Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño e implementación de un sistema centralizado de gestión de Indicadores de Compromiso para su exportación a detectores de intrusos de red

Design and implementation of a centralized Management System for Indicators of Compromise and their export to Network Intrusion Detectors

Autor

Alberto Inés Medina

Director

Álvaro Alesanco Iglesias

Ingeniería de Tecnologías y Servicios de Telecomunicación

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2023

Abstract

En una era digital creciente en complejidad y en constante evolución, este trabajo supone una contribución más al fortalecimiento de la ciberseguridad, aportando una solución en materia de detección de ciberamenazas en entornos de red. En este contexto, se desarrolla una infraestructura automatizada conformada por distintos sistemas independientes que, en conjunto, resuelven la necesidad de detectar incidentes de seguridad en entornos de red domiciliarios y pequeñas y medianas empresas.

El sistema implementado utiliza los Indicadores de Compromiso (IOCs), que son la unidad básica de información que describe un incidente de seguridad. Con estos IOCs, se crean reglas de detección para posteriormente crear ficheros con dichas reglas y exportarlos hacia detectores de intrusos de red (NIDS). Los NIDS se encargan de leer los ficheros de reglas para realizar la tarea de detección de conexiones maliciosas en equipos y redes. Se seleccionan dos NIDS de código abierto ampliamente reconocidos: *Suricata* y *Zeek*. Además se necesita de un plataforma de gestión y exportación de IOCs hacia los detectores de intrusos. Se elige *MISP*, un *software* de código abierto que permite la importación de IOCs desde diferentes fuentes *Open Source*, además de su almacenamiento en una base de datos y exportación en el formato correcto hacia los NIDS. Por último, se centralizan los *logs* (alertas) generados por los detectores en una solución de código abierto, *Elastic Stack*, con el objetivo de visualizar dicha información. El sistema se pondrá a prueba en un entorno de red doméstico real, con el fin de analizar todo el tráfico de la red y detectar posibles ciberamenazas.

Abstract

In an increasingly complex and ever-evolving digital era, this work represents another contribution to strengthening cybersecurity by providing a solution for detecting cyber-threats on network enviroments. In this context, an automated infrastructure composed of various independent systems is developed, which together address the need to detect security incidents in home network environments and small to medium-sized businesses.

The implemented system uses Indicators of Compromise (IOCs), which are the basic unit of information that describes a security incident. With these IOCs, detection rules are created to subsequently generate files containing these rules and export them to Network Intrusion Detection Systems (NIDS). NIDS are responsible for reading the rule files to perform the task of detecting malicious connections on computers and networks. Two widely recognized open-source NIDS are selected: *Suricata* and *Zeek*. Additionally, a platform for managing and exporting IOCs to the intrusion detectors is required. *MISP* is chosen, an open-source software that allows the import of IOCs from various *Open Source* sources, as well as their storage in a database and proper format export to the NIDS. Finally, the logs (alerts) generated by the detectors are centralized in an open-source solution, *Elastic Stack*, for the purpose of visualizing this information. The system will be tested in a real home network environment to analyze all network traffic and detect potential cyber threats.

Índice

Lista de acrónimos

IOC: Indicator of Compromise
IDS: Intrusion Detection System
NIDS: Network Intrusion Detection System
MISP: Malware Information Sharing Platform
MV: Máquina Virtual
VPN: Virtual Private Network
NAT: Network Address Translation
SSH: Secure Shell
API: Application Programming Interface
HTTP: Hypertext Transfer Protocol
SSL: Secure Socket Layer
TLS: Transport Layer Security

Capítulo 1

Introducción

1.1. Contexto y motivación

Según informa *El País* [1], el Instituto Nacional de Ciberseguridad (INCIBE) gestionó 118,820 incidentes relacionados con la seguridad en la red durante 2022, lo que representó un aumento del 8.8 % en comparación con 2021. Expertos en la materia sostienen que esta cifra continuará en aumento en los años venideros.

En la actualidad, existen diversas instituciones, organismos y empresas, tanto públicas (CERT, CCN-CERT, ENISA, NIST) como privadas (VirusTotal, Fortinet), cuyo propósito es proporcionar una respuesta frente a incidentes en el ámbito de la ciberseguridad. Estas entidades colaboran para establecer estándares, certificaciones y marcos de referencia comunes, con el objetivo de asegurar una protección efectiva contra ciberataques.

De acuerdo con el Instituto Nacional de Estándares y Tecnología (NIST) [2], la respuesta ante incidentes se divide en cuatro fases: preparación y prevención; detección y análisis; contención, erradicación y recuperación; y actividad posterior al incidente. El concepto central del proyecto surge como aplicación directa de la segunda fase, concentrándose en la subfase de detección. En esta etapa surge la idea de utilizar **IDSs**[1] como herramientas esenciales para el rastreo de ciberamenazas. Se seleccionan dos IDS de código abierto ampliamente reconocidos: *Suricata* [3] y *Zeek* [4]. Ambos programas pertenecen a la categoría de detectores de intrusos de red, conocidos como **NIDS** (*Network Intrusion Detection System*). Estas herramientas tienen la capacidad de detectar tráfico de red malicioso, pero requieren información precisa para llevar a cabo su función. Para ello, se utilizan los Indicadores de Compromiso (**IOCs**) [5], que son la unidad básica de información que describe un incidente de ciberseguridad. Los Indicadores de Compromiso abarcan desde direcciones IP hasta patrones de ejecución

[1]IDS: *Intrusion Detection System*. Software de seguridad cuya función es detectar accesos no autorizados en sistemas o redes de computadores y generar algún tipo de alerta hacia el administrador del sistema o red.

de comandos, por lo que carecen de una estructura definida.

Los IOCs se utilizan como base para crear reglas, que a su vez alimentarán a los IDSs. Por tanto, es necesario contar con una herramienta que centralice la gestión de estos IOCs y exporte las reglas de detección en el formato adecuado para los IDS seleccionados. La plataforma *MISP* [6] proporciona las capacidades necesarias para gestionar eficientemente los IOCs y exportar las reglas a los IDS escogidos. Además, con el objetivo de facilitar la administración del sistema, se propone almacenar toda la información generada (alertas, registros, etc.) por los NIDS en una plataforma para su posterior consulta y análisis. Para esta tarea, se ha optado por el uso de *Elastic Stack* [7], una solución de código abierto.

1.2. Objetivos

El objetivo final del proyecto es el despliegue de un sistema de gestión de Indicadores de Compromiso automatizado para alimentar sistemas de detección de intrusos en entornos domiciliarios o en pequeñas y medianas empresas [8]. El sistema se actualizará diariamente con IOCs provenientes de diversas fuentes públicas, centralizados en la plataforma MISP. Desde MISP, se exportarán a su vez diariamente como reglas a los detectores de intrusos *Suricata* y *Zeek*. La información generada por estos detectores se centralizará en Elastic Stack.

En cuanto a objetivos específicos, se encuentran los siguientes:

- Administración de sistemas GNU/Linux en red basados en la distribución Debian mediante el uso de máquinas virtuales (MVs).
- Estudio, configuración e implementación de la plataforma MISP en una máquina virtual.
- Estudio y elección de fuentes de datos *Open Source* para alimentar la plataforma MISP.
- Análisis y procesado de datos para su posterior exportación hacia detectores de intrusos de red.
- Estudio, configuración y despliegue de los detectores de intrusos *Suricata* y *Zeek* para el análisis de los datos procedentes de la plataforma MISP.
- Estudio, configuración e implementación del software Elastic Stack para recoger las alertas generadas por los IDS y visualizarlas de forma eficiente.
- Despliegue y pruebas de la infraestructura en un entorno con máquinas virtuales.

– Estudio y configuración del hardware Raspberry Pi 4 como analizador de tráfico en una red domiciliaria real.

– Despliegue y pruebas de la infraestructura en un entorno domiciliario real mediante una Raspberry Pi 4.

Con este Trabajo Fin de Grado, se pretende contribuir al fortalecimiento de la ciberseguridad mediante el desarrollo de una solución práctica y efectiva, mejorando así la capacidad de detección y respuesta ante posibles incidentes de seguridad. Esto ayuda a proteger la integridad y confidencialidad de la información en un entorno digital en constante evolución y creciente complejidad.

1.3. Herramientas

Los recursos y herramientas utilizados durante el desarrollo del proyecto se dividen en recursos hardware y recursos software.

– Recursos hardware:

  - **Ordenador personal**: Ordenador utilizado para la gestión de cada uno de los recursos software de forma sencilla mediante conexiones Secure Shell (**SSH**).
  - **Servidor privado Unizar**: Para acceder desde un ordenador personal, se debe realizar una conexión por **VPN**[2]
  - **Raspberry Pi 4**: Computador básico con sistema operativo de código abierto (Raspbian) utilizado para la realización de pruebas de detección en redes domésticas. El dispositivo cuenta con un puerto Ethernet de 1Gbps de ancho de banda además de una tarjeta de red Wifi.

– Recursos software: Se hace uso de máquinas virtuales Debian (versión 11), alojadas en un servidor privado de la Universidad de Zaragoza. Se accede a cada una de ellas mediante conexiones SSH.

  - **Máquina virtual MISP**: Máquina virtual donde se instala el software de código abierto MISP para la gestión centralizada de Indicadores de Compromiso. Esta máquina posee un almacenamiento superior a las demás (140 GB), puesto que debe manejar grandes volúmenes de IOCs y datos relacionados.

[2]VPN: *Virtual Private Network*. Tecnología de red que permite establecer una conexión encriptada y segura entre dos equipos a través de una red pública como Internet, garantizando la privacidad y la integridad de los datos.

- **Máquina virtual Zeek/Suricata**: Máquina virtual donde se instalan los NIDS *Suricata* y *Zeek* para realizar la tarea de detección de ciberamenazas en la red y generación de alertas.
- **Máquina virtual Elastic Stack**: Máquina virtual donde se instala el software de código abierto Elastic Stack para visualizar toda la información proporcionada por los NIDS. Dado que Elastic Stack requiere de recursos RAM, este máquina posee 8 GB, a diferencia de las demás que tienen la mitad.
- **Malware Information Sharing Platform (MISP)**: Plataforma software de código abierto para almacenar y compartir información de ciberamenazas (Threat Intelligence). Se usará para obtener la información desde fuentes *Open Source* y se almacenará en la base de datos ya estructurada que proporciona el propio MISP, para posterioremente exportar la información relevante hacia los NIDS. Dispone de fácil acceso y gestión de los datos a través de una API[3].
- **Suricata** y **Zeek**: Detectores de intrusos de red de código abierto ampliamente utilizado por organizaciones tanto públicas como privadas. En ambos casos, la detección de tráfico malicioso se realiza a través de reglas personalizadas. Estas reglas poseen un formato específico en cada caso.
- **Elastic Stack**: Conjunto de herramientas de código abierto que permiten el envío (*Filebeat*), almacenamiento (*Elasticsearch*) y visualización (*Kibana*) de los registros del un sistema. En este caso, se utiliza para visualizar los eventos y alertas que generan los NIDS.
- **Python**: Lenguaje de programación utilizado para la administración y automatización de la plataforma MISP mediante la creación de scripts y peticiones a la API.
    - *Requests*: Librería de Python utilizada para la gestión de petciones a la API de MISP.
- **Bash**: Intérprete de comandos en sistemas Unix y Linux que permite a los usuarios interactuar con el sistema operativo mediante comandos de texto. Se utiliza para la automatización de tareas mediante la creación de scripts.

Todos estos recursos conforman el escenario de trabajo, descrito en el siguiente capítulo.

[3]API: *Application Programming Interface*. Conjunto de reglas y protocolos que permite que diferentes aplicaciones y sistemas se comuniquen entre sí y compartan datos o funcionalidades de manera estructurada y segura.

1.4. Estructura y organización de la memoria

La memoria de este proyecto sigue en orden cronológico las tareas llevadas a cabo durante el mismo. En esta sección se describe de manera visual cada una de los apartados descritos en el trabajo:

- **Capítulo 1: Introducción**. Se describe a grandes rasgos el contexto y la motivación del proyecto, además de sus objetivos generales y los materiales y herramientas utilizadas. Por último, se presenta la estructura de la memoria para un fácil seguimiento de la misma.

- **Capítulo 2: Escenarios de trabajo y configuración**. En este capítulo se describen los dos escenarios utilizados para llevar a cabo el desarrollo. Se presenta un escenario con MVs un escenario domiciliario real.

- **Capítulo 3: Desarrollo y despliegue**. Se introducen todas la herramientas utilizadas en la infraestructura de la solución, así como las configuraciones llevadas a cabo para su implementación. Es el capítulo central de la memoria.

- **Capítulo 4: Pruebas y análisis de los resultados**. Tras la implementación de la solución completa, se realizan pruebas para verificar su funcionamiento y se discuten los resultados obtenidos. Los *tests* se reparten entre los distintos escenarios explicados en el Capítulo 2.

- **Capítulo 5: Conclusiones y líneas futuras**. Último capítulo de la memoria en el que se detallan las conclusiones finales tras la realización del trabajo. A continuación, se explican las futuras líneas de trabajo que se podrían seguir para seguir mejorando el sistema.

Capítulo 2

Escenarios de trabajo y configuración

Durante el desarrollo del proyecto, se han utilizado dos escenarios diferentes. En primer lugar, se emplea únicamente el entorno de red conformado por MVs. Una vez configurado el sistema y verificado su funcionamiento a través de pruebas, en la etapa final del proyecto se configura y se pone en funcionamiento el sistema en un entorno domiciliario real.

2.1. Escenario de red con MVs

Dadas las herramientas explicadas en el Capítulo 1, se establece el escenario de red conformado por MVs.

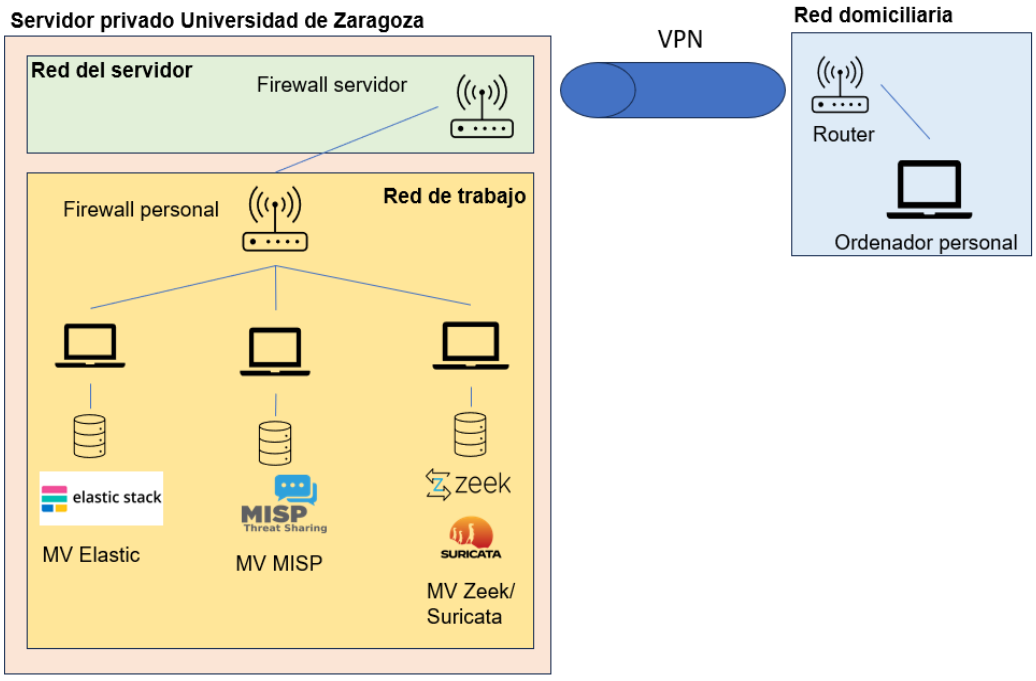


Figura 2.1: Escenario de red con MVs

Como se ilustra en la Figura 2.1, este entorno se compone de dos elementos claramente diferenciados: la red doméstica y el servidor privado de la Universidad de Zaragoza, los cuales se conectan mediante una VPN.

La red doméstica se compone únicamente de un ordenador personal y el *router* de acceso a Internet. Esta porción del entorno, aunque no relevante por sí misma, resulta esencial para el acceso al servidor a través de la VPN y a las MVs mediante conexiones SSH.

El servidor privado tiene en su primera capa un *Firewall Mikrotik* que lo protege de accesos indeseados. Este *firewall* cuenta con la terminación de túneles L2TP-IPsec para el establecimiento de túneles desde el exterior. Una vez creado el túnel entre el servidor y el ordenador personal del usuario, este último se encuentra virtualmente conectado a la interfaz interior del *firewall*. Desde esa posición en la red interna, se inician conexiones SSH a través del *firewall* personal. Dependiendo del puerto indicado en el protocolo TCP, el *firewall* hace una traducción de direcciones NAT para establecer la conexión con la máquina correspondiente.

En consecuencia, a nivel de redes en el servidor, se establecen dos redes distintas. Por un lado, existe una red principal en la que están conectados el *firewall* del servidor y virtualmente el ordenador personal. Por otro lado, se localiza una red conformada por la interfaz interior del *firewall* personal y las máquinas virtuales en las que se llevarán a cabo las operaciones.

2.2. Escenario de red domiciliario

Con el fin de abordar con precisión el análisis del sistema en un contexto real, se implementa una configuración que difiere del entorno de desarrollo original. En esta configuración, se continúa haciendo uso de un servidor privado, aunque se introduce una capa adicional de abstracción. Esta capa de abstracción hace 'invisible' al servidor privado en este escenario y únicamente es necesario analizar la red doméstica, tal y como se muestra en la Figura 2.2.

Una red domiciliaria convencional consta de un *router* de acceso a Internet y los dispositivos domésticos que se conectan a dicho *router*, como pueden ser *smartphones*, *smart* TVs, *tablets*, videoconsolas o electrodomésticos con acceso a Internet. Algunos de los dispositvos se conectan mediante puertos *Ethernet* al *router*, aunque la gran mayoría utiliza la interfaz WiFi para llevar a cabo la conexión.

Para poder aplicar nuestro sistema a un entorno doméstico real, se debe monitorizar todo el tráfico que pasa por él. De este modo, se estudia cuál es la manera adecuada para monitorizar todo el tráfico que llega a la interfaz interna del *router*. En este

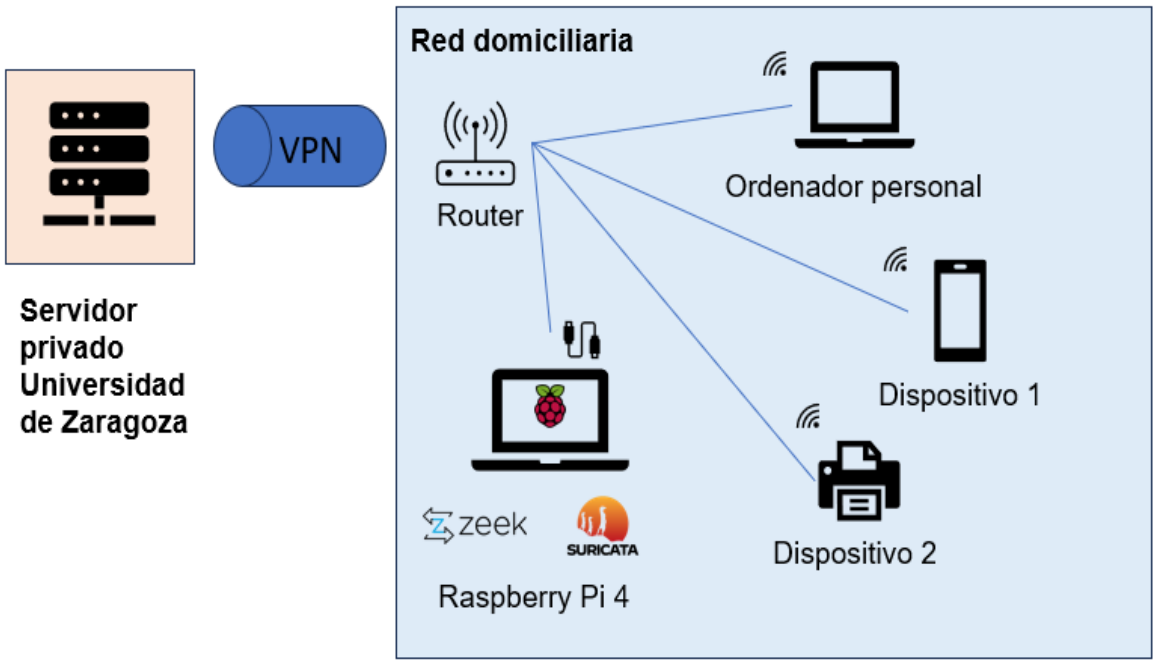


Figura 2.2: Escenario de red domiciliario

contexto, la primera idea que surge es activar el *port mirroring* en uno de los puertos del *router*. De esta forma, todo el tráfico procesado por el *router* sería reenviado al puerto habilitado con esta funcionalidad y solo se necesitaría un dispositivo conectado a dicho puerto que fuera capaz de visualizar ese flujo de información. Sin embargo, esta opción se descarta debido a que los *routers* instalados en las redes domésticas no cuentan con esta capacidad.

Así, aparece la idea de emplear una *Raspberry Pi 4*. Con este dispositivo y a través de un script denominado *spoof.py* ejecutado en conjunto con los detectores de intrusos de red, será posible supervisar y detectar tráfico malicioso en la red doméstica. Desde la *Raspberry Pi 4*, se lanza el script que actuará en la red doméstica realizando un ataque de *Man in the Middle* (MiTM). De acuerdo con esto, todos los dispositivos de la red mandarán su tráfico a la *Raspberry Pi 4* ('creyendo' que este dispositivo es el enrutador) y esta a su vez lo reenviará al *router*. De la misma manera, el *router* reenviará el tráfico procedente del exterior hacia la *Raspberry Pi 4*, 'creyendo' que es el dispositivo destinatario, y esta lo enviará a su destino real dentro de la red.

Una vez puesta en marcha esta funcionalidad, únicamente será necesario activar los NIDS: *Suricata* y *Zeek*. Estos detectores monitorizarán todo el tráfico de la red doméstica en tiempo real y generarán alertas a través de *logs* si algún parámetro de las conexiones de los dispositivos de la red coincide con alguna de las reglas personalizadas que tendrá cada NIDS.

Un concepto a tener en cuenta en el análisis de los resultados es la tolerancia a

pérdidas que tiene la arquitectura desarrollada. La *Raspberry Pi 4* tiene una capacidad de cómputo suficiente en su procesador (4 núcleos a 1.8GHz) para ejecutar los dos NIDS simultáneamente. Además, *Zeek* se configura en modo *cluster*, una arquitectura compleja que permite una monitorización exhaustiva del tráfico, aunque esto suponga un aumento en el consumo de recursos. En esta arquitectura existen nodos: *manager*, *proxy* y *workers*. El número de nodos se puede configurar y en ejecución realizan balanceo de carga. Teniendo en cuenta todo lo anterior, podemos aproximar las pérdidas diarias de tráfico que presenta la arquitectura a un 0.15 %. Estas pérdidas son significativamente pequeñas, por lo que el sistema es válido para llevar a cabo las pruebas.

Capítulo 3

Desarrollo y despliegue

En este capítulo se detallan cada una de las herramientas usadas durante el desarrollo del proyecto. Se sigue un orden cronológico en cuanto a la instalación, configuración e implementación de cada una de las partes.

3.1. MISP

3.1.1. Introducción

Malware Information Sharing Platform o **MISP** es una plataforma de código abierto diseñada para facilitar el intercambio de información sobre incidentes de ciberseguridad entre organizaciones y comunidades en todo el mundo. Su objetivo principal es mejorar la detección y mitigación de ciberamenazas, al permitir que los profesionales de seguridad compartan datos de manera eficiente y colaborativa.

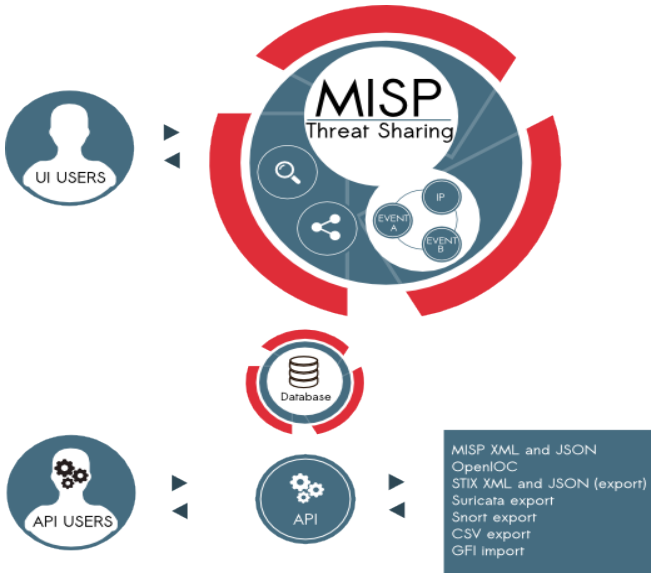


Figura 3.1: Arquitectura general de MISP

Las características clave de MISP incluyen la capacidad de recopilar y normalizar

datos de amenazas de diversas fuentes, como feeds de inteligencia, análisis de *malware* y eventos de seguridad. MISP proporciona una estructura para categorizar y etiquetar la información de manera estandarizada, lo que facilita la comprensión y el análisis de las amenazas. Además, MISP permite a cada usuario tener su propia plataforma con su base de datos de información, lo que se denomina **instancia** de MISP. Una vez instalado en el equipo, la instancia crea un servidor web en el propio equipo, desde el que se puede gestionar toda la información. Por otro lado, desde una instancia creada por un usuario se puede intercambiar información hacia otras instancias.

Los Indicadores de Compromiso (IOCs) son la unidad básica de información que describe un incidente de ciberseguridad. Estos IOCs desempeñan un papel crucial al permitirnos caracterizar de manera simple un ataque cibernético. Por ejemplo, conocer una dirección IP maliciosa que está relacionada con un tráfico que recibimos puede alertarnos sobre un posible ataque. De manera similar, disponer del valor *hash* sha256 de un archivo nos advierte sobre la peligrosidad de dicho archivo. Los IOCs establecen la base de la arquitectura de información de la plataforma MISP, como veremos en la siguiente sección.

3.1.2. Arquitectura de la información

Con el objetivo de organizar la información de manera intuitiva y estructurada, MISP plantea una estructura de información basada en **atributos**, **eventos** y ***feeds***. Cada una de estas estructuras está a su vez dividida en campos que nos muestran diferente información. Mediante esta ordenación sencilla se gestiona el intercambio de información dentro de la plataforma:

- **Atributos** (*attributes)*: Unidad básica de información en la plataforma, es decir, un **IOC**. A partir de ahora, se utilizarán de manera intercambiable los términos: dato, atributo e IOC. Entre los campos más importantes están:
  - *Date*: Fecha en la que se publicó el atributo en la instancia de la plataforma.
  - *Category*: Modo de agrupación que detalla el contexto del atributo. Por ejemplo: Actividad de red, detección de antivirus, fraude financiero, etc.
  - *Type*: Modo de agrupación específico que detalla el tipo de atributo. Por ejemplo: *ip-src/ip-dst, md5, sha256, domain* o *url*.
  - *Value*: Indica el valor del atributo en sí mismo. Para el tipo *ip-dst* el valor puede ser: *192.168.1.1*.
  - *Tag*: Etiqueta que clasifica atributos relacionados con un tipo de ataque en concreto o simplemente por contexto.

- *Flag IDS*: Valor booleano que nos indica si el atributo es exportable hacia IDS o no. Es una condición imprescindible para filtrar los IOCs válidos para nuestro análisis.

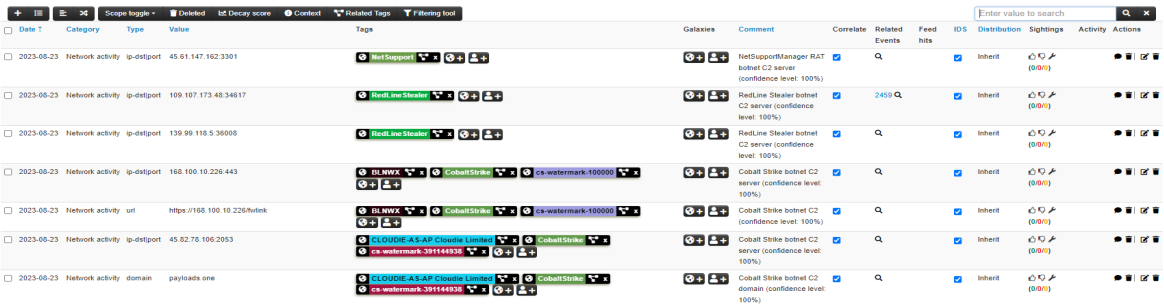


Figura 3.2: Visualización de atributos en la instancia MISP

– **Eventos** (*events*): Conjunto de atributos agrupados por una característica común. Esta característica puede ser la procedencia, la fecha o pertenencia al mismo incidente, entre otras. Los campos más relevantes son:

  - *ID*: Identificador único de cada evento.
  - *Published at*: Fecha en la que se publica el evento. Está relacionado en la búsqueda de eventos con el parámetro **publish_timestamp**.
  - *Last modified at*: Fecha en la que se modificó el evento por última vez. Está relacionado en la búsqueda de eventos con el parámetro **timestamp**.
  - *Info*: Breve descripción del evento.

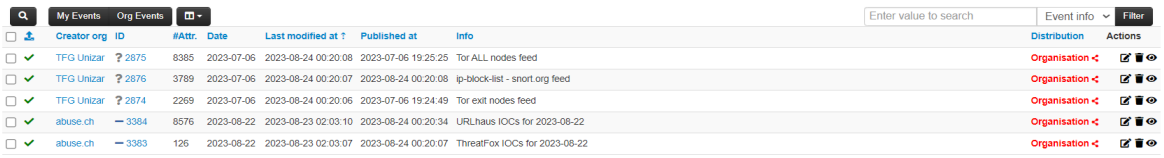


Figura 3.3: Visualización de eventos en la instancia MISP

– ***Feeds***: Fuentes *Open Source* de las que se descargan los eventos y se publican en la instancia de MISP. La fuente puede ser *local* (la información se obtiene desde el propio equipo) o *network* (la información se obtiene a través de Internet). Por otro lado, la información obtenida puede estar estructurada en 3 formatos distintos:

  1. *Misp*: Formato específico de MISP y más utilizado por la comunidad por ser el más completo. Permite caracterizar los atributos y los eventos con los campos descritos anteriormente.

2. *Csv*: Formato que separa la información mediante comas (,). Es más sencillo pero no permite personalizar algunos campos, MISP los autocompleta por defecto.

3. *Freetext*: Formato que separa la información por saltos de línea en un fichero. Al igual que el formato *csv*, no se pueden agregar campos específicos.

Los *feeds* se pueden administrar desde la instancia creándolos de forma personalizada o importando algunos ya creados por la comunidad. Para comenzar a recibir eventos desde estas fuentes, existen dos operaciones básicas: *enable* y *fetch*. La operación *enable* permite activar este feed para recibir sus eventos, mientras que la operación *fetch* descarga instantáneamente los nuevos eventos desde esa fuente activa hacia la instancia de MISP. El estudio de las fuentes **Open Source** se incluye en el Anexo A.

3.1.3. Ciclo de vida de un IOC

Cuando se manejan grandes volúmenes de datos, surge la cuestión fundamental de determinar cuánto tiempo es necesario retener esos datos en el sistema. El ciclo de vida de un IOC se define como el tiempo que transcurre desde que el indicador es detectado hasta que este resulta obsoleto. Este tiempo se puede aproximar mediante una función y existe una función única para cada Indicador de Compromiso. Determinar el valor exacto para cada dato resulta imposible. Además, nuestro sistema dispone de diferentes tipos de datos: una dirección IP maliciosa es mucho más volátil que el *hash* de un fichero, por lo tanto, el ciclo de vida de la dirección IP debería ser más corto.

La solución propuesta viene dada por el propio MISP, basándose en un artículo publicado por el CIRCL (*Computer Incident Response Center Luxembourg*) llamado *Decaying Indicators of Compromise* [9]. Este artículo estudia los Indicadores de Compromiso y realiza una aproximación certera de la función que describe el decaimiento del valor de los IOCs a lo largo de su vida útil. Esta función proporciona un valor denominado *score*, que va evolucionando con el transcurso del tiempo. Cuando este valor llega a un cierto umbral, se dice que el dato es inválido. MISP toma los IOCs que se exportan normalmente hacia los NIDS y crea su propio modelo partiendo de la función, ajustando los parámetros para esos IOCs. Para el cálculo de la función, se tienen en cuenta ciertos parámetros, que caracterizan cada dato:

- *Base score, base_score*: Toma valores de 0 a 100. Mide la confianza que se tiene sobre la fuente que proporciona el dato y los *tags* asociados al mismo. En la función de aproximación se toma el valor 80.

– *Lifetime*: Representa el tiempo de vida que tiene un dato, variando según el tipo de dato que sea. Se establece una media de un tiempo de vida de 120.

– *Seen time*: Tiempo que ha pasado entre que el dato fue visto o corroborado por primera y última vez. Esto dependerá de cada dato en concreto. Por aproximación en los cálculos se toma el tiempo en el que el IOC fue visto por última vez.

– *End time,* τ_a: Representa el tiempo en el cual el *score* llega a cero.

– *Decay rate,* δ_a: Caracteriza el grado de decaimiento de la función a lo largo del tiempo. MISP toma para su aproximación el valor 2.

– *Threshold*: Este parámetro no esta incluido en la función, pero MISP lo utiliza para determinar el valor límite que puede tomar la función para que el dato sea todavía válido. En este caso se toma como límite el valor 30.

La ecuación resultante al combinar todas las variables es la siguiente:

$$score_a = base\_score \cdot \left(1 - \left(\frac{t}{\tau_a}\right)^{\frac{1}{\delta_a}}\right)$$

En cuanto a los tipos de datos que usa MISP para su aproximación, se utilizan: *domain, ip, hostname y url*, en todas sus variantes (*ip-src, ip-dst, domain-port, etc.*). Estos son justamente los IOCs que nos servirán para exportar hacia los NIDS.

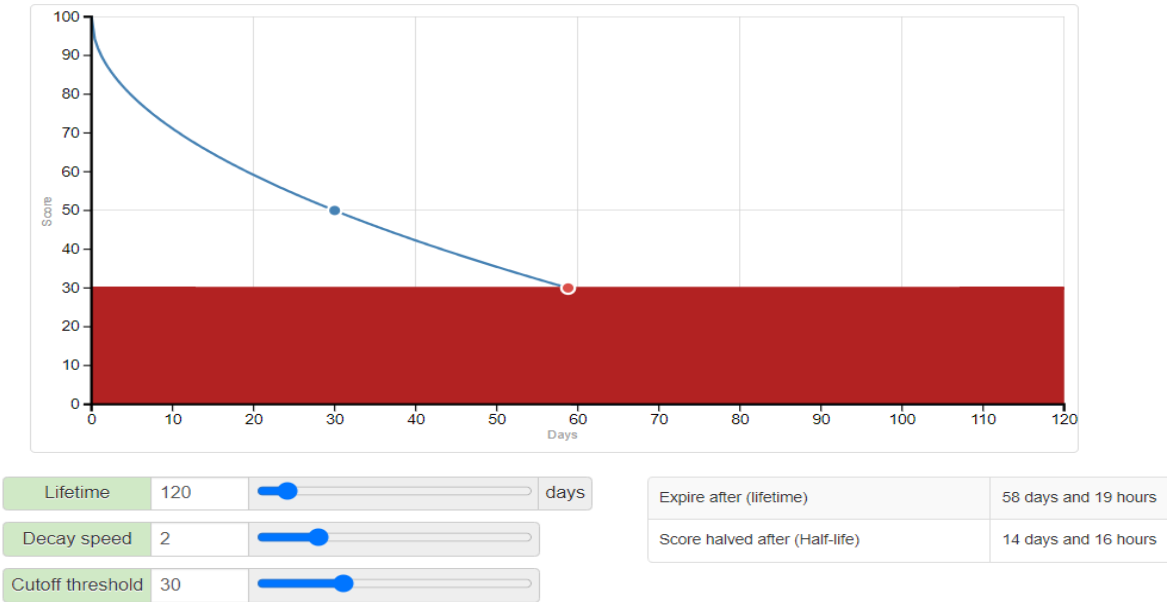


Figura 3.4: Herramienta de MISP para visualizar la función de decaimiento

En la Figura 3.4, se aprecia la función de decaimiento utilizada, con todos los parámetros definidos anteriormente. Se observa una función exponencial negativa que

comienza tomando el valor 120 (*lifetime*) en el tiempo 0. La función decae hasta llegar al valor 30, donde se considera que el dato ya no es válido. Se llega a la conclusión de que el tiempo que tarda un IOC en ser inválido desde que se ve por primera vez, es aproximadamente de 58 días y 19 horas. Esto también se puede calcular despejando el la fórmula el valor buscado. Este tiempo será el que utilicemos en la siguiente sección para establecer las políticas de rotación para nuestros datos.

3.1.4. Configuración y automatización

El objetivo principal por el que hemos utilizado la plataforma MISP es configurarla de forma automática para que diariamente exporte ficheros de reglas en la estructura adecuada hacia los detectores de intrusos de red. Con este fin, se usan las herramientas que MISP proporciona y se configura la plataforma de acuerdo con el objetivo.

La plataforma ofrece una gran cantidad de funcionalidades interesantes, pero en este proyecto se abordan únicamente las necesarias e imprescindibles para su desarrollo. Las siguientes secciones explican los procedimientos de configuración llevados a cabo para configurar y automatizar la plataforma.

Adquisición de datos

Como se explica en el Anexo A referente al estudio de fuentes *Open Source*, se utilizan los *feeds* para descargar los nuevos eventos procedentes de las fuentes [10] [11] [12]. Durante el desarrollo del proyecto, se han integrado 7 fuentes de datos distintas. Con ello, se pretende formar un conjunto de datos heterogéneo para comprobar la versatilidad de nuestro sistema, sin llegar a sobrecargarlo de información.

ID	Enabled	Caching	Name	Format	Provider	Org	Source	URL
1	✓	✗	CIRCL OSINT Feed	misp	CIRCL		network	https://www.circl.lu/doc/misp/feed-osint
2	✓	✗	The Botvrij.eu Data	misp	Botvrij.eu		network	https://www.botvrij.eu/data/feed-osint
4	✓	✗	Tor exit nodes	csv	TOR Node List from dan.me.uk - careful, this feed applies a lock-out after each pull. This is shared with the "Tor ALL nodes" feed.		network	https://www.dan.me.uk/torlist/?exit
5	✓	✗	Tor ALL nodes	csv	TOR Node List from dan.me.uk - careful, this feed applies a lock-out after each pull. This is shared with the "Tor exit nodes" feed.		network	https://www.dan.me.uk/torlist/
8	✓	✗	ip-block-list - snort.org	freetext	https://snort.org		network	https://snort.org/downloads/ip-block-list
65	✓	✗	Threatfox	misp	abuse.ch		network	https://threatfox.abuse.ch/downloads/misp/
67	✓	✗	URLhaus	misp	abuse.ch		network	https://urlhaus.abuse.ch/downloads/misp/

Figura 3.5: *Feeds* activos en la instancia de MISP

La configuración realizada para llevar esta integración comienza con la activación de las fuentes. Esta configuración se ha realizado a través del servidor web de la instancia, dada la facilidad que supone esta tarea. Se activa un *feed* de forma manual indicando la URL donde se encuentra la información para posteriormente ejecutar la activación.

Sobre los seis restantes sólo se ejecuta la operación *enable*, puesto que estas fuentes estaban incluidas por defecto en MISP. En la figura 3.5 se observa todas las fuentes integradas en el sistema.

Por otro lado, se adquieren de forma manual datos procedentes de la cuenta *C2intelfeedsbot* de la plataforma *Twitter*. Cada uno de los *tweets* publicados por esta cuenta son datos que se pueden integrar en MISP.

Ya activadas las fuentes, el siguiente paso es ejecutar la operación *fetch* sobre cada fuente para descargar los nuevos eventos en la base de datos de la instancia. MISP proporciona varias formas de gestionar las tareas en la plataforma:

- **Línea de comandos de la MV**: MISP permite automatizar ciertas tareas mediante la ejecución de comandos Linux en la MV donde se aloja la platafoma. Podría ser una buena solución, pero las tareas que se pueden automatizar son limitadas [13].

- **PyMISP**: Librería de Python creada por la comunidad de MISP. Utiliza la API de MISP para crear funciones en este lenguaje y poder ejecutar tareas desde *scripts* personalizados. Sin embargo, la librería está mal implementada en algunos casos e induce a errores, por lo que se descarta su uso [14].

- **MISP API**: A través de peticiones a la API de MISP de forma directa, podemos gestionar cualquier tarea relacionada con la instancia [13]. En este trabajo, las peticiones se realizan mediante la librería *requests* de Python por la facilidad para el posterior procesado de los datos. Con esta librería se piden o envían datos hacia la API mediante mensaje del protocolo HTTP[1]. Los dos tipos de mensajes que se utilizan son el HTTP GET y HTTP POST. En el Anexo C se detalla el uso de este tipo de peticiones.

Así, para la gestión de la tareas dentro de la instancia se opta por la creación de *scripts* en Python para realizar peticiones a la API de MISP. Se requiere para su utilización tanto la URL de la instancia como una *key* de autenticación. Los *scripts* desarrollados se incluyen anexos al final de la memoria.

Para realizar la operación *fetch* de todas las fuentes, se crea un *script* llamado *fetch_feeds_2.py* (versión 2) que realiza una solicitud POST a la siguiente URL: *https://misp.local/feeds/fetchFromAllFeeds*, donde *misp.local* coincide con la dirección donde se localiza la instancia (dirección IP de una red privada). Así, la instancia recibe la petición y procede a ejecutar la tarea de descargar los nuevos eventos desde todas las fuentes que estén activas.

[1]HTTP: *Hypertext Transfer Protocol*. Protocolo de comunicación que permite las transferencias de información a través de archivos (XML, HTML, etc.) en la World Wide Web.

Por otro lado, para la integración de los *tweets* desde la cuenta, se obtiene la información contenida en ellos a través de la API de *Twitter* y el lenguaje Python. En la Figura 3.6 se detalla el formato y contenido de un *tweet* de ejemplo para su integración en MISP. La primera línea es una descripción del atributo. Las siguientes dos líneas son los valores de los atributos que vamos a integrar. El resto de líneas son más datos, pero sin relevancia en su procesado.

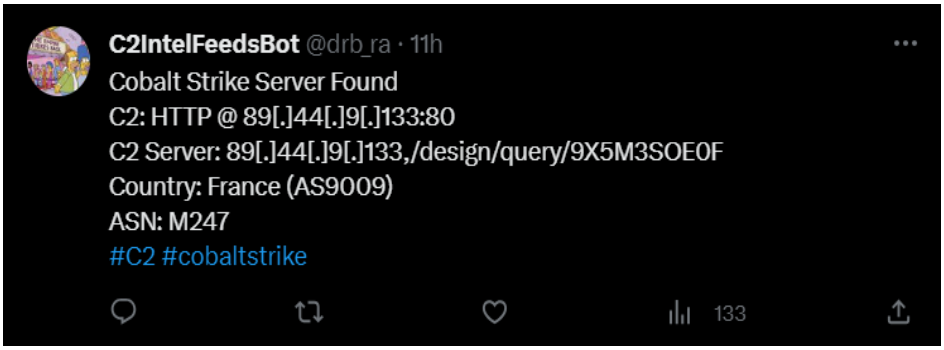


Figura 3.6: *Tweet* de ejemplo para su integración en MISP

A continuación, se procesan los datos del *tweet* para quedarnos con la información que vamos a integrar en MISP (fundamentalmente, el dato en sí mismo). Por último, se realizan dos peticiones POST a la API de MISP, de la misma forma que para la operación *fetch*. En la primera petición creamos un evento en la instancia mediante la URL: *https://misp.local/events/add*, con sus parámetros de creación (nombre, fecha, etc.). En la segunda petición POST, creamos un atributo en ese nuevo evento con la información extraída del *tweet*. Se hace a través de la URL: *https://misp.local/attributes/add/eventId*, donde en *eventId* se debe específicar el ID del evento donde se va a añadir el atributo. Además se deben incluir los parámetros propios del mismo, así como el valor, la fecha, etc. Todo esto se recoge en un *script* denominado *tweet_MISP.py*.

Procesado de datos

El procesado de los datos incluye todas las tareas automatizadas para la gestión de los datos una vez se encuentren en la plataforma. Dada la naturaleza del proyecto, existen ciertos criterios para determinar que datos son útiles y cuales no lo son. A continuación, se exponen las reglas de procesado, en el orden de filtrado real.

- ***Flag to IDS***: Todos los atributos que contengan este *flag* activado, son datos exportables hacia los IDS. Por lo tanto, el resto de IOCs que no tengan el *flag* son datos inservibles. Los datos que no disponen de este *flag* activado no se eliminan como tal de la instancia, sino que son excluidos durante la exportación.

- **Rotación de IOCs**: Tarea básica de procesado que consiste en eliminar todos aquellos datos que, según el criterio temporal que se elija, son inválidos. Se establecen las políticas descritas en la sección 3.1.3 y se consideran dos tipos de rotación:

    1. **Rotación estricta**: Consiste en consultar el valor de *score* calculado por MISP en su modelo de *Decaying IOCs*, en cada uno de los datos, y eliminar aquellos que no se consideren válidos (en el modelo de MISP se considera inválido si el valor de la función cae por debajo del *threshold*: 30). Esta rotación requiere consultar y calcular todos los valores de la función de cada indicador, lo que supone una gran complejidad de implementación además de un gran consumo de recursos. Por otro lado, el modelo es solo aplicable a unos tipos de datos concretos. Se opta por no utilizar este criterio.
    2. **Rotación suave**: Según el criterio obtenido en la sección 3.1.3, un atributo en media se vuelve inválido tras aproximadamente 60 días. Una opción cómoda y sencilla en implementación es consultar la fecha de última modificación de cada dato. Si esa fecha es mayor de 60 días, el atributo se borra de la base de datos. Se implementa esta rotación en el proyecto.

- **Tipo de atributo**: Se puede establecer un filtrado extra que aísle únicamente los IOCs que puedan ser detectados por los NIDS. Esto posee una ligera diferencia con el primer criterio de filtrado. Que un dato sea exportable hacia los IDS no quiere decir que un NIDS sea capaz de detectarlo. En esta fase no se utiliza este filtrado como tal, pero en secciones posteriores se ignorarán ciertos tipo de datos para su exportación por no ser de utilidad para el sistema.

La única política establecida en esta fase es la de rotación, el resto de procesado se hará en la fase de exportación. Como se ha explicado antes, se establece el criterio de **rotación suave**. Para implementarlo, recurrimos de nuevo a la API de MISP, a través de peticiones HTTP. La rotación suave se puede realizar de dos formas:

- A nivel de evento: Se buscan los eventos cuya fecha de última modificación sea mayor de 60 días. Se obtienen los *eventId* de cada uno de ellos para posteriormente eliminarlos. Es fácil de implementar y no consume muchos recursos, pero es posible que existan atributos dentro de un evento no eliminado que estén obsoletos. Al editar los eventos, se modifica su fecha de última modificación, pero no la de los atributos que están dentro. Así, se puede tener un evento modificado recientemente con atributos envejecidos.

– A nivel de atributo: Se hace lo mismo que a nivel de evento, pero con los atributos. Tanto su implementación como su consumo de recursos es costosa, pero la rotación a cambio es más limpia y eficiente.

En este caso, dadas las capacidades de cómputo de la MV MISP, se utiliza la rotación a nivel de evento. Se realiza una búsqueda de eventos con una solicitud POST a la URL: *https://misp.local/events/restSearch*, indicando en los parámetros de la petición el valor de *timestamp*. Este valor indica la fecha de última modificación de los eventos. Se especifica un rango de tiempo en el formato adecuado: ["120d", "60d"]. Con este parámetro, se indica que se requieren los eventos cuya última fecha de modificación se encuentre entre los 60 y 120 días.

Una vez hecho esto, la API proporciona como respuesta una lista de eventos en formato *json*[2] (así especificado en la cabecera de la solicitud). Mediante procesado de la lista con Python, obtenemos los *eventId* de cada evento y lo almacenamos en una variable. Para cada ID almacenado, realizamos una solicitud a la API para borrar cada evento, especificando el propio ID en la URL: *https://misp.local/events/delete/eventId*. Todo este proceso se recoge en un script de Python: *IOC_rotation.py*.

Filtrado y exportación de datos

Como última fase de automatización de la plataforma MISP, se van a exportar los datos adquiridos y procesados para poder alimentar los detectores de intrusos de red. Como se ha expuesto anteriormente, la exportación se hace a través de ficheros de reglas. La disposición de dos NIDS complica la exportación, puesto que cada uno de ellos posee un formato distinto a interpretar en los ficheros de reglas. En consecuencia, para cada uno de los detectores se deben exportar ficheros de reglas diferentes.

La estructura de la información en ficheros de reglas es una parte crucial a la hora de comenzar con la exportación. La primera idea que surge es utilizar un fichero para almacenar todas las reglas simultáneamente. Es una solución aceptable, sin embargo de esta forma no se tiene ninguna clasificación y la depuración de errores se vuelve más compleja. Por este motivo, se opta por dividir los ficheros de reglas por tipo de atributo, obteniendo así varios ficheros de reglas para cada NIDS.

A la hora de implementar la exportación, se utiliza un procedimiento diferente para cada uno de los NIDS, debido a las diferencias de formato. Sin embargo, en ambos casos la exportación se realiza a través de una solicitud HTTP POST hacia la URL: *https://misp.local/attributes/restSearch*. Con esta petición, se filtra la búsqueda de los atributos a exportar, introduciendo ciertos parámetros:

[2]JSON: *JavaScript Object Notation*. Formato ligero de intercambio de datos de fácil lectura y escritura para los usuarios. Además, es fácil de analizar y generar por parte de las máquinas.

– *returnFormat*: Formato de los datos que se solicitan. Aquí reside la principal diferencia de implementación entre *Suricata* y *Zeek*.

  - *Suricata*: Se especifica de forma nativa el formato de salida *'suricata'*. Las reglas se crean sin ningún procesado de datos posterior.
  - *Zeek*: Se usa el formato de salida *json*. A continuación se realiza procesado de esos datos para crear las reglas en el formato que *Zeek* interpreta. La exportación no es nativa desde MISP.

– *to_ids*: De todos los datos que existen en la plataforma, solo se exportan aquellos a partir de los cuales se puedan crear reglas. Es decir, los que tengan el *flag* activado.

– *timestamp*: Como la exportación a ficheros de reglas es diaria, se filtran los atributos cuya última fecha de modificación haya sido durante el día. En el formato de la API: '1d'.

– *type*: Se separa por tipo de IOC en los ficheros de reglas. Por lo tanto, especificamos el tipo de dato que vamos a filtrar.

Se realiza una petición a la API por cada tipo de dato y por cada *returnFormat*, dado que el resto de parámetros es constante.

Los tipos de datos a partir de los que vamos a crear ficheros de reglas son diferentes entre *Suricata* y *Zeek*, puesto que la exportación en un caso es nativa, pero en el otro requiere de procesado posterior. Se escogen los tipos de IOCs que cumplen la condición de ser **exportables** hacia los IDS y que además sean **detectables** por los mismos. Así, se tienen los distintos tipos de datos usados para la exportación:

– Suricata: *domain, domain-ip, hostname, hostname-port, ip-dst, ip-dst-port, ip-src, ip-src-port, url, md5, sha1, sha256, sha512.*

– Zeek: *domain, hostname, url, ip-src, ip-dst, md5, sha1, sha256, sha512, ja3-fingerprint-md5*

Tras realizar esta solicitud POST a la API y el posterior procesado de los datos en el caso de *Zeek*, se crea un fichero para cada tipo de atributo y se vuelcan las reglas en el formato adecuado. Se introduce una etiqueta temporal para diferenciar los días en los que se exportan las reglas, puesto que el día siguiente se exportan nuevas reglas hacia el mismo fichero. De esta forma, se controla tanto el tipo de dato como las fechas de exportación. Todo lo anterior se efectúa mediante el *script export_IOCs.py*.

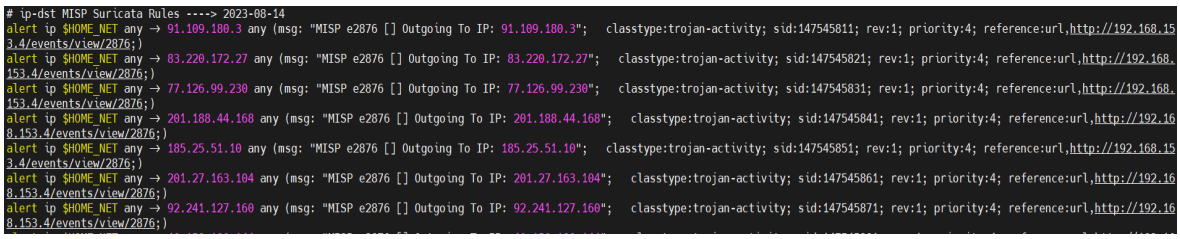


Figura 3.7: Fichero de reglas de *ip-dst* para Suricata

Los ficheros de reglas se almacenan en una carpeta en la máquina virtual donde está instalado MISP.

En las Figuras 3.7 y 3.8 se aprecia el formato y contenido que tiene un fichero de reglas para un tipo de dato en concreto tanto para *Suricata* como para *Zeek*. Se observa como existe una regla por cada una de las líneas del fichero.

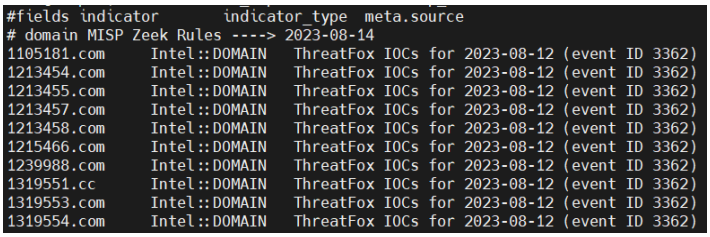


Figura 3.8: Fichero de reglas de *domain* para Zeek

Puede ocurrir que, por conflicto en alguna fecha, se exporte erróneamente dos veces la misma regla en un fichero. Esto podría derivar en un error en los IDS. Para solventar este tipo de problemas, se crea un *script* denominado *duplicate_lines.py*, que revisa los ficheros en busca de líneas duplicadas, después de haber exportado nuevas reglas. Si encuentra alguna, la elimina, dejando así el fichero sin líneas repetidas.

Automatización

Se han definido tres etapas para la configuración de la plataforma MISP y se han desarrollado las herramientas necesarias para mantener y automatizar la instancia. Por último, se deben establecer los procedimientos para lograr que todas estas herramientas se ejecuten cuando sea necesario. Las funcionalidades que se han implementado a lo largo de este apartado son en su totalidad *scripts* en el lenguaje de programación Python. Por lo tanto, necesitamos una herramienta extra que sea capaz de ejecutar el código necesario en el tiempo adecuado. Nuestra instancia de MISP está alojada en una MV con una distribución Debian 11 de Linux. Sabiendo esto, la opción más cómoda para la gestión de los tiempos de ejecución de *scripts* es la herramienta ***Crontab***.

Con *Crontab*, se pueden programar órdenes para ejecutar cualquier tarea como si se hiciera desde la consola de comandos a una hora y una fecha determinada. De esta

forma, se automatiza la ejecución de *scripts* que configuran y controlan la instancia de MISP. Se trata de añadir líneas en un fichero, indicando en un formato específico la fecha y la frecuencia de ejecución de los comandos que se deben lanzar. En el Anexo D se explica el funcionamiento de esta herramienta. En la Figura 3.9, se define el fichero de ***Crontab*** utilizado para la automatización de la instancia.

```
# Rotación IOCs
1 0 1 1,3,5,7,9,11 * sudo python3 /root/scripts/IOC_rotation.py

# Importacion de nuevos IOCs a la plataforma MISP
20 0 * * * sudo python3 /root/scripts/fetch_feeds_2.py

# Exportacion de los IOCs como archivos de reglas
30 0 * * * sudo python3 /root/scripts/export_IOCs.py

# Deteccion e eliminacion de lineas de reglas duplicadas
35 0 * * * sudo python3 /root/scripts/duplicate_lines.py

# Subida de archivos a repositorio privado de Github
40 0 * * * sudo bash /root/IDS_exportation/git.sh
```

Figura 3.9: Fichero de *Crontab* de la MV de MISP.

La primera línea corresponde a la automatización de la rotación de los IOCs. Para ello, se ejecuta el código *IOC_rotation.py* el día 1 a las 00:01 de los meses impares del año. De esta forma, se ejecuta la rotación cada 2 meses.

Los tres comandos siguientes corresponden a las fases de adquisición, procesado y exportación de los datos. Se ejecutan cada día a las 00:20, 00:30 y 00:35. Las horas de ejecución coinciden con la hora de actualización de los *feeds* introducidos en MISP.

El tiempo entre comandos se ha determinado realizando pruebas sobre el tiempo de ejecución de cada una de las tareas. En ningún caso, el tiempo de ejecución supera el minuto, la elección de los tiempos es orientativa.

Por último, se encuentra un comando que ejecuta un *script* de subida de los ficheros de reglas a un repositorio privado de *Github*. Este código se ejecuta diariamente a las 00:40, una vez terminado el tratamiento de los datos en la instancia. En la siguiente sección se entrará en más detalles sobre este procedimiento.

3.2. Almacenamiento de ficheros de reglas

Ya exportados los datos como ficheros de reglas, el siguiente paso es transmitir esos ficheros hacia los detectores de intrusos de red para que empiecen a funcionar. Un punto importante a tener en cuenta, es que los destinatarios de esta información son una MV

situada en la misma red que la MV de MISP, que contiene los datos, y otra máquina, la *Raspberry Pi 4*, localizada en una red privada remota. Existen una gran variedad de procedimientos para enviar estos archivos a través de la red. Las características que se buscan en el envío de estos ficheros son **integridad** y **confidencialidad**. A raíz de estas cualidades, surgen diferentes formas de realizar este transporte de datos:

- **SFTP**: *Secure File Transfer Protocol*. Protocolo de red que permite la transferencia segura de archivos entre un cliente y un servidor sobre SSH. A diferencia de FTP, SFTP cifra la información durante la transferencia. Tiene el inconveniente de que ocupa un gran ancho de banda en la red durante la transferencia. Además, posee una gran complejidad para alcanzar la red privada remota donde se encuentra la *Raspberry Pi 4*.

- **SCP**: *Secure Copy Protocol*. Similar a SFTP. Transferencia segura de archivos entre dos equipos remotos. SCP es más eficiente en la transmisión pero pierde en interactividad con los archivos a enviar. Dispone de los mismos problemas que SFTP.

- **Repositorio de *Github***: *Github* es una plataforma de alojamiento de proyectos con un control de versiones *Git*. Dispone de una funcionalidad de alojamiento de proyectos privados en Internet, con acceso seguro a través de *tokens*. La transferencia se puede realizar clonando el repositorio creado en el equipo destino. Se elige utilizar este sistema de almacenamiento y transferencia.

Se crea un repositorio privado de *Github*, al que solo se puede acceder desde una cuenta privada o a través de un token de autenticación expedido desde esa cuenta, el cual sólo el propietario conoce. De esta forma, se suben al repositorio en la nube diariamente los ficheros de reglas actualizados. Se usa un *script* desarrollado en *bash*, *git.sh*, que utiliza el token privado para poder ejecutar la operación *push* de *Git* y subir los archivos actualizados.

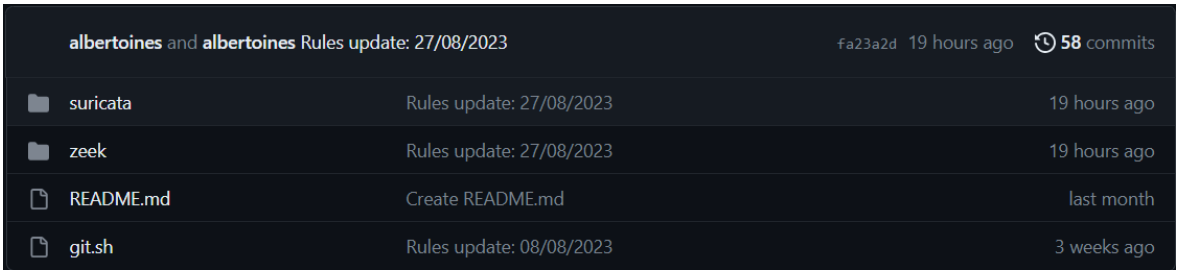


Figura 3.10: Repositorio privado de *Github* para el almacenamiento de ficheros de reglas.

Tras la subida de archivos al repositorio, se pueden descargar los archivos desde cualquier equipo que disponga del *token* de autenticación simplemente ejecutando la operación *clone* de *Git*.

3.3. Suricata como NIDS

3.3.1. Introducción

Suricata es un detector de intrusos de red (NIDS) de alto rendimiento, aunque también se utiliza como sistema de prevención de intrusos (IPS) o como un monitor de seguridad en la red. Es una herramienta de código abierto desarrollada por el OISF (*Open Information Security Foundation*).

En este proyecto se utiliza Suricata como NIDS, alimentado por reglas personalizadas y actualizadas diariamente con datos provenientes de la plataforma MISP. Las reglas (también denominadas *signatures* en el contexto de este IDS) son el eje entorno el que gira *Suricata*.

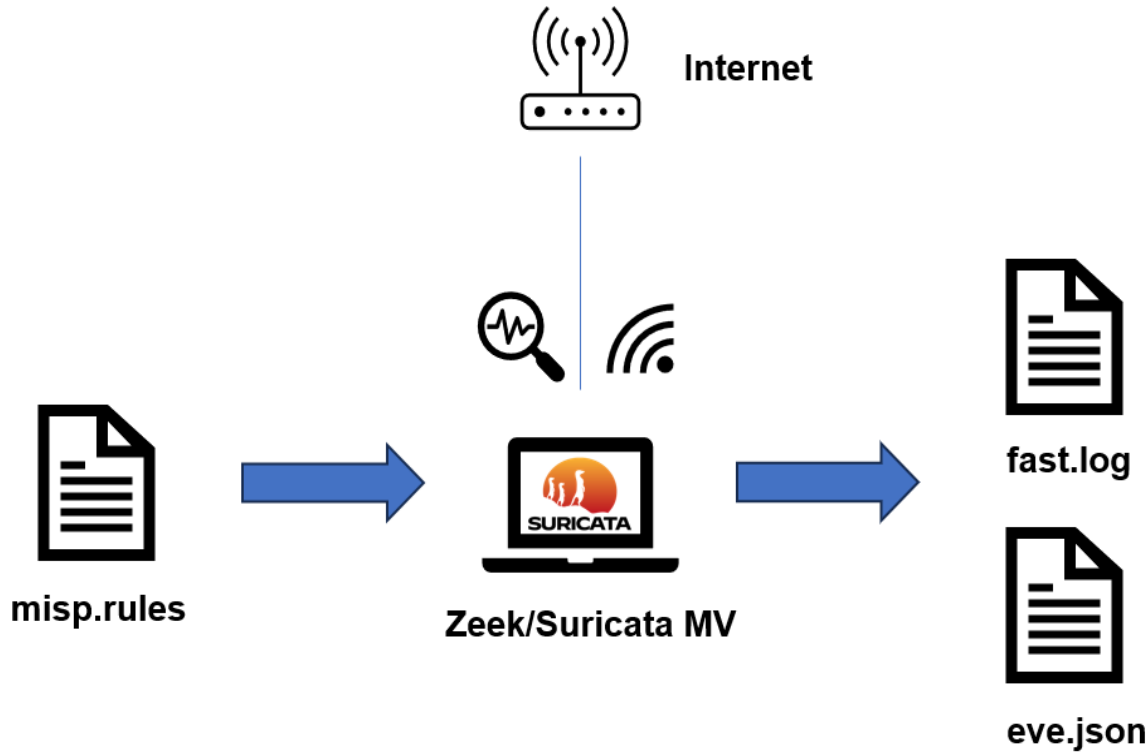


Figura 3.11: Funcionamiento básico de *Suricata*.

La herramienta interpreta los ficheros en los que se contienen *signatures* en el formato correcto y analiza el tráfico por las interfaces de red que se le indiquen en su configuración, tal y como se describe en la Figura 3.11. Si detecta la presencia de una regla, *Suricata* genera una alerta creando ficheros de logs donde se guarda la

información sobre la detección. Estos ficheros son: *fast.log* y *eve.json*. El primero recoge las alertas en texto plano para su consulta rápida. El segundo guarda en formato *json* las alertas generadas y se utiliza para exportar esa información de alertas hacia otros sistemas como *Elastic Stack*.

En el Anexo B, se incluye la definición del formato de las reglas de *Suricata*.

3.3.2. Configuración y automatización

Desde MISP se exportan ficheros de reglas clasificados por tipo de dato y dentro de cada fichero por fecha. El objetivo de este apartado es la puesta en marcha de *Suricata* para, con los ficheros de reglas exportados, analizar la interfaz de red de un equipo y detectar conexiones maliciosas.

Descarga de ficheros de reglas

La primera parte del proceso de configuración pasa por obtener los archivos de reglas procedentes de MISP. Como se ha detallado anteriormente, los ficheros se encuentran en un repositorio privado de *Github*, accesible únicamente a través de un *token* de autenticación. Desde un *script* en *bash*, *gitids.sh*, se introduce el *token* para acceder al repositorio privado y ejecutar la operación *git clone*, junto con la operación *git pull* para descargar los ficheros en el equipo donde se encuentra *Suricata*.

suricata.yaml

La configuración de la herramienta se hace a partir de este fichero escrito en el lenguaje YAML[3]. La primera configuración a implementar es la definición de la variable $HOME_NET. En esta variable se debe definir, según la documentación de *Suricata* [15], la dirección IP de la interfaz monitorizada, además de las redes locales que están en uso. En el caso del escenario con MVs, la red privada donde se monitoriza el tráfico coincide con la red: 192.168.153.0/24. Sin embargo, se utiliza un rango más grande, debido a la topología del escenario. Así, se usa la dirección: 192.168.0.0/16, como se ve en la Figura 3.12 . En el entorno doméstico, la red local por convenio suele ser la 192.168.1.0/24, pero puede haber variaciones como 192.168.0.0/24. Por otro lado, la variable $EXTERNAL_NET se considera como todas las redes que no son $HOME_NET.

El nombre de la interfaz donde se analiza el tráfico debe indicarse también en ese fichero, en el apartado *af-packet*. Por ejemplo, en el caso de la MV se utiliza la interfaz

[3]YAML: Formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl, así como en el formato de los correos electrónicos. Se utiliza para implementar archivos de configuración de cualquier programa.

eth0.

```
vars:
  # more specific is better for alert accuracy and performance
  address-groups:
    HOME_NET: "[192.168.0.0/16]"
    #HOME_NET: "[192.168.0.0/16]"
    #HOME_NET: "[10.0.0.0/8]"
    #HOME_NET: "[172.16.0.0/12]"
    #HOME_NET: "any"

    EXTERNAL_NET: "!$HOME_NET"
    #EXTERNAL_NET: "any"
```

Figura 3.12: Ejemplo de parte de fichero de *suricata.yaml*.

```
default-rule-path: /etc/suricata/rules

rule-files:
  - misp_url.rules
  - misp_sha512.rules
  - misp_sha256.rules
  - misp_sha1.rules
  - misp_md5.rules
  - misp_ip-src.rules
  - misp_ip-src-port.rules
  - misp_ip-dstport.rules
  - misp_ip-dst.rules
  - misp_hostname-port.rules
  - misp_hostname.rules
  - misp_domain.rules
  - misp_domain-ip.rules
```

Figura 3.13: Ejemplo de parte de fichero de *suricata.yaml*. Ruta de los ficheros de reglas utilizados.

Suricata comprende un sistema de actualización de reglas mediante fuentes proporcionadas por la propia herramienta, llamadas *ET Open ruleset*. Sin embargo, esta funcionalidad no se utiliza puesto que las reglas provienen de la instancia de MISP configurada con anterioridad.

Con ese fin, debemos añadir el directorio donde se van a encontrar las reglas, para que el programa sea capaz de leerlas y detectarlas. Para ello, en el apartado *default-rule-path* se deja la ruta por defecto: */etc/suricata/rules*. En las siguientes líneas, se especifican todos los nombres de los ficheros de reglas a utilizar, tal y como se muestra en la Figura 3.13. Una vez hecho esto, se deben incluir en la ruta por defecto los archivos descargados desde el repositorio de *Github*. Para ello, se realiza una simple copia de los archivos hacia esa ruta. El resto de configuraciones del fichero se dejan por defecto.

Llegados a este punto, solo debemos reiniciar *Suricata* para que la herramienta cargue los nuevos ficheros de reglas. Con ese objetivo, se ejecuta el comando:

```
systemctl restart suricata
```

Se obtiene en el archivo *suricata.log* los ficheros y reglas cargados, lo que se ilustra en la Figura 3.14.

```
27/8/2023 -- 01:15:02 - <Info> - fast output device (regular) initialized: fast.log
27/8/2023 -- 01:15:02 - <Info> - eve-log output device (regular) initialized: eve.json
27/8/2023 -- 01:15:02 - <Info> - stats output device (regular) initialized: stats.log
27/8/2023 -- 01:15:03 - <Info> - 13 rule files processed. 11225 rules successfully loaded, 0 rules failed
27/8/2023 -- 01:15:03 - <Info> - Threshold config parsed: 0 rule(s) found
```

Figura 3.14: Fichero de log *suricata.log* tras reiniciar el servicio.

Automatización

Como se ha hecho en la instancia de MISP, la automatización se realiza a través de *Crontab*. El fichero de *Crontab* de la máquina virtual comparte la configuración de *Suricata* y *Zeek*. El detalle se ilustra en la Figura 3.15.

```
# Clonado del repositorio de Github con ficheros de reglas
0 1 * * * sudo bash /root/gitids.sh

# Copiado de los ficheros de reglas a directorio de reglas de suricata y zeek
10 1 * * * sudo cp -r /root/IDS_exportation/suricata/* /etc/suricata/rules/
11 1 * * * sudo cp -r /root/IDS_exportation/zeek/* /opt/zeek/share/zeek/site/misp_zeek/

# Arranque de suricata con las nuevas reglas
15 1 * * * sudo systemctl restart suricata
16 1 * * * sudo zeekctl deploy
```

Figura 3.15: Fichero de *Crontab* de la MV de *Suricata/Zeek*.

La primera línea detalla la ejecución del *script* para descargar las reglas desde el repositorio, que continua en el tiempo inmediatamente después de terminar la exportación desde MISP. Esta tarea se ejecuta diariamente. Lo siguiente es el copiado, también diario, de los ficheros de reglas hacia la ruta donde los NIDS interpretan las reglas. Se hace para ambos IDS. Por último, se reinicia diariamente el servicio de ambos detectores para que carguen las nuevas reglas en sus sistemas.

3.4. Zeek como NIDS

3.4.1. Introducción

Al igual que *Suricata*, *Zeek* es algo más que un detector de intrusos de red. Esta herramienta dispone de una inmensa variedad de funcionalidades relacionadas con la

monitorización y seguridad en red, así como *frameworks* específicos para ejecutar tareas determinadas. Entre ellas se incluyen medidas de rendimiento y solución de problemas.

La característica definitoria de *Zeek* es la diversidad de *logs* que podemos consultar. Cada uno de estos *logs* se encarga de informar acerca de una cualidad del sistema analizado. Por ejemplo: *conn.log*, que informa de todas las conexiones de red que se realizan en la interfaz; *http.log*, usado para describir únicamente las conexiones HTTP o *intel.log*, fichero donde se almacenan todas las alertas generadas por el *framework* de inteligencia de *Zeek*, el cual usaremos para cargar reglas.

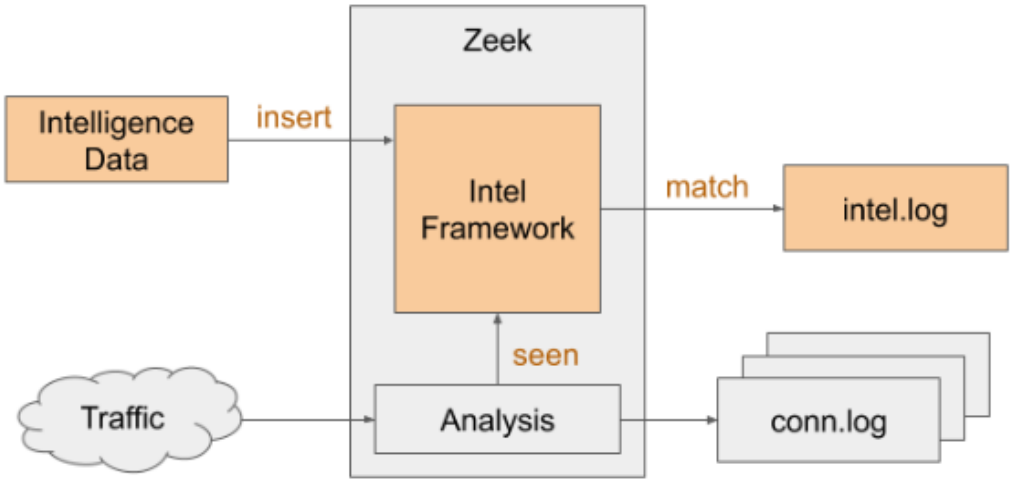


Figura 3.16: Funcionamiento básico de *Zeek*.

Por otro lado, *Zeek* dispone de dos modos de funcionamiento:

- **Modo *standalone***: Modo básico de funcionamiento. Utiliza un único proceso para monitorizar el tráfico de red. Consume pocos recursos del sistema, pero dependiendo de la configuración de *Zeek* existen casos donde este modo no funciona correctamente.

- **Modo *cluster***: Despliega un conjunto de nodos para la monitorización de la red. La arquitectura estándar está formada por: *manager, proxy, worker 1* y *worker 2*. *Zeek* permite cambiar esta configuración a través de un archivo de configuración. Consume más recursos del sistema a cambio de un mayor rendimiento. Usaremos esta configuración en ambos entornos de red.

Para integrar *Zeek* en nuestro sistema, haremos uso de *Intelligence Framework*, que permite la detección de tráfico malicioso a partir de reglas específicas. El funcionamiento es similar a Suricata: introducimos los ficheros de reglas en el lenguaje de *Zeek* (*Intelligence Data*), se analiza la red en busca de *matches* de las reglas y se generan alertas en *intel.log*, como se puede ver en la Figura 3.16.

En el Anexo B, se incluye la definición del formato de las reglas de *Zeek*.

3.4.2. Configuración y automatización

En este punto se deben configurar dos elementos por separado: *Zeek* y el *Intelligence Framework*. Se configuran en ese orden [16].

Configuración de Zeek

Lo primero que se debe hacer, es configurar el modo *cluster* para la arquitectura de detección. Para ello, se edita el fichero de configuración *node.cfg*. En la Figura 3.17 se detalla la configuración, estableciendo cada uno de los nodos. Los *workers* monitorizan la red en la interfaz *eth0*.

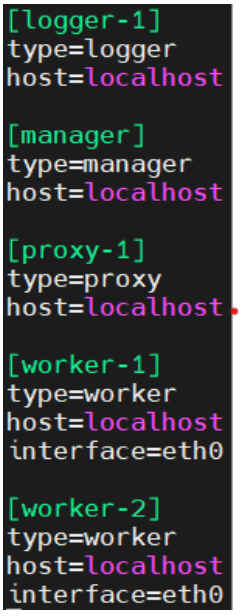


Figura 3.17: Ejemplo de *node.cfg* en modo *cluster*.

El siguiente paso, al igual que en *Suricata*, es configurar las redes donde se produce la monitorización. Esto se configura en el archivo *networks.cfg* y se detalla la red 192.168.0.0/16 en este caso.

De manera opcional, se puede establecer el periodo de rotación de los *logs*, pero no es relevante para este proyecto.

Una vez configurado, ya se puede lanzar *Zeek* usando el comando *zeekctl deploy*, comando que tendremos que usar a la hora de introducir nuevas reglas para que estas se carguen en el sistema.

Intelligence Framework

La configuración de este apartado requiere la creación de una carpeta extra en los archivos de *Zeek*, donde almacenaremos los ficheros de reglas y los *scripts* necesarios para inicializar este servicio.

El primer archivo que se escribe en esta carpeta es ***__load__.zeek***, donde indicamos que debe ejecutar un *script* llamado *main*, incluido en el mismo directorio. En ese archivo, incluiremos toda la información necesaria para activar y poner en marcha el *Intelligence Framework*. Para ello, debemos indicar a *Zeek* en este fichero que cargue algunos *scripts* relacionados con este *framework*, además de indicar la ruta hacia los ficheros de reglas. Esto se observa en la Figura 3.18. Los archivos de reglas se colocan en la ruta especificada.

```
##! Load Intel Framework
@load policy/integration/collective-intel
@load policy/frameworks/intel/seen
@load policy/frameworks/intel/do_notice
redef Intel::read_files += {
        "/opt/zeek/share/zeek/site/misp_zeek/misp_hostname.intel",
"/opt/zeek/share/zeek/site/misp_zeek/misp_domain.intel",
"/opt/zeek/share/zeek/site/misp_zeek/misp_url.intel",
"/opt/zeek/share/zeek/site/misp_zeek/misp_ip-src.intel",
"/opt/zeek/share/zeek/site/misp_zeek/misp_ip-dst.intel",
"/opt/zeek/share/zeek/site/misp_zeek/misp_md5.intel",
"/opt/zeek/share/zeek/site/misp_zeek/misp_sha1.intel",
"/opt/zeek/share/zeek/site/misp_zeek/misp_sha256.intel",
"/opt/zeek/share/zeek/site/misp_zeek/misp_sha512.intel",
"/opt/zeek/share/zeek/site/misp_zeek/misp_ja3-fingerprint-md5.intel"
};
```

Figura 3.18: Fichero *main* para la carga de *Intelligence Framework*.

El último punto a considerar es la modificación del archivo de configuración *local.zeek*, en el que se debe indicar que lea la carpeta donde hemos incluido los *scripts* y ficheros de reglas. También se debe detallar que el formato de los logs sea *json*, útil para su exportación hacia *Elastic Stack*. Para ello, se usan los comandos:

```
@load misp_zeek
@load policy/tuning/json-logs.zeek
```

Una vez hecho esto, reiniciamos el sistema con *zeekctl deploy* para cargar las nuevas configuraciones y el sistema estará listo para funcionar.

Automatización

La automatización se consigue de la misma forma que en *Suricata*, a través de *Crontab* en la misma máquina. En la Figura 3.9 se encuentran los comandos a utilizar para automatizar *Zeek*. Es el mismo proceso que con el otro IDS, solo cambiando la ruta, los archivos de reglas y el comando para reiniciar el programa.

3.5. Elastic Stack

Elastic Stack comprende un conjunto de herramientas de código abierto, diseñadas para adquirir datos desde cualquier plataforma, en cualquier formato para posteriormente buscar, analizar y visualizar estos datos en tiempo real. Aunque se trata de una solución gratuita, existen licencias de pago que introducen funcionalidades adicionales como X-Pack.

En este proyecto, se pretende usar la infraestructura de *Elastic* para centralizar los *logs* generados por los detectores de intrusos de red. En consecuencia, obtendremos un sistema que recoge todas las alertas en un entorno de red al completo.

Elastic Stack consta de una gran colección de herramientas que funcionan de forma independiente. En nuestro caso, se utilizan 3 de ellas para gestionar la centralización y visualización de los logs generados: *Filebeat, ElasticSearch* y *Kibana*.

- *Filebeat*: Herramienta instalada en la máquina donde se ejecutan los detectores de intrusos para recoger los *logs* generados por los mismos y enviarlos hacia *Elasticsearch*. Incluye módulos específicos para centralizar *logs* desde *Suricata* y *Zeek*, lo que facilita su configuración. También posee funciones de procesamiento, pero no se van a utilizar.

- *ElasticSearch*: Motor de búsqueda y analítica de datos integrado en *Elastic Stack*. Incluye soporte para cualquier tipo de dato, lo que lo hace muy versátil en cualquier entorno. Lo usaremos para recibir y procesar los *logs* en formato *json* procedentes de los NIDS en una máquina virtual diferente.

- *Kibana*: Plataforma de análisis y visualización de datos. Se conecta con *Elasticsearch* para obtener la información y representarla en diferentes formatos y figuras. Posee un portal web para la consulta de la información.

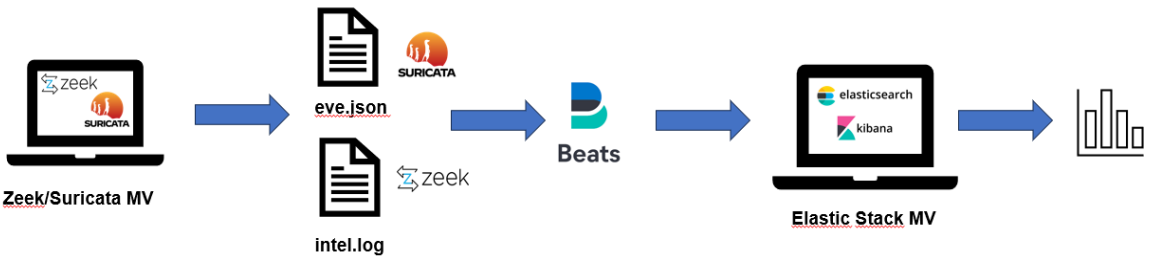


Figura 3.19: Infraestructura básica de *Elastic Stack* conectada con los NIDS.

En la Figura 3.19 se ilustra el funcionamiento básico de la infraestructura, conectada con los NIDS.

En conjunto las 3 herramientas conforman un sistema de centralización y visualización de *logs*, lo que podría denominarse un Centro de Operaciones de Red (NOC).

Por otra parte, al ser un sistema intercomunicado, es preciso tener en cuenta la seguridad de las transferencias de información. Por ello, en el apartado de configuración deberemos establecer las opciones necesarias para implementar la seguridad en las comunicaciones. Se utilizarán conexiones con el protocolo TLS.

La parte de instalación, configuración y automatización se incluye en el Anexo G al final de la memoria.

Capítulo 4

Pruebas y análisis de los resultados

Tras configurar y automatizar la infraestructura, se realizan una serie de pruebas con el objetivo de verificar el correcto funcionamiento del sistema al completo. La verificación mediante *tests* cobra sentido una vez los datos hayan sido exportados. Esto supone que la parte del sistema que involucra MISP no se ha puesto a prueba, sino que se ha ido depurando hasta que los datos han sido procesados y exportados correctamente. Las pruebas se realizan en los dos escenarios especificados en el Capítulo 2 e incluyen la detección de alertas por parte de ambos NIDS y su posterior visualización en *Kibana*.

4.1. Escenario de red con MVs

Se realizan las pruebas pertinentes en el escenario de red con máquinas virtuales. Este escenario es experimental y supone un primer acercamiento a un entorno de red real. Las pruebas se dividen en pruebas de detección en los NIDS y pruebas de visualización de las alertas generadas.

4.1.1. Detección de intrusos de red

Este escenario tiene la peculiaridad de que los NIDS están instalados en una única MV, por lo que el tráfico que monitorizan únicamente pertenece a esa máquina en cuestión. Esto facilita la ejecución de pruebas y su análisis posterior.

Para poner a prueba los NIDS, se consultan los ficheros de reglas cargados en los detectores. Se realizan acciones en dicha máquina con el objetivo de que los NIDS detecten alguna de las reglas definidas en los archivos. Por ejemplo, en el caso de direcciones IP, se realizan conexiones hacia esas IP, al igual que con dominios, *hostnames* o URLs. Para los *hash*, se realizan solicitudes HTTP POST, incluyendo en los datos de la petición alguno de los *hashes* maliciosos.

Tras realizar las pruebas con estos métodos, se incluyen resultados de las detecciones realizadas tanto por *Suricata* como por *Zeek*.

En el caso de *Suricata*, los logs que se generan a partir de las reglas se recogen en dos ficheros distintos, tal y como se explicó en el capítulo anterior. Estos ficheros son: *fast.log* y *eve.json*. Tras "forzar" manualmente alguna de las reglas, el resultado en estos ficheros se observa en las Figuras 4.1 y 4.2.

Figura 4.1: Archivo *fast.log* tras la detección de una IP maliciosa.

Figura 4.2: Archivo *eve.json* tras la detección de un dominio malicioso.

El archivo *eve.json* es el único que se envía hacia el sistema de *Elastic Stack*. La detección de reglas personalizadas con *Suricata* funciona correctamente en este entorno.

En el caso de *Zeek*, se utiliza un único fichero llamado *intel.log*. De la misma manera que en *Suricata*, realizamos pruebas de detección para comprobar si se generan alertas. En las Figuras 4.3 y 4.4, se ilustran dos ejemplos de detección de reglas, una de detección de dominio malicioso y otro de una URL maliciosa.

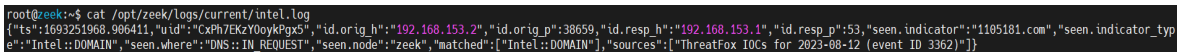


Figura 4.3: Archivo *intel.log* tras la detección de un dominio malicioso.

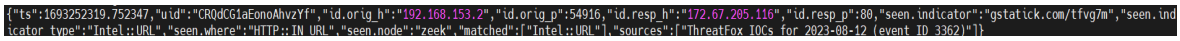


Figura 4.4: Archivo *intel.log* tras la detección de una URL maliciosa.

Este archivo se envía a la infraestructura de *Elastic Stack* para visualizar la información de forma más explicativa.

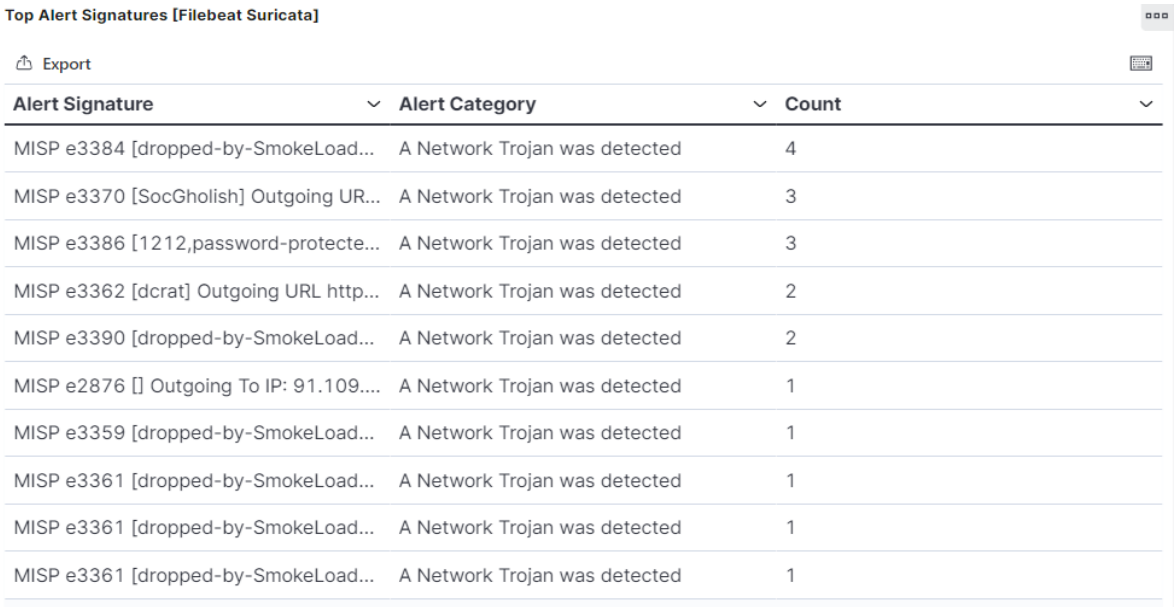
Se aprecia que los NIDS funcionan correctamente en la detección de reglas personalizadas procedentes de MISP, mostrando en los *logs* información relevante acerca de su procedencia, caracterización y descripción. En la siguiente sección, se muestran las visualizaciones que generan las alertas en *Kibana*.

4.1.2. Visualización de datos

Kibana posee unos *dashboards* predefinidos para visualizar la información sacada desde cada uno de los NIDS. Usaremos estas vistas para analizar la información recolectada por la plataforma.

Suricata

El *dashboard* de *Suricata* se divide en dos partes: eventos y alertas. Esto se debe a que el fichero *eve.json* no solo recoge las alertas de detección, sino también las conexiones que se realizan por esa interfaz, lo que se denominan eventos. Así, obtenemos una representación de todas las conexiones realizadas por la MV, además de un *display* dedicado exclusivamente a alertas. En las Figuras 4.5, 4.6 y 4.7; se observan diferentes representaciones de datos recogidos desde *Suricata*. Por un lado, se encuentran las alertas recogidas en una lista, mientras que los eventos se muestran en gráficos más complejos y tablas de datos.



Alert Signature	Alert Category	Count
MISP e3384 [dropped-by-SmokeLoad...	A Network Trojan was detected	4
MISP e3370 [SocGholish] Outgoing UR...	A Network Trojan was detected	3
MISP e3386 [1212,password-protecte...	A Network Trojan was detected	3
MISP e3362 [dcrat] Outgoing URL http...	A Network Trojan was detected	2
MISP e3390 [dropped-by-SmokeLoad...	A Network Trojan was detected	2
MISP e2876 [] Outgoing To IP: 91.109....	A Network Trojan was detected	1
MISP e3359 [dropped-by-SmokeLoad...	A Network Trojan was detected	1
MISP e3361 [dropped-by-SmokeLoad...	A Network Trojan was detected	1
MISP e3361 [dropped-by-SmokeLoad...	A Network Trojan was detected	1
MISP e3361 [dropped-by-SmokeLoad...	A Network Trojan was detected	1

Figura 4.5: Recopilatorio de alertas recogidas en *Kibana*.

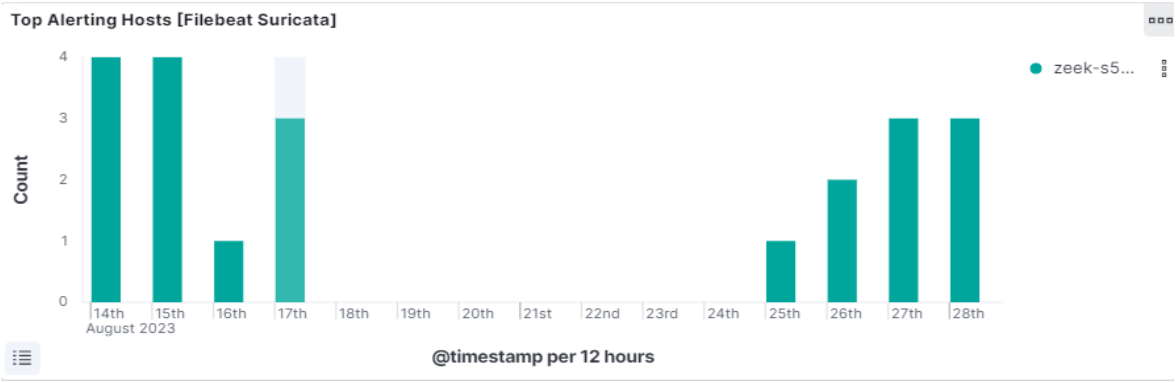


Figura 4.6: Recopilatorio de alertas clasificadas por día recogidas en *Kibana*.

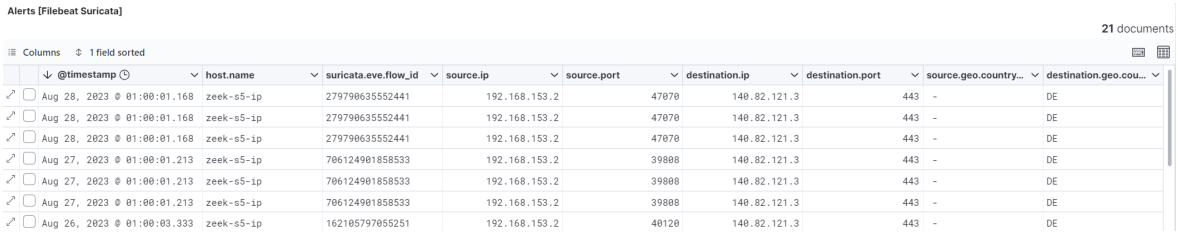


Figura 4.7: Recopilatorio de eventos de *Suricata* recogidos en *Kibana*.

Zeek

Zeek envía las alertas recogidas en el fichero *intel.log* hacia la plataforma de *Kibana*. Como en *Suricata*, existe un *dashboard* predefinido para mostrar la información que *Zeek* envía. En la Figura 4.8 se detalla una de las representaciones que se muestran tras la recepción de alertas. El resto de visualizaciones se mostrarán en el siguiente escenario.

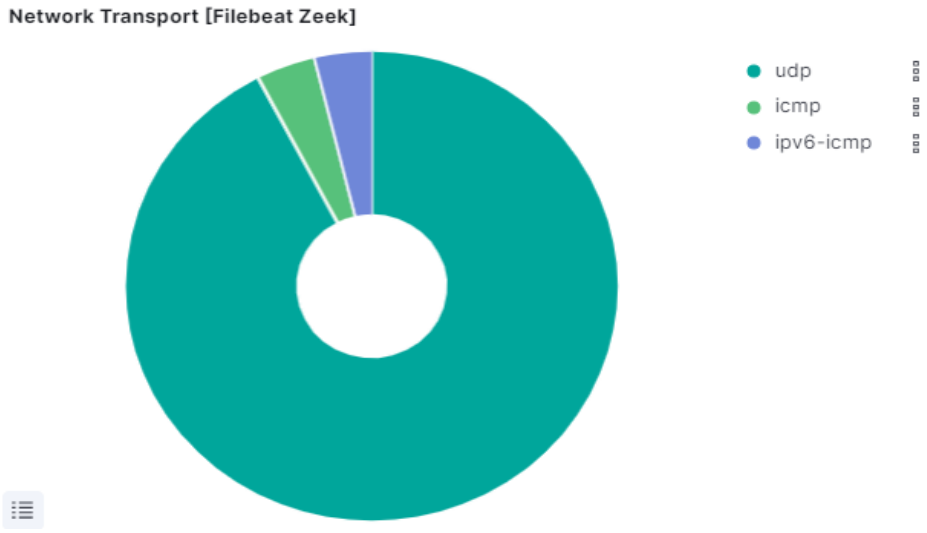


Figura 4.8: Gráfico de conexiones detectadas por *Zeek*.

4.2. Escenario de red domiciliario

A raíz de que la configuración ha sido la misma en ambos escenarios, los resultados obtenidos son similares. La única diferencia notable reside en el hecho de que, a través de la *Raspberry Pi 4*, se puede capturar el tráfico de toda la red y analizar el tráfico de cada dispositivo conectado. En el escenario virtual sólo se capturaba el tráfico de una única máquina. La capacidad de análisis es mayor, así como la cantidad de datos que se generan.

4.2.1. Detección de intrusos de red

El procedimiento utilizado es el mismo, así como los resultados obtenidos. Se analiza el fichero *intel.log* tras detectar una regla en la *Raspberry* procedente de una conexión realizada desde un ordenador conectado a la red doméstica, ilustrado en la Figura 4.9. La dirección origen 192.168.0.13 corresponde con la dirección del ordenador conectado a la red doméstica.

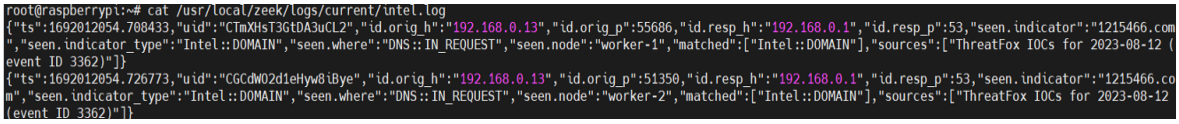


Figura 4.9: Detección de un dominio malicioso desde la *Raspberry Pi 4* en un ordenador conectado a la red doméstica.

Los *logs* de Suricata en la *Raspberry* son similares a los observados en el apartado anterior por lo que no se volverán a mostrar.

4.2.2. Visualización de datos

Del mismo modo que en el otro escenario, se envían los datos con *Filebeat* desde la *Raspberry* en la red doméstica hasta la MV de *Elastic Stack*. Para ello, la conexión debe pasar por el *firewall* privado del servidor de la Universidad de Zaragoza, realizando traducciones NAT para llegar a su destino. Tras realizar esto, se reciben las siguientes visualizaciones basadas en los *dashboards* de *Zeek*. De nuevo, los *logs* de Suricata, al haberse mostrado en el apartado anterior, no se mostrarán en este.

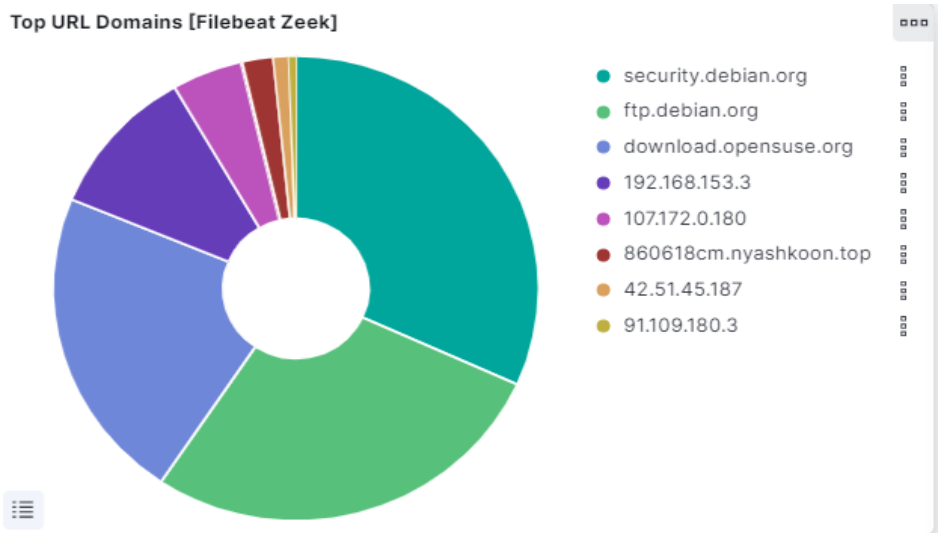


Figura 4.10: Gráfico de dominios más detectados por *Zeek*.

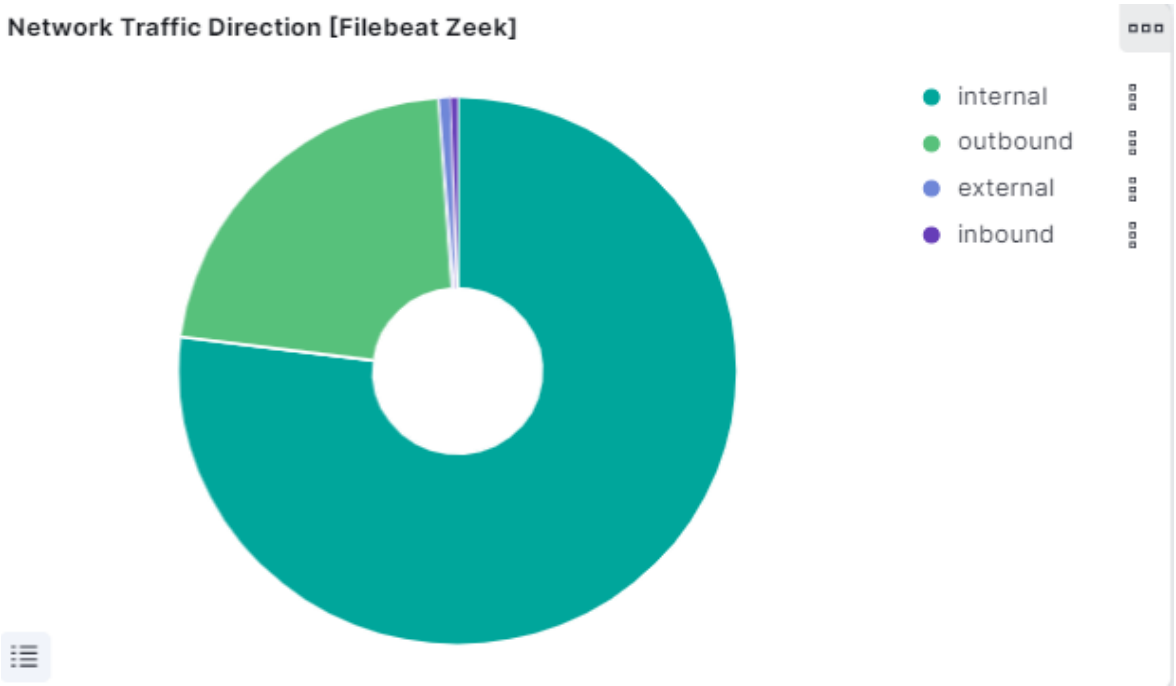


Figura 4.11: Gráfico de dirección de las conexiones detectadas por *Zeek*.

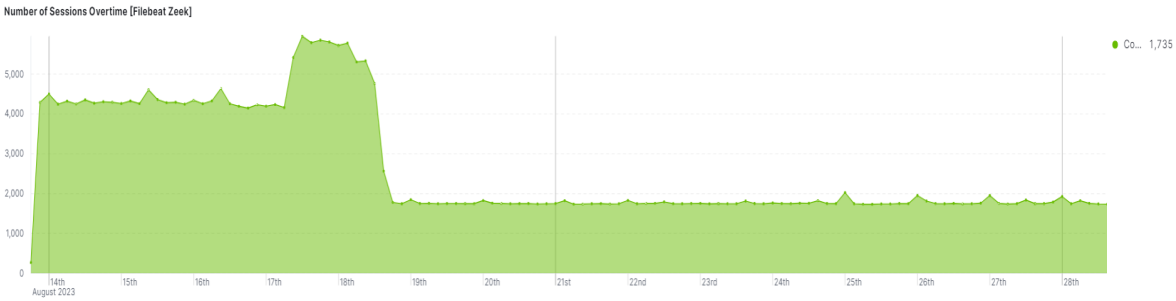


Figura 4.12: Gráfico de sesiones detectadas por *Zeek*.

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

A lo largo del proyecto, se han desarrollado un conjunto de herramientas, con el objetivo de entender su funcionamiento y sincronizar sus características. En consecuencia, se ha logrado conformar un sistema completo que se ejecuta de manera automática, cumpliendo los objetivos impuestos en el Capítulo 1.

La idea clave en este proyecto es la capacidad de orquestar cada uno de los componentes de la infraestructura, de tal forma que se ejecuten individualmente para aportar su rendimiento al sistema final. Por otro lado, las dificultades que se han enfrentado a lo largo del trabajo han sido provocadas por la heterogeneidad del conjunto global. En cada paso, se ha tenido que atender a los protocolos, formatos y configuraciones necesarias para que la herramienta estudiada se ejecute según nuestras necesidades, compatibilizando su funcionamiento con el resto de elementos de la infraestructura. Además, se ha llevado cada herramienta del trabajo hacia un equilibrio entre sus limitaciones y los requisitos necesarios.

La plataforma MISP se eligió dada su versatilidad a la hora de gestionar una base de datos ya estructurada y centralizada de Indicadores de Compromiso. Aunque MISP posee una gran variedad de funcionalidades interesantes, se han utilizado únicamente las necesarias para llevar a cabo las tareas que se requerían en el trabajo. La ventaja que supone MISP frente a la gestión "cruda" de datos obtenidos desde fuentes *Open Source* resulta abismal en términos de automatización, eficiencia, gestión, exportación y comunidad. En una primera aproximación, MISP suponía un lugar de almacenamiento y gestión de IOCs. Sin embargo, al final del trabajo se ha convertido en la plataforma central, desde la que se administra la información utilizada, manejándola a través de su API. Dicha API, entre las demás formas de manejo de datos, ha supuesto junto con el lenguaje Python la solución idónea dada su flexiblidad y sencillez. La exportación hacia los NIDS ha sido la fase más compleja de implementar en esta plataforma, dado

que se disponen de dos formatos distintos, uno nativo desde MISP. Además, se requiere estructurar la información exportada, para su fácil gestión en las siguientes fases. Esto supone la capacidad de adaptación que se ha tenido a lo largo del proyecto. Se ha encontrado una solución no nativa a través de Python para exportar los IOCs hacia los NIDS. También se ha logrado estructurar la información de las reglas de forma intuitiva y clara. Por último, la administración de los sistemas Linux es clave para llevar a cabo la administración en todas sus fases.

Suricata y *Zeek* suponen un reto añadido en el trabajo. Se han escogido ambos NIDS para afirmar que ambos pueden funcionar en un entorno como este, además de poder aprender el funcionamiento de cada uno, con sus configuraciones y puesta en marcha. Al fin y al cabo, estos detectores se han utilizado con el mismo propósito: detección de ciberamenazas a través de ficheros de reglas. Esto supone que, aun con sus diferencias, la configuración realizada ha sido similar. *Suricata* es un sistema sencillo, por lo que su configuración ha sido menos costosa. Sin embargo, la integración de *Zeek* es más compleja, dada su gran variedad de funcionalidades. Se debe configurar su modo de funcionamiento, además de estudiar el *framework* de inteligencia específico, entre los muchos otros que dispone. Además, al ser el formato de reglas más flexible, durante el desarrollo de la herramienta se detectaron algunos problemas de implementación, solventados con el tiempo. Los resultados obtenidos tras la exportación y detección de ciberamenazas son correctos. Se generan las alertas pertinentes cuando se provocan conexiones maliciosas, lo que significa el cumplimiento de uno de los objetivos del proyecto.

El producto *Elastic Stack* pasa a un segundo plano en este trabajo, siendo el resultado final obtenido tras realizar el resto de fases del proyecto. Este conjunto de programas supone una solución sencilla de implementar y visual, para la administración de los *logs* generados por los NIDS. De todas las herramientas implementadas, esta es la más sencilla y menos costosa de implementar. En cuanto a resultados de la plataforma, se han podido visualizar todas las alertas y eventos generados por ambos detectores de intrusos, lo que evidencia el correcto funcionamiento da cada una de las partes de la infraestructura.

En cuanto a la aplicación del sistema a un entorno doméstico real, los resultados evidencian la capacidad de adaptación que el proyecto posee, obteniendo resultados favorables en un entrono complejo y real. La configuración de la *Raspberry Pi 4*, permite el análisis completo de todo un entorno de red real, extrapolando las capacidades de los NIDS desde una máquina completa hasta un entorno de red real y complejo.

En conclusión, el desarrollo de este sistema heterogéneo global y automatizado supone un argumento firme de que cualquier sistema es configurable y automatizable,

haciendo uso de las herramientas precisas. Además, este trabajo significa una contribución más en el campo de la ciberseguridad, en un entorno digital en constante evolución y creciente complejidad. En términos personales, este trabajo ha significado un gran avance de mis conocimientos en este campo. Ha sido un experiencia muy enriquecedora además de interesante y satisfactoria, por la cantidad de horas dedicada, y por el resultado final obtenido.

5.2. Líneas futuras

Los desarrollos posteriores a este trabajo pueden ir enfocados hacia cada una de las partes que lo conforman. Así, existen varias líneas en las que se podría explorar para conseguir una ampliación de las capacidades del sistema.

En cuanto a la gestión y centralización de IOCs en MISP, existen varias líneas de trabajo que quedan abiertas. En la parte de adquisición de datos, dependiendo de las capacidades de almacenamiento de la plataforma, se podría importar más fuentes de datos, con el objetivo de enriquecer la base de datos del sistema. Esto se podría realizar de forma manual, a través de la publicación de eventos desde la API o importando más *feeds* incluidos por defecto en MISP. En el apartado de procesado de datos se nombraron varias formas de filtrar los datos, que no se acabaron implementando en el sistema final. Una de las líneas a seguir puede ser la limpieza profunda de la base de datos para obtener una plataforma pura de datos válidos para la exportación. Para ello, se podría limpiar la instancia quitando los atributos que no tengan el *flag to_ids* activado, entre otros cambios. La rotación se podría hacer de forma **estricta**, consiguiendo una base de datos más eficiente.

En el almacenamiento y envío de ficheros de reglas, se podrían explorar otras opciones de transferencia y almacenamiento de información, utilizando máquinas extra y otros protocolos de transporte, como los descritos en la sección 2 del Capítulo 3. Se busca mejorar la seguridad en el traspaso de información a través de protocolos más seguros.

Los NIDS no podrían cambiar su modo de funcionamiento, pero dependiendo de la máquina en la que esté instalado, sus capacidades de detección cambian. Si tratamos de aplicar el sistema a un entorno de red real analizando todo el tráfico de la red, el número de núcleos del procesador influye en el rendimiento de los detectores de intrusos. De esta forma, a mayor capacidad de procesamiento, mayores pueden ser las redes a analizar en cuanto a dispositivos o volumen de tráfico. Esta línea de trabajo tiene que ver con la escalabilidad del sistema.

La solución *Elastic Stack* tiene poco margen de mejora, puesto que solo se encarga

de visualizar los datos. La única forma de incrementar su rendimiento sería creando *dashboards* personalizados para la visión aún más eficiente de la información dispuesta.

Por último, en cuanto al sistema como conjunto global, se podría integrar junto con otros sistemas para contribuir en las siguientes fases de actuación frente a incidentes de seguridad. En esta línea, el sistema funcionaría junto con IPSs (*Intrusion Prevention Systems*) para neutralizar y prevenir las conexiones maliciosas que se realizan en los entornos de red reales.

Capítulo 6

Bibliografía

[1] Díez F. El instituto nacional de ciberseguridad gestionó 119.000 incidentes en 2022. *El País*, 2023.

[2] NIST. `https://www.cynet.com/incident-response`. Accedido por última vez en agosto de 2023.

[3] Suricata. `https://suricata.io/`. Accedido por última vez en agosto de 2023.

[4] Zeek. `https://zeek.org/`. Accedido por última vez en agosto de 2023.

[5] Indicador de Compromiso. `https://es.wikipedia.org/wiki/Indicador_de_compromiso/`. Accedido por última vez en agosto de 2023.

[6] MISP. `https://www.misp-project.org/`. Accedido por última vez en agosto de 2023.

[7] Elastic Stack. `https://www.elastic.co/es/elastic-stack`. Accedido por última vez en agosto de 2023.

[8] Security Operations Centers Working Group documentation. `https://wlcg-soc-wg-doc.web.cern.ch/index.html`. Accedido por última vez en agosto de 2023.

[9] Alexandre Dulaunoy Andras Iklody, Gerard Wagener and Sami Mokaddem. Decaying indicators of compromise. *CIRCL- Computer Incident Response Center Luxembourg*, 2018.

[10] awesome iocs. `https://github.com/sroberts/awesome-iocs`. Accedido por última vez en agosto de 2023.

[11] A List of the Best Open Source Threat Intelligence Feeds. `https://logz.io/blog/open-source-threat-intelligence-feeds/`. Accedido por última vez en agosto de 2023.

[12] The Ultimate List of Free and Open source Threat Intelligence Feedss. https://socradar.io/the-ultimate-list-of-free-and-open-source-threat-intelligence-feeds/. Accedido por última vez en agosto de 2023.

[13] MISP administration. https://www.circl.lu/doc/misp/administration/. Accedido por última vez en agosto de 2023.

[14] PyMISP. https://github.com/MISP/PyMISP. Accedido por última vez en agosto de 2023.

[15] Suricata User Guide. https://docs.suricata.io/en/suricata-6.0.10/. Accedido por última vez en agosto de 2023.

[16] Zeek Documentation. https://docs.zeek.org/en/master/. Accedido por última vez en agosto de 2023.

[17] abuse.ch. https://abuse.ch/#platforms. Accedido por última vez en agosto de 2023.

[18] AlienVaultOTX. https://otx.alienvault.com/browse/global/. Accedido por última vez en agosto de 2023.

[19] How to Use the Python Requests Module With REST APIs. https://www.nylas.com/blog/use-python-requests-module-rest-apis/. Accedido por última vez en agosto de 2023.

[20] How to Automate Tasks with cron Jobs in Linux. https://www.freecodecamp.org/news/cron-jobs-in-linux/. Accedido por última vez en agosto de 2023.

[21] Install MISP on Ubuntu 22.04/Ubuntu 20.04. https://kifarunix.com/install-misp-on-ubuntu/. Accedido por última vez en agosto de 2023.

[22] Spot suspicious activity on your local network with Suricata Intrusion Detection System (IDS) on Raspberry Pi. https://jufajardini.wordpress.com/2021/02/15/suricata-on-your-raspberry-pi/#architecture. Accedido por última vez en agosto de 2023.

[23] Installing the Elastic Stack. https://www.elastic.co/guide/en/elastic-stack/8.9/installing-elastic-stack.html. Accedido por última vez en agosto de 2023.

[24] IDS_exportation. https://github.com/albertoines/IDS_exportation. Accedido por última vez en agosto de 2023.

Lista de Figuras

Anexos

Anexos A

Estudio de fuentes Open Source

MISP proporciona por defecto una amplia gama de *feeds Open Source*. Estos *feeds* proceden de organizaciones, empresas e instituciones que ofrecen sus datos de inteligencia de ciberamenazas de forma gratuita. Los colectivos son de diversas procedencias y su información está diseñada para que adopte el formato correcto con el objetivo ser importada nativamente hacia la instancia de MISP, en cualquiera de las tres formas posibles (*misp, csv o freetext)*. Esto facilita mucho la adquisición de datos por parte de la plataforma [17] [18].

Por otro lado, MISP permite la creación personalizada de *feeds* por parte de los usuarios. Esta funcionalidad permite indicar fuentes de datos personalizadas. Sin embargo, se debe tener en cuenta que si el formato de los datos no es el correcto MISP es incapaz de adquirir esa información en su base de datos. De manera análoga, a través de la API de la plataforma podemos publicar nuestros propios eventos o incluso atributos, de nuevo estructurando los datos de forma adecuada.

En base a las formas que MISP nos ofrece para adquirir los datos hacia su plataforma, en este trabajo se realiza un estudio de todas las posibilidades de la ingesta de datos y su repercusión en la plataforma.

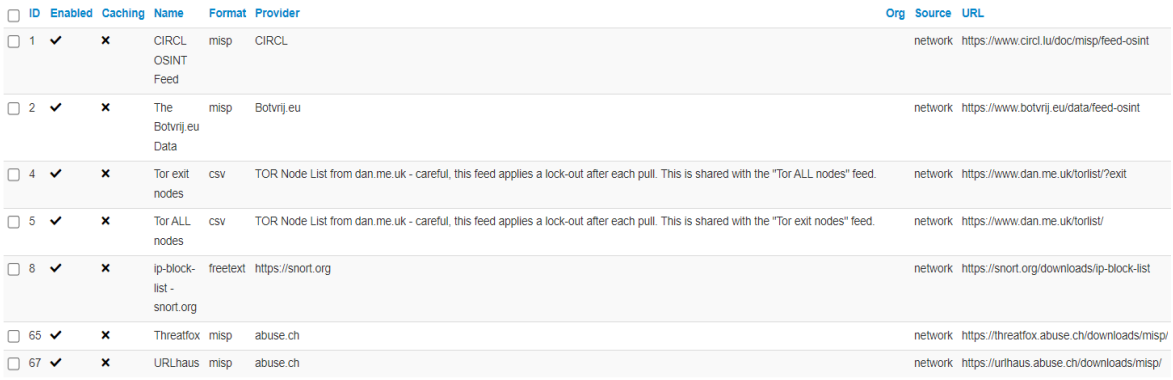
Inicialmente, en la fase de desarrollo se activaron 7 fuentes de datos proporcionadas por defecto por MISP, incluyendo los 3 formatos. Estos *feeds* ejecutan la operación *fetch* diariamente, descargando eventos nuevos hacia la instancia. Además, se incluyó una fuente configurada de forma manual y una publicación diaria de eventos en la plataforma mediante un *script* y la API de MISP.

Las fuentes por defecto incluidas en el desarrollo se dividen en dos grupos, según la naturaleza de los eventos que publican:

- ***Feeds* diarios o normales**: Esta clase de fuente se corresponde con el tipo de formato *misp*. Se caracterizan por seguir un criterio a la hora de publicar sus eventos. Por un lado, algunos siguen la regla de componer un evento con todos los atributos que han encontrado durante un día (periodicidad diaria) o simplemente

cuando descubren un incidente, agrupan todos los atributos relacionados con él y confeccionan un evento. Es el caso de los *feeds*: *CIRCL OSINT Feed, The Botvrij.eu Data, Threatfox y URLhaus.*

– ***Feeds* con sólo un evento**: Esta clase de fuente se corresponde con los formatos *csv* y *freetext*. Se trata de *feeds* que al activarlas crean un único evento que se sobreescribe cada vez que se ejecuta la operación de *fecth*. Esto se debe a la simplicidad del formato. Suelen ser listas de valores de atributos, como, por ejemplo, direcciones IP. Entre las usadas se encuentran: *ip-block-list - snort.org, Tor ALL nodes y Tor exit nodes.*



ID	Enabled	Caching	Name	Format	Provider	Org	Source	URL
1	✓	✗	CIRCL OSINT Feed	misp	CIRCL		network	https://www.circl.lu/doc/misp/feed-osint
2	✓	✗	The Botvrij.eu Data	misp	Botvrij.eu		network	https://www.botvrij.eu/data/feed-osint
4	✓	✗	Tor exit nodes	csv	TOR Node List from dan.me.uk - careful, this feed applies a lock-out after each pull. This is shared with the "Tor ALL nodes" feed.		network	https://www.dan.me.uk/torlist/?exit
5	✓	✗	Tor ALL nodes	csv	TOR Node List from dan.me.uk - careful, this feed applies a lock-out after each pull. This is shared with the "Tor exit nodes" feed.		network	https://www.dan.me.uk/torlist/
8	✓	✗	ip-block-list - snort.org	freetext	https://snort.org		network	https://snort.org/downloads/ip-block-list
65	✓	✗	Threatfox	misp	abuse.ch		network	https://threatfox.abuse.ch/downloads/misp/
67	✓	✗	URLhaus	misp	abuse.ch		network	https://urlhaus.abuse.ch/downloads/misp/

Figura A.1: Visualización de *feeds* usados en la instancia MISP

Las fuentes introducidas manualmente se comportan de igual manera que alguno de los dos grupos descritos anteriormente.

La publicación de eventos mediante un *script* y la API de MISP se trata en la sección 3.1.3: Configuración y automatización.

Ya caracterizado y comprobado el funcionamiento de los *feeds*, una pregunta interesante sería saber cual es el número idóneo de fuentes que deberíamos activar en nuestro sistema. La respuesta varía dependiendo de las condiciones de la propia infraestructura, su almacenamiento disponible y las políticas de rotación de los datos. Se debe tener en cuenta que conforme se activan más fuentes, los datos crecen de manera más rápida en la base de datos. Consecuentemente, esto supone que la plataforma esté más enriquecida con datos, a pesar de que los IDS tendrán que consumir más recursos según la información que les demos para detectar los IOCs. En conclusión, se trata de buscar un compromiso entre la cantidad de información y la carga que supone para nuestro sistema.

El último punto a tener en cuenta en este aspecto es el almacenamiento que supone la adquisición de todos estos datos por parte de la máquina virtual en la que está alojada el servidor de MISP. La idea inicial del proyecto incluye la actualización diaria

de la base de datos con eventos nuevos. Esto supone que si la base de datos crece de forma diaria llegará a un punto en el que la base de datos esté llena. La MV dispone de 140 GB de almacenamiento, lo que supone la limitación más importante en este punto.

Anexos B

Formato de reglas

B.1. Suricata

Una parte interesante del desarrollo del proyecto es estudiar el formato que se requiere para crear reglas personalizadas. Aunque la exportación de reglas es directa desde MISP, es necesario entender el formato de las reglas en *Suricata*, con el objetivo de solventar errores que puedan surgir [15].

Se analiza el formato con un ejemplo concreto de una regla:

```
alert ip $HOME_NET any -> 117.213.40.35 any (msg: "MISP e2876 []
Outgoing To IP: 117.213.40.35";  classtype:trojan-activity; sid:275161231;
rev:1; priority:4; reference:url,http://192.168.153.4/events/view/2876;)
```

Figura B.1: Ejemplo de regla en el formato de *Suricata*.

En la Figura B.1 se describe el formato que tiene una regla en el NIDS *Suricata*. La regla se puede fraccionar en 3 partes diferentes:

- *Action*: Define la acción a realizar cuando se detecta la regla (*alert, drop, pass, reject*).

- *Header*: Parte principal de la regla. Se divide en:

  - *Protocol*: Especifica el protocolo a nivel de red (ip), nivel de transporte (tcp, udp, icmp) o nivel de aplicación (http, ftp, tls, smb, dns).
  - *Source and Destination*: Detalla las direcciones IP origen y destino. Existen variedad de formas de indicar diercciones IP. Algunos ejemplos:
    - 192.168.1.1: Dirección IP 192.168.1.1.
    - ![192.168.1.1]: Todas las direcciones menos 192.168.1.1.

- $HOME_NET: Utiliza la variable definida en el fichero de configuración *suricata.yaml* para definir la dirección IP.
- any: Cualquier dirección IP.

- *Source and Destination ports*: Detalla los puertos origen y destino. Existen variedad de formas de indicar los puertos. Algunos ejemplos:
  - ![80, 83]: Todos los puertos menos 80 y 83.
  - ![80:83]: Todos los puertos menos del 80 hasta el 83.
  - any: Cualquier puerto.
- *Direction*: Simplemente indica el sentido de detección de la regla. Puede ser direccional o bidireccional.

– *Rule options*: Se especifican las configuraciones concretas para la implementación de la regla. Sigue el formato **name:setting**. Existe una gran variedad de opciones. Algunos ejemplos de *meta-keywords* son:

- *msg*: Mensaje que se muestra en los *logs* al saltar la regla.
- *classtype*: Clasifica el tipo de amenaza que supone la regla. Existen unos tipos definidos en la documentación de la herramienta.
- *sid*: Identificador único asignado a la regla.
- *rev*: Version de la regla.
- *priority*: Prioridad asignada a la regla. Puede ser desde 1 hasta 255.
- *reference:url*: Indica la fuente de información desde la que se ha obtenido la regla.

B.2. Zeek

Siguiendo en la línea de *Suricata*, *Zeek* posee un formato de reglas específico de para alimentar su *Intelligence Framework* [16]. El formato es más sencillo, y se detalla a continuación mediante un ejemplo:

```
#fields indicator     indicator_type  meta.source
1105181.com   Intel::DOMAIN   ThreatFox IOCs for 2023-08-12 (event ID 3362)
```

Figura B.2: Ejemplo de regla en el formato de *Zeek*.

En la Figura B.2 se describe el formato que tiene una regla en el NIDS *Zeek*. La regla se divide en 3 partes diferentes:

- *Indicator*: Valor del dato a detectar.
- *Indicator_type*: Describe el tipo de indicador que se utiliza, en este caso se trata de un dominio. Otros ejemplos de indicadores son: *Intel::URL, Intel::ADDR* o *Intel::FILE_HASH*.
- *Meta.source*: Breve descripción del dato detectado y la fuente del que proviene.

Todas las separaciones entre campos deben estar tabuladas una vez. Además, al inicio de cada fichero de datos se debe incluir la línea de cabecera.

Se permite ampliar el formato con más campos, pero, a pesar de su simpleza, este formato de reglas es efectivo.

Anexos C

Librería Python Requests para su uso en REST APIs

Las REST APIs son interfaces de comunicación entre sistemas de información (normalmente servicios web) que usan el protocolo HTTP para el transporte de los datos. Los datos intercambiados suelen estar en formato *json*, aunque también pueden estar en formato XML. Python es el lenguaje idóneo para interactuar con esta clase de APIs. Existe una librería diseñada específicamente para realizar intercambios de datos entre las REST APIs y Python. Esta librería se denomina *Requests* y es de gran utilidad en este proyecto para gestionar la instancia de MISP a través de su REST API [19].

C.1. Estructura de la petición

Cuando se quiere interactuar con la REST API, se necesita enviar una **petición**. La petición tiene diferentes campos:

- *Endpoint*: Es la URL que indica los datos a los que se está accediendo dentro de la API.
- *Method*: Se indica en este campo como se va a interactuar con los datos. Los métodos más usados son:
  - GET: Obtener datos de la API.
  - POST: Crear datos en la API.
  - PUT: Reeemplazar datos en la API.
  - DELETE: Borrar datos en la API.
- *Data*: Se involucra en los métodos que requieren cambiar datos en la API. Junto con la petición, se indican los datos que se quieren crear o con los que se quiere reemplazar otros datos (POST, PUT).

– *Headers*: Cabecera que incluye los metadatos en la petición. Se adjuntan en este campo *tokens* de autenticación, tipo de contenido que queremos recibir (*Content-type*) u otros datos relevantes.

C.2. Respuesta

Cuando se realiza una petición a la API, se obtiene una respuesta con el mismo formato. Por lo tanto, la respuesta tendrá una cabecera y contenido. El contenido tendrá los datos solicitados en el formato pedido. Normalmente, se utiliza el formato *json* para esta clase de peticiones.

C.3. Caso de uso: MISP

Lo primero que se debe hacer para poder usar esta librería es instalarla en el entorno de Python e importarla en el código. Una vez hecho esto, se accede a la API de MISP para realizar una búsqueda parametrizada de atributos. Para ello, se utiliza una función llamada *restSearch*, contenida en el *script functions.py*. Esta función, realizará una petición a la API y recibirá los datos solicitados para su consulta y procesado. En la

```python
def restSearch(misp_url, headers, type, timestamp):
    endpoint = misp_url + 'attributes/restSearch'
    data = {"returnFormat": "json", "to_ids": "1", "timestamp": timestamp ,"type": type }
    json_data = json.dumps(data)

    # Realiza la solicitud POST a la API de MISP
    response = requests.post(endpoint, headers=headers, data=json_data)

    # Verifica si la solicitud fue exitosa (código de respuesta 200)
    if response.status_code == 200:
        # Convierte la respuesta JSON en un diccionario de Python
        data = response.json()
        # Imprime los atributos
        data = response.content
        print(data.decode())
        #attr = data['response']['Attribute']
    else:
        # Imprime el mensaje de error si la solicitud no fue exitosa
        print('Error:', response.text)
```

Figura C.1: Función *restSearch* de Python para realizar una petición a la API de MISP.

Figura C.1, se observa la función descrita. Los parámetros de la función son: la URL de la instancia privada de MISP (*misp_url*), la cabecera de la petición(*headers*) y los dos parámetros de búsqueda (*type, timestamp*). En este caso, se buscan atributos presentes en la base de datos sean de un tipo de dato específico y creados en un rango de tiempo determinado.

Como se indica en la sección de estructura de la petición, creamos el *endpoint* con la URL de nuestra instancia y añadiendo la URL específica de la API para realizar búsquedas. Puesto que va a ser una solicitud POST, introducimos los datos para la búsqueda y formato de los datos. Indicamos que el formato es *json*, buscando atributos con el *flag to_ids*, de un tipo y en un rango de tiempo determinados. A continuación se realiza la petición POST. Si la respuesta es correcta, se reciben los datos solicitados y se decodifican para su uso. Si ocurre lo contrario, significa que ha habido un error.

Un ejemplo de datos recibidos se detalla a continuación:


```
"Event":{"org_id":"1","distribution":"0","id":"2875","info":"Tor ALL
nodes feed","orgc_id":"1","uuid":"66692329-f565-49d9-a79b-
e2c3512f3a99"}},
{"id":"27143022","event_id":"2875","object_id":"0","object_relation":
null,"category":"Network activity","type":"ip-
dst","to_ids":true,"uuid":"ca77b1f3-ebb7-4bb1-a0c2-
25ee29e2e927","timestamp":"1693347607","distribution":"5","sharing_gr
oup_id":"0","comment":"","deleted":false,"disable_correlation":false,
"first_seen":null,"last_seen":null,"value":"116.80.79.95","Event":
{"org_id":"1","distribution":"0","id":"2875","info":"Tor ALL nodes
feed","orgc_id":"1","uuid":"66692329-f565-49d9-a79b-e2c3512f3a99"}},
{"id":"27143023","event_id":"2875","object_id":"0","object_relation":
null,"category":"Network activity","type":"ip-
dst","to_ids":true,"uuid":"b6a87d7f-df4b-48cc-98ac-
2f2ca3ee9d7a","timestamp":"1693347607","distribution":"5","sharing_gr
oup_id":"0","comment":"","deleted":false,"disable_correlation":false,
"first_seen":null,"last_seen":null,"value":"117.252.193.166","Event":
{"org_id":"1","distribution":"0","id":"2875","info":"Tor ALL nodes
feed","orgc_id":"1","uuid":"66692329-f565-49d9-a79b-e2c3512f3a99"}},
{"id":"27143025","event_id":"2875","object_id":"0","object_relation":
null,"category":"Network activity","type":"ip-
dst","to_ids":true,"uuid":"fab52250-6e09-4c10-9f14-
2ab6378f7173","timestamp":"1693347607","distribution":"5","sharing_gr
oup_id":"0","comment":"","deleted":false,"disable_correlation":false,
"first_seen":null,"last_seen":null,"value":"129.80.239.117"
```

Anexos D

Uso de Crontab para la automatización de tareas

El uso de *Crontab* es imprescindible para las tareas de automatización en el proyecto. En este anexo, se detalla el funcionamiento de la herramienta, con sus posibilidades de configuración y ejemplos de uso [20].

Cron es una herramienta de gestión de tareas para sistemas *Unix* o similiar. El *daemon* crond activa *cron*, que se ejecuta en segundo plano. Este proceso lee un fichero llamado *Crontab*. En este archivo, utilizando la sintaxis adecuada, se pueden escribir tareas para que *cron* las ejecute en una fecha y hora determinada, con posiblidad de establecer periodicidad. De este modo, en cada uno de los procesos de automatización de las plataformas, se utiliza esta herramienta para establecer periodicidad en la ejecución de tareas.

Para acceder al fichero *Crontab*, se utiliza el siguiente comando:

```
crontab -e
```

Una vez dentro, podemos editar el fichero para automatizar nuestras tareas.

D.1. Sintaxis

Se detalla un ejemplo de tarea *cron*:

```
*   *   *   *   *  sh /path/to/script/script.sh
|   |   |   |   |              |
|   |   |   |   |      Comando or Script a ejecutar
|   |   |   |   |
|   |   |   |   |
|   |   |   |   |
|   |   |   | Día de la semana (0-6)
```

```
|   |   |   |
|   |   | Mes del año (1-12)
|   |   |
|   | Día del mes(1-31)
|   |
| Hora(0-23)
|
Minuto(0-59)
```

En el margen izquierdo se tienen 5 asteriscos. Estos se corresponden con los minutos, horas, días, meses y días de la semana de la tarea respectivamente. Los asteriscos indican para cada campo que la periodicidad para dicha variable es total. Por ejemplo, un asterisco en el campo "hora" indica que la tarea se ejecuta cada hora. Si ponemos un valor concreto, se indica que la tarea se ejecutará a esa hora precisa, de acuerdo con los demás campos del cron.

En el margen derecho se describe la tarea a ejecutar, en este caso es la ejecución de un *script* en bash en la ruta especificada.

D.2. Casos de uso: automatización

En la instancia de MISP, se requiere exportar los archivos de reglas actualizados diariamente. Para realizar la actualización de los ficheros de reglas se utiliza el *script export_IOCs.py*. Para que este código se ejecute diariamente, se utiliza una tarea *cron*:

```
# Exportacion de los IOCs como archivos de reglas
30 0 * * * sudo python3 /root/scripts/export_IOCs.py
```

Esta tarea cron indica que el código se ejecutará diariamente a las 00:30. Otro ejemplo sería la rotación de IOCs mediante el *script IOC_rotation.py*. La tarea *cron* es la siguiente:

```
# Rotación IOCs
1 0 1 1,3,5,7,9,11 * sudo python3 /root/scripts/IOC_rotation.py
```

Esta tarea, según la sintaxis, se ejecutará a las 00:01 del día 1 de los meses impares del año. Es decir, cada dos meses, el periodo de rotación de IOCs establecido.

Anexos E

Instalación de MISP

La instalación de la plataforma MISP en un sistema operativo como *Debian 11* supone un problema por temas de compatiblidad. Para instalar MISP se han seguido los pasos de instalación para una distribución parecida como es *Ubuntu* [21]. Siguiendo estos pasos, la instalación ha sido limpia y el programa funciona correctamente.

El primer paso es instalar las dependencias mediante los siguientes comandos:

```
sudo apt update
sudo apt install postfix mailutils curl gcc git
gpg-agent make libcaca-dev liblua5.3-dev \
python python3 openssl redis-server vim zip unzip
virtualenv libfuzzy-dev sqlite3 \
moreutils python3-dev python3-pip libxml2-dev
libxslt1-dev zlib1g-dev \
python-setuptools openssl cmake
```

A continuación, se crea un usuario para usar MISP y se instalan más dependencias necesarias como MariaDB o PHP:

```
sudo useradd -s /bin/bash -m -G adm,cdrom,sudo,dip,
plugdev,www-data,staff misp

sudo passwd misp
curl -LsS https://downloads.mariadb.com/MariaDB/mariadb_repo_setup
| sudo bash -s -- --mariadb-server-version=10.9

sudo apt install mariadb-client mariadb-server -y
apt install libapache2-mod-php php php-cli php-dev
php-json php-xml php-mysql php-opcache \
php-readline php-mbstring php-zip php-redis php-gnupg
```

```
php-intl php-bcmath php-gd php-curl
```

Se procede con la configuración de la base de datos de MISP. Se crea un usuario y se le otorgan permisos sobre la base de datos creada:

```
sudo systemctl start mariadb
sudo mysql_secure_installation
sudo mysql -u root -p -e "create database misp;"
sudo mysql -u root -p -e "grant all on misp.*
to mispadmin@localhost identified by 'MISP-DB-Password';"

sudo mysql -u root -p -e "flush privileges;"
sudo -Hu www-data cat /var/www/MISP/INSTALL/MYSQL.sql
| mysql -u mispadmin -p misp
```

Seguimos con la instalación de MISP. Se realiza mediante un clonado del repositorio oficial de *Github*. Además, se configura Python para funcionar en el sistema:

```
sudo mkdir /var/www/MISP
sudo git clone https://github.com/MISP/MISP.git
/var/www/MISP/

sudo git -C /var/www/MISP/ submodule update
--progress --init --recursive

sudo chown -R www-data: /var/www/MISP
sudo -u www-data git -C /var/www/MISP submodule foreach --recursive
git config core.filemode false

sudo -u www-data git -C /var/www/MISP config core.filemode false
sudo -u www-data virtualenv -p python3 /var/www/MISP/venv
sudo mkdir /var/www/.cache/
sudo chown -R www-data: /var/www/.cache/
sudo -u www-data /var/www/MISP/venv/bin/pip
install ordered-set python-dateutil six weakrefmethod

sudo -u www-data /var/www/MISP/venv/bin/pip install
/var/www/MISP/app/files/scripts/misp-stix
```

```
sudo -u www-data /var/www/MISP/venv/bin/pip install
/var/www/MISP/PyMISP
```

Se instala CakePHP y se configura PHP para establecer el servidor web:

```
sudo mkdir -p /var/www/.composer
sudo chown -R www-data: /var/www/.composer
cd /var/www/MISP/app
sudo -u www-data php composer.phar install --no-dev
sudo phpenmod redis
sudo phpenmod gnupg
sudo -u www-data cp -fa /var/www/MISP/INSTALL/setup/config.php
/var/www/MISP/app/Plugin/CakeResque/Config/config.php
```

Establecemos los permisos necesarios para los directorios de MISP:

```
sudo chown -R www-data: /var/www/MISP
sudo chmod -R 750 /var/www/MISP
sudo chmod -R g+ws /var/www/MISP/app/tmp /var/www/MISP/app/files
```

Pasamos a configurar la instancia de MISP con sus ficheros de configuración:

```
sudo -u www-data cp -a /var/www/MISP/app/Config/bootstrap{.default,}.php
sudo -u www-data cp -a /var/www/MISP/app/Config/database{.default,}.php
sudo -u www-data cp -a /var/www/MISP/app/Config/core{.default,}.php
sudo -u www-data cp -a /var/www/MISP/app/Config/config{.default,}.php
```

Configuramos el archivo **database.php** con el siguiente comando:

```
sudo nano /var/www/MISP/app/Config/database.php
```

El archivo debe quedar de la siguiente forma:

```
class DATABASE_CONFIG {

        public $default = array(
                'datasource' => 'Database/Mysql',
                //'datasource' => 'Database/Postgres',
                'persistent' => false,
                'host' => 'localhost',
                'login' => 'mispadmin',
                'port' => 3306, // MySQL & MariaDB
```

```
                //'port' => 5432, // PostgreSQL
                'password' => 'MISP-DB-Password',
                'database' => 'misp',
                'prefix' => '',
                'encoding' => 'utf8',
        );
}
```

A continuación, generamos la *MISP GnuPG key*:

```
sudo -u www-data gpg --homedir
/var/www/MISP/.gnupg --batch --gen-key
~/misp-gpg-batch-file

sudo -u www-data gpg --homedir /var/www/MISP/.gnupg --
export --armor admin@kifarunix-demo.com \
| sudo -u www-data tee /var/www/MISP/app/webroot/gpg.asc
```

Configuramos los workers de la instancia:

```
sudo systemctl daemon-reload
sudo systemctl enable --now misp-workers
systemctl status misp-workers.service
```

Inicializamos la configuración de MISP:

```
sudo -Hu www-data /var/www/MISP/app/Console/cake userInit -q
sudo -Hu www-data /var/www/MISP/app/Console/cake Admin runUpdates
sudo -Hu www-data /var/www/MISP/app/Console/cake Live 1
```

Configuramos el servidor web Apache para MISP:

```
sudo cp /var/www/MISP/INSTALL/apache.24.misp.ssl
/etc/apache2/sites-available/misp.conf

sudo nano /etc/apache2/sites-available/misp.conf
```

Actualizamos las líneas necesarias de este fichero para configurar correctamente la instancia. En vez de un nombre, se puede definir una dirección IP en el ServerName. Instalamos los certificados TLS:

```
sudo openssl req -newkey rsa:4096 -days 365 -nodes -x509 -
subj "/CN=*.kifarunix-demo.com" \
-keyout /etc/ssl/private/misp.local.key -out
/etc/ssl/private/misp.local.crt
```

Por último, se reinicia el servidor:

```
sudo systemctl restart apache2
```

Llegados a este punto, ya somos capaces de acceder via web a la instancia de MISP instalada, con una interfaz como la que se muestra en la Figura A.1.

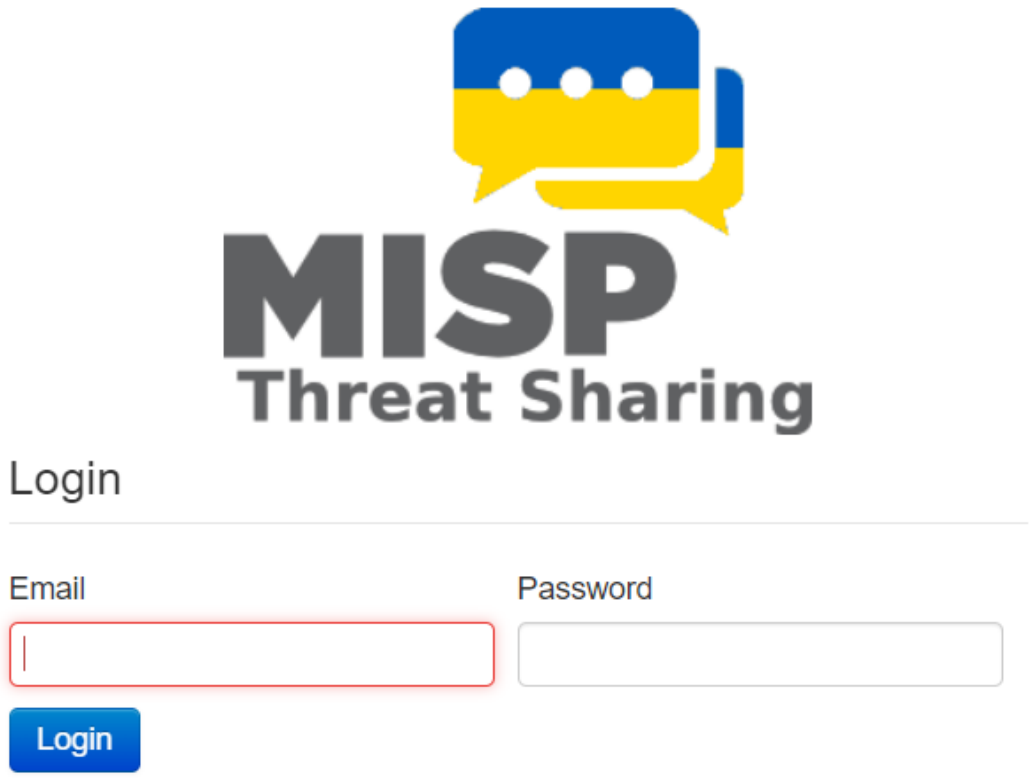


Figura E.1: Interfaz web de la instancia de MISP

Lo siguiente que se debe hacer es configurar los usuarios que acceden a la instancia. El resto de configuraciones se detallan en los capítulos de la memoria.

Anexos F

Instalación de Suricata y Zeek

Para la instalación de ambas herramientas, se recurre a las documentaciones oficiales de cada uno de ellos [15],[16]. En estas páginas, se detallan los pasos a seguir para realizar una instalación limpia de los NIDS. En este anexo únicamente se incluyen las instrucciones de instalación, descrita la configuración en el Capítulo 2 de la memoria.

F.1. Suricata

F.1.1. MV Debian 11

La instalación en la documentación hace referencia al sistema operativo *Ubuntu*, pero dada su similitud con *Debian 11*, la instalación es compatible con el sistema. Se comienza instalando los paquetes necesarios:

```
sudo add-apt-repository ppa:oisf/suricata-stable
sudo apt update
sudo apt install suricata jq
```

Para comprobar la versión, se utilizan los comandos:

```
sudo suricata --build-info
sudo systemctl status suricata
```

F.1.2. Raspberry Pi 4

Para el caso de la *Raspberry Pi 4*, la instalación es ligeramente diferente[22]. Se comienza instalando las dependencias necesarias en el dispositivo:

```
sudo apt install libpcre3 libpcre3-dbg libpcre3-dev build-essential libpcap-dev libyaml-0-2 libyaml-dev pkg-config zlib1g zlib1g-dev make libmagic-dev libjansson-dev rustc cargo python-yaml python3-yaml liblua5.1-dev
```

Se obtiene el código de *Suricata* y lo descomprimimos:

```
wget https://www.openinfosecfoundation.org/download/suricata-6.0.1.tar.gz
tar -xvf suricata-6.0.1.tar.gz
```

Desde la carpeta se configura la instalación:

```
cd $HOME/suricata-6.0.1/
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var --enable-
nfqueue --enable-lua
```

Una vez hecho esto, se procede finalmente con la instalación:

```
make
sudo make install
cd $HOME/suricata-6.0.1/
sudo make install-full
```

F.2. Zeek

F.2.1. MV Debian 11

Se sigue la documentación para la instalación en el caso del sistema operativo utilizado. Se realiza el proceso mediante el paquete binario:

```
echo 'deb http://download.opensuse.org/repositories/security:/zeek/Debian_11/
/' | sudo tee /etc/apt/sources.list.d/security:zeek.list
```

```
curl -fsSL
https://download.opensuse.org/repositories/security:zeek/Debian_11/Release.key
| gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/security_zeek.gpg >
/dev/null
```

```
sudo apt update
sudo apt install zeek
```

Una vez instalado, se debe ajustar la ruta del entorno:

```
export PATH=/opt/zeek/bin:$PATH
```

F.2.2. Raspberry Pi 4

Para la instalación de *Zeek* en el dispositivo, se instala el paquete directamente:

```
dpkg -i zeek_5.1.1_armhf.deb
```

Se instalan las dependencias necesarias para instalar af-packet y se termina con la instalación:

```
apt install python3-venv python3-pip
pip3 install GitPython semantic-version --user
export PATH=/usr/local/zeek/bin:$PATH
zeekctl stop
zkg autoconfig
apt-get install raspberrypi-kernel-headers
zkg install zeek/zeek/zeek-af_packet-plugin
```

Anexos G

Instalación y configuración de Elastic Stack

G.1. Instalación

Se detalla la instalación de las herramientas utilizadas en el proyecto, pertenecientes al producto *Elastic Stack*. En este caso, se trata de las instalaciones de *Elasticsearch, Kibana* y *Filebeat* [23]. Tanto en el sistema operativo *Debian 11* como en la *Raspberry Pi 4*, la instalación de estos paquetes es idéntica.

G.1.1. Elasticsearch

Utilizamos el paquete Debian para la instalación en ambos casos. En primer lugar, se importa la clave PGP del paquete:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --
dearmor -o /usr/share/keyrings/elasticsearch-keyring.gpg
```

A continuación se instala a través del repositotio de APT:

```
sudo apt-get install apt-transport-https
echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg]
https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo tee
/etc/apt/sources.list.d/elastic-8.x.list
```

Por último, se procede a la instalación de paquete Debian:

```
sudo apt-get update && sudo apt-get install elasticsearch
```

Para hacerlo de forma manual, se ejecutan los siguientes comandos (utilizados en la *Raspberry Pi 4*:

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-
8.9.1-amd64.deb
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-
8.9.1-amd64.deb.sha512

shasum -a 512 -c elasticsearch-8.9.1-amd64.deb.sha512
sudo dpkg -i elasticsearch-8.9.1-amd64.deb
```

G.1.2. Kibana

De la misma forma que en la sección anterior, se instala *Kibana*. Los comandos anteriores, al haber sido ejecutados anteriormente, no hace falta su reejecución:

```
sudo apt-get update && sudo apt-get install kibana
```

De forma manual(Raspberry Pi 4):

```
wget https://artifacts.elastic.co/downloads/kibana/kibana-8.9.1-amd64.deb
shasum -a 512 kibana-8.9.1-amd64.deb
sudo dpkg -i kibana-8.9.1-amd64.deb
```

G.1.3. Filebeat

Esta instalación se realiza en una máquina distinta en el escenario de MVs. En la *Raspberry* se hace de la misma forma. La instalación se realiza con el paquete Debian:

```
curl -L -O https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-
8.9.1-amd64.deb

sudo dpkg -i filebeat-8.9.1-amd64.deb
```

G.2. Configuración y automatización

Con el fin de configurar la infraestructura correctamente, se sigue un orden concreto en la puesta a punto de cada herramienta. Se empieza con *Elasticsearch*, después *Kibana* y por último se configura *Filebeat*.

Elasticsearch

Esta herramienta se instala y configura en la MV de *Elastic Stack*, dado que se encargará de la recepción de los datos, además del posterior análisis y búsqueda de los mismos. Como cada elemento en el sistema, *Elasticsearch* contiene sus configuraciones

en un fichero YAML: *elasticsearch.yml*. Para vincular la dirección IP a la herramienta, se utiliza la línea en el fichero:

```
network.bind_host: ["127.0.0.1", "yourprivateip"]
```

Donde "yourprivateip" se sustituye por la IP privada del equipo. La otra dirección es de *loopback*.

Al final del fichero de configuración, se añaden un par de líneas extra:

```
discovery.type: single-node
xpack.security.enabled: true
```

En el *discovery.type* indicamos que el modo de funcionamiento incluye un nodo único (no usamos un clúster). En la siguiente línea, activamos *xpack*, un paquete que proporciona una capa de seguridad extra al programa.

Con este par de configuraciones sencillas, *Elasticsearch* está listo para funcionar. Para iniciarlo se utiliza:

```
systemctl start elasticsearch.service
```

Por defecto, esta herramienta intercambia su tráfico a través del puerto 9200.

Kibana

El siguiente paso, es la configuración de *Kibana*. Este software se ejecutará en la MV de *Elastic Stack*, en paralelo con *Elasticsearch*. El objetivo de la configuración será conectar ambas herramientas a través de protocolos seguros. Durante la instalación de *Elasticsearch*, se genera una *Kibana enrollment-token*, utilizado por Kibana para conectarse al nodo donde se ejecuta *Elasticsearch*. De este modo, con *Elasticsearch* iniciado, arrancamos *Kibana* e introducimos el siguiente comando:

```
bin/kibana-setup --enrollment-token <enrollment-token>
```

Donde *enrollment-token* se sustituye por el *token* en cuestión. Así, conseguimos que, automáticamente, se produzcan las configuraciones de seguridad entre ambos programas.

De nuevo, existe un fichero denominado *kibana.yml* donde se recogen las configuraciones a implementar. Tras ejecutar el comando detallado anteriormente, se añaden unas líneas a este fichero de forma automática, como se muestra en la Figura G.1.

Por último, a este fichero se añade la siguiente línea:

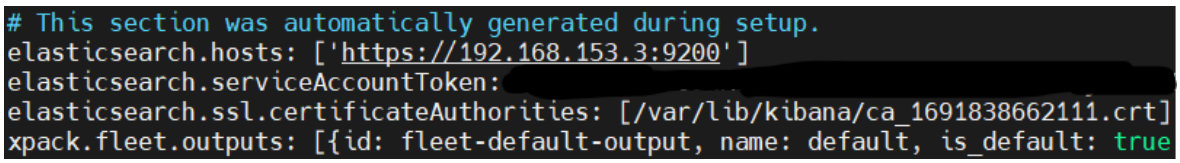


Figura G.1: Apartado del fichero *kibana.yml* donde se detalla la configuración automática.

```
server.host: "yourprivateip"
```

Donde "yourprivateip" se sustituye por la IP privada de la MV donde se ejecuta el software.

En estas condiciones ya se puede iniciar *Kibana*, que creará un servidor web al que se puede acceder desde el navegador, indicando la dirección IP y el puerto 5601. Para inicializar el servicio:

```
systemctl start kibana.service
```

Filebeat

Filebeat se instala en las máquinas donde se generan los logs a analizar. En este proyecto, serán la MV de *Zeek/Suricata* en el entorno de MVs y la *Raspberry Pi 4* en el entorno doméstico. Esta herramienta se conecta a *Elasticsearch* para enviarle los logs generados por los NIDS, que a su vez son exportados a *Kibana* para su visualización.

Tras la instalación, se genera el archivo de configuración *filebeat.yml*. Lo editamos para configurar la conexión con el servidor de *Elasticsearch* y *Kibana*.

La primera configuración del fichero es la conexión con el servidor de *Kibana*. Para ello, añadimos la siguiente línea en el fichero:

```
host: "yourprivateip:5601"
```

Donde "yourprivateip" indica la dirección IP donde se encuentra instalado la herramienta *Kibana*.

A continuación, se configura la conexión con *Elasticsearch*. Para ello, en la parte del archivo que corresponde, se deben añadir las siguientes líneas:

```
hosts: ["https://yourprivateip:9200"]
username: "elastic"
password: "xxx"
ssl:
    enabled: true
    ca_trusted_fingerprint: "xxx"
```

Donde "yourprivateip" es la IP donde se encuentra la herramienta, password es la contraseña creada durante la instalación y *ca_trusted_fingerprint* es el certificado SSL[1] para la autenticidad y cifrado de los datos.

Por último, se añaden los módulos correspondientes a los NIDS para que *Filebeat* extriaga sus *logs* y los exporte hacia *Elasticsearch*. Para activar los módulos se ejecutan los comandos:

```
filebeat modules enable suricata
filebeat modules enable zeek
```

Una vez hecho esto, se crean unos ficheros de configuración, a los cuales podemos acceder para indicarle al programa cuales son los *logs* que se quieren exportar. Estos archivos son *suricata.yml* y *zeek.yml*. En estos archivos, se indican las rutas de los ficheros de *logs* que queremos exportar. En este trabajo, se busca exportar en formato *json* (por simplicidad en el procesado en *Elasticsearch*) los ficheros *eve.json* e *intel.log*. En la figura G.2 se ilustra la estructura del fichero *suricata.yml*.

```
# Module: suricata
# Docs: https://www.elastic.co/guide/en/beats/filebeat/main/filebeat-module-suricata.html

- module: suricata
  # All logs
  eve:
    enabled: true
    # Set custom paths for the log files. If left empty,
    # Filebeat will choose the paths depending on your OS.
    #var.paths:
    var.paths: ["/var/log/suricata/eve.json"]
```

Figura G.2: Archivo *suricata.yml*.

El fichero *zeek.yml* tiene una estructura similar, pero con posiblidad de activar todos los *logs* de los que dispone el programa. En este caso solo se activa *intel.log*.

De esta forma, se da por terminada la configuración de la infraestructura *Elastic Stack*. *Filebeat* recoge los *logs* y se los envía a *Elasticsearch*. Este los procesa y *Kibana* utiliza esa información para visualizarlo en formato de sus *dashboards*.

[1]SSL: *Secure Socket Layer*. Protocolo de seguridad que ofrece privacidad, autenticación e integridad a las comunicaciones en Internet

Anexos H

Código, repositorio de Github

En el repositorio de Github [24], sigue en funcionamiento la automatización de MISP, pudiendo acceder a los ficheros de reglas actualizados diariamente. Por otro lado, en el repositorio se encuentran todos los *scripts* utilizados durante el desarrollo, disponible para aquellos interesados en el proyecto.