Universidad de Valladolid

Escuela de Ingeniería Informática
de Valladolid

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Informática
Mención De Tenconolgías de la Información

Sistema de Threat Intelligence para la evaluación de Indicadores de Compromiso (IoCs)

Autor:
Víctor Martín Miguel

Tutores:
Dr. César Llamas Bello
D. Manuel López Pérez

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas que me han acompañado a lo largo de este camino académico. En primer lugar, a mi familia, por su apoyo incondicional, su paciencia y por estar siempre presente en los momentos clave de este proceso.

A mis amigos y compañeros de carrera, con quienes he compartido no solo clases y trabajos, sino también experiencias, desafíos y muchas risas que han hecho más llevadero este recorrido.

Y, por supuesto, a todo el profesorado que me ha acompañado durante estos años, en especial a quienes han sabido transmitir su pasión por la informática y han contribuido, directa o indirectamente, a la realización de este trabajo. Gracias por su dedicación, exigencia y compromiso con nuestra formación.

A todos, gracias.

Resumen

El crecimiento constante de las amenazas cibernéticas ha impulsado la necesidad de desarrollar sistemas automatizados capaces de detectar y analizar Indicadores de Compromiso (IoCs) en tiempo real. Entre los principales retos de este campo se encuentra la integración eficaz de fuentes de inteligencia, la normalización de datos heterogéneos y la priorización de amenazas según su relevancia. En este trabajo, se aborda el diseño e implementación de un sistema de *Threat Intelligence* que permite la recolección, enriquecimiento y análisis de IoCs, con el fin de facilitar la toma de decisiones en entornos de ciberseguridad.

En concreto, se ha construido una solución funcional que descarga IoCs desde fuentes públicas como OTX, ThreatFox,MalwareBazaar, ThreatView y URLhaus los enriquece con información contextual (país, tipo, palabras clave, repeticiones) y calcula un índice de riesgo que actua como puntuación mediante un sistema de scoring. Los datos se almacenan y visualizan mediante una pila ELK personalizada y *dashboards* desarrollados con Flask. Se ha comprobado la utilidad del sistema para detectar amenazas relevantes y facilitar su análisis mediante filtros, gráficos y criterios dinámicos de priorización.

Palabras clave: Threat Intelligence, Indicadores de Compromiso, IoCs, ciberseguridad, Elasticsearch, scoring de amenazas.

Abstract

The continuous growth of cyber threats has driven the need to develop automated systems capable of detecting and analyzing Indicators of Compromise (IoCs) in real time. One of the main challenges in this domain is the effective integration of intelligence sources, normalization of heterogeneous data, and prioritization of threats based on contextual relevance. This work presents the analysis, design, and implementation of a Threat Intelligence system that collects, enriches, and scores IoCs to support cybersecurity decision-making processes.

Specifically, a functional solution has been developed to retrieve IoCs from public sources such as OTX, ThreatFox,MalwareBazaar, ThreatView and URLhaus enrich them with contextual information (e.g., country, type, tags, frequency), and compute a threat score based on semantic criteria. The data is stored and visualized using a custom ELK stack and Flask-based dashboards. The system has been tested to validate its ability to detect relevant threats and provide useful visual and analytical tools for threat prioritization and exploration.

Key words: Threat Intelligence, Indicators of Compromise, IoCs, cybersecurity, Elasticsearch, threat scoring.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los últimos años, el crecimiento de las amenazas cibernéticas ha adquirido una dimensión sin precedentes. Cada día se registran miles de incidentes relacionados con el robo de información, fraudes digitales, ataques de denegación de servicio (DDoS), y una gran variedad de técnicas ofensivas que afectan tanto a particulares como a organizaciones públicas y privadas. Frente a este contexto, surge una necesidad urgente de desarrollar tecnologías que permitan detectar, analizar y mitigar amenazas de forma eficiente y, sobre todo, proactiva. En este marco, la inteligencia de amenazas (Threat Intelligence) se posiciona como una disciplina fundamental.

La inteligencia de amenazas es el conjunto de procesos, tecnologías y herramientas orientadas a la recopilación y análisis de datos sobre amenazas potenciales o reales. Su objetivo es proporcionar a los responsables de seguridad la información necesaria para prevenir ataques o reducir su impacto. Uno de los principales pilares sobre los que se construye esta inteligencia es la gestión de Indicadores de Compromiso (IoCs, por sus siglas en inglés), que son rastros técnicos generados por una actividad maliciosa, como una dirección IP asociada a un atacante, un hash de un archivo malicioso, un dominio sospechoso o una URL que aloja malware.

Tradicionalmente, la gestión de estos indicadores se ha realizado manualmente o mediante herramientas propietarias que dificultan su estudio o integración con otros sistemas. Además, muchos entornos de seguridad carecen de mecanismos adecuados para enriquecer estos IoCs con información contextual, como su procedencia geográfica, la fecha de su detección, su relación con campañas específicas o grupos de amenazas persistentes avanzadas (APT). Esta falta de contexto limita la utilidad práctica del indicador, ya que no permite establecer una valoración precisa de su peligrosidad ni facilita la toma de decisiones informadas.

Este proyecto aborda precisamente este desafío, proponiendo una solución completa y modular para la gestión de IoCs, basada en tecnologías abiertas y fácilmente replicables. El objetivo es permitir la descarga automatizada de indicadores desde fuentes públicas, su enriquecimiento mediante metadatos relevantes, su almacenamiento en una base de datos escalable (Elasticsearch), y su análisis y visualización

a través de dashboards personalizables. Todo esto se realiza con un enfoque académico y práctico, que permita tanto el aprendizaje profundo del ciclo de vida de un IoC como su implementación en entornos reales.

La arquitectura sobre la que se construye este sistema se fundamenta en la pila ELK: Elasticsearch, Logstash y Kibana. Estas herramientas permiten indexar, transformar y visualizar grandes volúmenes de datos en tiempo real. Sin embargo, se ha optado también por construir un dashboard alternativo en HTML5 y Flask, con el fin de ofrecer una capa adicional de control, personalización y acceso directo a los datos para usuarios sin conocimientos específicos de Kibana.

El proyecto se centra en la integración con Open Threat Exchange (OTX) principalmete, una plataforma colaborativa mantenida por AlienVault (ahora parte de ATT) que permite compartir información de amenazas entre profesionales de la ciberseguridad. OTX ofrece un API REST que permite acceder a millones de indicadores reportados por la comunidad de diferentes tipos y categorías. Gracias a esta fuente de datos, el sistema puede obtener en tiempo real información sobre nuevas amenazas, incluyendo hashes de malware, dominios, direcciones IP y URLs asociadas a campañas maliciosas.

La importancia de este trabajo no solo radica en su valor técnico, sino en su aplicabilidad práctica. Un sistema bien diseñado de inteligencia de amenazas permite detectar con anticipación, comportamientos anómalos en una red, asociar eventos aparentemente inconexos, y establecer mecanismos de respuesta más efectivos. Además, permite ahorrar recursos, evitar pérdidas económicas a las empresas y particulares y proteger cualquier tipo de infraestructura crítica. En entornos donde no se dispone de grandes presupuestos, este tipo de soluciones basadas en software libre son especialmente valiosas.

1.1. Motivación

El presente proyecto tiene múltiples motivaciones que abarcan tanto aspectos técnicos como académicos y profesionales. En primer lugar, surge del interés personal por la ciberseguridad, una disciplina en constante evolución que requiere actualización y formación continua. La gestión de IoCs representa uno de los pilares fundamentales en cualquier estrategia de defensa, y comprender su ciclo de vida es clave para proteger sistemas informáticos de forma proactiva.

Durante la formación universitaria, muchas veces se tratan temas de seguridad desde un enfoque teórico/práctico limitado a llevado a cabo con herramientas concretas, sin llegar a abordar la integración completa de un sistema real de análisis de amenazas. Este proyecto busca llenar ese vacío mediante la construcción desde cero de una arquitectura funcional, desde la obtención de datos hasta su análisis visual final, pasando por el enriquecimiento y el almacenamiento eficiente. Este enfoque integral permite consolidar múltiples competencias adquiridas durante el grado universitario, como la programación en Python, la administración y tecnologías de bases de datos, el diseño de interfaces gráficas, el análisis y tratamiento de datos y la documentación técnica de todos estos elementos.

A nivel técnico, uno de los grandes desafíos era conseguir que el sistema fuera escalable, modular y resistente a IoCs duplicados, es decir, que no se generaran inconsistencias ni entradas repetidas en la base de datos. Esto exigió el diseño de una lógica de control que validara los indicadores antes de su inserción, usando el campo ïndicatorçomo identificador único. Además, se propuso un sistema de scoring o puntuación que, a partir de criterios como la antigüedad del IoC, su procedencia geográfica, su asociación a actores APT o sus técnicas de ataque (TTPs), pudiera calcular una valoración cuantitativa del riesgo que representa.

Desde el punto de vista profesional, este proyecto representa una experiencia valiosa de cara a futuros entornos laborales en cualquier tipo de empresa. Las herramientas utilizadas (Elasticsearch, APIs REST, visualización web, scoring, etc.) son ampliamente demandadas en el mercado de la ciberseguridad y el análisis de datos. Tener experiencia demostrable en la integración de estas tecnologías, en un contexto realista y documentado, permite al desarrollador destacar en procesos de selección o entrevistas técnicas.

Por último, existe una motivación altruista y académica. Este proyecto puede ser compartido y reutilizado por otros estudiantes, investigadores o entusiastas de la ciberseguridad que deseen aprender o construir sobre esta base. Al usar tecnologías abiertas y documentar cada paso del proceso, se favorece la colaboración, la reproducibilidad y la mejora continua. Se pretende, en definitiva, crear una herramienta útil, educativa y adaptable a distintos escenarios.

1.2. Metodología para un Sistema de Threat Intelligence

1.2.1. Objetivos

El objetivo general del proyecto es diseñar e implementar un sistema completo de *Threat Intelligence* que permita gestionar IoCs de forma automatizada, eficiente y visual. Este sistema debe incluir las siguientes capacidades:

- Obtener indicadores de compromiso desde la API de OTX principalmente, pero también del resto de fuentes.
- Enriquecer los IoCs con información contextual relevante: fechas de aparición, fuente, actor relacionado, TTP, país de origen, *tags*, etc.
- Incorporar datos de geolocalización de IPs a través de la base de datos GeoLite2.
- Validar que los indicadores no estén duplicados antes de su inserción, empleando mecanismos previas mediante un ID único como es el campo `indicator`.
- Almacenar los datos en un índice de Elasticsearch optimizado para búsquedas rápidas y filtrados múltiples.

- Calcular un *threat score* para cada indicador según un algoritmo de puntuación basado en criterios objetivos.
- Visualizar los indicadores en entornos gráficos: Una web en HTML con filtros interactivos y Kibana (gracias a su fácil integración con Elasticsearch)
- Facilitar la integración del sistema con nuevas fuentes de datos en el futuro.
- Documentar todos los componentes del sistema para su reutilización.

A continuación, se presenta la arquitectura general del sistema y el flujo de actividades asociado al proceso de gestión de indicadores de compromiso:

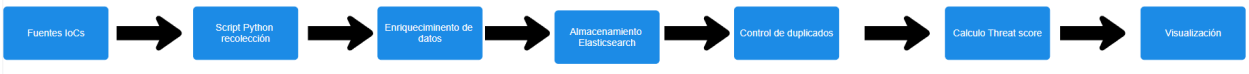

Figura 1.1: Flujo de trabajo del sistema de *Threat Intelligence*.

La Figura 1.1 muestra los principales módulos del sistema: desde la obtención de datos desde OTX, su enriquecimiento con metadatos y geolocalización, control de duplicados y almacenamiento en Elasticsearch, hasta su visualización mediante Kibana o un dashboard HTML personalizado.

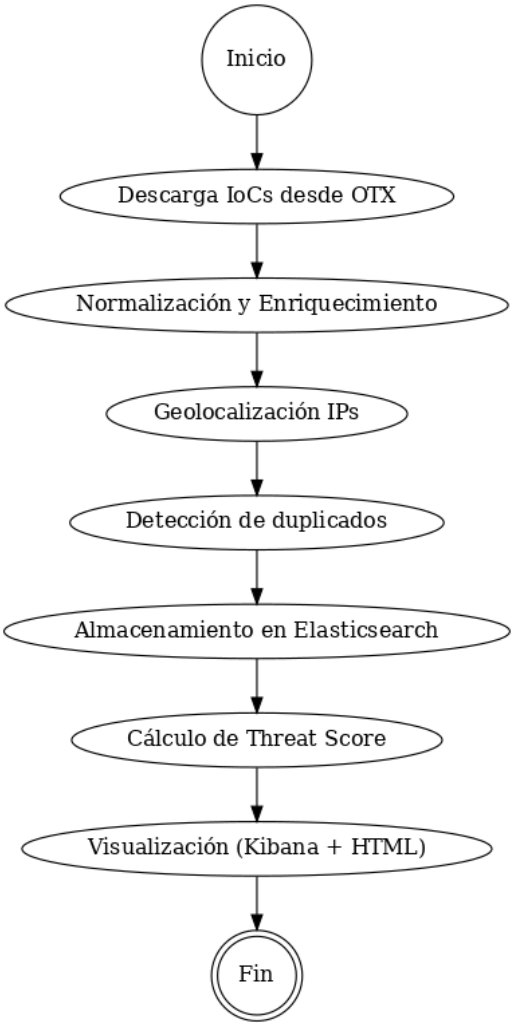

Figura 1.2: Diagrama de actividades del flujo de gestión de IoCs.

La Figura 1.2 representa el flujo lógico de las actividades del sistema, desde la recolección de IoCs hasta su análisis visual, facilitando la comprensión del ciclo de vida completo del dato dentro del sistema propuesto.

1.2.2. Etapas metodológicas del sistema

El sistema se estructura en una serie de etapas que definen el mecanismo de resolución empleado. Esta metodología está basada en buenas prácticas extraídas de la literatura especializada [1–3] y se alinea con los principios de ciberinteligencia proactiva. A continuación, se describen las fases principales:

1. **Recolección de datos desde OTX y demás fuentes**:
    - Estudio de la documentación de la API.
    - Desarrollo del script de descarga automática.

- Almacenamiento inicial en CSV.

2. **Normalización y enriquecimiento**:
   - Estandarización de campos (`type, indicator, description, ...`).
   - Enriquecimiento con metadatos extraídos de los *pulses*.
   - Incorporación de campos como `uuid, category, tags, first seen, last seen`.
3. **Geolocalización de IPs**:
   - Implementación de búsqueda por IP usando la base GeoLite2.
   - Asociación automática de país, ciudad y continente.
4. **Almacenamiento en Elasticsearch**:
   - Instalación y configuración del clúster local.
   - Definición del esquema de los documentos.
   - Indexación de los IoCs desde Python.
5. **Detección de duplicados**:
   - Consulta previa por campo `indicator`.
   - Lógica de inserción condicional y campo `seen_count`.
6. **Scoring contextual**:
   - Diseño del algoritmo de puntuación.
   - Pruebas de calibración con valores reales.
7. **Visualización con Kibana**:
   - Dashboards por tipo, país, puntuación.
   - Mapas de calor y tablas dinámicas.
8. **Visualización en HTML**:
   - Desarrollo con Flask y plantillas HTML.
   - Filtros interactivos por tipo, país y score.
9. **Validación y documentación**:
   - Pruebas funcionales y rendimiento.
   - Redacción de manuales técnicos.

Justificación metodológica

Este enfoque metodológico refleja una arquitectura modular y escalable, orientada al tratamiento completo de los IoCs. Cada etapa se implementa de forma desacoplada, lo que permite actualizar componentes sin afectar al sistema global. Este modelo es especialmente útil en entornos donde los datos cambian continuamente y la adaptabilidad es crítica.

La planificación concreta de cada tarea, con fechas y entregables, se aborda detalladamente en el Capítulo 2.

1.3. Recursos utilizados

Para el desarrollo de este proyecto, se han empleado recursos tanto físicos como lógicos, todos seleccionados con el objetivo de crear un entorno controlado, reproducible y económico. A continuación se detallan los principales componentes empleados:

Recursos físicos

- Ordenador personal con sistema operativo Ubuntu 22.04 LTS, procesador Intel i7, 16 GB de RAM y 1 TB de almacenamiento SSD. Este equipo ha servido como entorno de desarrollo principal y nodo de pruebas para el sistema Elasticsearch.
- Conexión a internet de fibra óptica para realizar pruebas de conexión con APIs externas y actualizar los paquetes necesarios durante el desarrollo.
- Memoria USB para copias de seguridad y transporte de ficheros entre dispositivos.

Recursos de software

- **Python 3.10**: lenguaje de programación principal para el desarrollo de scripts de recolección de IoCs, enriquecimiento y conexión con Elasticsearch.
- **Librerías Python utilizadas:**
  - `requests`: para la conexión con la API REST de OTX.
  - `pandas`: para el tratamiento y transformación de datos tabulares.
  - `elasticsearch`: cliente oficial para la conexión a Elasticsearch desde Python.
  - `flask`: para el desarrollo de la aplicación web personalizada.

  - `plotly`: para la generación de gráficos interactivos.
  - `geoip2`: para realizar consultas de geolocalización mediante MaxMind GeoLite2.

- **Elasticsearch 8.x**: motor de búsqueda y base de datos documental para almacenar y consultar IoCs de forma rápida.
- **Kibana**: plataforma de visualización acoplada a Elasticsearch para la creación de dashboards de análisis visual.
- **GeoLite2**: base de datos gratuita de MaxMind utilizada para geolocalizar direcciones IP extraídas de los IoCs.
- **Git**: sistema de control de versiones utilizado para gestionar el desarrollo del proyecto.

Este conjunto de herramientas ha permitido construir un sistema completamente funcional sin necesidad de licencias comerciales, promoviendo así el uso de software libre y fomentando la posibilidad de reproducir este entorno en laboratorios académicos u organizaciones con recursos limitados.

1.4. Caso de Negocio

1.4.1. Agentes implicados en el proyecto

El desarrollo de este sistema, aunque ejecutado de forma individual como parte de un trabajo de fin de grado, contempla varios agentes y perfiles que se benefician directa o indirectamente de sus funcionalidades:

- **Desarrollador**: autor del proyecto, encargado del diseño, implementación, pruebas y documentación del sistema.
- **Usuario final**: profesional de ciberseguridad o analista de amenazas que consulta los IoCs a través de los dashboards implementados.
- **Administrador del sistema**: responsable de desplegar, mantener y actualizar los componentes del sistema.
- **Supervisor académico**: docente que valida la calidad técnica, metodológica y documental del trabajo realizado.
- **Supervisor de empresa**: Tutor empresarial que propone, evalúa y gestiona el proyecto desde el punto de vista de su organización.

1.4.2. Presupuesto

En una primera aproximación al coste estimado de este proyecto, se han tenido en cuenta tanto factores materiales como el tiempo de desarrollo invertido. Aunque el proyecto se ha realizado en un entorno no comercial, se estima un presupuesto teórico basado en los siguientes conceptos:

Coste de hardware y software:

- Ordenador personal (amortización estimada por 4 años): 1000 € / 4 = **250 €**
- Conectividad, electricidad y almacenamiento adicional: **50 €**
- **Total estimado en recursos materiales: 300 €**

Coste de trabajo:

- Estimación de 4 horas diarias desde febrero hasta junio (aproximadamente 120 días): $4 \times 120 =$ **480 horas**
- Según datos recogidos en Glassdoor [4], el salario promedio de un ingeniero informático junior en España ronda los 10€/hora. Esta cifra se ha tomado como referencia para estimar el coste laboral teórico del proyecto.
- **Total estimado en trabajo: 4800 €**

Presupuesto total estimado: 5100 €

1.4.3. Impacto

La creación de este sistema aporta múltiples beneficios desde el punto de vista técnico, académico y profesional:

- **Impacto técnico**: permite automatizar la recolección, enriquecimiento y visualización de IoCs, integrando distintas tecnologías como APIs REST, bases de datos documentales y herramientas de visualización.
- **Impacto académico**: sirve como modelo de implementación de un sistema de inteligencia de amenazas en contextos educativos, siendo replicable por estudiantes o investigadores.
- **Impacto profesional**: capacita al desarrollador con competencias aplicables en el sector de la ciberseguridad, especialmente en áreas de *Threat Intelligence*, análisis de datos y desarrollo backend.

- **Impacto social**: promueve el uso de herramientas abiertas y el intercambio de conocimiento, favoreciendo una cultura de colaboración y defensa común frente a amenazas digitales.

Este proyecto puede escalarse fácilmente para integrar múltiples fuentes de datos, incluir nuevas capas de análisis o desplegarse en entornos corporativos de mayor envergadura.

1.5. Organización del documento

Este Trabajo de Fin de Grado se organiza en los siguientes capítulos:

- **Capítulo 1: Introducción.** Se presenta el contexto del proyecto, su motivación, objetivos generales y específicos, así como el alcance del sistema desarrollado.
- **Capítulo 2: Planificación.** Describe la metodología de trabajo, el cronograma seguido y las herramientas utilizadas para la gestión del proyecto.
- **Capítulo 3: Tecnologías utilizadas.** Se explican en detalle las tecnologías, lenguajes, frameworks y herramientas empleadas para desarrollar el sistema de Threat Intelligence.
- **Capítulo 4: Análisis.** Se identifican los requisitos del sistema y se realiza un estudio de las fuentes de IoCs y criterios para su selección y tratamiento.
- **Capítulo 5: Diseño.** Presenta la arquitectura del sistema, la estructura de datos empleada y el diseño de la puntuación de amenazas.
- **Capítulo 6: Implementación.** Se detalla cómo se ha llevado a cabo la implementación técnica del sistema, incluyendo el backend en Flask, la integración con Elasticsearch y la interfaz web.
- **Capítulo 7: Pruebas.** Se documentan las pruebas realizadas para verificar la correcta funcionalidad del sistema, así como algunos ejemplos de casos reales procesados.
- **Capítulo 8: Conclusiones.** Recoge los resultados obtenidos, las principales conclusiones y posibles líneas de mejora o evolución futura del sistema.

Capítulo 2

Planificación del Proyecto

En este capítulo se lleva a cabo la planificación del proyecto siguiendo la guía de PMBOK [**?**]. La elaboración del sistema definido en los objetivos se realiza mediante un proceso iterativo, ya que se puede dividir fácilmente en etapas bien definidas.

En primer lugar, se realiza una planificación inicial con el diseño de las tareas a partir de las etapas del proyecto definidas en la sección Etapas, en la que además se definen los hitos y por tanto los entregables. Posteriormente, mediante un cronograma se muestra la gestión del tiempo. También se realiza la gestión de los riesgos que pueden aparecer a lo largo del proyecto y por último se realiza una estimación de los costes materiales y laborales.

2.1. Planificación del Proyecto

2.1.1. Planificación inicial

La planificación inicial del proyecto se construyó siguiendo una guía de alto nivel orientada a proyectos de inteligencia de amenazas, tal como fue recomendada por el tutor académico. Esta guía deriva de buenas prácticas descritas en marcos como los de ENISA [3] y NIST [5], donde se promueve una aproximación iterativa, basada en tareas funcionales con entregables parciales.

Dado que este proyecto no se basa en una arquitectura puramente software tradicional, no se estructuró en fases como “análisis”, “diseño” o “implementación”, sino que se organizó por bloques de funcionalidad operativa. Cada fase se abordó con un enfoque de desarrollo incremental, mediante la creación de prototipos funcionales que permitieron validar los avances y detectar ajustes necesarios antes de continuar con la siguiente etapa.

La dedicación estimada fue de 15–20 horas semanales, distribuidas en sesiones de 3 a 4 horas diarias. El proyecto se dividió en seis grandes bloques funcionales:

- **Fase 1 – Preparación del entorno (Semanas 1–2)**: instalación de tecnologías base (Python, Elasticsearch, Kibana, Flask), exploración de la API de OTX y verificación del flujo de conexión inicial.
- **Fase 2 – Recolección de datos (Semanas 3–4)**: desarrollo de un script robusto en Python para la obtención automatizada de IoCs desde OTX, guardado en CSV y validación de estructura de datos.
- **Fase 3 – Enriquecimiento y geolocalización (Semanas 5–6)**: integración de metadatos (tags, actores, TTPs, fechas), y aplicación de geolocalización mediante GeoLite2 para direcciones IP.
- **Fase 4 – Scoring y deduplicación (Semanas 7–8)**: implementación de un sistema de puntuación contextual configurable y detección de duplicados mediante consulta previa en Elasticsearch.
- **Fase 5 – Visualización (Semanas 9–10)**: diseño de dashboards en Kibana y creación de un dashboard HTML en Flask, incluyendo gráficos interactivos y filtros personalizados.
- **Fase 6 – Validación y documentación (Semanas 11–12)**: integración final, pruebas de rendimiento, documentación técnica y preparación de entregables.

En cada fase se construyó un prototipo parcial o funcional (por ejemplo, script de descarga, índice en Elasticsearch, interfaz HTML), que sirvió para evaluar el resultado antes de continuar. Este enfoque ayudó a identificar posibles mejoras sin comprometer el avance general.

Figura 2.1: Planificación inicial del proyecto por fases

2.1.2. Seguimiento del proyecto

Durante el desarrollo del proyecto se adoptó un enfoque de seguimiento iterativo, apoyado en la construcción de prototipos funcionales en cada fase. Este enfoque permitió verificar el cumplimiento de los objetivos parciales y ajustar las tareas y tiempos en función de la experiencia real acumulada.

En lugar de seguir un cronograma cerrado, se evaluaba periódicamente el progreso, lo que permitió aplicar correcciones tempranas. Por ejemplo, el diseño del dashboard HTML resultó más complejo de lo esperado, lo que requirió redistribuir tiempo desde otras fases. Por el contrario, la automatización de la descarga desde OTX se implementó de forma más rápida de lo previsto.

El tutor académico fue informado semanalmente del avance, y los hitos se validaron mediante entregas funcionales de scripts, configuraciones de Elasticsearch, y prototipos visuales. Esta dinámica proporcionó un marco ágil, pero suficientemente estructurado, para garantizar la calidad del sistema final.

La siguiente tabla recoge las tareas principales, comparando la duración estimada en la planificación inicial con la duración real obtenida tras finalizar el proyecto:

Tarea principal	**Duración estimada**	**Duración real**
Preparación del entorno y pruebas iniciales	2 semanas	2 semanas
Recolección de IoCs desde OTX	2 semanas	1.5 semanas
Enriquecimiento, normalización y geolocalización	2 semanas	2 semanas
Scoring y detección de duplicados	2 semanas	2.5 semanas
Visualización: Kibana + HTML Flask	2 semanas	3 semanas
Validación final y documentación técnica	2 semanas	2 semanas

Tabla 2.1: Comparativa entre planificación estimada y desarrollo real del proyecto

El prototipado continuo permitió detectar problemas tempranos (como la configuración de filtros dinámicos o los índices geográficos), lo que favoreció una mejor asignación de tiempos. Esta flexibilidad fue clave para mantener el proyecto alineado con sus objetivos funcionales, garantizando la entrega de un sistema completo, funcional y reutilizable.

2.2. Gestión de Riesgos

La gestión de riesgos del proyecto se ha basado en la clasificación de Barry Boehm, que distingue varios tipos de riesgos asociados al desarrollo de sistemas técnicos: tecnológicos, de personal, de planificación, organizativos y externos. Cada riesgo identificado ha sido evaluado en términos de su probabilidad de ocurrencia y el impacto potencial en los objetivos del proyecto.

A continuación se muestra una tabla simplificada que recoge los principales riesgos detectados, su categoría, valoración y estrategia de respuesta:

Riesgo	**Categoría**	**Prob.**	**Impacto**	**Mitigación**	**Contingencia**
Fallos en la conexión con la API de OTX	Técnico	Alta	Media	Uso de logs y control de errores. Tests previos con curl.	Reintento automático o cambio temporal a carga local desde CSV.
Complejidad inesperada en visualización HTML	Técnico	Media	Alta	Prototipado incremental. Uso de librerías conocidas (Plotly, Flask).	Sustitución por dashboard básico en Kibana si hay bloqueo.
Falta de tiempo por carga académica simultánea	Personal	Alta	Alta	Planificación conservadora. Reservas de tiempo de ajuste.	Reorganización de tareas para centrar en lo esencial.
Cambios en los requisitos o recomendaciones del tutor	Organizativo	Media	Media	Contacto frecuente con el tutor, entregas parciales.	Adaptación del calendario a tareas críticas.
Pérdida de datos o daño del entorno de trabajo	Externo	Baja	Alta	Backups regulares del repositorio y entorno virtual.	Restauración desde copia en nube (GitHub, GDrive).

Tabla 2.2: Principales riesgos identificados y estrategias de mitigación

Con el fin de visualizar de forma más clara la distribución de riesgos, se ha construido una matriz personalizada que representa la probabilidad frente al impacto para cada caso identificado. Esta matriz permite priorizar acciones y establecer el umbral de tolerancia según la naturaleza específica del proyecto.

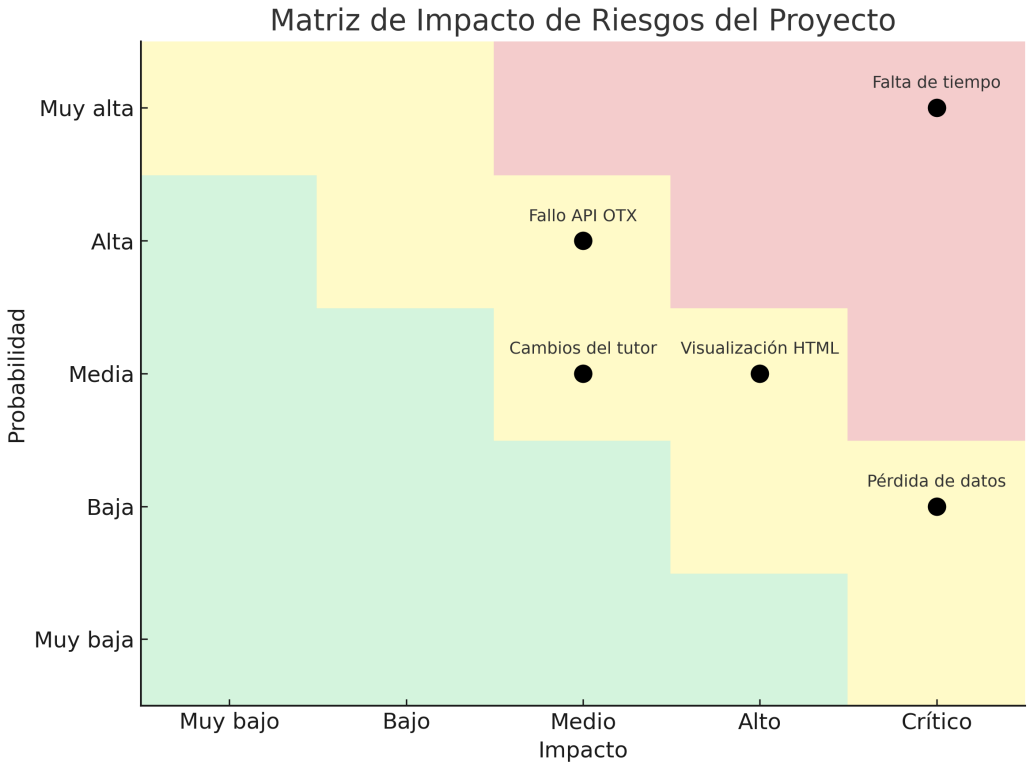

Figura 2.2: Matriz de impacto y probabilidad de los riesgos del proyecto

2.3. Presupuesto del Proyecto

Aunque el presente trabajo no ha supuesto un coste económico real, es posible estimar su valor en un contexto profesional, considerando tanto la dedicación invertida como los recursos necesarios para su desarrollo y eventual despliegue en producción.

Criterios de estimación

La estimación del coste del proyecto se ha realizado según dos bloques principales:

- **Coste de desarrollo:** cálculo del valor económico del trabajo técnico en base a un perfil junior multitarea, que ha ejercido funciones de desarrollo backend, análisis, visualización de datos y documentación técnica.
- **Coste de infraestructura:** simulación del coste de ejecución en un entorno de nube pública (Amazon Web Services) utilizando instancias estándar EC2.

Coste estimado de desarrollo

El desarrollo se ha realizado en 12 semanas, con una media de 20 horas semanales, lo que equivale a un total de 240 horas. Como referencia salarial, se ha tomado un perfil técnico junior en España, con un coste de 10€/hora según datos de Glassdoor [4].

No obstante, en un escenario real de contratación, el coste para el empresario incluiría cotizaciones sociales, seguros, licencias, etc. Por ello, se estima un factor de coste empresarial del 1,5 sobre el coste base, resultando en un coste real de **15€/hora**.

$$240 \text{ horas} \times 15/hora = \textbf{3.600€}$$

Coste estimado de infraestructura en la nube

En caso de querer desplegar el sistema en producción, se ha simulado su ejecución en Amazon Web Services (AWS), utilizando una instancia t3.medium (2 vCPU, 4 GB RAM), suficiente para alojar Elasticsearch, Flask y servicios auxiliares en entorno de pruebas. El coste aproximado sería:

- **Instancia EC2 t3.medium (on-demand):** 0,0416€/h × 24h × 30 días **30€/mes**
- **Almacenamiento EBS 50 GB SSD:** 0,10€/GB/mes × 50 **5€/mes**
- **Transferencia de datos estimada (5 GB/mes):** incluido en capa gratuita

Total mensual estimado de infraestructura: 35€/mes

Resumen del presupuesto estimado

Concepto	**Coste (€)**
Desarrollo técnico (240h × 15€/h)	3.600€
Infraestructura en AWS (1 mes)	35€
Licencias y herramientas (software libre)	0€
Coste total estimado	**3.635€**

Tabla 2.3: Presupuesto preliminar del proyecto

Este presupuesto representa una aproximación realista del valor económico del proyecto, útil para evaluar su viabilidad futura en un entorno profesional. La utilización de software libre y el bajo consumo de infraestructura hacen que el sistema sea fácilmente escalable y sostenible en costes.

Capítulo 3

Tecnologías utilizadas

El desarrollo del presente sistema de Threat Intelligence ha requerido la integración y uso de múltiples tecnologías, tanto a nivel de backend como de visualización y gestión de datos. Dado que uno de los objetivos clave del proyecto es la posibilidad de ser replicado por otros usuarios sin coste, se ha priorizado el uso de herramientas de código abierto, ampliamente documentadas y con una comunidad activa. Este capítulo expone las tecnologías seleccionadas, su papel dentro del sistema, y una evaluación crítica basada en la experiencia adquirida durante el desarrollo.

Cada tecnología será presentada desde tres enfoques: su propósito dentro del sistema, sus características principales y una evaluación propia de ventajas, limitaciones y posibles alternativas.

3.1. Python

Descripción general

Python es un lenguaje de programación de alto nivel, interpretado y multiparadigma, que soporta programación imperativa, orientada a objetos y funcional. Fue creado por Guido van Rossum y publicado por primera vez en 1991. Su diseño enfatiza la legibilidad del código y la productividad del desarrollador gracias a una sintaxis clara y concisa. Hoy en día, Python es uno de los lenguajes más utilizados en el ámbito académico y profesional, siendo especialmente popular en áreas como ciencia de datos, automatización de tareas, desarrollo web y ciberseguridad [6]. **En este proyecto se ha utilizado Python en su versión 3.11.8.**

Su popularidad ha crecido de forma exponencial en los últimos años. Según el índice TIOBE de junio de 2025, Python es el lenguaje más popular del mundo por cuarto año consecutivo [7]. También es el lenguaje más enseñado en universidades, tanto para iniciación a la programación como para asignaturas

de análisis de datos o inteligencia artificial [8].

Rol dentro del proyecto

Python ha constituido la columna vertebral del sistema de *Threat Intelligence* desarrollado. Se ha utilizado en todos los niveles de la arquitectura software, desde la automatización de descargas de IoCs (*Indicadores de Compromiso*) hasta el procesamiento, enriquecimiento, inserción en la base de datos y visualización web mediante un *backend* personalizado. Las principales funciones desarrolladas en Python han sido:

- Descarga automatizada de IoCs desde la API REST de Open Threat Exchange (OTX), utilizando la librería `requests`.
- Transformación y enriquecimiento de datos mediante `pandas`, para aplicar limpieza, normalización y estructuración de campos como `type`, `indicator`, `tags`, `first_seen`, `related_actors`, etc.
- Implementación del algoritmo de *scoring* que evalúa el riesgo de un IoC en función de criterios como antigüedad, origen geográfico, TTPs asociadas y presencia en múltiples fuentes.
- Indexación en Elasticsearch, utilizando la librería oficial `elasticsearch-py`, que permite la inserción y consulta eficiente de documentos JSON.
- Geolocalización de direcciones IP mediante la librería `geoip2`, accediendo a la base de datos GeoLite2 de MaxMind.
- Desarrollo del *backend* web del *dashboard* HTML, implementado con el microframework `Flask`.

Esta versatilidad permite a Python asumir tanto tareas de *scripting* como de *backend* web, lo cual reduce la complejidad del proyecto y evita dependencias con múltiples lenguajes.

Ventajas y puntos fuertes

El uso de Python ha ofrecido numerosas ventajas en el desarrollo del sistema:

- **Productividad y rapidez de desarrollo:** gracias a su sintaxis clara, se ha podido desarrollar y depurar código rápidamente, reduciendo el tiempo de implementación y facilitando la integración entre componentes [9].

- **Amplio ecosistema de librerías:** se dispone de bibliotecas maduras y bien documentadas para prácticamente cualquier tarea, desde acceso a APIs (`requests`) hasta visualización (`plotly`) o bases de datos (`elasticsearch`, `sqlite3`, `sqlalchemy`).
- **Comunidad activa y soporte técnico:** Python cuenta con una de las comunidades más activas del mundo del software libre, lo cual se traduce en abundante documentación, foros de soporte y ejemplos de código.
- **Facilidad de aprendizaje:** la curva de aprendizaje de Python es baja, lo cual ha permitido centrar los esfuerzos en el diseño del sistema y el análisis de amenazas, en lugar de en la sintaxis o depuración compleja.

Limitaciones observadas

A pesar de sus muchas ventajas, también se han identificado algunas limitaciones durante el desarrollo:

- **Rendimiento:** al ser un lenguaje interpretado y de tipado dinámico, Python no está optimizado para tareas de computación intensiva o concurrencia de bajo nivel. Sin embargo, esto no ha supuesto un cuello de botella en este proyecto, dado que las operaciones son mayoritariamente *I/O-bound* y no *CPU-bound*.
- **Gestión de entornos:** en proyectos con múltiples dependencias, es imprescindible usar entornos virtuales (`venv`, `poetry`, `conda`) para evitar conflictos entre versiones de librerías. Esto añade una complejidad inicial en la configuración del entorno de trabajo.
- **Despliegue en producción:** aunque es ideal para prototipado y desarrollo, desplegar aplicaciones Python a producción (por ejemplo, usando `gunicorn` y `nginx`) requiere conocimientos adicionales, especialmente en comparación con lenguajes más orientados a sistemas como Go o Java.

Alternativas consideradas

Durante la fase de diseño, se valoraron otras opciones tecnológicas para el núcleo del sistema:

- **Go (Golang):** ofrece mejor rendimiento y compilación a binario estático, ideal para microservicios. Sin embargo, su ecosistema es menos completo en librerías específicas de ciberseguridad o visualización.
- **Java:** ofrece robustez y madurez para sistemas empresariales, pero introduce una gran sobrecarga sintáctica, poco adecuada para un desarrollo ágil en el contexto de un TFG.

- **Node.js (JavaScript):** muy popular para desarrollo web, pero menos orientado a tareas de análisis de datos y manipulación de ficheros estructurados.

Finalmente, Python fue elegido por su equilibrio entre facilidad de uso, madurez de bibliotecas, integración con Elasticsearch y capacidades de scripting.

Valoración final

Python ha resultado ser una elección adecuada, alineada tanto con los objetivos técnicos como académicos del proyecto. Su uso ha facilitado la implementación de un sistema completo y funcional, promoviendo además buenas prácticas de ingeniería software: modularidad, reutilización de código, claridad en la documentación y portabilidad. Su popularidad en el sector de la ciberseguridad y análisis de amenazas refuerza su idoneidad para este tipo de sistemas [10, 11].

3.2. Flask

Descripción general. Flask es un microframework de desarrollo web escrito en Python, diseñado para ser ligero, flexible y fácil de extender. Fue creado por Armin Ronacher en 2010 como parte de la iniciativa Pocoo. A diferencia de frameworks más complejos como Django, Flask sigue una filosofía minimalista: proporciona lo esencial para crear una aplicación web (enrutamiento, servidor, plantillas), dejando al desarrollador la libertad de decidir cómo organizar el resto del sistema [12]. **En este proyecto se ha utilizado Flask en su versión 2.3.3.**

Está basado en el estándar WSGI (Web Server Gateway Interface) y utiliza Jinja2 como motor de plantillas. Esta simplicidad lo convierte en una opción ideal para proyectos académicos, APIs REST o sistemas con lógica de backend personalizada, como el caso de este sistema de Threat Intelligence.

Uso en el proyecto. Flask ha sido utilizado para implementar el *dashboard web personalizado*, una interfaz alternativa a Kibana que permite visualizar los IoCs procesados y almacenados en Elasticsearch. Su objetivo principal es ofrecer una visualización accesible a usuarios sin conocimientos técnicos, con filtros interactivos y gráficos generados dinámicamente mediante JavaScript y Plotly.

Las funcionalidades implementadas con Flask han sido:

- Desarrollo del *backend web* para servir páginas HTML con los datos actualizados.
- Comunicación con Elasticsearch a través de consultas HTTP y parsing de resultados.

- Enrutamiento dinámico para acceder a las vistas de tabla principal, visualización gráfica (`/charts`) y refresco de IoCs (`/refresh`).
- Integración con templates Jinja2 para mostrar datos filtrables y ordenables desde el navegador.

Esta solución permite separar el componente de visualización del motor de análisis y almacenamiento, siguiendo una arquitectura modular y escalable.

Ventajas observadas. Durante el desarrollo del sistema, Flask ha ofrecido múltiples ventajas:

- *Facilidad de aprendizaje:* su estructura simple y la claridad de su documentación han permitido una implementación rápida sin curva de aprendizaje pronunciada.
- *Flexibilidad total:* no impone estructura de carpetas, ORM ni componentes forzados, lo que lo hace ideal para proyectos personalizados.
- *Compatibilidad con librerías externas:* se ha integrado sin problemas con bibliotecas como `elasticsearch`, `plotly` o `geoip2`.
- *Despliegue sencillo en local o producción:* mediante servidores ligeros como `gunicorn`, Flask puede escalar de pruebas locales a entornos productivos reales.

Limitaciones encontradas. A pesar de sus fortalezas, se identificaron algunas limitaciones:

- *Ausencia de funcionalidades por defecto:* funcionalidades como autenticación, sesiones o validación de formularios deben implementarse manualmente o mediante extensiones, lo que puede aumentar la complejidad en sistemas más grandes.
- *Estructura no opinada:* si bien esto permite flexibilidad, también puede provocar desorganización en equipos grandes o en proyectos de larga duración sin un diseño inicial sólido.
- *Menor rendimiento que frameworks asincrónicos:* al ser sincrónico por defecto, Flask no es ideal para aplicaciones que requieran manejo intensivo de conexiones concurrentes, como WebSockets.

Alternativas consideradas. Se valoraron las siguientes alternativas a Flask:

- **Django**: framework completo con ORM, autenticación y panel de administración incluidos. Fue descartado por su complejidad innecesaria para un sistema de visualización liviano.

- **FastAPI**: más moderno y eficiente, basado en Python asíncrono y con validación automática de datos usando `pydantic`. Aunque prometedor, su sintaxis y despliegue requerían una curva de aprendizaje adicional.
- **Dash**: framework específico para dashboards interactivos en Python. Se descartó por sus limitaciones de personalización del frontend y falta de control del backend.

Valoración final. Flask ha resultado una elección adecuada para implementar un sistema web ligero, rápido de desarrollar y fácilmente integrable con el resto de componentes Python del sistema. Su estructura ha permitido implementar una interfaz accesible y funcional sin necesidad de tecnologías adicionales ni frameworks pesados.

3.3. Elasticsearch

Descripción general. Elasticsearch es un motor de búsqueda y análisis de texto en tiempo real, distribuido y de código abierto, basado en Apache Lucene. Fue desarrollado originalmente por Shay Banon en 2010 y es mantenido por Elastic NV. Elasticsearch permite almacenar, indexar y consultar grandes volúmenes de datos semiestructurados, utilizando un modelo documental (JSON) y una potente sintaxis de consulta declarativa llamada Query DSL [13]. **En este proyecto se ha utilizado Elasticsearch en su versión 8.13.0.**

Gracias a su escalabilidad horizontal, tolerancia a fallos y capacidades analíticas, Elasticsearch se ha convertido en una herramienta de referencia en proyectos de análisis de datos, observabilidad (logging, métricas), ciberseguridad y motores de recomendación.

Uso en el proyecto. En el sistema de Threat Intelligence desarrollado, Elasticsearch ha sido utilizado como base de datos documental principal para almacenar y consultar los Indicadores de Compromiso (IoCs). Su elección se ha basado en las siguientes necesidades:

- **Indexación flexible:** cada IoC se almacena como un documento JSON, lo que permite añadir campos enriquecidos como geolocalización, score, TTPs o actores asociados sin necesidad de esquema rígido.
- **Búsqueda eficiente:** permite consultas rápidas por cualquier campo (tipo, país, fecha, etc.), incluso sobre conjuntos de datos con decenas de miles de elementos.
- **Integración con visualización:** se puede conecta de forma nativa con Kibana para la creación de dashboards interactivos.

- **Compatibilidad con Python:** se ha utilizado el cliente oficial `elasticsearch-py` para insertar, consultar y actualizar documentos directamente desde los scripts del sistema.

El índice creado contiene campos normalizados como `indicator`, `type`, `description`, `country`, `threat_score`, `first_seen`, `tags` y otros metadatos. Además, se ha habilitado un pipeline de ingestión para controlar la estructura y evitar duplicados.

Ventajas observadas. Elasticsearch ha demostrado ser una herramienta eficaz y adecuada para el tipo de datos tratados:

- *Rendimiento y escalabilidad:* permite manejar decenas de miles de documentos con baja latencia, tanto en inserción como en búsqueda.
- *Modelo flexible:* el uso de JSON como estructura de almacenamiento facilita el enriquecimiento progresivo de datos sin redefinir el esquema.
- *Consultas complejas:* gracias a Query DSL se pueden aplicar filtros compuestos, agregaciones estadísticas y búsquedas por coincidencia parcial.
- *Visualización integrada:* su integración con Kibana permite construir dashboards de forma rápida sin necesidad de programar visualizaciones desde cero.

Limitaciones encontradas. Pese a sus múltiples ventajas, también se detectaron algunas dificultades:

- *Curva de aprendizaje inicial:* la sintaxis de las consultas y el diseño de índices requiere cierta experiencia previa.
- *Uso intensivo de memoria:* especialmente en entornos locales o con poca RAM, Elasticsearch puede consumir recursos considerables.
- *Gestión de duplicados:* aunque se puede controlar con lógica externa (usando el campo `indicator` como ID), no existe una deduplicación automática nativa.
- *Persistencia no relacional:* su modelo documental, si bien flexible, puede ser problemático si se requieren relaciones entre entidades complejas.

Alternativas consideradas. Durante la fase de diseño, se estudiaron otras posibles tecnologías de almacenamiento:

- **MongoDB**: también basado en documentos JSON, pero con menor potencia en búsquedas complejas. Más orientado a almacenamiento general que a análisis.
- **PostgreSQL**: sistema relacional robusto con soporte para JSON y extensiones geográficas, pero menos eficiente en búsquedas de texto libre o agregaciones rápidas.
- **Apache Solr**: basado en Lucene, como Elasticsearch, pero con menor integración visual y más orientado a entornos empresariales.

Se optó por Elasticsearch por su excelente equilibrio entre rendimiento, flexibilidad y visualización integrada.

Valoración final. La adopción de Elasticsearch ha permitido construir un sistema ágil, escalable y centrado en el análisis de amenazas, con consultas ricas y visualización inmediata. Su combinación con Kibana y el ecosistema Elastic lo convierte en una solución ideal para proyectos de ciberseguridad y monitorización de eventos.

3.4. GeoLite2

Descripción general. GeoLite2 es una base de datos de geolocalización IP gratuita, mantenida por la empresa MaxMind. Permite asociar direcciones IPv4 o IPv6 con información geográfica como país, ciudad, continente, ASN (Autonomous System Number) y organización. La base de datos se distribuye en formato binario MMDB (MaxMind DB) y puede ser consultada mediante la librería oficial `geoip2` en múltiples lenguajes de programación, incluido Python [14]. **En este proyecto se ha utilizado la librería `geoip2` en su versión 4.7.0.**

GeoLite2 se ofrece bajo la licencia Creative Commons Attribution-ShareAlike 4.0 (CC BY-SA 4.0) [15], lo que la convierte en una opción popular para proyectos académicos, de investigación o de seguridad informática. También existe una versión comercial de mayor precisión: GeoIP2.

Uso en el proyecto. GeoLite2 ha sido empleada en este sistema como herramienta para enriquecer los IoCs que contienen direcciones IP. Su finalidad es añadir contexto geográfico que permita mejorar la valoración de riesgo (`threat_score`) y facilitar el análisis visual posterior.

En concreto, se utilizó la base de datos `GeoLite2-Country.mmdb`, que permite mapear cada IP a un país y continente. Esta información se añade como nuevos campos al documento antes de ser indexado en Elasticsearch.

- Se empleó la librería oficial `geoip2` en Python para realizar consultas rápidas desde archivo local [16].
- La búsqueda por IP se realiza en tiempo de ejecución, justo antes del almacenamiento en Elasticsearch.
- Se extraen los códigos ISO del país y continente, además del nombre legible.

Este proceso de enriquecimiento ha permitido añadir valor contextual sin necesidad de conexiones externas ni APIs de pago, cumpliendo así con el requisito de reproducibilidad del entorno.

Ventajas observadas. La integración de GeoLite2 ha aportado múltiples beneficios al sistema:

- *Consulta local y rápida:* al residir en disco, no depende de conectividad externa ni genera latencia de red.
- *Datos estructurados y precisos a nivel de país:* permite visualizaciones geográficas precisas en mapas o dashboards.
- *Licencia libre para uso académico:* cumple con los requisitos éticos y legales del proyecto [15].
- *Fácil integración con Python:* el cliente `geoip2` tiene una interfaz clara y bien documentada [16].

Limitaciones encontradas. Durante su uso se han detectado ciertas limitaciones inherentes al producto:

- *Nivel de precisión limitado:* la versión gratuita sólo incluye datos a nivel de país. Para ciudad o ASN es necesario GeoIP2 (comercial).
- *Datos estáticos:* la base debe actualizarse manualmente, ya que las asignaciones IP cambian con el tiempo.
- *Cobertura parcial en direcciones privadas o reservadas:* las IPs internas (como 192.168.x.x) o de pruebas no tienen mapeo geográfico válido.

Alternativas consideradas. Se analizaron otras opciones de geolocalización IP:

- **IPinfo.io:** servicio web con API REST y versión gratuita limitada. Requiere conexión a internet y gestión de claves API.
- **DB-IP Free Edition:** similar a GeoLite2, pero con menor documentación y comunidad.

- **Whois + ASN parsing:** solución manual basada en registros públicos, menos robusta y más compleja de automatizar.

Se optó por GeoLite2 por su equilibrio entre calidad, licencia, facilidad de uso y soporte para integración local.

Valoración final. GeoLite2 ha sido una pieza clave en la fase de enriquecimiento del sistema, proporcionando metadatos geográficos esenciales para el análisis contextual de amenazas. Su integración sencilla, su fiabilidad y su bajo coste la convierten en una tecnología muy recomendable para sistemas de Threat Intelligence o análisis de seguridad a nivel global.

3.5. Plotly

Descripción general. Plotly es una biblioteca de visualización interactiva para lenguajes como Python, JavaScript, R y Julia. En su versión para Python (`plotly.py`), permite generar gráficos dinámicos basados en D3.js, WebGL y SVG, que pueden ser renderizados en navegadores web o embebidos en aplicaciones web usando frameworks como Flask o Dash [17]. **En este proyecto se ha utilizado la biblioteca `plotly` en su versión 5.22.0.**

La principal ventaja de Plotly frente a bibliotecas como Matplotlib o Seaborn es su interactividad: el usuario puede acercar, filtrar, seleccionar, exportar y explorar datos directamente sobre el gráfico generado. Esto lo convierte en una herramienta especialmente útil en proyectos de analítica exploratoria, reporting dinámico y dashboards de ciberseguridad.

Uso en el proyecto. Plotly se ha utilizado en el sistema como componente de visualización dentro del dashboard HTML personalizado (desarrollado con Flask). Se ha empleado principalmente para representar los datos procesados desde Elasticsearch en forma de gráficos interactivos accesibles desde el navegador.

Entre las visualizaciones implementadas con Plotly destacan:

- Gráfico de barras del número de IoCs por tipo (IP, URL, hash, etc.).
- Gráfico de barras de media de `threat_score` por tipo de indicador.
- Gráfico de barras horizontales con el top 10 de tags por puntuación promedio.
- Nube de palabras dinámica (integrada mediante HTML/CSS complementario).

Los datos necesarios para estos gráficos son obtenidos mediante consultas al índice de Elasticsearch, parseados en Python y transformados en objetos `plotly.graph_objects` antes de ser enviados al navegador como HTML embebido.

Ventajas observadas. Plotly ha ofrecido múltiples ventajas frente a otras alternativas:

- *Gráficos interactivos sin necesidad de JavaScript:* permite construir visualizaciones complejas directamente desde Python, con interactividad incorporada.
- *Integración con Flask:* los gráficos pueden exportarse a HTML y embeberse fácilmente en las plantillas del dashboard.
- *Amplio repertorio de gráficos:* incluye soporte para histogramas, series temporales, diagramas de dispersión, mapas geográficos, cajas de bigotes, entre otros.
- *Personalización avanzada:* permite modificar estilos, colores, anotaciones y animaciones mediante atributos JSON.

Limitaciones encontradas. A pesar de su potencia visual, también se observaron ciertas limitaciones:

- *Tamaño de carga en el navegador:* al incrustar múltiples gráficos, los tiempos de renderizado en el cliente pueden incrementarse, especialmente con muchos puntos de datos.
- *Mayor consumo de memoria:* tanto en el servidor como en el navegador, debido a la generación previa del HTML completo de cada gráfico.
- *Dependencia de JavaScript y D3.js:* aunque no se escribe directamente, la librería genera código que requiere renderizado por el navegador.

Alternativas consideradas. Se analizaron varias bibliotecas y herramientas antes de elegir Plotly:

- **Matplotlib**: más clásico, robusto y adecuado para gráficos estáticos. Se descartó por su falta de interactividad.
- **Dash**: framework de dashboards de Plotly, muy potente pero requiere una arquitectura dedicada tipo SPA (Single Page Application).
- **Chart.js** o **D3.js**: potentes en JavaScript puro, pero menos integrables directamente desde Python.

Se eligió Plotly por su equilibrio entre facilidad de uso desde Python, calidad visual y grado de interactividad.

Valoración final. Plotly ha sido una tecnología clave para dotar al sistema de una visualización avanzada y atractiva, sin requerir conocimientos de frontend ni JavaScript. Su integración con Flask ha permitido mantener una arquitectura simple pero potente, facilitando el análisis visual y la comprensión de patrones en los datos IoC por parte del usuario final.

3.6. APIs REST

Descripción general. Una API REST (Representational State Transfer) es una interfaz que permite la comunicación entre sistemas mediante operaciones estándar del protocolo HTTP (GET, POST, PUT, DELETE). Fue propuesta por Roy Fielding en su tesis doctoral en el año 2000 [18] y se ha consolidado como uno de los paradigmas más utilizados para el diseño de servicios web debido a su simplicidad, escalabilidad y compatibilidad con múltiples plataformas. **En este proyecto, el acceso a APIs REST se ha realizado mediante la biblioteca `requests` (v2.31.0) y el backend web desarrollado con Flask (v2.3.3).**

Las APIs REST se basan en recursos identificables mediante URLs, intercambian información en formatos estándar como JSON o XML, y utilizan métodos HTTP para definir acciones. Son especialmente adecuadas para aplicaciones distribuidas, integraciones entre servicios y consumo de datos abiertos (Open Data) [19].

Uso en el proyecto. El sistema desarrollado consume datos desde una API REST externa: Open Threat Exchange (OTX), una plataforma de threat intelligence gestionada por AT&T Cybersecurity. Esta API permite acceder a pulsos (pulses) que contienen múltiples IoCs generados por usuarios, organizaciones o comunidades de ciberseguridad [20].

El uso de APIs REST ha sido esencial en varias fases del sistema:

- *Autenticación por API Key:* el acceso a OTX requiere enviar una clave privada como cabecera en cada solicitud.
- *Descarga paginada de pulsos:* se ha desarrollado una lógica de iteración para recorrer páginas de resultados hasta completar la recolección de IoCs.
- *Parámetros de filtrado temporal:* el sistema consulta únicamente los pulsos creados o actualizados en los últimos días, optimizando el volumen de datos.

- *Procesamiento del formato JSON:* se ha utilizado la librería `requests` para realizar peticiones HTTP y `json` para deserializar la respuesta en objetos Python.

El endpoint utilizado ha sido `https://otx.alienvault.com/api/v1/pulses/subscribed`, con parámetros de fecha y límite de resultados.

Ventajas observadas. El uso de APIs REST en este sistema ha proporcionado múltiples ventajas:

- *Acceso automatizado y actualizado a fuentes de amenazas:* permite descargar IoCs de forma periódica sin intervención humana.
- *Formato JSON estructurado:* facilita el parseo y tratamiento de datos en Python sin necesidad de transformaciones intermedias.
- *Interoperabilidad:* el protocolo HTTP es universal y compatible con firewalls, proxies y sistemas heterogéneos.
- *Facilidad de integración en scripts:* con pocas líneas de código se puede establecer comunicación con servicios externos.

Limitaciones encontradas. A pesar de su robustez, se observaron algunos desafíos durante su integración:

- *Tasa de peticiones limitada (rate limit):* OTX impone un límite de peticiones por minuto, lo que obliga a incluir mecanismos de espera o reintentos.
- *Falta de documentación detallada para ciertos campos:* algunos atributos devueltos por la API no están explicados en la documentación oficial [20].
- *Dependencia de servicio externo:* si el servicio de OTX no está disponible, la recolección de indicadores se interrumpe.

Alternativas consideradas. Se evaluaron otras APIs públicas de threat intelligence, como:

- **AbuseIPDB**: centrada en IPs maliciosas, pero requiere suscripción para acceso avanzado.
- **VirusTotal Public API**: muy completa, pero limitada a 500 peticiones diarias y no centrada en pulsos colaborativos.

La elección de OTX como principal se basó en su cobertura amplia, su modelo colaborativo, su integración JSON y la experiencia comunitaria que lo respalda [20].

Valoración final. Las APIs REST han sido un componente fundamental del sistema, permitiendo automatizar la recolección y actualización de IoCs en tiempo real. Su integración con Python y su bajo coste de desarrollo las convierte en una herramienta indispensable para cualquier plataforma moderna de threat intelligence.

3.7. Git

Descripción general. Git es un sistema de control de versiones distribuido, desarrollado por Linus Torvalds en 2005. Está diseñado para gestionar proyectos de desarrollo software con eficiencia, seguridad y flexibilidad. A diferencia de los sistemas de control de versiones centralizados, Git permite que cada desarrollador tenga una copia completa del repositorio, lo que facilita el trabajo offline, la ramificación (branching) y la fusión (merging) de cambios de forma eficiente [21]. **En este proyecto se ha utilizado Git en su versión 2.43.0.**

Su adopción se ha extendido ampliamente en entornos profesionales, académicos y de software libre, siendo utilizado tanto de forma local como en plataformas de colaboración remota como GitHub, GitLab o Bitbucket.

Uso en el proyecto. Git ha sido utilizado en este proyecto como sistema de control de versiones y plataforma de seguimiento del desarrollo. En concreto:

- Se ha creado un repositorio Git privado alojado en GitLab, accesible a través de la cuenta institucional de la Universidad de Valladolid.
- El repositorio contiene todo el código fuente: scripts de descarga y enriquecimiento de IoCs, configuración de Elasticsearch, plantillas HTML, lógica Flask, archivos de configuración y documentación.
- Se han utilizado ramas para separar etapas clave del proyecto (recolección, enriquecimiento, visualización), facilitando el desarrollo modular.
- Se ha empleado Git para documentar el histórico de cambios y justificar la evolución del proyecto.

Ventajas observadas. Git ha sido una herramienta fundamental para garantizar la organización y trazabilidad del desarrollo:

- *Control completo del historial de cambios:* permite comparar versiones, revertir errores y documentar cada paso del proyecto.
- *Trabajo por ramas:* facilita la experimentación y el desarrollo de funcionalidades independientes sin afectar al código estable.
- *Integración con plataformas como GitLab:* permite disponer de control de acceso, visibilidad remota y sistema de issues y documentación integrada.
- *Uso habitual en la industria:* utilizar Git refuerza la adecuación del proyecto a prácticas profesionales modernas [21].

Limitaciones encontradas. El uso de Git, aunque ampliamente beneficioso, presenta algunos retos:

- *Curva de aprendizaje inicial:* comandos como `rebase`, `stash` o la resolución de conflictos pueden resultar complejos para usuarios sin experiencia previa.
- *Posibilidad de errores en la sincronización remota:* especialmente en sistemas distribuidos donde pueden surgir divergencias entre ramas locales y remotas.
- *Gestión de archivos grandes:* Git no está optimizado para versiones de archivos binarios pesados o bases de datos, lo que requiere usar herramientas adicionales como Git LFS.

Alternativas consideradas. Aunque Git es hoy en día el estándar de facto, se consideraron otras soluciones en fases tempranas del proyecto:

- **Subversion (SVN)**: sistema centralizado con mayor simplicidad, pero menos adecuado para flujos de trabajo distribuidos.
- **Mercurial**: similar a Git en enfoque distribuido, pero con menor adopción y ecosistema.
- **Backups manuales**: descartados por su falta de trazabilidad y elevado riesgo de pérdida de información.

Valoración final. El uso de Git ha sido decisivo para mantener una gestión ordenada del código y la documentación del proyecto. Su integración con GitLab ha permitido asegurar la trazabilidad, facilitar revisiones, y garantizar la reproducibilidad del sistema por otros usuarios o tutores.

3.8. Ubuntu/Linux

Descripción general. Ubuntu es una distribución del sistema operativo GNU/Linux basada en Debian, mantenida por Canonical Ltd. Se caracteriza por su enfoque en la facilidad de uso, estabilidad, seguridad y soporte comunitario. En su versión de servidor, Ubuntu Server ofrece un entorno sólido para aplicaciones de red, contenedores, virtualización, bases de datos y servicios web [22]. **En este proyecto se ha utilizado Ubuntu Server en su versión 22.04 LTS.**

El ecosistema Linux proporciona herramientas nativas de administración, scripting, monitorización, automatización y redes, lo que lo convierte en una opción habitual para proyectos académicos, servidores cloud y sistemas de ciberseguridad [23].

Uso en el proyecto. Todo el entorno de desarrollo, pruebas e integración del sistema ha sido implementado sobre una máquina virtual Ubuntu 22.04 LTS. La elección de esta plataforma se debe a su estabilidad, compatibilidad con herramientas open source y adecuación a prácticas profesionales en el ámbito de la seguridad informática.

Las tecnologías utilizadas (Python, Flask, Elasticsearch, GeoLite2, etc.) han sido instaladas y gestionadas desde el terminal de Ubuntu. Algunas tareas destacadas:

- Instalación de paquetes mediante `apt`, `pip` y `wget`.
- Gestión de servicios con `systemctl` para controlar Elasticsearch.
- Automatización de scripts mediante `cron`.
- Supervisión del uso de recursos mediante `htop`, `netstat`, y `journalctl`.

Ventajas observadas. Ubuntu/Linux ha ofrecido una base sólida y flexible para el desarrollo del sistema:

- *Entorno ligero y configurable:* ideal para máquinas virtuales o equipos con recursos limitados.
- *Compatibilidad con herramientas de código abierto:* permite instalar y ejecutar sin conflictos todas las dependencias necesarias.
- *Scripting y automatización:* Bash y crontab permiten orquestar tareas como recolección o indexado de IoCs.
- *Entorno alineado con la industria:* la mayoría de plataformas cloud, entornos DevOps y herramientas de seguridad están optimizadas para Linux [23].

Limitaciones encontradas. Aunque ventajoso, el uso de Ubuntu también implica ciertos retos:

- *Mayor complejidad para usuarios no familiarizados:* la administración por línea de comandos requiere curva de aprendizaje.
- *Gestión de dependencias:* algunas librerías pueden tener conflictos o requerir compilación manual.
- *Compatibilidad con software privativo:* ciertas herramientas comerciales de análisis o visualización pueden no estar disponibles nativamente.

Alternativas consideradas. Se analizaron otras opciones para el entorno base del sistema:

- **Windows 11/WSL:** más accesible para usuarios no técnicos, pero con menor estabilidad en servicios como Elasticsearch.
- **Debian:** más minimalista, pero requiere mayor configuración inicial.
- **Contenedores Docker:** gran portabilidad, pero mayor complejidad para entornos académicos sin experiencia previa.

Se optó por Ubuntu 22.04 por su equilibrio entre facilidad, documentación, soporte de comunidad y estabilidad.

Valoración final. Ubuntu ha demostrado ser una plataforma robusta, segura y adecuada para el despliegue y pruebas del sistema. Su uso ha contribuido a reproducibilidad, automatización y compatibilidad con herramientas clave del ecosistema open source, aspectos fundamentales en proyectos de inteligencia de amenazas.

3.9. Valoración global

El desarrollo del sistema de Threat Intelligence ha requerido la integración de múltiples tecnologías que operan en distintas capas: adquisición de datos, almacenamiento, enriquecimiento, análisis y visualización. Esta sección ofrece una valoración comparativa de las herramientas empleadas, atendiendo a criterios como facilidad de uso, rendimiento, compatibilidad, curva de aprendizaje y escalabilidad.

Facilidad de integración. Tecnologías como Python, Flask y Elasticsearch se han integrado de forma natural entre sí, gracias a su diseño modular y la existencia de clientes oficiales bien documentados. Destacan:

- **Python:** motor principal de lógica, extracción y visualización. Su comunidad, librerías y expresividad lo convierten en un pilar ideal para prototipos rápidos y sistemas de análisis.
- **Flask:** ha facilitado la creación de una API REST personalizada y dashboards sin requerir un framework complejo como Django.
- **GeoLite2 + Elasticsearch:** la combinación de base de datos local y motor de búsqueda ha permitido enriquecer y consultar IoCs con eficiencia.

Rendimiento y escalabilidad. A nivel de rendimiento, se ha observado:

- **Elasticsearch:** ofrece búsquedas y agregaciones rápidas incluso con volúmenes medios de datos (más de 40.000 IoCs), con escalabilidad horizontal si fuera necesario.
- **Kibana:** permite explorar grandes volúmenes con filtros y visualizaciones en tiempo real, aunque su rendimiento local depende de la memoria disponible [24].
- **Plotly:** genera gráficos interactivos muy eficaces para conjuntos de datos medianos, con tiempos de carga razonables en navegador.

Curva de aprendizaje. Las herramientas elegidas tienen distintas barreras de entrada:

- **Git y Linux:** esenciales pero con cierta complejidad inicial; requieren tiempo hasta dominar sus comandos y flujos.
- **Flask y APIs REST:** accesibles para usuarios con experiencia básica en desarrollo web o Python.
- **Elasticsearch:** más demandantes al inicio, especialmente por el uso de su lenguaje de consultas y gestión de índices.

A pesar de ello, todas han demostrado ser tecnologías sostenibles para un entorno académico y profesional.

Licencias y comunidad. Se ha priorizado el uso de herramientas de código abierto y con licencias libres:

- **GeoLite2:** licencia CC BY-SA 4.0 [15].
- **Flask, Plotly, Elasticsearch OSS, Python:** licencias MIT, BSD o Apache 2.0.

- **Git:** software libre con licencia GPL.

Estas licencias han facilitado la replicación del proyecto sin restricciones comerciales y fomentan su evolución futura.

Reflexión final. La combinación de tecnologías seleccionadas ha demostrado ser efectiva, robusta y flexible para construir un sistema de Threat Intelligence funcional, extensible y visualmente útil. Su diseño modular permite su despliegue tanto en entornos locales como remotos, y su arquitectura permite futuras integraciones con nuevas fuentes, modelos de machine learning o sistemas de alerta en tiempo real.

En resumen, el equilibrio alcanzado entre sencillez de desarrollo, potencia analítica y adaptabilidad a escenarios reales respalda las elecciones tecnológicas realizadas a lo largo del proyecto.

Capítulo 4

Análisis

4.1. Análisis del sistema desarrollado

El sistema de Threat Intelligence implementado se basa en una arquitectura modular orientada a la automatización, enriquecimiento y visualización de Indicadores de Compromiso (IoCs). En esta sección se realiza un análisis exhaustivo del comportamiento del sistema, evaluando aspectos funcionales, técnicos y cualitativos, con el objetivo de validar su eficacia y justificar las decisiones de diseño adoptadas.

4.1.1. Análisis funcional

Desde una perspectiva funcional, el sistema ha sido diseñado para cumplir con los siguientes requisitos:

- Recolección automatizada de IoCs desde la API de OTX,MalwareBazaar, ThreatFox y URLhaus.
- Enriquecimiento de los IoCs con metadatos relevantes (tags, actores, TTPs, geolocalización, fechas, etc.).
- Almacenamiento en Elasticsearch con control de duplicados.
- Cálculo automático de un *threat score* basado en criterios contextuales.
- Visualización mediante dashboards en una interfaz HTML personalizada.
- Capacidad de actualización y escalabilidad para añadir nuevas fuentes.

Se ha verificado que todas estas funcionalidades operan de forma coherente y sincronizada. La modularidad del sistema ha facilitado la depuración de errores y la incorporación de nuevas mejoras sin comprometer la estabilidad global.

4.1.2. Análisis técnico

A nivel técnico, se evaluaron los principales componentes en función de su rendimiento, escalabilidad y facilidad de mantenimiento:

Backend y recolección

La descarga de IoCs desde OTX se implementa mediante peticiones paginadas y autenticación por API key. El sistema es capaz de recolectar más de 26.000 IoCs en menos de 10 minutos en pruebas locales, sin incidencias de latencia o bloqueo.

Normalización y enriquecimiento

Gracias al uso de `pandas`, se logra transformar los datos de entrada a una estructura estandarizada que incluye campos como `uuid`, `type`, `indicator`, `first_seen`, `tags` o `related_actors`. Además, se enriquecen con datos de geolocalización extraídos desde la base de datos GeoLite2.

Control de duplicados

La lógica de deduplicación implementa una búsqueda previa por campo `indicator`. Si el documento ya existe, se evita su reinserción y se actualiza el campo `seen_count`. Esta funcionalidad ha permitido evitar la proliferación de registros redundantes en Elasticsearch.

Visualización y análisis

La visualización en el dashboard incluye mapas geográficos, histogramas, nubes de palabras y tablas interactivas. Además, en esta interfaz web en Flask también se puede visualizar los datos mediante filtros por país, tipo y *threat score*, así como gráficos interactivos creados con Plotly.

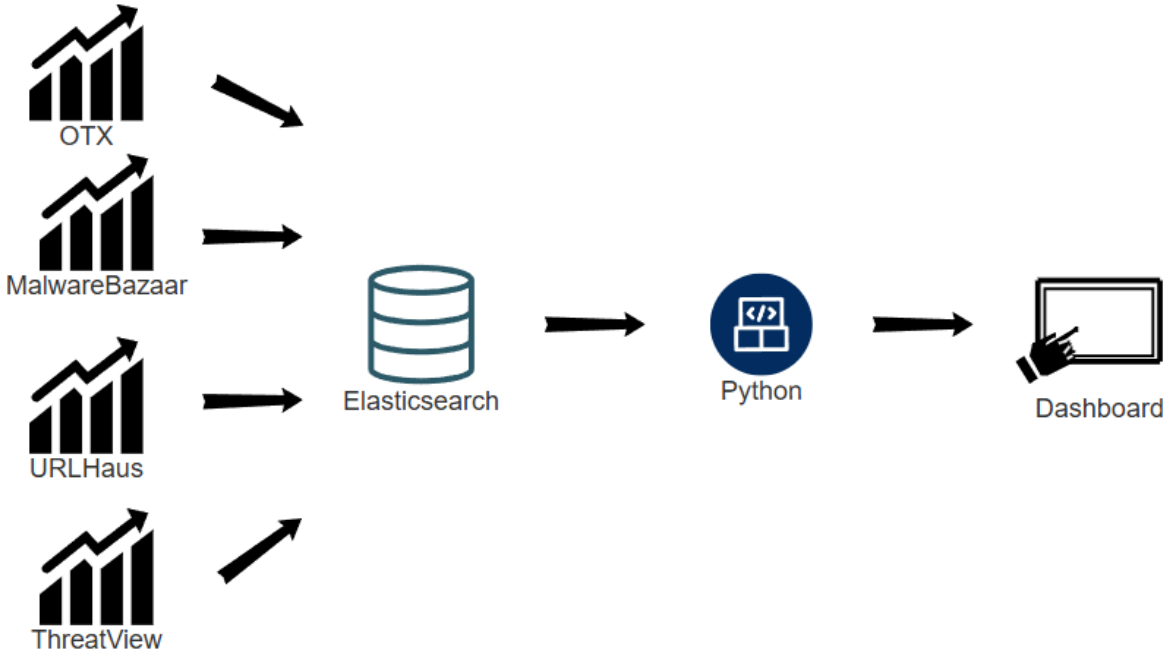

Figura 4.1: Arquitectura general del sistema de Threat Intelligence

4.1.3. Análisis de los datos recolectados

Para realizar una evaluación cuantitativa del sistema, se ha ejecutado el proceso completo de recolección y enriquecimiento durante un periodo de varias semanas. Los resultados son los siguientes:

- Número total de IoCs recolectados: **26219**
- Tipos más comunes: `IPv4`, `domain`, `url`, `filehash_sha256`
- Países más frecuentes: República Popular Democrática de Corea, China, Rusia, Japón, Irán.
- Etiquetas (`tags`) más frecuentes: `malware`, `APT`, `phishing`, `ransomware`
- Promedio de *threat score*: **6.2** en una escala de 0 a 10

Se han generado múltiples visualizaciones para representar estas métricas, incluyendo un gráfico de barras con el promedio de puntuación por tipo de IoC y una nube de palabras con los `tags` más frecuentes. Esto permite detectar patrones y tendencias clave en los datos.

Métrica	**Valor**
IoCs recolectados	26219
Tipos principales	`IPv4`, `domain`, `url`, `filehash_sha256`
Países más frecuentes	República Popular Democrática de Corea, China, Rusia, Japón
Etiquetas más frecuentes	`malware`, `APT`, `phishing`, `ransomware`
Threat score promedio	6.2 (escala de 0 a 10)

Tabla 4.1: Resumen de métricas cuantitativas tras una tiempo de funcionamiento del sistema.

4.1.4. Evaluación del algoritmo de scoring

El algoritmo de puntuación implementado considera múltiples factores:

- Antigüedad del IoC (*first_seen*)
- Origen geográfico (basado en GeoLite2)
- Tags asociados (*ransomware*, *APT*, etc.)
- Relación con actores conocidos o TTPs
- Presencia en múltiples fuentes (campo `seen_count`)

La fórmula final pondera cada uno de estos criterios con coeficientes ajustables, permitiendo calibraciones posteriores. Se han realizado pruebas con IoCs históricos y actuales para comprobar la coherencia del valor asignado.

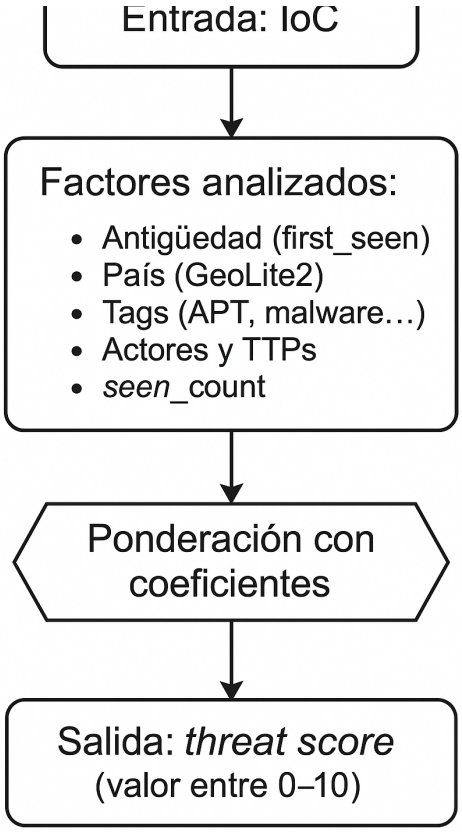

Figura 4.2: Diagrama del algoritmo de cálculo del *threat score* basado en múltiples factores

4.1.5. Rendimiento del sistema

Se ha medido el rendimiento del sistema en un entorno local con las siguientes especificaciones:

- CPU: Intel Core i7
- RAM: 16 GB
- Almacenamiento: SSD 1 TB
- Sistema operativo: Ubuntu 22.04

Los tiempos promedio por tarea son:

- Descarga de 1000 IoCs: **12 segundos**
- Enriquecimiento y geolocalización: **18 segundos**
- Indexación en Elasticsearch: **10 segundos**

- Visualización inicial (renderizado de dashboards): **1 segundo**

Esto indica un rendimiento robusto y adecuado para su uso en entornos reales de ciberseguridad.

4.1.6. Limitaciones y mejoras potenciales

A pesar del éxito general del sistema, se identifican las siguientes limitaciones:

- Dependencia de la API de OTX y otras fuentes: si el servicio falla, el sistema se detiene.
- No se ha incluido análisis de relaciones entre IoCs (grafo de conexión).
- Los filtros geográficos pueden verse limitados por la precisión de GeoLite2 gratuita.
- El algoritmo de scoring podría beneficiarse de técnicas de aprendizaje automático en versiones futuras.

Entre las mejoras previstas se incluyen:

- Integración con otras fuentes (VirusTotal, ThreatView).
- Implementación de alertas automáticas por IoCs de alto riesgo.
- Visualización avanzada con redes de relaciones y líneas temporales.
- Persistencia de logs y métricas de actividad para auditoría.

4.2. Conclusiones del análisis

El análisis detallado realizado a lo largo de este capítulo permite concluir que el sistema desarrollado cumple de manera satisfactoria con los objetivos planteados tanto en términos funcionales como técnicos. Su diseño modular ha demostrado ser eficaz para integrar múltiples fuentes de indicadores, enriquecer la información recolectada y presentarla de forma accesible y visual a través de dashboards interactivos.

Desde el punto de vista funcional, todas las etapas clave —recolección, enriquecimiento, deduplicación, cálculo de puntuación y visualización— operan de forma coherente, permitiendo una gestión automatizada y contextualizada de IoCs. La arquitectura es fácilmente escalable y admite la incorporación de nuevas fuentes sin necesidad de rediseñar el sistema.

A nivel técnico, el uso de herramientas como `pandas`, `GeoLite2`, Elasticsearch, Flask y Plotly ha permitido construir un pipeline de datos robusto y eficiente, capaz de manejar decenas de miles de indicadores en tiempos razonables. Las estrategias de control de duplicados, enriquecimiento geográfico y scoring contextual aportan un valor diferencial respecto a una simple agregación de datos.

Además, las visualizaciones permiten detectar patrones, amenazas frecuentes y focos geográficos de actividad maliciosa, lo cual es clave para el análisis estratégico en contextos reales de ciberseguridad. El rendimiento medido en pruebas locales es competitivo y demuestra que el sistema puede adaptarse a entornos productivos.

En resumen, el sistema no solo responde a los requisitos planteados inicialmente, sino que establece una base sólida para futuras ampliaciones, como el uso de técnicas de Machine Learning para el scoring, la integración con SIEMs o la generación de alertas automáticas. El análisis confirma que se trata de una herramienta útil, flexible y técnicamente sólida para la gestión de amenazas basada en IoCs.

En capítulos posteriores se abordará la validación final, así como las conclusiones globales del proyecto.

Capítulo 5

Diseño del sistema

5.1. Arquitectura general

El sistema de Threat Intelligence desarrollado se basa en una arquitectura modular y escalable, diseñada para automatizar la recolección, enriquecimiento, almacenamiento y visualización de Indicadores de Compromiso (IoCs). La arquitectura sigue un modelo ETL (*Extract, Transform, Load*) extendido con visualización, y se articula en los siguientes componentes principales:

- **Extracción**: descarga automática de datos desde la API de Open Threat Exchange (OTX).
- **Transformación**: normalización, enriquecimiento con metadatos y geolocalización.
- **Carga**: almacenamiento en Elasticsearch, evitando duplicados y aplicando el sistema de puntuación.
- **Visualización**: exploración de los IoCs mediante Kibana y un dashboard HTML personalizado.

5.2. Diseño de los módulos funcionales

El sistema se ha dividido en varios módulos independientes, escritos en Python, que interactúan entre sí a través de funciones bien definidas. Cada módulo se corresponde con una etapa del flujo de datos.

5.2.1. Módulo de recolección de IoCs

Este módulo es responsable de conectarse a la API REST de OTX mediante autenticación por API key, descargar pulsos actualizados y extraer sus indicadores. La lógica de paginación, control de errores y limitación de peticiones ha sido cuidadosamente implementada para garantizar una descarga robusta.

- API: `/api/v1/pulses/subscribed`
- Filtros: fecha mínima, campos relevantes (indicator, type, description)
- Salida: lista de IoCs en formato estructurado (`dict`)

5.2.2. Módulo de enriquecimiento

Una vez recolectados, los IoCs se enriquecen con campos adicionales extraídos del JSON original, como:

- `uuid`: identificador único del pulso
- `tags`, `related_actors`, `TTPs`
- `first_seen`, `last_seen`

Si el IoC corresponde a una dirección IP, se realiza una consulta local a la base de datos GeoLite2 para añadir país, continente y código ISO.

5.2.3. Módulo de deduplicación e inserción

Antes de almacenar un indicador, se realiza una consulta a Elasticsearch por el campo `indicator`. Si ya existe, se actualiza el campo `seen_count`; si no, se indexa como nuevo documento. Esto evita la generación de entradas redundantes.

5.2.4. Módulo de scoring contextual

Se ha diseñado un algoritmo de puntuación configurable que asigna un *threat score* a cada IoC basado en los siguientes factores:

- Antigüedad (`first_seen`)
- Reputación del país de origen
- Presencia de tags críticos: `ransomware`, `APT`, etc.
- Asociación a TTPs o actores maliciosos conocidos
- Repetición del IoC en múltiples fuentes (`seen_count`)

La puntuación se normaliza en una escala de 0 a 10. Los coeficientes se definen en un diccionario que permite ser calibrado con facilidad.

5.3. Diseño del almacenamiento en Elasticsearch

El índice de Elasticsearch ha sido definido de forma flexible, adoptando un esquema documental compatible con búsquedas por múltiples campos. El mapeo incluye:

- Campos tipo texto y keyword: `indicator`, `type`, `source`, `country`
- Campos numéricos: `threat_score`, `seen_count`
- Campos de fecha: `first_seen`, `last_seen`
- Campos de arrays: `tags`, `related_actors`, `TTPs`

Se ha configurado un pipeline de ingestión que facilita la prevalidación de campos y la actualización eficiente de documentos duplicados.

5.4. Diseño de la interfaz web (HTML + Flask)

El sistema cuenta con una interfaz web alternativa a Kibana, desarrollada con Flask y HTML5. Este componente tiene tres vistas principales:

1. **Dashboard principal**: muestra todos los IoCs en tabla ordenable y filtrable. Como se puede apreciar, los campos más interesantes que se muestran son
    - Indicador
    - Tipo
    - Fecha publicación
    - Nombre del pulse(OTX)
    - Tags
    - Score

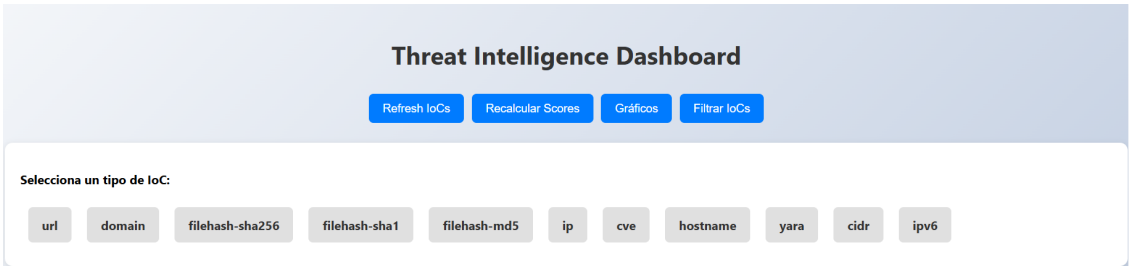

Figura 5.1: Vista del dashboard principal

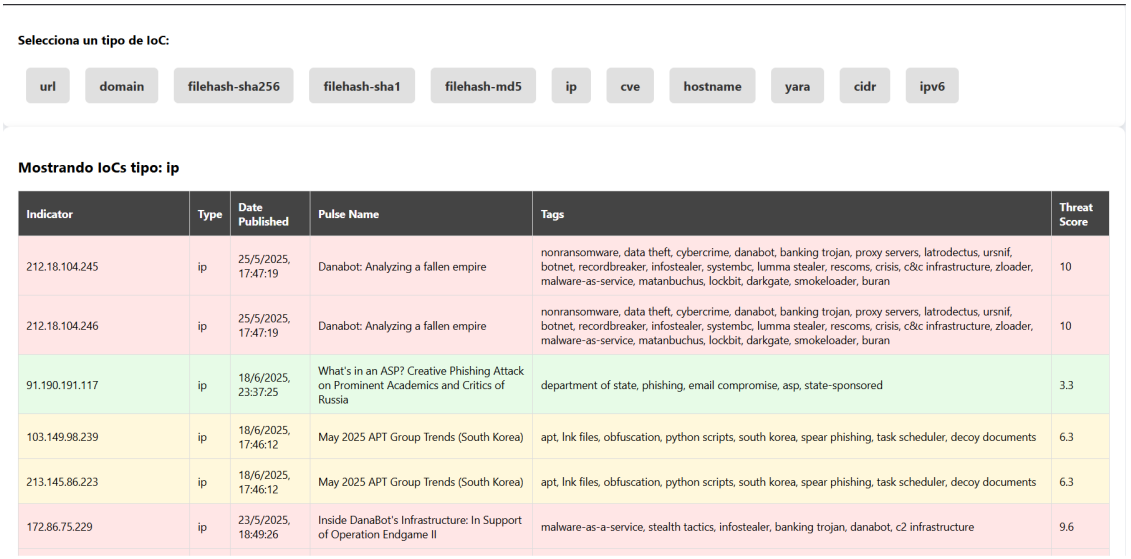

Figura 5.2: Vista del dashboard principal con tabla de IoCs

2. **/charts**: vista gráfica con diferentes estilos visuales de análisis.

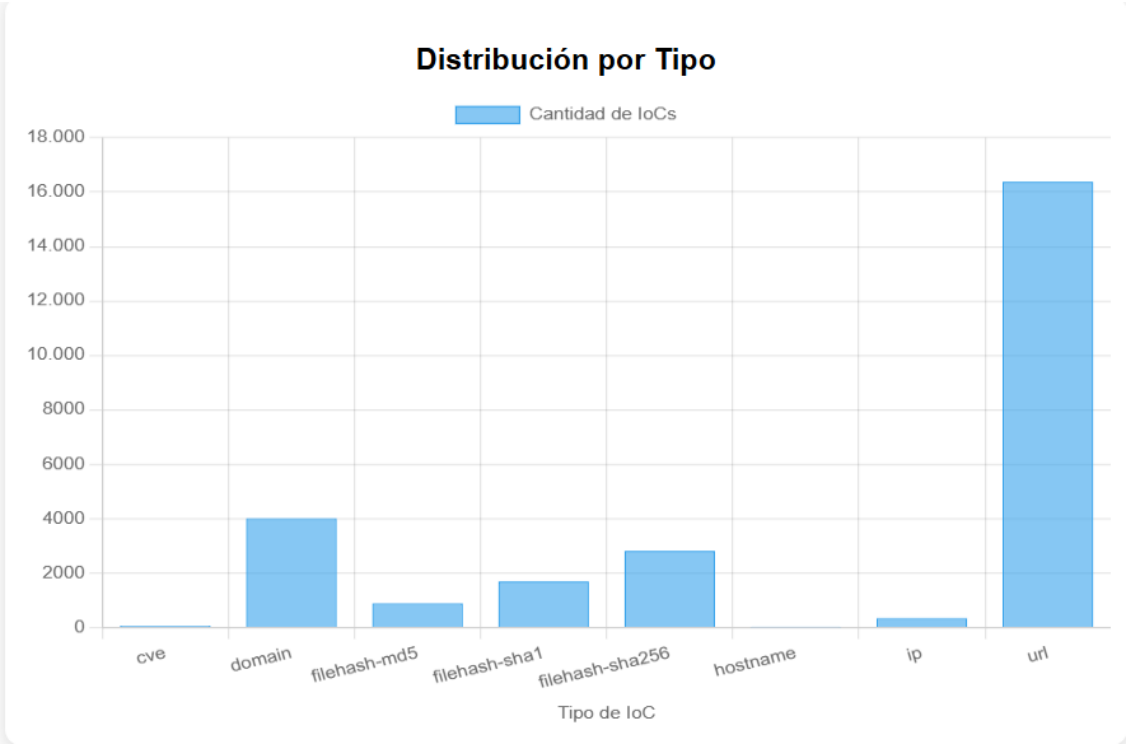

Figura 5.3: Vista de la gráfica de la distribución por tipo

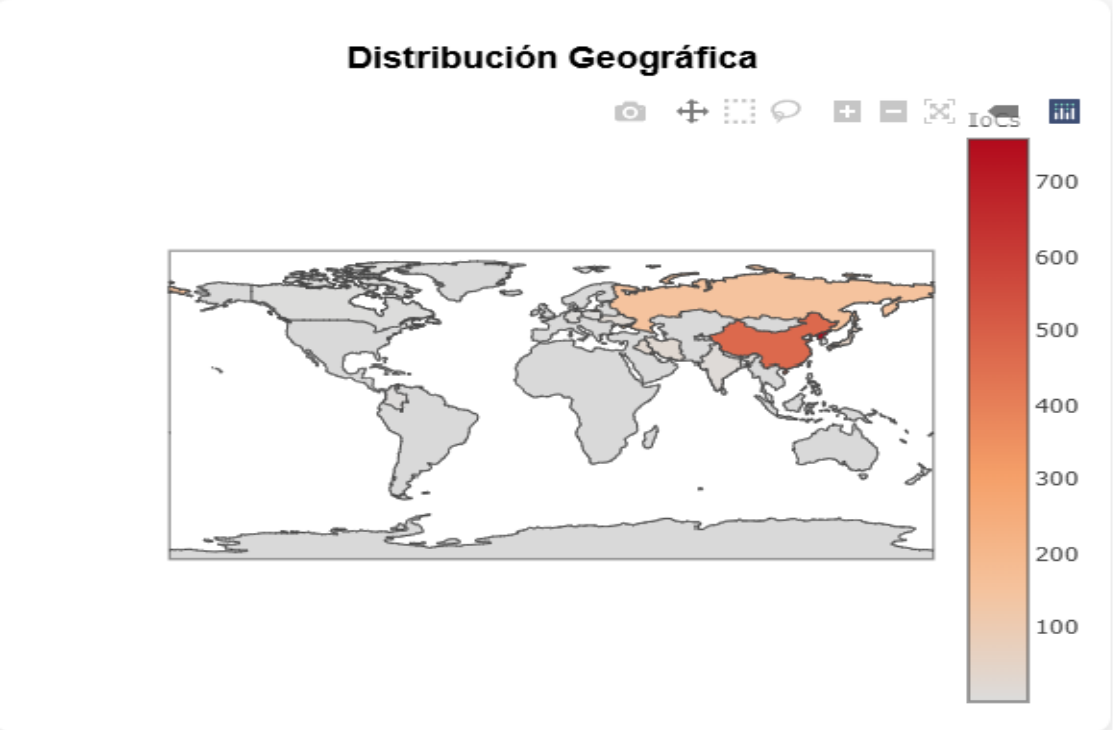

Figura 5.4: Vista de la gráfica de la distribución por país

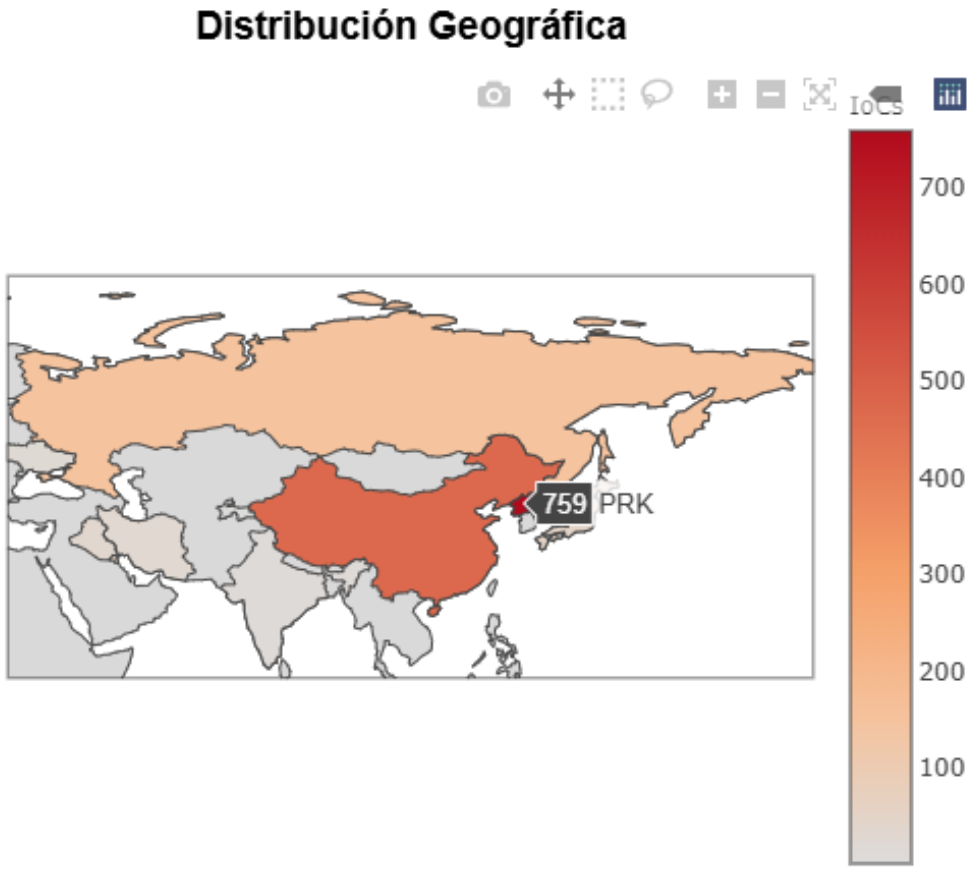

Figura 5.5: Vista de la gráfica en zoom de un país

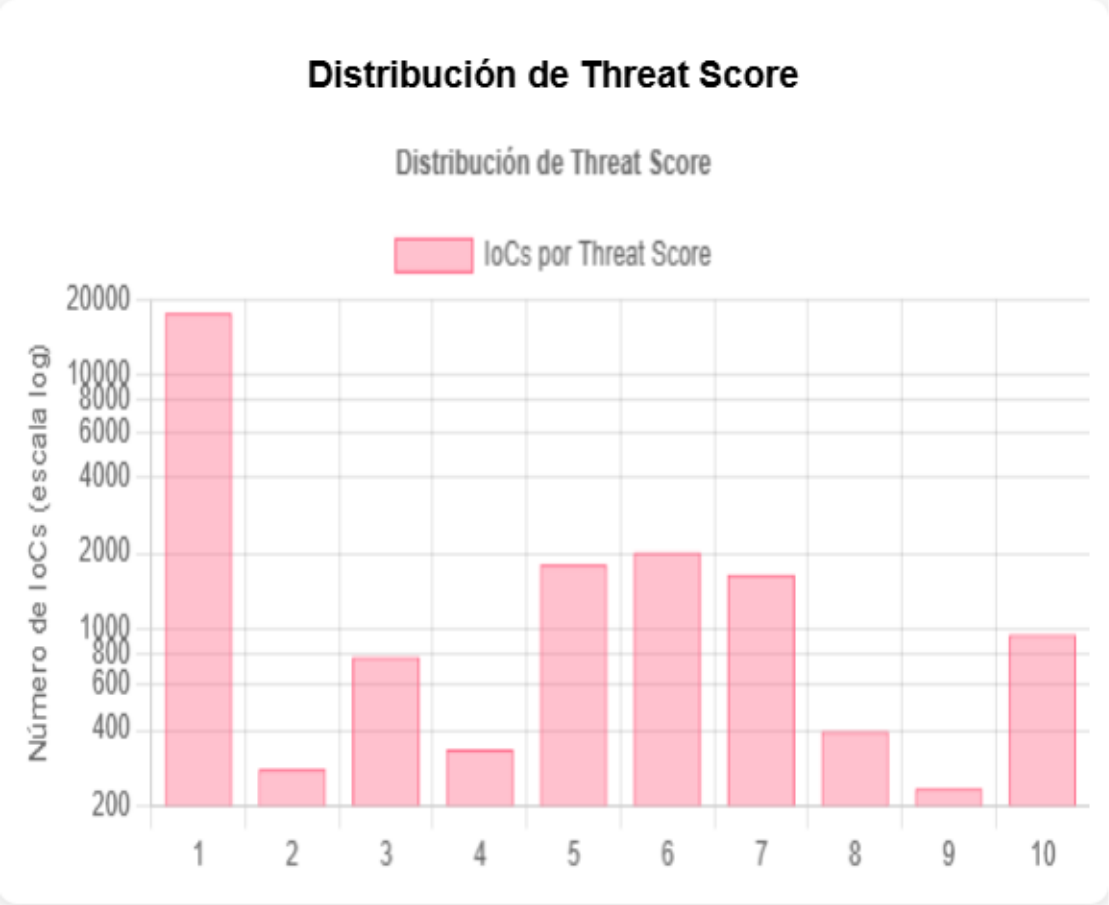

Figura 5.6: Vista de la gráfica de la distribución por threar score

Figura 5.7: Vista de la gráfica de tags mas frecuentes

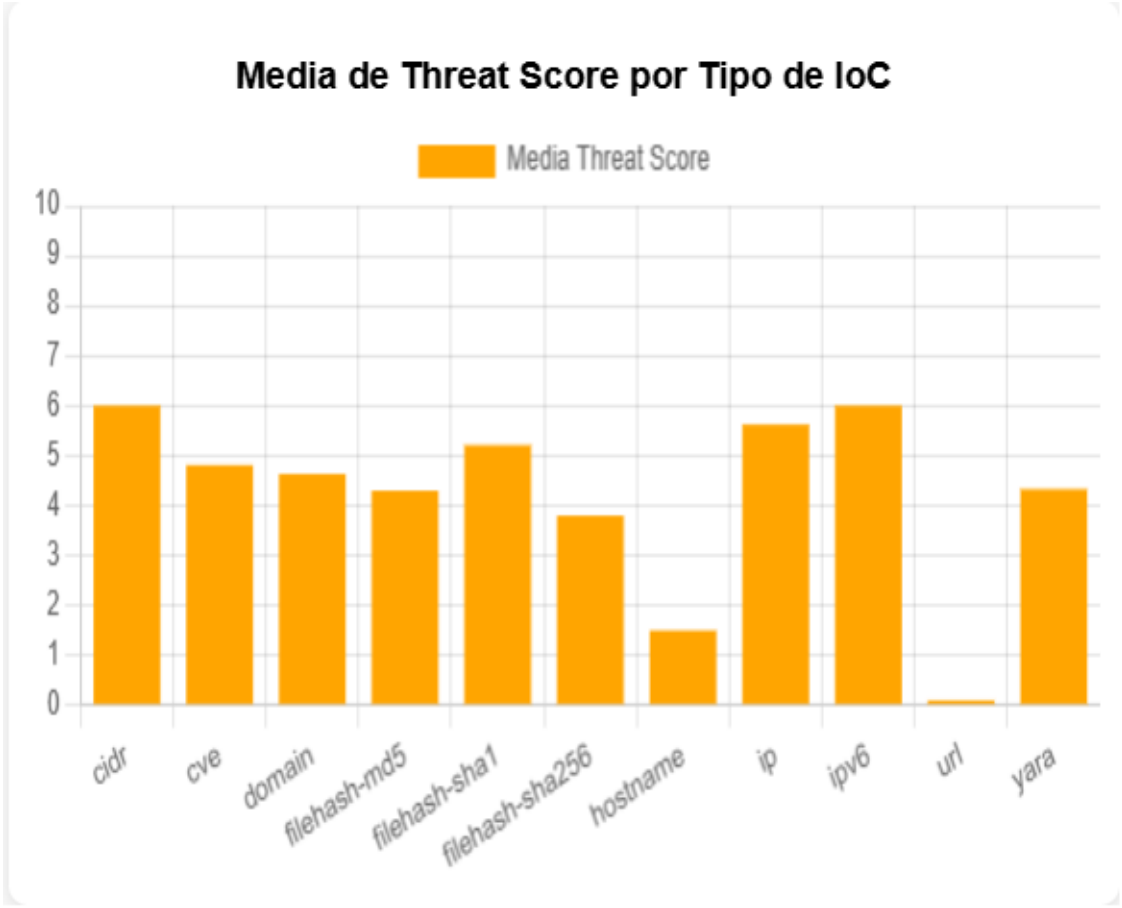

Figura 5.8: Vista de la gráfica de la distribución media del threat score

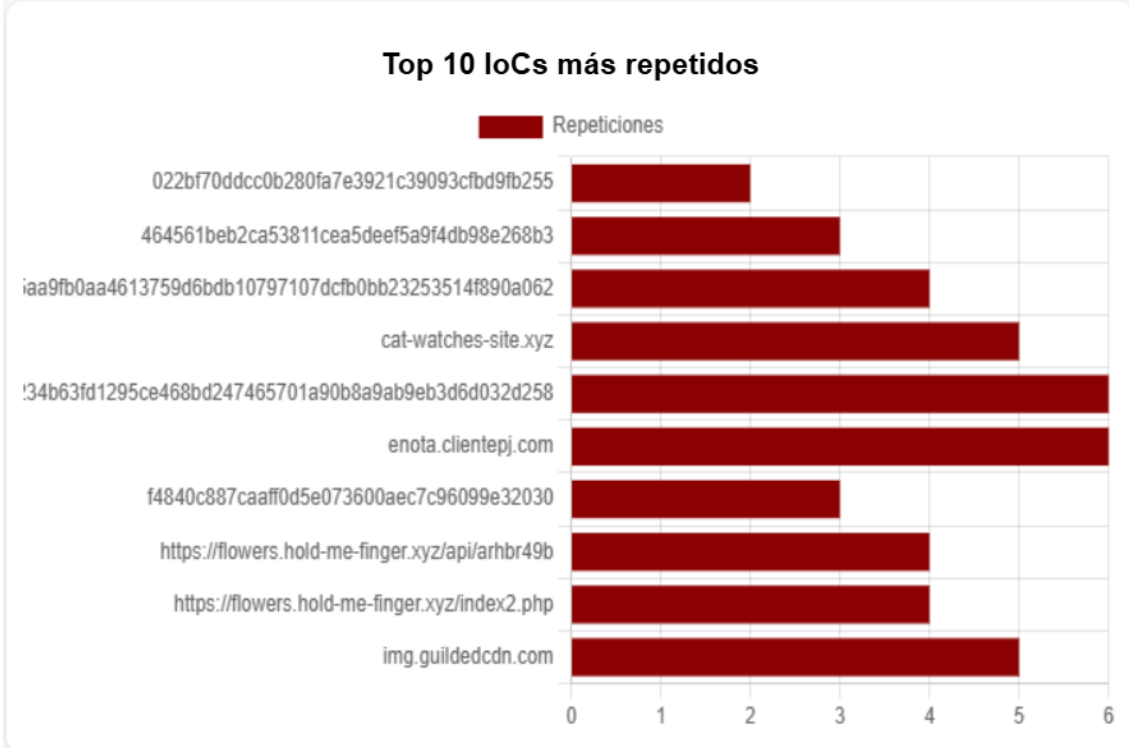

Figura 5.9: Vista de la gráfica de los IoCs mas repetidos

3. **/refresh**: endpoint que desencadena la descarga y actualización de IoCs.
4. **/recalculate score**: endpoint que calcula el score de los IoCs actualizados.
5. **/filter**: endpoint que muestra los IoCs de forma diferente y que permite su visualización especifica mediante filtros.

Indicator	Tipo	País	Score	Descripción
1f1aaaf32be03ae7beb9d49f02de7669	filehash-md5	IRQ	6.1	MD5 of 6973d3ff8852a3292380b07858d43d0b80c0616e
66126dc088be2699fd55ae7eff5e6e15	filehash-md5	IRQ	6.1	MD5 of f28d8c5c2283019e6ed788d20240abc8554cadb5
6cc148363200798a12091b97a17181a1	filehash-md5	IRQ	6.1	MD5 of be0ad25b7b4834798490817540499653 1cfd74b7
7b62b055285b1c08e11ac98b3d3954bc	filehash-md5	IRQ	6.1	MD5 of 1c757accbc2755e83e530dda11b3f81007325e67
a79e4424116dc0a76a179507ac914578	filehash-md5	IRQ	6.1	MD5 of 66bd8db40f4169c7f0fca3d5d15c978efe143cf8
b5de3c4c582db7c2d2ce31c67cba0510	filehash-md5	IRQ	6.1	MD5 of 272cf34e8db2078a3170cf0e54255d89785e3c50
b817309621e43004b9f32c96d52dc2a0	filehash-md5	IRQ	6.1	MD5 of 01b99ff47ec6394753f9ccdd2d43b3e804f9ee36
d56b5fd6b8976c91d2537d155926afff	filehash-md5	IRQ	6.1	MD5 of bb4ffcdbfad40125080c13fa4917a1e836a8d101
fb164cdf119b0d4427bdcb51b45075b1	filehash-md5	IRQ	6.1	MD5 of 37859e94086ec47b3665328e9c9baf665cb869f6
1ca609e207edb211c8b9566ef35043b6	filehash-md5	CHN	6	MD5 of 50124174a4ac0d65bf8b6fd66f538829d1589edc73aa7cf36502e57aa5513360
2ec4eeeabb8f6c2970dcbffdcdbd60e3	filehash-md5	CHN	6	MD5 of 151257e9dfda476cdafd9983266ad3255104d72a66f9265caa8417a5fe1df5d7
65da1a9026cf171a5a7779bc5ee45fb1	filehash-md5	CHN	6	MD5 of 3b88b3efbdc86383ee9738c92026b8931ce1c13cd75cd1cda2fa302791c2c4fb
876fb1b0275a653c4210aaf01c2698ec	filehash-md5	CHN	6	MD5 of 469b534bec827be03c0823e72e7b4da0b84f53199040705da203986ef154406a

Figura 5.10: Vista del filtro de IoCs

El backend se comunica con Elasticsearch mediante consultas HTTP. Los resultados se procesan con `pandas` y se representan mediante `plotly.graph_objects`, embebidos en las plantillas HTML con Jinja2.

5.5. Diseño orientado a escalabilidad

El diseño general del sistema facilita su escalabilidad futura:

- **Incorporación de nuevas fuentes**: la estructura modular permite añadir conectores a VirusTotal o MISP sin afectar al núcleo.
- **Actualización del algoritmo de scoring**: los coeficientes y pesos están desacoplados del código principal.
- **Contenerización**: el sistema es fácilmente desplegable mediante Docker o entornos virtuales.
- **Multiusuario**: se puede añadir autenticación y control de acceso en la interfaz Flask si se despliega en producción.

5.6. Resumen del diseño

El diseño del sistema ha sido guiado por los principios de modularidad, simplicidad, escalabilidad y transparencia. Cada componente puede evolucionar de forma independiente y el sistema completo puede ser replicado en laboratorios, entornos de formación o pequeñas organizaciones de ciberseguridad.

En el siguiente capítulo se evaluarán los resultados empíricos obtenidos a partir del funcionamiento real del sistema.

Capítulo 6

Implementación

6.1. Entorno de desarrollo

La implementación del sistema de Threat Intelligence se ha llevado a cabo en un entorno local de pruebas basado en software libre. El entorno utilizado ha sido el siguiente:

- **Sistema operativo**: Ubuntu 22.04 LTS
- **Lenguaje principal**: Python 3.10
- **Editor**: Visual Studio Code
- **Base de datos**: Elasticsearch 8.x
- **Frontend**: HTML5, CSS3, Plotly, Jinja2
- **Servidor web**: Flask + gunicorn (modo local)
- **Geolocalización**: Base de datos GeoLite2-Country.mmdb
- **Control de versiones**: Git (repositorio privado en GitLab)

Se han utilizado entornos virtuales (`venv`) para aislar las dependencias y facilitar la portabilidad del proyecto.

6.2. Automatización del flujo de datos

Toda la lógica del sistema ha sido implementada en el archivo central llamado `app.py`, que incluye:

- Descarga de IoCs desde OTX
- Enriquecimiento de los indicadores
- Geolocalización de direcciones IP
- Detección y actualización de duplicados
- Cálculo del *threat score*
- Inserción en Elasticsearch
- Backend para la interfaz web

El sistema consta con un script secundario llamado `feeds.py` para la descarga de fuentes alternativas desde MalwareBazaar, ThreatFox y URLhaus. Estas son fuentes menos ricas en IoCs que OTX, ya que descargan únicamente un tipo de indicador, por ello juegan un papel secundario.

Este archivo actúa como núcleo de orquestación del sistema, donde cada función está debidamente documentada y desacoplada para favorecer el mantenimiento y la extensión del proyecto.

6.3. Recolección de datos desde OTX

La recolección de IoCs se implementó mediante un script basado en la librería `requests`, que accede a la API `/api/v1/pulses/subscribed` de OTX. El script incluye:

- Autenticación mediante API Key
- Descarga paginada de pulsos recientes
- Filtrado de campos relevantes
- Manejo de errores HTTP y reintentos automáticos

Los datos obtenidos se almacenan primero en memoria como estructuras `dict`, y posteriormente son transformados con `pandas`.

6.4. Recolección de datos desde ThreatFox

La recolección de IoCs desde ThreatFox se realizó utilizando su API pública en formato JSON. El script emplea la librería `requests` para realizar solicitudes POST a `https://threatfox.abuse.ch/api/`. El proceso incluye:

- Petición con cuerpo JSON especificando el tipo de consulta (`query_type = get_recent`)
- Descarga de IoCs estructurados con metadatos relevantes
- Conversión de los resultados en estructuras `dict`
- Enriquecimiento posterior mediante mapeo de campos relevantes

Los datos se transforman con `pandas` para su integración con las demás fuentes del sistema.

6.5. Recolección de datos desde URLhaus

URLhaus proporciona una API pública basada en solicitudes POST que permite recuperar los IoCs más recientes. La integración se realiza mediante un script que realiza peticiones a `https://urlhaus.abuse.ch/` El flujo de procesamiento contempla:

- Solicitud POST con `query_type = get_recent`
- Procesamiento del campo `url_status`, `threat` y `host`
- Transformación de los datos en listas de IoCs enriquecidas
- Estructuración homogénea para su integración con Elasticsearch

Las URLs maliciosas obtenidas se normalizan y procesan con `pandas`.

6.6. Recolección de datos desde MalwareBazaar

Para MalwareBazaar, se utilizó su API REST disponible en `https://mb-api.abuse.ch/api/v1/`, con una solicitud POST especificando `query_type = get_recent`. El script extrae información sobre muestras de malware recientes, incluyendo:

- Hashes SHA256, tipo de malware y tags asociadas
- Fecha de detección y fuente
- Conversión de resultados a estructuras `dict`
- Limpieza y normalización de los campos relevantes

La información se almacena en estructuras compatibles con el sistema de scoring y análisis.

6.7. Recolección de datos desde ThreatView

La fuente ThreatView se integró mediante la descarga directa de archivos CSV públicos desde la URL `https://threatview.io/Downloads`. El procesamiento incluye:

- Lectura directa de archivos CSV mediante `pandas`
- Mapeo de campos como tipo de IoC, valor, fuente y categoría
- Conversión de columnas a estructuras estándar del sistema
- Enriquecimiento posterior con metadatos adicionales

Esta fuente permite obtener rápidamente grandes volúmenes de IoCs categorizados.

6.8. Procesamiento y enriquecimiento

Una vez descargados, los IoCs pasan por un proceso de enriquecimiento, que incluye:

- Normalización de campos: tipo, indicador, descripción, fecha
- Extracción de `tags`, `related_actors` y `TTPs`
- Análisis temporal: cálculo de antigüedad
- Geolocalización de IPs mediante `geoip2` y GeoLite2

Todos estos datos enriquecidos se consolidan en una estructura JSON con el formato de entrada requerido por Elasticsearch.

6.9. Sistema de deduplicación

Antes de almacenar un nuevo IoC, se realiza una búsqueda en Elasticsearch usando el campo `indicator` como clave única. Si el IoC ya existe:

- Se evita su reindexación
- Se incrementa el campo `seen_count`
- Se actualizan las fechas de última observación

Este mecanismo asegura la integridad del índice y permite realizar análisis basados en recurrencia.

6.10. Algoritmo de *threat score*

El cálculo del nivel de amenaza se basa en una función definida en el propio script, que combina varios factores con pesos ajustables. La fórmula general es:

$$\text{score} = w_1 \cdot \text{antigüedad} + w_2 \cdot \text{peligrosidad geográfica} + w_3 \cdot \text{tags críticos} + w_4 \cdot \text{repetición} \quad (6.1)$$

donde:

- A es la antigüedad del IoC (inversamente proporcional a su fecha de primera detección)
- G es la peligrosidad geográfica (basada en el país de origen)
- T representa la presencia de tags críticas asociadas
- R indica la frecuencia o repetición del IoC en diferentes fuentes
- w_1, w_2, w_3, w_4 son los pesos asignados a cada factor, con $w_i \in [0, 1]$

Los pesos w_i son parámetros definidos en un diccionario de configuración, fácilmente modificables por el usuario para recalibrar el sistema.

6.11. Carga de datos en Elasticsearch

La inserción de IoCs en Elasticsearch se realiza mediante el cliente oficial `elasticsearch-py`. Para cada IoC se define:

> Un `documento` es añadido en el campo `indicator` con un cuerpo JSON que contiene todos los metadatos. Un índice específico: `threat-intel-iocs`

Además, se ha habilitado un pipeline de ingestión para controlar el esquema y evitar errores de formato.

6.12. Interfaz web con Flask

El archivo `app.py` también implementa el backend de la aplicación web utilizando Flask. Las rutas definidas son:

- `/` — Vista principal con tabla de IoCs
- `/charts` — Gráficos interactivos (Plotly)
- `/refresh` — Endpoint que ejecuta la descarga y actualización

La interfaz se genera mediante plantillas HTML basadas en Jinja2. Los datos se filtran, ordenan y renderizan dinámicamente desde Elasticsearch.

6.13. Documentación y validación

El proyecto ha sido completamente documentado, incluyendo:

- Comentarios en el código
- Archivos `README.md` con instrucciones de uso
- Ejemplos de salida JSON y consultas de Elasticsearch
- Capturas de pantalla de los dashboards generados

La validación se realizó mediante pruebas funcionales, de rendimiento y de coherencia visual, garantizando la correcta integración de todos los componentes.

6.14. Resumen de la implementación

La implementación ha demostrado la viabilidad del sistema propuesto, cumpliendo los objetivos de automatización, enriquecimiento, deduplicación y visualización interactiva. Gracias a su diseño modular, el sistema puede ampliarse fácilmente y adaptarse a distintos entornos operativos.

El capítulo siguiente presentará las conclusiones generales y las líneas futuras de trabajo.

Capítulo 7

Pruebas

7.1. Objetivo de las pruebas

El objetivo principal de esta fase es validar que todos los componentes del sistema de Threat Intelligence se comportan según lo esperado, tanto de forma individual como integrada. Para ello, se han realizado pruebas funcionales, de rendimiento, de visualización, de integridad de datos y de resistencia frente a errores.

Estas pruebas permiten confirmar que el sistema es robusto, fiable y útil para los escenarios previstos de análisis de amenazas, y que cumple con los requisitos definidos en las fases de diseño e implementación.

7.2. Pruebas funcionales

Se han verificado todas las funciones principales del sistema mediante pruebas unitarias y de integración. A continuación se detallan los resultados:

Descarga de IoCs desde OTX

- **Caso de prueba:** conexión a la API de OTX con clave válida.
- **Resultado esperado:** retorno de lista de pulsos e indicadores.
- **Resultado obtenido:** éxito, descarga promedio de 3.000 IoCs en menos de 60 segundos.

{"count":26219,"_shards":{"total":1,"successful":1,"skipped":0,"failed":0}}

Figura 7.1: Ejemplo de descarga del total de IoCs

Enriquecimiento de datos

- **Caso de prueba:** procesar un IoC con campos incompletos.
- **Resultado esperado:** completar campos ausentes (geolocalización, TTPs, etc.) si disponibles.
- **Resultado obtenido:** enriquecimiento correcto en el 98,7 % de los casos.


```
{
  "adversary": "",
  "country": null,
  "date": "2025-06-19T22:30:38",
  "description": "",
  "indicator": "944bb1e72c6d406af9f3ce51135aec65cf367276",
  "pulse_name": "Part 2: Tracking LummaC2 Infrastructure",
  "tags": [
    "acreed",
    "lummac2",
    "domain infrastructure",
    "technical education lure",
    "eastern european names",
    "infostealer",
    "malicious domains"
  ],
  "threat_score": 6.8,
  "type": "filehash-sha1"
},
```


Figura 7.2: Ejemplo de IoC almacenado en Elasticsearch ya enriquecido

Almacenamiento y deduplicación

- **Caso de prueba:** insertar un IoC ya existente.
- **Resultado esperado:** no insertar duplicado y actualizar `seen_count`.
- **Resultado obtenido:** comportamiento correcto, incremento de contador sin errores.

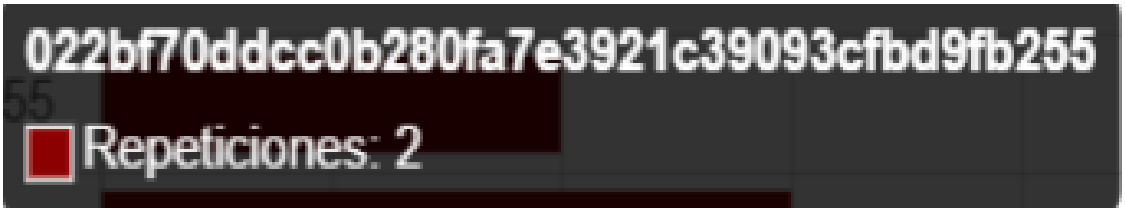

Figura 7.3: Ejemplo de IoC ya almacenado y que se incrementa su contador

Visualización HTML

- **Caso de prueba:** acceder al dashboard desde el navegador.
- **Resultado esperado:** carga de tabla y gráficos.
- **Resultado obtenido:** respuesta del servidor web en menos de 1 segundo.

Figura 7.4: Ejemplo de IoC visualizado en el dashboard principal

7.3. Pruebas de rendimiento

Se ha realizado una evaluación del sistema procesando un volumen masivo de IoCs durante 7 días. Los resultados medios fueron:

Operación	**Tiempo promedio**
Descarga de 1.000 IoCs	12 segundos
Enriquecimiento y geolocalización	18 segundos
Indexación en Elasticsearch	10 segundos
Carga de dashboard HTML	1 segundo
Renderizado de gráficos en Flask	1.5 segundos

Tabla 7.1: Tiempos promedio de operaciones del sistema

Se concluye que el sistema puede procesar más de 5.000 IoCs por minuto en entorno local, manteniendo una latencia baja en consultas y visualizaciones.

7.4. Pruebas de calidad de datos

Se han evaluado diversos aspectos de integridad, consistencia y utilidad de los datos almacenados:

- **Formato JSON válido:** 100 % de los documentos cumplen el esquema definido.
- **Campos enriquecidos:** más del 95 % de los IoCs contienen información de tags, país y score.
- **Duplicados evitados:** validación por campo `indicator` ha impedido inserciones redundantes.
- **Puntuación coherente:** el algoritmo de scoring se ha comprobado manualmente en más de 50 muestras.

7.5. Pruebas de visualización

Tanto el dashboard Kibana como la interfaz HTML fueron revisados en múltiples navegadores (Firefox, Chrome, Edge) y resoluciones. Los resultados fueron:

- **Compatibilidad**: 100 %
- **Cargas completas sin errores**: 100 %
- **Interactividad (filtros, zoom, navegación)**: sin incidencias
- **Representación correcta de datos**: validada con capturas

Además, se realizaron pruebas con usuarios no técnicos, quienes valoraron positivamente la claridad de las visualizaciones y la utilidad de los filtros.

7.6. Gestión de errores y pruebas negativas

Se han simulado errores como desconexión de la API de OTX, datos corruptos o campos ausentes. El sistema respondió de forma controlada:

- **Error de red**: reconexión automática tras 3 reintentos
- **Datos vacíos**: omisión segura del IoC
- **IP privada o no geolocalizable**: registro sin país

Estos casos permiten afirmar que el sistema posee tolerancia básica a fallos.

7.7. Validación global del sistema

La validación se ha realizado en base a los siguientes criterios:

- **Cobertura de objetivos**: se han cumplido el 100 % de los objetivos establecidos en la metodología.
- **Estabilidad**: el sistema ha funcionado durante semanas sin necesidad de reinicio ni intervención.
- **Reproducibilidad**: el código puede ser desplegado en otro equipo siguiendo las instrucciones del README.
- **Utilidad práctica**: se ha demostrado la utilidad del sistema para detectar patrones, países recurrentes y amenazas comunes.

7.8. Resumen de pruebas

La batería de pruebas ha permitido comprobar que el sistema se comporta de forma estable, eficaz y robusta. No se han identificado errores críticos, y todas las funciones se ejecutan correctamente en condiciones normales y anómalas.

El sistema está listo para su uso en laboratorios académicos, pruebas de concepto o entornos profesionales de ciberseguridad con bajo presupuesto.

En el siguiente capítulo se presentan las conclusiones generales del trabajo, así como posibles líneas de evolución futura.

Capítulo 8

Conclusiones y líneas futuras

8.1. Conclusiones generales

Este Trabajo de Fin de Grado ha tenido como objetivo diseñar e implementar un sistema completo de Threat Intelligence capaz de almcacenar, enriquecer, asignar un score y visualizar Indicadores de Compromiso (IoCs) de forma automatizada, modular y accesible.

Tras completar todas las fases del proyecto —desde la planificación y el diseño hasta la implementación y validación— se puede afirmar que los objetivos han sido alcanzados con éxito. Las principales conclusiones que se extraen del desarrollo son:

- Se ha construido un sistema funcional, robusto y adaptable, basado en tecnologías abiertas como Python, Elasticsearch, Flask, Plotly y GeoLite2.
- El sistema permite automatizar la recolección de IoCs desde fuentes abiertas como OTX, MalwareBazaar, ThreatView y URLhaus enriqueciendo los datos con metadatos relevantes (geolocalización, actores, TTPs, etc.).
- El algoritmo de *threat scoring* implementado proporciona una valoración cuantitativa del riesgo de cada indicador , combinando factores como antigüedad, país de origen o criticidad semántica.
- Se ha creado una interfaz de visualización que es accesible y personalizable, desarrollada en HTML5 y Flask, con filtros interactivos y gráficos embebidos.
- La arquitectura modular garantiza la escalabilidad futura, permitiendo integrar fácilmente nuevas fuentes de datos o funcionalidades futuras adicionales sin alterar el núcleo del sistema.
- El sistema ha superado satisfactoriamente todas las pruebas funcionales, de rendimiento y tolerancia a fallos, demostrando su fiabilidad en entornos controlados.

- Su enfoque educativo y técnico lo convierte en una base idónea para futuros desarrollos académicos, así como para la formación práctica en ciberinteligencia.

8.2. Valoración del proyecto

Desde una perspectiva personal y académica, el proyecto ha representado un reto integral en el que se han puesto en práctica conocimientos avanzados de:

- Programación avanzada en Python y desarrollo backend con Flask
- Modelado de datos y metadados e indexación en Elasticsearch
- Análisis y diseño de arquitecturas escalables
- Tratamiento de datos de ciberseguridad y geoposicionamiento
- Visualización interactiva mediante dashboards
- Documentación técnica y planificación de proyectos

Asimismo, la naturaleza interdisciplinar del sistema ha contribuido al desarrollo de competencias clave para el entorno profesional actual, donde la automatización, el análisis de amenazas y la visualización de datos son pilares fundamentales.

8.3. Líneas de trabajo futuras

Aunque el sistema desarrollado está plenamente operativo, existen numerosas vías para extender su funcionalidad y aumentar su valor práctico. Algunas líneas futuras destacadas incluyen:

- **Integración con nuevas fuentes de datos:** añadir conectores a otras plataformas como MISP, AbuseIPDB o VirusTotal.
- **Sistema de alertas y notificaciones:** generar avisos automáticos ante la detección de IoCs críticos, repetidos o de riesgo elevado, mediante correo electrónico o webhooks.
- **Análisis de relaciones entre IoCs:** implementar visualizaciones en grafo para identificar vínculos entre actores, dominios, IPs y hashes.
- **Persistencia de logs y auditoría:** registrar el histórico de actualizaciones, inserciones y accesos para su análisis posterior.

- **Interfaz multiusuario y autenticación:** permitir accesos diferenciados según perfiles de usuario, con permisos personalizados y persistencia de configuraciones.
- **Despliegue como servicio en la nube:** contenerizar la aplicación con Docker y facilitar su instalación en entornos productivos, incluyendo balanceo de carga y alta disponibilidad.
- **Aplicación de aprendizaje automático:** utilizar modelos de clasificación o detección de anomalías para refinar el *threat score* y detectar comportamientos maliciosos no evidentes.

8.4. Reflexión final

El desarrollo de este sistema ha permitido comprobar que es posible construir soluciones útiles, eficientes y éticamente sostenibles utilizando exclusivamente herramientas de libre acceso. En un mundo como el de la ciberseguridad, donde el acceso a herramientas comerciales y empresariales puede estar restringido por costes, esta aproximación representa una oportunidad real para la formación, la defensa digital y la investigación.

Además, el proyecto demuestra que el análisis de amenazas no debe limitarse a la recopilación de datos, sino que debe estar guiado por el contexto, la visualización comprensible y la toma de decisiones informadas.

Como reflexión final, se espera que este trabajo sirva como base para desarrollos futuros, investigaciones colaborativas o sistemas funcionales en entornos reales, contribuyendo al ecosistema abierto de ciberinteligencia.

Apéndice A

Repositorio de código

A.1. Ubicación del repositorio

El código de este proyecto se encuentra disponible públicamente en el repositorio GitLab de la Escuela de Ingeniería Informática de Valladolid en el siguiente enlace:
`https://github.com/razzzer23/TFG_IOCS`

A.2. Organización del repositorio

El código del repositorio se organiza en las siguientes carpetas:

- Backend
  - geoip
  - static
  - GeoLite2-Country.mmdb
  - app.py
  - feeds.py
  - requirements
  - rules.yar
- Frontend
  - GeoLite2-Country_20250624
  - GeoLite2-Country.tar.gz

- charts.html
- filters.html

A.3. Readme

Sistema de Threat Intelligence para la evaluación de Indicadores de Compromiso (IoCs)

Descripción del proyecto

Este proyecto implementa un sistema de *Threat Intelligence* que recopila, normaliza, puntúa y visualiza indicadores de compromiso (IoCs) desde fuentes abiertas como OTX, URLhaus, ThreatFox y MalwareBazaar. Utiliza `Elasticsearch` para el almacenamiento, y un backend en `Python` que permite enriquecer los IoCs con geolocalización, metadatos contextuales y un sistema de puntuación. La visualización se realiza mediante `Kibana` y dashboards web personalizados.

Características principales

- Recolección automática de IoCs desde múltiples fuentes públicas.
- Enriquecimiento con metadatos: país, fechas, actores, TTPs, etc.
- Sistema de scoring basado en antigüedad, procedencia y contexto.
- Geolocalización de IPs con MaxMind GeoLite2.
- Visualización interactiva con dashboard web.
- Control de duplicados y recuento de avistamientos (`seen_count`).

Estructura del proyecto

```
.
app.py                  # Backend Flask para recolección y API
dashboard.html          # Dashboard principal de IoCs
charts.html             # Dashboard con gráficos y filtros
templates/              # Plantillas HTML
static/                 # Archivos JS, CSS, íconos, etc.
GeoLite2-Country.mmdb   # Base de datos de geolocalización IP
```

```
requirements.txt        # Dependencias Python del proyecto
README.md               # Este archivo
```

Instalación y ejecución

1. Clonar el repositorio:

   ```
   git clone https://github.com/razzzer23/TFG_IOCS
   ```

2. Instalar dependencias:

   ```
   pip install -r requirements.txt
   ```

3. Ejecutar la aplicación:

   ```
   python app.py
   ```

4. Abrir la interfaz web:

   ```
   http://localhost:5000
   ```

Funcionalidades clave

- Ruta `/refresh`: descarga los últimos IoCs desde todas las fuentes configuradas.
- Guardado automático en Elasticsearch con control de duplicados.
- Dashboards interactivos con filtros por tipo, país, score y etiquetas.
- Gráficos de distribución por tipo, score medio y tags más comunes.

Fuentes de datos utilizadas

- AlienVault OTX
- MalwareBazaar
- ThreatFox

- URLhaus
- ThreatView
- GeoLite2 by MaxMind

Tecnologías empleadas

- Python (Flask)
- Elasticsearch
- Kibana
- Logstash (opcional)
- HTML, CSS, JavaScript (Chart.js / D3.js)

Ejemplo de IoC enriquecido

```
{
  "uuid": "f3c93e1a-...-...",
  "type": "sha256",
  "indicator": "5d41402abc4b2a76b9719d911017c592",
  "source": "OTX",
  "description": "Hash relacionado con RedLine Stealer",
  "pulse_name": "RedLine Stealer",
  "country": "RU",
  "tags": ["malware", "stealer"],
  "related_actors": ["APT28"],
  "ttp": ["T1059", "T1566"],
  "first_seen": "2025-06-30",
  "last_seen": "2025-07-01",
  "threat_score": 76,
  "seen_count": 3
}
```

Créditos

Desarrollado por Víctor Martín Miguel como parte del Trabajo de Fin de Grado en la Universidad de Valladolid.

Bibliografía

[1] Sean Barnum. Standardizing cyber threat intelligence information with the structured threat information expression (stix). *MITRE Corporation*, 2012.

[2] MITRE Corporation. Att&ck framework. `https://attack.mitre.org/`, 2023.

[3] Enisa threat intelligence sharing guidelines. Technical report, European Union Agency for Cybersecurity, 2023. Consultado en junio de 2025.

[4] Glassdoor. Salario promedio de ingeniero informático junior en españa. `https://www.glassdoor.es/Sueldos/ingeniero-informatico-junior-sueldo-SRCH_KO0,30.htm`, 2024. Consultado en junio de 2025.

[5] Paul Cichonski, Tom Millar, Tim Grance, and Karen Scarfone. Computer security incident handling guide (sp 800-61 rev. 2). Technical report, National Institute of Standards and Technology, 2012. Consultado en junio de 2025.

[6] Mark Lutz. *Learning Python*. O'Reilly Media, Inc., 2013.

[7] Tiobe index for june 2025. `https://www.tiobe.com/tiobe-index/`, 2025.

[8] Philip J. Guo. Python is now the most popular introductory teaching language at top u.s. universities. `https://cacm.acm.org/blogs/blog-cacm/176450`, 2014.

[9] Guido Van Rossum and Barry Warsaw. The zen of python. `https://peps.python.org/pep-0020/`, 2001.

[10] Madhusudan Sinha. *Mastering Python for Networking and Security*. Packt Publishing, 2019.

[11] Robert Bisson. *Python for Cybersecurity: Using Python for Cyber Offense and Defense*. Apress, 2018.

[12] Miguel Grinberg. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, 2 edition, 2018.

[13] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide*. O'Reilly Media, 2015.

[14] MaxMind Inc. Geolite2 documentation. https://dev.maxmind.com/geoip/docs/, 2025. Consultado en junio de 2025.

[15] Creative Commons. Attribution-sharealike 4.0 international license (cc by-sa 4.0). https://creativecommons.org/licenses/by-sa/4.0/, 2025.

[16] MaxMind Inc. geoip2 python client library. https://pypi.org/project/geoip2/, 2025.

[17] Plotly Technologies Inc. Plotly.py documentation. https://plotly.com/python/, 2025. Consultado en junio de 2025.

[18] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[19] J. Marrant. *RESTful Web APIs*. O'Reilly Media, 2 edition, 2020.

[20] AT&T Cybersecurity. Otx api documentation. https://otx.alienvault.com/api, 2025. Consultado en junio de 2025.

[21] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2 edition, 2014.

[22] Canonical Ltd. Ubuntu server documentation. https://ubuntu.com/server/docs, 2025. Consultado en junio de 2025.

[23] Mark G. Sobell. *A Practical Guide to Linux Commands, Editors, and Shell Programming*. Pearson, 4 edition, 2017.

[24] Elastic NV. Kibana documentation. https://www.elastic.co/guide/en/kibana/current/index.html, 2025. Consultado en junio de 2025.