Facultad

de

Ciencias

GEMELOS DIGITALES PARA AUDITORÍAS DE SEGURIDAD CON INTELIGENCIA ARTIFICIAL

DIGITAL TWINS FOR SECURITY AUDITS WITH ARTIFICIAL INTELLIGENCE

Trabajo de Fin de Grado

para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Adrián Ceballos Berasategui

Director: Carlos Blanco Bueno

Co-Director: Ángel Ramos Obregón

Febrero – 2025

Agradecimientos

En primer lugar, me gustaría darle las gracias a mi familia, pareja y amigos por el apoyo incondicional recibido durante el transcurso del grado. Sin vuestra ayuda hubiera sido todo mucho más difícil.

En segundo lugar, me gustaría agradecer a todos los profesores por el conocimiento que me han permitido adquirir durante estos cuatro años, especialmente a Carlos Blanco, por aceptar la dirección de este TFG e introducirme al mundo de la ciberseguridad.

Finalmente, gracias a mis compañeros Ismael y Mario, así como a Ángel, por darme la oportunidad de formar parte del departamento de ciberseguridad en Ambar Telecomunicaciones, así como dirigirme y ayudarme durante el transcurso del proyecto.

Resumen

La ciberseguridad es un aspecto clave en la protección de los activos digitales de una empresa, y las auditorías de seguridad son una parte fundamental para identificar todo tipo de vulnerabilidades en los sistemas, para así poder prevenir cualquier ataque.

Por otra parte, los gemelos digitales han surgido hace poco tiempo como una tecnología innovadora, que permite la simulación precisa de entornos físicos en entornos virtuales, pudiéndose utilizar para el análisis y pruebas de seguridad en un entorno controlado. La integración de la Inteligencia Artificial en este proceso permite no solo detectar posibles anomalías de forma eficiente, sino también automatizar la resolución de estas amenazas.

En este Trabajo Fin de Grado se propone la construcción de un gemelo digital que represente una red real utilizando Docker, sobre el cual se llevarán a cabo pruebas de intrusión y distintos tipos de ataque. Posteriormente, los logs generados por el tráfico en la red se recolectarán en un SIEM utilizando ELK Stack, analizando mediante técnicas de IA posibles anomalías y tratar de resolverlas de manera automatizada.

El objetivo es desarrollar un prototipo que demuestre cómo los gemelos digitales, junto con la IA, pueden ser aplicados en el ámbito de la seguridad informática, mejorando la precisión del proceso de detección y mitigación de amenazas sin poner en riesgo la arquitectura real de la empresa.

Palabras clave

Gemelos digitales, ciberseguridad, detección de anomalías, IA, Docker, ELK.

Abstract

Cybersecurity is a key aspect in the protection of a company's digital assets, and security audits are fundamental for identifying vulnerabilities in systems to prevent potential attacks.

On the other hand, digital twins have recently emerged as an innovative technology, that allows for the accurate simulation of physical on virtual environment, making them useful for security analysis and testing in a controlled environment. The integration of artificial intelligence into this process enables the efficient detection of anomalies and automates the resolution of these threats.

This Final Degree Project proposes the construction of a digital twin representing a real network using Docker, where anomalous intrusion tests and some types of attacks will be carried out. Subsequently, logs generated by network traffic will be collected by SIEM using ELK stack, and potential anomalies will be analyzed using AI techniques to solve them automatically.

The objective is to develop a protoype demonstrating how digital twins, combined with AI, can be applied in the field of cybersecurity, enhancing the accuracy of threat detection and mitigation without jeopardizing the company's actual architecture.

Keywords

Digital twins, cybersecurity, anomaly detection, AI, Docker, ELK.

Índice

Contenido

Índice de imágenes

1. Introducción

En la actualidad, la tecnología está presente prácticamente en todos sitios, desde una empresa hasta un hogar, pasando por un comercio, un hospital o incluso un coche. Los avances tecnológicos avanzan a una velocidad de vértigo, aportándonos muchas cosas positivas y haciéndonos la vida mucho más fácil.

Pero no todo es bueno, ya que todas estas nuevas tecnologías pueden traer consigo fallos de seguridad que provoquen grandes pérdidas, como el robo de datos en los usuarios de una aplicación, o el aprovechamiento de una vulnerabilidad para conseguir acceder al servidor de una empresa y encriptar los datos, pidiendo un rescate que no te asegura su recuperación.

Por ello, la ciberseguridad toma un papel muy importante en la sociedad actual, intentando prevenir estos ataques, así como mitigándolos en caso de que sucedan.

Este proyecto ha sido realizado en la empresa Ambar Telecomunicaciones, concretamente en el departamento de ciberseguridad.

1.1 Ciberseguridad

En esta nueva era digital, la ciberseguridad se ha convertido en un aspecto fundamental para proteger los sistemas y los datos, tanto de empresas como de usuarios particulares.

La ciberseguridad no solo se centra en proteger los datos y los accesos no autorizados a los sistemas, sino también de garantizar la tríada CIA (Confidencialidad, Integridad y Disponibilidad).

Con el creciente número de ataques informáticos, las empresas deben adoptar diversas soluciones para conseguir detectar y mitigar estas amenazas en tiempo real. De lo contrario, podrán sufrir un ataque y perderlo todo de un día para otro. La ciberseguridad ha pasado de ser una preocupación secundaria a convertirse en una prioridad. [9]

El uso de servicios como los alojados en la nube, IoT, inteligencia artificial, etc. Ha hecho que las amenazas cibernéticas vayan evolucionando a la par, mejorando las técnicas de ataque para aprovecharse de alguna vulnerabilidad de cualquiera de estos servicios emergentes y hacerse con el sistema.

Para hacer frente a estos ataques, las empresas suelen utilizar dispositivos y herramientas de protección como firewalls, sistemas de detección de intrusos (IDS) y herramientas SIEM para visualizar y analizar el tráfico de la red.

También se han comenzado a desarrollar nuevas estrategias de defensa como la basada en Inteligencia Artificial, en la que se utiliza Machine Learning para analizar patrones de ataques en tiempo real. Otra estrategia muy popular actualmente es la de Zero Trust, en la que ningún usuario o dispositivo es confiable por defecto, y necesita cumplir ciertos requisitos para serlo.

Los sistemas de seguridad tradicionales, como los antivirus, han demostrado no ser suficiente frente a las amenazas actuales. Muchos ataques sofisticados consiguen evadir los mecanismos de protección convencionales con técnicas avanzadas como la ofuscación del código de un malware o el tráfico encriptado.

Por ello, se propone una solución innovadora para probar estrategias de seguridad en un entorno controlado sin que comprometa la infraestructura real de la empresa.

1.2 Importancia de los gemelos digitales en la actualidad

Un gemelo digital es una representación virtual precisa de un producto, un servicio o un proceso físico, que se utiliza para emular, predecir y optimizar el funcionamiento de cualquier tipo de sistema ciberfísico.

Sus primeras aplicaciones se remontan a la misión espacial Apolo de la NASA, donde los ingenieros crearon réplicas de los sistemas enviados al espacio con el objetivo de poder ayudar a los astronautas desde la tierra, reproduciendo su comportamiento. [1]

Actualmente, los gemelos digitales forman parte de diferentes sectores como el de la salud, en el que se están desarrollando gemelos digitales para simular y monitorizar las operaciones hospitalarias, así como en el área de transporte, en el que se analizan en tiempo real las flotas de vehículos para evitar y controlar atascos.

También se utiliza mucho en el ámbito aeroespacial y defensa, por ejemplo, para realizar las pruebas de diferentes componentes que se quieren implementar sin la necesidad de realizar vuelos de manera real. En el sector de la agricultura se está implementando para monitorizar en tiempo real el estado de los cultivos, y así mejorar el rendimiento de los mismos.

En el ámbito de la ciberseguridad, un gemelo digital va a permitir simular ataques sin afectar a la infraestructura real, así como probar nuevas estrategias de detección y mitigación de amenazas antes de implementarlas en producción en la red real.

1.2 Inteligencia Artificial

La Inteligencia Artificial (IA) es una de las tecnologías más innovadoras de todos los tiempos. Actualmente, se utiliza en múltiples sectores, y avanza a un ritmo muy grande.

La IA se basa en la creación de un sistema capaz de realizar tareas que requieran inteligencia humana. Es muy utilizado en el reconocimiento de patrones, como en la medicina para detectar algún tipo de tumor maligno, así como en la toma de decisiones y la automatización de procesos.

Existen varias disciplinas dentro de la IA, como son:

- Machine Learning (Aprendizaje Automático): Son algoritmos que aprenden de los datos que ingieren para hacer predicciones.
- Deep Learning (Aprendizaje Profundo): Se basa en redes neuronales profundas, utilizadas comúnmente en el reconocimiento de imágenes y el procesamiento del lenguaje natural.
- Procesamiento de Lenguaje Natural (NLP): Se aplica en los famosos chatbots o asistentes virtuales, así como en analizadores de texto.

La IA, como he dicho antes, puede aplicarse a una gran cantidad de campos. En este caso nos centraremos en la IA aplicada a la ciberseguridad. Se está utilizando mucho para la detección de amenazas, ya que no se limita sólo a detectar un patrón de ataque conocido, sino que los algoritmos que se utilizan van aprendiendo y detectando ataques mucho más sofisticados. [10]

Además, tienen la capacidad de realizar una toma de decisiones, pudiendo responder a los incidentes si se desea sin la necesidad de una tercera persona, reduciendo significativamente el tiempo de respuesta, evitando así que los ataques se propaguen o que lleguen a causar daños importantes.

1.3 Motivación y objetivo del proyecto

Tras analizar las necesidades de seguridad en las redes empresariales, el uso emergente tanto de gemelos digitales como de IA en ciberseguridad, se plantea la siguiente pregunta.

¿Es posible crear un entorno seguro de pruebas basado en un gemelo digital que permita detectar y mitigar ataques en tiempo real?

Por tanto, la finalidad del proyecto es la investigación y puesta en marcha de un proyecto basado en gemelos digitales, los cuales ayuden a simular el entorno real de una empresa para poder realizar ataques y comprobar mediante un SIEM la eficacia de los sistemas implantados, así como la detección y mitigación de anomalías mediante técnicas de Inteligencia Artificial.

Para la consecución del objetivo planteado se establecen los siguientes subobjetivos:

- Simulación de una red empresarial simplificada con diferentes servicios como servidores web, bases de datos, endpoints, firewall.
- Recolección y análisis de logs mediante herramientas como Filebeat, Logstash y Elasticsearch.
- Visualización de esos logs integrando herramientas como Kibana y Grafana.
- Aplicación de diferentes técnicas de detección de anomalías, tanto manuales como automáticas utilizando IA.
- Automatización de la respuesta a incidentes mediante el bloqueo de las IPs anómalas en el firewall de la red.
- Visualización en tiempo real de los resultados obtenidos mediante la generación de gráficas.

2. Herramientas y métodos

En este punto se detallarán las herramientas utilizadas a lo largo del trabajo y la explicación de por qué se ha escogido cada una de ellas, así como de los métodos y la planificación que se ha utilizado para la realización del TFG.

2.1 Elección de herramientas

Para la realización de este TFG se utilizarán tecnologías diferentes para abordar cada parte del trabajo:

- VMWare
- Docker
- ELK (Elasticsearch, Logstash, Kibana)
- Grafana
- Filebeat
- Python
- Isolation Forest

La elección de estas tecnologías se ha basado en varios factores que explicaré más adelante, pero que comparten una naturaleza común. Todas estas herramientas son de origen open-source, aunque alguna tiene variantes de pago, pero para el uso que se le ha dado en la realización del TFG con la variante gratuita es más que suficiente.

Además, todas ellas proporcionan una gran flexibilidad y escalabilidad a la hora de trabajar, sumado al amplio contenido y documentación de cada una, haciendo el trabajo de investigación y desarrollo más llevadero.

2.1.1 VMWare

VMWare es una plataforma de virtualización que permite ejecutar una cantidad muy grande de sistemas operativos en una única máquina física. Es decir, consigue abstraer el hardware de un ordenador en una máquina virtual, que simula y se comporta como una máquina física diferente. [2]

En este caso he utilizado la versión gratuita VMWare Player, que me ha parecido mejor opción que VirtualBox por su capacidad de personalización y facilidad de uso, aunque las dos son buenas opciones para este caso.

El sistema operativo que he utilizado ha sido Ubuntu 22.04, que es una distribución de Linux, caracterizada por su simplicidad, fiabilidad y por ser de carácter open-source también.

Las especificaciones de la máquina virtual son las siguientes.

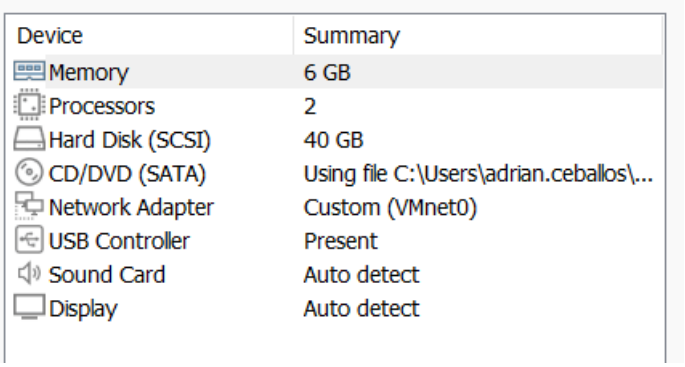

Figura 1: Especificaciones generales del sistema

En esta máquina virtual se van a alojar las diferentes herramientas comentadas anteriormente. Se pensó en utilizar muchas máquinas virtuales que simulasen la arquitectura del sistema y los diferentes componentes, pero se descartó esa opción por la cantidad de memoria RAM que consumiría todo.

En lugar de eso, se ha utilizado **Docker** para unir todo en diferentes contenedores que cumpliesen esas mismas funcionalidades, pero sin utilizar tantos recursos.

2.1.2 Docker

Docker es una plataforma de contenedores de código abierto que permite ejecutar aplicaciones en entornos controlados y aislados de la máquina denominados contenedores. [3]

Estos contenedores incluyen todo lo necesario para ejecutar lo que se desee, así sea un sistema operativo o una simple herramienta. El uso de Docker se justifica por las limitaciones del hardware a la hora de simular todos los dispositivos de la red.

El PC utilizado para este trabajo cuenta con un procesador Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz, 8 GB de RAM y 512 GB de almacenamiento interno.

Por ello, se ha optado por la utilización de esta herramienta, ya que otra de las opciones que se valoraron fueron la del software GNS3[8], que permite la creación de una arquitectura de red para posteriormente ser simulada mediante máquinas virtuales.

Esta opción ha sido descartada por la cantidad de recursos innecesaria que se utilizan para la ejecución de las máquinas virtuales al mismo tiempo, pero no se descarta en una implementación futura, ya que mejoraría la capacidad de personalización bastante respecto a Docker.

Se han desplegado los siguientes dispositivos:

- 2 endpoints Linux (Ubuntu)
- 1 máquina atacante

- 1 máquina que actúa como SIEM (Elasticsearch)
- 2 máquinas que actúa como visualizador de datos (Kibana, Grafana)
- 1 máquina que recoge los logs y los analiza

Con la opción de GNS3, todas estas máquinas hubieran colapsado los recursos del portátil, por eso se ha optado por Docker, creando contenedores que simulan todas estas máquinas.

2.1.3 ELK

ELK es un conjunto de herramientas open-source desarrollado por la empresa Elastic. Estas herramientas están pensadas para gestionar, analizar y visualizar volúmenes de datos muy grandes, generados por sistemas y aplicaciones de diferentes ámbitos. [4]

ELK se caracteriza por su fácil integración y compatibilidad con un gran abanico de sistemas. En este TFG se van a utilizar los 3 componentes más famosos y utilizados de ELK, que son Elasticsearch, Logstah y Kibana, que explicaré a continuación.

2.1.3.1 Elasticsearch

Elasticsearch es una herramienta de código abierto diseñada para trabajar con una gran cantidad de datos. Se caracteriza por lo siguiente [4]:

- Organiza los datos en diferentes índices, algo así como lo que hace una BBDD con las tablas. Esto permite que se puedan realizar búsquedas muy rápidas a través de su estructura e indexación.
- Distribuye los datos en diferentes nodos para mejorar el rendimiento, además de que es capaz de manejar millones de registros simultáneamente.
- Soporta búsquedas de gran complejidad, con diferentes filtros para extraer información detallada de lo que se desee.
- Se puede procesar la información y consultar datos prácticamente en tiempo real, es decir, una vez que se envían los datos de logstash a elasticsearch, ya pueden ser consultados directamente.
- Integración directa con diferentes sistemas mediante la API. Esta utilidad permite buscar datos dentro de un índice en tiempo real.

En este trabajo se utiliza Elasticsearch como nexo de unión para almacenar, indexar y buscar la información deseada entre los logs que se van generando en el gemelo digital. En el script que se desarrollará más adelante, se explicará como integrar Elasticsearch mediante la API para analizar los logs con el uso de la librería pandas de Python y el LLM Isolation Forest para la detección de anomalías, así como otras técnicas manuales de detección de ataques mediante logs.

2.1.3.2 Logstash

Logstash es una herramienta de procesamiento de datos. Su función principal es transformar datos sin estructura en datos convertidos a formato JSON, para su posterior almacenamiento en Elasticsearch. A estos datos vacíos le incluye una serie de metadatos e información adicional creando un log más completo y característico. [4]

En este TFG, Logstash desempeña un papel muy importante en la recolección, transformación y envío de logs generado por los distintos endpoints y herramientas del gemelo digital de la infraestructura creada.

Esta herramienta recogerá logs desde múltiples fuentes del gemelo digital, entre los que se encuentran las máquinas que actúan de servidores web como NGINX o Apache.

Una vez recolectados todos estos logs, se normalizarán y se enviarán a Elasticsearch, donde se almacenarán, como he mencionado antes, de manera estructurada e indexada para su posterior análisis.

2.1.3.3 Kibana

Kibana es una herramienta de visualización de datos que forma parte de ELK. Se basa en una interfaz gráfica diseñada para interactuar con los datos que almacena e indexa Elasticsearch, para así poder visualizarlos de forma gráfica en dashboards interactivos, que permitirán al usuario filtrar logs por una gran cantidad de campos dependiendo de las necesidades buscadas. [4]

La conexión con Elasticsearch es mediante su API, y permite observar el comportamiento de los diferentes elementos de la red para visualizar comportamientos sospechosos, aunque esto no es lo que se busca en el TFG, sino que se busca un análisis automatizado que se desarrollará más adelante.

2.1.4 Grafana

Grafana es otra plataforma de código abierto diseñada para la monitorización y visualización de datos de diferentes fuentes. Es una alternativa a Kibana que se ha optado por su sencilla creación de dashboards interactivos y personalizables, que permite una fácil integración con Elasticsearch mediante su API. [5]

También se pueden crear alertas personalizadas, que se podrían llegar a utilizar en el caso de que haya un número elevado de peticiones a un servidor, por ejemplo, para alertar al administrador de una posible anomalía, por correo electrónico u otro sistema compatible con Grafana.

2.1.5 Filebeat

Filebeat es una herramienta de recolección de logs parecida a logstash. Forma parte de ELK y está diseñada para para capturar logs desde servidores, aplicaciones y otros sistemas hacia un destino centralizado, normalmente logstash y elasticsearch. [6]

Se caracteriza por su fácil configuración, en el que se utiliza un archivo llamado *filebeat.yml* para definir las rutas de los logs en el sistema, el formato de salida y alguna otra operación. Este archivo de configuración se explicará en detalle más adelante.

2.1.6 Python

Python es un lenguaje de programación de alto nivel. Es mundialmente famoso por su legibilidad, simplicidad y amplia gama de bibliotecas, que permiten el desarrollo de software en diferentes áreas como el desarrollo web, el análisis de datos, la inteligencia artificial o incluso el uso de scripting para Linux. Esto último puede ser de gran ayuda para multitud de tareas, como la automatización de procesos.

En este TFG se ha optado por el uso de Python, ya que es un lenguaje amplio y fácil de utilizar, que en este caso se utilizará fundamentalmente para el análisis de logs y la detección de anomalías en los mismos. [7]

Para ello, utilizaré herramientas como **pandas**, que facilita la manipulación de grandes cantidades de datos para su posterior análisis, así como la biblioteca **scikit-learn** y similares para implementar el modelo de IA llamado Isolation Forest, el cual se caracteriza por su gran capacidad de detección de anomalías. Lo explicaré con más detalle en el siguiente apartado.

Por otra parte, se ha escogido este lenguaje por su capacidad de automatizar tareas y procesos, ya que se ha creado un script que automatiza la llamada a la API de Elasticsearch para la recolección de datos, la posterior recogida y procesamiento del log, el análisis del mismo con el modelo de IA, la creación de tablas y gráficas mostrando las IPs normales y las anómalas con la biblioteca **matplotlib** y la posterior acción, ya sea bloqueando la IP maliciosa directamente o si se desea generar un aviso para que un tercero decida si bloquear o no esa dirección.

2.1.7 Isolation Forest

Isolation Forest es un algoritmo de aprendizaje automático que está diseñado especialmente para la detección de anomalías. Se basa en una premisa de que los puntos de datos anómalos y distintos son más sencillos de aislar que los datos normales. Por este motivo, el algoritmo utiliza árboles de decisión para dividir los datos de forma completamente aleatoria hasta conseguir aislar los datos atípicos.

Se caracteriza por su alta eficiencia, ya que cuenta con una complejidad computacional de O(n log n), lo que hace que sea un algoritmo muy útil para el manejo de grandes cantidades de datos. [8]

Por ello ha sido escogido en este TFG, además de su naturaleza open-source, y su gran capacidad para ser integrado con facilidad en Python.

Más adelante se explicará la parte del código en la que se ha utilizado este modelo junto a su funcionamiento puesto en marcha, pero se resume en la ingesta de datos, el entrenamiento del modelo con esos datos, y por último la detección de las anomalías utilizando un índice de contaminación deseado para establecer el grado de "dureza" del algoritmo frente a las anomalías.

Cuanto mayor es el índice de contaminación, mayor será la tasa de falsos positivos en la detección de anomalías.

2.2 Métodos y planificación

El desarrollo del proyecto se ha llevado a cabo siguiendo una metodología iterativa incremental, en la que se han ido añadiendo y modificando funcionalidades en cada fase del proyecto.

Las etapas principales han sido:

- Investigación previa (9/9/2024 – 30/9/2024): Investigación sobre todo lo relacionado con el proyecto, es decir, gemelos digitales, herramientas a utilizar, modelos de Inteligencia Artificial, etc.
- Diseño del entorno (1/10/2024 – 21/10/2024): Creación de la red de la empresa y el gemelo digital utilizando Docker.
- Implementación del SIEM (22/10/2024 – 8/11/2024): Configuración de los contenedores ELK (Elasticsearch, Logstash, Kibana) y visualización de los logs en dashboards interactivos.
- Desarrollo de IA para detección de anomalías (11/11/2024 – 2/12/2024): Implementación del modelo Isolation Forest y puesta en marcha del script detector de logs anómalos.
- Automatización de respuesta a amenazas (3/12/2024 – 16/12/2024): Modificación del script para automatizar el bloqueo de logs anómalos en el firewall de la red simulada.
- Pruebas y ajustes del sistema (17/12/2024 – 13/1/2025): Simulación de distintos ataques para ver cómo se comporta el sistema.
- Redacción de la memoria (14/1/2025 – 7/2/2025): Redacción de la memoria del proyecto.

Para la planificación inicial de las tareas, se ha utilizado el software **GanttProject,** que se caracteriza por su fácil uso y naturaleza open-source. Se adjunta a continuación un diagrama de Gantt en el que se puede ver lo que la planificación que he seguido junto a las diferentes etapas de manera visual.

Nombre	Fecha de inicio	Fecha de fin
Investigación previa	9/9/24	30/9/24
Revisión del estado del arte	9/9/24	23/9/24
Evaluación de tecnologías	23/9/24	30/9/24
Diseño del entorno	1/10/24	21/10/24
Creación del gemelo digital con Docker	1/10/24	14/10/24
Configuración de redes y contenedores	14/10/24	21/10/24
Implementación del SIEM	22/10/24	8/11/24
Configuración de Filebeat, Logstash y Elasticsearch	22/10/24	4/11/24
Visualización de logs en Kibana y Grafana	4/11/24	8/11/24
Desarrollo de IA para detección de anomalías	11/11/24	2/12/24
Implementación de Isolation Forest	11/11/24	25/11/24
Generación y análisis de tráfico anómalo	25/11/24	2/12/24
Automatización de respuesta a amenazas	3/12/24	16/12/24
Integración con firewall UFW	3/12/24	16/12/24
Pruebas y ajustes del sistema	17/12/24	13/1/25
Simulación de ataques y evaluación del sistema	17/12/24	13/1/25
Redacción de la memoria	14/1/25	7/2/25

Figura 2: Diagrama de Gantt

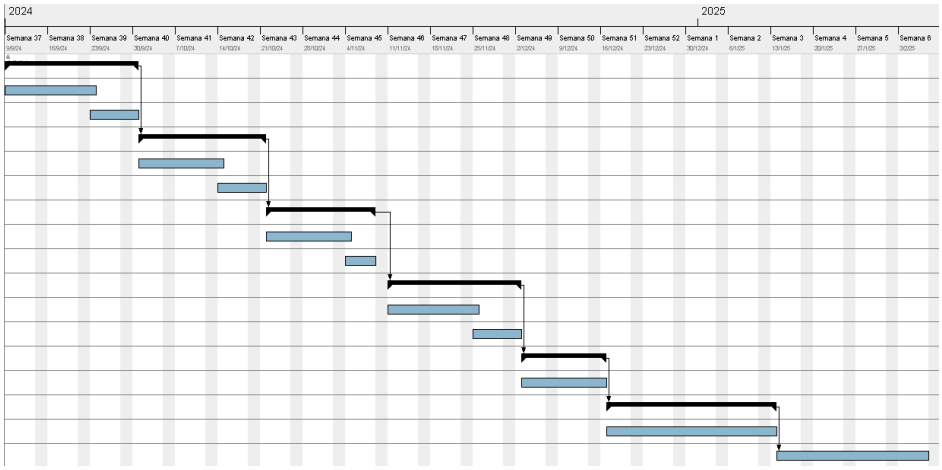

Figura 3: Diagrama de Gantt 2

3. Análisis del problema

La seguridad en las redes empresariales es un aspecto crítico que pueden afectar en gran medida la protección de los datos tanto de clientes como de trabajadores, y por tanto pone en riesgo la continuidad del negocio. Las empresas están expuestas a multitud de ataques que pueden comprometer los tres factores clave en la ciberseguridad: la integridad, disponibilidad y confidencialidad de la información.

Las principales amenazas en redes empresariales son [9][11]:

- Ataques de denegación de servicio (DDoS): Tiene como objetivo la saturación de un servidor o una red con la realización de un volumen masivo de solicitudes simultáneas, impidiendo que se puedan acceder a los diferentes servicios del servidor.
  Los atacantes utilizan una red de dispositivos infectados para enviar millones de peticiones por segundo, haciendo que se consuman los recursos del servidor.
  Existen varios tipos de ataques DDoS, como el volumétrico, en el que se envían grandes cantidades de datos para consumir el ancho de banda, el ataque de protocolo, en el que se aprovechan de una vulnerabilidad del protocolo de comunicación o el ataque a la capa de aplicación, en el que se simula tráfico real para sobrecargar servidores web o BBDD.
  En 2020, Amazon Web Services (AWS), alcanzó un tráfico de 2.3 terabits por segundo, siendo uno de los ataques DDoS más grandes de la historia.

- Escaneo de puertos: Los atacantes buscan los servicios que se utilizan en la empresa a través de los puertos abiertos, para identificar la versión de cada servicio junto a sus posibles vulnerabilidades.
  Escanean direcciones IP con herramientas como Nmap, una vez identificados los puertos analizan los servicios expuestos permitiendo a los atacantes entender la arquitectura de la empresa.

- Ataques de fuerza bruta: Intentos repetidos de acceso a la red, a través del servidor web, del servidor de directorio activo, por SSH… mediante la prueba masiva de diferentes credenciales.
  Los atacantes utilizan herramientas como Hydra o John The Ripper para probar una gran cantidad de credenciales por segundo. Estos ataques por fuerza bruta suelen realizarse con el uso de un diccionario, en el que se prueban contraseñas y combinaciones comunes.
  Estos ataques pueden mitigarse fácilmente implementando autenticación multifactor, bloqueando direcciones IP que intenten acceder tras varios inicios de sesión fallidos o utilizando contraseñas seguras.

- Filtración de datos sensibles: Una mala gestión de los datos, logeo en páginas poco fiables o malas prácticas de los usuarios pueden llevar a cabo filtraciones sensibles en los datos de la empresa.
  Cada poco tiempo surgen filtraciones de datos muy sonadas, como la de la DGT en España. Estas BBDD en la gran mayoría de casos acaban filtrándose en la Deep Web, haciendo que los atacantes tengan acceso a nuestra información privada para llevar a cabo ataques de phishing o de fuerza bruta.

- Phising: Los atacantes se hacen pasar por una persona o entidad legítima con el fin de obtener credenciales, pagos fraudulentos o recompensas del estilo.
  Suelen darse a través del email, enviándose un correo falso simulando ser una empresa de confianza. Este correo incluye un enlace a un sitio web

falso, donde se le pide a la víctima introducir unas credenciales o realizar un pago. También cabe la posibilidad de que se pida descargar un archivo importante, que en realidad contenga oculto un ransomware, el cual se instale en nuestro equipo y nos encripte los datos, pidiéndonos un rescate (también falso) para recuperarlos.

- Botnets: Tráfico automatizado de dispositivos infectados que buscan explotar fallos de seguridad o encontrar información sensible en páginas web mayoritariamente. Los atacantes infectan dispositivos mediante troyanos, los cuales hacen que, de manera silenciosa, el dispositivo realice lo que el ciberdelincuente decida.

El reto que se propone en este proyecto es el de la simulación de estos tipos de ataques comunes en una red empresarial. Como esto en un entorno real podría ser complicado, poniendo en riesgo los activos de la empresa, se propone la creación de un gemelo digital que simule la red, para así poder probar los ataques sin ningún tipo de problema. A esto se le desea integrar un modelo de Inteligencia Artificial que sea capaz de detectar anomalías y pueda aprender y adaptarse a patrones de tráfico sospechosos en tiempo real.

Para abordar estos problemas, el sistema debe cumplir con los siguientes requisitos, tanto funcionales como no funcionales. Dentro de los requisitos funcionales, el sistema debe ser capaz de:

- Simular una red empresarial realista, incluyendo servidores web, bases de datos, firewall y endpoints que simulen a los usuarios legítimos.
- Recolectar logs de tráfico en tiempo real desde múltiples fuentes.
- Detectar anomalías en el tráfico de la red mediante modelos de Machine Learning.
- Identificar patrones de ataques comunes como los mencionados anteriormente, deseablemente de manera automatizada con el modelo de ML.
- Automatizar la respuesta a incidentes, bloqueando las IPs sospechosas en el Firewall de la red.
- Visualizar estos eventos en gráficas en tiempo real.

En cuanto a los requisitos no funcionales, el sistema debe garantizar:

- Escalabilidad: Capacidad de añadir más servicios/dispositivos a la red cuando se desee.
- Personalización: El sistema debe permitir personalizar de manera realista cada servicio/dispositivo para que represente de la manera más realista posible la red real de la empresa.
- Portabilidad: Debe ser fácil desplegar en cualquier sitio el gemelo digital.
- Automatización: El sistema debe contar con mecanismos de autoaprendizaje y respuesta automática ante incidentes.

4. Diseño e implementación de la solución

4.1 Arquitectura General del Sistema

El sistema propuesto consiste en un gemelo digital de una red empresarial, donde se han desplegado diferentes servicios utilizando Docker. He escogido el uso de Docker por su fácil configuración, ya que con la creación de un simple archivo .yml se puede desplegar toda la red, así como por su eficiencia y poco uso de recursos.

Gracias a esta herramienta también se puede mover toda la infraestructura de una máquina a otra de una manera muy sencilla, pudiendo desplegar el gemelo digital donde se desee. Una vez creado el archivo, con un simple comando se desplegará toda la red.

Para que el sistema funcione correctamente, también se debe configurar cada contenedor de manera individual. Cada empresa tendrá unos dispositivos que se desplegarían con Docker, pero a su vez cada dispositivo tendría una configuración determinada que habría que replicar en el gemelo digital. En este apartado se despliega y configura una red básica.

Antes de comenzar a explicar por separado cada componente, se muestra en la imagen siguiente la arquitectura simplificada del sistema.

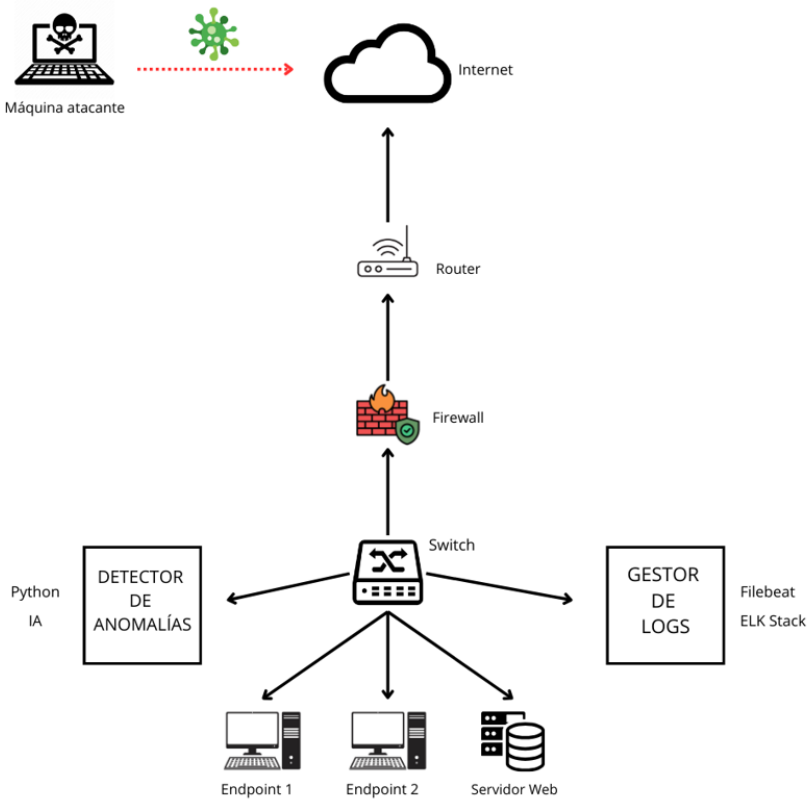

Figura 4: Arquitectura General

Los componentes principales de la arquitectura de manera desglosada son:

- Servidores web (NGINX y Apache)
- Base de datos MySQL
- Endpoints Ubuntu
- Firewall UFW
- Sistema de detección de anomalías (Python + IA)
- ELK Stack (Elasticsearch, Logstash, KIbana)
- Filebeat
- Grafana

A continuación, se describe cómo interactúan los diferentes componentes del sistema entre sí y el papel que toma cada uno:

1. Generación de tráfico: Se generar solicitudes legítimas y maliciosas desde los **endpoints** de la red y desde una máquina atacante.
2. Captura y almacenamiento de logs: **Filebeat** recoge los logs de NGINX, Apache y otros servicios. **Logstash** extra la información clave y lo envía a **Elasticsearch,** puediendo visualizar este tráfico en dashboards interactivos como **Kibana** o **Grafana.**
3. Análisis de logs: Creación de un script de análisis de tráfico con el uso de **Python e IA,** así como la creación de diversas funciones que detecten patrones anómalos que coincidan con ataques comunes.
4. Automatización de la respuesta: Se bloquea automáticamente la IP que se detecte como maliciosa en el **UFW** mediante SSH.
5. Visualización y monitoreo: Creación de gráficas en tiempo real que muestren las distintas IPs de la red, indicando si son anómalas o no.

4.2 Simulación de la red empresarial

Para simular la red y crear el gemelo digital, he utilizado Docker. Cada componente descrito anteriormente será un contenedor distinto que se configurará por separado.

A continuación, se detalla el contenido del archivo **docker-compose.yml**, que define y despliega los servicios de Docker que van a formar parte del gemelo digital, que incluye componentes básicos en una arquitectura de este estilo como son los servidores, el firewall, el SIEM y las herramientas de monitorización de la red.

Una vez este fichero de configuración está correctamente creado, lo único que hay que hacer es ejecutar el comando **docker compose up -d** para inicializar todas las máquinas. Con este comando se asegura la portabilidad del sistema, ya que con el archivo .yml se podrá desplegar todo donde se desee sin ningún inconveniente. También se podrá ampliar en cualquier momento asegurando la escalabilidad del mismo.

4.2.1 Creación de las redes de comunicaciones

En primer lugar, se definen las redes de la arquitectura. Para ello, se han definido dos, que he dividido en direcciones privadas en distintas subredes por simplicidad. Esto en una arquitectura real sería mucho más grande y complejo, dependiendo de la empresa en la que se vaya a implementar.

- **External-network:** Representa la red externa de la empresa, por donde puede entrar el tráfico malicioso.
- **Internal-network**: Esta red se encarga de conectar los distintos servicios internos entre sí sin exponerlos a la red externa.

```
networks:
  external-network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.22.1.0/24
  internal-network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.22.2.0/24
```

Figura 5: Configuración de las redes

4.2.2 Creación de los servidores web

He creado dos contenedores que representan los servidores web que podrían existir en una empresa, en este caso NGINX y Apache, a los cuales accederán tanto los usuarios de la empresa como los posibles atacantes.

Posteriormente, una vez creados los endpoints, se configurará un script para generar tráfico constante a este servidor. Esto es necesario no sólo para crear un escenario más realista, sino también para generar tráfico que sirva para entrenar el modelo de detección de anomalías.

He utilizado la imagen oficial tanto de NGINX como de Apache, exponiendo el puerto 80 para recibir el tráfico HTTP, asignando en la red interna unas IP estática 172.22.2.9 y 172.22.2.10 y montando en un directorio local ./nginx.conf un fichero de configuración y en un directorio local ./nginx-logs y ./apache-logs la recolección de logs de los dos servidores web.

Se utiliza la política restart always para reiniciar el contenedor independientemente del status de salida.

```yaml
nginx:
  image: nginx:latest
  container_name: nginx_server
  ports:
    - "8080:80"
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf
    - ./nginx-logs:/var/log/nginx
  networks:
    internal-network:
      ipv4_address: 172.22.2.9
  restart: always
```

Figura 6: Configuración del contenedor NGINX

```yaml
apache:
  image: httpd:latest
  container_name: apache_server
  ports:
    - "8081:80"
  volumes:
    - ./apache-logs:/var/log/apache2
  networks:
    internal-network:
      ipv4_address: 172.22.2.10
  restart: always
```

Figura 7: Configuración de Apache

4.2.3 Creación de la Base de Datos MySQL

Este servicio se encarga de almacenar los datos de la empresa y es el principal objetivo en ataques de tipo SQL Injection.

Se utiliza la imagen oficial de MySQL, se asigna una contraseña por defecto la cual se cambiará cuando entremos por primera vez en el contenedor. Se le asigna los puertos que se utiliza MySQL por defecto, que es el 3306, una IP estática y un volumen en el que se guardarán los logs, para analizarlos y saber si hay alguien no deseado intentando acceder.

```yaml
mysql:
  image: mysql:latest
  container_name: mysql_server
  environment:
    MYSQL_ROOT_PASSWORD: admin
  ports:
    - "3306:3306"
  volumes:
    - ./mysql-logs:/var/log/mysql
  networks:
    internal-network:
      ipv4_address: 172.22.2.11
  restart: always
```

Figura 8: Configuración de MySQL

4.2.4 Creación del Firewall UFW

Este contenedor va a simular un firewall que tendrá como objetivo principal el bloqueo de tráfico sospechoso.

Se utiliza una imagen de Ubuntu con UFW (Uncomplicated Firewall), que es un firewall con unos ajustes básicos pero que sirve perfectamente para simular el uso de un dispositivo real. Su objetivo es el de bloquear las IPs maliciosas o poner en cuarentena las que se consideren anómalas para que un administrador se encargue de investigar esa IP y decidir qué hacer con ella, trabajo que se suele desempeñar en un SOC.

Un SOC o Centro de Operaciones de Seguridad se dedica a monitorizar, detectar, analizar y responder en tiempo real a las amenazas de ciberseguridad que pueden ir surgiendo en las empresas.

En este contenedor también se han habilitado permisos especiales (NET_ADMIN) para manipular reglas de red.

Por otra parte, se ha creado un script de inicialización para que el firewall se inicie sin problema en caso de error, el cual básicamente actualiza e instala el firewall en el contenedor Ubuntu, crea unas reglas por defecto y habilita el UFW.

Por último, le he asignado dos IPs, una en la subred interna que tendrá visión de los contenedores internos de la red, y una externa que se encargará de entablar la conexión con Internet.

```yaml
ufw_firewall:
  image: ubuntu:latest
   container_name: ufw_firewall
   privileged: true
   cap_add:
     - NET_ADMIN
     - NET_RAW
   entrypoint: ["/usr/local/bin/start_ufw.sh"]
   volumes:
     - /home/adrian/elastic_stack/scripts/start_ufw.sh:/usr/local/bin/start_ufw.sh:ro
   networks:
     internal-network:
       ipv4_address: 172.22.2.2
     external-network:
       ipv4_address: 172.22.1.2
   restart: always
```

Figura 9: Configuración del firewall

```bash
#!/bin/bash
apt update && apt install -y ufw
# Aseguro que el firewall está configurado como necesito y habilitado
ufw default deny incoming
ufw default allow outgoing
ufw enable
```

Figura 10: Script de inicio de UFW

4.2.5 Creación de ELK (Elasticsearch, Logstash, Kibana)

Estos tres servicios son los encargados de centralizar los logs del tráfico de la red. El primer servicio es Elasticsearch, que utiliza la imagen oficial de Docker en la versión 7.15.0. Los puertos por defecto son el 9200 y el 9300. Por último, se utiliza una IP interna estática.

```
elasticsearch:
  image: docker.elastic.co/elasticsearch/elasticsearch:7.15.0
  container_name: elasticsearch
  environment:
    - discovery.type=single-node
  ports:
    - "9200:9200"
    - "9300:9300"
  networks:
    internal-network:
      ipv4_address: 172.22.2.4
  restart: always
```

Figura 11: Configuración de Elasticsearch

Por otro lado, está el servicio Logstash, que más adelante se explicará su configuración interna, pero de momento la configuración de la creación del contenedor con Docker es básica, utiliza la imagen oficial en la versión 7.15.0 como en Elasticsearch, se guarda un volumen en local con la configuración, se le asignan los puertos por defecto (5044 y 9600) y una IP interna estática.

```
logstash:
  image: docker.elastic.co/logstash/logstash:7.15.0
  container_name: logstash
  volumes:
    - ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf
  networks:
    internal-network:
      ipv4_address: 172.22.2.6
  ports:
    - "5044:5044"
    - "9600:9600"
  restart: always
```

Figura 12: Configuración de Logstash

Por último, en el contenedor de Kibana se ha vuelto a utilizar la imagen oficial de Docker en su versión 7.15.0, el puerto por defecto 5601 y una IP estática en la red interna de la arquitectura.

```
kibana:
  image: docker.elastic.co/kibana/kibana:7.15.0
  container_name: kibana
  ports:
    - "5601:5601"
  networks:
    internal-network:
      ipv4_address: 172.22.2.5
  depends_on:
    - elasticsearch
  restart: always
```

Figura 13: Configuración de Kibana

4.2.6 Creación de Grafana

Se utiliza este servicio para visualizar el tráfico en tiempo real. Como expliqué en la introducción, es algo parecido a lo que ofrece Kibana pero quizá algo más intuitivo y visual.

He utilizado la imagen oficial, le he asignado el puerto que suele utilizar por defecto que es el 3000, una variable de entorno con la contraseña por defecto del administrador, que una vez se entra es importante cambiarla. Por último, se le ha asignado una IP estática en la red interna.

```
grafana:
  image: grafana/grafana
  container_name: grafana
  ports:
    - "3000:3000"
  volumes:
    - grafana-data:/var/lib/grafana
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=admin
  networks:
    internal-network:
      ipv4_address: 172.22.2.8
  restart: always
```

Figura 14: Configuración de grafana

4.2.7 Creación de Filebeat

El servicio de Filebeat se ejecuta como un contenedor más dentro de Docker, utilizando su imagen oficial en la versión 7.10.0.

He montado múltiples volúmenes con las rutas para capturar los logs de NGINX, Apache y MySQL, lo que permite que Filebeat acceda a los logs de todos los servicios de la arquitectura en el gemelo digital.

También le he asignado una IP estática como en los demás contenedores, así como la creación de la directiva **depends_on,** que define las dependencias con los servicios de ELK (Elasticsearch, Logstash y Kibana).

```
filebeat:
  image: docker.elastic.co/beats/filebeat:7.10.0
  container_name: filebeat
  volumes:
    - ./filebeat.yml:/usr/share/filebeat/filebeat.yml:ro
    - ./nginx-logs:/usr/share/filebeat/nginx-logs
    - ./apache-logs:/usr/share/filebeat/apache-logs
    - ./mysql-logs:/usr/share/filebeat/mysql-logs
    - /var/lib/docker/containers:/var/lib/docker/containers:ro
    - /var/run/docker.sock:/var/run/docker.sock
  networks:
    internal-network:
      ipv4_address: 172.22.2.7
  depends_on:
    - logstash
    - elasticsearch
    - kibana
  restart: always
```

Figura 15: Configuración de filebeat

4.2.8 Creación de los endpoints

Esta es la última parte de la configuración del fichero **docker-compose.yml**, y es la encargada de crear los contenedores que van a simular los diferentes endpoints del sistema.

Para ello he creado 3 endpoints, dos usuarios normales que podrían ampliarse a los que se quiera (lo cual mejoraría la detección de anomalías del modelo Isolation Forest), pero que por simplicidad y ahorro de recursos se ha dejado en dos. El otro dispositivo será el del atacante, que se intentará colar entre el tráfico normal de la red.

Estos 2 endpoints propios de la empresa tienen una configuración muy similar, que se basa en una imagen Ubuntu, una IP estática en la red interna y dos scripts muy simples que se ejecutan al iniciar el contenedor.

```
ubuntu-endpoint:
  image: ubuntu
  container_name: ubuntu_endpoint1
  volumes:
    - /home/adrian/elastic_stack/scripts/start_ubuntu.sh:/usr/local/bin/start_ubuntu.sh:ro
    - /home/adrian/elastic_stack/scripts/trafico_bueno.sh:/usr/local/bin/trafico_bueno.sh:ro
  networks:
    internal-network:
      ipv4_address: 172.22.2.3
  entrypoint: ["/usr/local/bin/start_ubuntu.sh"]
  restart: always
```

Figura 16: Configuración de endpoint 1

```
ubuntu-endpoint2:
  image: ubuntu
  container_name: ubuntu_endpoint2
  volumes:
    - /home/adrian/elastic_stack/scripts/start_ubuntu.sh:/usr/local/bin/start_ubuntu.sh:ro
    - /home/adrian/elastic_stack/scripts/trafico_bueno.sh:/usr/local/bin/trafico_bueno.sh:ro
  networks:
    internal-network:
      ipv4_address: 172.22.2.12
  entrypoint: ["/usr/local/bin/start_ubuntu.sh"]
  restart: always
```

Figura 17: Configuración de endpoint 2

El primer script lo único que hace es un **apt update** para actualizar los paquetes en caso de que sea necesario e instala unas herramientas para la verificación de la red, que he utilizado para comprobar que todo funcionaba correctamente. Una vez hecho eso se llama al segundo script para que se ejecute.

```
#!/bin/bash
apt update && apt install -y curl net-tools nano traceroute
bash /usr/local/bin/trafico_bueno.sh
```

Figura 18: Script de inicialización de endpoints

El segundo script es un bucle infinito que realiza una petición a los servidores web NGINX y Apache cada 5 segundos de manera constante. Esto es importante para generar tráfico lícito en la red y entrenar el modelo de IA con logs no anómalos.

```
#!/bin/bash
while true; do
        curl http://172.22.2.9:80
        curl http://172.22.2.10:80
        sleep 5
done
```

Figura 19: Script de generación de tráfico lícito

Por último, el contenedor de la máquina atacante tiene una configuración básica que cuenta con una imagen Ubuntu, una dirección IP en la red externa y un comando cada vez que se inicia para que no se detenga por inactividad.

```
ubuntu-atacante:
  image: ubuntu
  container_name: ubuntu_atacante
  networks:
    external-network:
      ipv4_address: 172.22.1.2
  command: ["bash -c tail -f /dev/null"]
  restart: always
```

Figura 20: Configuración de la máquina atacante

4.3 Gestión de logs

Para entender lo que ocurre en la red y detectar las posibles amenazas que puedan aparecer, se ha implementado un sistema de gestión de logs, que recopila y analiza toda la actividad de la infraestructura.

El servidor web NGINX es el servicio principal que se va a utilizar en el proyecto. Los diferentes endpoints o máquinas atacantes realizarán peticiones sobre él. Filebeat recogerá los logs que genere el servidor web y lo enviará a Logstash.

Logstash será el encargado de procesar y transformar los datos en un formato útil para su análisis, y lo mandará a Elasticsearch. Desde Elasticsearch se almacenarán los logs en índice que podremos visualizar desde Kibana en la propia plataforma de Elastic o desde otra plataforma como es Grafana.

Todo esto es necesario para el posterior análisis de los logs, en el que se detectarán patrones sospechosos en el tráfico de la red.

Para el correcto funcionamiento del sistema, lo primero que se debe hacer es configurar cada servicio correctamente. Para ello, se ha accedido a cada contenedor de los creados anteriormente en el **docker-compose.yml** y se han configurado de manera que todo funcione como se desea. Los servicios que no se comenten a continuación, es porque no se ha tocado nada a destacar en los archivos de configuración y no es relevante para su funcionamiento.

Para acceder a un contenedor específico, como por ejemplo el del servidor web NGINX, que es el primero que voy a configurar, hay que ejecutar el comando **docker exec -it <nombre-contenedor> /bin/bash**, con esto podremos acceder a la terminal del servidor y configurarlo a nuestro gusto.

4.3.1 Configuración del servidor web NGINX

Este archivo de configuración sirve para modificar valores del servidor web y registrar accesos en los logs. Es la parte central del servidor y contiene una serie de directivas y parámetros que controlan su comportamiento y su interacción con los usuarios del sistema. El archivo de configuración se encuentra en la ruta /etc/nginx y su contenido es el siguiente.

Se define la estructura de los logs, que en el caso de este servidor web se divide en dos, los **access-log** y los **error_log**, y se especifica una ruta donde se desea guardar esos logs. Por otro lado, se configura el servidor para escuchar por el puerto 80, y también se define la raíz donde se guardarán los archivos de la web.

```
root@720e335b719b:/etc/nginx# cat nginx.conf
events {
        worker_connections 1024;
}

http {
        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;

        server {
                listen 80;

                location / {
                        root /usr/share/nginx/html;
                        index index.html index.htm;
                }
        }
}
```

Figura 21: Configuración servidor web NGINX

4.3.2 Configuración de Filebeat

Filebeat es el encargado de recolectar los logs de los diferentes servicios y enviarlos a Logstash o a Elasticsearch. La ruta del archivo de configuración se encuentra en **/etc/filebeat/filebeat.yml** y en el archivo se definen las rutas de los servicios que quiero que se recojan sus logs, modificando el atributo paths.

Para enviar estos logs a Logstash, se modifica el atributo output.logstash para indicar la dirección y el puerto donde se quiere enviar, en este caso como todo está en la misma red y en contenedores de Docker, con indicar el nombre del contenedor es suficiente, pero en un caso real habría que poner la dirección IP de la máquina que contiene Logstash y el puerto en el que se haya configurado, normalmente el 5044.

```
filebeat.inputs:
  - type: log
    enabled: true
    paths:
      - /usr/share/filebeat/nginx-logs/*.log
      - /usr/share/filebeat/apache-logs/*.log
      - /usr/share/filebeat/mysql-logs/*.log
      - /var/lib/docker/containers/*/*.log
      - /var/log/nginx/*.log
      - /var/log/mysql/*.log
    fields:
      log_type: service_logs

output.logstash:
  hosts: ["logstash:5044"]
```

Figura 22: Configuración de Filebeat

4.3.3 Configuración de Logstash

El objetivo de Logstash es el de recibir los logs que pasan por Filebeat y redirigirlos a Elasticsearch, para posteriormente ser procesados y enviados a Kibana, donde se podrán visualizar en diferentes cuadros de mando.

El archivo de configuración es simple, aunque se puede modificar mucho dependiendo de las necesidades, sobre todo en el apartado de **filters**, donde se pueden crear filtros y definir lo que se quiere ver de los logs o la información relevante a extraer.

En el atributo **input** se define el puerto por el que se van a recibir los logs, en este caso el 5044. El atributo **filter** lo he dejado vacío porque en mi caso no quiero extraer ninguna parte del log, me interesa tenerlo completo para visualizar toda la información posteriormente en Elasticsearch y extraer lo que me interese más adelante, durante la ejecución del programa principal que actúa como detector de anomalías.

Por último, en el apartado **output** se define la ruta a la que se mandan los logs procesados y que corresponden con la dirección IP y el puerto que especifiqué anteriormente para el contenedor de Elasticsearch. Se procesan con el índice filebeat-YYYY.MM.dd.

```
input {
        beats {
                port => 5044
        }
}

filter {
        #Agregar filtros segun necesidad
}

output {
        elasticsearch {
                hosts => ["http://172.22.2.4:9200"]
                index => "filebeat-%{+YYYY.MM.dd}"
        }
}
```

Figura 23: Configuración de Logstash

4.3.4 Configuración de Elasticsearch

Elasticsearch es el servicio encargado de almacenar y gestionar los logs indexados. En este contenedor no hay que configurar gran cosa, ya que, si los pasos anteriores están bien configurados, una vez entremos en la plataforma web ya debería aparecernos la página web en formato JSON indicándonos la información relevante de Elasticsearch.

Aun así, el archivo de configuración se encuentra en **/config/elasticsearch.yml** y contiene el nombre del cluster, y la dirección de red a la que se permite el acceso, en este caso 0.0.0.0 para que todos los servicios puedan acceder si se desea.

```
bash-4.4# cat elasticsearch.yml
cluster.name: "docker-cluster"
network.host: 0.0.0.0
```

Figura 24: Configuración de Elasticsearch

Para verificar que todo esté funcionando correctamente, antes de pasar a la ejecución de Kibana, lo que debemos hacer es entrar en la página de configuración de Elasticsearch indicando la dirección IP y el puerto en el que se ejecuta, en este caso sería http://172.22.2.4:9200.

Como se ve a continuación, Elasticsearch está configurado correctamente y listo para usarse.

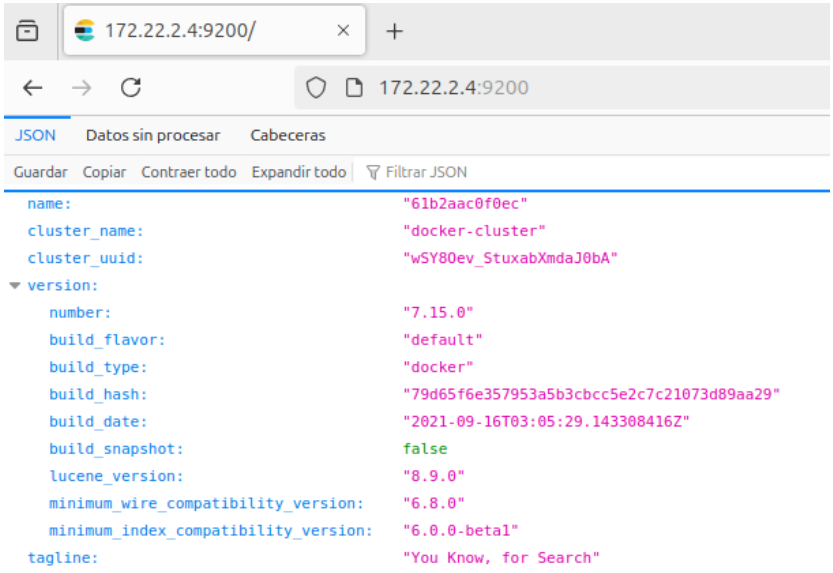

Figura 25: Página principal de Elasticsearch

4.3.5 Configuración de Kibana

El servicio de Kibana proporciona una visualización clara de los datos ingeridos desde Elasticsearch. Su archivo de configuración también está por defecto, y simplemente permite el acceso a su interfaz web desde cualquier host (“0”), y se conecta a la dirección del contenedor Elasticsearch por el puerto 9200, para recoger toda la ingesta de datos y mostrarlos con claridad.

```
# Default Kibana configuration for docker target
server.host: "0"
server.shutdownTimeout: "5s"
elasticsearch.hosts: [ "http://elasticsearch:9200"
monitoring.ui.container.elasticsearch.enabled: true
```

Figura 26: Configuración de Kibana

Para acceder a la plataforma web de ELK, debemos acceder a la dirección IP del contenedor y su puerto, en este caso http://172.22.2.5:5601, una vez dentro nos aparecerá una interfaz como la siguiente.

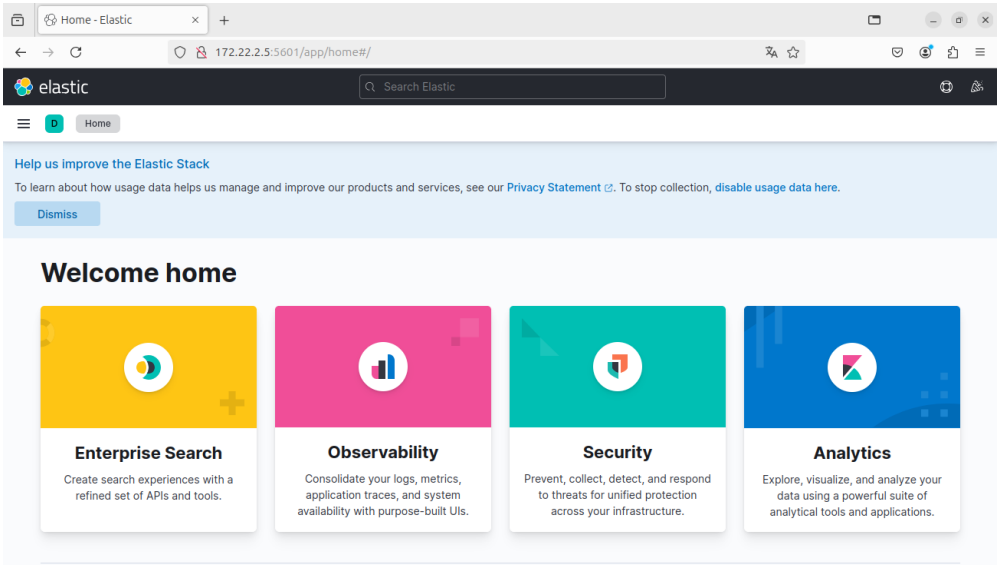

Figura 27: Interfaz principal de Elastic

La configuración básica de Kibana se encuentra en el apartado Analytics > Discover > Stack Management > Kibana > Index patterns. Una vez ahí debemos configurar el nombre del índice que hemos configurado anteriormente en logstash, que era filebeat-YYYY.MM.dd.

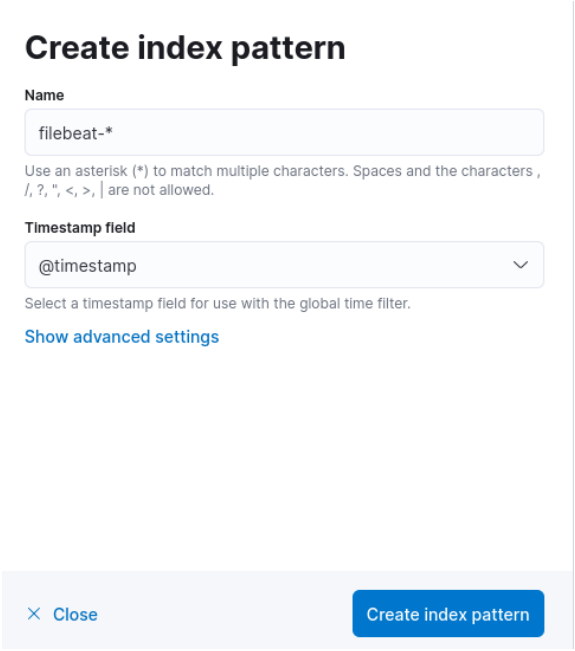

Figura 28: Creación del index pattern

Una vez creados el index pattern, si desplegamos el menú lateral, veremos una gran cantidad de opciones, pero para lo que realmente interesa en este TFG es suficiente con fijarse en el apartado de Analytics y en el Observability, ya que esto nos mostrará los logs que están apareciendo en el sistema. Para este caso utilizaremos las opciones que se pueden utilizar de manera gratuita en la plataforma, en este caso he utilizado dos de las múltiples opciones que proporciona ELK.

La primera de ellas es la del apartado **Log** en Observability. Este apartado nos enseñará los logs en tiempo real, junto a su información detallada si lo deseamos.

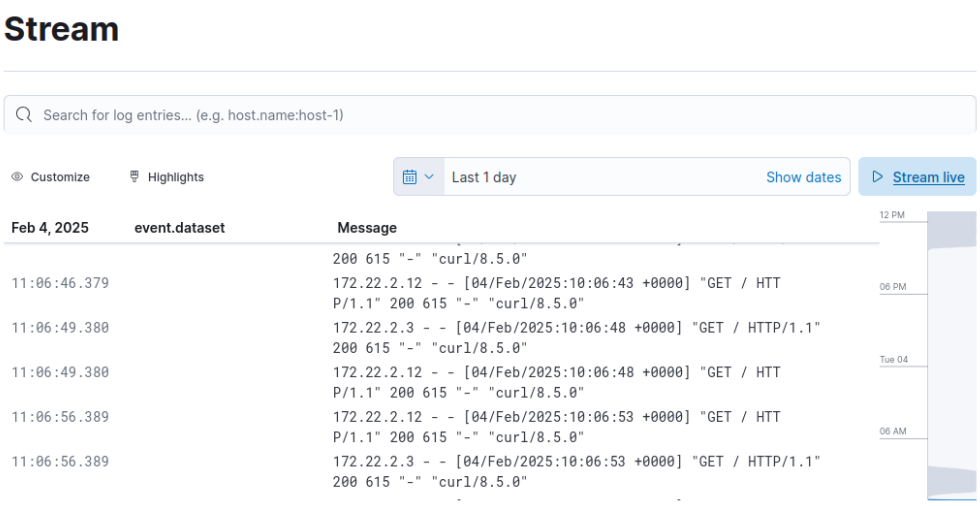

Figura 29: Stream en tiempo real de los logs

Si nos fijamos, en la información resumida de cada log aparece la fecha y el mensaje del log, que es lo más importante, lo utilizaremos más adelante para el análisis de anomalías.

Al hacer clic en cualquiera de los logs, podemos ver sus detalles, donde nos aparecerán las propiedades completas del log. Se nos ofrecen hasta 51 propiedades distintas del log. Vamos a visualizar la información detallada de un log procedente de la IP 172.22.2.12, que proviene de uno de los endpoints con un script programado para generar tráfico.

Para la finalidad del TFG sólo voy a hacer uso de los atributos más importantes, que son:

- @timestamp: fecha y hora en la que ha sido generado el log

Figura 30: Timestamp en log

- agent.type: el tipo de agente responsable de la recolección del log, en mi caso desde filebeat

Figura 31: Agent.type en log

- host.ip: la IP de procedencia del log, en este caso aparece la dirección IP del contenedor que actúa como filebeat, ya que es el intermediario que procesa los logs, pero más adelante en el mensaje aparecerá la verdadera IP de procedencia

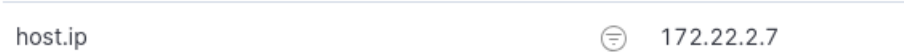

Figura 32: Host.ip en log

- log.file.path: la ruta de procedencia del log, en este caso es el access.log de nginx.

Figura 33: Log.file.path en log

- message: el mensaje que proporciona el servidor web en este caso sobre el log, en el que se especifica su dirección IP, le fecha, y la acción que ha realizado sobre el servidor

Figura 34: Message en log

La segunda de las opciones que he utilizado de la plataforma web para la visualización y análisis de los logs es la de Analytics > Discover. En esta pantalla me aparece una gráfica con los logs en cada instante de tiempo.

Como se puede ver en la imagen que adjunto a continuación, los logs se distribuyen más o menos por igual en todos los instantes de tiempo, esto es debido a la programación del script en los 2 endpoints que actúan como clientes, ya que generan la misma cantidad de logs en un intervalo de 5 segundos. En un escenario real esto no sería así, habría mucho más tráfico de logs a diferentes servicios.

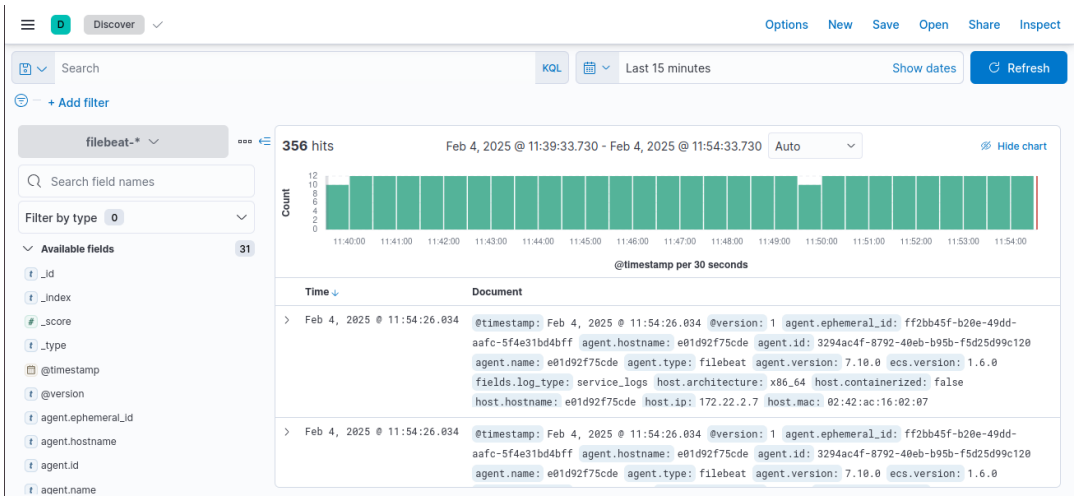

Figura 35: Gráfica de logs en tiempo real

En el panel de la izquierda se pueden observar los diferentes campos que hemos explicado anteriormente. En la parte superior la gráfica con los logs en cada instante de tiempo, en este caso cada 30 segundos, y en la parte central-inferior la información completa de cada log.

4.3.6 Configuración de Grafana

Grafana visualiza métricas y logs desde diferentes fuentes, es otra opción a tener en cuenta para la visualización de logs de una manera clara. La ubicación de su archivo de configuración es **/etc/grafana/grafana.ini**, y su configuración está por defecto, no hace falta ajustar nada en este archivo para que funcione correctamente, con configurar correctamente la plataforma web es más que suficiente.

Para configurar la plataforma web, accedemos con la IP del contenedor en el que esté instalado y el puerto, en este caso http://172.22.2.8:3000 , y en la barra lateral navegamos hasta Home > Connections > Add new connection y buscamos la herramienta con la que queramos integrar grafana, que como se puede ver hay una gran cantidad de ellas, unas 200 integraciones fácilmente.

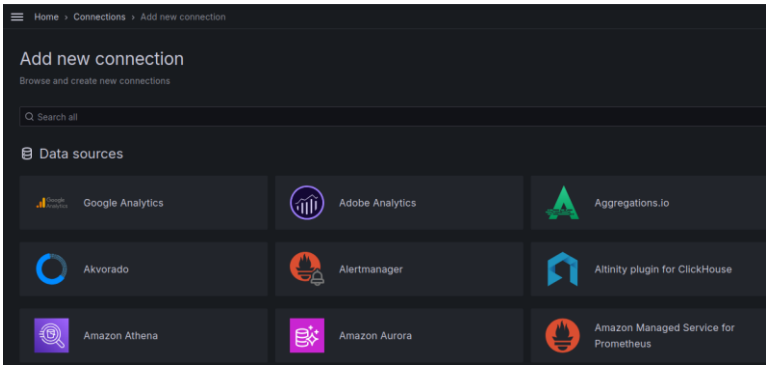

Figura 36. Integraciones en Grafana

En nuestro caso queremos integrarlo con Elasticsearch, por lo que buscamos por nombre y añadimos los campos que se nos requieren para la configuración, que son la URL en la que esté alojado elastic, la autenticación si la hay, y los detalles de la configuración de la fuente de datos.

Una vez configurado todo, podremos ver el dashboard de Elastic como veíamos antes en Kibana, pero con la posibilidad de cambiar entre tipos de gráficas, filtrar por un montón de opciones o integrarlo con cualquier otra herramienta.

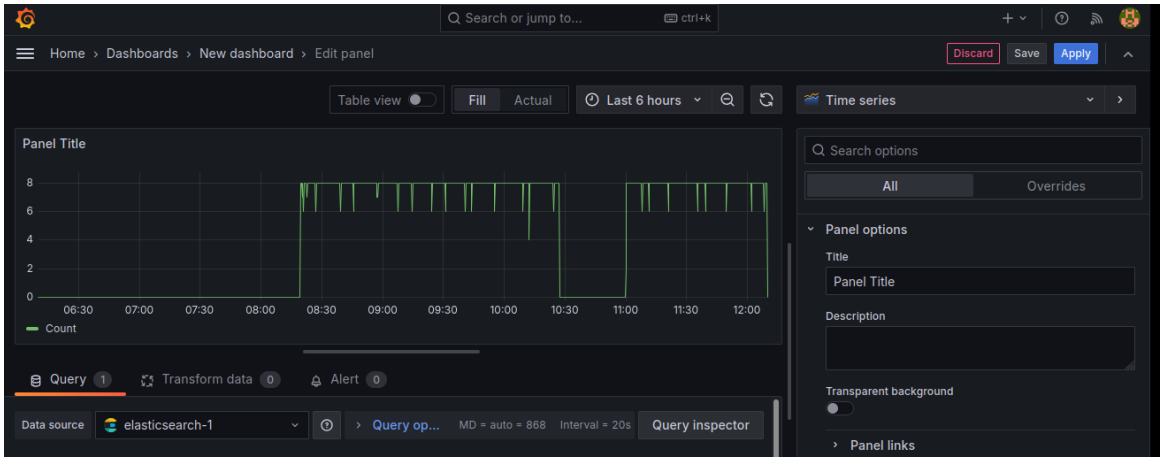

Figura 37: Dashboard en Grafana

4.4 Transformación de los logs

El objetivo de esta sección es transformar los logs brutos desde Elasticsearch en información útil para identificar posibles amenazas. Se van a utilizar distintas técnicas para analizar los registros que han ido generando el servidor web NGINX y así poder detectar tráfico sospechoso.

En primer lugar, se extraerá la información relevante del log, ya que de todos los campos que nos proporciona solo necesitaremos unos pocos.

Posteriormente se estudiará la frecuencia de las solicitudes y la actividad de cada IP para encontrar patrones inusuales.

Por último, se diferenciará entre tráfico legítimo y tráfico anómalo, para poder automatizar la respuesta al incidente y mostrar los datos en gráficas en tiempo real.

4.4.1 Librerías utilizadas

En este script he utilizado las siguientes librerías necesarias para el correcto funcionamiento del programa:

- Elasticsearch: Se utiliza para conectarse al servidor Elasticsearch y realizar las consultas sobre la ingesta de logs.
- Pandas: Permite manejar los datos estructurados en forma de DataFrames.

- Re: Se usa para trabajar con expresiones regulares y extraer la información que se desea del log.
- Time: Para medir y calcular distintos tiempos.
- IsolationForest: Es el algoritmo de machine learning que detecta anomalías en los logs.
- Matplotlib: Sirve para crear gráficas de todo tipo, en este caso serán gráficas en tiempo real.
- Numpy: Para manejar arrays y realizar operaciones matemáticas algo más complejas.
- Counter: Para contar elementos en listas.
- Scan: Es una herramienta de Elasticsearch muy útil, sirve para iterar sobre grandes volúmenes de datos.
- Paramiko: Biblioteca utilizada para establecer conexiones por SSH y ejecutar comandos de manera remota.

```
from elasticsearch import Elasticsearch
import pandas as pd
import re
import time
from sklearn.ensemble import IsolationForest
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import numpy as np
from collections import Counter
from elasticsearch.helpers import scan
import paramiko
```

Figura 38: Librerías utilizadas

4.4.2 Conexión a servicios

En esta parte del script se establecen las conexiones por SSH al firewall y por HTTP a Elasticsearch.

4.4.2.1 Conexión SSH al Firewall UFW

Se establece una conexión por SSH al firewall con dirección IP interna 172.22.2.2 para poder ejecutar comandos de manera remota y automatizada, en este caso se utilizará para enviar comandos de bloqueo de IPs anómalas.

Para ello, se ha configurado SSH en el contenedor que actúa como Firewall, asignándole un usuario y una contraseña. Lo ideal sería almacenar estas credenciales de manera cifrada, y no mostrarlo directamente en el código, pero por simplicidad y a modo de prueba se ha hecho de la siguiente manera la conexión.

```
# Configuración de conexión SSH
FIREWALL_IP = '172.22.2.2'
SSH_USERNAME = 'ufw'
SSH_PASSWORD = 'ufw1234'
```

Figura 39: Configuración de conexión SSH

4.4.2.2 Conexión a Elasticsearch

Por otra parte, se establece la conexión con Elasticsearch, donde se almacenan los logs del tráfico de la red que ha recolectado Filebeat y que ahora queremos utilizar y procesar para su posterior análisis. Para ello, se va a declarar una variable para la conexión con Elasticsearch y otra para indicar el índice que se está utilizando.

```
# Conexion a Elasticsearch
es = Elasticsearch(["http://172.22.2.4:9200"])
# Declaro el indice que estoy utilizando en Elasticsearch
index = 'filebeat-*'
```

Figura 40: Conexión con Elasticsearch

4.4.3 Obtención y procesamiento de los logs

En esta parte del script vamos a obtener los logs desde Elasticsearch y a procesarlos de la manera que más nos interesa utilizando regex para analizarlos después.

4.4.3.1 Obtención de los logs desde Elasticsearch

Para extraer los logs desde Elasticsearch, se ha creado una función que almacena los resultados tras un escaneo utilizando **scan** para obtener los últimos 10000 logs del tráfico generado.

Estos 10000 logs se almacenan por páginas, y una vez se completan estos 10000 logs, se pasa de página y se comienza de nuevo, por lo que el número de logs que se pueden analizar es ilimitado.

```
# Función para obtener los logs de Elasticsearch
def get_logs():
    logs = []
    results = scan(
        es,
        index='filebeat-*',
        query={"query": {"match_all": {}}},
        size=10000 # Número de logs por página
    )
```

Figura 41: Extracción de los logs

4.4.3.2 Extracción de la información relevante del log

De toda la información que nos proporciona Elasticsearch, se extrae lo que realmente nos puede hacer falta para el análisis, que como expliqué en el apartado 3.2.5, vamos a extraer la IP de origen, el timestamp, el método utilizado (GET, POST...), la URL solicitada, el código de estado y el tamaño de la respuesta.

Para ello he creado un patrón regex para extraer el log desde los resultados que almacené anteriormente, y con un bucle que recorre esos resultados añadir en un array de logs los campos que me interesan.

```
# Patrón regex para extraer la IP, fecha, método, URL y código de respuesta
log_pattern = r'(?P<ip>[\d\.]+) - - \[(?P<timestamp>[^\]]+)\] "(?P<method>\w+)(?P<url>[^\s]+) HTTP/[\d\.]+"(?P<status>\d+) (?P<size>\d+)'
```

Figura 42: Patrón regex

```
# Asumiendo que los resultados de Elasticsearch están almacenados en "results"
for hit in results:
    log_message = hit['_source']['message']

    match = re.search(log_pattern, log_message)
    if match:
        logs.append({
            "ip": match.group("ip"),
            "timestamp": match.group("timestamp"),
            "method": match.group("method"),
            "url": match.group("url"),
            "status": match.group("status"),
            "size": match.group("size"),
            "host_ip": hit['_source']['host']['ip'][0]  # IP del host del contenedor
        })
```

Figura 43: Obtención de información relevante del log

4.4.3.3 Transformación en DataFrames

Este paso es muy importante, ya que se necesita convertir las direcciones IP extraídas de los logs en valores numéricos. Esto facilitará el análisis posterior del modelo Isolation Forest.

Para ello se ha creado una función que se encarga de separar el conjunto de números entre ‘.’ y convertirlos en un número. Posteriormente, se crea un DataFrame con los logs que se han extraído y se le añade una columna con esta nueva IP numérica que será analizada.

```
# Función para convertir una IP en una lista de números
def ip_to_numeric(ip):
    parts = ip.split('.')
    return [int(part) for part in parts]

# Crear el DataFrame con los logs extraídos
df_logs = pd.DataFrame(logs)
# Creación de una nueva columna en el DataFrame con las IPs transformadas
df_logs['ip_numeric'] = df_logs['ip'].apply(ip_to_numeric)
```

Figura 44: Conversión de las IPs

4.5 Detección de las anomalías

En este apartado se van a analizar las anomalías que puedan surgir en la red. Para ello, se va a proceder a crear diversas funciones que se encarguen de ello. Estas funciones detectarán anomalías de dos formas, manual y automáticamente mediante Machine Learning.

4.5.1 Detectar posibles ataques de manera manual

Para detectar posibles ataques de manera manual, hay que conocer los ataques más frecuentes que pueden sufrir las infraestructuras de una empresa, ya mencionados en el apartado 1.1.

Para ello, el atacante va a realizar diferentes intentos de ataque. En primer lugar, va a realizar un ataque DDoS. Este se basa en el envío de una gran cantidad de tráfico malicioso a un servidor o red con el objetivo de saturarlo y que los usuarios dejen de poder utilizar el servicio.

Se suelen utilizar técnicas bastante elaboradas, una de las más comunes es la creación de una red de bots, que es un conjunto de dispositivos infectados con malware con el objetivo de ser controlados remotamente para acceder a un servidor todos los dispositivos al mismo tiempo y tumbarla.

Para detectar esto manualmente, he creado una función en el script en la que se considera una IP como sospechosa si realiza más de 100 peticiones al servidor web en un periodo de tiempo menor a 60 segundos.

```python
# Función para contar solicitudes por IP en un intervalo de tiempo (últimos X segundos=time_window)
def detect_traffic_spike(df_logs, time_window=60, request_treshold=100):
    anomalous_ips = []

    # Calcula las solicitudes por IP en la últim ventana de tiempo
    end_time = df_logs['timestamp'].max()
    start_time = end_time - time_window
    recent_logs = df_logs[(df_logs['timestamp'] >= start_time) & (df_logs['timestamp'] <= end_time)]

    ip_counts = recent_logs['ip'].value_counts()
    for ip, count in ip_counts.items():
        if count > request_treshold:
            anomalous_ips.append(ip)

    print(f"IPs con picos de tráfico detectadas: {anomalous_ips}")

    return anomalous_ips
```

Figura 45: Función para detectar ataque DDoS

En segundo lugar, otro de los ataques más comunes es el intento de acceso al servidor por fuerza bruta, ya sea intentando acceder a la cuenta de un usuario a través de un formulario de login o mediante técnicas más sofisticadas de acceso no autorizado al servidor web.

Para nuestro caso, lo que vamos a realizar es una comprobación de códigos de estado, en la que se comprobará si el valor del campo **status** dentro del DataFrame

coincide con los valores más frecuentes de intentos no autorizados o erróneos a alguna llamada al servidor. Estos códigos de estado son:

- 401 - Unauthorized: El cliente requiere de autorización para acceder al servidor.
- 403 - Forbidden: El cliente no posee los permisos necesarios para lo que quiere visualizar, por lo que el servidor le rechaza la petición.
- 500 - Internal Server Error: El servidor ha encontrado un error inesperado.

Si estos valores superan un límite, en este caso 20, se asignará como IP maliciosa la que quiera estar accediendo al servidor.

```
def detect_brute_force(df_logs, failed_login_treshold=20):
    failed_attempts = df_logs[df_logs['status'].isin(['401', '403', '500'])]
    brute_force_ips = failed_attempts['ip'].value_counts()
    attack_ips = brute_force_ips[brute_force_ips > failed_login_treshold].index.tolist()

    print(f"Posible ataque de fuerza bruta detectado")
    return attack_ips
```

Figura 46: Función para detectar ataques de fuerza bruta

Por último, otra de las comprobaciones que he realizado de manera manual es la de un posible ataque por una botnet. Una botnet es una red de ordenadores infectados con un tipo de malware que permite al atacante controlar el dispositivo remotamente. Las botnets usualmente intentan encontrar información privilegiada o accesos no autorizados en subdominios comunes.

En este caso he añadido unos subdominios comunes para comprobar, pero hay muchos más.

- Robots.txt: Este archivo contiene instrucciones que indican a los bots beneficiosos a qué páginas web pueden acceder. Se utiliza para que los rastreadores web analicen las páginas web e indexen el contenido para que aparezca en los motores de búsqueda. Esto puede ser aprovechado por los bots maliciosos y conseguir información privilegiada en una mala práctica.
- Admin: Suele ser un subdominio de logeo del admin, si el atacante consigue acceder por fuerza bruta o una mala configuración/vulnerabilidad del servidor, tendrá acceso total al mismo.
- Wp-login.php: Esto es un panel de logeo que se suele dejar visible en las páginas wordpress. Si se consigue acceder a él y logearse, se podrá modificar cualquier aspecto de la página.
- Login: Es un subdominio muy utilizado para acceder también a un panel de login, pero no necesariamente en una página hecha con wordpress.

En esta función, cualquier campo URL del log que contenga cualquier subdominio de los mencionados, se marcará como IP maliciosa.

```
suspicious_urls = ['/robots.txt', '/admin', '/wp-login.php', '/login']

def detect_bots(df_logs):
    bot_ips = df_logs[df_logs['url'].isin(suspicious_urls)]['ip'].unique().tolist()

    print(f"Posible actividad de bot detectada")
    return bot_ips
```

Figura 47: Función para detectar botnets

4.5.2 Aplicar Machine Learning con Isolation Forest

Una vez realizadas estas comprobaciones manuales, pasamos a la automatización de IP anómalas utilizando el modelo Isolation Forest. Su modo de funcionamiento es muy sencillo, se basa en el entrenamiento del modelo con logs legítimos, es decir, ejecutar una función que entrene en tiempo real al modelo con DataFrames legítimos, en este caso vamos a entrenarle con el campo IP del DataFrame.

Dejaremos que se genera tráfico durante un tiempo en la red, y una vez hecho esto, el modelo estará entrenado para encontrar anomalías. Se le puede agregar un parámetro llamado **contamination**, en el que se estima el porcentaje de logs que se espera encontrar como anómalas, lo cual debe ser muy bajo, ya que el porcentaje de logs anómalos en una red con mucho tráfico no suele ser mayor de un 2%.

Una vez configurado el modelo, se guarda en otro campo del DataFrame los logs anómalos detectados por el modelo de forma automatizada. Estos logs anómalos serán los que el modelo establece con valor -1.

```
# Creación de un DataFrame de las IPs numéricas
ip_df = pd.DataFrame(df_logs['ip_numeric'].tolist(), columns=['octet1', 'octet2', 'octet3', 'octet4'])

# Aplico IsolationForest para la detección de anomalías
model = IsolationForest(contamination=0.02) # El 2% se estima que son anomalías
df_logs['anomaly'] = model.fit_predict(ip_df)
# Cuento IPs anómalas y normales
normal_ips = df_logs[df_logs['anomaly'] == 1]['ip'].value_counts()
isolation_anomalous_ips = df_logs[df_logs['anomaly'] == -1]['ip'].value_counts()
```

Figura 48: Uso del modelo Isolation Forest

4.5.3 Unificación de detecciones

Tras la implementación de estas funciones, es el momento de unificar los resultados y bloquear los logs que se consideren maliciosos. Para ello, se unen todos los resultados considerados maliciosos (anomalous_ips, attack_ips, bot_ips, isolation_anomalous_ips).

Una vez hecho esto ya se podrá comenzar con el bloqueo de estas IPs de manera automatizada en el firewall de la infraestructura.

```
all_malicious_ips = set(anomalous_ips + attack_ips + bot_ips + isolation_anomalous_ips)
```

Figura 49: Unificación de las IPs anómalas

4.6 Respuesta a incidentes de manera automatizada

Una vez se detecta un ataque, el sistema debe reaccionar rápidamente para evitar daños. Para ello, se ha implementado una respuesta ante incidentes de manera automatizada basada en el bloqueo de direcciones IP sospechosas.

Con todas las IPs maliciosas detectadas, el último paso que queda es el de la automatización del bloqueo de la misma. Para ello, he creado una función que se conecta al Firewall por SSH para poder ejecutar comandos de manera remota.

Para realizar los bloqueos, es suficiente con ejecutar el comando **ufw deny from <ip_anómala>.** Con esto, esa IP cuando intente pasar de nuevo por el firewall para acceder al servidor web, se bloqueará y no podrá acceder.

```python
# Función para bloquear la IP en el firewall UFW
def bloquear_ip(all_malicious_ips):
    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(FIREWALL_IP, username=SSH_USERNAME, password=SSH_PASSWORD)

        # Ejecutar el comando UFW para bloquear la IP
        comando = f"sudo ufw deny from {all_malicious_ips}"
        stdin, stdout, stderr = ssh.exec_command(comando)

        # Confirmación en consola
        print(f"Bloqueo de IP {all_malicious_ips} ejecutado: {stdout.read().decode()}")

        ssh.close()
    except Exception as e:
        print(f"Error al intentar bloquear la IP {all_malicious_ips}: {e}")
```

Figura 50: Función de bloqueo de IPs maliciosas

5. Pruebas y resultados obtenidos

Uno de los aspectos clave de este proyecto es la monitorización del tráfico de la red de manera continua para detectar anomalías en tiempo real. Para ello se han creado gráficos dinámicos que se actualizan de manera constante con los datos más recientes obtenidos de los logs.

Una vez creado el script, sólo falta el último paso, la realización de las diferentes pruebas y la comprobación de los resultados. Para ello, he generado diferentes gráficas para comprobar paso a paso lo que he ido realizando en el script. Todas estas gráficas recogen y analizan los logs en tiempo real, mostrando las IPs que están accediendo al servidor web NGINX y el número de solicitudes que están realizando.

En primer lugar, la siguiente gráfica muestra la red en un estado normal, es decir, cuando los endpoints 172.22.2.3 y 172.22.2.12 están haciendo solicitudes normales al servidor. Como ninguna función ni el modelo Isolation Forest lo detectan como algo anómalo, se muestran en color verde.

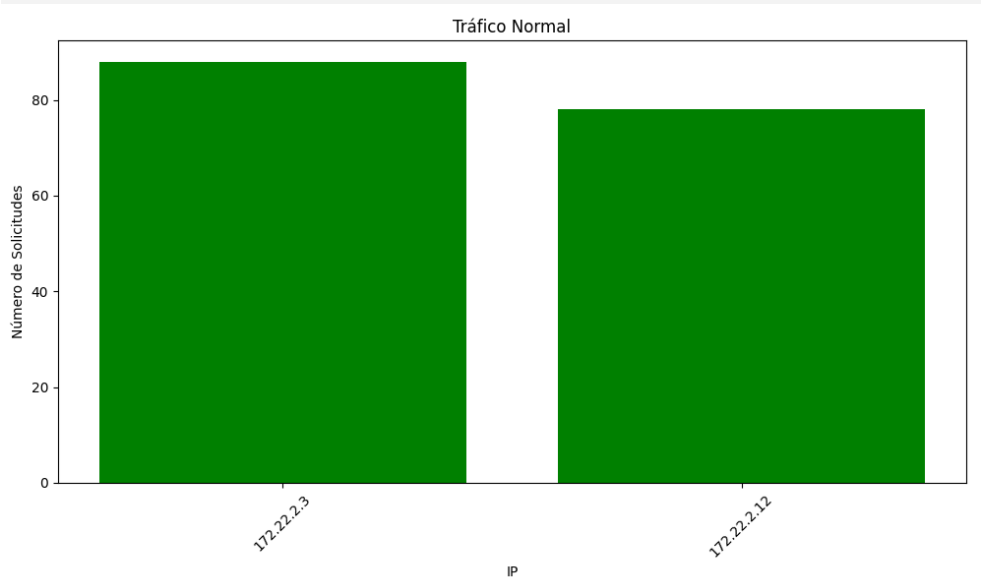

Figura 51: Gráfica 1 - Tráfico legítimo

En segundo lugar, se muestra en la siguiente gráfica el caso en el que, desde la IP de la máquina atacante creada anteriormente, se realiza un ataque DDoS, para ello he generado mucho tráfico con el uso del comando **curl.** Como se puede ver en la gráfica, aparece marcada en rojo la IP de la máquina atacante junto al número de solicitudes al servidor, que supera el límite de 100 que establecí en la función de detección.

Se puede simular este ataque con un comando tal que:

for i in {1..800}; do curl -s http://<dirección-servidor> & done

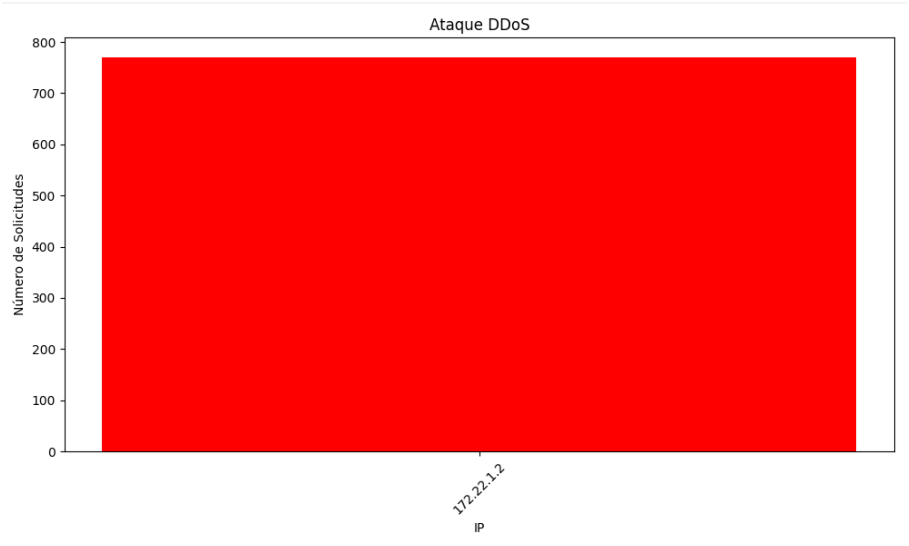

Figura 52: Gráfica 2 - Ataque DDoS

En tercer lugar, se comprueba el ataque de fuerza bruta al servidor, para ello, he simulado también con el comando **curl** la petición de unas cuantas solicitudes para que me devuelva códigos de estado de no autorización, como el 401 o el 403. Una vez se sobrepasa el límite establecido, esta IP se actualiza en la gráfica y se colorea de color rojo.

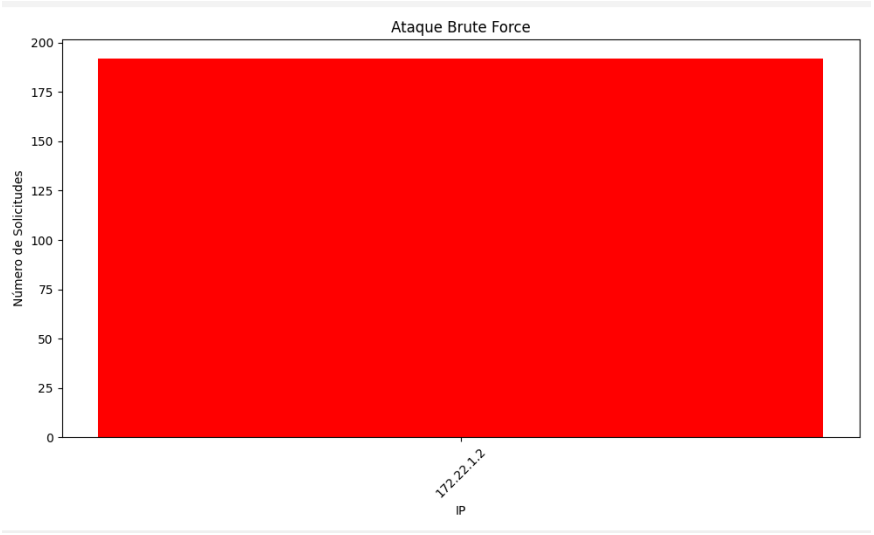

Figura 53: Gráfica 3 - Ataque de fuerza bruta

En cuarto lugar, he simulado un ataque botnet. Para ello, he creado otros 3 contenedores más con una configuración similar al de la máquina atacante.

Con ello, he realizado solicitudes también con **curl** a los diferentes subdominios que establecí como peligrosos.

Una vez realizadas las solicitudes, en la gráfica de ataque botnet aparece en tiempo real las solicitudes que han realizado cada dirección IP atacante a nuestro servidor.

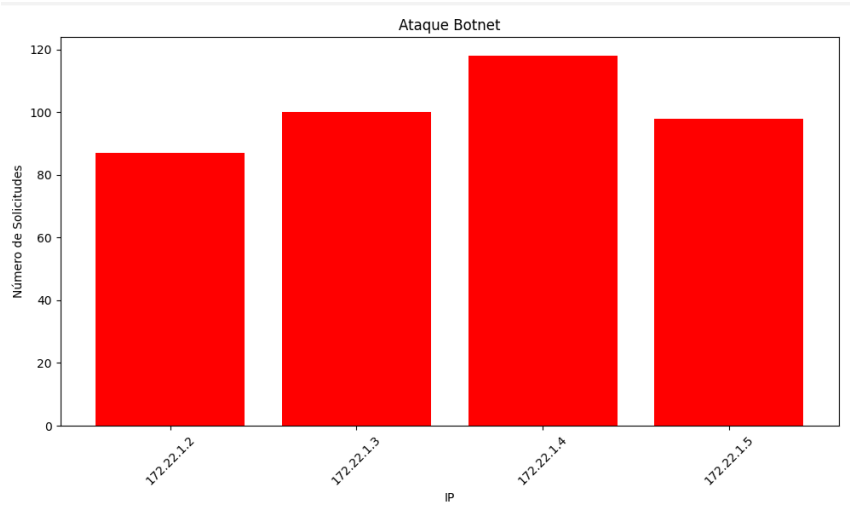

Figura 54: Gráfica 4 - Ataque Botnet

En quinto lugar, se muestra la gráfica con la detección del modelo Isolation Forest, en el que se deja durante un tiempo que los endpoints manden solicitudes al servidor web, y una vez entrenado en modelo con estas solicitudes legítimas, al intentar acceder desde otro dispositivo, en este caso el atacante, el modelo lo detectará como una IP anómala y lo marcará de color rojo.

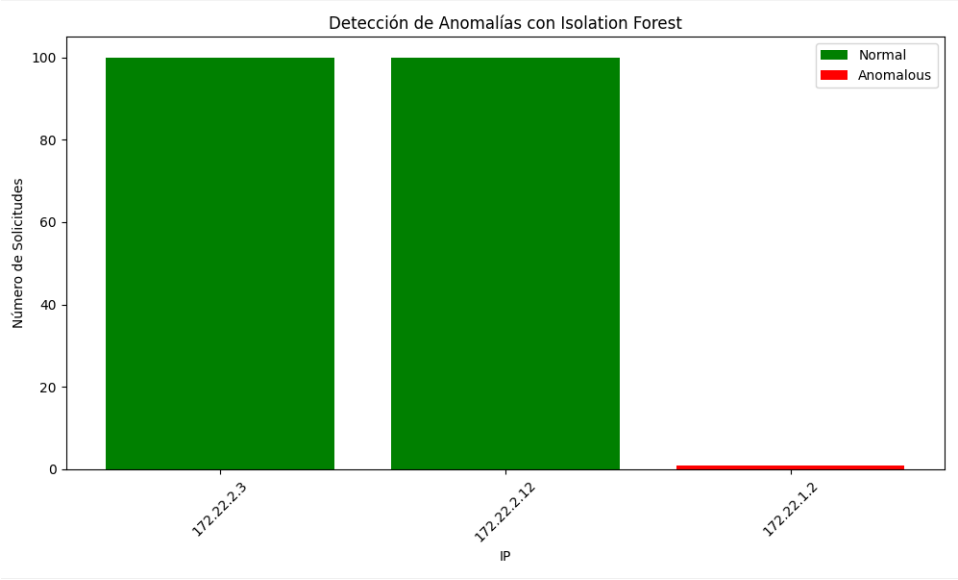

Figura 55: Gráfica 5 - Detección con Isolation Forest

Por último, se muestra una gráfica comparativa de los diferentes ataques que se han realizado desde cada dirección IP y también el tráfico legítimo en color verde. Todos esto datos se van actualizando en tiempo real en cada una de las gráficas conforme llegan los datos ingeridos de Elasticsearch.

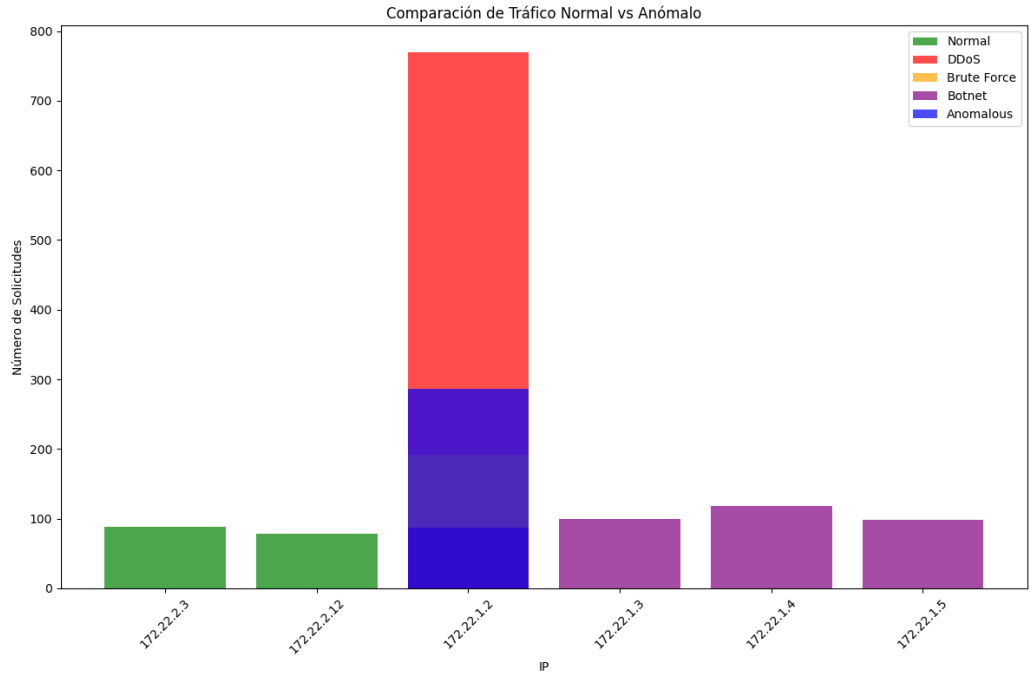

Figura 56: Gráfica 6 - Comparación final

6. Conclusiones y trabajo futuro

La implementación de un gemelo digital en una red simulada me ha permitido investigar y aprender sobre la eficacia de la combinación de la inteligencia artificial y de los métodos de detección de ataques junto al monitoreo en tiempo real para la detección y mitigación de las amenazas de ciberseguridad.

Considero que durante la realización del trabajo he conseguido cumplir una serie de objetivos que había establecido junto a mi director de la empresa Ángel, como son:

- Se ha simulado una red empresarial simplificada con diferentes servicios como servidores web, bases de datos, endpoints, firewall.
- Se han recolectado y analizado los logs mediante herramientas como Filebeat, Logstash y Elasticsearch.
- Se ha visualizado los logs integrando herramientas como Kibana y Grafana.
- Se han aplicado diferentes técnicas de detección de anomalías, tanto manuales como automáticas utilizando IA.
- Se ha automatizado la respuesta a incidentes mediante el bloqueo de las IPs anómalas en el firewall de la red.
- Se ha visualizado en tiempo real de los resultados obtenidos mediante la generación de gráficas.

Creo que este proyecto me ha aportado muchos nuevos conocimientos, y que estas simulaciones simplificadas se pueden llevar a las grandes empresas, recreando de manera virtual con más recursos y el uso de máquinas virtuales personalizadas la red real de cualquier cliente, pudiendo simular un ataque para comprobar la respuesta de la infraestructura sin afectar a la red real.

Al utilizar máquinas virtuales, podrían simularse cualquier tipo de equipo con la configuración exacta, simulando de manera muy precisa el escenario real. Esto unido a la automatización, reduciría en gran medida la intervención del humano en labores de detección y mitigación de amenazas.

También se han encontrado otras limitaciones y posibles mejoras, como el uso de un modelo de IA superior, al que se le puedan pasar más cantidad de datos, lo cual consume más recursos, pero ayuda a una detección mucho más precisa.

Se desea añadir en un futuro nuevas detecciones a más tipos de ataques y más sofisticados, intentando que todo esto se automatice con el uso de la inteligencia artificial, sin ser necesaria la creación de funciones por cada tipo de ataque.

7. Bibliografía

1. de Prada, C., Galán-Casado, S., Pitarch, J. L., Sarabia, D., Galán, A., & Gutiérrez, G. (2022). Gemelos digitales en la industria de procesos. *Revista Iberoamericana de Automática e Informática industrial*, *19*(3), 285-296.
2. Walters, B. (1999). VMware virtual platform. *Linux journal*, *1999*(63es), 6-es.
3. DOCKER, Inc. Docker. *línea].[Junio de 2017]. Disponible en: https://www. docker. com/what-docker*, 2020.
4. STOLERIU, Răzvan; PUNCIOIU, Alin; BICA, Ion. Cyber attacks detection using open source elk stack. En *2021 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. IEEE, 2021. p. 1-6.

   CIVANTOS MARTOS, Carla, et al. *Uso y ventajas de Elasticsearch en bases de datos no relacionales*. 2019. Tesis de Licenciatura.

   ARANDA, Nelson; AGUIRRE, Nestor; BALICH, Néstor. Plataforma internet industrial de las cosas como servicio local. *INNOVA UNTREF. Revista Argentina de Ciencia y Tecnología*, 2022.

   CARRIÓN RAMÍREZ, Byron. Diseño e implementación de una solución de gestión centralizada de logs de aplicaciones, sistemas y dispositivos basada en Logstash. 2015.

5. CHAKRABORTY, Mainak; KUNDAN, Ajit Pratap. Grafana. En *Monitoring cloud-native applications: Lead agile operations confidently using open source softwar*e. Berkeley, CA: Apress, 2021. p. 187-240.
6. DIOTALEVI, Tommaso, et al. Collection and harmonization of system logs and prototypal Analytics services with the Elastic (ELK) suite at the INFN-CNAF computing centre. *arXiv preprint arXiv:2106.02612*, 2021.
7. CHALLENGER-PÉREZ, Ivet; DÍAZ-RICARDO, Yanet; BECERRA-GARCÍA, Roberto Antonio. El lenguaje de programación Python. *Ciencias Holguín*, 2014, vol. 20, no 2, p. 1-13.
8. DÍAZ CERVANTES, Lisset. Evaluación de la herramienta GNS3 con conectividad a enrutadores reales. 2010.
9. AGUILAR, Luis Joyanes. Introducción. Estado del arte de la ciberseguridad. *Cuadernos de estrategia*, 2011, no 149, p. 11-46.
10. ROUHIAINEN, Lasse. Inteligencia artificial. *Madrid: Alienta Editorial*, 2018, p. 20-21.
11. GUAÑA MOYA, Edison Javier, et al. Ataques informáticos más comunes en el mundo digitalizado. 2022.