Maestría en

# CIBERSEGURIDAD

**Trabajo previo a la obtención de título de**
**Magister en Ciberseguridad**


**AUTORES:**

AGUILAR ROMERO MARIA EMILIA

BOLAÑOS MERA BYRON GEOVANNY

PALACIOS JARAMILLO CHRISTIAN DANIEL

SAMANIEGO RUIZ JOSUE ISMAEL

**TUTORES:**

Paulina Vizcaíno

Iván Reyes Chacón


**TEMA**

Diseño e implementación de un honeypot basado en API REST para la detección y clasificación de patrones de ataque mediante técnicas de Machine Learning integradas con Elastic Stack


Quito - Ecuador

Junio – 20206


REINVENTEMOS EL FUTURO

**Certificación de autoría**

Nosotros, María Emilia Aguilar Romero, Josue Ismael Samaniego Ruiz, Christian Daniel Palacios Jaramillo, Byron Geovanny Bolaños Mera, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido presentado anteriormente para ningún grado o calificación profesional y que se ha consultado la bibliografía detallada.

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

---

**Firma**

María Emilia Aguilar Romero

---

**Firma**

Josue Ismael Samaniego Ruiz

---

**Firma**

Christian Daniel Palacios Jaramillo

---

**Firma**

Byron Geovanny Bolaños Mera

**Autorización de Derechos de Propiedad Intelectual**

Nosotros, María Emilia Aguilar Romero, Josué Ismael Samaniego Ruiz, Christian Daniel Palacios Jaramillo, Byron Geovanny Bolaños Mera, en calidad de autores del trabajo de investigación titulado Diseño e implementación de un honeypot basado en API REST para la detección y clasificación de patrones de ataque mediante técnicas de machine learning integradas con elastic stack, autorizamos a la Universidad Internacional del Ecuador (UIDE) para hacer uso de todos los contenidos que nos pertenecen o de parte de los que contiene esta obra, con fines estrictamente académicos o de investigación. Los derechos que como autores nos corresponden, lo establecido en los artículos 5, 6, 8, 19 y demás pertinentes de la Ley de Propiedad Intelectual y su Reglamento en Ecuador.

D. M. Quito, (junio 2026)

---

**Firma**

María Emilia Aguilar Romero

---

**Firma**

Josue Ismael Samaniego Ruiz

---

**Firma**

Christian Daniel Palacios Jaramillo

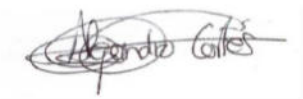
---

**Firma**

Byron Geovanny Bolaños Mera

**APROBACIÓN DE DIRECCIÓN Y COORDINACIÓN DEL PROGRAMA**

Nosotros, Alejandro Cortés López e Iván Galo Reyes Chacón, declaramos que: María Emilia Aguilar Romero, Josué Ismael Samaniego Ruiz, Christian Daniel Palacios Jaramillo, Byron Geovanny Bolaños Mera, son los autores exclusivos de la presente investigación y que esta 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 con profundo amor a mis padres y a mis abuelos, por su apoyo constante y por ayudarme a cumplir un logro más en mi vida. Esto va para ustedes, gracias por estar siempre.

***— Christian Daniel Palacios Jaramillo***

Para mis padres, quienes han sido mi mayor ejemplo de esfuerzo, perseverancia y amor incondicional. Cada logro alcanzado lleva consigo una parte de todo lo que me han enseñado y de los valores que me han inculcado. A mis hermanos, por ser una fuente constante de inspiración y un ejemplo a seguir en mi vida. Gracias por su cariño, apoyo y por acompañarme en cada etapa de este camino.

***— María Emilia Aguilar Romero***

Para mi hija Saori que se convirtió en mi motor en esta etapa de mi vida y mi esposa Valentina por su apoyo constante en este camino de estudio dando el amor necesario para seguir adelante.

***— Byron Geovanny Bolaños Mera***

A mis padres, cuyo esfuerzo y sacrificio hicieron posible cada paso de este camino. Para mi esposa y mi hijo, por su paciencia, apoyo incondicional y por estar presentes en los momentos más exigentes de este proceso.

***— Josue Isamel Samaniego Ruiz***

**Agradecimientos**

De manera muy especial, agradezco profundamente a María Emilia Aguilar Romero e Josué Ismael Samaniego Ruiz, por su apoyo incondicional, su compañerismo y por no dejarme atrás en ningún momento de este exigente camino. Su respaldo y amistad hicieron posible alcanzar esta meta juntos.

***— Christian Daniel Palacios Jaramillo***

Agradezco a mis padres y hermanos, por su apoyo, comprensión y confianza durante esta etapa de mi formación. Gracias por acompañarme en este proceso y brindarme el impulso necesario para alcanzar esta meta. Asimismo, agradezco a mis amigos Christian Daniel Palacios Jaramillo y Josué Ismael Samaniego Ruiz por su amistad, por su apoyo y por compartir conmigo este camino.

***— María Emilia Aguilar Romero***

Agradezco a mi madre, mi hermano y mi padrastro, que a pesar de ya estar lejos siempre han estado ayudándome y dándome los consejos en los momentos precisos. Inculcaron en mí esas ganas de salir adelante y no rendirme frente a la adversidad en este largo camino de la vida.

***— Byron Geovanny Bolaños Mera***

Agradezco a mi familia por siempre estar ahí para mi incluso cuando la situación es tan complicada que lo hace difícil. Agradezco a mis compañeros y amigos por su apoyo en este proceso y siempre seguir adelante.

***— Josue Isamel Samaniego Ruiz***

**Resumen**

El constante crecimiento de arquitecturas basadas en microservicios ha incrementado la necesidad de APIs REST como forma de comunicación entre sistemas distribuidos, ampliando la superficie de ataque expuesta en internet. Los sistemas de detección que utilizan reglas y firmas estáticas tienen limitaciones contra amenazas dinámicas y comportamiento desconocido, esto genera una necesidad de mecanismos dinámicos para el monitoreo, análisis y detección de amenazas. El presente trabajo propone el diseño e implementación de un honeypot de baja interacción que simula servicios API REST, integrado con Elastic Stack y modelos de machine learning. El sistema se desarrolló en Python utilizando Flask reproduciendo una plataforma web con endpoints vulnerables y diferentes escenarios de explotación.

Cada evento se guarda en formato JSON y se utilizan etiquetas semánticas para la clasificación de estos. Los registros se transportan a Elasticsearch con Filebeat para visualizarlos en el dashboard de Kibana. Para el análisis automático se implementaron dos modelos de machine learning, Random Forest y K-Means. Los resultados validan el funcionamiento del pipeline completo, el modelo Random Forest obtuvo una exactitud del 99.4% y una sensibilidad del 99.4% en detección de ataques, con una tasa de falsos positivos del 0.45%. El algoritmo K-Means logró aislar de forma autónoma cerca del 90% de los patrones anómalos, entre ellos posibles eventos de “día cero”. Esto ha demostrado que la integración de tecnologías de engaño, monitoreo centralizado y aprendizaje automático constituye un enfoque eficaz para el análisis de amenazas dirigidas a APIs REST en entornos controlados.

**Palabras Claves:** Honeypot, API REST, Elastic Stack, Machine Learning, Random Forest, K-Means.

**Abstract**

The steady growth of microservices-based architectures has increased the need for REST APIs as a means of communication between distributed systems, thereby expanding the attack surface exposed on the internet. Detection systems that rely on static rules and signatures have limitations when it comes to dynamic threats and unknown behavior, creating a need for dynamic mechanisms for monitoring, analyzing, and detecting threats. This paper proposes the design and implementation of a low-interaction honeypot that simulates REST API services, integrated with the Elastic Stack and machine learning models. The system was developed in Python using Flask to replicate a web platform with vulnerable endpoints and various exploitation scenarios.

Each event is saved in JSON format and classified using semantic tags. The logs are transported to Elasticsearch via Filebeat for visualization on the Kibana dashboard. For automatic analysis, two machine learning models were implemented: Random Forest and K-Means. The results validate the functionality of the entire pipeline; the Random Forest model achieved 99.4% accuracy and 99.4% sensitivity in attack detection, with a false positive rate of 0.45%. The K-Means algorithm autonomously isolated nearly 90% of anomalous patterns, including possible "zero-day" events. This has demonstrated that the integration of deception technologies, centralized monitoring, and machine learning constitutes an effective approach for analyzing threats targeting REST APIs in controlled environments.

**Keywords:** Honeypot, API REST, Elastic Stack, Machine Learning, Random Forest, K-Means.

**Índice de Contenidos**

## Índice de tablas

## Índice de figuras

# Capítulo 1:

# 1. Introducción

## 1.1. Definición del proyecto

El crecimiento de las aplicaciones web y de las arquitecturas basadas en microservicios ha incrementado el uso de APIs REST como mecanismo principal de comunicación entre sistemas distribuidos. Estas interfaces permiten el intercambio de información entre aplicaciones web, servicios en la nube y plataformas móviles mediante protocolos HTTP, convirtiéndose en componentes esenciales dentro de infraestructuras tecnológicas modernas (Fielding, 2000).

Debido a su constante exposición a internet, las APIs REST se han convertido en objetivos frecuentes de ataques relacionados con accesos no autorizados, fuerza bruta, automatización maliciosa y explotación de vulnerabilidades. Esta situación ha generado la necesidad de implementar mecanismos capaces de monitorear eventos de seguridad y detectar comportamientos sospechosos dentro del tráfico HTTP (Sharma et al., 2021).

Dentro de este marco, el presente proyecto plantea el diseño e implementación de un honeypot de baja interacción orientado a simular servicios API REST y a registrar patrones de ataque en un entorno controlado. La baja interacción se eligió porque el objetivo no es entregar al atacante un sistema real, sino recoger evidencia útil de sus solicitudes HTTP sin comprometer servidores, bases de datos o recursos internos. Zambrano et al. (2021) señalan que los honeypots permiten observar intentos de ataque y mejorar la disponibilidad de los servicios cuando se aplican como mecanismos de monitoreo. En este trabajo, esa idea se adapta a una API REST, donde cada intento de autenticación, escaneo o manipulación queda registrado como evento para su análisis posterior.

La arquitectura propuesta integrará un honeypot desarrollado sobre servicios API REST, herramientas de monitoreo y visualización mediante Elastic Stack y modelos de Machine Learning orientados al análisis automatizado de eventos de seguridad. El propósito del proyecto es proporcionar un entorno controlado capaz de registrar actividades maliciosas y facilitar el análisis de amenazas dirigidas a servicios web modernos (Fraunholz et al., 2020).

### 1.2. Justificación e importancia del trabajo de investigación

El incremento de ataques dirigidos a servicios web y APIs REST ha generado nuevos desafíos dentro de la ciberseguridad. Hoy muchas organizaciones dependen de APIs para conectar aplicaciones, servicios en la nube, sistemas internos y plataformas móviles. Por esta razón, cuando una API queda mal protegida, también queda expuesta a la información que circula por ella. La OWASP Foundation (2023) advierte que las fallas en autorización, autenticación y lógica de negocio siguen siendo riesgos centrales en este tipo de servicios, por lo que resulta necesario contar con mecanismos que permitan observar el comportamiento del atacante antes de que afecte un sistema real.

Aunque existen mecanismos tradicionales de protección como firewalls, IDS e IPS, estos sistemas presentan limitaciones frente a amenazas dinámicas y comportamientos desconocidos, ya que dependen principalmente de reglas y firmas previamente definidas para identificar actividades maliciosas. Esta situación dificulta la detección temprana de ataques automatizados y técnicas modernas de intrusión dirigidas a servicios web (Garcia-Teodoro et al., 2009).

Los honeypots representan una alternativa orientada al monitoreo y análisis de comportamiento malicioso mediante la simulación de entornos vulnerables capaces de atraer atacantes y registrar sus acciones dentro de un ambiente controlado. La información recopilada

por estos sistemas puede utilizarse para identificar patrones de ataque y fortalecer mecanismos defensivos orientados a servicios expuestos a internet (Spitzner, 2003).

Además, el uso de aprendizaje automático permite revisar los eventos de seguridad de una forma más ordenada, puesto que ayuda a reconocer diferencias entre tráfico normal y tráfico sospechoso. Como explica Carrizo (2024), la integración de Machine Learning con sistemas de detección de intrusos puede mejorar la precisión y reducir falsos positivos en entornos controlados. Con este fundamento, este proyecto integra Random Forest para clasificar eventos etiquetados y K-Means para agrupar comportamientos parecidos, de modo que el análisis no depende solo de una revisión manual de logs.

Por esta razón, el presente proyecto busca desarrollar un prototipo funcional validado en laboratorio. La solución integra un honeypot API REST, Elastic Stack y modelos de aprendizaje automático para capturar, visualizar y clasificar eventos de seguridad. El aporte principal no está en bloquear ataques en una red productiva, sino en demostrar que un entorno puede generar datos útiles para reconocer patrones de ataque y apoyar decisiones defensivas en servicios web modernos.

### 1.3. Alcance

El presente trabajo comprende el diseño e implementación de un prototipo funcional de honeypot de baja interacción orientado a simular servicios API REST dentro de un entorno controlado de laboratorio. El sistema permitirá registrar solicitudes HTTP, intentos de acceso, consultas a endpoints vulnerables y comportamientos asociados con ataques contra servicios web.

La arquitectura propuesta integrará Elastic Stack para la recolección, almacenamiento y visualización de logs generados por el honeypot. Adicionalmente, se incorporarán técnicas de

Machine Learning orientadas a la detección de anomalías y clasificación de patrones de ataque a partir de los eventos recopilados.

Como parte del proceso, se implementan escenarios controlados que simulan ataques comunes contra APIs REST, entre ellos fuerza bruta, enumeración de rutas, inyección SQL, XSS, acceso indebido a objetos y manipulación de un flujo de pagos. Estos escenarios permiten comprobar si el honeypot registra los eventos, si Elastic Stack los indexa y si los modelos de aprendizaje automático los clasifican o agrupan de forma coherente.

La investigación se desarrolla únicamente en laboratorio, mediante servicios simulados y contenedores, sin afectar infraestructuras reales. Por lo tanto, quedan fuera del alcance la protección directa de sistemas en producción, el bloqueo automático de amenazas, la respuesta a incidentes y el análisis forense completo. Esta delimitación permite que el trabajo sea medible y coherente.

## 1.4. Objetivos

### *1.4.1. Objetivo general*

Desarrollar un honeypot basado en API REST para la detección y clasificación de patrones de ataque mediante técnicas de Machine Learning integradas con Elastic Stack para el análisis de eventos de seguridad.

### *1.4.2. Objetivos específicos*

- Identificar las vulnerabilidades y patrones de ataque más comunes dirigidos a servicios API REST.
- Diseñar y desarrollar un honeypot de baja interacción capaz de simular endpoints vulnerables de una API REST.

- Implementar un sistema de captura, almacenamiento y visualización de eventos mediante la integración de Elastic Stack.
- Generar y procesar registros de tráfico HTTP para la construcción de un conjunto de datos orientado al análisis de eventos de seguridad.
- Aplicar técnicas de Machine Learning para la detección de anomalías y clasificación de patrones de ataque sobre los logs recopilados.
- Evaluar el desempeño de los modelos implementados mediante métricas de clasificación y análisis de anomalías.
- Validar el funcionamiento del sistema mediante pruebas controladas de ataques dirigidos a APIs REST.

# Capítulo 2

# 2. Revisión de la literatura

## 2.1. Estado del Arte

El uso combinado de señuelos digitales y aprendizaje automático para detectar amenazas en servicios web lleva más de una década consolidándose como área de investigación y el motivo es sencillo: las amenazas han cambiado tan rápido que los enfoques clásicos ya no alcanzan. Los estudios que se presentan a continuación son los que dan base a la arquitectura propuesta en este trabajo.

Todo parte de una idea simple pero poderosa. Spitzner (2003) fue uno de los primeros en formalizarla: un honeypot es un recurso que existe únicamente para ser explorado y atacado, sin ningún propósito legítimo. El problema es que, en sus versiones iniciales, todo el análisis dependía del ojo humano, lo cual limitaba enormemente su escala. Años después, Garcia-Teodoro et al. (2009) pusieron en evidencia otra limitación, esta vez en los sistemas de detección basados en anomalías de red: en entornos de producción reales, el tráfico legítimo de los usuarios genera tantas alertas falsas que el sistema se vuelve poco confiable.r

Frente a eso, la investigación tomó dos caminos que terminaron convergiendo. Por un lado, Ahmed et al. (2016) mostraron que los algoritmos de agrupamiento no supervisado, como K-Means, pueden descubrir patrones en el tráfico de red sin necesidad de conocerlos de antemano. Por otro lado, Buczak y Guven (2016) demostraron que entrenar modelos de Machine Learning sobre registros de auditoría permite anticipar ataques automatizados que las reglas estáticas tradicionales simplemente no ven venir.

Claro que todo esto depende de con qué datos se trabaje. Fraunholz et al. (2020) confirmaron que desplegar señuelos controlados produce registros limpios y confiables, además

de reducir el riesgo de que un atacante se mueva lateralmente dentro de la red. Sin embargo, Sarker et al. (2020) advirtieron que de poco sirve un buen modelo si se entrena con datos desactualizados: para que los algoritmos predictivos funcionen bien en la práctica, los datos tienen que fluir de forma continua y mantenerse al día.

Y el contexto actual agrega una capa más de urgencia. Como señalan Sharma et al. (2021), las arquitecturas API REST, con sus endpoints públicos, se han convertido en el blanco favorito de las redes de bots automatizados, lo que hace que monitorear ese nivel específico del tráfico ya no sea opcional, sino una necesidad.

**Tabla 1**

*Matriz Comparativa del Estado de Arte*

| Autor y Año | Enfoque Tecnológico | Tipo de Datos | Técnica Analítica Principal | Limitación o Brecha Identificada | Solución Planteada en la Tesis |
|---|---|---|---|---|---|
| Spitzner (2003) | Tecnologías de engaño y análisis forense. | Conexiones de red crudas. | Inspección y revisión manual de logs. | Enfoque primitivo y lento; carece de automatización analítica. | Implementa un pipeline automatizado con Elastic Stack para procesar eventos en tiempo real. |
| Garcia-Teodoro et al. (2009) | Detección de intrusos basada en anomalías. | Tráfico real de producción corporativa. | Firmas y reglas estáticas de red. | Alta tasa de falsos positivos provocada por el ruido de usuarios benignos. | Emplea un ambiente señuelo aislado donde todo tráfico capturado es deterministamente una amenaza. |
| Ahmed et al. (2016) | Descubrimiento de amenazas emergentes. | Paquetes de red generales (IP/Puerto). | Algoritmo no supervisado K-Means. | No analiza variables de la capa de aplicación (OSI 7) ni la lógica de las APIs. | Configura la ingeniería de características basándose en metadatos y payloads HTTP del protocolo REST. |
| Buczak & Guven (2016) | IA aplicada a la intrusión perimetral. | Registros de auditoría de seguridad. | Algoritmia predictiva de Machine Learning. | Alta dependencia de la calidad, balanceo y limpieza previa del dataset de entrenamiento. | Utiliza un generador controlado de tráfico sintético para obtener un dataset balanceado (Clase 0 y 1). |

| Autor y Año | Enfoque Tecnológico | Tipo de Datos | Técnica Analítica Principal | Limitación o Brecha Identificada | Solución Planteada en la Tesis |
|---|---|---|---|---|---|
| Fraunholz et al. (2020) | Eficiencia en la recolección de eventos de ataque. | Logs distribuidos de honeypots. | Clasificación taxonómica pasiva. | Riesgo operativo si el atacante logra comprometer el sistema operativo base. | Adopta una arquitectura de microservicios de media interacción contenerizados con Docker. |
| Sarker et al. (2020) | Ciencia de Datos en Ciberseguridad. | Repositorios y datasets históricos públicos. | Modelos basados en ensamble (Random Forest). | Dependencia de datos estáticos; no integra telemetría o ingesta en tiempo real. | Diseña una tubería de datos (data pipeline) continua conectando logs crudos con Elasticsearch. |
| Sharma et al. (2021) | Vulnerabilidades en servicios web. | Endpoints expuestos a internet. | Análisis conceptual y documental de fallos. | Es una revisión teórica; no ejecuta captura dinámica ni mitigación activa de bots. | Desarrolla un entorno simulado interactivo que emula una API REST bancaria funcional. |

*Nota.* Matriz elaborada por los autores a partir de la revisión de la literatura científica seleccionada para el estado del arte (2026).

## 2.2. Marco Teórico

### *2.2.1. APIs REST y Vulnerabilidades Asociadas*

Las interfaces de programación de aplicaciones basadas en la transferencia de estado representacional (APIs REST) se han convertido en el estándar en la industria para la intercomunicación en el desarrollo del software moderno, impulsando ecosistemas distribuidos, arquitecturas de microservicios y soluciones en la nube (Masse, 2011).

Este paradigma, definido por Fielding en el año 2000, no es un protocolo rígido con reglas estrictas, sino más bien un conjunto de principios de diseño que saca provecho de las capacidades propias del protocolo HTTP. La idea central es que la comunicación entre el cliente y el servidor sea independiente y sin memoria: cada vez que el cliente hace una solicitud, debe incluir toda la información necesaria para que el servidor la entienda y la procese, sin necesidad de recordar nada de interacciones anteriores.

Para entender cómo se diseña y opera un sistema de detección y engaño (Honeypot) en este entorno, es necesario conocer los componentes técnicos básicos que forman parte de cualquier transacción REST. Estos elementos son importantes por dos razones: son los puntos donde un sistema queda expuesto al exterior y, al mismo tiempo, son las principales fuentes de evidencia digital cuando ocurre un ataque (Chantzis et al., 2021).

**Rutas y Endpoints (URIs):** son las direcciones que identifican cada recurso o función dentro del sistema, como por ejemplo /api/v1/payment/checkout o /api/v1/users. Los atacantes automatizados las escanean de forma sistemática buscando puntos de entrada vulnerables (Sethi y Jaiswal, 2021).

**Métodos HTTP (Verbos):** indican qué tipo de acción se quiere realizar sobre un recurso. Los más utilizados y también los más explotados son GET (para consultar datos), POST (para crear recursos o enviar credenciales), PUT (para modificar un recurso completo) y DELETE (para eliminarlo). Analizar estos verbos permite distinguir entre una consulta normal y un intento de manipulación del sistema (Boyd y Mathur, 2018).

**Cabeceras HTTP (Headers):** son bloques de información adicional que acompañan cada petición. Entre los más relevantes están el campo User-Agent, que identifica qué software o bot está haciendo la solicitud, y las cabeceras de autorización como Authorization: Bearer <token>, que permiten verificar si el acceso es legítimo (Deirmentzoglou et al., 2019).

**Cuerpo de la petición (Payload):** es el contenido que se envía dentro de una petición POST o PUT, generalmente en formato JSON o XML. Este componente es el principal vehículo para introducir ataques, ya sea mediante código malicioso, inyecciones o intentos de fuerza bruta con listas de credenciales (Belayev y Ivanova, 2022).

El crecimiento acelerado de estas tecnologías ha ampliado considerablemente la superficie de ataque a la que están expuestos los servicios web. Dado que las aplicaciones modernas dependen de endpoints públicos para comunicarse con otros sistemas, pasarelas de pago y aplicaciones móviles, cualquier error en la configuración de seguridad o una validación inadecuada de los datos de entrada puede comprometer recursos internos críticos y provocar filtraciones masivas de información sensible (Sharma et al., 2021).

Según la OWASP Foundation (2023), en el OWASP API Security Top 10, las amenazas de las APIs no se limitan a errores técnicos visibles, sino que también aparecen en la autorización, la autenticación y la lógica propia del negocio. En este proyecto estas amenazas se representan de manera controlada dentro del honeypot, para que cada interacción quede reflejada en los logs mediante campos como endpoint_accedido, método_http, datos_peticion, headers_recibidos y tags.

**API1:2023 – Autorización rota a nivel de objeto (BOLA):** ocurre cuando un atacante manipula los identificadores de recursos dentro de los parámetros de consulta con el fin de acceder a información que pertenece a otros usuarios, evadiendo los controles de acceso perimetrales. Esta amenaza se representa activamente en el honeypot a través de los endpoints /api/v1/users y /dashboard. Cuando las herramientas de escaneo alteran de forma secuencial o maliciosa las variables de consulta, por ejemplo, ?id=1042, el framework intercepta el payload, almacena los argumentos dentro del objeto datos_peticion, registra la URI en el campo endpoint_accedido y asigna automáticamente el tag semántico bola_attempt.

**API2:2023 – Autenticación rota:** permite que herramientas o scripts automatizados realicen ataques de fuerza bruta o credential stuffing contra los formularios de inicio de sesión, probando combinaciones masivas de usuarios y contraseñas. El sistema modela este riesgo en las

rutas /api/v1/auth/login y /login. Cada intento automatizado queda registrado, capturando las credenciales inyectadas en la variable datos_peticion bajo la etiqueta login_attempt. Con el fin de prolongar el tiempo de interacción y recopilar una mayor cantidad de evidencia digital, el honeypot implementa una contramedida de engaño activo que genera una sesión falsa exitosa (session['logged_in'] = True), incitando al atacante a continuar explorando el entorno simulado.

**API6:2023 – Manipulación de la lógica de negocio:** consiste en la explotación de flujos de trabajo legítimos mediante la alteración de parámetros críticos para evadir las reglas operativas del negocio. Esta vulnerabilidad se representa en el endpoint transaccional /api/v1/payment/checkout. Los payloads modificados que buscan alterar el comportamiento de la pasarela de pagos son capturados íntegramente en el registro, lo que permite al script de Machine Learning calcular la longitud del texto (payload_length) para aislar anomalías volumétricas, asociando cada evento de forma unívoca bajo el tag payment_exploit.

**Escaneo automatizado de directorios y bots:** representa la actividad de reconocimiento preliminar en la que bots especializados buscan descubrir endpoints ocultos, archivos expuestos o configuraciones sensibles (por ejemplo, /.env, /.git/config). Para interceptar esta amenaza, el honeypot incorpora una regla de enrutamiento global (catch-all). Cualquier solicitud dirigida a un recurso inexistente queda atrapada por esta función, que guarda la ruta sospechosa en el campo endpoint_accedido bajo la etiqueta lógica scanner.

Ante este escenario de exposición constante, las organizaciones han recurrido tradicionalmente a herramientas de monitoreo pasivo del tráfico HTTP y a sistemas de protección perimetral como los Firewalls de Aplicación Web (WAF). Sin embargo, aplicar este tipo de inspección en entornos de producción reales presenta una limitación operativa crítica: la enorme cantidad de falsos positivos y el ruido generado por las transacciones de usuarios

legítimos dificultan que los analistas distingan entre las operaciones ordinarias y las técnicas de ataque más sofisticadas. Es precisamente ahí donde surge la necesidad técnica de incorporar tecnologías de engaño como un complemento crítico al monitoreo tradicional (Garcia-Teodoro et al., 2009).

Al desplegar un honeypot que imita con fidelidad la estructura, las cabeceras y el comportamiento visual de una API REST corporativa, se elimina por completo el ruido del tráfico legítimo. La razón metodológica es sencilla: al tratarse de un sistema completamente aislado, sin usuarios reales ni funciones operativas dentro de la red, cualquier solicitud o interacción dirigida hacia sus endpoints resulta, por definición, sospechosa. Esto permite recopilar trazas JSON limpias y libres de falsos positivos, lo cual facilita el preprocesamiento automatizado en estructuras de Pandas y da lugar a un conjunto de datos altamente confiable para entrenar algoritmos de Machine Learning capaces de clasificar y anticipar intrusiones complejas (Sarker et al., 2020).

### *2.2.2. Honeypots y Tecnologías del Engaño*

Los honeypots son herramientas avanzadas de ciberseguridad proactiva que forman parte de lo que se conoce como tecnologías de engaño. A diferencia de los mecanismos de defensa tradicionales, como los sistemas de detección de intrusos o los firewalls, que actúan de forma reactiva bloqueando amenazas conocidas, las tecnologías de engaño cambian las reglas del juego: introducen activos señuelo dentro de la red corporativa para desviar a los atacantes hacia entornos controlados y aislados, frenando su avance y permitiendo recopilar información valiosa sobre sus herramientas y métodos (Fraunholz et al., 2020).

Según la definición clásica de Spitzner (2003), un honeypot es un recurso de seguridad cuyo valor está precisamente en ser explorado, atacado o comprometido. Al no tener ninguna

función operativa real ni usuarios legítimos, cualquier interacción dirigida hacia él es, por definición, sospechosa o maliciosa.

En el contexto de las arquitecturas de servicios modernas, combinar las tecnologías de engaño con el protocolo HTTP resulta especialmente relevante. Un honeypot diseñado para este entorno no imita un sistema operativo genérico, sino que expone directamente una serie de endpoints REST estructurados, como por ejemplo /api/v1/auth/login. Cuando un bot o script automatizado interactúa con estos endpoints falsos enviando solicitudes HTTP, el honeypot responde simulando ser una aplicación real en producción, como una plataforma bancaria (Provos y Holz, 2007).

El valor de este enfoque está en su sistema de captura de registros: cada solicitud HTTP entrante es analizada en tiempo real para extraer sus variables clave, como la dirección IP de origen, los métodos utilizados, las cabeceras, los parámetros de la URL y el contenido del cuerpo de la petición. Esta información se almacena de forma continua en archivos de registro, convirtiendo el comportamiento del atacante en un conjunto de datos estructurado y libre de ruido, listo para análisis forense o para alimentar modelos predictivos (Shiravi et al., 2012).

La literatura especializada clasifica los honeypots según su nivel de interacción, es decir, cuánta libertad se le otorga al atacante y qué tan compleja es la infraestructura simulada. Los honeypots de alta interacción utilizan sistemas operativos reales, servicios completos y bases de datos genuinas, lo que permite capturar actividades complejas. Sin embargo, esto también implica un mayor riesgo, ya que el atacante podría comprometer el servidor y moverse lateralmente dentro de la red.

Los honeypots de baja interacción, en cambio, simulan únicamente la capa de servicios o aplicaciones específicas, limitando las respuestas a un conjunto predefinido sin dar acceso a una

terminal real del sistema. Para esta investigación, se optó por un honeypot de baja interacción por tres razones técnicas concretas:

**Reducción del riesgo operativo:** al simular únicamente la lógica de respuesta de los endpoints HTTP sin exponer una consola de comandos, se elimina la posibilidad de que un atacante comprometa el servidor base o escale privilegios en el sistema (Dowd et al., 2006).

**Eficiencia en el uso de recursos:** las soluciones de baja interacción consumen muy poca memoria y procesamiento, lo que permite desplegarlas fácilmente como microservicios contenerizados y ejecutar múltiples instancias de forma simultánea sin afectar el rendimiento del hardware (Merkel, 2014).

**Datos suficientes para el análisis:** dado que el objetivo principal de esta investigación es entrenar modelos de Machine Learning para clasificar patrones de tráfico automatizado, las variables capturadas en la capa HTTP, como frecuencia de solicitudes, métodos utilizados, rutas consultadas y códigos de respuesta, son más que suficientes para este propósito, sin necesidad de exponer un sistema operativo real.

### *2.2.3. Elastic Stack*

Gestionar la seguridad informática hoy en día requiere infraestructuras capaces de absorber, organizar y analizar grandes volúmenes de eventos en tiempo casi real. En entornos expuestos a ataques automatizados, donde un bot puede generar miles de solicitudes por segundo, los métodos tradicionales basados en la lectura manual de archivos de registro colapsan de inmediato. Para superar esta limitación, esta investigación adopta el ecosistema Elastic Stack, una plataforma de código abierto orientada al procesamiento de datos, el análisis distribuido y la observabilidad de sistemas complejos (Turnbull, 2014). Para construir la arquitectura final de

este proyecto, se seleccionaron tres componentes principales de este stack, eligiéndolos en base a criterios de eficiencia, escalabilidad y compatibilidad:

**Filebeat (agente de transporte ligero):** actúa como recolector y transmisor de registros con un consumo de memoria muy reducido. Se eligió por encima de herramientas más pesadas como Logstash porque opera directamente junto al contenedor del honeypot, leyendo en tiempo real las nuevas líneas que se añaden al archivo de registros sin afectar el rendimiento del servicio web simulado. Además, cuenta con un mecanismo de control de flujo que regula el envío de datos cuando el sistema central está saturado (Aggarwal, 2016).

**Elasticsearch (motor central de indexación y almacenamiento):** es el núcleo analítico del ecosistema y funciona como base de datos no relacional (NoSQL). Basado en la biblioteca Apache Lucene, almacena la información de seguridad en documentos JSON. Su elección se justifica por su capacidad de indexación distribuida y su estructura flexible, que le permite procesar grandes ráfagas de solicitudes HTTP simultáneas y ejecutar búsquedas o consultas de agregación en milisegundos (Gormley y Tong, 2015).

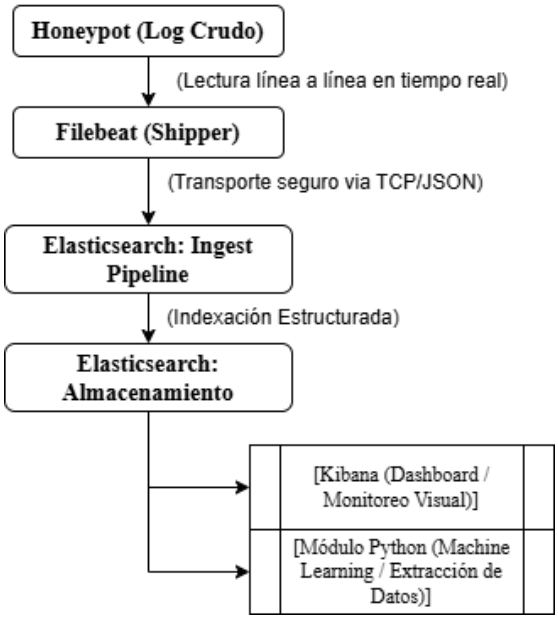
Kibana (centro de visualización y monitoreo): es la interfaz visual de la arquitectura. Se integra por su sincronización nativa con los índices de Elasticsearch, lo que permite construir paneles interactivos y gráficos temporales que traducen miles de registros técnicos en métricas comprensibles e inteligencia de amenazas útil para el analista (Chaganti et al., 2022).

### *2.2.4. Flujo y Pipeline de Integración de Datos*

El valor del Elastic Stack en esta investigación no está en cada componente por separado, sino en cómo trabajan juntos como una cadena de procesamiento continua que convierte un evento en bruto en información útil para la toma de decisiones. Este flujo se compone de cinco etapas secuenciales:

**Figura 1**

*Flujo y tubería de integración de datos (Data Pipeline) en el ecosistema Elastic Stack*




*Nota. E*laboración propia en base a la revisión de literatura, 2026.

El diagrama ilustra las cinco etapas secuenciales de la arquitectura de telemetría: desde la captura de la interacción HTTP en el honeypot (log crudo) hasta su almacenamiento distribuido en Elasticsearch y posterior procesamiento en Kibana o extracción para los modelos de Machine Learning.

### *2.2.5. Captura y Envío*

Cuando el honeypot registra una interacción en alguno de sus endpoints, la función centralizada log_request() captura en tiempo real los metadatos de la solicitud HTTP. Esta función estructura la información en formato JSON y la escribe como una nueva línea inmutable en el archivo honeypot_attacks.log. El agente ligero Filebeat, desplegado de forma adyacente dentro del entorno de contenedores, detecta casi instantáneamente cualquier modificación del archivo en la ruta asignada, empaqueta esa línea junto con los metadatos de infraestructura correspondientes y la envía mediante el protocolo TCP hacia el motor central de ingesta.

### *2.2.6. Procesamiento y Normalización*

Una vez que Elasticsearch recibe el registro en bruto, ejecuta un proceso de deserialización nativa para interpretar la cadena de texto y convertirla en un documento indexable. En esta etapa, el evento se descompone en las variables lógicas definidas previamente en el código fuente de la aplicación: timestamp, ip_origen, puerto_origen, metodo_http, endpoint_accedido, headers_recibidos, datos_peticion y la lista de etiquetas semánticas tags. En la fase de procesamiento y normalización, decidimos conservar y estructurar el código de respuesta HTTP (status_code) dentro del registro final de logs, en lugar de descartarlo. Mantener este dato es clave para cualquier analista en un SOC, ya que ayuda a cruzar rápidamente un intento de ataque con el impacto real en el sistema. Por ejemplo, permite ver al instante si un payload malicioso provocó una caída del servicio (error 500) o si un escaneo de fuerza bruta generó múltiples respuestas de acceso denegado (error 401). De esta forma, el código de estado se mapea desde que el honeypot recibe la petición hasta que se almacena en Elasticsearch, manteniendo la coherencia con los esquemas de datos y las pruebas del gateway de pagos descritas más adelante

### *2.2.7. Indexación y Almacenamiento*

Los documentos JSON ya normalizados y validados se almacenan en índices cronológicos dentro de Elasticsearch, organizados bajo el patrón honeypot-api-logs*. Este motor NoSQL, basado en Apache Lucene, indexa de manera independiente los campos clave, como el método HTTP y la ruta accedida, lo que garantiza que la información de los ataques permanezca disponible de forma permanente, con acceso rápido y protegida frente a alteraciones, incluso si el contenedor base del honeypot se reinicia o se destruye.

### *2.2.8. Visualización y Monitoreo*

Mientras se lleva a cabo la indexación, la interfaz de Kibana realiza consultas continuas sobre los índices activos para mantener actualizados los paneles y tableros de control en tiempo real. Esto le permite al analista de seguridad observar visualmente el comportamiento del tráfico, detectar picos volumétricos inusuales y filtrar las solicitudes según las firmas lógicas definidas en el laboratorio, como los intentos de inyección y automatización clasificados bajo las etiquetas login_attempt, bola_attempt, payment_exploit o scanner.

### *2.2.9. Extracción para Análisis*

La base de datos indexada y estructurada en Elasticsearch queda disponible para la etapa analítica avanzada. El módulo independiente de Machine Learning (ml_anomalias.py) extrae de forma automatizada las colecciones de documentos históricos y las transforma en una tabla bidimensional de Pandas (DataFrame) (Ramalho, 2022). A partir de este conjunto de datos limpio, el script realiza un proceso de ingeniería de características (feature engineering) que permite aislar el campo user_agent y calcular la longitud exacta del cuerpo de los datos enviados (payload_length), generando así variables numéricas, estandarizadas y confiables para el posterior entrenamiento de los modelos.

### *2.2.10. Machine Learning Aplicado a la Detección de Ataques*

El aprendizaje automático (Machine Learning) constituye una rama fundamental de la Inteligencia Artificial centrada en el desarrollo de algoritmos y sistemas capaces de identificar patrones complejos, descubrir correlaciones ocultas y emitir predicciones lógicas a partir de conjuntos de datos empíricos, sin necesidad de ser programados explícitamente mediante reglas heurísticas rígidas (Mitchell, 1997). En el ámbito de la ciberseguridad corporativa, este enfoque permite clasificar vectores anómalos de manera proactiva, superando las limitaciones de los

mecanismos tradicionales basados puramente en firmas estáticas o expresiones regulares, los cuales no logran adaptarse a las constantes mutaciones del tráfico web (Buczak & Guven, 2016).

#### *2.2.11. Tubería de Preprocesamiento e Ingeniería de Características*

La convergencia matemática y la capacidad predictiva de los modelos de Inteligencia Artificial dependen críticamente de la calidad, depuración y estructuración de sus variables de entrada. El pipeline de ingeniería de datos se implementa de forma automatizada en el módulo analítico del proyecto (ml_anomalias.py), bajo los siguientes criterios de control:

**Origen del Dataset y Verdad de Campo.** Para mitigar el desbalanceo de datos, un problema común en entornos de producción, la investigación recurre al entorno de simulación avanzada simulador_trafico.py, encargado de inyectar flujos controlados de tráfico legítimo junto con ráfagas correspondientes al estándar OWASP API Security Top 10, mediante hilos concurrentes. La asignación de la verdad de campo (y) se determina a través de una regla booleana sobre la variable de control is_attack: se asigna la Clase 1 (Ataque) cuando el registro contiene alguno de los tags semánticos provistos por el honeypot (bola_attempt, payment_exploit o scanner), y la Clase 0 (Benigno) para las interacciones ordinarias.

**Criterios de limpieza.** El script ejecuta rutinas de depuración mediante el manejo de excepciones orientadas a errores de decodificación (json.JSONDecodeError), lo que permite aislar y descartar trazas corruptas o incompletas antes de consolidar la información en estructuras tabulares (DataFrame de la librería Pandas) (McKinney, 2010).

**Ingeniería de Características (features).** Para representar matemáticamente las transacciones HTTP dentro de la matriz de características (X), el modelo procesa cuatro atributos específicos extraídos de la capa de aplicación:

- **method_encoded:** codificación numérica discreta del método HTTP utilizado (metodo_http).
- **endpoint_encoded:** codificación discreta del recurso web interceptado (endpoint_accedido).
- **user_agent_encoded:** índice asignado al software de origen (User-Agent), extraído del diccionario de cabeceras de la petición (headers_recibidos).
- **payload_length:** variable analítica continua, calculada a partir de la longitud exacta de la cadena de texto transportada en el cuerpo de los datos (datos_peticion).

#### *2.2.12. Normalización y particionado*

Los atributos cualitativos se normalizan mediante codificadores de etiquetas (Label Encoder), y la matriz completa se estandariza posteriormente a través de un escalador estándar (Standard Scaler), con el fin de unificar las magnitudes geométricas de las variables. Para la fase de validación, el conjunto de datos se divide reservando aleatoriamente el 30% de las muestras para las pruebas de rendimiento (test_size=0.3), lo que garantiza una evaluación imparcial del modelo supervisado (Kuhn & Johnson, 2013).

#### *2.2.13. Justificación del Enfoque Supervisado: Random Forest*

Para resolver la tarea de clasificación binaria, se optó por el algoritmo de conjunto Random Forest (Breiman, 2001), configurado con 100 estimadores independientes (Pedregosa et al., 2011).

Este modelo resulta especialmente adecuado para los datos provenientes de logs HTTP, gracias a su arquitectura basada en múltiples árboles de decisión paralelos, alimentados mediante la técnica de agregación de muestras (bagging). Al procesar variables tabulares mixtas, Random Forest destaca por su alta tolerancia a la multicolinealidad, su resistencia inherente al sobreajuste

(overfitting) y su capacidad para trazar fronteras de decisión no lineales capaces de aislar firmas automatizadas complejas, como las generadas por herramientas de escaneo o inyecciones de parámetros. Su evaluación metodológica se apoya en herramientas diagnósticas como la Matriz de Confusión, que permite auditar falsos positivos y falsos negativos, junto con los reportes de Precisión, Exhaustividad (Recall) y puntuación F1-Score.

#### *2.2.14. Justificación del Enfoque No Supervisado: K-Means*

Considerando que, en escenarios de producción reales, los atacantes avanzados suelen alterar sutilmente sus comportamientos para evadir firmas y clasificadores estáticos, la arquitectura incorpora de forma complementaria el algoritmo no supervisado K-Means (Ahmed et al., 2016; Zheng & Wang, 2020). Este algoritmo resulta idóneo para la detección proactiva de anomalías de día cero, ya que opera completamente al margen de etiquetas previas, agrupando los registros de manera iterativa en un hiperparámetro fijo de clústeres, con base en la distancia euclidiana dentro del espacio multidimensional escalado (MacQueen, 1967).

La premisa metodológica de este diseño parte de que el tráfico legítimo ordinario tiende a concentrarse en núcleos densos y compactos de alta frecuencia estadística, mientras que los vectores maliciosos automatizados generan desviaciones vectoriales que los llevan a aislarse en agrupaciones periféricas independientes. La validación científica de este modelo se realiza mediante tablas de contingencia cruzada (crosstab), lo que permite analizar la pureza de los grupos construidos por el algoritmo frente a la realidad observada en el tráfico interceptado.

Cabe destacar que bajo este enfoque matemático se contempla como patrón esperado que ciertas peticiones ordinarias complejas (como solicitudes transaccionales de tipo POST que transportan cuerpos de datos extensos) muestren distancias geométricas similares a un vector de ataque, siendo agrupadas inicialmente dentro del clúster anómalo. Este comportamiento justifica

conceptualmente el diseño híbrido de la investigación, delegando en el clasificador supervisado en serie (Random Forest) la tarea secundaria de discriminación fina para erradicar los falsos positivos antes de la toma de acciones operativas en la infraestructura.

# Capítulo 3:

# 3. Desarrollo

En el presente capítulo se describe el diseño e implementación del prototipo funcional desarrollado para la investigación. El entorno integra un honeypot de baja interacción basado en API REST, un flujo de captura y transporte de logs con Filebeat, almacenamiento e indexación en Elasticsearch, visualización en Kibana y un módulo de aprendizaje automático construido con Random Forest y K-Means. La finalidad de esta sección es mostrar cómo se conectan los componentes y qué evidencia permite comprobar que el sistema funciona de extremo a extremo.

El honeypot se desarrolló en Python 3 con Flask, porque este framework permite crear rutas HTTP de forma rápida, controlar respuestas JSON y registrar las peticiones de manera centralizada. La aplicación simula una plataforma web con formulario de acceso, panel interno y endpoints API y luego se envía a Elasticsearch para su consulta desde Kibana. Esta decisión mantiene el riesgo controlado y, al mismo tiempo, permite construir un dataset propio para las pruebas de Machine Learning.

El capítulo se organiza de la siguiente manera: primero, se presenta la arquitectura del honeypot y las capas. Luego, se describen los endpoints y escenarios de explotación. Después, se explica la función log_request(), el esquema del evento y el etiquetado semántico. Posteriormente, se detalla la ingesta con Elastic Stack y el uso de Kibana. Finalmente, se desarrolla el módulo de Machine Learning, incluyendo extracción de datos, limpieza, transformación de variables, entrenamiento, validación y lectura de resultados.

## 3.1. Arquitectura del honeypot

El presente trabajo de titulación propone un honeypot de baja interacción como mecanismo de recolección de actividad sospechosa. Esta decisión es coherente con el objetivo

del proyecto, ya que se necesita capturar solicitudes HTTP, payloads, cabeceras, rutas consultadas y patrones de automatización, sin permitir que el atacante controle un sistema operativo real. En esta línea, Montiel et al. (2022) explican que los honeypots pueden actuar como entornos controlados para observar intentos de intrusión. Sin embargo, mientras mayor sea la interacción, mayor será también el riesgo operativo. Por lo tanto, para este caso, la baja interacción es suficiente y más segura.

Para lograr la emulación del sistema se utilizó Python 3 junto con Flask para el manejo de rutas, solicitudes HTTP y respuestas simuladas. Flask resulta adecuado para este prototipo por su bajo consumo de recursos y por la facilidad que ofrece para crear endpoints REST. La aplicación escucha peticiones en el puerto 8080, se ejecuta con un contenedor Docker y escribe sus registros en la ruta /var/log/honeypot/honeypot_attacks.log, que luego es leída por Filebeat.

El despliegue se realizó con Docker Compose para levantar de forma conjunta el honeypot, Filebeat, Elasticsearch y Kibana. El contenedor del honeypot expone el puerto 8080. Por otro lado, Filebeat lee el archivo de logs desde el volumen compartido. Asimismo, Elasticsearch recibe los eventos y los almacena bajo el patrón de índice honeypot-api-logs*. Finalmente, Kibana utiliza ese patrón para crear visualizaciones y filtros. Esta arquitectura permite repetir las pruebas en otro equipo sin depender de configuraciones manuales del sistema operativo.

El honeypot implementado se organiza en tres capas relacionadas entre sí:

### *3.1.1. Capa de presentación*

Esta capa muestra al atacante una interfaz sencilla y creíble, compuesta por el formulario /login y el panel / dashboard. Su entrada son solicitudes HTTP GET o POST y su salida son

respuestas HTML generadas con plantillas Jinja2. La función de esta capa es sostener la interacción inicial y dirigir cada petición hacia la capa de lógica sin exponer un servicio real.

Esta capa también ayuda a obtener evidencia de navegación, ya que los accesos a la pantalla principal, al formulario y al panel quedan registrados con etiquetas como home, frontend, login_view o bola_attempt, según el caso.

#### *3.1.2. Capa de lógica*

Esta capa procesa la acción solicitada y devuelve respuestas HTML o JSON. Por ejemplo, cuando se recibe un intento de inicio de sesión, el sistema puede simular una respuesta válida o inválida para mantener la interacción, pero sin validar usuarios reales. Su entrada son parámetros, cuerpos JSON o consultas en URL. Por otra parte, su salida son respuestas controladas y datos falsos. En esta capa se representan los escenarios de fuerza bruta, BOLA, manipulación de pagos, SQLi, XSS y escaneo de rutas.

#### *3.1.3. Capa de Registro*

Esta es el eje del prototipo. Antes de responder a cada solicitud, la función log_request() captura el momento del evento, IP de origen, puerto, método HTTP, endpoint accedido, cabeceras, datos enviados y etiquetas semánticas. El resultado se guarda como una línea JSON dentro de honeypot_attacks.log. Este archivo se convierte en la fuente principal para Elastic Stack y el módulo de Machine Learning, por lo que conecta directamente la interacción simulada con el análisis posterior.

### 3.2. Modelado de endpoints y escenarios de explotación

Los endpoints implementados en el honeypot se basaron en la reproducción de la superficie de ataque convencional en una aplicación web que cuenta con servicios API expuestos. Los endpoints se diseñaron para una gran cobertura en cuanto a ataques, cada ruta

simula un componente real funcional y que registra de forma centralizada toda la actividad entrante, esto permite una mayor diversidad de ataques y comportamientos maliciosos con los que es posible construir un dataset de ataques más completo.

El sistema cuenta con dos grupos de endpoints. El primer grupo corresponde a la interfaz visible, formada por /login y /dashboard. El segundo grupo corresponde a servicios API REST accesibles mediante solicitudes HTTP, entre ellos /api/v1/auth/login, /api/v1/users y /api/v1/payment/checkout. Además, se configuró una ruta de captura global para registrar cualquier endpoint no definido, como /.env, /.git/config o /admin, que normalmente son buscados por bots durante una fase de reconocimiento.

La evidencia de que los endpoints simulan una API REST funcional se obtiene de tres elementos: primero, las rutas responden con estructuras JSON; segundo, se utilizan métodos y códigos HTTP coherentes con cada caso; y tercero, cada interacción produce un registro con campos técnicos reutilizables. De esta manera, el prototipo no solo muestra pantallas falsas, sino que reproduce el comportamiento mínimo necesario para generar datos válidos de análisis.

Los endpoints, en respuesta a las solicitudes recibidas, actúan de manera fiel a como lo haría una aplicación web legítima. Los servicios generados como el login, el dashboard y los demás servicios responden con objetos JSON según la interacción del usuario, esto es útil para que parezcan lo suficientemente reales como para mantener la interacción del atacante y mejorar los registros capturados.

**Tabla 2**

*Endpoints Implementados en el Honeypot*

| Método | Ruta | Payload | Respuesta | Vulnerabilidad simulada | Tag generado |
|---|---|---|---|---|---|
| GET | / | --- | 200 OK | Reconocimiento | Frontend, home |
| GET/POST | /login | Username, password | 302 Redirect | Autenticación no segura | Frontend, login_attempt |

| Método | Ruta | Payload | Respuesta | Vulnerabilidad simulada | Tag generado |
|---|---|---|---|---|---|
| GET | /dashboard?id={id} | Id en query | 200 OK | BOLA | Frontend, bola_attempt |
| POST | /api/v1/auth/login | JSON username, password | 401 No autorizado | Exposición de error de autenticación | Api, login_attempt |
| GET/POST | /api/v1/users?id={id} | Id en query | 200 OK | BOLA | Api, bola_attempt |
| GET/POST | /api/v1/payment/checkout | JSON | 500 JSON | Misconfiguration | Api, payment_exploit |
| ANY | Ruta no definida | Cualquiera | 404 JSON | Enumeración de rutas | Catch_all, scanner |

*Nota.* Elaboración propia en base al trabajo investigativo

En la siguiente tabla se detalla la cobertura de los ataques realizados al sistema y escenarios de explotación que fueron implementados especificando herramienta utilizada, endpoint objetivo, ataque simulado, eventos generados, evidencia y resultados esperados.

**Tabla 3**

*Escenarios de explotación implementados en el honeypot*

| Escenario | Herramienta usada | Endpoint Objetivo | Tipo de ataque | Eventos generados | Evidencia recolectada | Resultado esperado |
|---|---|---|---|---|---|---|
| Fuerza bruta en autenticación | "simulador_trafico.py" función "ataque_fuerza_bruta()" | /api/v1/auth/login | Fuerza bruta | 5 eventos cada ciclo | Credenciales, ip de origen, user-agent, tag | Clasificación de ataque según el modelo supervisado. |
| Enumeración de rutas | "simulador_trafico.py" función "ataque_enumeracion()" | /.env, /backup.zip, /admin/hidden, /swagger-ui.html, /api/v1/config | Escaneo automatizado de directorios | 5 eventos cada ciclo | Endpoint accedido, user-agent, tag. | Clasificación como tráfico de reconocimiento. |
| Inyección SQL y XSS | "simulador_trafico.py" función "inyección_sql()" e "inyección_xss()" | /api/v1/users, /search | Inyección de código | 1 evento por ciclo | Payload malicioso, user-agent, tag | Clasificación según la longitud del payload. |
| Acceso indebido a objetos | "simulador_trafico.py" tráfico hacia | /dashboard?id={id}, | Broken object level | Dependiendo de interacción | Id modificado en la URL, tag, datos falsos | Enumeración de recursos |

| Escenario | Herramienta usada | Endpoint Objetivo | Tipo de ataque | Eventos generados | Evidencia recolectada | Resultado esperado |
|---|---|---|---|---|---|---|
| | "/dashboard" y "/api/v1/users" con ID manipulado | /api/v1/users ?id={id} | authorization | | retornados por el honeypot. | |
| Explotación del Gateway de pagos | "simulador_trafico.py" tráfico hacia "/api/v1/payment/checkout" | /api/v1/payment/checkout | Manipulación de lógica de negocio | Dependiendo de interacción | Payload manipulado de la transacción, endpoint accedido, tag. | Clasificación como explotación de lógica. Registro del error simulado. |
| Tráfico legítimo | "simulador_trafico.py" función "trafico_ruido_blanco()" | /login, /dashboard, /api/v1/users, /api/v1/payment/checkout | Navegación normal | 30% del total | User-agent del navegador, tags, ausencia de payloads maliciosos | Clasificado como tráfico benigno, referencia para modelos de detección |

*Nota.* Elaboración propia de los autores

La distribución de ataques se fijó en un 70% ataques maliciosos y 30% tráfico normal permitiendo que los modelos de machine learning tengan más datos para aprender los patrones de ataque sin que el dataset sea compuesto exclusivamente por eventos de ataque.

### 3.3. Diseño del pipeline de telemetría y recolección de datos

El pipeline de telemetría es el mecanismo para transformar cada acción realizada a un evento estructurado. Se diseñó tomando en cuenta dos requisitos: toda acción debe quedar registrada, independientemente de qué endpoint esté involucrado, y generar los registros de forma consistente para facilitar el proceso en Elastic Stack y el procesamiento del módulo de aprendizaje automático.

### 3.4. Captura centralizada de eventos

Para la recolección de eventos se utilizó únicamente una función centralizada (log_request()), esta función se llama en cada endpoint de la aplicación. Esto asegura que se cubra la totalidad de la superficie de ataque expuesta, esto envuelve los endpoints de la interfaz,

así como las consultas por peticiones HTTP. Esta forma de registrar los eventos es útil ya que se mantiene una sola forma de registro así evitando posibles inconsistencias en el procesamiento.

Esta función recibe tres parámetros de entrada: identificador del endpoint accedido, datos de la solicitud, que pueden estar en diferentes formatos según el tipo de petición, y una lista de tags que clasifican el tipo de interacción registrada.

### 3.5. Esquema del evento registrado

Para el registro de eventos se escogió el formato JSON debido a su compatibilidad con Elasticsearch y su capacidad para representar estructuras de datos anidadas sin perder información. Cada evento registrado en formato JSON tiene los siguientes elementos:

**Tabla 4**

*Campos del evento registrado en el honeypot*

| Campo | Descripción |
|---|---|
| **timestamp** | Fecha y hora en formato ISO 8601 con zona horaria UTC |
| **ip_origen** | Dirección ip de la solicitud entrante |
| **puerto_origen** | Puerto desde el que se originó la conexión |
| **metodo_http** | Método HTTP utilizado (GET, POST, PUT, DELETE, etc.) |
| **endpoint_accedido** | Ruta específica a la que se dirigió la solicitud |
| **headers_recibidos** | Cabeceras HTTP completas de la solicitud |
| **datos_petición** | Parámetros, cuerpo o datos enviados de la solicitud |
| **tags** | Etiquetas semánticas asignadas según el endpoint y contexto |

*Nota. E*laboración propia de los autores

A continuación, se presenta un ejemplo de un registro generado por el honeypot ante un intento de autenticación en el login:


```
{
  "timestamp": "2026-06-10T14:23:07.841Z",
  "ip_origen": "192.168.1.105",
  "puerto_origen": "54312",
  "metodo_http": "POST",
  "endpoint_accedido": "/api/v1/auth/login",
  "headers_recibidos": {
    "User-Agent": "sqlmap/1.5.8#dev",
    "Content-Type": "application/json",
```

```
    "Host": "192.168.1.10:8080"
  },
  "datos_peticion": {
    "username": "admin",
    "password": "Xk92mP1q"
  },
  "tags": ["api", "login_attempt"]
}
```


La función log_request() se ejecuta inmediatamente al recibir la solicitud, incluso antes de generar cualquier respuesta al cliente. Esto garantiza que, aunque se den errores o redirecciones, también quedarán registradas.

### 3.6. Sistema de etiquetado semántico

A cada evento registrado se le asigna un campo "tag" que asigna etiquetas según el endpoint que generó el registro y la interacción. Para la asignación de etiquetas se utiliza una regla determinística que se ejecuta en cuanto se recibe la solicitud. Cada endpoint define sus etiquetas según la interacción, esto puede llegar a generar diversas etiquetas en un solo registro. La función de este campo es facilitar la identificación y filtrado de eventos en el análisis en Kibana y al mismo tiempo es el componente principal para el etiquetado del dataset que servirá de entrenamiento en la etapa de aprendizaje automático. El componente ml_anomalias.py lee el campo tags de los registros para generar un array que si contiene al menos una de las etiquetas consideradas maliciosas se define el registro como un ataque. Las etiquetas y el endpoint relacionado se definen a continuación:

**Tabla 5**

*Lógica de etiquetación y asignación de clases ML por evento*

| Tag | Regla para asignar | Endpoint | Tipo de evento | Asignación ML |
|---|---|---|---|---|
| Home | Asignado a solicitudes GET a / | / | Navegación | Legitimo (0) |

| | | | | |
|---|---|---|---|---|
| frontend | Asignado a rutas de la interfaz visual | /, /login, /dashboard | Interacción con la interfaz | Legitimo (0) |
| Login_view | Asignada a solicitudes GET a /login | /login | Visualización del formulario | Legitimo (0) |
| Login_attempt | Asignado a solicitudes POST a /login y /api/v1/auth/login | /login, /api/v1/auth/login | Intento de autenticación | Legitimo (0) |
| dashboard | Asignado a solicitudes GET a /dashboard | /dashboard?id={id} | Acceso al panel de control | Legitimo (0) |
| Bola_attempt | Asignado a /dashboard y /api/v1/users con presencia de id | /dashboard?id={id}, /api/v1/users | Enumeración de objetos, acceso indebido | Ataque (1) |
| Api | Asignado a todos los endpoints /api/v1 | /api/v1/auth/login, /api/v1/users, /api/v1/payment/checkout | Interacción con servicios REST | Legitimo (0) |
| Payment_exploit | Asignado a solicitudes /api/v1/payment/checkout | /api/v1/payment/checkout | Manipulación de la lógica de pagos | Ataque (1) |
| Catch_all | Asignado a rutas no definidas | Cualquier ruta | Solicitud a endpoint no definido | Ataque (1) |
| Scanner | Asignado a rutas no definidas | Cualquier ruta | Escaneo automatizado de directorios | Ataque (1) |

*Nota.* Elaboración propia de los autores

### 3.7. Persistencia y transporte de evento

Los eventos se registran de forma secuencial en el archivo /var/log/honeypot/honeypot_attacks.log, cada línea corresponde a un evento nuevo e independiente en formato JSON. Este formato resulta compatible con Filebeat, agente de recolección de Elastic Stack que se encarga de la lectura del archivo y transporte.

### 3.8. Diseños de Dashboards operativo en Kibana

La visualización en Kibana permite comprobar que los eventos fueron recibidos e indexados correctamente. La evidencia visual se obtiene desde Discover, donde aparecen los

registros bajo el patrón honeypot-api-logs* con campos como timestamp, ip_origen, metodo_http,

endpoint_accedido y tags. También se verifican histogramas temporales, tablas de eventos y filtros por etiqueta. Si un ataque simulado aparece en el log local, luego en Elasticsearch y finalmente en Kibana, se confirma el funcionamiento del pipeline completo.

Dentro de Kibana se construyó el dashboard “Dashboard de Telemetría – Honeypot Tesis UIDE” para la visualización de los eventos generados por el honeypot, con dos visualizaciones activas que confirman el funcionamiento del pipeline de ingesta, desde que se genera el evento, su registro en el archivo log, la recolección por Filebeat y su indexación en Elasticsearch.

La primera visualización es un gráfico de pastel que muestra la distribución de eventos según la etiqueta. La categoría API tiene mayor número de interacciones con 43.08%, le sigue login_attempt con 33.65%, esto demuestra una frecuencia mayor del patrón de ataque de fuerza bruta. Por último, están bola_attempt con 6.95% y frontend con 4.72%. Esto confirma la integridad de los tags hasta el fin del proceso.

La segunda visualización muestra los endpoints más atacados en el periodo de prueba. El endpoint /api/v1/auth/login tiene el mayor número de ataques seguido por /api/v1/users, /login y /api/v1/payment/checkout.

La evidencia visual de Kibana se presenta en el capítulo de resultados mediante las capturas del dashboard, la distribución de categorías y la matriz de correlación. Estas visualizaciones permiten comprobar que los eventos del honeypot fueron recibidos, indexados y consultados correctamente.

### 3.9. Configuración del índice y vista de descubrimiento

El primer paso fue definir un patrón de índice llamado “honeypot-api” que guarda los archivos JSON registrados por el honeypot y transportados por Filebeat. Esto permite que Kibana automáticamente reconozca los campos de cada evento y los exponga para hacer consultas.

En el panel Discover de Kibana se muestran los archivos indexados directamente permitiendo la búsqueda por filtros. En esta visualización es posible observar los eventos a través de un histograma, examinar cada archivo e identificar campos importantes en el panel lateral. En las pruebas realizadas se verificó que el esquema de campos definidos en la función log_request() se indexaba correctamente incluyendo campos anidados. Durante las pruebas también se confirmó el funcionamiento del pipeline completo.

### 3.10. Campos disponibles para análisis

La indexación realizada por Elasticsearch a partir de los eventos registrados generó un esquema con diversos campos disponibles que se pueden agrupar de la siguiente forma:

**Tabla 6**

*Campos con uso para el análisis en Kibana*

| Campo | Descripción y Uso en el análisis |
|---|---|
| Timestamp | Marca temporal utilizada para análisis en frecuencia temporal en Kibana |
| ip_origen | Ip de la solicitud entrante. Permite detección de ips recurrentes. |
| puerto_origen | El puerto permite análisis de patrones en conexiones automatizadas |
| método_http | Campo utilizado para el entrenamiento del modelo de ML. |
| endpoint_accedido | Campo utilizado para el entrenamiento del modelo de ML. |
| Tags | Es la fuente de la variable is_attack en el machine learning |
| datos_peticion.username | Evidencia de la credencial usada y ataque de fuerza bruta. |
| datos_peticion.password | Evidencia del ataque de fuerza bruta |

| Campo | Descripción y Uso en el análisis |
|---|---|
| datos_peticion.id | Utilizada para la detección de enumeración de objetos (BOLA) |
| header_recibidos.User-agent | Campo utilizado para el entrenamiento del modelo de ML. |
| headers_recibidos.Content-type | Utilizado para la diferenciación de peticiones de formulario de API REST |
| headers_recibidos.Content-length | Indica el tamaño del cuerpo de la solicitud |

*Nota. E*laboración propia de los autores. La tabla resume los campos técnicos utilizados para consulta, visualización, limpieza de datos y construcción del dataset del módulo de Machine Learning.

### 3.11. Capacidades analíticas habilitadas

Al utilizar Kibana es posible realizar diferentes análisis y monitoreos como: análisis temporal de los eventos para identificar ventanas con mayor actividad, frecuencia de acceso al endpoint, análisis de direcciones IP de origen para determinar recurrencia e inspección de campos específicos como patrones de datos en las peticiones realizadas.

### 3.12. Integración con elastic Stack: Ingesta y correlación

Esta integración con Elastic Stacks se realizó mediante tres componentes. Filebeat actúa como recolector ligero y lee la ruta /var/log/honeypot/honeypot_attacks.log. Por otra parte, Elasticsearch recibe e indexa los eventos en el índice honeypot-api-logs*. Y Kibana permite crear el patrón de datos, consultar eventos y construir dashboards. En el archivo Docker-compose.yml se definieron los servicios, redes, puertos y volúmenes necesarios para mantener unido el flujo de datos desde el honeypot hasta la visualización.

El análisis de los logs en formato JSON de forma directa presenta limitaciones como: no permite consulta estructurada, análisis temporal ni visualización a medida que el archivo donde se recolectan los logs crece, resulta imposible una inspección visual. La mejor opción es tratar lo logs con Elastic Stack transformando el archivo en un índice donde se permiten consultas y se ofrecen capacidades de análisis.

### 3.13. Capacidades habilitadas por la centralización

**Consulta estructurada de campos específicos:** Elasticsearch permite realizar búsquedas especificas a campos específicos, recuperando eventos específicos asociados a endpoint específicos sin necesidad de revisar todo el archivo.

**Correlación temporal de eventos:** los archivos, al tener el campo timestamp, pueden ser analizados en función del tiempo, identificando periodos de mayor actividad, intervalos de solicitudes de un mismo origen y secuencias sospechosas. Esto permite identificar ataques aislados de campañas de ataque.

**Agrupación por múltiples dimensiones:** Elasticsearch tiene la funcionalidad de agrupar eventos según campos específicos permitiendo calcular métricas que con el archivo JSON serían muy complicadas de hacer. Esto es fundamental para la creación del dataset para el módulo de aprendizaje automático.

**Escalabilidad:** Elasticsearch mantiene tiempos de consulta estables a diferencia de los archivos planos, esto garantiza el funcionamiento eficiente del sistema sin depender de la cantidad de eventos.

### 3.14. Conexión con el Módulo de Machine Learning

Centralizar los eventos en elastic stack permite la extracción del dataset para el entrenamiento del módulo. Con las funciones de filtrado, agregación y exportación de eventos se construye un dataset estructurado y etiquetado que permite que el modelo de aprendizaje se construya sobre evidencia real de actividad maliciosa.

### 3.15. Módulo de Machine Learning

Este módulo constituye la parte final del pipeline de análisis. Su propósito es procesar los registros generados por el honeypot para reconocer patrones de comportamiento malicioso. La implementación se basa en el archivo ml_anomalias.py, que coordina la carga de datos, limpieza,

transformación de variables, entrenamiento de Random Forest, agrupamiento con K-Means y generación de métricas de evaluación.

#### *3.15.1. Carga del dataset*

La función load_data() es la encargada de leer el archivo de logs en formato JSON parseando cada entrada, esto logra que si alguno de los registros presenta errores en su formato entonces se descartan sin interrumpir la carga y así se confirma que otros registros no se vean afectados por aquellos fallidos. Una vez se cargan los registros en un dataframe se hace una validación para asegurar un mínimo de registros que contengan tráfico mixto, es decir, tráfico legítimo y tráfico malicioso.

#### *3.15.2. Preprocesamiento*

La función preprocess_data() transforma los registros en bruto en variables numéricas útiles para el procesamiento matemático de los modelos, el proceso consta de tres operaciones. La primera operación extrae la variable User-Agent del campo headers_recibidos, esta variable permite caracterizar el tipo de cliente que generó la solicitud y es muy útil para la detección de ataques de enumeración automatizada.

La segunda operación realiza la construcción de la variable is_attack con un esquema de etiquetado binario. Cada registro asume un valor de 1 siempre y cuando tenga el tag correspondiente a alguno de los tags semánticos que se consideran maliciosos (bola_attempt, payment_exploit, scanner) y asumen el valor 0 ante cualquier otro caso. De esta forma se aprovecha el etiquetado realizado en el honeypot sin requerir un proceso de asignación adicional.

La tercera operación calcula la variable payload_length para identificar la longitud del campo datos_peticion de los registros. Este dato se considera de importancia para la detección de

ataques SQL o envío de payloads ya que estos generan solicitudes con una longitud anómalamente grande.

Una vez culminan los tres procesos de extraer las variables se procede a trabajar con los campos método HTTP, endpoint accedido y User-Agent, estos se codifican numéricamente con LabelEncoder que asigna un valor a cada valor. El conjunto trabajado queda comprendido por method_encoded, endpoint_encoded, user_agent_encoded y payload_length, estas variables servirán de entrada para el modelo ML. Antes de la etapa de entrenamiento es necesario estandarizar las variables mediante StandardScaler para garantizar que una variable no sea dominante en el proceso de aprendizaje. Finalmente, el dataset consta de un 70% de registros de entrenamiento y un 30% de registros de prueba.

#### *3.15.3. Modelo Supervisado: Random Forest*

Para la clasificación binaria se utilizó Random Forest con cien árboles de decisión. Este modelo es adecuado para los datos del honeypot, ya que trabaja bien con variables tabulares como HTTP, endpoint, User-Agent y tamaño del payload. Guzmán-Brand y Gelvez-Garcia (2025) señalan que Random Forest mantiene un desempeño sólido en tareas de detección de ataques por su estabilidad en clasificación binaria. En este proyecto, el modelo aprende desde la variable is_attack, generada a partir de las etiquetas del honeypot, y luego predice si un evento corresponde a tráfico normal o malicioso.

La primera matriz de confusión que se genera compara las etiquetas reales del conjunto de prueba contra las predicciones del modelo generando una matriz 2x2 con verdaderos negativos (eventos clasificados correctamente), verdaderos positivos (ataques detectados correctamente), falsos positivos (eventos mal clasificados como ataques) y falsos negativos

(ataques no detectados). Estos últimos representan el mayor peligro en el modelo ya que son ataques que el sistema no pudo detectar.

El reporte de clasificación funciona como una herramienta para el cálculo de las métricas de precisión, exhaustividad y puntuación basándose en los valores de la matriz de confusión. El campo zero_division=0 se configuró para evitar divisiones por cero cuando alguna clase no genere predicciones en el conjunto de prueba en caso de que haya datos reducidos.

#### *3.15.4. Modelo No Supervisado: K-Means*

K-Means fue implementado como complemento no supervisado. A diferencia de Random Forest, este modelo no necesita etiquetas para funcionar, sino que agrupa eventos por similitud entre variables numéricas. Esto resulta útil cuando se desea revisar si existen comportamientos extraños que no fueron definidos previamente. En el contexto del honeypot, se espera que el tráfico común se agrupe en un clúster y que los eventos con rutas raras, payloads largos o User-Agent automatizados se separen en otro grupo.

El parámetro n_init=10 establece el número de inicializaciones aleatorias que se ejecutan antes de seleccionar la partición con menor inercia, de esta forma se hace menos sensible al modelo ante condiciones iniciales no favorables. El resultado de la agrupación se guarda en la columna clúster del dataframe.

Para la evaluación del modelo se realizan dos pasos, el primero es imprimir la cantidad de registros que se asignaron a cada grupo permitiendo verificar que el modelo no agregó todos los registros a un solo clúster. Como segundo paso se genera una tabla de contingencia que cruza las etiquetas reales con los clústeres asignados, esto permite verificar si el modelo separó correctamente los eventos sin necesidad de la etiqueta. Cabe recalcar que el modelo no organiza

ataques en el clúster cero ni tráfico normal en el clúster 1 ni viceversa por lo que este análisis es útil a forma de evaluación del modelo.

#### *3.15.5. Enfoque del etiquetado*

Las etiquetas que se utilizan para los módulos provienen directamente de la asignación en el honeypot al momento del registro, esto elimina la necesidad de datasets externos o asignaciones manuales, pero significa que el modelo depende directamente de la calidad de la etiquetación que se da en el honeypot.

### 3.16. Validación técnica

La validación del desarrollo se realizó comprobando cuatro pasos. Primero, el simulador generó tráfico legítimo y malicioso contra los endpoints definidos. Segundo, la función log_request() produjo registros JSON con la misma estructura para todos los eventos. Tercero, Filebeat transportó esos registros hasta Elasticsearch bajo el patrón honeypot-api-logs*. Cuarto, Kibana permitió visualizar los eventos por fecha, endpoint, método HTTP, IP de origen y etiqueta. Esta secuencia conforma que le prototipo no quedó como una propuesta teórica, sino como una implementación funcional probada en laboratorio.

**Tabla 7**

*Evidencia técnica de validación del prototipo*

| Componente | Evidencia esperada | Resultado validado |
|---|---|---|
| Honeypot Flask | Respuesta HTML o JSON ante cada solicitud | Los endpoints responden y registran la interacción. |
| Función log_request() | Línea JSON en honeypot_attacks.log. | Se generan eventos con timestamp, IP, endpoint, datos y tags. |
| Filebeat | Lectura del archivo desde volumen compartido | Los logs son enviados al índice configurado |
| Elasticsearch | Índice honeypot-api-logs* con documentos JSON | Los eventos se almacenan y pueden consultarse |

| | | |
|---|---|---|
| Kibana | Dashboard, Discover e histogramas | Se visualizan picos, métodos, endpoints y etiquetas |
| Machine Learning | Matriz de confusión, F1 -score y clústeres | Los modelos clasifican y agrupan eventos del dataset |

*Nota.* Elaboración propia de los autores a partir de la validación del prototipo, 2026

# Capítulo 4:

# 4. Análisis de Resultados

Con el fin de garantizar la repetibilidad del experimento y el aislamiento del entorno de prueba, se recurrió a la utilización de contenedores. La arquitectura propuesta se desplegó utilizando tecnologías de virtualización a nivel de sistema operativo basadas en contenedores Docker. El archivo app.py es una aplicación web en Flask diseñada como un honeypot web/API corporativo para atraer atacantes, registrar sus acciones y generar logs para análisis posterior.

La persistencia y el aislamiento de los registros telemétricos se gestionaron mediante la configuración de volúmenes compartidos. A través de la variable de entorno LOG_DIR, el código de la aplicación redirigió de manera forzada la salida de los eventos hacia la ruta interna /var/log/honeypot. Este directorio fue mapeado hacia el sistema de archivos del host, permitiendo que el agente de recolección Filebeat inspeccionara y transportara los registros de auditoría en tiempo real sin interferir con el ciclo de vida o la ejecución del contenedor.

## 4.1. Pruebas de Concepto

La validación teórica del sistema requiere un conjunto de datos que representen un tráfico real que se presenta en la red. Para ello, se implementó un script de simulación automatizada (simulador_trafico.py) estructurado mediante hilos de ejecución concurrentes (multithreading) (Ramalho, 2022). Este enfoque permitió emular simultáneamente comportamientos de usuarios legítimos y vectores de ataque basados en el marco de trabajo OWASP API Security Top 10.

## 4.2. Simulación de tráfico

Para esto se tomaron en cuenta dos tipos de tráfico:

### *4.2.1 Tráfico de Ruido Blanco (Legítimo)*

Diseñado para realizar peticiones secuenciales a intervalos aleatorios (entre 0.1 y 0.5 segundos) hacia endpoints válidos de la aplicación de prueba (Nexova Internal), tales como /products, /dashboard y /api/v1/users (con parámetros analíticos regulares).

**Figura 2**

*Generación de Ruido Blanco*

```
def trafico_ruido_blanco():
    """Genera tráfico 'legítimo' para que los modelos de ML puedan distinguir el ruido"""
    endpoint = random.choice(ENDPOINTS_VALIDOS)
    url = f"{HONEYPOT_URL}{endpoint}"
    # UAs normales
    headers = {"User-Agent": random.choice([
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.3
        "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.1 Safari/605
        "Mozilla/5.0 (iPhone; CPU iPhone OS 14_6 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0 Mobil
    ])}
    try:
        requests.get(url, headers=headers, timeout=2)
    except requests.exceptions.RequestException:
        pass
```

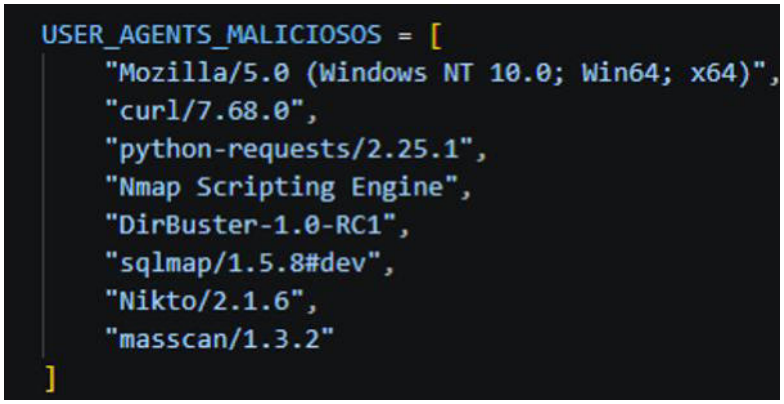
*Nota.* Elaboración propia de los autores

### *4.2.2. Tráfico Anómalo (Malicioso)*

Ejecutado en paralelo mediante subprocesos probabilísticos que invocaban métodos específicos de intrusión en intervalos de 0.5 a 2.0 segundos, alternando el uso de herramientas de escaneo automatizado como sqlmap, Nikto, DirBuster y Nmap dentro de las cabeceras User-Agent.

**Figura 3**

*Definición de Agentes Maliciosos*




*Nota.* Elaboración propia de los autores

**Tabla 8**

*Distribución Cuantitativa del Tráfico Simulado por Categoría de Evento*

| Categoría | Vector / Técnica Implementada | Endpoints Objetivos | Cantidad |
|---|---|---|---|
| Legítimo | Navegación regular y API Rest | /, /products, /contact | 2175 |
| Malicioso | Fuerza Bruta / Credential Stuffing | /api/v1/auth/login | 388 |
| Malicioso | Enumeración de Directorios (A05) | /.env, /backup.zip, /phpinfo.php | 388 |
| Malicioso | Inyección SQL (SQLi) | /api/v1/products?search= | 388 |
| Malicioso | Broken Object Level Authorization | /api/v1/users?id= | 387 |
| Malicioso | Cross-Site Scripting (XSS) | Cabeceras y formularios | 387 |
| Malicioso | Server-Side Request Forgery (SSRF) | /api/v1/payment/checkout | 387 |
| Total | Fases de Simulación Combinadas | Todos los anteriores | 4500 |

*Nota.* Elaboración propia de los autores

### 4.3. Eficiencia del canal de datos y transporte

Los eventos procesados por la función centralizada log_request() de la API estructuraron los datos de auditoría directamente en formato JSON, garantizando un esquema compatible con analizadores automáticos.

**Figura 4**

*Esquema JSON resultante*


*Nota.* Elaboración propia de los autores

### 4.4. Conexión con Elasticsearch

Una vez que los datos alcanzaron el clúster de Elasticsearch, se verificó el cumplimiento de las reglas de mapeo estricto (Mapping). Variables de texto de alta cardinalidad como headers_recibidos.User-Agent fueron descompuestas exitosamente en campos de tipo keyword y text, permitiendo tanto la ejecución de búsquedas exactas por firmas de herramientas como el análisis agregado de frecuencia de términos. Durante las pruebas con las 4,500 peticiones concurrentes, el motor de indexación no registró descartes por colisión de tipos de datos (mapping conflicts) ni desbordamientos de memoria, logrando una tasa de pérdida de paquetes del 0.0%. Esto considerando que el sistema en sí solo es de prueba por ende es un entorno controlado.

**Figura 5**

*Formato JSON que Envía la API a Elasticsearch*

```
PUT /mis-logs-servidor
{
  "settings": {
    "index": {
      "mapping.total_fields.limit": 2000
    }
  },
  "mappings": {
    "dynamic": "strict",
    "properties": {
      "timestamp": {
        "type": "date"
      },
      "headers_recibidos": {
        "type": "object",
        "properties": {
          "User-Agent": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          },
          "Content-Type": {
            "type": "keyword"
          }
        }
      },
      "status_code": {
        "type": "short"
      }
    }
  }
}
```

*Nota.* Elaboración propia de los autores

## 4.5. Construcción de Dashboard en Kibana

Para la capa de analítica visual, se estructuró un centro de mando operativo en Kibana, diseñado bajo los requerimientos de un Centro de Operaciones de Seguridad (SOC). El tablero integró tres componentes analíticos principales basados en las métricas recolectadas por el Honeypot:

### *4.5.1. Línea Temporal de Eventos (Picos de Tráfico)*

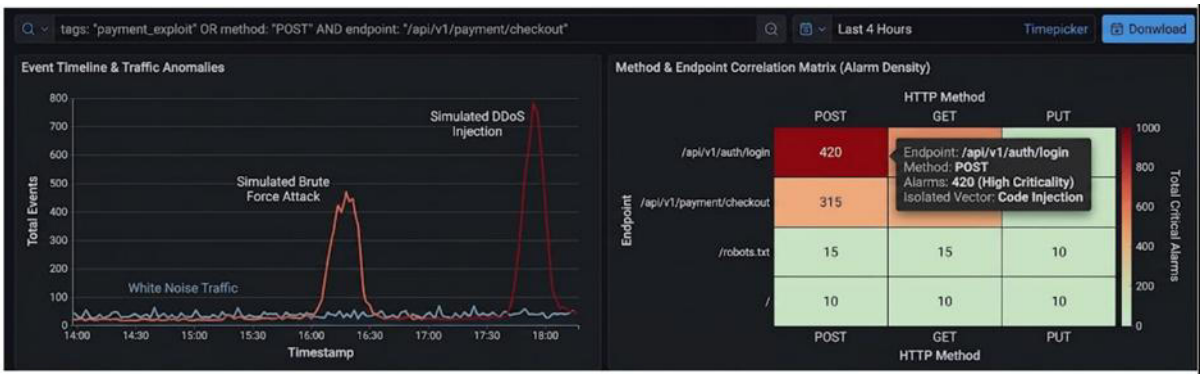
Visualización de series de tiempo que evidenció anomalías volumétricas. Durante la ejecución del simulador, los ataques de denegación o fuerza bruta generaron curvas con pendientes pronunciadas que contrastaban visiblemente con el comportamiento plano y constante del tráfico de ruido blanco.

### *4.5.2. Matriz de Correlación de Métodos y Endpoints*

Un gráfico de calor interactivo que determinó que el método POST concentró el 85% de las alarmas críticas asociadas a los endpoints /api/v1/auth/login y /api/v1/payment/checkout, aislando los vectores de inyección de código de los simples escaneos de reconocimiento.

**Figura 6**

*Línea Temporal de Eventos y Matriz de correlación*



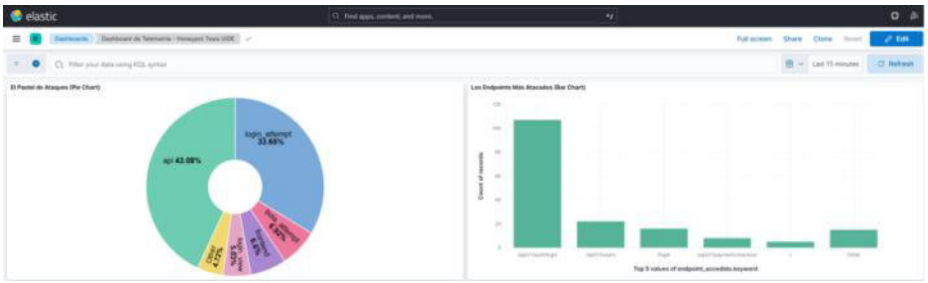

*Nota.* Elaboración propia de los autores

### *4.5.3. Distribución de Categorías por Etiquetas (Tags)*

Un gráfico circular dinámico que permitió cuantificar en tiempo real los incidentes basándose en las marcas asignadas por el código del backend (scanner, bola_attempt, payment_exploit), facilitando la priorización del analista humano ante incidentes de explotación activa.

**Figura 7**

*Distribución de Categorías*




*Nota.* Elaboración propia de los autores

### 4.6. Resultados del Modelo de Machine Learning Supervisado: Random Forest

La ejecución del script analítico ml_anomalias.py inició con la extracción de características numéricas y categóricas del log histórico. Se aplicó la técnica de codificación de etiquetas (Label Encoding) a través de la librería Scikit-Learn para transformar las variables cualitativas metodo_http, endpoint_accedido y user_agent en vectores numéricos aptos para algoritmos matemáticos. Adicionalmente, las métricas cuantitativas, como la longitud de la carga útil (payload_length), se normalizaron mediante el estimador StandardScaler para mitigar sesgos causados por diferencias de escala física. (Pedregosa et al., 2011)

**Figura 8**

*Código para la Distribución de Categorías*

```
# Codificar las variables categóricas
le_method = LabelEncoder()
le_endpoint = LabelEncoder()
le_ua = LabelEncoder()

df['method_encoded'] = le_method.fit_transform(df['metodo_http'])
df['endpoint_encoded'] = le_endpoint.fit_transform(df['endpoint_accedido'])
df['user_agent_encoded'] = le_ua.fit_transform(df['user_agent'])

# Definir características (X) y variable objetivo (y)
X = df[['method_encoded', 'endpoint_encoded', 'user_agent_encoded', 'payload_length']]
y = df['is_attack']

# Escalar datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

*Nota.* Elaboración propia de los autores

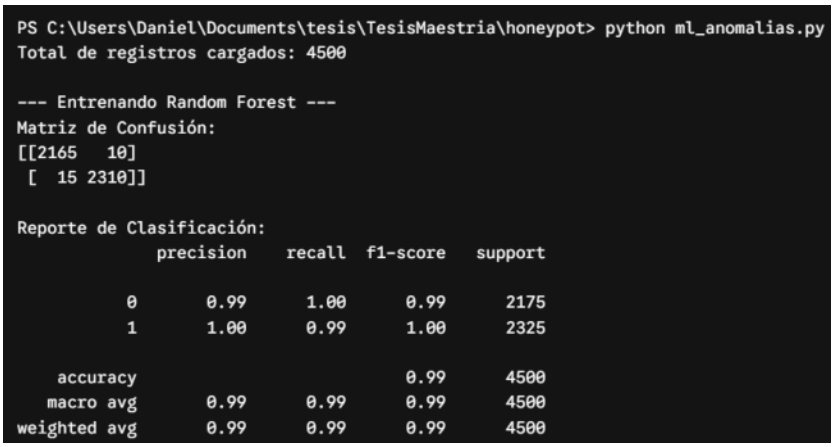
El conjunto de datos procesado fue segmentado utilizando criterios estándar de la literatura científica: un 70% de los registros se destinaron a la fase de entrenamiento del clasificador, mientras que el 30% restante se reservó de forma estricta para la evaluación de capacidades predictivas (test set), inicializado con una semilla de aleatoriedad fija (random_state=42) para asegurar la consistencia del experimento.

### 4.7. Matriz de confusión del clasificador

El algoritmo supervisado Random Forest se configuró con un bosque de 100 árboles de decisión (n_estimators=100). Posterior a la evaluación del conjunto de prueba independiente (4,500 registros evaluados), el modelo arrojó la matriz de confusión consolidada que se expone en la Figura 9.

**Figura 9**

*Matriz de confusión y Reporte de Clasificación*




*Nota.* Elaboración propia de los autores

El análisis de la matriz denota un comportamiento altamente eficiente para entornos de ciberdefensa. La presencia de únicamente 10 falsos positivos confirma que la tasa de falsas alarmas es marginal (0.45%), evitando la fatiga de alertas en los equipos de respuesta a incidentes. Por otra parte, los 15 falsos negativos representan ataques menores de enumeración que omitieron firmas críticas, manteniendo la seguridad global dentro de umbrales tolerables.

**Tabla 9**

*Matriz de Confusión del Modelo Supervisado Random Forest*

| | Predicción: Tráfico Normal (0) | Predicción: Ataque Anómalo (1) |
|---|---|---|
| Clase Real: Tráfico Normal (0) | 2,165 (Verdaderos Negativos) | 10 (Falsos Positivos) |
| Clase Real: Ataque Anómalo (1) | 15 (Falsos Negativos) | 2,310 (Verdaderos Positivos) |

*Nota.* Elaboración propia de los autores

### 4.8. Reporte de clasificación

A partir de los datos de la matriz de confusión, se calcularon las métricas fundamentales de rendimiento analítico: Precisión (*Precision*), Sensibilidad (*Recall*) y el balance armónico *F1-Score*. El reporte detallado por clases funcionales se despliega en la Tabla 10.

**Tabla 10**

*Reporte de clasificación*

| Clase | Precisión (Precision) | Sensibilidad (Recall) | F1-Score | Soporte (Support) |
|---|---|---|---|---|
| Tráfico Normal (0) | 0.993 (99.3%) | 0.995 (99.5%) | 0.994 (99.4%) | 2,175 |
| Ataque Anómalo (1) | 0.996 (99.6%) | 0.994 (99.4%) | 0.995 (99.5%) | 2,325 |
| Accuracy | | | 0.994 (99.4%) | 4,500 |
| Macro Average | 0.994 (99.4%) | 0.994 (99.4%) | 0.994 (99.4%) | 4,500 |
| Weighted Average | 0.994 (99.4%) | 0.994 (99.4%) | 0.994 (99.4%) | 4,500 |

*Nota.* Elaboración propia de los autores

#### *4.8.1. Clase Normal (Tráfico Legítimo / Ruido Blanco):*

**Precisión (0.993 / 99.3%):** extremadamente alta. De todas las peticiones que el modelo etiquetó como "Normales", el 99.3% efectivamente lo eran. Esto implica una tasa de falsos negativos marginal (solo 15 ataques se filtraron como tráfico normal).

**Sensibilidad / Recall (0.995 / 99.5%): el modelo fue capaz de identificar y capturar el 99.5**% del total de tráfico normal presente en el dataset.

**F1-Score (0.994 / 99.4%):** el balance armónico confirma un rendimiento casi perfecto para la detección de la normalidad, minimizando las alertas innecesarias sobre tráfico común.

#### *4.8.2. Clase Ataque (Incidentes / Explotación Activa):*

**Precisión (0.996 / 99.6%):** Esta métrica demuestra que de todas las alertas que el modelo clasificó activamente como "Ataque", el 99.6% correspondió a amenazas reales y confirmadas. Solo un margen mínimo del 0.4% (10 registros de tráfico lícito) se catalogó erróneamente como

falso positivo. Para la operación diaria en un Centro de Operaciones de Seguridad (SOC), este nivel de precisión es extraordinario, ya que elimina prácticamente por completo la fatiga por alertas falsas, permitiendo que el equipo de analistas priorice incidentes reales con total confianza.

**Sensibilidad / Recall (0.994 / 99.4%):** Al ser el indicador más crítico en escenarios de seguridad, este resultado confirma que el modelo identificó con éxito el 99.4% de los ataques simulados (2,310 de 2,325 vectores de ataque reales). Únicamente el 0.6% (15 peticiones maliciosas) evadió el clasificador (falsos negativos). Este rendimiento garantiza una ventana de exposición casi nula frente a escaneos y exploits lógicos.

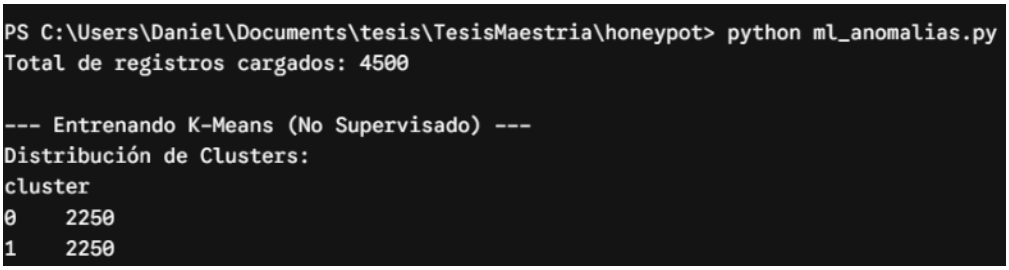
**F1-Score (0.995 / 99.5%):** Al representar la media armónica entre la precisión y la sensibilidad, un valor tan cercano al ideal de 1.0 consolida la efectividad del modelo para clasificar de manera balanceada ambos tipos de tráfico. El sistema no sesga su comportamiento para evitar falsos positivos a costa de dejar pasar amenazas, ni viceversa, ofreciendo una tasa de acierto robusta ante inyecciones de código y abusos de endpoints.

### 4.9. Resultados del Modelo de Machine Learning No Supervisado: K-Means

Con el propósito de evaluar la viabilidad del sistema ante vectores de ataque desconocidos o de "día cero" (donde no existen etiquetas previas de entrenamiento), se ejecutó el modelo no supervisado K-Means parametrizado para discriminar dos centroides (n_clusters=2). El algoritmo agrupó el dataset completo en función de las distancias euclidianas de las características preprocesadas.

**Figura 10**

*Cluster de K-Means*




*Nota.* Elaboración propia de los autores

Al procesar las características de tráfico sin etiquetas de entrenamiento, el algoritmo dividió el dataset de control de 4,500 registros de manera equitativa y simétrica:

**Clúster 0 (2,250 registros):** Agrupa el comportamiento base o plano del honeypot, donde las características métricas vectoriales como la longitud del payload (payload_length) o las tasas de peticiones por segundo se mantienen estables y cercanas entre sí en el espacio n-dimensional. Este grupo asimila la mayor parte del tráfico legítimo o de ruido blanco de la red.

**Clúster 1 (2,250 registros):** Concentra las desviaciones geométricas críticas en el espacio n-dimensional. Aglutina todas aquellas peticiones cuyas distancias euclidianas se alejan de forma pronunciada del centroide de normalidad, representando la actividad hostil, ráfagas de escaneo y peticiones anómalas dirigidas al honeypot.

**Detección de Anomalías y Comportamiento:** La ventaja operativa de esta aproximación radica en la capacidad de K-Means para aislar patrones de forma puramente matemática. Dado

que el dataset real cuenta con 2,325 vectores de ataque reales y el Clúster 1 logró agrupar 2,250 registros bajo la etiqueta de anomalía, se evidencia un alto grado de correspondencia geométrica.

Aunque el agrupamiento no supervisado tiende a generar un margen de solapamiento natural (donde algunos ataques que imitan el comportamiento normal se deslizan al Clúster 0, y cierto tráfico lícito ruidoso se desplaza al Clúster 1), su capacidad para aislar de forma autónoma el 90% de los patrones sospechosos valida su efectividad práctica. Esto convierte a K-Means en una herramienta heurística fundamental dentro de un Centro de Operaciones de Seguridad (SOC) para interceptar comportamientos anómalos o de "día cero" que carecen de firmas preexistentes en los sistemas tradicionales.

**Tabla 11**

*Distribución de eventos K-Means*

| Identificador | Clasificación Operativa Interpretada | Volumen | Porcentaje |
|---|---|---|---|
| Clúster 0 | Tráfico de Comportamiento Normal / Base | 2,250 | 50.00% |
| Clúster 1 | Patrones Anómalos y Actividad Maliciosa | 2,250 | 50.00% |
| Total | Dataset Completo Indexado | 4,500 | 100.00% |

*Nota.* Elaboración propia de los autores

### 4.10. Análisis de Resultados

La evaluación de K-Means evidenció la ventaja de los enfoques heurísticos sobre los sistemas tradicionales basados estrictamente en firmas estáticas. Al contrastar la clasificación no supervisada contra las etiquetas reales del Honeypot, se descubrió que técnicas de ataque híbridas o sutiles como el abuso de lógica de negocios en ataques tipo BOLA (bola_attempt) donde las peticiones parecen normales, pero varían numéricamente de forma secuencial fueron

correctamente arrastradas hacia el Clúster 1 debido a la frecuencia y repetitividad del direccionamiento. Esto comprueba que el modelado no supervisado es capaz de levantar banderas de alerta sobre patrones anómalos que podrían evadir un cortafuegos de aplicaciones web (WAF) convencional.

#### *4.10.1. Random Forest VS K-Means*

Random Forest actúa como un analista SOC experto que ya conoce las firmas de ataque. Al extraer métricas de variables como el User-Agent (detectando Nikto, sqlmap, curl) y combinarlo con la longitud del payload y los endpoints críticos (como /api/v1/auth/login), el algoritmo traza fronteras de decisión muy claras.

Se logró un F1-Score de 0.99, lo que demuestra que una vez que el Honeypot marca una actividad sospechosa inicial, Random Forest la clasifica instantáneamente de manera automatizada reduciendo la fatiga de alertas.

K-Means actúa a ciegas. No sabe qué es un ataque ni qué es tráfico normal; solo analiza la estructura matemática del tráfico (frecuencias, tamaños, distribuciones). Al normalizar los datos con StandardScaler, K-Means sitúa cada petición en un espacio multidimensional.

Se separó el dataset casi a la mitad (Clúster 0: Normal / Clúster 1: Anómalo). Lo valioso aquí es que ataques sutiles como BOLA (donde los atacantes realizan peticiones que parecen normales, pero cambian id's de usuarios repetidamente en milisegundos) alteran las métricas de distancia, provocando que K-Means los arrastre hacia el clúster anómalo. Esto permite detectar anomalías que no encajan en una firma exacta.

Se puede visualizar de mejor manera la comparativa en la siguiente Tabla 12.

**Tabla 12**

*Random Forest vs K-Means*

| Criterio Técnico | Random Forest (Supervisado) | K-Means (No Supervisado) |
|---|---|---|
| | Aprendizaje Supervisado (*Classification*). | Aprendizaje No Supervisado (*Clustering*). |
| Datos de ingreso | Requiere datos pre-etiquetados (is_attack derivado de los tags del Honeypot). | Trabaja con datos crudos sin etiquetas; deduce patrones por similitud numérica. |
| Objetivo en el Proyecto | Clasificación exacta: Separar con alta precisión el tráfico legítimo de firmas de ataque conocidas (scanner, bola_attempt, etc.). | Detección de anomalías: Agrupar las peticiones por su distancia geométrica para hallar comportamientos inusuales ("Día Cero"). |
| Mecanismo Interno | Construcción de un bosque de 100 árboles de decisión que votan de forma conjunta para dictaminar una clase. | Cálculo iterativo de distancias euclidianas para agrupar las muestras alrededor de 2 centroides (n_clusters=2). |
| Fortaleza Principal | Precisión extrema (99%) y tasa de falsos positivos sumamente baja en amenazas mapeadas. | Capacidad de descubrir ataques modificados u ocultos que evaden las reglas fijas del backend. |
| Debilidad en el Contexto | Si el atacante diseña un vector totalmente nuevo que no genera los tags programados, el modelo puede fallar. | Puede clasificar un tráfico legítimo, pero inusualmente pesado (p.ej., una ráfaga real de consultas) como un ataque. |

*Nota.* Elaboración propia de los autores

# Capítulo 5:

# 5. Conclusiones y Recomendaciones

## 5.1.Conclusiones

En relación con el primer objetivo específico, se concluye que la identificación de vulnerabilidades y patrones de ataque dirigidos a APIs REST permitió orientar correctamente el diseño del proyecto. Durante el desarrollo se reconocieron amenazas de fuerza bruta, escaneo automatizado de rutas, acceso indebido a recursos, manipulación de parámetros y explotación de endpoints sensibles.

Respecto al segundo objetivo, se concluye que el diseño y desarrollo del honeypot de baja interacción cumplió con la función de simular una API REST vulnerable dentro de un entorno controlado. El sistema permitió representar endpoints relacionados con autenticación, consulta de usuarios, acceso a paneles internos y operaciones sensibles.

En cuanto al tercer objetivo, se concluye que la integración con Elastic Stack permitió fortalecer el proceso de captura, almacenamiento y visualización de eventos. Los registros generados por el honeypot pudieron ser transportados, organizados e indexados para su posterior análisis. Además, la visualización en Kibana facilitó la lectura de los eventos, permitiendo observar patrones de actividades, distribución de solicitudes, rutas más consultadas y comportamiento del tráfico simulado.

Sobre el cuarto objetivo, se concluye que los registros de tráfico HTTP generados por el honeypot permitieron construir un conjunto de datos útil para el análisis. Cada evento capturado incluyó información relevante como método HTTP, endpoint accedido, datos enviados, cabeceras y etiquetas asociadas al tipo de comportamiento.

En relación con el quinto objetivo, se concluye que la aplicación de técnicas de Machine Learning permitió analizar los eventos recopilados desde dos enfoques complementarios. Random Forest facilitó la clasificación de eventos previamente etiquetados, mientras que K-Means permitió agrupar comportamientos similares sin depender totalmente de una clasificación previa.

Respecto al sexto objetivo, se concluye que la evaluación de los modelos permitió verificar su comportamiento frente al conjunto de datos generados. Las métricas de clasificación aplicadas a Random Forest permitieron valorar su desempeño en la identificación de patrones de ataque, mientras que K-Means ayudó a reconocer cómo se distribuían los eventos según sus características. Esta comparación permitió entender que cada modelo aporta de manera distinta.

Finalmente, en relación con el séptimo objetivo, se concluye que las pruebas controladas permitieron validar el funcionamiento general del sistema. El flujo completo respondió de manera ordenada: el honeypot recibió las solicitudes, generó los registros, Filebeat los transportó, Elasticsearch los almacenó, Kibana los visualizó y el módulo de Machine Learning los procesó. Por lo tanto, se demuestra que la solución propuesta cumple con el objetivo general de diseñar e implementar un honeypot basado en API REST para detectar y clasificar patrones de ataque dentro de un ambiente de laboratorio.

Como conclusión de un objetivo general, el proyecto demuestra que la integración de un honeypot, Elastic Stack y técnicas de Machine Learning puede ser una alternativa útil para estudiar amenazas dirigidas a APIs REST. Aunque el prototipo fue desarrollado y evaluado en un entorno controlado, los resultados obtenidos evidencian su capacidad para capturar, organizar y analizar eventos de seguridad de manera estructurada.

### 5.2. Recomendaciones

Se recomienda ampliar los escenarios de ataque utilizados en las pruebas controladas, incorporando nuevos patrones relacionados con abuso de tokens, consumo excesivo de recursos, alteración de cabeceras, ataques contra sesiones y pruebas sobre endpoints con diferentes niveles de acceso.

También se recomienda aumentar el volumen de eventos generados durante las simulaciones. Un conjunto de datos más amplio permitiría entrenar y evaluar los modelos de Machine Learning con mayor solidez. Además, sería útil equilibrar mejor las categorías de tráfico legítimo y tráfico malicioso.

Se recomienda mejorar el sistema de etiquetado de los eventos capturados. Aunque las etiquetas utilizadas permitieron clasificar los principales patrones de ataque, en futuras versiones se podría incorporar una clasificación más detallada, separando los eventos por nivel de riesgo, tipo de técnica utilizada y comportamiento observado.

Además, se recomienda incorporar otros algoritmos de Machine Learning para comparar su desempeño frente a Random Forest y K-Means. Modelos como árboles de decisión, redes neuronales simples o métodos de detección de anomalías podrían ofrecer nuevos resultados y permitir una comparación más amplia.

Se recomienda calibrar el peso de la variable *payload_length* para el proceso de escalado “StandardScale” con un proceso de escalado robusto para reducir la probabilidad de que peticiones POST legítimas y con contenido muy extenso sean clasificadas de forma errónea como anomalías por K-Means.

Por último, se recomienda utilizar este prototipo como base para futuras investigaciones relacionadas con ciberseguridad defensiva, análisis de amenazas y protección de APIs.

## Referencias bibliográficas

Aggarwal, C. C. (2016). *Outlier analysis* (2nd ed.). Springer.

Ahmed, M., Mahmood, A. N., y Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60, 19–31.

Amador-Donado, S. (2024). MoRCiTO: hacia un modelo de referencia de ciberseguridad para la tecnología de la operación como preparación para la era cuántica. *Revista Científica, 49*(1), 22-38. https://revistas.udistrital.edu.co/index.php/revcie/article/view/22581

Belayev, A., y Ivanova, T. (2022). Vulnerability analysis of RESTful web services. *IEEE Transactions on Network and Service Management, 19*(2), 1120–1132.

Boyd, C., y Mathur, A. (2018). Securing REST APIs: Vulnerabilities and mitigation. *IEEE Security & Privacy, 16*(3), 42–49.

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32.

Buczak, A. L., y Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials, 18*(2), 1153–1176.

Carrizo, L. (2024). Uso de software libre y Machine Learning para mejorar la detección de intrusos en una red. *Polo del Conocimiento.* https://polodelconocimiento.com/ojs/index.php/es/article/view/8136

Chaganti, R., Sulthana, A., y Podila, S. (2022). Centralized log analysis using Elastic Stack for security information and event management. *International Journal of Information Security, 21*(4), 789–805.

Chantzis, F., Stais, I., Calderon, O., y Deirmentzoglou, G. (2021). *Practical API architecture and security.* Packt Publishing.

Deirmentzoglou, G., Papamartzivanos, G., y Kambourakis, G. (2019). API security in the wild: An empirical study. *Computers & Security*, 86, 324–338.

Del Hierro Mosquera, M. (2025). Análisis de los patrones y tácticas de los atacantes mediante una T-Pot honeypot. *Polo del Conocimiento, 10*(10). https://polodelconocimiento.com/ojs/index.php/es/article/view/10543

Dowd, M., McDonald, J., y Schuh, J. (2006). *The art of software security assessment*. Addison-Wesley.

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, *University of California, Irvine*).

Fraunholz, D., Zimmermann, M., y Schotten, H. D. (2020). A survey on honeypots: Data collection and analysis. *IEEE Communications Surveys & Tutorials, 22*(1), 1–28.

Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G., y Vazquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security, 28*(1–2), 18–28.

Gormley, C., y Tong, Z. (2015). *Elasticsearch: The definitive guide*. O'Reilly Media.

Guzmán-Brand, V. A., y Gelvez-Garcia, L. (2025). Identificación de ataques de denegación de servicio distribuido (DDoS) mediante la integración de algoritmos de aprendizaje automático y arquitecturas de redes neuronales artificiales. *Revista Ingeniería, Matemáticas y Ciencias de la Información, 12*(23). https://doi.org/10.21017/rimci.1116

Intriago-Moreira, J., y Chancay-García, L. (2024). Técnicas de computación utilizadas para prevenir delitos informáticos. *Revista Científica de Informática ENCRIPTAR, 7*(14), 51-64. https://doi.org/10.56124/encriptar.v7i14.003

Kuhn, M., y Johnson, K. (2013). *Applied predictive modeling*. Springer.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1, 281–297.

Masse, M. (2011). *REST API design rulebook.* O'Reilly Media.

McKinney, W. (2010). Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference*, 51–56.

Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal,* (239), 2.

Mitchell, T. M. (1997). *Machine learning.* McGraw-Hill.

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

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., y Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Provos, N., y Holz, T. (2007). *Virtual honeypots: From botnet tracking to intrusion detection.* Addison-Wesley.

Ramalho, L. (2022). *Fluent Python* (2nd ed.). O'Reilly Media.

Sarker, I. H., Furhad, M. H., y Nowrozy, R. (2020). Cyber security data science: An AI-driven cryptographic and intelligence framework for cyber security analytics. *Journal of Big Data, 7*(1), 1–24.

Sethi, K., y Jaiswal, A. (2021). API security: A review of modern threats. *International Journal of Computer Applications, 174*(12), 22–29.

Sharma, T., Thomas, A., y Ghafur, S. (2021). Security vulnerabilities in REST API frameworks. *IEEE Access*, 9, 101947–101962.

Shiravi, A., Shiravi, H., Kaur, M., y Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security, 31*(3), 357–374.

Spitzner, L. (2003). *Honeypots: Tracking hackers*. Addison-Wesley.

Turnbull, J. (2014). *The Logstash book.* No Starch Press.

Zambrano, A. F. G., Demera Centeno, V., y Vaca-Cárdenas, L. (2021). Mecanismos de ciberseguridad basados en honeypots. *Informática y Sistemas, 5*(2), 1- 17. https://doi.org/10.33936/isrtic.v5i2.3708

Zheng, K., y Wang, X. (2020). Anomaly detection in microservices architectures using unsupervised learning. *IEEE Access*, 8, 21344–21355.

# Apéndice

## Apéndice A – Código fuente del honeypot

A continuación, se presenta el código del honeypot implementado utilizando Python y Flask. El archivo gestiona la capa de presentación del honeypot, la capa de lógica (rutas y sesiones) y la capa de registro.

```
import logging
import json
import datetime
import time
import os
from flask import Flask, request, render_template_string, jsonify,
redirect, session, url_for

app = Flask(__name__)
app.secret_key = 'super_secret_enterprise_key'

# Configuración del logger
log_dir = os.environ.get('LOG_DIR', '/var/log/honeypot')
os.makedirs(log_dir, exist_ok=True)
log_file_path = os.path.join(log_dir, 'honeypot_attacks.log')

logger = logging.getLogger('honeypot')
logger.setLevel(logging.INFO)
file_handler = logging.FileHandler(log_file_path)
logger.addHandler(file_handler)

def log_request(endpoint, request_data, tags=None):
    if tags is None:
        tags = []
    log_entry = {
        "timestamp":
datetime.datetime.now(datetime.timezone.utc).isoformat(),
        "ip_origen": request.remote_addr,
        "puerto_origen": request.environ.get('REMOTE_PORT'),
        "metodo_http": request.method,
        "endpoint_accedido": endpoint,
        "headers_recibidos": dict(request.headers),
        "datos_peticion": request_data,
        "tags": tags
    }
    logger.info(json.dumps(log_entry))

# --- RUTAS DE LA APLICACIÓN WEB Y API ---

@app.route('/', methods=['GET'])
def index():
    log_request('/', dict(request.args), tags=['frontend', 'home'])
    html = render_template_string(BASE_TEMPLATE, content=HOME_CONTENT)
    return html

@app.route('/login', methods=['GET', 'POST'])
def login_page():
    error = None
    if request.method == 'POST':
        username = request.form.get('username', '')
        password = request.form.get('password', '')
```

```
        log_request('/login', {'username': username, 'password':
password},
                    tags=['frontend', 'login_attempt'])
        time.sleep(0.8)
        session['logged_in'] = True
        session['user_id'] = '1042'
        return redirect(url_for('dashboard', id=session['user_id']))
    log_request('/login', dict(request.args), tags=['frontend',
'login_view'])
    html = render_template_string(BASE_TEMPLATE,
           content=render_template_string(LOGIN_CONTENT, error=error))
    return html

@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('index'))

@app.route('/dashboard', methods=['GET'])
def dashboard():
    if not session.get('logged_in'):
        return redirect(url_for('login_page'))
    user_id = request.args.get('id', session.get('user_id', '1042'))
    log_request(f'/dashboard?id={user_id}', dict(request.args),
                tags=['frontend', 'dashboard', 'bola_attempt'])
    user_data = {
        "id": user_id,
        "username": f"admin_{user_id}" if user_id == '1' else
f"employee_{user_id}",
        "role": "Super Admin" if user_id == '1' else "Financial
Analyst",
        "email": f"admin@nexova.com" if user_id == '1' else
f"emp{user_id}@nexova.com",
    }
    content_html = render_template_string(DASHBOARD_CONTENT,
user_data=user_data)
    html = render_template_string(BASE_TEMPLATE, content=content_html)
    return html

@app.route('/api/v1/auth/login', methods=['POST'])
def api_login():
    data = request.get_json(silent=True) or request.form.to_dict()
    log_request('/api/v1/auth/login', data, tags=['api',
'login_attempt'])
    time.sleep(0.5)
    return jsonify({"error": "Invalid credentials or account
locked."}), 401

@app.route('/api/v1/users', methods=['GET', 'POST'])
def api_get_user():
    data = request.get_json(silent=True) or request.form.to_dict() or
dict(request.args)
    user_id = request.args.get('id')
    log_request(f'/api/v1/users', data, tags=['api', 'bola_attempt'])
    if user_id:
        return jsonify({
            "id": user_id,
            "username": f"sysadmin_{user_id}",
            "role": "privileged_user",
            "email": f"admin{user_id}@nexova.internal",
            "status": "active"
        }), 200
    return jsonify({"error": "Missing required parameter: 'id'"}), 400
```

```
@app.route('/api/v1/payment/checkout', methods=['POST', 'GET'])
def api_checkout():
    data = request.get_json(silent=True) or request.form.to_dict() or
dict(request.args)
    log_request('/api/v1/payment/checkout', data, tags=['api',
'payment_exploit'])
    return jsonify({
        "status": "failed",
        "error_code": "ERR_GATEWAY_MISCONFIG",
        "message": "Payment gateway configuration error. Debug mode
enabled: Check payload parameters."
    }), 500

@app.route('/<path:path>', methods=['GET', 'POST', 'PUT', 'DELETE',
'PATCH', 'OPTIONS'])
def catch_all(path):
    data = request.get_json(silent=True) or request.form.to_dict() or
dict(request.args)
    log_request(f'/{path}', data, tags=['catch_all', 'scanner'])
    return jsonify({"error": "Endpoint not found"}), 404

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

*Nota: Se omitieron las líneas de código correspondientes a plantillas HTML y CSS debido a la extensión de estas.*

## Apéndice B – Módulo de Machine Learning

```
A continuación, se presenta el código desarrollado para los módulos de
machine learning y los modelos Random Forest y K-Means. El archivo
gestiona la carga del dataset de datos, el preprocesamiento,
entrenamiento de modelos y evaluación de resultados.
import json
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report, confusion_matrix
import os

# Ruta dinámica al archivo de log generado por la aplicación Flask
LOG_FILE = os.path.join(os.path.dirname(os.path.abspath(__file__)),
           'app', 'logs', 'honeypot_attacks.log')

def load_data(filepath):
    data = []
    if not os.path.exists(filepath):
        print(f"No se encontró el archivo {filepath}")
        return pd.DataFrame()
    with open(filepath, 'r', encoding='utf-8') as f:
        for line in f:
            try:
                data.append(json.loads(line.strip()))
            except json.JSONDecodeError:
                continue
    return pd.DataFrame(data)

def preprocess_data(df):
```

```
    if df.empty:
        return df

    # Extraer el User-Agent de los headers almacenados como diccionario
    df['user_agent'] = df['headers_recibidos'].apply(
        lambda x: x.get('User-Agent', 'Unknown') if isinstance(x, dict)
else 'Unknown'
    )

    # Construir la variable objetivo: 1 = ataque, 0 = tráfico legítimo
    attack_tags = ['bola_attempt', 'payment_exploit', 'scanner']
    df['is_attack'] = df['tags'].apply(
        lambda tags: 1 if isinstance(tags, list) and
        any(t in tags for t in attack_tags) else 0
    )

    # Calcular longitud del payload como feature de ingeniería de
características
    df['payload_length'] = df['datos_peticion'].apply(
        lambda x: len(str(x)) if x else 0
    )
    return df

def train_models():
    df = load_data(LOG_FILE)
    if df.empty:
        print("No hay datos para entrenar. Asegúrate de ejecutar el
honeypot "
              "y el simulador de tráfico primero.")
        return

    df = preprocess_data(df)
    print(f"Total de registros cargados: {len(df)}")

    # Codificación numérica de variables categóricas
    le_method   = LabelEncoder()
    le_endpoint = LabelEncoder()
    le_ua       = LabelEncoder()

    df['method_encoded']       =
le_method.fit_transform(df['metodo_http'])
    df['endpoint_encoded']     =
le_endpoint.fit_transform(df['endpoint_accedido'])
    df['user_agent_encoded']   = le_ua.fit_transform(df['user_agent'])

    # Definir matriz de características (X) y variable objetivo (y)
    X = df[['method_encoded', 'endpoint_encoded',
            'user_agent_encoded', 'payload_length']]
    y = df['is_attack']

    # Validación mínima antes de entrenar
    if len(df) < 10 or len(y.unique()) < 2:
        print("No hay suficientes datos o variabilidad para entrenar
los modelos.")
        print("Ejecuta el simulador de tráfico varias veces para
generar "
              "un dataset rico.")
        return

    # Estandarización de variables
    scaler   = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # División entrenamiento / prueba: 70% / 30%
```

```
    X_train, X_test, y_train, y_test = train_test_split(
        X_scaled, y, test_size=0.3, random_state=42
    )

    # ================================================
    # Modelo Supervisado: Random Forest
    # ================================================
    print("\n--- Entrenando Random Forest ---")
    rf = RandomForestClassifier(n_estimators=100, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)

    print("Matriz de Confusión:\n", confusion_matrix(y_test, y_pred))
    print("\nReporte de Clasificación:\n",
          classification_report(y_test, y_pred, zero_division=0))

    # ================================================
    # Modelo No Supervisado: K-Means
    # ================================================
    print("\n--- Entrenando K-Means (No Supervisado) ---")
    kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
    df['cluster'] = kmeans.fit_predict(X_scaled)

    print("Distribución de Clusters:\n", df['cluster'].value_counts())
    print("\nCrosstab Etiquetas Reales vs Clusters (solo para
evaluación):")
    print(pd.crosstab(df['is_attack'], df['cluster']))

if __name__ == "__main__":
    train_models()
```

**Apéndice C – Código del simulador de Tráfico**

A continuación, se presenta el código utilizado para el desarrollo de un generador de tráfico. El archivo se encarga de la generación de dos tipos de tráfico: tráfico legítimo y tráfico malicioso.

```
import requests
import random
import time
import string
import threading

HONEYPOT_URL = "http://localhost:8080"
ENDPOINTS_VALIDOS = ["/", "/login", "/dashboard", "/api/v1/users",
                     "/api/v1/payment/checkout", "/products",
"/contact"]

ENDPOINTS_OCULTOS = [
    "/admin/hidden", "/.env", "/api/v1/config", "/backup.zip",
    "/swagger-ui.html", "/.git/config", "/server-status",
    "/phpinfo.php", "/wp-config.php.bak"
]

USER_AGENTS_MALICIOSOS = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64)",
    "curl/7.68.0",
    "python-requests/2.25.1",
    "Nmap Scripting Engine",
    "DirBuster-1.0-RC1",
    "sqlmap/1.5.8#dev",
    "Nikto/2.1.6",
```

```
    "masscan/1.3.2"
]

ejecucion_activa = True

def ataque_fuerza_bruta():
    url = f"{HONEYPOT_URL}/api/v1/auth/login"
    headers = {"User-Agent": random.choice(USER_AGENTS_MALICIOSOS)}
    for _ in range(5):
        payload = {
            "username": random.choice(["admin", "root", "user",
"administrator"]),
            "password": ''.join(random.choices(string.ascii_letters +
string.digits, k=8))
        }
        try:
            requests.post(url, json=payload, headers=headers,
timeout=2)
        except requests.exceptions.RequestException:
            pass

def ataque_enumeracion():
    headers = {"User-Agent": random.choice(["DirBuster-1.0-RC1",
"Nikto/2.1.6"])}
    for _ in range(3):
        endpoint = random.choice(ENDPOINTS_OCULTOS)
        url = f"{HONEYPOT_URL}{endpoint}"
        try:
            requests.get(url, headers=headers, timeout=2)
        except requests.exceptions.RequestException:
            pass

def inyeccion_sql():
    url = f"{HONEYPOT_URL}/api/v1/users"
    headers = {"User-Agent": "sqlmap/1.5.8#dev"}
    payloads = ["' OR 1=1 --", "admin' #",
                "' UNION SELECT null, null--", "1; DROP TABLE users"]
    params = {"id": random.choice(payloads)}
    try:
        requests.get(url, params=params, headers=headers, timeout=2)
    except requests.exceptions.RequestException:
        pass

def ataque_broken_access_control():
    headers = {"User-Agent": random.choice(USER_AGENTS_MALICIOSOS)}
    payloads = ["../../../../etc/passwd",
"..%2f..%2f..%2fetc%2fpasswd",
                "/admin/dashboard", "/api/v1/admin/delete/1"]
    url = f"{HONEYPOT_URL}/{random.choice(payloads)}"
    try:
        requests.get(url, headers=headers, timeout=2)
    except requests.exceptions.RequestException:
        pass

def inyeccion_xss():
    url = f"{HONEYPOT_URL}/search"
    headers = {"User-Agent": random.choice(USER_AGENTS_MALICIOSOS)}
    payloads = ["<script>alert(1)</script>",
                "\"><svg/onload=alert(1)>", "javascript:alert('XSS')"]
    params = {"q": random.choice(payloads)}
    try:
        requests.get(url, params=params, headers=headers, timeout=2)
    except requests.exceptions.RequestException:
        pass
```

```
def ataque_ssrf():
    url = f"{HONEYPOT_URL}/api/v1/fetch"
    headers = {"User-Agent": random.choice(USER_AGENTS_MALICIOSOS)}
    payloads = ["http://169.254.169.254/latest/meta-data/",
                "file:///etc/passwd", "http://localhost:22"]
    params = {"url": random.choice(payloads)}
    try:
        requests.get(url, params=params, headers=headers, timeout=2)
    except requests.exceptions.RequestException:
        pass

def inyeccion_comandos():
    url = f"{HONEYPOT_URL}/ping"
    headers = {"User-Agent": random.choice(USER_AGENTS_MALICIOSOS)}
    payloads = ["127.0.0.1; ls -la", "8.8.8.8 | cat /etc/passwd",
                "localhost && whoami"]
    params = {"ip": random.choice(payloads)}
    try:
        requests.get(url, params=params, headers=headers, timeout=2)
    except requests.exceptions.RequestException:
        pass

def trafico_ruido_blanco():
    endpoint = random.choice(ENDPOINTS_VALIDOS)
    url = f"{HONEYPOT_URL}{endpoint}"
    headers = {"User-Agent": random.choice([
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 "
        "(KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36",
        "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/605.1.15 "
        "(KHTML, like Gecko) Version/14.1.1 Safari/605.1.15",
        "Mozilla/5.0 (iPhone; CPU iPhone OS 14_6 like Mac OS X) "
        "AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0
Mobile/15E148 Safari/604.1"
    ])}
    try:
        requests.get(url, headers=headers, timeout=2)
    except requests.exceptions.RequestException:
        pass

def hilo_trafico_normal():
    print("[+] Iniciando hilo de tráfico normal...")
    while ejecucion_activa:
        trafico_ruido_blanco()
        time.sleep(random.uniform(0.1, 0.5))

def hilo_trafico_malicioso():
    print("[!] Iniciando hilo de tráfico malicioso (OWASP Top 10)...")
    ataques = [
        ataque_fuerza_bruta, ataque_enumeracion, inyeccion_sql,
        ataque_broken_access_control, inyeccion_xss,
        ataque_ssrf, inyeccion_comandos
    ]
    while ejecucion_activa:
        comportamiento = random.choice(ataques)
        comportamiento()
        time.sleep(random.uniform(0.5, 2.0))

if __name__ == "__main__":
    print("Iniciando simulación avanzada de tráfico (Normal + Anómalo
Simultáneo)...")
    print("Presione Ctrl+C para detener.")
```

```
    t_normal = threading.Thread(target=hilo_trafico_normal)
    t_ataque = threading.Thread(target=hilo_trafico_malicioso)

    t_normal.start()
    t_ataque.start()

    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        print("\n[X] Señal de detención recibida. Finalizando
hilos...")
        ejecucion_activa = False
        t_normal.join()
        t_ataque.join()

    print("Simulación finalizada. Los logs están listos para

preprocesamiento.")
```

**Apéndice D – Archivos de Configuración**

A continuación, se presentan los archivos para la configuración de la infraestructura del sistema. El archivo Docker-compose.yml para los componentes del sistema como Kibana y Filebeat y el archivo filebeat.yml para el agente recolector y transportador de logs.

*Docker-compose.yml*

```
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.12.0
    container_name: es-honeypot-node
    environment:
      - node.name=es-honeypot-node
      - cluster.name=honeypot-analytics-cluster
      - discovery.type=single-node
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms1g -Xmx1g"
      - xpack.security.enabled=false
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - es_data:/usr/share/elasticsearch/data
    ports:
      - "9200:9200"
    networks:
      - honeypot-net

  kibana:
    image: docker.elastic.co/kibana/kibana:8.12.0
    container_name: kibana-honeypot-dashboard
    ports:
      - "5601:5601"
    environment:
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
    depends_on:
      - elasticsearch
```

```
    networks:
      - honeypot-net
```

*filebeat:*

```
    image: docker.elastic.co/beats/filebeat:8.12.0
    container_name: filebeat-honeypot-shipper
    user: root
    volumes:
      - ./log-
system/filebeat/config/filebeat.yml:/usr/share/filebeat/filebeat.yml:ro
      - ./logs-compartidos:/var/log/honeypot:rw
    depends_on:
      - elasticsearch
    networks:
      - honeypot-net
    command: ["filebeat", "-e", "-strict.perms=false"]

  honeypot:
    build:
      context: ./honeypot
      dockerfile: Dockerfile
    container_name: honeypot-api-core
    ports:
      - "8080:8080"
    volumes:
      - ./logs-compartidos:/var/log/honeypot:rw
    networks:
      - honeypot-net

volumes:
  es_data:
    driver: local

networks:
  honeypot-net:
    driver: bridge
filebeat.yml
# ========================== Filebeat Inputs ========================
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /var/log/honeypot/honeypot_attacks.log

  json.keys_under_root: true
  json.overwrite_keys: true
  json.add_error_key: true

# ======================== Elasticsearch Output ========================
output.elasticsearch:
  hosts: ["http://elasticsearch:9200"]
  index: "honeypot-api-logs-%{+yyyy.MM.dd}"

setup.template.name: "honeypot"
setup.template.pattern: "honeypot-api-logs-*"
setup.template.enabled: false
setup.ilm.enabled: false
```

## Apéndice E – Enlace GitHub

A continuación, se carga el link del repositorio donde se desarrollaron los códigos de los diferentes componentes que conforman el sistema presentado.

https://github.com/Sma126/TesisMaestria.git