba de APIs, con el objetivo de verificar la accesibilidad de los recursos y determinar si existían controles de autenticación previos al acceso.

Durante esta actividad se identificaron las siguientes rutas asociadas a los microservicios desplegados:

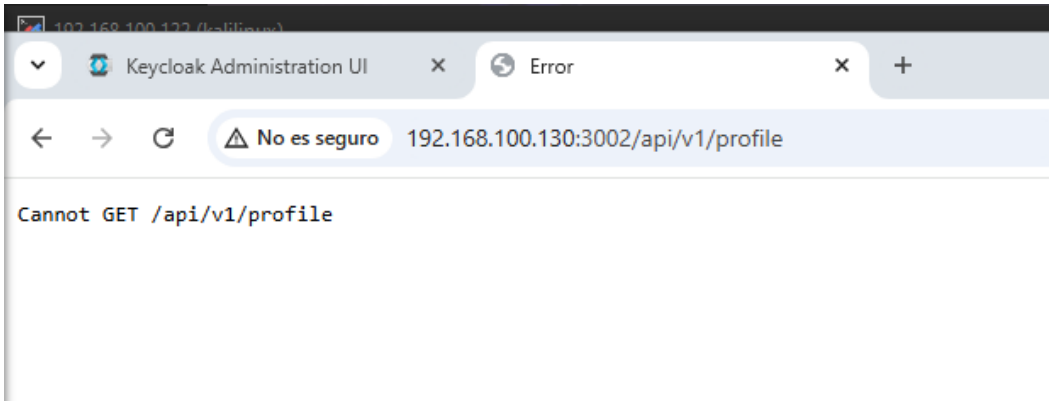
- /api/v1/catalog
- /api/v1/profile
- /api/v1/admin

Figura 10. *Exposición de información a través de la API de catálogo sin mecanismos de autenticación.*



Nota. Consumo directo y exposición pública del servicio de catálogo sin intermediación perimetral de seguridad.

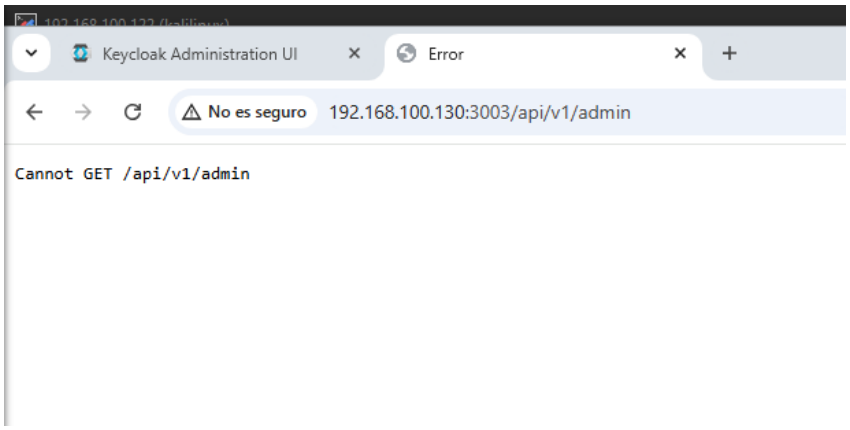
Figura 11. *Identificación de rutas no válidas durante la fase de reconocimiento de la API Profile.*



Nota. Fallo en el acceso directo al microservicio de perfil y validación de la restricción del tráfico perimetral no autorizado.

Intento de acceso al endpoint /api/v1/profile mediante navegador web durante la fase de reconocimiento. La aplicación responde con el mensaje "Cannot GET /api/v1/profile", indicando que la ruta consultada no corresponde a un recurso válido o no se encuentra implementada para el método HTTP utilizado.

Figura 12. *Enumeración inicial de la API administrativa durante la fase de reconocimiento.*



Nota. Restricción de acceso directo al microservicio administrativo y validación del aislamiento lógico perimetral.

Las Figuras 11 y 12 evidencian que los endpoints identificados responden a las solicitudes realizadas, confirmando que los servicios se encuentran activos y operativos. Sin embargo, las rutas evaluadas no proporcionan acceso directo a información sensible ni a funcionalidades protegidas.

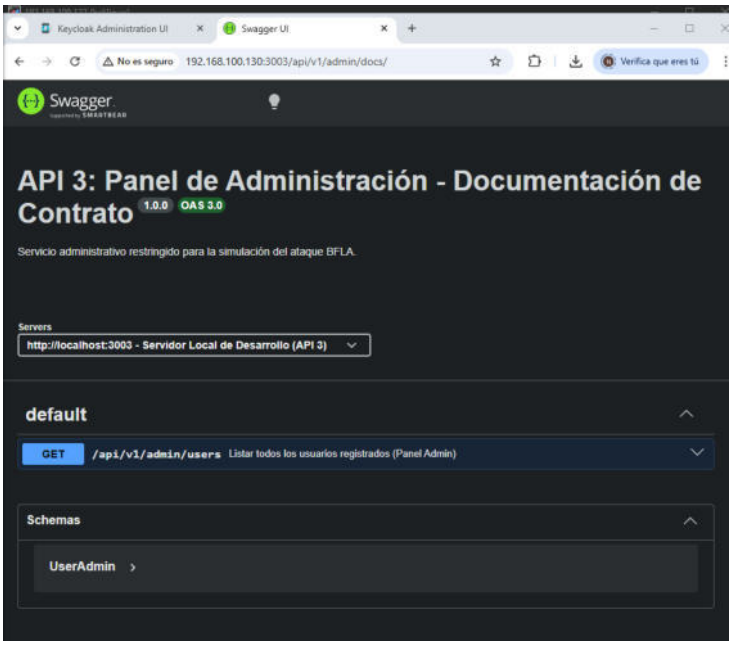
Durante la fase de enumeración se identificó la presencia de interfaces de documentación Swagger expuestas públicamente en los tres microservicios, tal como se muestra en la Figura 13. Este tipo de interfaces proporciona información detallada sobre los endpoints disponibles, métodos HTTP soportados, parámetros de entrada y respuestas esperadas de la API.

Los endpoints identificados fueron los siguientes:

- **/api/v1/catalog/docs**

- **/api/v1/profile/docs**
- **/api/v1/admin/docs**

Figura 13. *La interfaz Swagger detectada durante el proceso de reconocimiento.*



Nota. Interfaz gráfica de Swagger UI para la documentación técnica y especificación de contrato de la API 3 (Administrativa).

La información obtenida permitió disponer de una visión completa de los servicios expuestos y de los recursos disponibles sin necesidad de autenticación previa. Esto facilitó la identificación de los mecanismos de protección implementados y la planificación de las pruebas posteriores orientadas a evaluar los controles de autenticación y autorización de las APIs.

3.3.1.1.2. Identificación de brechas de seguridad.

Durante la evaluación de los controles de acceso se identificó que los microservicios expuestos no requerían mecanismos de autenticación para acceder a determinados recursos. Esta condición fue validada mediante múltiples solicitudes HTTP realizadas directamente a los endpoints disponibles, sin incluir encabezados de autorización ni credenciales de usuario.

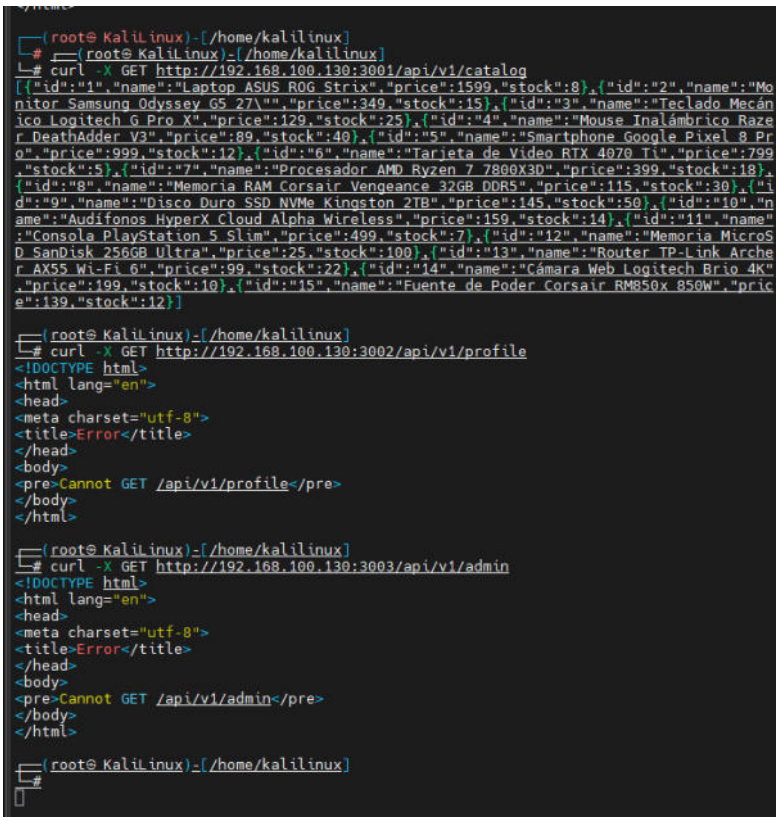
Las respuestas obtenidas confirmaron que las APIs procesaban las solicitudes y devolvían información sin verificar la identidad del solicitante, evidenciando la ausencia de controles efectivos de autenticación en la capa de aplicación.

3.3.1.1.3. Fase de explotación.

Como se observa en la Figura 14, el servidor procesó la solicitud y devolvió correctamente la información solicitada, permitiendo el acceso a los recursos expuestos sin realizar ninguna validación de identidad. El código de respuesta obtenido confirmó que la petición fue aceptada y atendida satisfactoriamente por el servicio.

Este comportamiento evidencia la ausencia de controles de autenticación en el endpoint evaluado, permitiendo que cualquier usuario con acceso a la API pueda consultar información sin demostrar previamente su identidad.

Figura 14. *Validación de endpoints expuestos mediante solicitudes HTTP directas.*

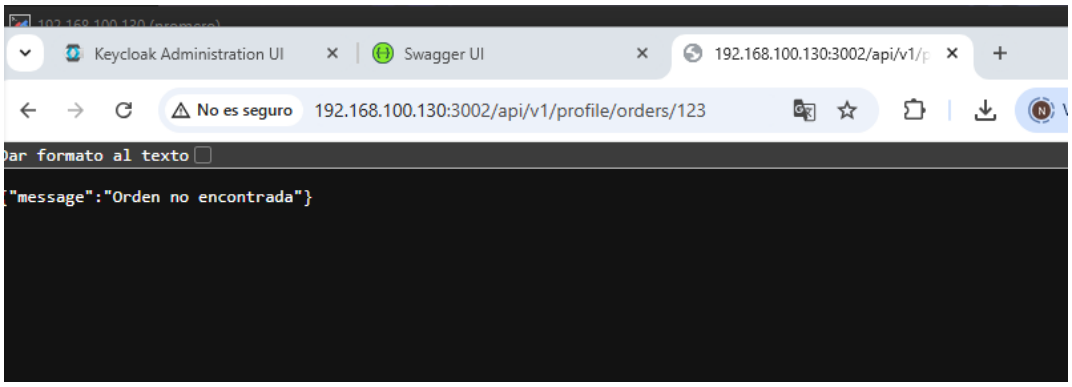


Nota. Conectividad y solicitudes automatizadas mediante el uso de cURL desde el nodo auditor hacia las interfaces de los microservicios.

La evidencia obtenida confirma que al menos uno de los microservicios expone información sin realizar validaciones de identidad, permitiendo el acceso a recursos mediante solicitudes anónimas. Esta condición constituye una debilidad de seguridad que puede facilitar la exposición no autorizada de información y será utilizada como escenario base para comparar posteriormente la implementación de controles de autenticación y autorización mediante OAuth 2.0 y JSON Web Tokens (JWT).

La Figura 15 muestra que el endpoint responde a la solicitud realizada sin requerir credenciales ni un token JWT válido. Aunque el recurso consultado no existe y el sistema devuelve el mensaje “Orden no encontrada”, la respuesta evidencia que la petición fue procesada sin aplicar mecanismos de autenticación previos.

Figura 15. *Identificación de un endpoint accesible sin autenticación durante la evaluación de la API Profile.*



Nota. Respuesta estructurada del endpoint de perfiles ante la consulta directa de identificadores de recursos para la evaluación de BOLA.

Esta condición confirma la presencia de una vulnerabilidad BOLA, ya que permite a un atacante acceder potencialmente a información perteneciente a otros usuarios, comprometiendo gravemente la confidencialidad y seguridad del sistema.

La Figura 16 evidencia que el endpoint administrativo /api/v1/admin/users responde a solicitudes realizadas sin autenticación previa, devolviendo información correspondiente a usuarios registrados en el sistema, incluyendo identificadores, nombres de usuario, correos electrónicos y roles asignados.

Figura 16. *Exposición de información administrativa sin autenticación.*

Nota. Exposición de registros de identidad y fuga de información sensible por ausencia de control de acceso a nivel de función.

La respuesta obtenida demuestra la ausencia de mecanismos de autenticación y control de acceso sobre un recurso de carácter administrativo. Como resultado, un usuario no autenticado puede acceder a información sensible que debería estar restringida exclusivamente a personal autorizado.

3.3.2. Matriz comparativa entre el escenario vulnerable y el escenario protegido

Con el fin de evidenciar el aporte del modelo propuesto, la Tabla 3 presenta una comparación entre las debilidades identificadas durante las pruebas ofensivas realizadas sobre el entorno sin controles de seguridad y los mecanismos implementados posteriormente para su mitigación. Esta comparación permite visualizar la relación directa entre cada vulnerabilidad detectada, el control aplicado y el resultado esperado dentro del escenario seguro.

Tabla 4.

Comparación entre escenarios sin controles y con controles de seguridad

Vulnerabilidad identificada	**Evidencia en escenario vulnerable**	**Control implementado**	**Escenario protegido**	**Resultado obtenido**
Acceso sin autenticación a recursos	Consumo de /api/v1/catalog sin token JWT	Validación obligatoria de JWT mediante Kong y middleware de autenticación	Solicitudes sin token son rechazadas	HTTP 401 Unauthorized
Exposición de documentación técnica	Interfaces Swagger accesibles públicamente	Restricción del acceso a recursos expuestos y centralización mediante API Gateway	Solo usuarios autorizados acceden a servicios protegidos	Reducción de la superficie de ataque
BOLA (Broken Object Level Authorization)	Acceso potencial a recursos identificados mediante manipulación de IDs	Validación de propiedad del recurso utilizando el claim sub del JWT	El usuario únicamente accede a sus propios recursos	HTTP 403 Forbidden ante recursos ajenos
BFLA (Broken Function Level Authorization)	Acceso a funciones administrativas	Implementación de RBAC basado en	Solo usuarios con rol admin acceden a	HTTP 403 para usuarios sin privilegios y HTTP

	sin validación de privilegios	realm_access.roles	funciones críticas	200 para administradores
Manipulación de JWT	Alteración de claims o uso de configuraciones inseguras	Validación de firma digital, emisor, audiencia y JWKS	Tokens modificados o inválidos son rechazados	Acceso denegado
Uso de tokens expirados	Aceptación potencial de credenciales fuera de vigencia	Verificación del claim exp	Tokens expirados no son procesados	HTTP 401 Unauthorized

Fuente: Elaboración propia

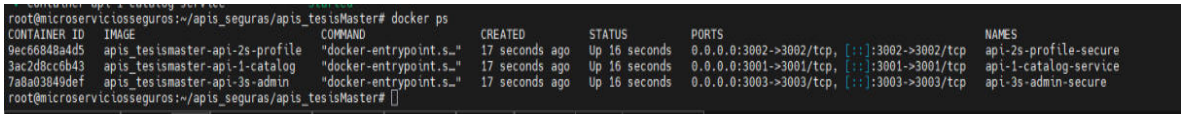
Los resultados comparativos evidencian que la incorporación de mecanismos de autenticación y autorización basados en OAuth 2.0, JWT, Keycloak y Kong API Gateway permitió eliminar las debilidades identificadas en el escenario inicial. En consecuencia, el entorno protegido no solo valida la identidad del usuario, sino que también verifica la integridad del token, los privilegios asociados y la propiedad de los recursos solicitados, fortaleciendo significativamente la seguridad de las APIs evaluadas.

3.3.3. Escenario con controles de seguridad

3.3.2.1 Despliegue de api's seguras

Una vez identificadas las debilidades presentes en el escenario inicial, se procedió a desplegar una nueva versión de los microservicios incorporando mecanismos de autenticación y autorización basados en OAuth 2.0 y JSON Web Tokens (JWT). El objetivo de esta implementación fue restringir el acceso a los recursos expuestos y garantizar que únicamente usuarios o sistemas autorizados puedan consumir los servicios.

Figura 17. *Despliegue de los microservicios protegidos mediante Docker.*



Nota. Verificación del estado operativo y levantamiento del clúster de microservicios en el entorno seguro y robustecido de Docker.

Se observa la ejecución de los contenedores correspondientes a las APIs seguras (api-1-catalog-secure, api-2-profile-secure y api-3-admin-secure), los cuales incorporan mecanismos de autenticación y autorización basados en OAuth 2.0 y JWT para proteger el acceso a los recursos expuestos.

La arquitectura segura fue desplegada mediante contenedores Docker independientes para cada microservicio, manteniendo la separación lógica de responsabilidades y facilitando la administración del entorno. Como se observa en la Figura 17, los servicios API-1-Catalog, API-2-Profile y API-3-Admin se encuentran operativos y ejecutándose correctamente dentro de la infraestructura definida para el proyecto.

A diferencia del escenario vulnerable, esta versión incorpora validaciones de autenticación mediante tokens JWT emitidos por el proveedor de identidad, así como controles de autorización sobre los recursos protegidos. De esta manera, cada solicitud debe ser validada antes de permitir el acceso a la información o funcionalidades disponibles en las APIs.

La correcta ejecución de los contenedores confirma la disponibilidad de la plataforma sobre la cual se realizarán las pruebas de seguridad orientadas a verificar la efectividad de los controles implementados y su capacidad para mitigar los riesgos identificados en el escenario sin protección.

3.3.2.2. Esquema de autenticación segura

Con el fin de mitigar las vulnerabilidades identificadas en el escenario inicial, se implementó una arquitectura de autenticación y autorización basada en OAuth 2.0 y JSON Web Tokens (JWT),

incorporando un proveedor de identidad centralizado y un API Gateway como punto de control de acceso.

Figura 18. *Esquema de autenticación y autorización implementado en la arquitectura segura.*

Nota. Diagrama de secuencia lógica para el flujo de autenticación y autorización centralizado mediante OAuth 2.0 y JSON Web Tokens.

Como se observa en la Figura 18, el proceso de autenticación inicia cuando el cliente solicita un token de acceso al servidor de identidad Keycloak. Una vez que las credenciales son validadas correctamente, Keycloak emite un token JWT firmado digitalmente que contiene la información necesaria para identificar al usuario y definir sus permisos de acceso.

El cliente incluye el token obtenido en el encabezado de autorización de cada solicitud dirigida a las APIs protegidas. Antes de reenviar la petición al backend, el API Gateway Kong verifica la validez del token, comprobando aspectos como la firma digital, el emisor, la audiencia y la vigencia temporal. Solo aquellas solicitudes que superan satisfactoriamente estas validaciones son enviadas a los microservicios correspondientes.

3.3.2.3. Configuración de proveedor de identidad (keycloak)

En esta primera etapa se debe crear el realm correspondiente al servicio y definir un cliente asociado a las APIs. Para el presente proyecto se creó el cliente en función a las siguientes configuraciones:

- Realm: TESIS CIBERSEGURIDAD
- Tipo de Cliente: OpenID Connect
- Cliente ID: apis-backend
- Autenticación de cliente: ON
- Flujo de Autenticación:
  - Estándar Flow: ON
  - Direct Access grants: ON

Como parte de la implementación de los controles de autenticación y autorización, se configuró Keycloak como proveedor centralizado de identidad dentro de la arquitectura. Este componente es responsable de autenticar a los usuarios, emitir los tokens de acceso y proporcionar la información necesaria para que las APIs puedan validar la identidad y los permisos asociados a cada solicitud.

La primera actividad consistió en la creación de un Realm denominado TESIS CIBERSEGURIDAD, el cual actúa como un dominio independiente de autenticación y administración de usuarios. Dentro de este Realm se registró un cliente encargado de representar a las APIs protegidas del entorno de pruebas.

Las configuraciones aplicadas al cliente fueron las siguientes:

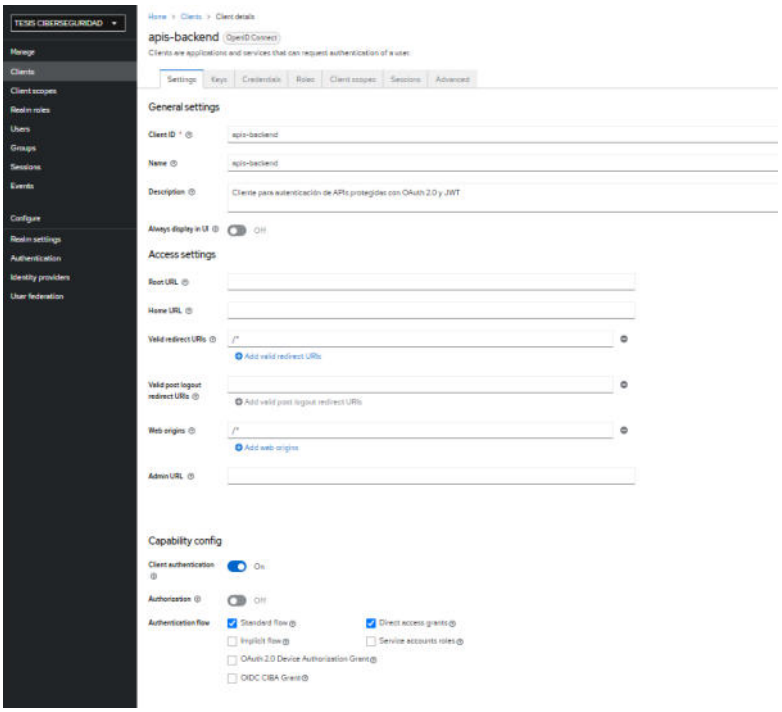
Tabla 5.

Configuraciones de keyloack

Parámetro	Configuración
Realm	**TESIS CIBERSEGURIDAD**
Tipo de cliente	OpenID Connect
Client ID	apis-backend
Client Authentication	Habilitado
Standard Flow	Habilitado
Direct Access Grants	Habilitado

Nota. Parámetros de configuración del cliente en el servidor de identidades Keycloak. **Fuente:** Elaboración propia.

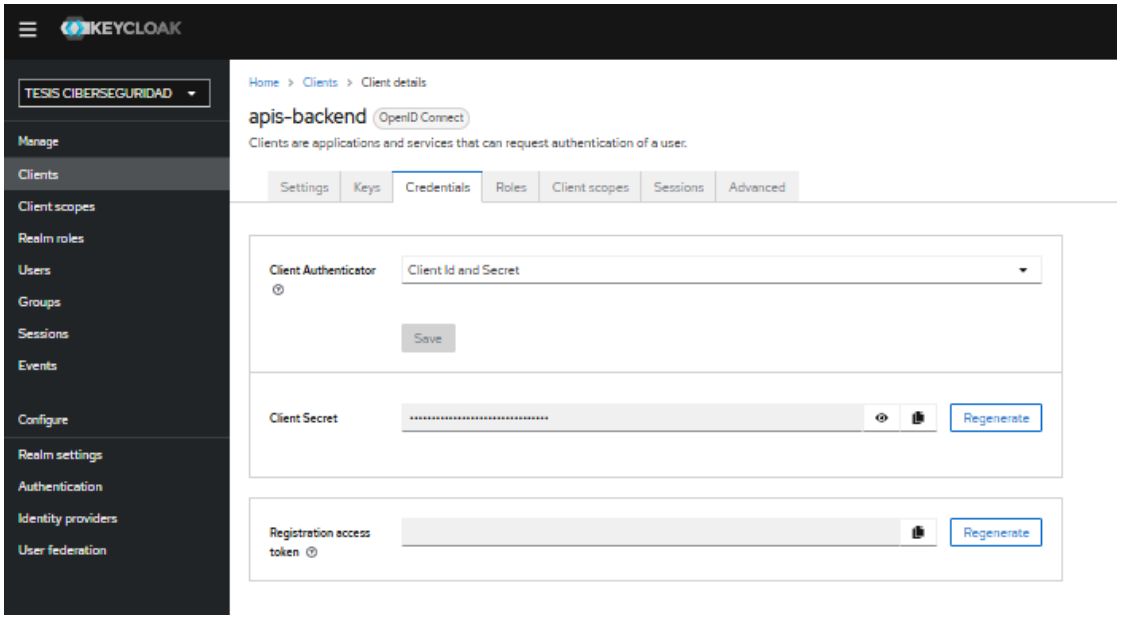
Figura 19. *Configuración del cliente apis-backend en Keycloak.*



Nota. Configuración detallada de las propiedades y capacidades del cliente en la interfaz gráfica de Keycloak.

Se observa la creación del cliente asociado a las APIs protegidas, utilizando el protocolo OpenID Connect y habilitando los mecanismos de autenticación necesarios para la emisión de tokens OAuth 2.0.

Figura 20. *Generación del Client Secret para el cliente apis-backend.*



Nota. Generación y administración del secreto de cliente para la autenticación confidencial del backend en Keycloak.

Keycloak genera un secreto criptográfico utilizado para autenticar aplicaciones confiables durante el proceso de solicitud de tokens y validación de identidad.

3.3.2.3.1. Crear los roles.

Con el propósito de implementar mecanismos de autorización sobre los recursos protegidos, se definieron dos roles dentro del Realm de Keycloak: user y administrator. Estos roles permiten establecer diferentes niveles de acceso a las APIs y verificar que los controles de autorización funcionen correctamente una vez autenticado el usuario.

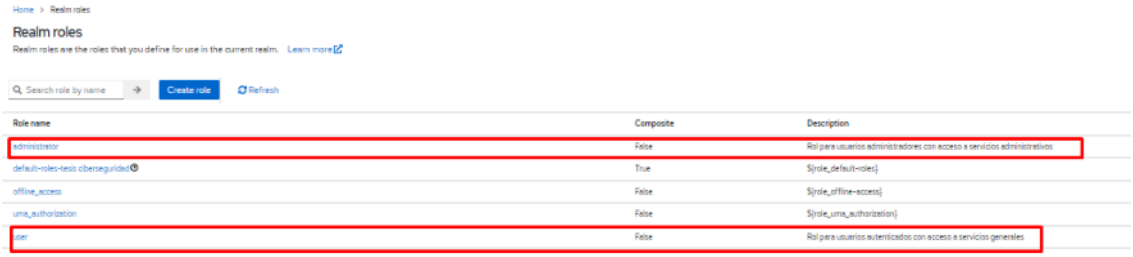
La asignación de roles se realizó directamente en Keycloak, incorporándolos posteriormente dentro de los tokens JWT emitidos por el proveedor de identidad. De esta manera, cada microservicio puede determinar los permisos asociados al usuario autenticado a partir de la información contenida en el token.

Los roles definidos fueron los siguientes:

- User: Usuario autenticado con acceso a funcionalidades generales del sistema. user ->Puede consumir /profile/1
- Administrator: Usuario con privilegios administrativos y acceso a recursos restringidos. admin -> puede consumir /admin/logs

La separación de privilegios permite aplicar el principio de mínimo privilegio, garantizando que cada usuario únicamente pueda acceder a los recursos necesarios para desempeñar sus funciones.

Figura 21. *Configuración de roles de autorización en Keycloak.*

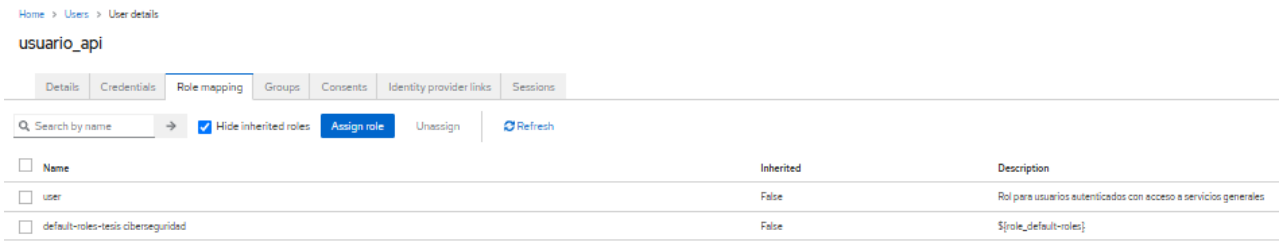


Nota. Configuración y catálogo de roles globales del dominio en Keycloak para la gestión de privilegios jerárquicos.

La Figura 21 muestra los roles user y administrator creados dentro del Realm del proyecto. Estos roles son incorporados en los tokens JWT emitidos por Keycloak y utilizados por las APIs para controlar el acceso a recursos según el nivel de privilegio asignado a cada usuario.

Una vez definidos los roles de autorización, se procedió a crear los usuarios que serán utilizados durante las pruebas de acceso a las APIs protegidas. El objetivo de esta configuración es validar el correcto funcionamiento de los mecanismos de autenticación y autorización implementados en la arquitectura.

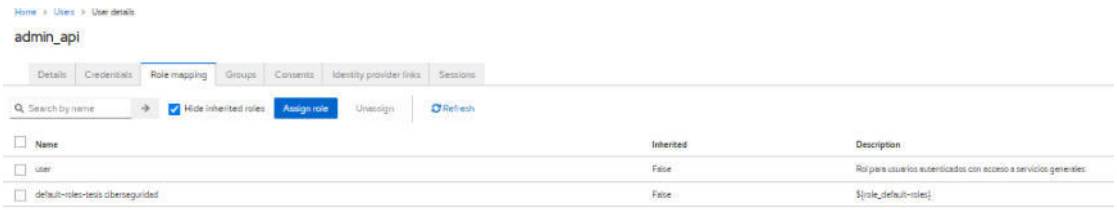
Figura 22. *Asignación del rol user al usuario usuario_api.*



Nota. Asignación y mapeo del rol "user" a una cuenta de identidad dentro de la interfaz de administración de Keycloak.

Se observa la configuración del usuario estándar utilizado para validar el acceso a recursos generales protegidos por OAuth 2.0 y JWT.

Figura 23. *Asignación de roles al usuario admin_api.*



Nota. Configuración de privilegios de la cuenta "admin_api" en el panel de mapeo de roles de Keycloak.

Configuración del usuario destinado a las pruebas de acceso administrativo y validación de controles de autorización basados en roles.

Las Figuras 21 y 22 muestran la asignación de roles realizada a los usuarios definidos para las pruebas del proyecto.

La asignación diferenciada de roles permite validar diversos escenarios de seguridad, incluyendo acceso autorizado, acceso denegado por privilegios insuficientes y protección de funciones administrativas. Estos escenarios resultan fundamentales para demostrar la efectividad de

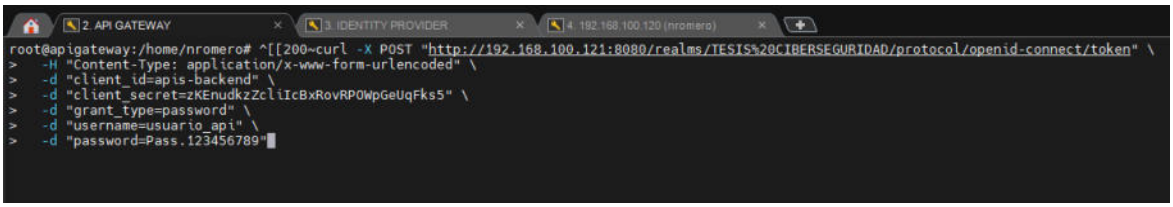
los mecanismos de autorización implementados y la mitigación de vulnerabilidades relacionadas con el control de acceso.

3.3.2.4. Validación de emisión de tokens jwt.

Una vez creados los usuarios y asignados los roles correspondientes, se verificó el correcto funcionamiento del proveedor de identidad Keycloak mediante la obtención de tokens de acceso OAuth 2.0. Esta prueba tuvo como objetivo confirmar que el servidor de identidad autentica correctamente a los usuarios registrados y emite tokens JWT válidos para el acceso a las APIs protegidas.

La solicitud de autenticación se realizó utilizando la herramienta cURL, enviando las credenciales del usuario junto con los parámetros requeridos por el protocolo OpenID Connect. Entre estos parámetros se incluyen el identificador del cliente (Client ID), el secreto del cliente (Client Secret), el nombre de usuario y la contraseña previamente configurados en Keycloak.

Figura 24. *Solicitud de autenticación al proveedor de identidad Keycloak.*



Nota. Solicitud programática de token de acceso JWT mediante cURL utilizando el flujo de credenciales de contraseña en el servidor Keycloak.

Como se observa en la Figura 24, la solicitud es enviada al endpoint de autenticación de Keycloak utilizando el protocolo OpenID Connect.

Solicitud HTTP POST realizada mediante cURL al endpoint OpenID Connect de Keycloak para obtener un token de acceso asociado al usuario autenticado.

Figura 25. *Emisión exitosa de tokens JWT por parte de Keycloak.*

Nota. Estructura de la respuesta del servidor Keycloak y recepción del token de acceso JWT en formato JSON.

La Figura 25 muestra la respuesta generada por Keycloak, donde se observa la emisión satisfactoria del *access token* y del *refresh token*.

Respuesta generada por Keycloak tras la validación de credenciales. Se observa la entrega del *access token*, *refresh token*, tiempo de expiración y parámetros de sesión utilizados por los mecanismos de autenticación y autorización de la arquitectura.

La obtención exitosa de estos tokens confirma el correcto funcionamiento del proveedor de identidad y constituye la base para las pruebas posteriores de autorización, validación de roles y protección de recursos mediante OAuth 2.0 y JSON Web Tokens (JWT).

Tras validar correctamente las credenciales proporcionadas, Keycloak respondió con un conjunto de tokens y parámetros de sesión asociados al usuario autenticado. Entre los elementos más relevantes se encuentran:

- **access_token:** Token JWT utilizado para acceder a los recursos protegidos.
- **expires_in:** Tiempo de validez del token de acceso.
- **refresh_token:** Token utilizado para renovar el acceso sin necesidad de volver a autenticarse.
- **token_type:** Tipo de token emitido.
- **scope:** Alcances autorizados para el usuario autenticado.

Se realiza el mismo proceso de validación de generación de token para el segundo usuario

Figura 26. *Emisión exitosa de tokens JWT por parte de Keycloak.*

Nota. Petición programática y recepción de token de acceso JWT para la cuenta "admin_api" desde el nodo de microservicios robustecidos.

3.3.2.5. Verificar el contenido del jwt

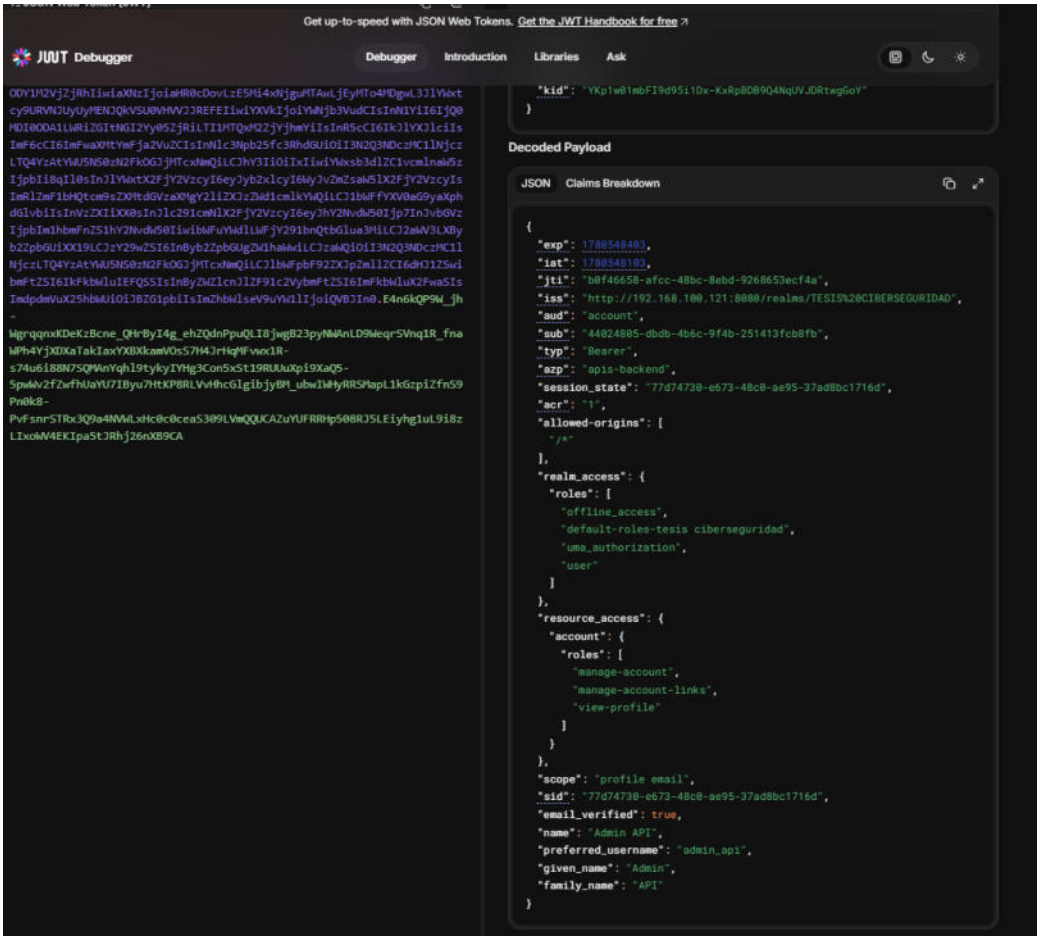
Una vez verificada la emisión de tokens para los usuarios configurados en Keycloak, se procedió a analizar su contenido con el fin de validar que la información de autenticación y autorización se incorporara correctamente dentro del JWT.

Para ello, el token emitido fue decodificado utilizando una herramienta de análisis JWT, permitiendo inspeccionar los *claims* incluidos en el payload. Como se observa en la Figura 26, el token contiene información relacionada con la identidad del usuario, el emisor del token, los tiempos de vigencia y los roles asignados durante el proceso de autenticación.

Entre los *claims* más relevantes identificados se encuentran:

- iss, Identificador del proveedor de identidad que emitió el token.
- sub, Identificador único del usuario autenticado.
- aud, Audiencia para la cual fue emitido el token.
- iat, Fecha y hora de emisión del token.
- exp, Fecha y hora de expiración del token.
- preferred_username, Nombre de usuario autenticado.
- realm_access.roles, Roles asignados al usuario dentro de Keycloak.

Figura 27. *Análisis del contenido del token JWT emitido por Keycloak.*



Nota. Inspección gráfica y decodificación de los atributos criptográficos del token JWT para el usuario "admin_api" mediante la herramienta JWT.io

Dentro del campo realm_access.roles se verificó la presencia de los roles asociados al usuario autenticado:

realm_access": {

"roles": [

"offline_access",

"default-roles-tesis ciberseguridad",

"uma_authorization",

"user"

]

La inclusión de esta información dentro del JWT permite que los componentes de la arquitectura, como el API Gateway y los microservicios protegidos, puedan tomar decisiones de autorización sin necesidad de consultar continuamente al proveedor de identidad.

Asimismo, se verificó que el token incorpora los tiempos de emisión y expiración (iat y exp), permitiendo controlar la vigencia de las credenciales y reducir el riesgo asociado al uso indebido de tokens comprometidos.

La correcta incorporación de estos claims confirma que la configuración de Keycloak fue realizada satisfactoriamente y que los tokens emitidos contienen la información necesaria para implementar mecanismos de autenticación y autorización basados en OAuth 2.0 y JSON Web Tokens (JWT).

Con esta configuración, se configuró y preparó Keycloak para emitir token OAuth 2.0/JWT. Una vez realizado esto se debe usar los tokens emitidos por el proveedor de identidad para probar las APIs seguras.

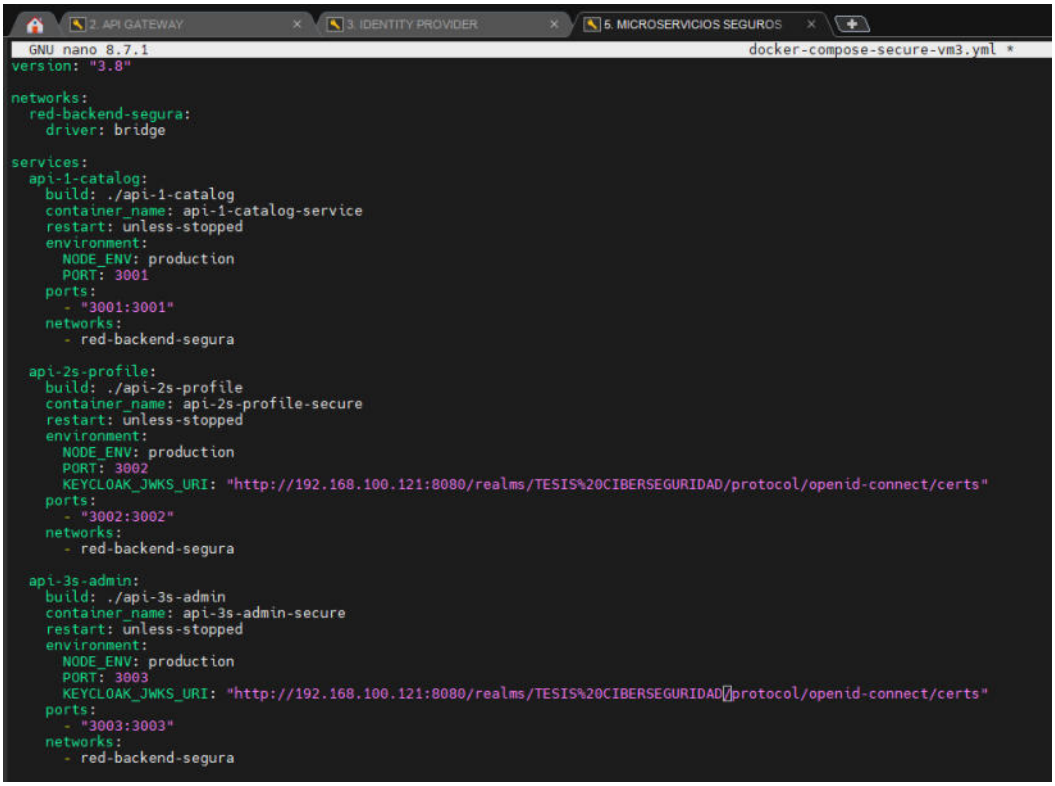
3.3.2.6. Configuración de archivo de despliegue de api para validación de tokens emitidos por keycloak

En esta etapa se configuró el archivo docker-compose-secure-vm3.yml, encargado de desplegar las APIs seguras dentro de una red Docker independiente denominada red-backend-segura. Esta configuración permite ejecutar cada microservicio en un contenedor separado, manteniendo una estructura modular y controlada.

Dentro del archivo se definieron los servicios api-1-catalog, api-2-profile y api-3-admin, cada uno expuesto en los puertos 3001, 3002 y 3003, respectivamente. Para los servicios que requieren validación de identidad y autorización, se incorporó la variable de entorno KEYCLOAK_JWKS_URI, la cual apunta al endpoint de certificados de Keycloak.

Esta configuración permite que las APIs consulten las claves públicas publicadas por Keycloak mediante JWKS, con el fin de validar la firma de los tokens JWT recibidos en cada solicitud. De esta manera, los microservicios pueden verificar que el token haya sido emitido por el proveedor de identidad autorizado, que no haya sido alterado y que contenga los atributos necesarios para aplicar los controles de acceso definidos.

Figura 28. *Configuración del archivo Docker Compose para APIs seguras.*



Nota. Archivo de manifiesto declarativo Docker Compose para el entorno robustecido e inyección de variables de validación criptográfica.

Se muestra la configuración de despliegue de los microservicios seguros mediante Docker Compose, incluyendo la red interna red-backend-segura, los puertos expuestos y la variable KEYCLOAK_JWKS_URI, utilizada para validar tokens JWT emitidos por Keycloak mediante el endpoint JWKS.

3.4. ATAQUE 2: Broken Object Level Authorization (BOLA)

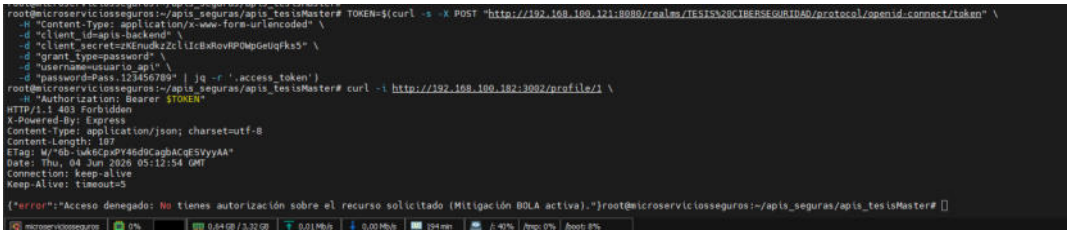
Una vez implementados los mecanismos de autenticación y autorización basados en OAuth 2.0 y JWT, se realizó una prueba orientada a verificar la protección contra vulnerabilidades de tipo BOLA.

Para ello, el usuario usuario_api, asociado al rol user, obtuvo un token JWT válido emitido por Keycloak y posteriormente intentó acceder al recurso /profile/1. La solicitud fue enviada incluyendo el token en el encabezado Authorization: Bearer, permitiendo que el sistema validara la identidad del usuario antes de procesar la petición.

Como se observa en la Figura 28, la solicitud fue rechazada por el servidor con el código de estado HTTP 403 Forbidden, indicando que el usuario autenticado no posee autorización para acceder al recurso solicitado.

3.4.1. **Acceso con usuario “user” (Sin privilegio)**

Figura 29. *Validación de autorización y mitigación de vulnerabilidad BOLA.*



Nota. Validación empírica de la mitigación de la vulnerabilidad BOLA en el microservicio de perfiles mediante respuesta restrictiva HTTP 403 Forbidden.

La respuesta generada por la API fue la siguiente:

{

“error”: “Acceso denegado: No tienes autorización sobre el recurso solicitado (Mitigación BOLA activa).”

}

Este resultado confirma que el sistema realiza correctamente las validaciones de autorización sobre los recursos consultados, verificando que el usuario autenticado únicamente pueda acceder a la información que le pertenece. Aunque el token JWT fue validado satisfactoriamente, la API detectó que el identificador solicitado no correspondía al usuario autenticado y bloqueó el acceso.

Durante la prueba se verificó el correcto funcionamiento de los siguientes controles de seguridad:

Tabla 6.

Matriz de los controles de seguridad

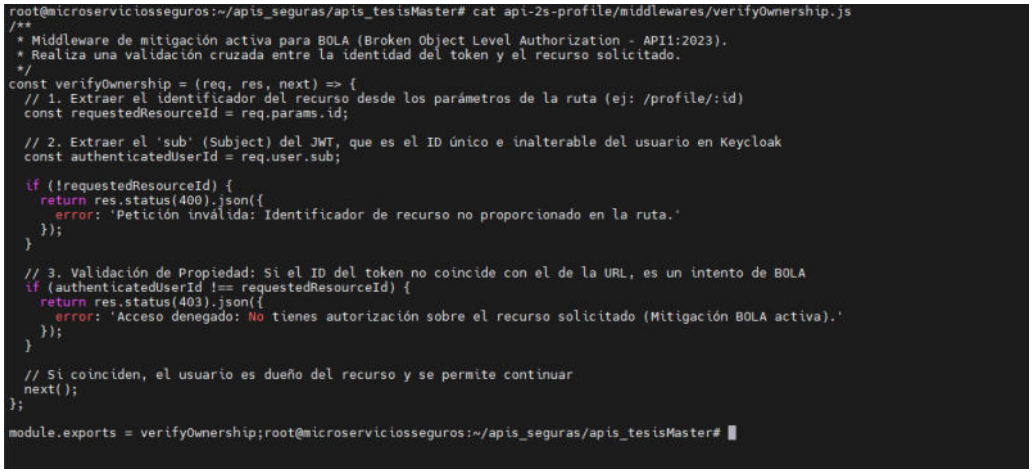
Control de seguridad	Resultado
Validación del JWT	Correcta
Verificación de firma digital	Correcta
Validación mediante JWKS	Correcta
Autenticación del usuario	Correcta
Control de autorización sobre el recurso	Correcto
Mitigación de BOLA	Activa

Nota. Matriz de resultados y efectividad de los controles de seguridad en el microservicio de perfiles robustecido.

En esta etapa se verificó el mecanismo implementado para controlar el acceso a recursos asociados a un usuario específico. Para ello, se revisó el middleware verifyOwnership.js, encargado de validar que el identificador del recurso solicitado en la ruta coincida con el identificador del usuario autenticado contenido en el token JWT.

El control funciona extrayendo dos valores principales: el parámetro id recibido en la URL y el claim sub del JWT, que representa el identificador único del usuario autenticado en Keycloak. Si ambos valores no coinciden, la API rechaza la solicitud con un código HTTP 403 Forbidden, impidiendo que un usuario acceda a recursos que no le pertenecen.

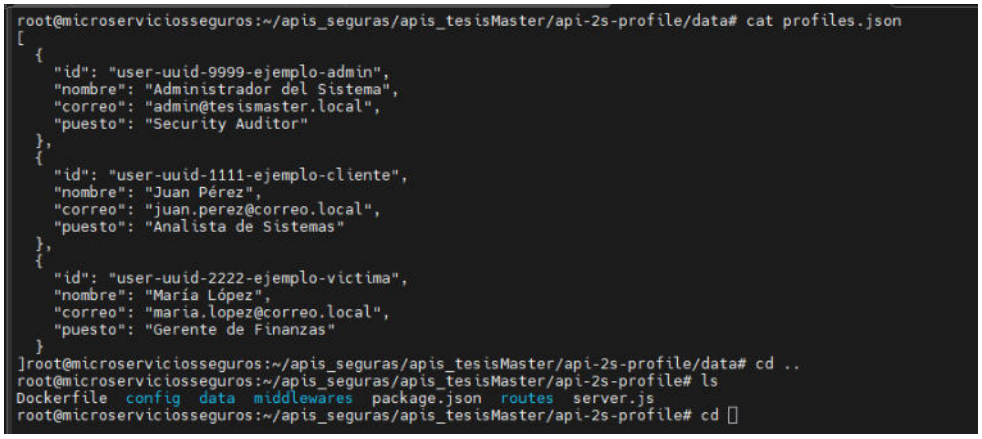
Figura 30. *Middleware de validación de propiedad del recurso.*



Nota. Implementación lógica del middleware de control de acceso contextual en Node.js para la mitigación activa de BOLA.

Se muestra la lógica implementada en verifyOwnership.js, donde se compara el identificador recibido en la URL con el claim sub del JWT. Si los valores no coinciden, la solicitud es rechazada con un código HTTP 403, activando la mitigación contra BOLA.

Figura 31. *Datos de perfiles utilizados para pruebas de autorización.*



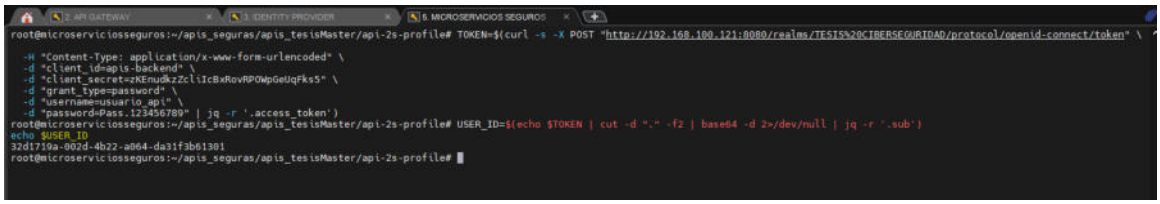
Nota. Estructura del almacén de datos local en formato JSON y arquitectura de archivos para el microservicio de perfiles.

Archivo profiles.json utilizado como base de datos simulada para representar diferentes perfiles de usuario dentro del entorno de pruebas.

El ID real del usuario usuario_api es:

32d1719a-002d-4b22-a064-da31f3b61301

Figura 32. *Extracción del identificador del usuario usuario_api desde el JWT.*



Nota. Extracción programática del identificador de sujeto (claim "sub") a partir del token de acceso JWT en la terminal del host protegido.

Se obtiene el valor del claim sub contenido en el token JWT emitido por Keycloak, correspondiente al usuario estándar utilizado en las pruebas.

Del usuario admin_api:

44024805-dbdb-4b6c-9f4b-251413fcb8fb

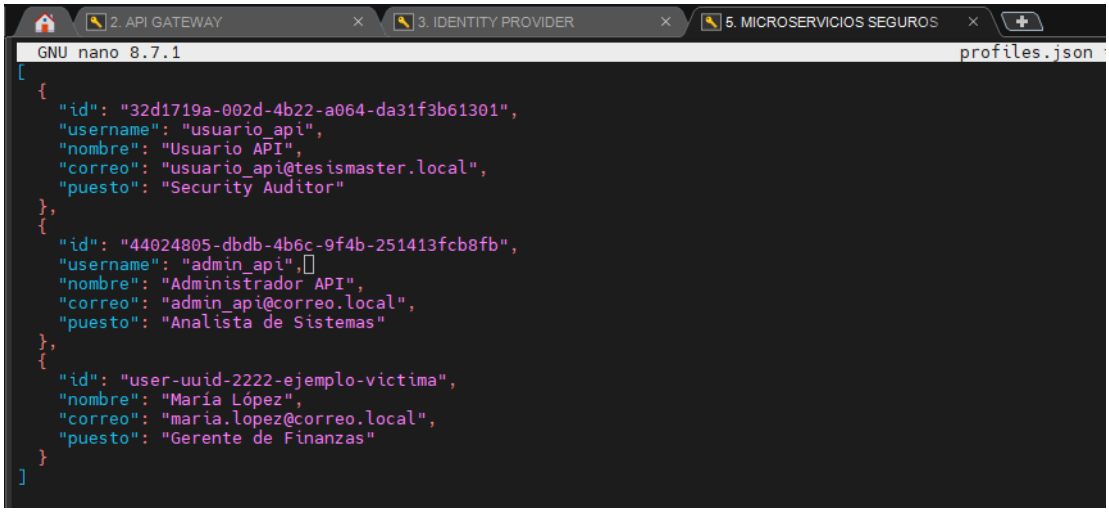
Figura 33. *Extracción del identificador del usuario admin_api desde el JWT.*

Nota. Aislamiento por consola y decodificación del identificador de sujeto (claim "sub") para el token de acceso de la cuenta "admin_api".

Se obtiene el valor del claim sub contenido en el token JWT emitido por Keycloak, correspondiente al usuario administrador utilizado en las pruebas.

Cambiamos los datos de los usuarios asignando los ID reales de los usuarios creados en KEYCLOAK los cuales tienes roles establecidos y autorización en el proveedor de identidad

Figura 34. *Asociación de usuarios de la aplicación con identificadores emitidos por Keycloak.*



Nota. Actualización y mapeo de identificadores de persistencia con esquemas UUIDv4 síncronos al Proveedor de Identidad en el archivo profiles.json

Archivo de datos de la API Profile donde se observa la asignación de los identificadores únicos (sub) generados por Keycloak a los usuarios utilizados durante las pruebas. Esta configuración permite validar la propiedad de los recursos y aplicar controles de autorización basados en identidad para mitigar vulnerabilidades BOLA.

3.4.2. **Autorización Exitosa**

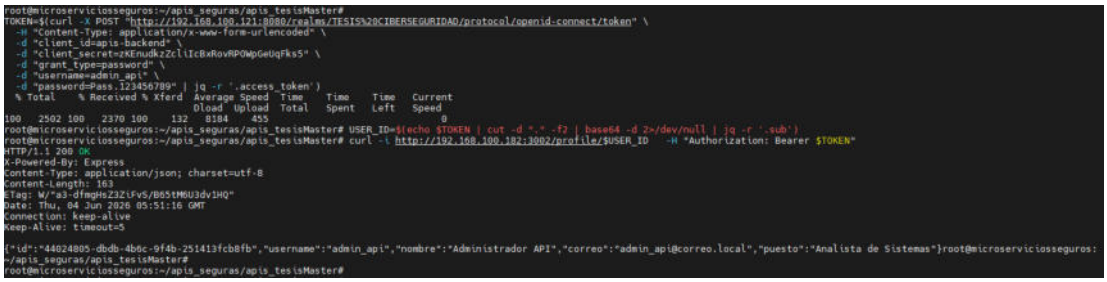
Una vez implementado el mecanismo de validación de propiedad del recurso, se realizaron pruebas para verificar que únicamente los usuarios propietarios de la información pudieran acceder a sus respectivos perfiles. El objetivo fue comprobar la efectividad de los controles implementados para mitigar vulnerabilidades de tipo BOLA.

Figura 35. *Acceso autorizado del usuario usuario_api a su propio recurso.*

Nota. Validación del acceso legítimo (caso de éxito) en el microservicio de perfiles seguro mediante respuesta HTTP 200 OK.

El usuario autenticado accede al perfil asociado a su identificador contenido en el token JWT. La API valida correctamente la autenticación y autorización, respondiendo con HTTP 200 OK.

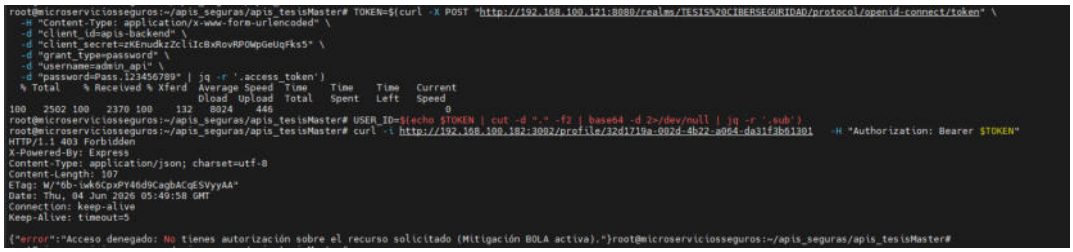
Figura 36. *Acceso autorizado del usuario admin_api a su propio recurso.*



Nota. Validación de la consistencia operativa y acceso legítimo para la cuenta "admin_api" mediante respuesta HTTP 200 OK.

El usuario administrador consulta el recurso asociado a su identidad. La validación del token JWT y la comprobación de propiedad del recurso permiten el acceso legítimo a la información.

Figura 37. *Bloqueo de acceso a recursos pertenecientes a otro usuario.*



Nota. Prueba de ataque dinámico y bloqueo cruzado ante un intento de explotación de BOLA en el microservicio de perfiles robustecido.

El usuario admin_api intenta acceder al perfil asociado al identificador de usuario_api. La API detecta que el recurso solicitado no corresponde al propietario autenticado y rechaza la solicitud con HTTP 403 Forbidden, evidenciando la mitigación activa de la vulnerabilidad BOLA.

Los resultados obtenidos permiten confirmar el correcto funcionamiento de los controles implementados:

Tabla 7.

Cuadro del estado de HTTP

Escenario evaluado	Resultado
JWT válido + recurso propio	HTTP 200 OK
JWT válido + recurso ajeno	HTTP 403 Forbidden
Solicitud sin JWT	HTTP 401 Unauthorized

Nota. Matriz de comportamiento síncrono y códigos de estado HTTP según el escenario de autorización evaluado.

Estos resultados evidencian que la arquitectura implementada no solo valida la autenticidad de los tokens emitidos por Keycloak, sino que también verifica la autorización sobre cada recurso solicitado. De esta manera, se evita que usuarios autenticados accedan a información perteneciente a terceros mediante la manipulación de identificadores, mitigando efectivamente la vulnerabilidad BOLA identificada en el escenario sin controles de seguridad.

3.4.3.Api gateway (kong)

Una vez implementados los mecanismos de autenticación y autorización en los microservicios, se procedió a configurar el API Gateway Kong como punto central de acceso a los servicios protegidos.

Para ello, se definieron rutas específicas para cada API dentro del archivo de configuración declarativa de Kong. Cada ruta fue asociada al microservicio correspondiente, permitiendo centralizar el acceso a los recursos mediante un único punto de entrada.

Como se observa en la Figura 37, se configuraron las siguientes rutas:

Tabla 8.

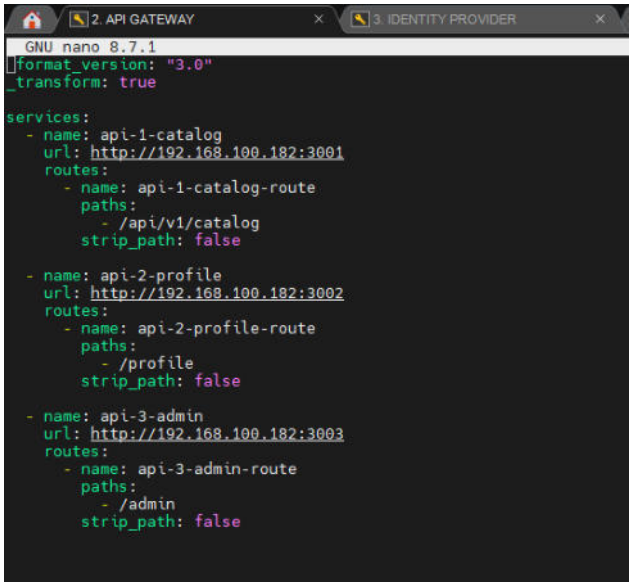
Direccionamiento de la red local

Servicio	URL Backend	Ruta expuesta
API-1-Catalog	http://192.168.100.182:3001	/api/v1/catalog
API-2-Profile	http://192.168.100.182:3002	/profile
API-3-Admin	http://192.168.100.182:3003	/admin

Nota. Mapeo de direccionamiento de red local, segmentación de puertos TCP y rutas lógicas de los microservicios expuestos.

Esta configuración permite que todas las solicitudes dirigidas a las APIs sean procesadas inicialmente por Kong antes de ser reenviadas a los microservicios internos.

Figura 38. *Configuración de rutas y servicios en Kong API Gateway.*



Nota. Archivo de configuración declarativa YAML en Kong API Gateway para el enrutamiento perimetral de los microservicios.

Archivo de configuración declarativa utilizado para registrar los microservicios y las rutas expuestas a través de Kong, permitiendo centralizar el acceso a las APIs protegidas.

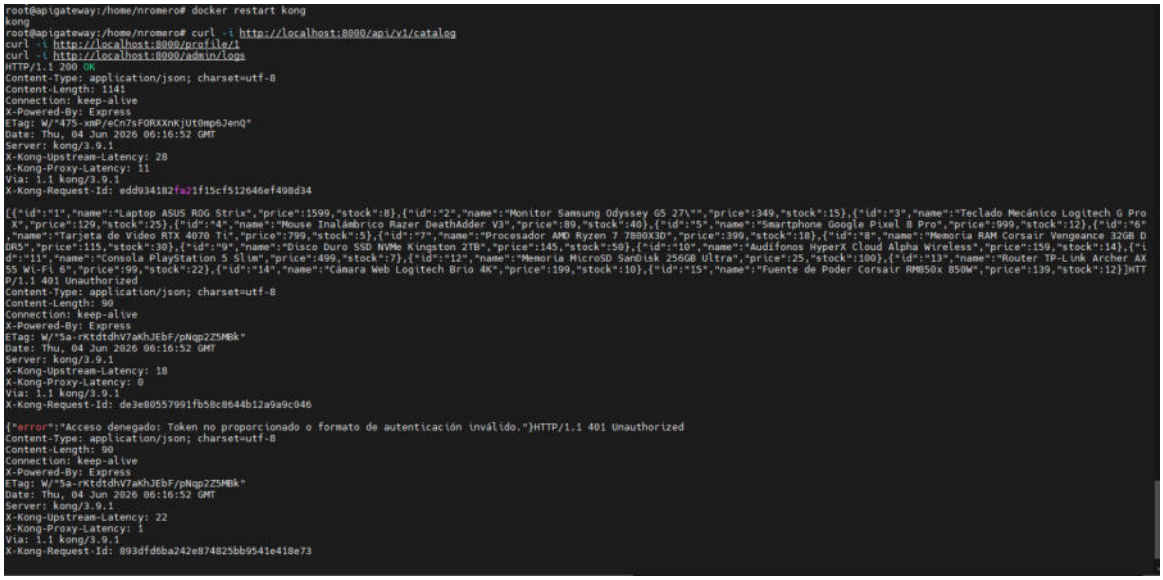
3.4.4. Validación consumo de las API desde API GATEWAY

Una vez publicadas las rutas, se realizaron pruebas de conectividad para verificar que Kong reenviara correctamente las solicitudes hacia los servicios backend.

Como se muestra en la Figura 38, el acceso al endpoint de catálogo fue procesado exitosamente obteniendo una respuesta HTTP 200 OK, confirmando que la comunicación entre Kong y el microservicio correspondiente se encontraba operativa.

Durante la misma prueba se intentó acceder a recursos protegidos sin proporcionar un token JWT válido. En estos casos, Kong y los servicios backend respondieron con el código HTTP 401 Unauthorized, indicando que la solicitud fue rechazada debido a la ausencia de credenciales válidas.

Figura 39. *Validación de acceso a través del API Gateway.*



Nota. Auditoría de enrutamiento y validación perimetral de solicitudes HTTP con y sin credenciales en Kong API Gateway.

La respuesta obtenida fue:

{

"error": "Acceso denegado: Token no proporcionado o formato de autenticación inválido."

}

Este comportamiento demuestra que los controles de autenticación se encuentran activos y que las APIs no permiten el acceso a recursos protegidos sin una identidad previamente validada.

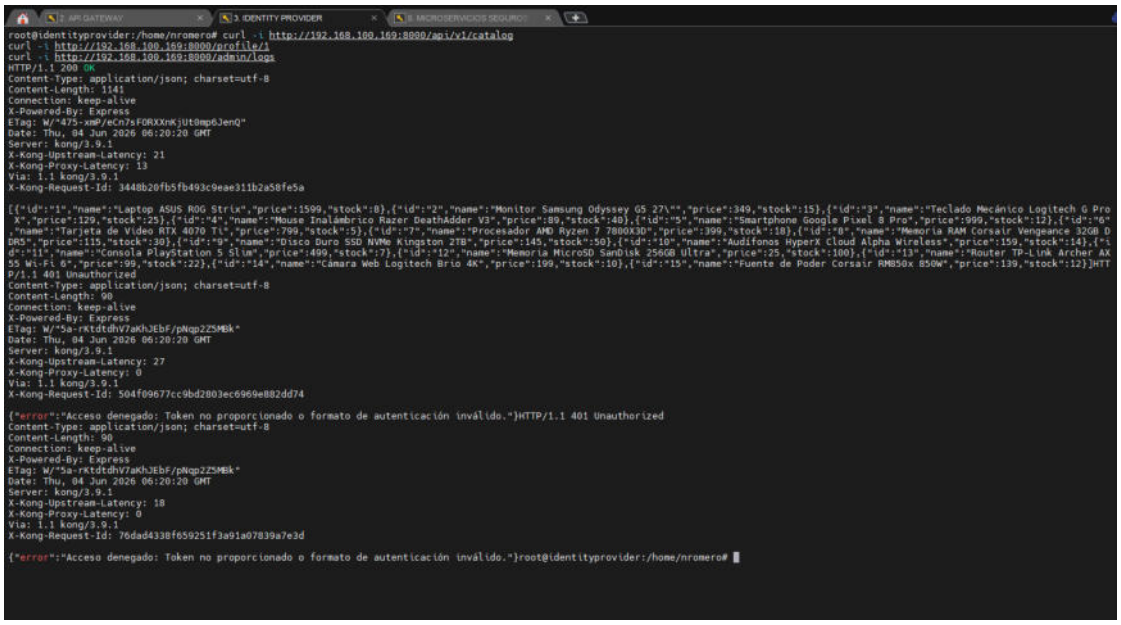
3.4.5. Validación del consumo de las apis mediante API Gateway desde otra máquina

Para verificar el comportamiento de la arquitectura desde un punto de vista similar al de un consumidor real de la API, se realizaron pruebas desde una máquina diferente al servidor donde se ejecuta Kong.

Como se observa en la Figura 39, las solicitudes realizadas al API Gateway fueron procesadas correctamente, permitiendo acceder únicamente a los recursos públicos y rechazando aquellas peticiones que requerían autenticación.

Esta prueba confirma que la arquitectura mantiene el mismo comportamiento independientemente del origen de la solicitud, centralizando las políticas de acceso en el API Gateway.

Figura 40. *Consumo de APIs mediante Kong desde un equipo externo.*



Nota. Auditoría de políticas perimetrales del API Gateway mediante solicitudes HTTP remotas desde el nodo del Proveedor de Identidad.

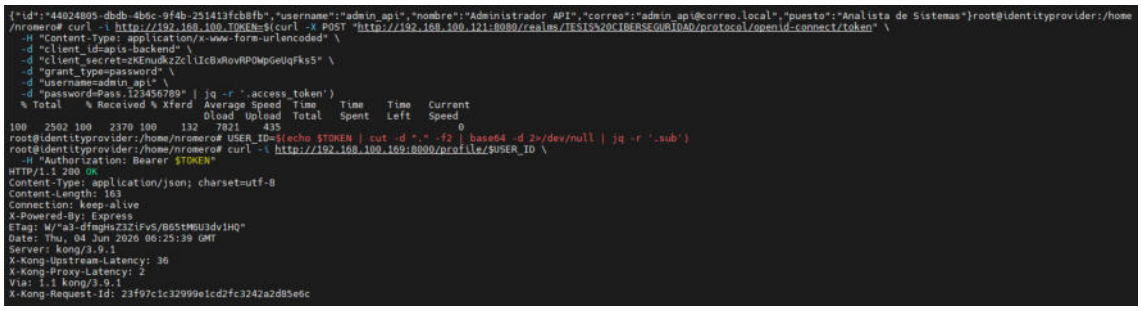
Solicitudes realizadas desde una máquina diferente al servidor de aplicaciones, demostrando la correcta publicación de los servicios y la aplicación centralizada de controles de acceso.

Finalmente, se validó el flujo completo de autenticación y autorización utilizando tokens JWT emitidos por Keycloak.

Para ello, se obtuvo un token de acceso válido mediante el endpoint OpenID Connect del proveedor de identidad y posteriormente se incluyó en el encabezado:

Tal como se muestra en la Figura 40, una vez presentado un token válido, Kong permitió el acceso al recurso solicitado y reenvió la petición al microservicio correspondiente, obteniendo una respuesta HTTP 200 OK.

Figura 41. *Acceso autorizado mediante token JWT por Keycloack.*



Nota. Validación del ciclo completo de autenticación, extracción programática de identidad y consumo seguro a través de Kong API Gateway desde un host remoto.

Flujo de autenticación exitoso donde el usuario obtiene un token JWT desde Keycloak y posteriormente accede a un recurso protegido a través de Kong API Gateway, obteniendo una respuesta HTTP 200 OK.

3.5. ATAQUE 3: Ataque BFLA (Acceso denegado)

Con el objetivo de verificar la efectividad de los controles de autorización implementados, se realizó una prueba orientada a evaluar la protección contra vulnerabilidades de tipo BFLA.

La prueba consistió en autenticar un usuario con privilegios estándar mediante un token JWT válido emitido por Keycloak e intentar acceder a una funcionalidad administrativa expuesta a través del endpoint:

GET /admin/logs

Aunque la solicitud incluía un token válido y la autenticación fue completada satisfactoriamente, el usuario no disponía del rol administrativo requerido para ejecutar dicha función.

Como se observa en la Figura 41, la API rechazó la solicitud respondiendo con el código HTTP 403 Forbidden, generando el siguiente mensaje:

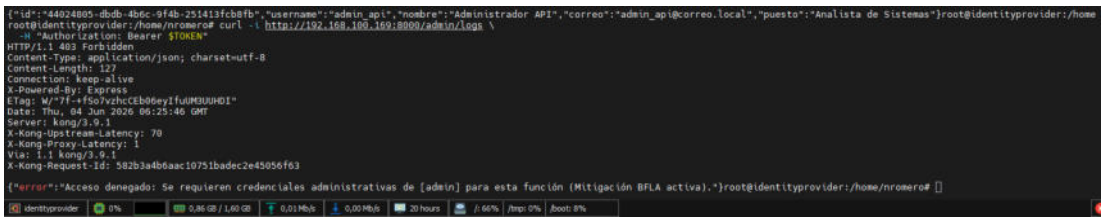
{

"error": "Acceso denegado: Se requieren credenciales administrativas de [admin] para esta función (Mitigación BFLA activa)."

}

El resultado obtenido confirma que el sistema no se limita a validar la autenticidad del token JWT, sino que además verifica los roles y privilegios asociados al usuario autenticado antes de permitir el acceso a funcionalidades sensibles.

Figura 42. *Mitigación de Broken Function Level Authorization (BFLA).*



Nota. Prueba de ataque dinámico y bloqueo perimetral ante un intento de explotación de BFLA en el endpoint de administración robustecido.

La prueba permitió confirmar la diferencia entre autenticación y autorización dentro de la arquitectura implementada. Mientras que la autenticación verifica la identidad del usuario mediante OAuth 2.0 y JWT, la autorización determina qué operaciones puede ejecutar en función de los roles incorporados en el token.

Este comportamiento mitiga eficazmente la vulnerabilidad BFLA (API5:2023) descrita por OWASP, evitando escenarios donde usuarios legítimos puedan elevar privilegios o acceder a funciones administrativas mediante la manipulación de solicitudes.

3.5.1.Código de validación de role

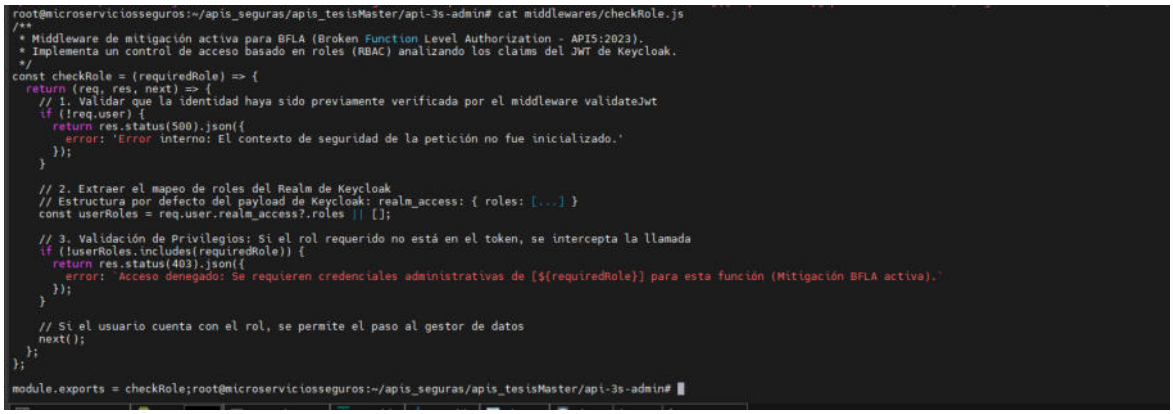
Para mitigar vulnerabilidades de tipo BFLA, se implementó un middleware denominado checkRole.js, encargado de verificar los privilegios asociados al usuario autenticado antes de permitir el acceso a funciones administrativas.

Como se observa en la Figura 42, el middleware es ejecutado después de la validación del token JWT. Su función principal consiste en analizar los *claims* de autorización contenidos en el token emitido por Keycloak y determinar si el usuario posee el rol requerido para acceder al recurso solicitado.

El mecanismo implementado realiza las siguientes validaciones:

1. Verifica que la identidad del usuario haya sido previamente autenticada mediante el middleware de validación JWT.
2. Extrae los roles contenidos en el claim realm_access.roles del token JWT.
3. Comprueba que el rol requerido por el endpoint se encuentre dentro de los privilegios asignados al usuario.
4. Rechaza la solicitud con un código HTTP 403 Forbidden cuando el usuario no posee los permisos necesarios.
5. Permite la ejecución de la operación cuando el rol requerido está presente en el token.

Figura 43. *Middleware de validación de roles para mitigación de BFLA.*



Nota. Implementación lógica del middleware de control de acceso basado en roles en Node.js para la mitigación activa de BFLA.

La implementación del middleware permite impedir que usuarios autenticados accedan a funcionalidades que excedan sus privilegios. Cuando un usuario intenta ejecutar una operación administrativa sin disponer del rol correspondiente, la solicitud es interceptada y rechazada antes de llegar a la lógica de negocio de la aplicación.

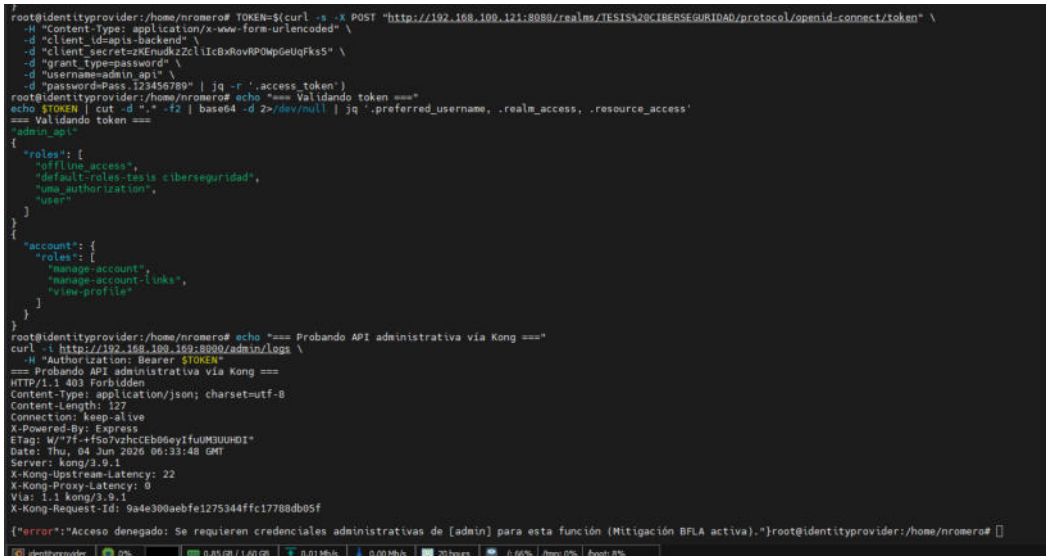
Este mecanismo evita escenarios de escalamiento horizontal o vertical de privilegios y constituye uno de los controles fundamentales para mitigar la vulnerabilidad API5:2023 BFLA del estándar OWASP API Security Top 10.

La validación basada en roles complementa los mecanismos de autenticación implementados mediante OAuth 2.0 y JWT, garantizando que no solo se verifique la identidad del usuario, sino también los permisos asociados a dicha identidad.

Con el objetivo de verificar la correcta aplicación de los controles de autorización basados en roles, se realizó una prueba utilizando el usuario admin_api, el cual se encontraba autenticado correctamente en Keycloak pero no tenía asignado el rol administrator dentro del Realm.

Como se observa en la Figura 43, inicialmente se validó el contenido del token JWT emitido por Keycloak, verificando los roles incorporados dentro del claim realm_access.roles. El análisis del token evidenció que el usuario disponía únicamente de los siguientes privilegios:

Figura 44. *Rechazo de acceso administrativo por ausencia del rol requerido.*



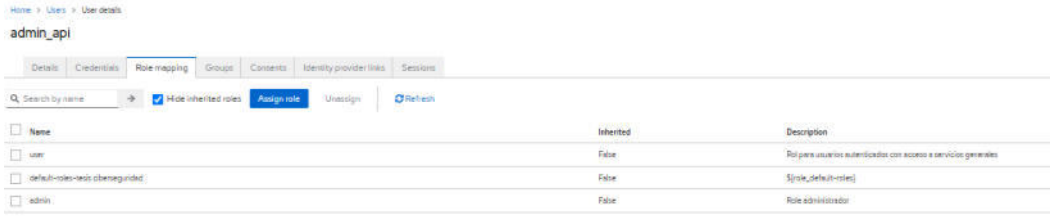
Nota. Secuencia automatizada de validación criptográfica de roles e interceptación perimetral del ataque BFLA desde un host remoto.

El resultado obtenido confirma que la autenticación por sí sola no es suficiente para acceder a funcionalidades privilegiadas. La aplicación verifica adicionalmente los roles contenidos en el JWT y únicamente autoriza la ejecución de operaciones administrativas cuando el usuario posee explícitamente el rol requerido.

Esta validación demuestra la correcta implementación del modelo de control de acceso basado en roles (RBAC), evitando que usuarios autenticados puedan acceder a funciones administrativas mediante la simple posesión de un token válido.

La incorporación del rol admin permite que Keycloak incluya dicho privilegio dentro del claim realm_access.roles del token JWT emitido al usuario. Posteriormente, este atributo será evaluado por el middleware checkRole.js, responsable de verificar si el usuario posee autorización para ejecutar operaciones administrativas.

Figura 45. *Asignación del rol administrativo al usuario admin_api en Keycloack.*



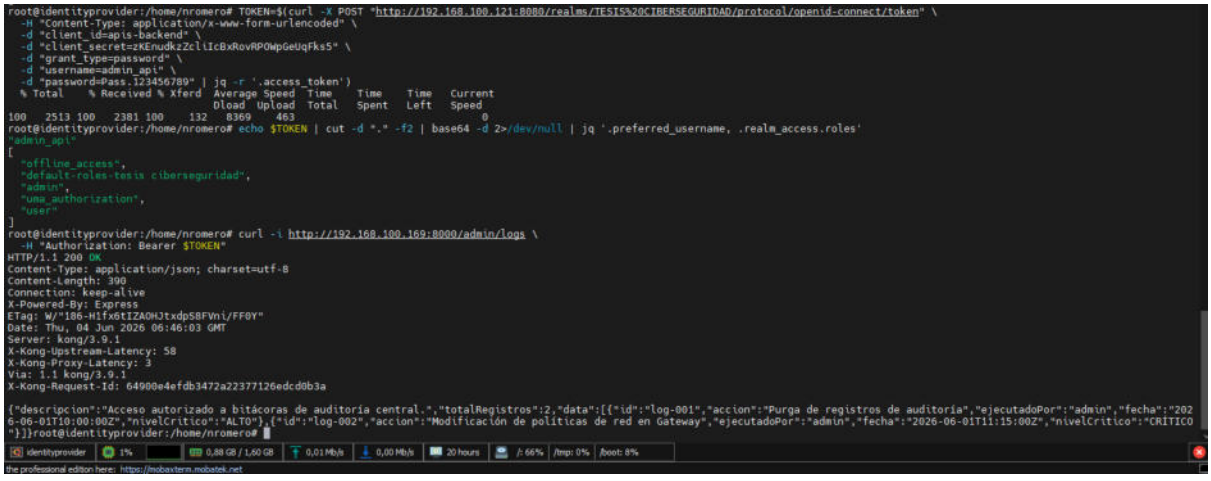
Nota. Asignación efectiva del rol de alta jerarquía "admin" a la cuenta de identidad "admin_api" en el panel de Keycloak.

Configuración de roles realizada en el proveedor de identidad Keycloak. Se observa la asignación del rol admin al usuario admin_api, permitiendo que dicho privilegio sea incorporado en los tokens JWT emitidos y utilizado posteriormente por los mecanismos de autorización para controlar el acceso a funciones administrativas.

Una vez asignado el rol admin al usuario admin_api dentro de Keycloak, se generó un nuevo token JWT con el fin de verificar que el privilegio fuera incorporado correctamente dentro del claim realm_access.roles.

Como se observa en la Figura 45, el análisis del token emitido por Keycloak confirmó la presencia del rol administrativo.

Figura 46. *Acceso autorizado a funciones administrativas mediante rol admin.*



Nota. Validación del control positivo para BFLA y acceso autorizado al endpoint de administración a través de Kong API Gateway.

La solicitud fue procesada exitosamente y la API respondió con el código HTTP 200 OK, entregando la información de las bitácoras de auditoría centralizadas del sistema. La respuesta obtenida confirma que el middleware de autorización basado en roles validó correctamente la presencia del rol admin dentro del token JWT y permitió el acceso a la funcionalidad administrativa protegida.

Los resultados obtenidos evidencian que la autorización depende de los privilegios contenidos en el token JWT emitido por Keycloak y no únicamente de la autenticación del usuario. Este comportamiento permite mitigar eficazmente la vulnerabilidad BFLA (API5:2023) al impedir que usuarios sin privilegios administrativos ejecuten funciones restringida.

3.5.2. Consumo de la API de forma segura con API GATEWAY (KONG)

El consumo de las APIS de forma directa representa una vulnerabilidad, ya que no se tiene un control sobre las peticiones realizadas por el cliente. Un atacante puede realizar solicitudes masivas en un intervalo de tiempo pequeño, logrando realizar un ataque de denegación de servicio, enumeración masiva, ataques de fuerza bruta, entre otro. Por este motivo, se realizó configuraciones seguras en el servicio API GATEWAY con el objetivo que realice un control de seguridad sobre las peticiones o solicitudes realizadas por los clientes que consumen las API.

3.5.3. Restricción del acceso directo a los microservicios

El proyecto no contempla el diseño integral de controles perimetrales ni políticas de hardening de red. No obstante, podrán emplearse mecanismos básicos de aislamiento dentro del entorno experimental con el único propósito de preservar el flujo de autenticación y autorización definido en la arquitectura propuesta.

Se simuló la configuración de red de un firewall perimetral para bloquear el tráfico hacia y desde las APIS expuestas en los puertos 3001, 3002 y 3003 para direcciones IP diferentes a las del

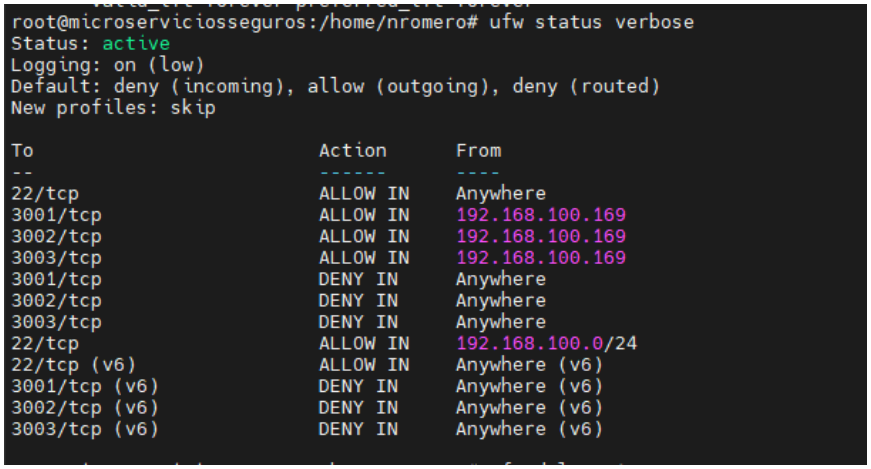
API GATEWAY. El bloqueo se realizó utilizando el firewall local del servidor que contiene los contenedores Docker que exponen las APIs conforme se muestra en la Figura 46.

Figura 47. *Reglas de firewall local para bloquear el tráfico de entrada y salida hacia los puertos de las APIs.*

Nota. Configuración de políticas de aislamiento de red mediante el cortafuegos UFW en el host de microservicios seguros.

La Figura 47 muestra las reglas creadas para permitir y bloquear el tráfico hacia los puertos de consumo de las APIs.

Figura 48. *Reglas creadas en firewall local.*



Nota. Verificación y estado detallado de las políticas de filtrado del cortafuegos UFW en el host de microservicios.

Debido a que las APIs utilizadas están desplegadas en Docker, es necesario realizar reglas de filtrado de tráfico específicas para el servicio Docker, ya que, por defecto el funcionamiento de Docker salta las reglas de configuración del firewall local. La Figura 48 muestra las reglas de filtrado creadas.

Figura 49. *Reglas de filtrado de tráfico para servicio Docker.*

```
root@microservicioss eguros:/home/nromero# iptables -I DOCKER-USER 1 -p tcp -s 192.168.100.169 --dport 3001 -j ACCEPT
root@microserviciosseguros:/home/nromero# iptables -I DOCKER-USER 2 -p tcp -s 192.168.100.169 --dport 3002 -j ACCEPT
root@microserviciosseguros:/home/nromero# iptables -I DOCKER-USER 3 -p tcp -s 192.168.100.169 --dport 3003 -j ACCEPT
root@microserviciosseguros:/home/nromero# iptables -I DOCKER-USER 4 -p tcp --dport 3001 -j DROP
iptables -I DOCKER-USER 5 -p tcp --dport 3002 -j DROP
iptables -I DOCKER-USER 6 -p tcp --dport 3003 -j DROP
root@microserviciosseguros:/home/nromero# iptables -L DOCKER-USER -n -v --line-numbers
Chain DOCKER-USER (1 references)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 ACCEPT     tcp  --  *      *       192.168.100.169      0.0.0.0/0            tcp dpt:3001
2        0     0 ACCEPT     tcp  --  *      *       192.168.100.169      0.0.0.0/0            tcp dpt:3002
3        0     0 ACCEPT     tcp  --  *      *       192.168.100.169      0.0.0.0/0            tcp dpt:3003
4        0     0 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp dpt:3001
5        0     0 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp dpt:3002
6        0     0 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp dpt:3003
root@microserviciosseguros:/home/nromero#
```

Nota. Configuración e inspección de directivas de aislamiento perimetral en la cadena DOCKER-USER de iptables para el entorno de contenedores.

La Figura 49 y 50 muestran las pruebas de consumo de los servicios antes y después de la configuración de las reglas de filtrado de tráfico.

3.5.4. Pruebas de consumo desde la máquina atacante

- **Antes de bloqueo**

Se observa que antes de realizar las configuraciones de filtrado de red, una máquina atacante o cliente era capaz de consumir la API sin necesidad de pasar por el API GATEWAY, logrando evitar controles de seguridad aplicados en el componente mencionado.

Figura 50. *Consumo de la API sin necesidad de API GATEWAY.*

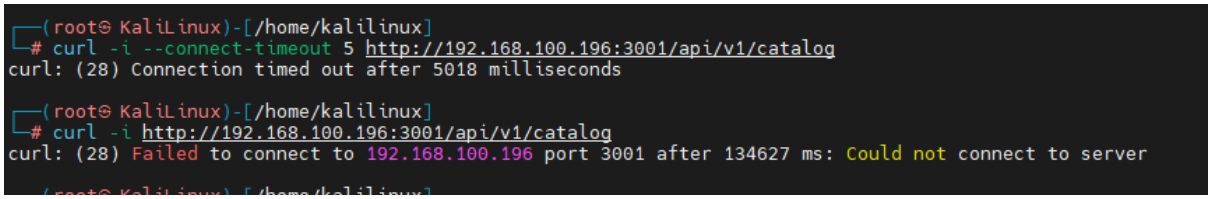
```
┌──(root㉿KaliLinux)-[/home/kalilinux]
└─# curl -i http://192.168.100.196:3001/api/v1/catalog
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 1141
ETag: W/"475-xmP/eCn7sFORXXnKjUt0mp6JenQ"
Date: Sun, 07 Jun 2026 09:41:23 GMT
Connection: keep-alive
Keep-Alive: timeout=5

[{"id":"1","name":"Laptop ASUS ROG Strix","price":1599,"stock":8},{"id":"2","name":"Monitor Samsung Odyssey G5 27\"","price":349,"stock":15},{"id":"3","name":"Teclado Mecánico
 Logitech G Pro X","price":129,"stock":25},{"id":"4","name":"Mouse Inalámbrico Razer DeathAdder V3","price":89,"stock":40},{"id":"5","name":"Smartphone Google Pixel 8 Pro","pr
ice":999,"stock":12},{"id":"6","name":"Tarjeta de Video RTX 4070 Ti","price":799,"stock":5},{"id":"7","name":"Procesador AMD Ryzen 7 7800X3D","price":399,"stock":18},{"id":"8"
,"name":"Memoria RAM Corsair Vengeance 32GB DDR5","price":115,"stock":30},{"id":"9","name":"Disco Duro SSD NVMe Kingston 2TB","price":145,"stock":50},{"id":"10","name":"Audifo
nos HyperX Cloud Alpha Wireless","price":159,"stock":14},{"id":"11","name":"Consola PlayStation 5 Slim","price":499,"stock":7},{"id":"12","name":"Memoria MicroSD SanDisk 256GB
 Ultra","price":25,"stock":100},{"id":"13","name":"Router TP-Link Archer AX55 Wi-Fi 6","price":99,"stock":22},{"id":"14","name":"Cámara Web Logitech Brio 4K","price":199,"stoc
k":10},{"id":"15","name":"Fuente de Poder Corsair RM850x 850W","price":139,"stock":12}]
```

Nota. Demostración empírica del vector de ataque Gateway Bypass mediante el consumo directo del microservicio de catálogo desde una estación de auditoría externa.

- **Después del bloqueo**

Una vez configuradas las reglas de restricción de tráfico, se observa que el consumo directo de las APIs no es posible, teniendo un error de Connection Time Out ya que, las APIS no responden a peticiones que no provengan del API Gateway. De este mod, se logra tener un control del tráfico que consume las APIs y se puede implementar reglas específicas para controlar ataques externos.

Figura 51. *Error de Time Out al consumir las APIS sin pasar por el API Gateway.*



Nota. Validación experimental de la mitigación de Gateway Bypass mediante la confirmación de expiración de conexión desde el host remoto.

La Figura 51 muestra que, al realizar solicitudes a las APIs mediante el API Gateway, esta responde con una respuesta exitosa.

Figura 52. *Consumo exitoso de la API mediante el componente API Gateway.*

Nota. Validación de la disponibilidad y consumo legítimo del catálogo a través de Kong API Gateway desde la estación Kali Linux.

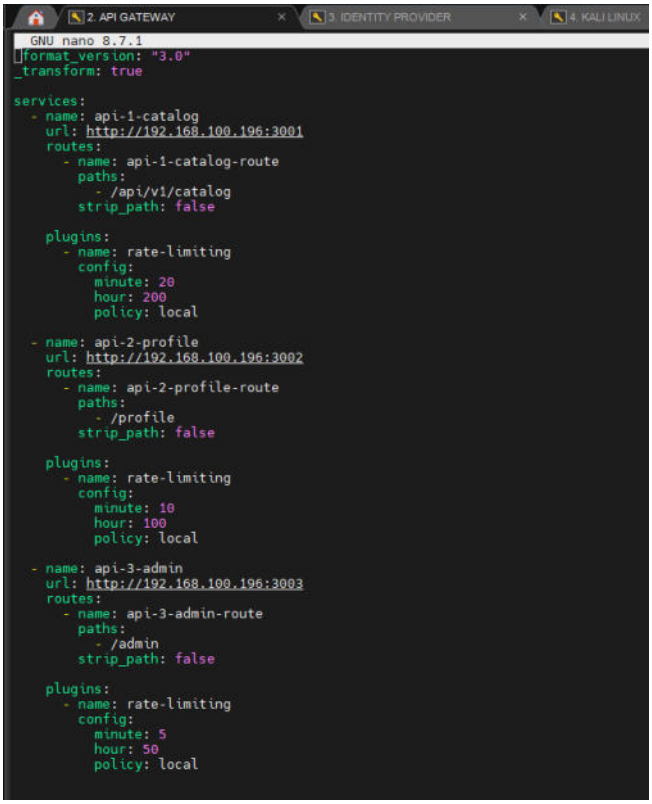
3.5.5.Configuración RATE LIMIT

La configuración de *rate limiting* se implementó en el API Gateway Kong con el objetivo de controlar la cantidad de solicitudes permitidas hacia las APIs internas. Este mecanismo permite reducir intentos de abuso, enumeración masiva de recursos y posibles escenarios de denegación de servicio a nivel de aplicación.

La política fue aplicada directamente en Kong, antes de que las solicitudes sean reenviadas a los microservicios. De esta manera, cuando un cliente supera el límite establecido para una API, el API Gateway rechaza la petición y devuelve el código HTTP 429 Too Many Requests, evitando que el tráfico excedente llegue al backend.

Como se observa en la Figura 52, la configuración del archivo kong.yml define límites diferenciados para cada microservicio. Para la API de catálogo se estableció un máximo de 20 solicitudes por minuto, para la API de perfil 10 solicitudes por minuto y para la API administrativa 5 solicitudes por minuto. Estos valores permiten aplicar controles más estrictos sobre los servicios que manejan información sensible o funciones privilegiadas.

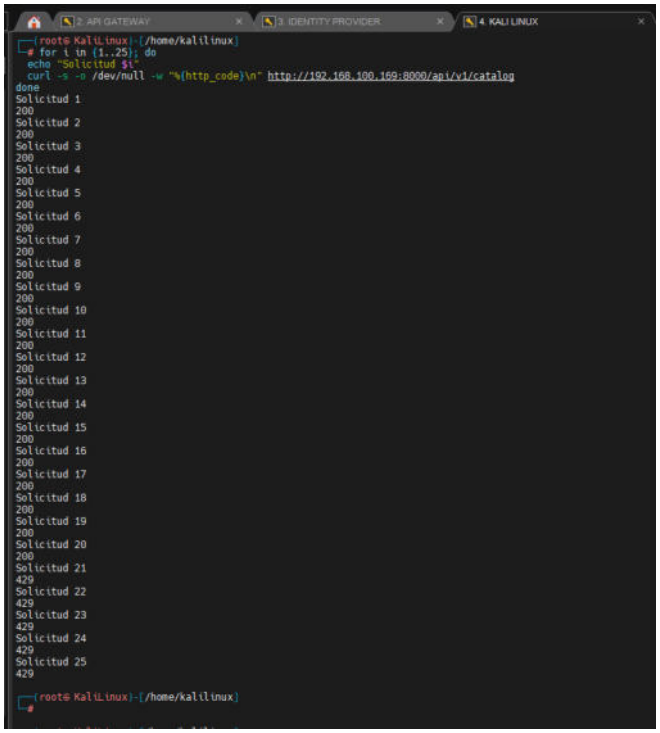
Figura 53. *Configuración Rate Limit.*



Nota. Manifiesto declarativo YAML de Kong API Gateway para la inyección perimetral de políticas de limitación de tasa (Rate Limiting).

La Figura 52 muestra la ejecución de un script para enviar una serie de 25 peticiones seguidas al microservicio. El rate limit configurado para el api 1 es un máximo de 20 solicitudes por minuto por lo que, como se observa en la Figura 53, las primeras 20 peticiones devuelven una respuesta exitosa, mientras que las últimas 5 al no ser aceptadas por el API Gateway no reciben ninguna respuesta y arrojan un error 429 Too many Request, conforme a lo configurado.

Figura 54. *Funcionamiento Rate Limit - API 1.*



Nota. Auditoría dinámica de denegación de servicio mediante ráfaga automatizada de peticiones y respuesta perimetral HTTP 429.

CAPÍTULO 4

4. Análisis de resultados

En este capítulo se presentan los resultados obtenidos a partir de las pruebas de concepto ejecutadas sobre el entorno desarrollado. Las evaluaciones tuvieron como finalidad medir la efectividad de los controles implementados para mitigar las vulnerabilidades identificadas en el escenario sin protección, particularmente aquellas relacionadas con acceso sin autenticación, BOLA, BFLA y abuso del consumo de recursos mediante solicitudes masivas.

Los resultados expuestos constituyen la validación experimental del modelo propuesto basado en OAuth 2.0, JWT, Keycloak y Kong API Gateway. Para ello, se emplearon pruebas automatizadas orientadas a cuantificar el comportamiento del entorno seguro frente a diferentes escenarios de uso y ataque, utilizando indicadores que permitieran comparar objetivamente el desempeño de los mecanismos de protección implementados.

4.1. Pruebas de Concepto

4.1.1. Índice de Eficacia Perimetral (IEP)

Con el propósito de cuantificar la efectividad de los mecanismos de protección implementados en el API Gateway, se definió el Índice de Eficacia Perimetral (IEP) como un indicador que relaciona la capacidad del entorno para bloquear solicitudes no autorizadas y permitir únicamente aquellas que cumplen con los criterios de autenticación y autorización establecidos.

Para obtener este indicador se ejecutaron pruebas automatizadas de consumo sobre las APIs protegidas, simulando tanto solicitudes legítimas como intentos de acceso que incumplían las políticas definidas. Las pruebas permitieron evaluar el comportamiento del Gateway frente a escenarios de abuso, superación de límites de consumo y peticiones sin credenciales válidas.

Como apoyo a la automatización del proceso de medición, se desarrollaron cinco scripts encargados de generar solicitudes HTTP, parametrizar los escenarios de prueba, configurar los

umbrales de consumo evaluados y calcular automáticamente el valor del indicador. Sin embargo, el objetivo principal de esta sección no es describir la implementación de dichos scripts, sino analizar los resultados obtenidos y determinar el grado de efectividad alcanzado por los controles perimetrales propuestos.

Tabla 9.

Descripción de los scripts empleados para la evaluación del Índice de Eficacia Perimetral (IEP).

Script	**Objetivo**	**Herramienta**	**Entrada**	**Salida esperada**	**Cantidad de solicitudes**	**Criterio de éxito**
00_config.sh	Centralizar parámetros de prueba	Bash	URL de Kong, endpoints, rate limit, archivo CSV	Variables cargadas para los demás scripts	No aplica	Configuración disponible sin errores
01_inicializar_resultados.sh	Crear el archivo de resultados	Bash	Nombre del archivo CSV	Archivo resultados_iep.csv con cabeceras	No aplica	Archivo creado correctamente
02_prueba_401_tokens_invalidos.sh	Medir bloqueo de tokens inválidos	Bash, cURL	Tokens malformados, inválidos o simulados	Respuestas HTTP 401	5 solicitudes	Todas deben ser bloqueadas
03_precalentar_rate_limiting.sh	Consumir el umbral permitido del rate limit	Bash, cURL	Endpoint de catálogo y límite configurado	Solicitudes previas aceptadas	20 solicitudes	Preparar el escenario sin contar en el IEP
04_prueba_429_rate_limiting.sh	Medir bloqueo por exceso de solicitudes	Bash, cURL	Solicitudes posteriores al umbral	Respuestas HTTP 429	15 solicitudes	Solicitudes excedentes bloqueadas
05_calcular_iep.sh	Calcular el indicador final	Bash, AWK	Archivo resultados_iep.csv	Porcentaje IEP	Total registrado	IEP calculado correctamente

Fuente: Elaboración propia

El cálculo del IEP se realizó a partir de las solicitudes rechazadas correctamente por el API Gateway frente a dos escenarios principales: tokens inválidos y exceso de solicitudes. Para ello, los scripts registraron los códigos HTTP obtenidos en cada prueba y consolidaron los resultados en un archivo CSV. Posteriormente, el script de cálculo procesó estos datos para obtener un valor porcentual que representa la eficacia del perímetro de seguridad para bloquear tráfico no autorizado o abusivo antes de que llegue a los microservicios internos.

La Figura 54 presenta el archivo de configuración general utilizado durante la ejecución de las pruebas automatizadas del IEP. En este archivo se definieron los endpoints protegidos por Kong, los límites de solicitudes configurados y los parámetros requeridos para garantizar la reproducibilidad de las mediciones realizadas. Su función fue centralizar la parametrización del entorno de prueba.

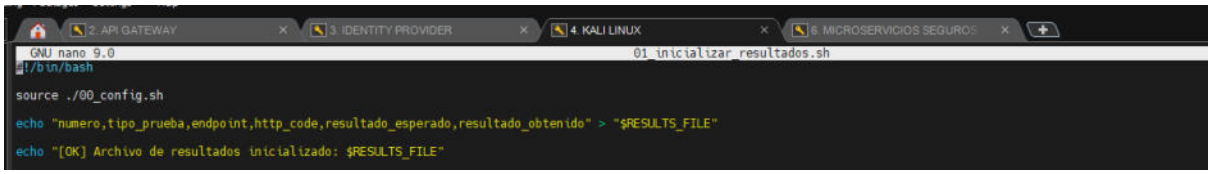
Figura 55. *Archivo de configuración para pruebas IEP.*

Nota. Script de parametrización global en Bash para la automatización de auditorías dinámicas y recolección de métricas perimetrales.

La Figura 55 muestra la ejecución del script encargado de inicializar el archivo de almacenamiento utilizado durante las pruebas del Índice de Eficacia Perimetral (IEP). Este procedimiento creó el archivo `resultados_iep.csv` e incorporó la estructura de datos necesaria para registrar de forma organizada los resultados obtenidos en cada escenario evaluado.

La consolidación de esta información permitió disponer de un repositorio único para almacenar los códigos de respuesta HTTP, el tipo de prueba ejecutada y el resultado obtenido, facilitando posteriormente el procesamiento automático y el cálculo del indicador IEP.

Figura 56. *Archivo de preparación de almacenamiento de los datos de prueba.*

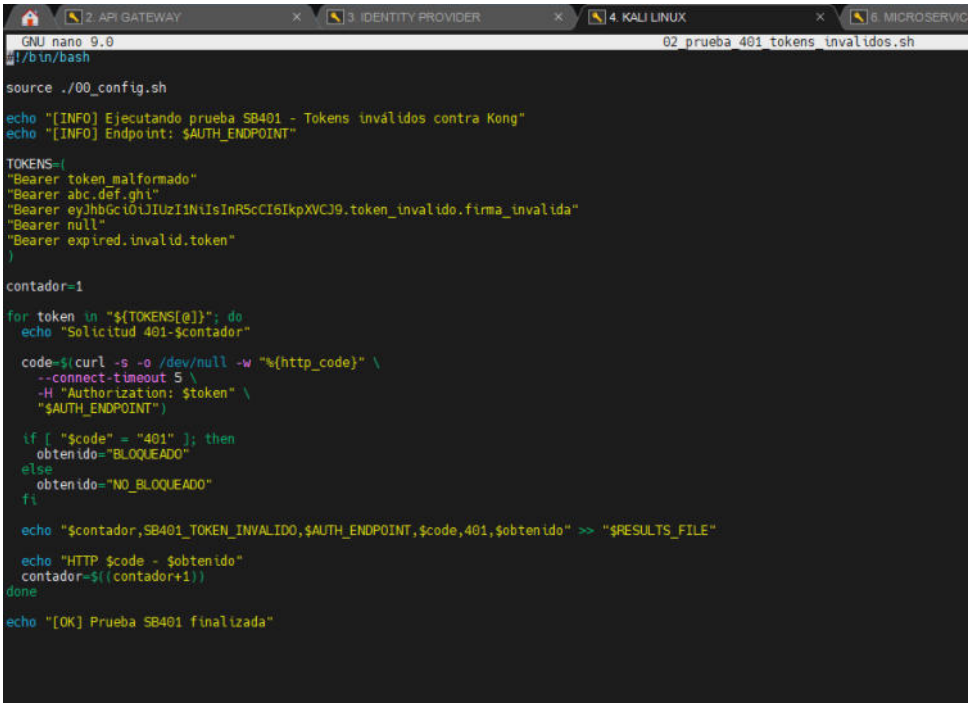


Nota. Script de inicialización de entornos lógicos y estructuración de cabeceras CSV para el repositorio de métricas experimentales.

La Figura 56 muestra la ejecución de la prueba orientada a validar la capacidad del API Gateway para bloquear solicitudes que presentan credenciales inválidas. Para ello, se realizaron cinco solicitudes HTTP utilizando diferentes tipos de tokens no válidos, incluyendo tokens malformados, alterados, con firmas inválidas, nulos y con estructura incorrecta.

En cada caso, se registró el código de respuesta devuelto por Kong, con el propósito de verificar que las peticiones fueran rechazadas antes de alcanzar los microservicios internos. El criterio de éxito establecido para esta prueba consistió en obtener respuestas HTTP 401 Unauthorized en la totalidad de las solicitudes ejecutadas, evidenciando el correcto funcionamiento de los mecanismos de validación de autenticación implementados.

Figura 57. *Archivo para ejecutar pruebas de token inválido.*



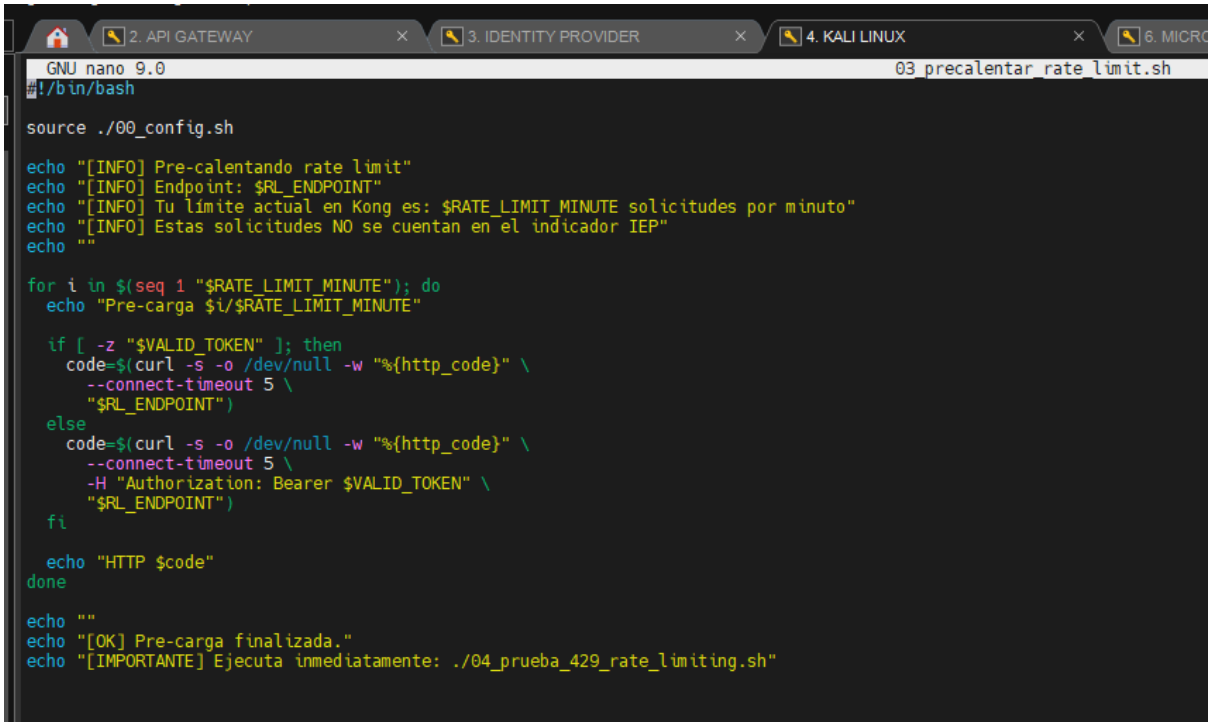
Nota. Script de automatización en Bash para la inyección secuencial de tokens de acceso inválidos y registro de exclusión perimetral.

La Figura 57 muestra la ejecución del script de precalentamiento del mecanismo de *rate limiting*. Durante esta fase se realizaron 20 solicitudes HTTP preparatorias, equivalentes al umbral máximo permitido por minuto configurado en el API Gateway para el endpoint evaluado.

Estas peticiones tuvieron como único propósito consumir el límite establecido y generar las condiciones necesarias para la prueba posterior de bloqueo por exceso de solicitudes. Por esta razón, sus resultados no fueron considerados en el cálculo del Índice de Eficacia Perimetral (IEP).

Únicamente las solicitudes ejecutadas posteriormente mediante el script 04_prueba_429_rate_limiting.sh, orientadas a verificar la generación de respuestas HTTP 429 Too Many Requests, fueron incluidas dentro del cálculo final del indicador.

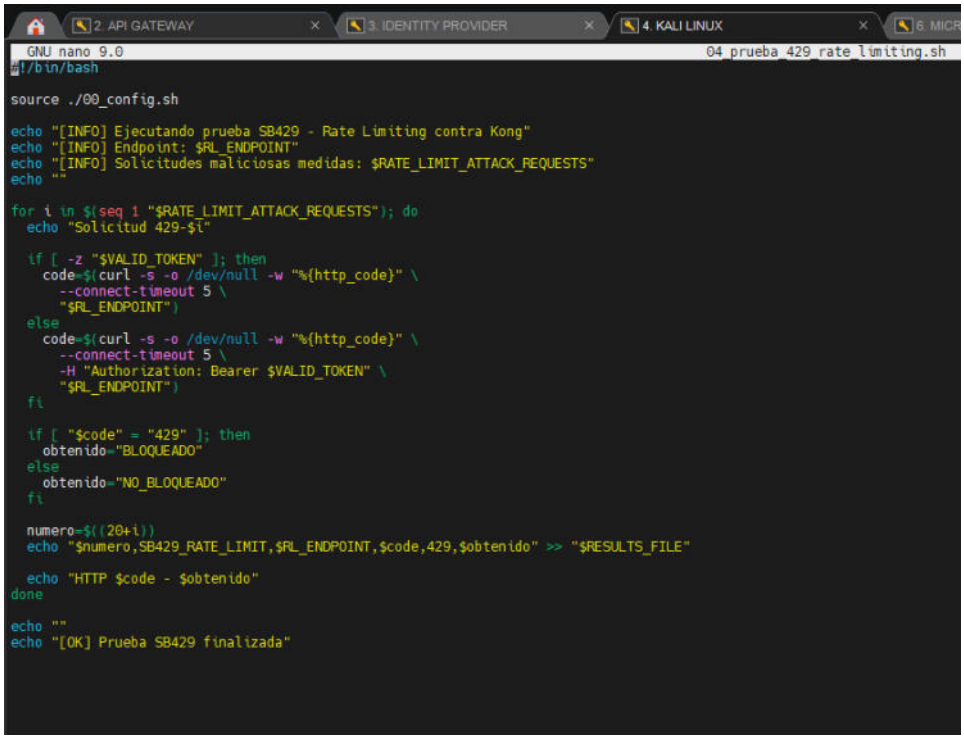
Figura 58. *Archivo precalentar rate limiting.*



Nota. Script de automatización en Bash para la saturación controlada del umbral de peticiones por minuto en la puerta de enlace.

La Figura 58 muestra la ejecución de solicitudes posteriores al límite permitido. El rechazo de estas peticiones mediante respuestas HTTP 429 confirmó la capacidad del API Gateway para mitigar intentos de abuso y consumo excesivo de recursos antes de alcanzar los microservicios internos.

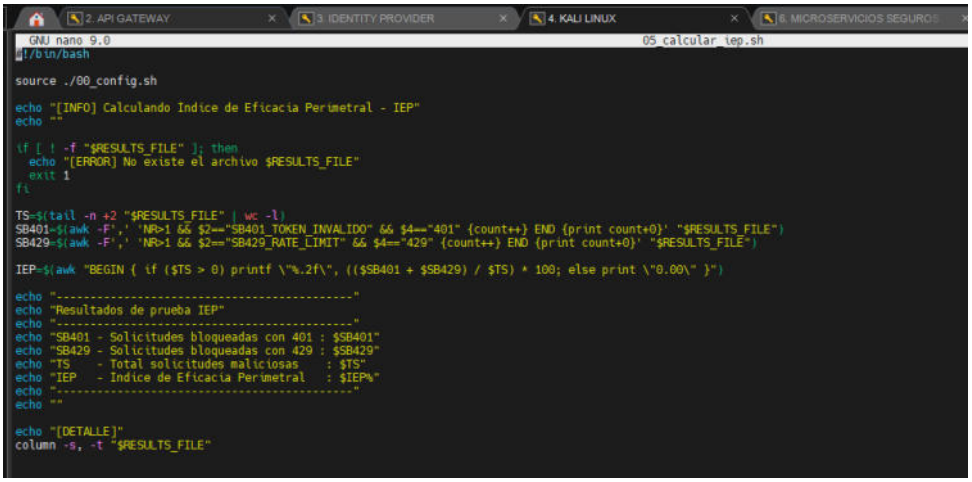
Figura 59. *Archivo para ejecutar pruebas de rate limit.*



Nota. Script de automatización en Bash para la verificación empírica de mitigación por Consumo Irrestricto de Recursos y registro de estados HTTP 429.

La Figura 59 evidencia el procesamiento automatizado de los resultados registrados durante las pruebas. A partir del total de solicitudes maliciosas evaluadas y del número de peticiones bloqueadas correctamente, se obtuvo el valor final del Índice de Eficacia Perimetral, utilizado para cuantificar la efectividad de los controles implementados.

Figura 60. *Archivo de procesamiento de resultados.*



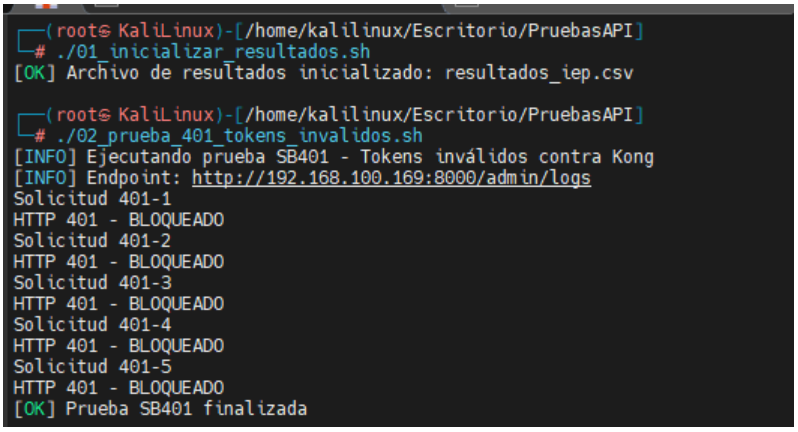
Nota. Script de automatización en Bash para el cálculo analítico del Índice de Eficacia Perimetral y tabulación de métricas de seguridad.

Los resultados obtenidos evidencian que el API Gateway fue capaz de identificar y bloquear solicitudes que incumplían las políticas de seguridad definidas. Las peticiones realizadas con tokens inválidos fueron rechazadas mediante respuestas HTTP 401, mientras que aquellas que excedieron el umbral configurado fueron bloqueadas con respuestas HTTP 429. Estos resultados permitieron calcular el Índice de Eficacia Perimetral, proporcionando una medida cuantitativa de la capacidad del perímetro de seguridad para filtrar tráfico no autorizado antes de que alcanzara los microservicios internos.

4.1.2. Ejecución de pruebas para IEP

En la Figura 60 se observa la ejecución de la prueba de validación de token, en la cual se realizó un ataque con 5 peticiones todas con tokens inválidos, observando que el API Gateway detecta las 5 peticiones y envía un mensaje 401 correspondiente a un acceso no autorizado.

Figura 61. *Ataque con credenciales no autorizadas y token inválido.*



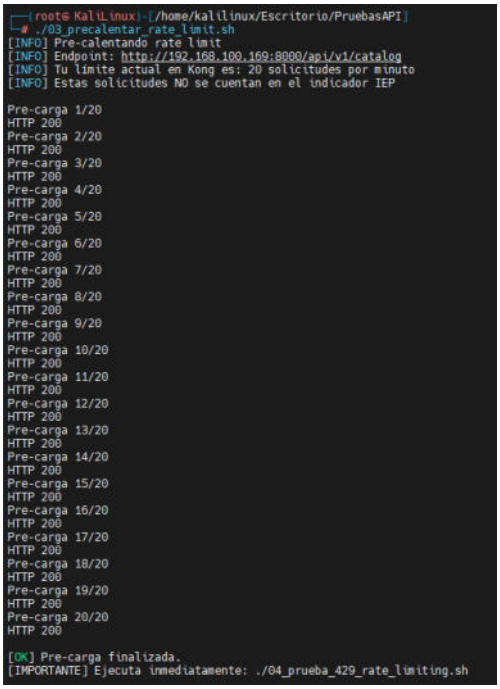
Nota. Validación del control de autenticación implementado mediante OAuth 2.0 y JWT frente a intentos de acceso con tokens inválidos, mitigando escenarios asociados a la vulnerabilidad OWASP API2:2023 Broken Authentication.

Continuando con las pruebas, se ejecutó el script encargado de preparar el escenario para la evaluación del mecanismo de rate limiting. Como se observa en la Figura 61, se enviaron 20 solicitudes HTTP al microservicio de catálogo, valor que corresponde al umbral máximo de peticiones por minuto configurado en el API Gateway para dicho servicio. Debido a que estas solicitudes se encontraban dentro del límite permitido, todas fueron procesadas exitosamente y recibieron respuestas válidas.

Es importante destacar que estas 20 solicitudes tuvieron un carácter preparatorio y no fueron consideradas dentro del cálculo del Índice de Eficacia Perimetral (IEP). Su finalidad fue consumir la cuota autorizada por Kong y generar las condiciones necesarias para la prueba posterior.

En consecuencia, únicamente las solicitudes adicionales ejecutadas después de alcanzar este umbral, orientadas a verificar el bloqueo mediante respuestas HTTP 429 Too Many Requests, fueron utilizadas para evaluar la efectividad del control e incorporadas al cálculo final del indicador.

Figura 62. *Script de preparación de umbral para pruebas de rate limiting.*

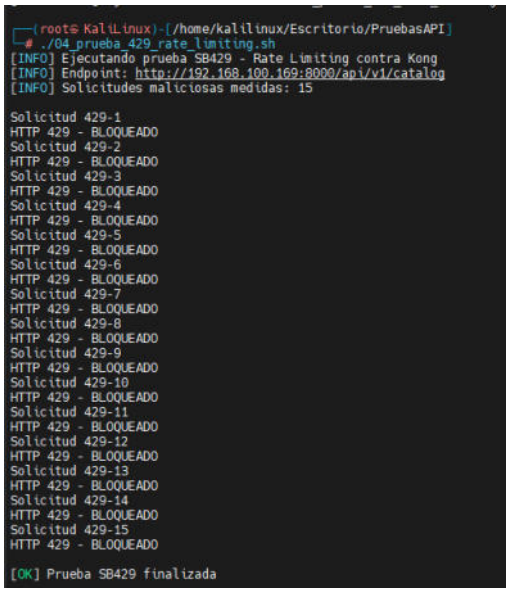


Nota. Ejecución de solicitudes preparatorias destinadas a consumir el umbral permitido por el API Gateway. Estas peticiones no forman parte del cálculo del IEP y únicamente preparan el escenario para la validación del control de rate limiting.

La Figura 62 presenta los resultados obtenidos durante la validación del mecanismo de rate limiting implementado en el API Gateway. Una vez consumido el umbral permitido de 20 solicitudes por minuto mediante el proceso de precalentamiento, se ejecutaron 15 solicitudes HTTP adicionales hacia el mismo microservicio con el propósito de evaluar la capacidad de bloqueo del control configurado.

Los resultados evidenciaron que las 15 solicitudes excedentes fueron rechazadas exitosamente, recibiendo respuestas HTTP 429 Too Many Requests. Este comportamiento confirmó que Kong aplicó correctamente la política de limitación de tasa, evitando que el tráfico excesivo alcanzara los microservicios internos y mitigando escenarios asociados al consumo indiscriminado de recursos.

Figura 63. *Cierre operativo de las pruebas de limitación de tasa (Rate Limiting).*



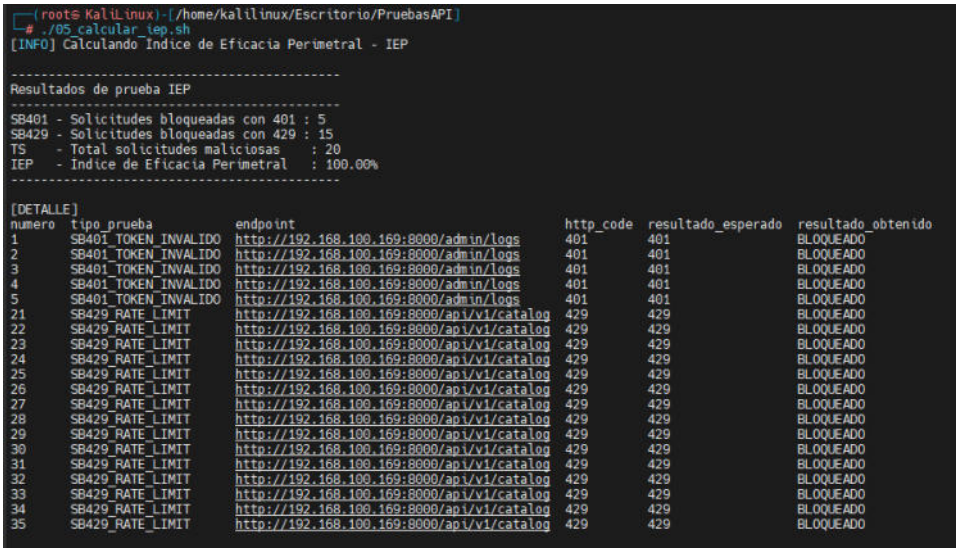
Nota. Se ejecutaron 15 solicitudes posteriores al umbral permitido, obteniéndose respuestas HTTP 429 en la totalidad de los casos, lo que evidenció el correcto funcionamiento del mecanismo de rate limiting implementado en Kong.

La Figura 63 presenta el procesamiento consolidado de los resultados obtenidos durante las pruebas del Índice de Eficacia Perimetral (IEP). Para el cálculo del indicador se consideraron únicamente las solicitudes orientadas a evaluar la efectividad de los controles implementados, excluyendo las peticiones preparatorias utilizadas para el precalentamiento del mecanismo de rate limiting.

Los resultados evidenciaron que se ejecutaron 20 solicitudes maliciosas, de las cuales 5 correspondieron a intentos de acceso utilizando tokens inválidos (SB401) y 15 a solicitudes que excedieron el umbral permitido de peticiones (SB429). En ambos escenarios, el API Gateway bloqueó la totalidad de las solicitudes evaluadas, devolviendo respuestas HTTP 401 Unauthorized y HTTP 429 Too Many Requests, respectivamente.

A partir de estos resultados, se obtuvo un Índice de Eficacia Perimetral (IEP) del 100 %, lo que demuestra que los controles implementados en Kong fueron efectivos para impedir que solicitudes no autorizadas o abusivas alcanzaran los microservicios internos. Estos hallazgos evidencian la capacidad del perímetro de seguridad para mitigar escenarios asociados a autenticación deficiente y consumo excesivo de recursos, contribuyendo al fortalecimiento de la protección de las APIs desplegadas.

Figura 64. *Resultado de IEP.*



```
┌──(root㉿KaliLinux)-[/home/kalilinux/Escritorio/PruebasAPI]
└─# ./05_calcular_iep.sh
[INFO] Calculando Indice de Eficacia Perimetral - IEP

-----------------------------------------
Resultados de prueba IEP
-----------------------------------------
SB401 - Solicitudes bloqueadas con 401 : 5
SB429 - Solicitudes bloqueadas con 429 : 15
TS    - Total solicitudes maliciosas   : 20
IEP   - Indice de Eficacia Perimetral  : 100.00%
-----------------------------------------

[DETALLE]
numero  tipo_prueba          endpoint                                  http_code  resultado_esperado  resultado_obtenido
1       SB401_TOKEN_INVALIDO http://192.168.100.169:8000/admin/logs    401        401                 BLOQUEADO
2       SB401_TOKEN_INVALIDO http://192.168.100.169:8000/admin/logs    401        401                 BLOQUEADO
3       SB401_TOKEN_INVALIDO http://192.168.100.169:8000/admin/logs    401        401                 BLOQUEADO
4       SB401_TOKEN_INVALIDO http://192.168.100.169:8000/admin/logs    401        401                 BLOQUEADO
5       SB401_TOKEN_INVALIDO http://192.168.100.169:8000/admin/logs    401        401                 BLOQUEADO
21      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
22      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
23      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
24      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
25      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
26      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
27      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
28      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
29      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
30      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
31      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
32      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
33      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
34      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
35      SB429_RATE_LIMIT     http://192.168.100.169:8000/api/v1/catalog 429        429                 BLOQUEADO
```

Nota. El cálculo del IEP se realizó sobre 20 solicitudes maliciosas evaluadas: 5 asociadas a tokens inválidos (HTTP 401) y 15 relacionadas con exceso de solicitudes (HTTP 429). La totalidad de las peticiones fueron bloqueadas correctamente, obteniéndose un IEP de 100 %.

4.2. Análisis del Índice de Eficacia de la Autorización Lógica (IEA)

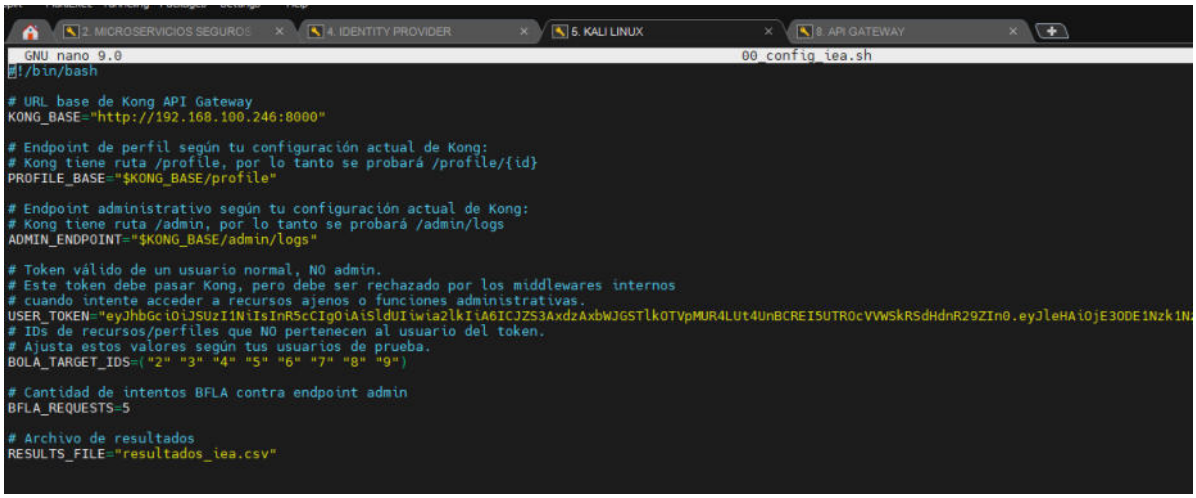
Con el propósito de evaluar la capacidad de los mecanismos internos de autorización implementados en los microservicios seguros, se desarrolló un conjunto de pruebas automatizadas orientadas a medir el Índice de Eficacia de la Autorización Lógica (IEA). Este indicador permite determinar la efectividad de los middlewares de aplicación para detectar y bloquear intentos de acceso no autorizados que hayan superado previamente la validación perimetral del API Gateway.

Para la ejecución de las pruebas se utilizó una máquina atacante basada en Kali Linux, desde la cual se configuró el script config_iea.sh. Este script definió como objetivo la dirección del API Gateway desplegado en Kong, utilizando la URL base http://192.168.100.246:8000. Asimismo, se establecieron dos endpoints de evaluación: el endpoint /profile, destinado a validar ataques relacionados con BOLA, y el endpoint /admin/logs, utilizado para evaluar escenarios de BFLA.

4.2.1. Configuración del entorno de pruebas

La Figura 65 muestra la configuración del script config_iea.sh, donde se establecen los parámetros necesarios para la ejecución de las pruebas. Entre ellos destacan la dirección del API Gateway, los endpoints sometidos a evaluación, el token JWT asociado a un usuario sin privilegios administrativos, el conjunto de identificadores utilizados para las pruebas BOLA y el número de intentos definidos para la simulación de ataques BFLA.

Figura 65. *Configuración del script config_iea.sh para la ejecución automatizada de pruebas IEA.*



Nota: Se muestran los parámetros utilizados durante la evaluación del Índice de Eficacia de la Autorización Lógica, incluyendo la URL del API Gateway, los endpoints evaluados, el token JWT de

usuario estándar, los identificadores empleados en las pruebas BOLA y la cantidad de intentos configurados para las pruebas BFLA. Elaboración propia.

4.2.2. Inicialización del registro de resultados del IEA

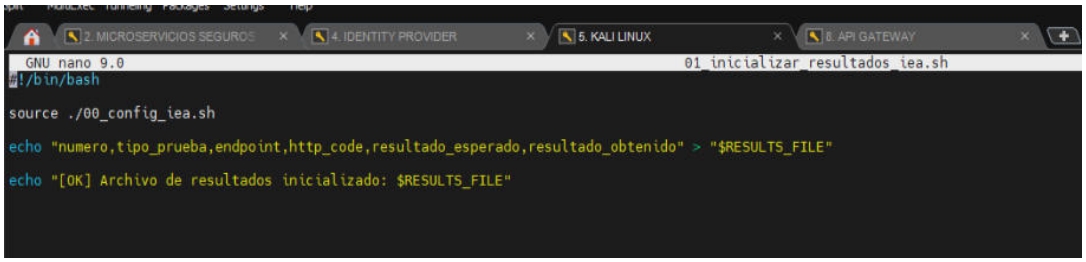
Previo a la ejecución de las pruebas destinadas a evaluar el Índice de Eficacia de la Autorización Lógica (IEA), se implementó un mecanismo automatizado para el registro estructurado de los resultados obtenidos durante cada intento de explotación. Para ello, se desarrolló el script denominado `01_inicializar_resultados_iea.sh`, cuya función consistió en crear e inicializar el archivo donde posteriormente se almacenaría la evidencia generada por las pruebas de autorización.

El script incorporó la carga de los parámetros definidos previamente en el archivo de configuración `00_config_iea.sh`, permitiendo reutilizar variables como el nombre del archivo de resultados y mantener una estructura modular del proceso de evaluación. Una vez cargada la configuración, el sistema generó un archivo en formato CSV denominado `resultados_iea.csv`, incorporando una cabecera con los campos requeridos para el análisis posterior.

La estructura definida para el registro incluyó las siguientes columnas: número secuencial de la prueba ejecutada, tipo de vulnerabilidad evaluada, endpoint objetivo, código HTTP devuelto por el sistema, resultado esperado y resultado obtenido. Esta organización permitió documentar de manera ordenada cada intento realizado contra la arquitectura segura y facilitó la posterior comparación entre el comportamiento esperado y la respuesta efectiva del sistema.

La utilización de un archivo CSV como repositorio de evidencias permitió centralizar la información generada durante las pruebas automatizadas, facilitando tanto el cálculo de indicadores cuantitativos como la trazabilidad de los eventos registrados. De esta manera, cada solicitud enviada durante los escenarios BOLA y BFLA quedó debidamente documentada, garantizando la reproducibilidad del experimento y la transparencia en el análisis de resultados.

Figura 66. *Inicialización del archivo de resultados de las pruebas IEA.*

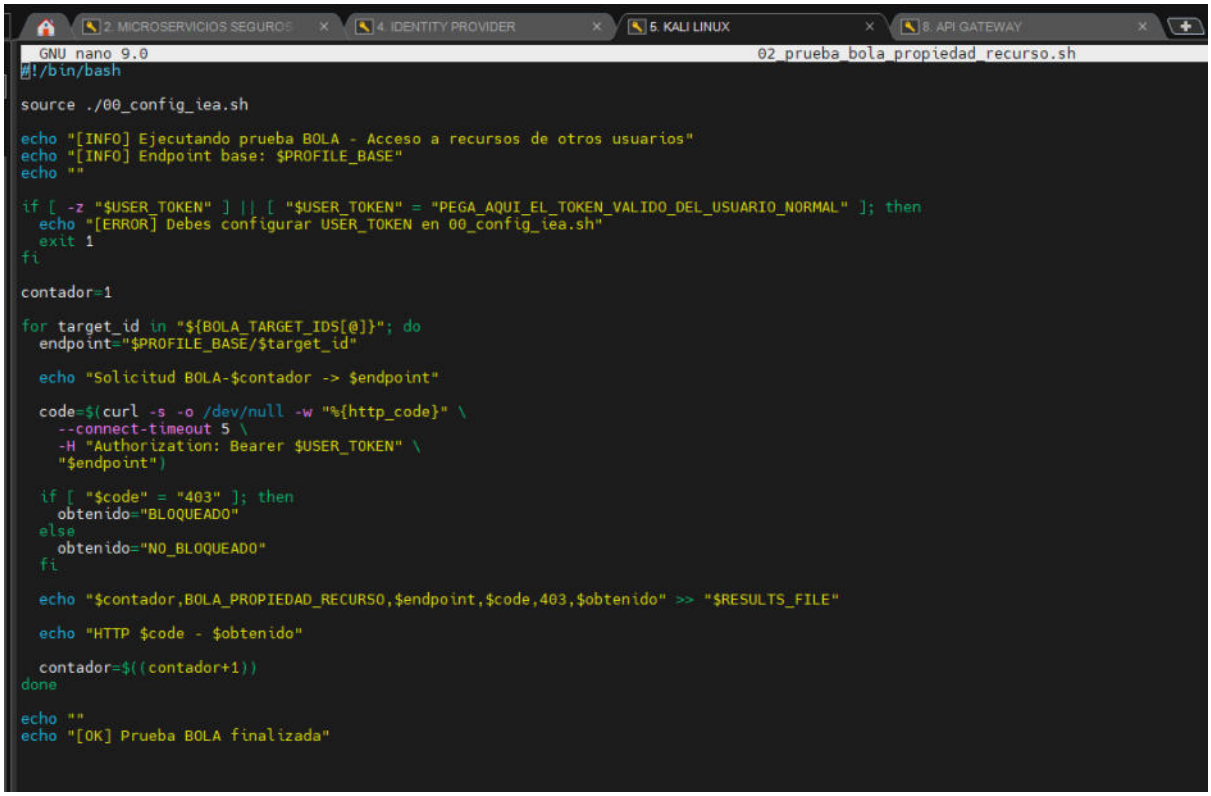


Nota: El script genera el archivo `resultados_iea.csv`, incorporando la estructura necesaria para registrar el número de prueba, vulnerabilidad evaluada, endpoint objetivo, código HTTP obtenido y resultados esperados.

4.2.3. Pruebas BOLA

Con el objetivo de simular intentos de acceso a recursos pertenecientes a otros usuarios, se desarrolló un script automatizado encargado de modificar secuencialmente los identificadores del endpoint /profile/{id}.

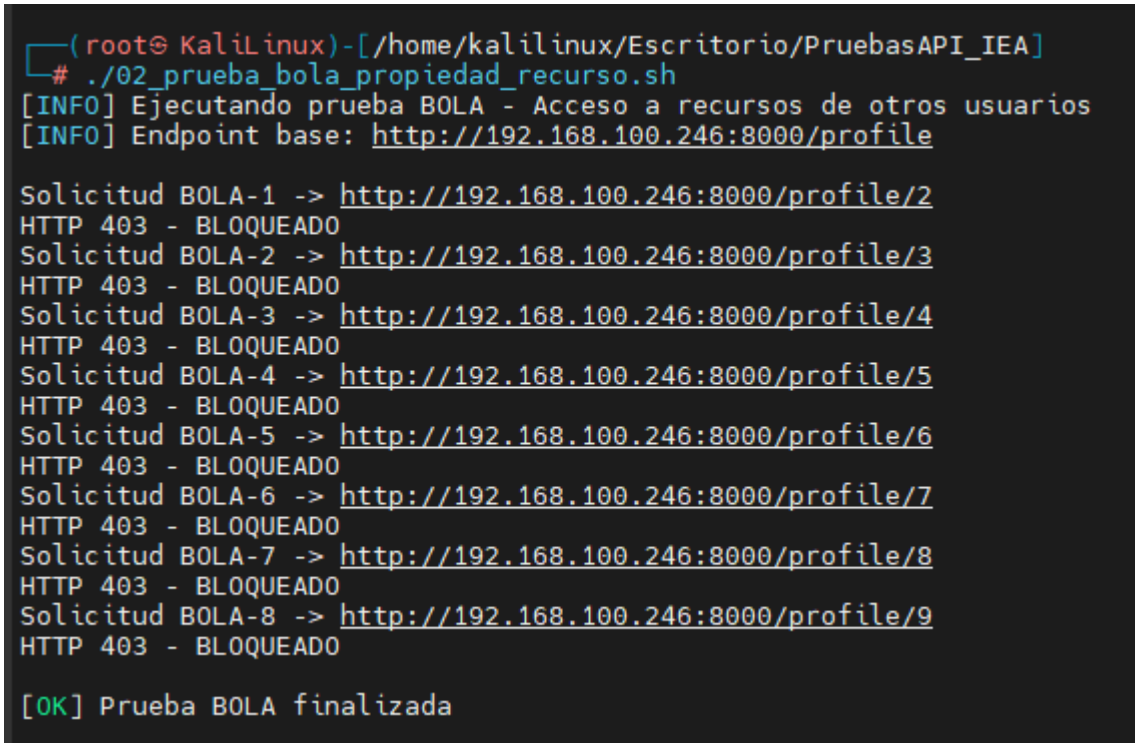
Figura 67. *Script automatizado para la ejecución de pruebas BOLA.*



Nota. El script utiliza un token JWT válido de un usuario estándar para realizar solicitudes dirigidas a recursos cuya propiedad corresponde a otros usuarios, registrando automáticamente las respuestas del sistema.

Se realizaron ocho intentos consecutivos de acceso no autorizado utilizando identificadores distintos al propietario legítimo del recurso.

Figura 68. *Resultados obtenidos durante la ejecución de las pruebas BOLA.*

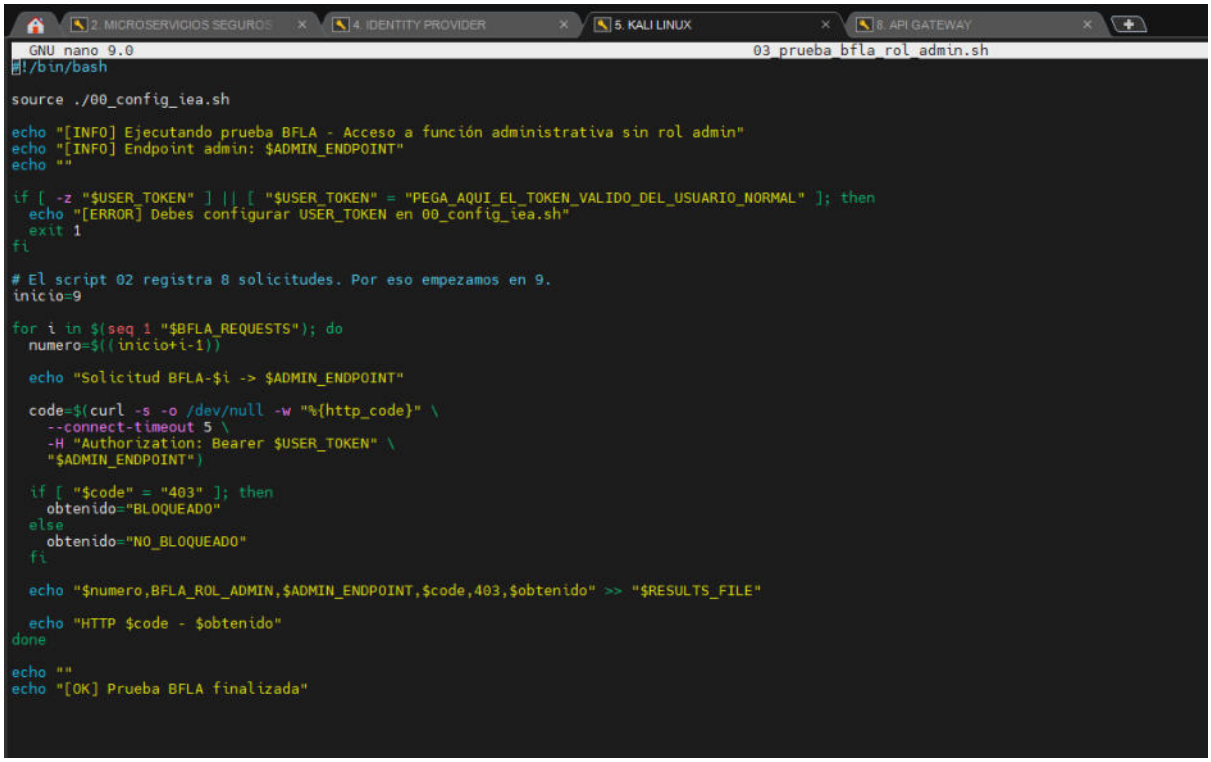


Nota. Todas las solicitudes dirigidas a recursos ajenos fueron rechazadas mediante respuestas HTTP 403 (Forbidden), evidenciando la correcta validación de la propiedad del recurso por parte de los middlewares internos. Elaboración propia.

4.2.4. Pruebas BFLA

Para evaluar la resistencia frente a intentos de escalamiento de privilegios, se desarrolló un script automatizado orientado a consumir funciones administrativas mediante un token sin privilegios de administrador.

Figura 69. *Script automatizado para la ejecución de pruebas BFLA*



Nota. El script realiza solicitudes consecutivas al endpoint administrativo utilizando un token JWT perteneciente a un usuario estándar y registra las respuestas obtenidas durante cada intento. Elaboración propia.

Los primeros intentos fueron bloqueados mediante respuestas HTTP 403, mientras que las solicitudes posteriores activaron la política de limitación de tráfico configurada en Kong.

Figura 70. *Resultado del Índice de Eficacia de Autorización (IEA).*

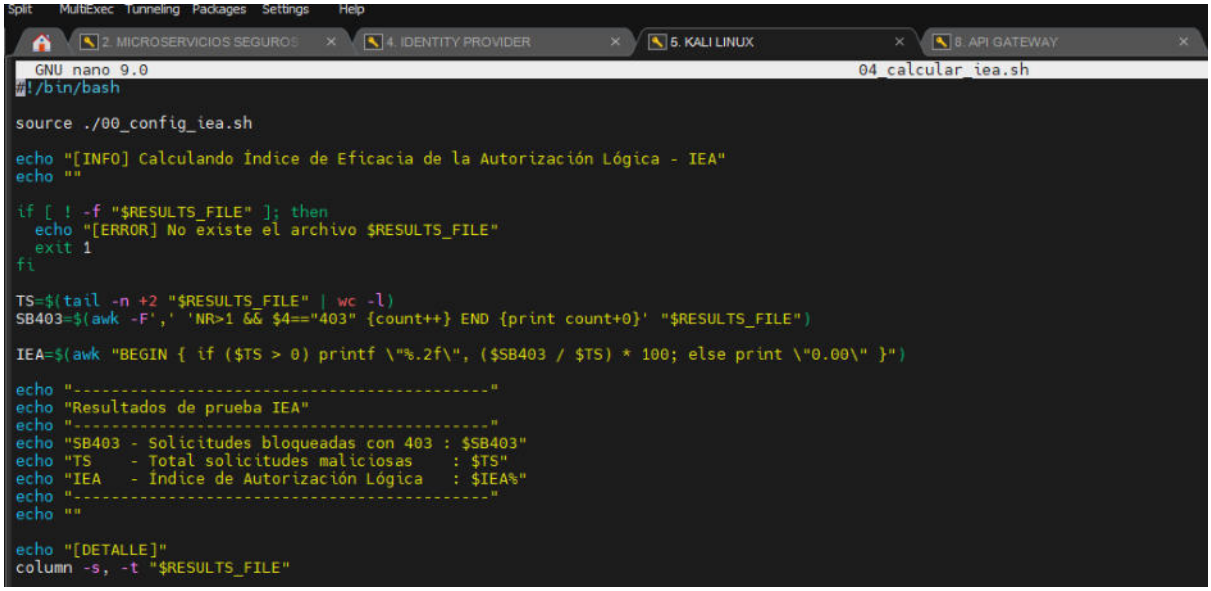
```
┌──(root㉿KaliLinux)-[/home/kalilinux/Escritorio/PruebasAPI_IEA]
└─# ./03_prueba_bfla_rol_admin.sh
[INFO] Ejecutando prueba BFLA - Acceso a función administrativa sin rol admin
[INFO] Endpoint admin: http://192.168.100.246:8000/admin/logs

Solicitud BFLA-1 -> http://192.168.100.246:8000/admin/logs
HTTP 403 - BLOQUEADO
Solicitud BFLA-2 -> http://192.168.100.246:8000/admin/logs
HTTP 403 - BLOQUEADO
Solicitud BFLA-3 -> http://192.168.100.246:8000/admin/logs
HTTP 403 - BLOQUEADO
Solicitud BFLA-4 -> http://192.168.100.246:8000/admin/logs
HTTP 403 - BLOQUEADO
Solicitud BFLA-5 -> http://192.168.100.246:8000/admin/logs
HTTP 403 - BLOQUEADO
Solicitud BFLA-6 -> http://192.168.100.246:8000/admin/logs
HTTP 429 - NO_BLOQUEADO
Solicitud BFLA-7 -> http://192.168.100.246:8000/admin/logs
HTTP 429 - NO_BLOQUEADO

[OK] Prueba BFLA finalizada
```

Nota. Se ejecutaron siete solicitudes hacia el endpoint administrativo /admin/logs utilizando un usuario sin privilegios administrativos. Seis solicitudes fueron bloqueadas mediante HTTP 403 por el control RBAC y una solicitud adicional recibió HTTP 429 debido a la activación del mecanismo de *rate limiting* del API Gateway.

4.2.5. Recalculo del Índice de Eficacia de la Autorización Lógica (IEA)

Dado que el objetivo del Índice de Eficacia de la Autorización Lógica (IEA) consiste en medir exclusivamente la efectividad de los controles internos encargados de validar privilegios y propiedad de los recursos, se procedió a recalcular el indicador considerando únicamente aquellas solicitudes cuya respuesta correspondió a HTTP 403 Forbidden, asociadas directamente a decisiones de autorización.

Figura 71. *Script de cálculo del Índice de Eficacia de la Autorización Lógica (IEA).*



Nota. Script en Bash encargado de procesar el archivo de resultados y calcular automáticamente el IEA a partir de las respuestas HTTP 403 obtenidas durante las pruebas BOLA y BFLA.

La Figura 71 presenta el archivo de resultados generado durante la ejecución de las pruebas automatizadas del IEA. En una primera ejecución se registraron quince solicitudes maliciosas, obteniéndose trece respuestas HTTP 403 y dos respuestas HTTP 429. Esto produjo un valor preliminar del indicador equivalente al 86,67 %.

Figura 72. *Resultado preliminar del Índice de Eficacia de la Autorización Lógica (IEA).*

```
┌──(root㉿KaliLinux)-[/home/kalilinux/Escritorio/PruebasAPI_IEA]
└─# ./04_calcular_iea.sh
[INFO] Calculando Índice de Eficacia de la Autorización Lógica - IEA

-----------------------------------------
Resultados de prueba IEA
-----------------------------------------
SB403 - Solicitudes bloqueadas con 403 : 13
TS    - Total solicitudes maliciosas    : 15
IEA   - Índice de Autorización Lógica   : 86.67%
-----------------------------------------

[DETALLE]
numero  tipo_prueba            endpoint                                  http_code  resultado_esperado  resultado_obtenido
1       BOLA_PROPIEDAD_RECURSO  http://192.168.100.246:8000/profile/2     403        403                 BLOQUEADO
2       BOLA_PROPIEDAD_RECURSO  http://192.168.100.246:8000/profile/3     403        403                 BLOQUEADO
3       BOLA_PROPIEDAD_RECURSO  http://192.168.100.246:8000/profile/4     403        403                 BLOQUEADO
4       BOLA_PROPIEDAD_RECURSO  http://192.168.100.246:8000/profile/5     403        403                 BLOQUEADO
5       BOLA_PROPIEDAD_RECURSO  http://192.168.100.246:8000/profile/6     403        403                 BLOQUEADO
6       BOLA_PROPIEDAD_RECURSO  http://192.168.100.246:8000/profile/7     403        403                 BLOQUEADO
7       BOLA_PROPIEDAD_RECURSO  http://192.168.100.246:8000/profile/8     403        403                 BLOQUEADO
8       BOLA_PROPIEDAD_RECURSO  http://192.168.100.246:8000/profile/9     403        403                 BLOQUEADO
9       BFLA_ROL_ADMIN          http://192.168.100.246:8000/admin/logs    403        403                 BLOQUEADO
10      BFLA_ROL_ADMIN          http://192.168.100.246:8000/admin/logs    403        403                 BLOQUEADO
11      BFLA_ROL_ADMIN          http://192.168.100.246:8000/admin/logs    403        403                 BLOQUEADO
12      BFLA_ROL_ADMIN          http://192.168.100.246:8000/admin/logs    403        403                 BLOQUEADO
13      BFLA_ROL_ADMIN          http://192.168.100.246:8000/admin/logs    403        403                 BLOQUEADO
14      BFLA_ROL_ADMIN          http://192.168.100.246:8000/admin/logs    429        403                 NO_BLOQUEADO
15      BFLA_ROL_ADMIN          http://192.168.100.246:8000/admin/logs    429        403                 NO_BLOQUEADO
```

Nota. El cálculo inicial del IEA evidenció 13 bloqueos exitosos mediante HTTP 403 sobre un total de 15 solicitudes evaluadas, obteniéndose un valor de 86,67 %. Las respuestas HTTP 429 fueron descartadas del cálculo final al corresponder a controles perimetrales

Se realizaron nuevamente las pruebas orientadas a validar exclusivamente los mecanismos de autorización lógica implementados en los microservicios. En el escenario BOLA se realizaron ocho intentos consecutivos de acceso a recursos pertenecientes a otros usuarios utilizando un token JWT válido asociado a un usuario sin privilegios especiales.

Como se observa en la Figura 72, las ocho solicitudes fueron rechazadas mediante respuestas HTTP 403 Forbidden, evidenciando que el middleware encargado de verificar la propiedad del recurso impidió exitosamente el acceso a información ajena.

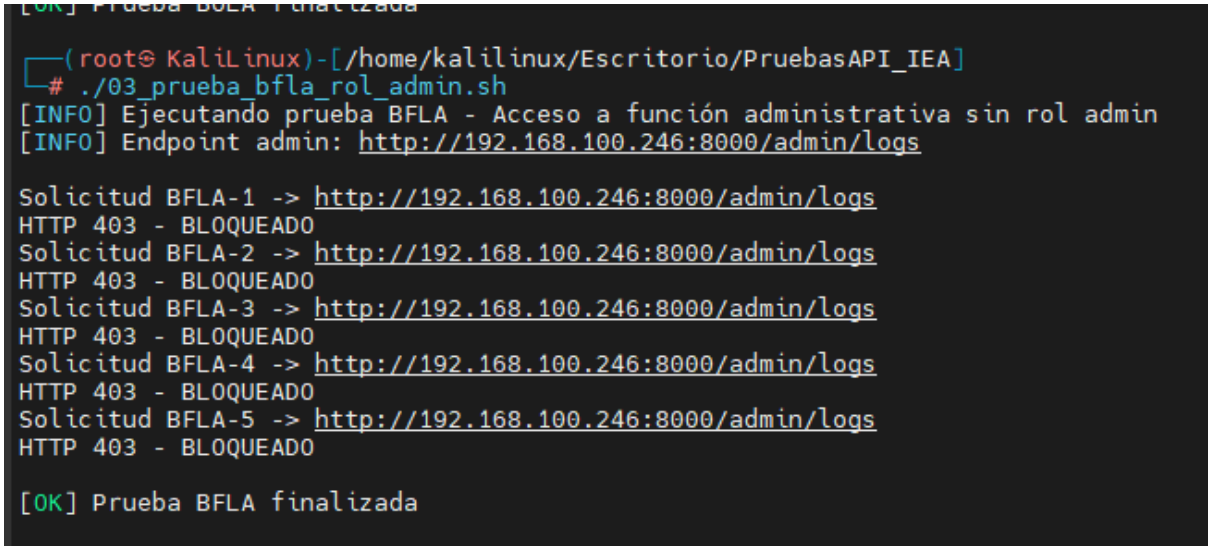
Figura 73. *Resultados obtenidos durante la reejecución de las pruebas BOLA.*

```
┌──(root㉿KaliLinux)-[/home/kalilinux/Escritorio/PruebasAPI_IEA]
└─# ./01_inicializar_resultados_iea.sh
[OK] Archivo de resultados inicializado: resultados_iea.csv

┌──(root㉿KaliLinux)-[/home/kalilinux/Escritorio/PruebasAPI_IEA]
└─# ./02_prueba_bola_propiedad_recurso.sh
[INFO] Ejecutando prueba BOLA - Acceso a recursos de otros usuarios
[INFO] Endpoint base: http://192.168.100.246:8000/profile

Solicitud BOLA-1 -> http://192.168.100.246:8000/profile/2
HTTP 403 - BLOQUEADO
Solicitud BOLA-2 -> http://192.168.100.246:8000/profile/3
HTTP 403 - BLOQUEADO
Solicitud BOLA-3 -> http://192.168.100.246:8000/profile/4
HTTP 403 - BLOQUEADO
Solicitud BOLA-4 -> http://192.168.100.246:8000/profile/5
HTTP 403 - BLOQUEADO
Solicitud BOLA-5 -> http://192.168.100.246:8000/profile/6
HTTP 403 - BLOQUEADO
Solicitud BOLA-6 -> http://192.168.100.246:8000/profile/7
HTTP 403 - BLOQUEADO
Solicitud BOLA-7 -> http://192.168.100.246:8000/profile/8
HTTP 403 - BLOQUEADO
Solicitud BOLA-8 -> http://192.168.100.246:8000/profile/9
HTTP 403 - BLOQUEADO

[OK] Prueba BOLA finalizada
```

Nota. Se realizaron ocho intentos de acceso a recursos pertenecientes a otros usuarios y la totalidad de las solicitudes fueron bloqueadas mediante HTTP 403, confirmando la efectividad del control de validación de propiedad del recurso.

Con el fin de evaluar únicamente el control de autorización basado en roles, se ejecutaron cinco solicitudes al endpoint administrativo /admin/logs utilizando un JWT válido de un usuario sin privilegios administrativos. Como se observa en la Figura 73, las cinco peticiones fueron rechazadas con respuestas HTTP 403 Forbidden, evidenciando la efectividad del mecanismo RBAC implementado para mitigar vulnerabilidades BFLA.

Figura 74. *Resultados de la reejecución de pruebas BFLA.*

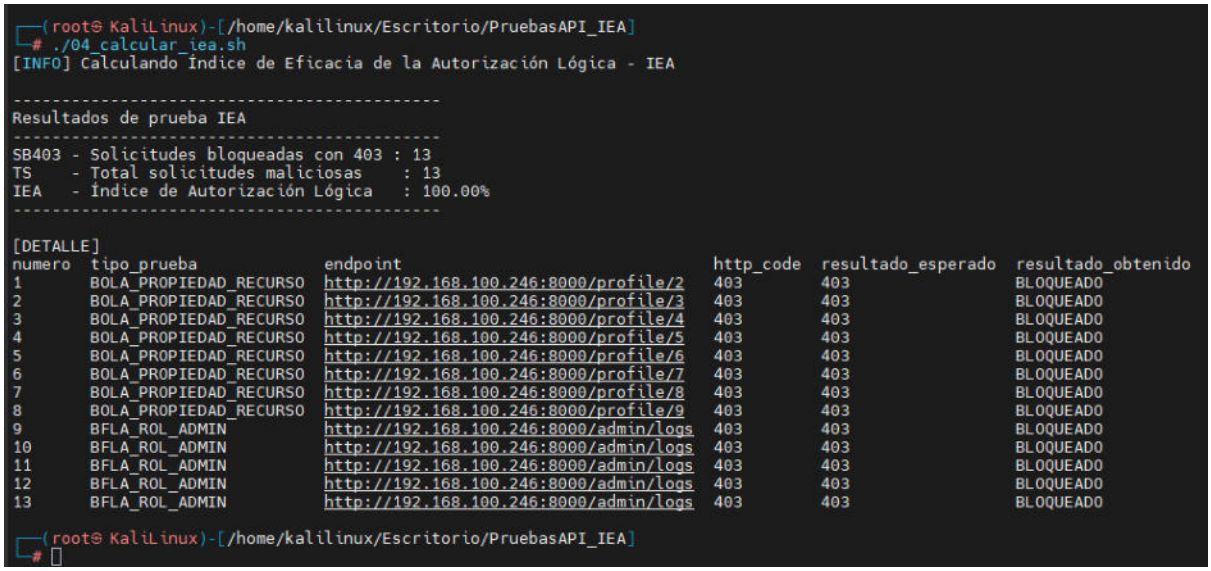


Nota. Cinco intentos de acceso a funciones administrativas utilizando un usuario sin rol admin fueron bloqueados mediante respuestas HTTP 403 Forbidden, confirmando la efectividad del control RBAC frente a ataques OWASP API5:2023 BFLA.

En la Figura 74 presenta el resultado consolidado del Índice de Eficacia de la Autorización Lógica (IEA), calculado exclusivamente a partir de los eventos asociados a decisiones de autorización implementadas en el backend.

Los resultados evidenciaron que se evaluaron trece solicitudes maliciosas: ocho correspondientes a intentos de explotación BOLA y cinco asociadas a escenarios BFLA. En todos los casos, los controles implementados respondieron con HTTP 403 Forbidden, bloqueando exitosamente los intentos de acceso no autorizado, teniendo el valor definitivo del indicador alcanzó un IEA del 100 %.

Figura 75. *Resultado final del Índice de Eficacia de la Autorización Lógica (IEA).*



Nota. El IEA se calculó sobre trece solicitudes maliciosas evaluadas, obteniéndose trece respuestas HTTP 403. El resultado final del indicador fue del 100 %, evidenciando la efectividad de los controles de autorización implementados.

Tabla 10.

Resumen de resultados obtenidos durante las pruebas del Índice de Eficacia de la Autorización Lógica (IEA).

Vulnerabilidad evaluada	Escenario de prueba	Solicitudes ejecutadas	Respuesta esperada	Respuesta obtenida	Solicitudes bloqueadas	Resultado
BOLA (API1:2023)	Acceso a recursos pertenecientes a otros usuarios mediante JWT válido	8	HTTP 403 Forbidden	HTTP 403 Forbidden	8	Mitigado
BFLA (API5:2023)	Acceso a funciones administrativas sin poseer el rol admin	5	HTTP 403 Forbidden	HTTP 403 Forbidden	5	Mitigado
Total IEA	Solicitudes maliciosas evaluadas	13	Bloqueo del acceso no autorizado	HTTP 403 Forbidden	13	IEA = 100 %

Fuente: Elaboración Propia

4.2.6. Índice de Robustez ante Manipulación de JWT (IRM-JWT)

El Índice de Robustez ante Manipulación de JWT (IRM-JWT) permite medir la capacidad del API Gateway para detectar y rechazar tokens JWT alterados antes de que estos sean procesados por los microservicios internos. Este indicador evalúa la efectividad de los controles implementados para validar la integridad y autenticidad de los tokens emitidos por el proveedor de identidad.

El indicador se calcula mediante la siguiente expresión:

$$IRM_{JWT} = \frac{SB401_{JWT}}{TS_{JWT}} * 100$$

Donde:

- IRM-JWT: Índice de Robustez ante Manipulación de JWT.
- SB401JWT: Número de solicitudes con JWT manipulados bloqueadas mediante respuestas HTTP 401 Unauthorized.
- TSJWT: Total de solicitudes enviadas utilizando tokens JWT alterados o inválidos

Se considera una prueba exitosa cuando el API Gateway responde con el código HTTP 401 Unauthorized, impidiendo que el token manipulado alcance los servicios internos.

Tabla 11.

Resumen de pruebas utilizadas para el cálculo del IRM-JWT

Tipo de prueba	Descripción	Resultado esperado
JWT con firma alterada	Sustitución parcial de la firma original	HTTP 401
JWT con firma modificada	Alteración de caracteres de la firma	HTTP 401
JWT con payload modificado	Intento de agregar el rol admin	HTTP 401

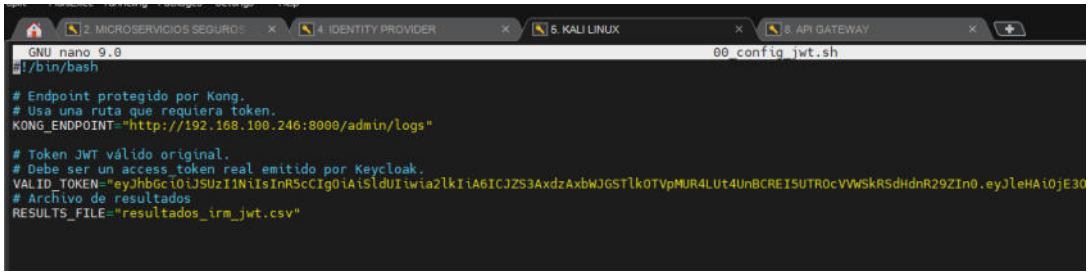
JWT con algoritmo none	Eliminación del mecanismo de firma	HTTP 401
JWT con algoritmo HS256	Cambio del algoritmo esperado	HTTP 401

Nota. Todas las pruebas buscan validar la capacidad del API Gateway para detectar modificaciones no autorizadas sobre tokens JWT emitidos por Keycloak.

4.2.6.1. Descripción de scripts empleados

El script 00_config_jwt.sh centraliza la configuración utilizada durante las pruebas de manipulación de tokens. En este archivo se define el endpoint protegido evaluado, el JWT válido emitido por Keycloak y el archivo donde serán almacenados los resultados experimentales.

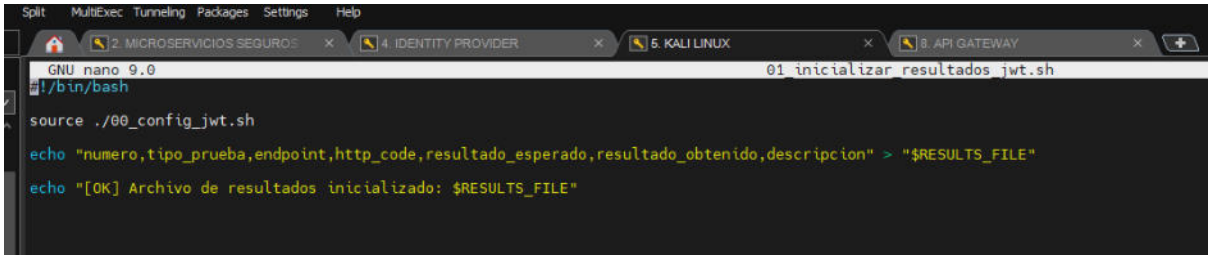
Figura 76. *Archivo de configuración para pruebas IRM-JWT.*



Nota. Script de parametrización global para la automatización de pruebas de robustez ante manipulación de JWT.

El script 01_inicializar_resultados_jwt.sh prepara el archivo resultados_irm_jwt.csv, donde se registrarán las evidencias obtenidas durante la ejecución de las pruebas.

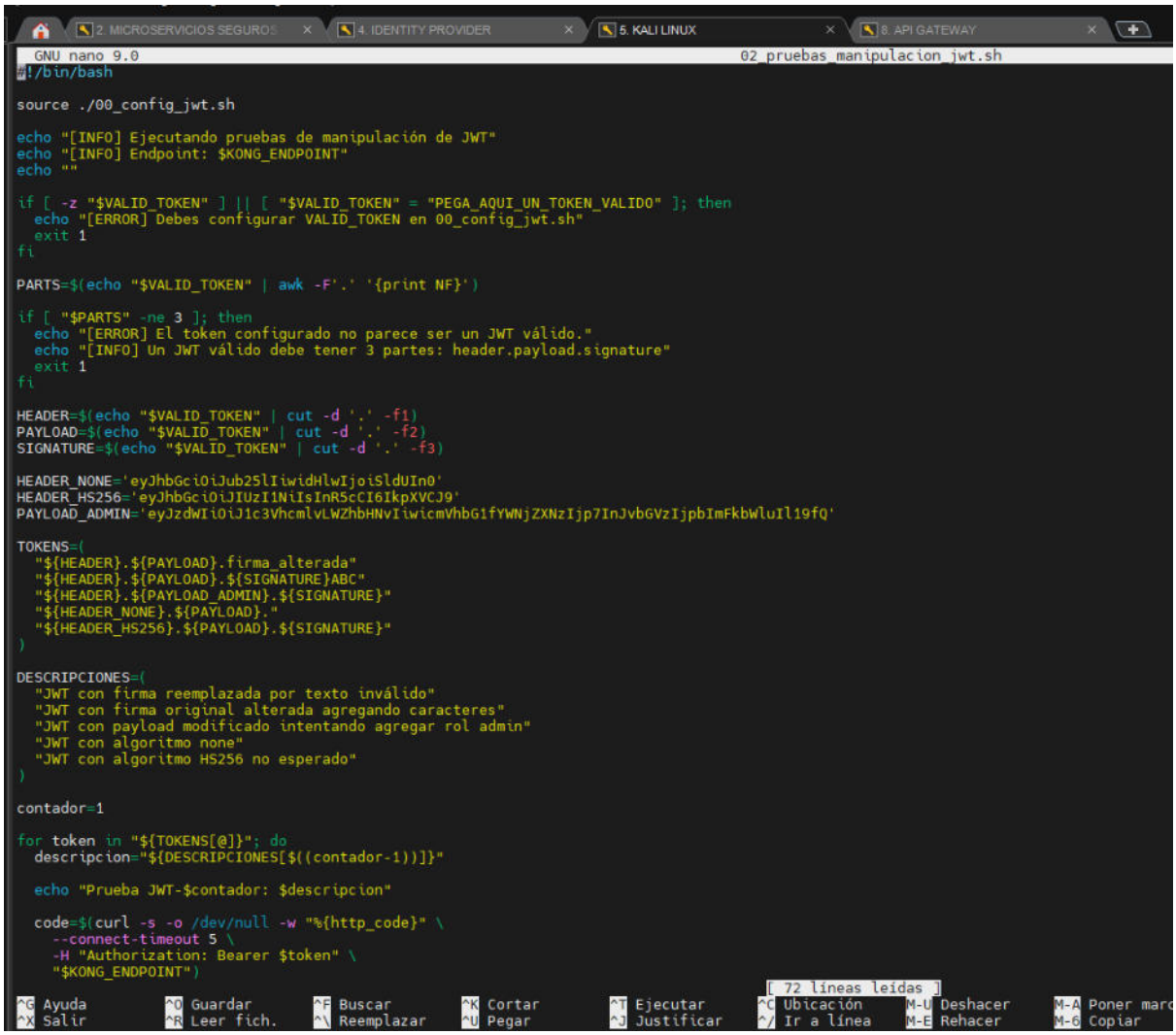
Figura 77. *Inicialización del archivo de resultados para pruebas IRM-JWT.*



Nota. Script de inicialización del repositorio de resultados para el cálculo del IRM-JWT.

El script 02_pruebas_manipulacion_jwt.sh genera diferentes variantes maliciosas del token original y ejecuta solicitudes hacia el endpoint protegido, registrando el código HTTP devuelto por el API Gateway.

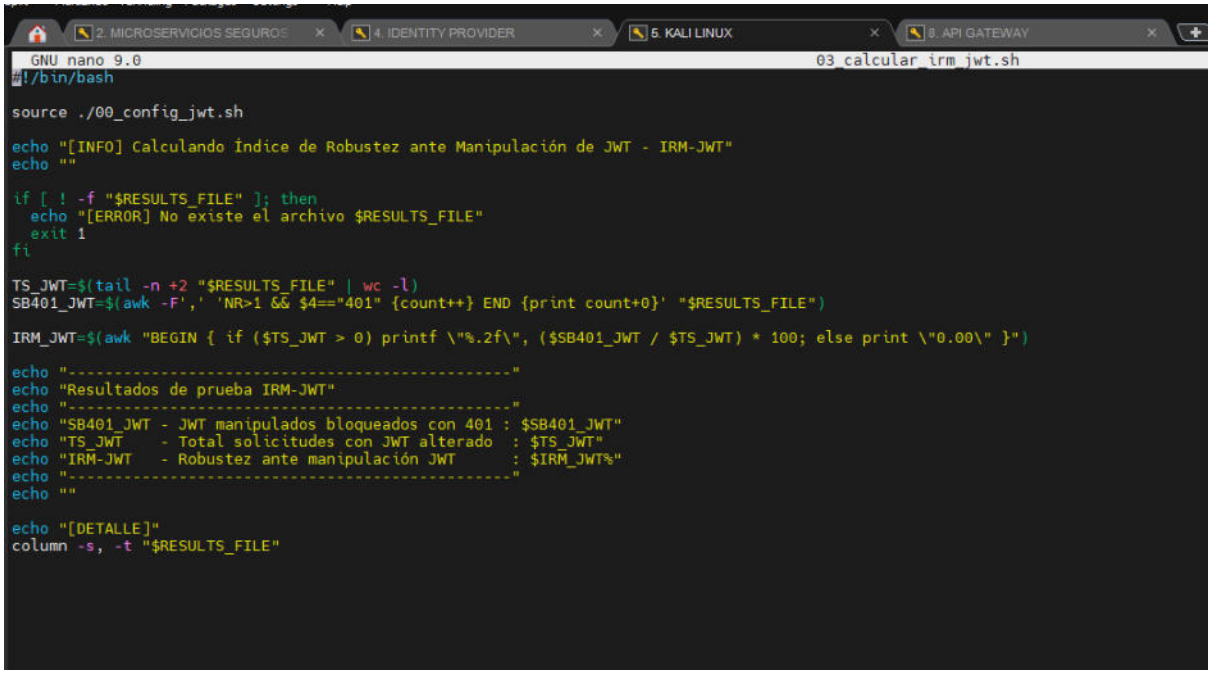
Figura 78. *Script automatizado para pruebas de manipulación de JWT.*



Nota. Script de automatización para la inyección secuencial de JWT manipulados y registro de respuestas de autenticación.

El script 03_calcular_irm_jwt.sh procesa los resultados registrados y calcula automáticamente el valor del indicador IRM-JWT.

Figura 79. *Script de cálculo del Índice de Robustez ante Manipulación de JWT (IRM-JWT).*



Nota. Script de procesamiento y cálculo del indicador de robustez frente a manipulación de JWT.

4.2.6.2. Ejecución de pruebas para el IRM-JWT

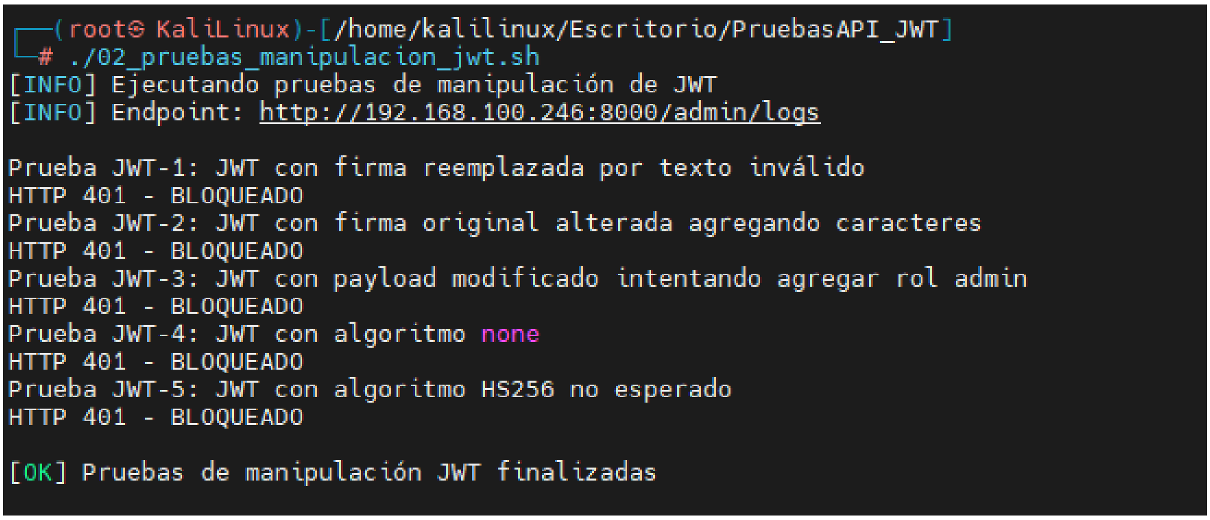
La Figura 80 muestra la preparación del entorno experimental mediante la carga del token válido emitido por Keycloak y la inicialización del archivo donde se almacenarán los resultados obtenidos durante las pruebas.

Figura 80. *Preparación del entorno para pruebas de manipulación de JWT.*

Nota. Inicialización del archivo de resultados y configuración del token base utilizado durante las pruebas.

Se ejecutaron cinco solicitudes utilizando diferentes variantes de JWT manipulados. Como se observa en la Figura 81, el API Gateway detectó todas las alteraciones realizadas sobre los tokens y respondió con códigos HTTP 401 Unauthorized, bloqueando exitosamente los intentos de acceso.

Figura 81. *Resultados de las pruebas de manipulación de JWT.*



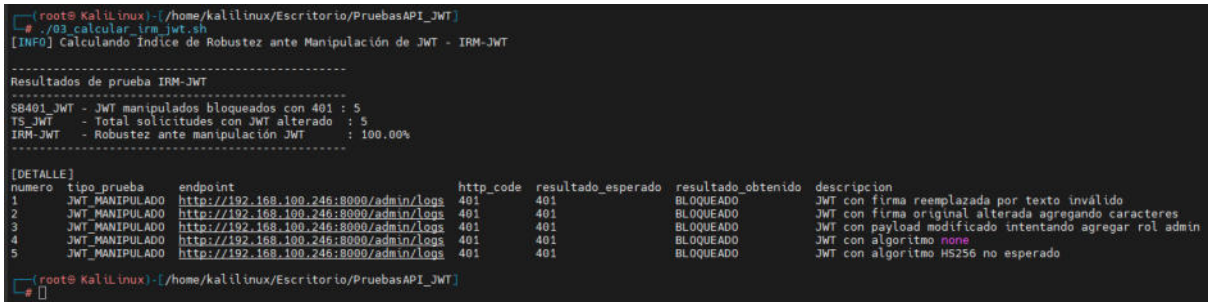
Nota. Cinco solicitudes utilizando JWT alterados fueron rechazadas mediante respuestas HTTP 401, evidenciando la validación adecuada de la firma e integridad del token.

La Figura 81 presenta el cálculo del Índice de Robustez ante Manipulación de JWT (IRM-JWT). Los resultados muestran que las cinco solicitudes realizadas utilizando JWT manipulados fueron bloqueadas correctamente por el API Gateway.

Se obtuvo:

- Solicitudes con JWT manipulados bloqueadas: 5.
- Total de solicitudes con JWT alterados: 5.
- Índice de Robustez ante Manipulación de JWT: 100 %.

Figura 82. *Resultado del Índice de Robustez ante Manipulación de JWT (IRM-JWT).*



Nota. El indicador se calculó sobre cinco solicitudes realizadas con JWT alterados, obteniéndose cinco bloqueos mediante HTTP 401 Unauthorized y un valor final del IRM-JWT del 100 %.

Tabla 12.

Resumen de resultados del IRM-JWT

Prueba	Solicitudes ejecutadas	HTTP esperado	HTTP obtenido	Resultado
Firma reemplazada por texto inválido	1	401	401	Bloqueado
Firma original alterada	1	401	401	Bloqueado
Payload modificado (rol admin)	1	401	401	Bloqueado
Algoritmo none	1	401	401	Bloqueado
Algoritmo HS256 no esperado	1	401	401	Bloqueado
Total IRM-JWT	**5**	—	**5 bloqueos**	**IRM-JWT = 100 %**

Nota. La totalidad de los JWT manipulados fueron detectados y rechazados por el API Gateway antes de alcanzar los microservicios internos.

Los resultados evidencian que los mecanismos de validación implementados en el API Gateway fueron efectivos para detectar alteraciones sobre la estructura, contenido y firma de los tokens JWT emitidos por Keycloak. La obtención de un IRM-JWT del 100 % demuestra la capacidad de la solución propuesta para mitigar intentos de suplantación o manipulación de credenciales, fortaleciendo la autenticidad e integridad del proceso de autenticación basado en OAuth 2.0 y JWT.

4.2.7. Índice de Control de Expiración de JWT (ICE-JWT)

El Índice de Control de Expiración de JWT (ICE-JWT) mide la capacidad del API Gateway para detectar y rechazar tokens cuyo atributo de expiración (exp) ha caducado, impidiendo que credenciales vencidas sean utilizadas para acceder a los recursos protegidos.

El indicador se calcula mediante la siguiente expresión:

$$ICE_{JWT} = \frac{SB401_{EXP}}{TS_{EXP}} * 100$$

Donde:

- ICE-JWT: Índice de Control de Expiración de JWT.
- SB401EXP: Número de solicitudes con tokens expirados bloqueadas mediante HTTP 401 Unauthorized.
- TSEXP: Total de solicitudes realizadas utilizando JWT vencidos.

Se considera una prueba exitosa cuando el API Gateway responde con HTTP 401 Unauthorized, rechazando el token expirado antes de permitir el acceso a las APIs internas.

Tabla 13.

Resumen de pruebas utilizadas para el cálculo del ICE-JWT

Objetivo	Herramienta	Entrada	Salida esperada	Solicitudes	Criterio de éxito
Verificar el rechazo de JWT expirados	Bash, Curl, Kong y Keycloak	JWT válido con atributo exp vencido	HTTP 401 Unauthorized	5	Todas las solicitudes deben ser bloqueadas

Nota. El indicador evalúa exclusivamente la validación del atributo de expiración (exp) de los JWT emitidos por Keycloak.

4.2.7.1. Descripción de scripts empleados

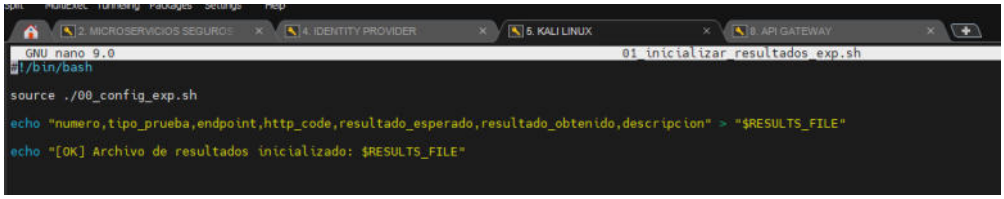
El script 00_config_exp.sh centraliza la configuración utilizada durante las pruebas de expiración del token. Define el endpoint protegido, el JWT válido emitido por Keycloak, el número de solicitudes a ejecutar y el archivo destinado al almacenamiento de resultados.

Figura 83. *Archivo de configuración para pruebas ICE-JWT.*

Nota. Script de parametrización global para automatizar pruebas de validación del atributo exp de los JWT.

El script 01_inicializar_resultados_exp.sh genera el archivo resultados_exp_jwt.csv, donde se almacenarán las evidencias obtenidas durante la ejecución experimental.

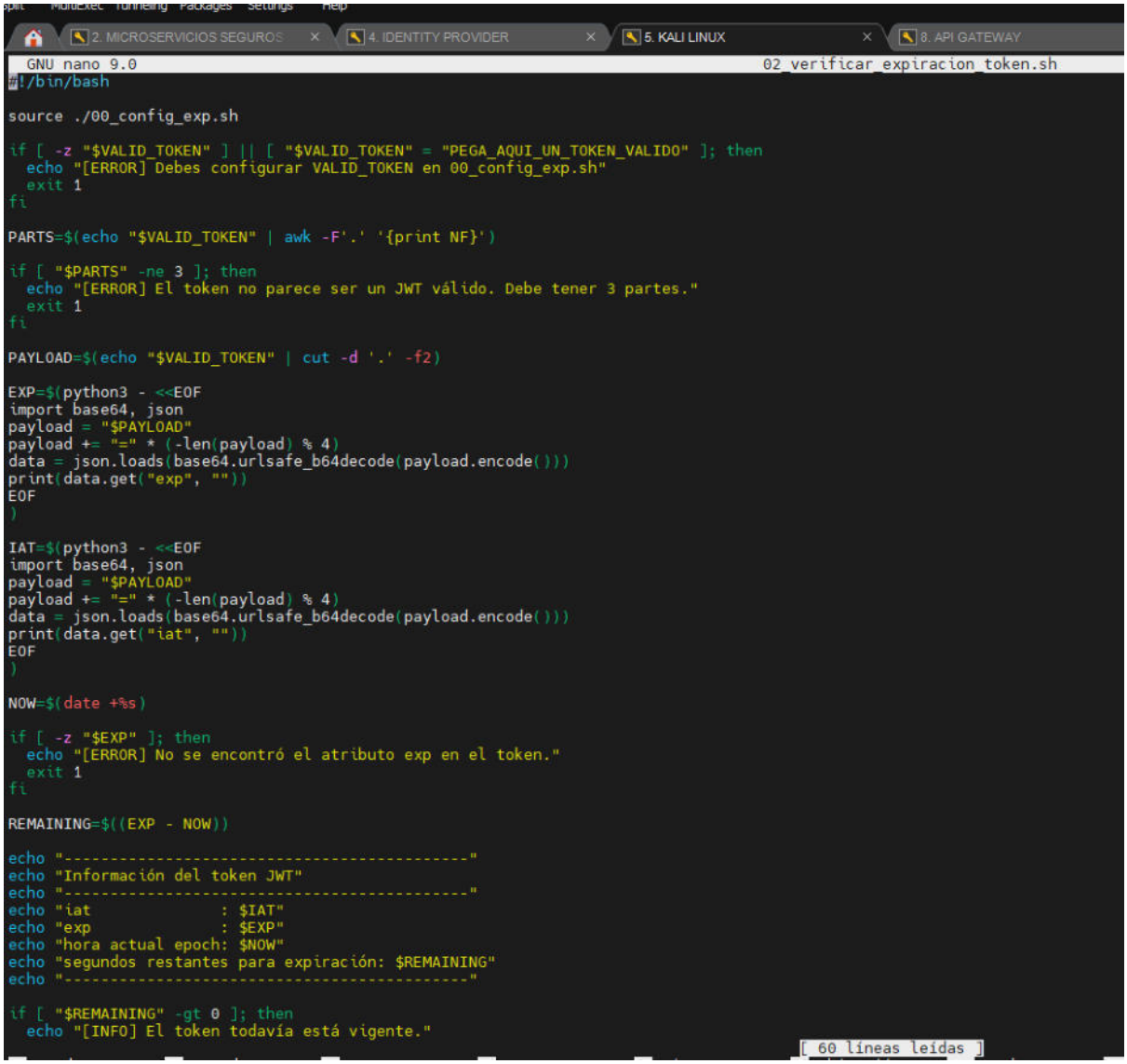
Figura 84. *Inicialización del archivo de resultados para pruebas ICE-JWT.*



Nota. Script de preparación del repositorio de resultados para el cálculo del indicador ICE-JWT.

El script 02_verificar_expiracion_token.sh analiza el contenido del JWT y extrae el valor del atributo exp, permitiendo determinar el tiempo restante antes de su expiración.

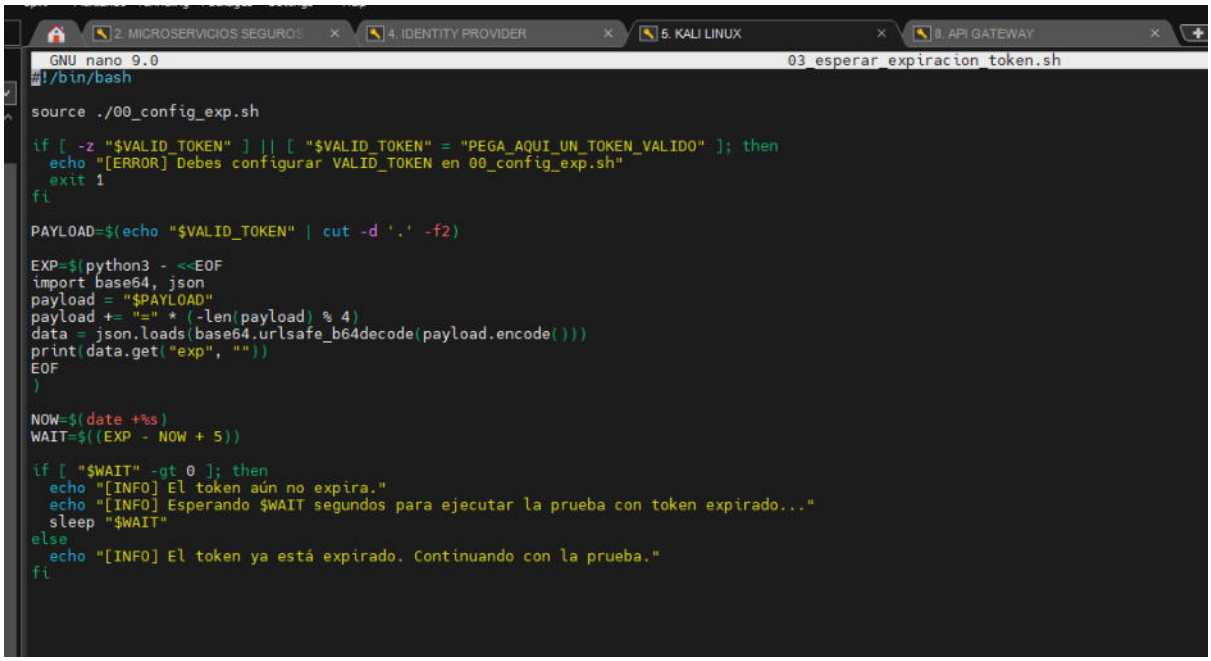
Figura 85. *Espera controlada hasta la expiración del JWT*



Nota. Script de sincronización temporal para asegurar la ejecución de pruebas utilizando JWT expirados.

El script 04_prueba_token_expirado.sh ejecuta solicitudes utilizando el token vencido y registra las respuestas HTTP generadas por el API Gateway.

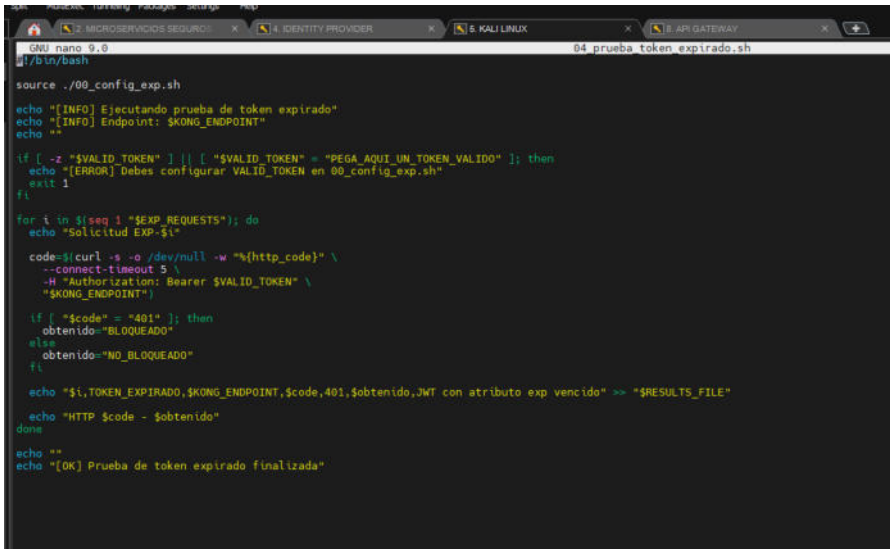
Figura 86. *Script automatizado para pruebas con JWT expirados.*



Nota. Script de automatización para validar el rechazo de credenciales expiradas.

El script 05_calcular_ice_jwt.sh procesa los resultados almacenados y calcula automáticamente el valor del indicador.

Figura 87. *Script de cálculo del Índice de Control de Expiración de JWT (ICE-JWT).*



Nota. Script de consolidación y cálculo del ICE-JWT.

4.2.7.2. Ejecución de pruebas para el ICE-JWT

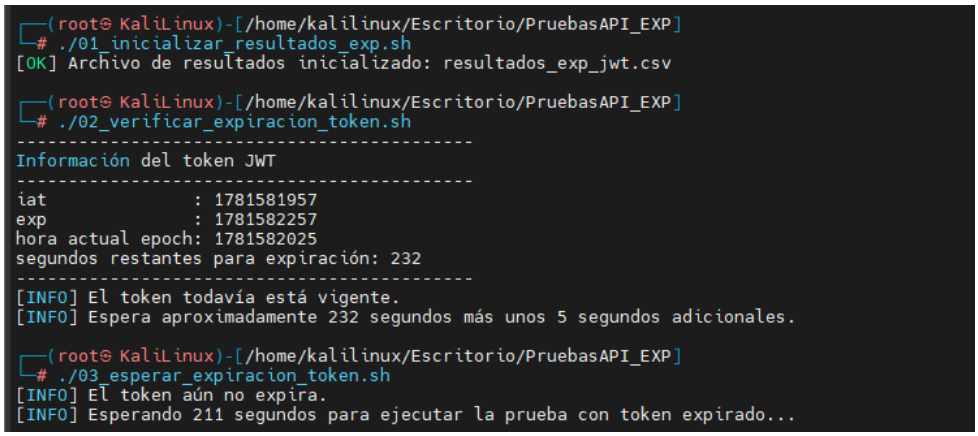
La Figura 87 muestra la obtención del token emitido por Keycloak y la inicialización del archivo donde se almacenarán los resultados experimentales.

Figura 88. *Preparación del entorno para pruebas con tokens expirados.*

Nota. Obtención del JWT válido e inicialización del archivo de resultados previo a la ejecución de las pruebas.

Se verificó el tiempo restante de vigencia del token y se esperó hasta que el atributo exp alcanzara su fecha de expiración. Como se observa en la Figura 88, el sistema determinó que el token aún se encontraba vigente y esperó automáticamente el tiempo necesario para garantizar la ejecución correcta del escenario de prueba.

Figura 89. *Verificación y espera hasta la expiración del JWT.*



Nota. Validación automática del atributo exp y sincronización temporal para asegurar el uso de un token vencido durante la prueba.

Una vez expirado el token, se realizaron cinco solicitudes consecutivas hacia el endpoint administrativo protegido. La Figura 89 evidencia que todas las peticiones fueron rechazadas mediante respuestas HTTP 401 Unauthorized, impidiendo que credenciales vencidas fueran utilizadas para acceder a recursos protegidos.

Figura 90. *Resultados de las pruebas con JWT expirados.*

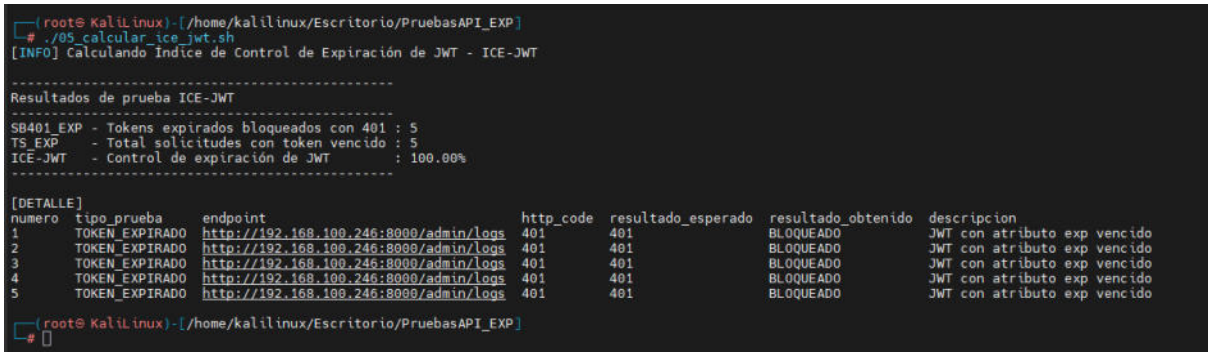
Nota. Se ejecutaron cinco solicitudes utilizando un JWT con el atributo exp vencido; todas fueron bloqueadas mediante HTTP 401 Unauthorized.

La Figura 90 presenta el cálculo del Índice de Control de Expiración de JWT (ICE-JWT). Los resultados muestran que la totalidad de los intentos realizados con credenciales vencidas fueron bloqueados exitosamente.

Se obtuvo:

- Solicitudes con JWT expirado: 5.
- Solicitudes bloqueadas con HTTP 401: 5.
- ICE-JWT: 100 %.

Figura 91. *Resultado del Índice de Control de Expiración de JWT (ICE-JWT).*



Nota. El indicador se calculó sobre cinco solicitudes realizadas con JWT expirados, obteniéndose cinco respuestas HTTP 401 Unauthorized y un valor final del ICE-JWT del 100 %.

Tabla 14.

Resumen de resultados del ICE-JWT

Tipo de prueba	Solicitudes ejecutadas	HTTP esperado	HTTP obtenido	Solicitudes bloqueadas	Resultado
JWT con atributo exp vencido	5	401	401	5	Bloqueado
Total ICE-JWT	**5**	—	**5 bloqueos**	**5**	**ICE-JWT = 100 %**

Nota. Todas las solicitudes realizadas que utilizaron tokens expirados fueron detectadas y rechazadas por el API Gateway antes de alcanzar los microservicios internos.

Los resultados evidencian que el API Gateway valida correctamente el atributo de expiración (exp) presente en los JWT emitidos por Keycloak. La obtención de un ICE-JWT del 100 % demuestra que la arquitectura propuesta impide la reutilización de credenciales vencidas, mitigando riesgos asociados al uso indebido de tokens expirados y fortaleciendo los controles de autenticación implementados mediante OAuth 2.0 y JWT.

4.3. Análisis de Resultados

4.3.1. Índice de Eficacia Perimetral (IEP)

Los resultados obtenidos mediante la ejecución de las pruebas automatizadas evidencian que los mecanismos de protección perimetral implementados en el API Gateway respondieron adecuadamente ante las solicitudes maliciosas evaluadas durante el escenario experimental.

El indicador se calculó considerando un total de 20 solicitudes maliciosas, distribuidas en dos escenarios de prueba. En el primer escenario, se ejecutaron 5 solicitudes utilizando tokens inválidos o manipulados, obteniéndose como resultado el bloqueo de la totalidad de las peticiones mediante respuestas HTTP 401 Unauthorized. En el segundo escenario, se realizaron 15 solicitudes que excedieron el límite de peticiones configurado en el API Gateway, las cuales fueron bloqueadas mediante respuestas HTTP 429 Too Many Requests.

El cálculo del Índice de Eficacia Perimetral se realizó mediante la siguiente expresión:

$$IEP = \left(\frac{SB401 + SB429}{TS}\right) * 100\%$$

$$IEP = \left(\frac{5 + 15}{20}\right) * 100\%$$

$$IEP = 100\%$$

El resultado obtenido demuestra que la solución implementada fue capaz de bloquear el 100 % de las solicitudes consideradas maliciosas durante las pruebas realizadas. Esto evidencia que el API Gateway actuó correctamente como punto de control perimetral, validando los mecanismos de autenticación y aplicando controles de limitación de tráfico antes de permitir el acceso a los microservicios internos.

Desde una perspectiva de seguridad, los resultados confirman la efectividad de los controles implementados para mitigar intentos de acceso no autorizado y ataques asociados al consumo excesivo de recursos. Asimismo, se valida que la centralización de los mecanismos de protección en el API Gateway reduce la superficie de exposición de las APIs y fortalece la arquitectura de seguridad propuesta para el entorno de microservicios.

En consecuencia, durante la ejecución de las pruebas no se identificaron solicitudes maliciosas que lograran superar los controles perimetrales definidos, obteniéndose un Índice de Eficacia Perimetral (IEP) del 100 %, lo que evidencia un comportamiento satisfactorio de la solución implementada dentro de las condiciones evaluadas.

CAPÍTULO 5

5. Conclusiones y recomendaciones

5.1. Conclusiones

Se diseñó e implementó exitosamente una arquitectura de autenticación y autorización basada en OAuth 2.0, JSON Web Tokens (JWT), Keycloak y Kong API Gateway, permitiendo centralizar los mecanismos de control de acceso y proteger los microservicios desplegados en contenedores Docker dentro de un entorno controlado y reproducible.

Se determinó que las configuraciones predeterminadas de los servicios de gestión de identidades (IDP), los servicios de frontera y las APIs desarrolladas sin mecanismos de seguridad complementarios, como *middlewares*, pueden proporcionar la funcionalidad requerida, pero no garantizan un nivel adecuado de protección. Estas implementaciones suelen mantener vulnerabilidades y brechas de seguridad que incrementan el riesgo de exposición de información sensible y de acceso no autorizado a los servicios.

Las pruebas realizadas evidenciaron que una configuración sin controles de seguridad expone las APIs a vulnerabilidades críticas, permitiendo accesos no autorizados a recursos y funcionalidades sensibles. En contraste, la implementación de validaciones de autenticación, autorización basada en roles y controles de propiedad de recursos permitió mitigar eficazmente escenarios asociados a OWASP API Security Top 10, particularmente BOLA (Broken Object Level Authorization) y BFLA (Broken Function Level Authorization).

Se concluyó que la incorporación de controles de seguridad desde el diseño y desarrollo de los servicios constituye un factor determinante para fortalecer la protección de los sistemas. Asimismo, los resultados obtenidos demuestran la necesidad de implementar mecanismos de seguridad adicionales de manera integral en todo desarrollo de software, con el fin de reducir vulnerabilidades y garantizar la confidencialidad, integridad y disponibilidad de la información.

5.2. Recomendaciones

Con base en los resultados obtenidos durante la investigación, se recomienda incorporar mecanismos de seguridad adicionales en el desarrollo de APIs, de manera que estas cuenten con capacidades de autoprotección frente a posibles amenazas y vulnerabilidades. Este enfoque permite que, incluso en escenarios donde las capas de seguridad superiores sean comprometidas, las APIs puedan implementar controles internos para detectar, mitigar y responder a incidentes de seguridad.

En un entorno real productivo se recomienda complementar el modelo implementado con una autenticación mutua mediante MTLS, que permite verificar tanto la identidad del cliente como la del servidor durante el establecimiento de la conexión. Así mismo, se debe conservar al API Gateway utilizado como único punto de entrada y restringir el acceso directo a las APIS, logrando que componentes no autorizados se conecten directamente a la arquitectura propuesta.

En un escenario de pruebas se recomienda ajustar el umbral del rate limiting configurado, con el objetivo de evitar respuestas HTTP 429 que interfieren en la evaluación de los controles de autorización. A su vez, el rate limiting debe validarse de manera independiente para medir correctamente su eficacia.

Referencias Bibliográficas

Alqahtani, S., El-Meligy, M., & Alghamdi, A. (2022). Security challenges and solutions in containerized environments. *Sensors Journal.*, 22(3), 1–18.

Bandana, K. (2026). Broken Object Level Authorization in the Wild: An Empirical Taxonomy from 100+ Bug Bounty Disclosures. *arXiv*, 25.

Berners-Lee, Fielding, & Masinter. (s.f.). *datatracker*. datatracker: https://datatracker.ietf.org/doc/html/rfc3986

Borchert, O., Rose, S., Mitchell, S., & Connelly, S. (2020). Zero Trust Architecture. *National Institute of Standards and Technology Walter Copan, NIST Director and Under Secretary of Commerce for Standards and Technology*, 59.

Corporation, O. (2024). *Oracle Corporation*. virtualbox: https://www.virtualbox.org/

Corradini, D., Ceccato, M., & Ghafari, M. (2025). Automated Testing of Broken Authentication Vulnerabilities in Web APIs with AuthREST. *ResearchGate*, 4.

Cuthbert, D., Grossman, J., & Lang, E. (2025). *Github*. Github: https://github.com/OWASP/ASVS

Descope. (19 de Agosto de 2025). *descope.com*. Descope: https://www.descope.com/learn/post/jwt-claims

Dharmaadi, I., Alhanahnah, M., Pham, V.-T., Mohsen, F., & Turkmen, F. (2025). BACFuzz: Exposing the Silence on Broken Access Control Vulnerabilities in Web Applications. *arXiv*, 12.

Docker. (2023). *Docker Documentation*. https://docs.docker.com/

Fatunmbi, T. (8 de Febrero de 2022). *A Comparison of Cookies and Tokens for Secure Authentication*. okta Developer: https://developer.okta.com/blog/2022/02/08/cookies-vs-tokens

Fielding, Nottingham, & Reschke. (Junio de 2022). *Datatracker*. Datatracker: https://datatracker.ietf.org/doc/html/rfc9110

Filho, A., Rodríguez, R., & Feitosa , E. (2025). Automated broken object-level authorization attack detection in REST APIs through OpenAPI to colored petri nets transformation. *International Journal of Information Security*, 19.

Genai. (01 de Enero de 2026). genai.owasp.org: https://genai.owasp.org/2026/04/14/owasp-genai-exploit-round-up-report-q1-2026/

Godula, M. (13 de Febrero de 2026). *inflo.tech*. nflo: https://nflo.tech/knowledge-base/api-penetration-testing-complete-guide/

Griffeth, N. (10 de Noviembre de 2025). *fastly*. fastly.com: https://www.fastly.com/blog/new-2025-owasp-top-10-list-what-changed-what-you-need-to-know

Hardt, D. (2012). *The OAuth 2.0 Authorization Framework (RFC 6749)*. Internet Engineering Task Force (IETF): https://doi.org/10.17487/RFC6749

IBM. (12 de Julio de 2024). *What is a REST API*. IBM Think: https://www.ibm.com/think/topics/rest-apis

Jones, Bradley, & Sakimura. (2015). JSON Web Token (JWT). *Datatracker*, 29.

Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT) (RFC 7519)*. Internet Engineering Task Force (IETF): https://doi.org/10.17487/RFC7519

Joshi, S. (15 de Abril de 2024). *Introduction to APIs: Types of APIs, API architecture and beyond*. Sendbird Blog: https://sendbird.com/developer/tutorials/introduction-to-apis-types-of-apis-api-architecture-and-beyond

Lauter, K., & Stange, K. (2008). The elliptic curve discrete logarithm problem and equivalent hard problems for elliptic divisibility sequences. *Cornell University*, 18.

Lindemulder, G., & Kosinski, M. (s.f.). *IBM*. IBM: https://www.ibm.com/think/topics/rbac

Mahfud, I., Utomo, P., & Winarno, B. (2025). Implementation of Rivest-Shamir-Adleman (RSA) Cryptography System with Secure Hash Algorithm (SHA-256) on REST API Authentication Mechanism. *AIP Conference Proceedings (Scopus)*, 12.

Mahindrakar, P., & Pujeri, U. (2020). Insights of JSON Web Token. *International Journal of Recent Technology and Engineering (IJRTE)*, 4.

Merino. (24 de Julio de 2024). *Cookies y tokens: qué son exactamente estas piezas de información que nos permiten interactuar con las plataformas web*. GENBETA: https://www.genbeta.com/a-fondo/cookies-tokens-que-exactamente-estas-piezas-informacion-que-nos-permiten-interactuar-plataformas-web

Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal*, 239.

Mitchell, R., & Saad, E. (2024). *Github*. Github: https://github.com/OWASP/wstg/blob/master/document/4-Web_Application_Security_Testing/04-Authentication_Testing/README.md

Nguyen, T. (15 de Ovtubre de 2024). *ECDSA, EdDSA and Schnorr — the anatomy of elliptic curve-based signature schemes.* Medium: https://medium.com/@barchitect/ecdsa-eddsa-and-schnorr-the-anatomy-of-elliptic-curve-based-signature-schemes-583bb96df076

NIST, N. I. (2011). *National Institute of Standards and Technology*. Guide to Security for Full Virtualization: https://doi.org/10.6028/NIST.SP.800-125

Nugraha, A. F., Kabetta, H., Buana, I. S., & R Budiarto , H. (2023). Performance and Security Comparison of Json Web Tokens (JWT) and Platform Agnostic Security Tokens (PASETO) on RESTful APIs. *IEEE Xplore*, 8.

Oracle. (2023). *Oracle VM VirtualBox User Manual*. Oracle VM VirtualBox User Manual: https://www.virtualbox.org/manual/

OWASP. (2023). *OWASP API Security Top 10*. OWASP API Security Top 10: https://owasp.org/API-Security/

Pahl, C. (2015). Containerization and the PaaS cloud. *IEEE Cloud Computing*, 24-31.

Red Hat. (10 de Mayo de 2024). *¿Qué es GraphQL?* Red Hat: https://www.redhat.com/es/topics/api/what-is-graphql

Sakimura, Bradley, & Agarwal. (Septiembre de 2015). *Datatracker*. Proof Key for Code Exchange by OAuth Public Clients: https://datatracker.ietf.org/doc/html/rfc7636

Salt Security. (24 de Febrero de 2024). *State of API Security Report*. Salt Security: https://23167483.fs1.hubspotusercontent-na1.net/hubfs/23167483/State%20of%20API%20Security%20-%20V4.pdf

Serengil, S., & Ozpinar, A. (2025). LightDSA: A Python-Based Hybrid Digital Signature Library and Performance Analysis of RSA, DSA, ECDSA and EdDSA in Variable Configurations, Elliptic Curve Forms and Curves. *arXiv*, 16.

Sharma, P., Chen, Y., & Sheth, A. (2021). Toward practical container security: A survey. . *IEEE Access*, 9, 1-20. IEEE Access.

Singh, S., & Chana, I. (2021). Cloud resource provisioning: Survey, status and future research directions. *Knowledge-Based Systems*, 79, 435–448. Cloud resource provisioning: Survey, status and future research directions. Knowledge-Based Systems.

Tanveer, F., Iradat, F., Iqbal, W., & Ahmad, A. (2025). Towards Secure APIs: A Survey on RESTful API Vulnerability Detection. *Tech Science Press*, 35.

Torres, R. (2024). Criptografía simétrica y asimétrica y su aplicación en medios digitales como las imágenes, video y audio. *UNAD Universidad Nacional Abierta y a Distancia*, 82.

Valdes, D., & Tejedor, M. (n.d). HASHING. UN CONCEPTO. UNA REALIDAD. *Universidad Tecnológica de Panamá, C.R. de Coclé, Grupo de investigación SoftSolution Group*, 7.

Viswanathan, J., Kumar. N, D., & Kumar, S. (2024). Zero Trust Security for Web Applications in Microservice-Based Environments. *IEEE Xplore*, 494.

Vmware. (2024). *VMware by Broadcom*. VMware: https://www.vmware.com/

Zhang, Q. C. (2020). Security analysis of virtualization technologies in cloud computing. *Journal of Cloud Computing*, 9(1), 1–15. Journal of Cloud Computing.

Apéndices

Link en el que se encuentra compartidos los laboratorios prácticos desarrollados

https://github.com/Mick971/Trabajo-Titulacion-Ciberseguridad/blob/main/README.md