UNIVERSIDADE DA CORUÑA

FACULTAD DE INFORMÁTICA

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Itinerario de Ingeniería de Computadores

Implementación de nuevas herramientas de análisis de datos para mejorar la seguridad en el Centro de Supercomputación de Galicia

Autor: Adrián Estévez Barreiro

Directores: Patricia González Gómez

Natalia Costas Lago

Javier Cacheiro López

A Coruña, Febrero de 2019

Agradecimientos

Antes de nada darle las gracias a mi familia, especialmente a mis padres, por apoyarme a lo largo de estos años, y a mi hermana, por ser mi hermana. También a mis amigos, por ser como son y ayudarme cuando los necesito.

Gracias a Patricia González Gómez, a Natalia Costas Lago y a Javier Cacheiro López por brindarme la oportunidad de realizar este proyecto y poder trabajar en el CESGA.

No me olvido de Diego Nieto Caride, que, aunque ajeno al proyecto, me ayudó en todos las dudas que pude tener a lo largo de las prácticas. Tampoco de Juan, Pablo y Paco, que me hicieron sentir integrado desde el primer día.

Finalmente, agradecer al resto de compañeros del CESGA su amabilidad, cortesía, y profesionalidad en el trato, y a todos los profesores de la FIC por ayudarme a llegar hasta aquí.

Resumen

A día de hoy, todos los equipos conectados a la red son un objetivo potencial de los ciberdelincuentes, pero en especial los sistemas de grandes empresas, administraciones públicas e investigación son los más afectados. Estos sistemas manejan gran cantidad de información que deben analizar con el fin de encontrar amenazas, tales como posibles ataques externos. Igual de importante es controlar el estado de todos los dispositivos que forman parte del sistema para poder prestar el mejor servicio con la mayor seguridad.

El Centro de Supercomputación de Galicia (CESGA) ya dispone de múltiples sistemas para monitorizar y obtener información del estado de las infraestructuras y servicios desplegados. Esta información es diversa: métricas y registros de servidores y equipamiento de red, así como del equipamiento de soporte (climatización, suministro eléctrico, etc.). En la actualidad, se vuelca gran parte de dicha información en un repositorio central, basado en *Apache Hadoop*, para su análisis. Este análisis es limitado, pues depende en gran medida de los administradores. Por ello, y debido a la gran cantidad de datos manejados en el centro, es necesario buscar una alternativa que permita automatizar el análisis de registros y métricas obtenidas de las diferentes infraestructuras y servicios desplegados. Para ello, se ha analizado la solución actual, se han estudiado diferentes herramientas, se ha realizado el despliegue de una de ellas, sobre un entorno de pruebas para conocer sus componentes, y, finalmente, se han introducido fuentes de datos reales para su análisis.

Palabras clave

Análisis de Datos, Apache Metron, Big Data, Ciberseguridad, Hadoop

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

En los últimos años la cantidad de empresas con servicios disponibles a través de Internet ha aumentado exponencialmente. En 1989 se inventa la *World Wide Web*. Dos años después, en 1991, aparece la primera Web. Hoy, ese número ha aumentado hasta superar los mil millones de páginas web, con un total de 4 mil millones de usuarios conectados a la red. En la figura 1.1 se puede ver la distribución de estos usuarios en los diferentes países, con China y la India a la cabeza. Como contrapartida, en la figura 1.2 se puede ver la distribución de los ciber-ataques, comprobando que la mayoría de los ataques no son dirigidos a países con mayor población, sino a aquellos con mayor riqueza, lo que indica que los objetivos principales son empresas y administraciones.

De todos modos, esta cantidad de usuarios conectados aumenta las oportunidades y número de ciber-delincuentes, por lo que no es de extrañar que, cada año, el cibercrimen genere unos costes estimados de 450 mil millones de dólares [41].

Pero este problema no es nuevo, pues ya entre los años 2005 y 2008 el cibercrimen observa un aumento considerable, demostrando la escasez de expertos en seguridad, que eran conglomerados por grandes multinacionales con servicios bancarios y financieros, así como grandes hospitales y empresas de telecomunicaciones y defensa. Esta acumulación de especialistas dejaba

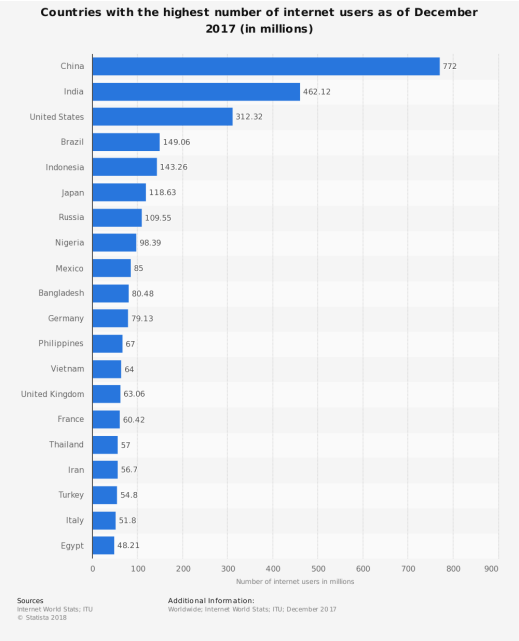

Figura 1.1: Número de usuarios de Internet por país

al resto de organizaciones sin un equipo de seguridad fiable. En 2015, y sólo en Estados Unidos, un total de 209.000 puestos relacionados con la ciberseguridad quedaron sin cubrir, y el déficit de profesionales de este campo ronda los 825.000 dentro de la UE [42]. En esa época, empresas como Cisco aprovechan esta demanda, utilizando la experiencia en seguridad que habían acumulado a lo largo del tiempo para crear servicios que permitieran a estas organizaciones sin recursos manejar las diferentes tareas de seguridad.

A partir de 2008 llega la era del *Big Data*, aumentando considerablemente la cantidad de datos que debían ser analizados, así como el precio de los diferentes sistemas de gestión de eventos e información de seguridad, conocidos como SIEM, por sus siglas en inglés "*Security Information and Event Management*". Esto dificultaba la capacidad de Cisco de operar sus Centros de Operaciones de Seguridad (SOC), y los vendedores de dispositivos de seguridad comienzan a dominar el mercado.

En 2013 Cisco decide crear su propia herramienta, apareciendo OpenSOC, con la idea de, no solo reemplazar las herramientas existentes, sino de

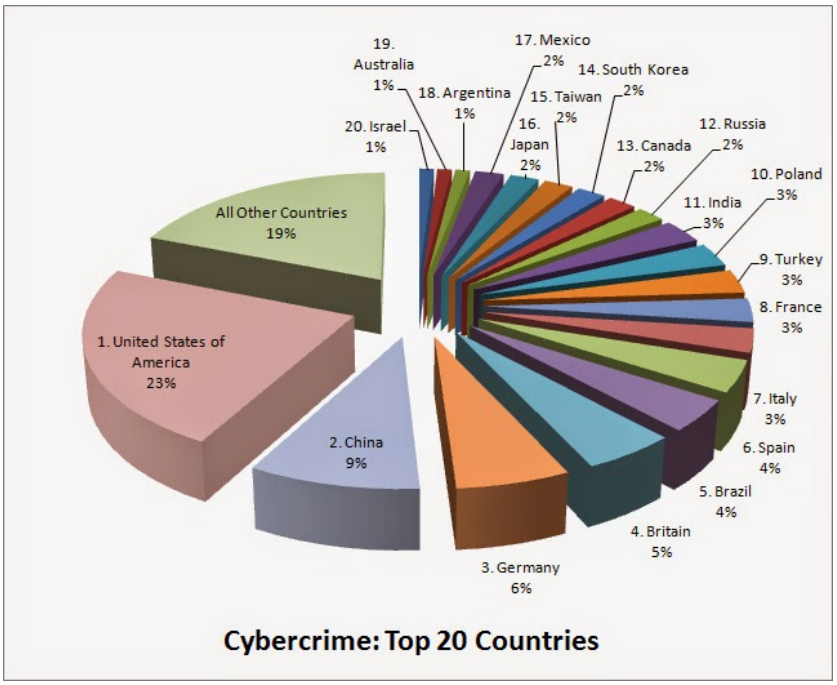

Figura 1.2: Porcentaje de ciberataques por país. Fuente [45]

mejorarlas y modernizarlas a través de la filosofía *open source*. OpenSOC es el primer proyecto que combina herramientas enfocadas al tratamiento masivo de datos, como Storm [5], Hadoop [4] y Kafka [12], creando una solución potente y con visión de futuro. Durante este tiempo, Hortonworks comienza a apoyar el proyecto de Cisco, y juntos trabajan para crear servicios SOC construidos a partir de otras tecnologías de análisis de datos de código abierto.

En estas circunstancias, y conociendo el crecimiento del valor del mercado de la ciberseguridad, que ha pasado de unos 3.5 mil millones de dólares en 2004 a más de 100 mil millones en 2017 [44], otros gigantes de la industria como Accenture, Cloudera, Dell, Intel o McAfee también se han decidido a desarrollar herramientas que permitan reducir los riesgos inherentes a la ciberseguridad. El principal objetivo de estas herramientas es utilizar diferentes técnicas de análisis de datos para mejorar la detección de amenazas, responder a ataques y cuantificar posibles pérdidas de datos, con el fin de

poder reducir costes, daños y el cibercrimen en general.

1.1. Objetivos

El Centro de Supercomputación de Galicia (CESGA) [39] es un centro de cálculo de altas prestaciones cuyo cometido es proporcionar servicios avanzados de computación a la comunidad científica gallega, al Sistema Universitario Gallego (SUG) y al Consejo Superior de Investigaciones Científicas (CSIC), así como a múltiples centros de desarrollo I+D dentro de nuestra comunidad autónoma.

La exposición de los diversos servicios a Internet para que sean accesibles por sus usuarios hace que puedan ser objetivo de ciberataques con diversos propósitos (obtención de información científica/industrial, alojamiento de contenidos ilícitos, control de servidores para incorporación a redes de comando y control, etc). Además, la gran diversidad de sistemas, en relación a sistemas operativos y software, dificulta la aplicación de políticas básicas que faciliten la securización de los mismos.

Actualmente, el centro cuenta con múltiples sistemas de monitorización y obtención de métricas e información de las infraestructuras y servicios desplegados. Esta información es diversa, pues incluye métricas y registros de servidores y equipamiento de red, así como información sobre el equipamiento de soporte. Estos datos se almacenan en un servidor central de la plataforma Big Data del centro, basada en un clúster de Apache Hadoop, para su posterior análisis por parte del personal técnico responsable de las diferentes infraestructuras y sistemas.

Por lo tanto, con el fin de mejorar la seguridad del centro, el objetivo de este proyecto es implementar mecanismos de detección de ataques y vulnerabilidades en las infraestructuras y servicios del CESGA, de forma que se puedan aplicar medidas reactivas desde el punto de vista de la seguridad que no precisen de la intervención de un administrador. Para ello:

1. Se ha revisado la cantidad y calidad de la información incorporada al

servidor central, al mismo tiempo que se analizaba el entorno actual.

2. Se han estudiado las diferentes herramientas de código abierto de detección de amenazas disponibles, teniendo en cuenta los requisitos del centro, y evitando interferir en la plataforma de producción, que aloja la información en la actualidad.

3. Se ha desplegado la solución escogida sobre un entorno de pruebas virtual.

4. Se han añadido fuentes de datos reales a la herramienta para simular su funcionamiento en un entorno real.

1.2. Planificación y Costes

Antes de comenzar el proyecto se realizó una planificación del mismo, dado que se planteaba en el contexto de unas prácticas extracurriculares. Para ello se establecieron las diferentes fases del proyecto, que se muestran a continuación:

- **Fase 1:** Análisis del sistema actual y de la cantidad y calidad de la información recogida para su análisis.
- **Fase 2:** Búsqueda de alternativas, analizando las ventajas e inconvenientes de las diferentes soluciones y la viabilidad de su despliegue.
- **Fase 3:** Despliegue de la solución escogida en un entorno de pruebas virtual.
- **Fase 4:** Configuración y administración de la herramienta para conocer sus capacidades.
- **Fase 5:** Implementación de diferentes sensores en la herramienta, utilizados para probar el análisis de datos con fuentes controladas.
- **Fase 6:** Análisis de datos de fuentes reales.

Rol	Horas	Precio Hora	Total
Alumno	440	20	8800
Director UDC	20	30	600
Director CESGA 1	50	30	1500
Director CESGA 2	50	30	1500
Total	-	-	12400

Cuadro 1.1: Costes del proyecto

Además, durante todo el desarrollo del proyecto, se han documentado los procesos convenientemente para los técnicos del CESGA, de modo que una vez finalizado el trabajo pudieran revisar, modificar y añadir funcionalidades a la solución desplegada.

En la figura 1.3 se puede observar planificación junto con la programación temporal. Mediante un diagrama de Gantt se representan las diferentes tareas a realizar y su extensiónn, distribuyéndolas a lo largo de la duración el proyecto.

Todos los equipos, máquinas y recursos necesarios serán proporcionados por el CESGA, por lo que no se calculan los costes generados por estos elementos. En cuanto a los recursos humanos, en el análisis se tienen en cuenta a cuatro personas, con diferente grado de implicación en el proyecto: el alumno, dos directores, y un técnico del CESGA. Como se puede ver en el cuadro 1.1, la estimación final es de 480 horas, y un coste total de 9500.

1.3. Estructura de la memoria

El contenido del documento se estructura de la siguiente manera. El primer capítulo expone las motivaciones del proyecto y la línea de desarrollo del mismo. En el capítulo 2 se analiza el entorno actual y, en él, se especifican los requisitos que debe cumplir la nueva herramienta con el fin de buscar la solución más conveniente. Posteriormente se describen las soluciones dispo-

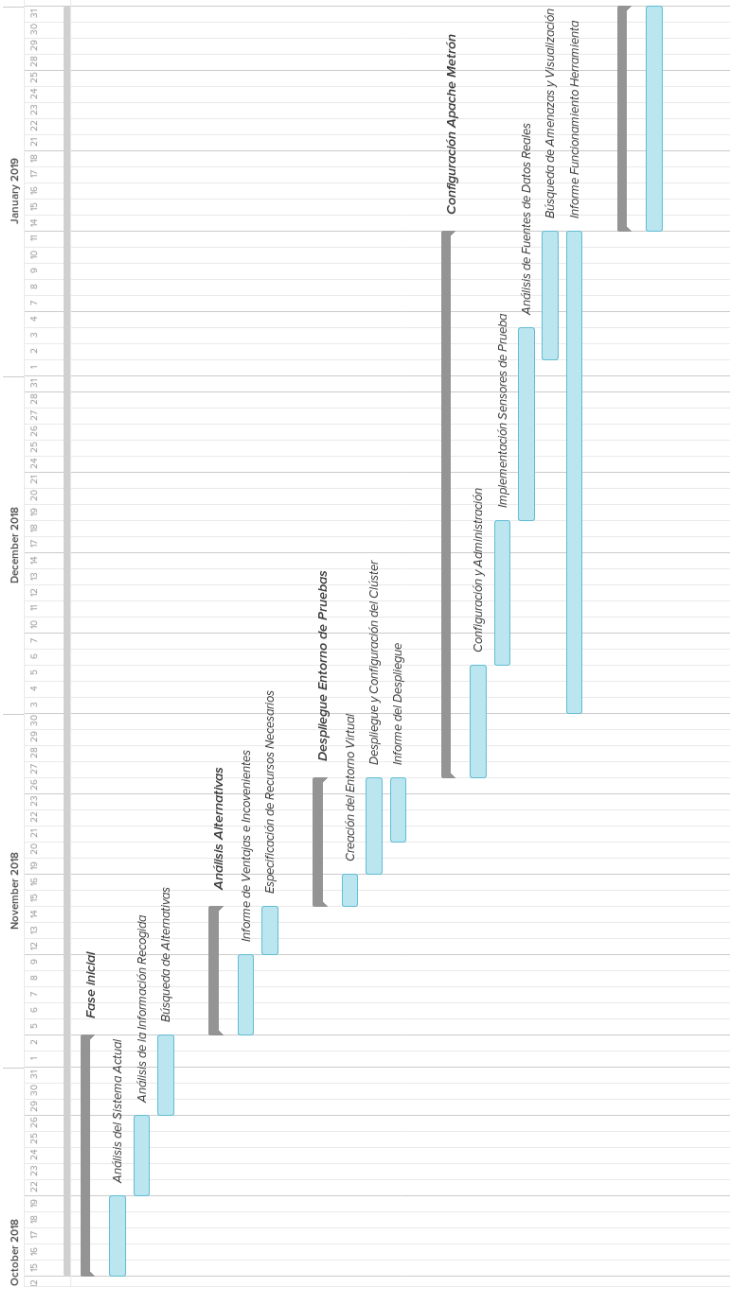

Figura 1.3: Diagrama de Gantt con la planificación del proyecto

nibles y se decide cuál es la más adecuada. En el capítulo 3 se describen los diferentes componentes de la herramienta detalladamente, con el fin de conocer la solución más a fondo antes de instalarla. En el capítulo 4 se detalla el despliegue de la herramienta en un entorno de prueba para poder conocer a fondo sus características y grado de integración con el entorno real. En el capítulo 5 se detalla la implementación de la solución, así como los diferentes pasos realizados en el análisis de datos. También se comenta la introducción y análisis de datos reales obtenidos en el CESGA. Finalmente, en el capítulo 6 se recogen las conclusiones extraídas del trabajo realizado y se exponen diferentes posibilidades para continuar el proyecto.

Adicionalmente, se incluyen diferentes apéndices con información de carácter técnico, como apoyo al trabajo expuesto. El apéndice A detalla la instalación de Apache Metron en un clúster HDP 2.5 con máquinas CentOS 7. El apéndice B detalla la instalación de diferentes herramientas utilizadas en Apache Metron para capturar datos, y la configuración de los diferentes procesamientos. Finalmente, el apéndice C contiene información adicional a tener en cuenta a la hora de administrar Apache Metron.

Capítulo 2

Antecedentes y análisis de soluciones

En este capítulo se realizará un estudio del entorno actual usado en el CESGA para recoger y tratar los datos en busca de amenazas para las infraestructuras del centro, poniendo especial atención en las carencias del mismo, y los requisitos planteados por el centro, a la hora de buscar una solución. También se analizarán algunas de las diferentes soluciones disponibles.

2.1. Entorno actual

La plataforma Big Data del CESGA [40], donde están implementados todos los servicios de seguridad y control, es un clúster de Apache Hadoop basado en la plataforma de datos de Hortonwork, haciendo uso de la versión HDP 2.4, aunque se planea una actualización a la plataforma de Cloudera.

Este clúster está formado por un total de 38 nodos. Todos ellos tienen 2 CPUs *Intel Xeon E5-2620 v3*, 12 (2x6) cores, 64GB de RAM y conectividad 10GbE. Los 4 nodos maestros disponen de 8 discos SSD de 480GB, mientras que los 34 esclavos restantes usan 12 discos SATA de 2TB, resultando en un almacenamiento total de 816TB.

Este entorno se administra a través de *Apache Ambari* [2], y su arquitec-

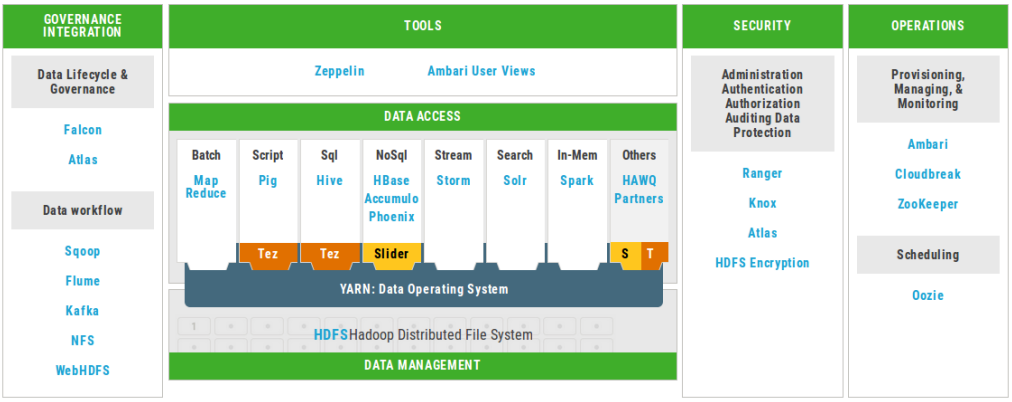

Figura 2.1: Arquitectura del entorno Big Data del CESGA

tura es similar a la representada en la figura 2.1. Usa HDFS como sistema de ficheros distribuido y YARN como gestor de recursos, permitiendo repartir tanto almacenamiento como recursos a la hora de completar las diferentes tareas. Encima de ellos existen diferentes servicios, como Kafka, HBase o Hive, necesarios para la ejecución de diferentes aplicaciones en la plataforma.

Para recoger y guardar registros y métricas se utilizan dos sistemas diferentes. En el caso de los registros, *Apache Flume* [3], un servicio distribuido que permite recolectar y mover grandes cantidades de datos, se encarga de recibir los mensajes y reenviarlos a través de Kafka hacia HDFS. Este almacenamiento se puede consultar a través de Hive, mediante consultas SQL. Además, estos datos también se almacenan en ElasticSearch mediante Filebeat. Para guardar las métricas obtenidas de las infraestructuras del centro, se utiliza una base de datos especializada en métricas, llamada OpenTSDB, que funciona sobre HBase. Estos datos se pueden consultar a través de las herramientas de visualización *Grafana* y *Jupyter*.

Para realizar el análisis de los registros de datos se utiliza *Apache Spark* [10], un motor de análisis de datos a gran escala. Este análisis se realiza ejecutando diferentes scripts sobre los datos almacenados en HDFS. A modo de ejemplo, uno de estos scripts comprueba la IP de todas las conexiones SSH fallidas, creando una lista negra con todas aquellas que superen 5 intentos

en un tiempo determinado. Una vez creada, esta lista se bloquea durante 2 días, rechazando las conexiones SSH provenientes de esas IPs.

Estos scripts deben ser programados por un administrador. Esta programación no es trivial, y conlleva un trabajo extra para cada nuevo análisis que se quiera realizar, pues supone un proceso de desarrollo y búsqueda de algoritmos y herramientas adecuadas para los diferentes casos de uso. Es por ello que el objetivo de este proyecto es encontrar una herramienta capaz de facilitar este proceso al administrador, ofreciendo una solución más efectiva y eficiente al análisis actual.

2.2. Estudio de mercado

A la hora de escoger la herramienta más adecuada debemos comenzar por estudiar las necesidades del CESGA. En los siguientes apartados se abordará la recogida de requisitos, se mostrarán las herramientas encontradas y por qué son seleccionadas o descartadas. Como último paso se estudiarán con más profundidad las herramientas más prometedoras, antes de tomar la decisión final sobre el despliegue a realizar.

2.2.1. Requisitos

Con el fin de facilitar la integración de la solución en el entorno de seguridad actual, y cumplir con el presupuesto y filosofía de trabajo del centro, se estipula que la herramienta seleccionada cumpla los siguientes requisitos:

- Ha de ser una herramienta Open Source.
- Ha de permitir diferentes fuentes de datos.
- Ha de ser fácilmente integrable en el entorno actual.
- Tiene que ser compatible con otras herramientas similares.
- Debe permitir el tratamiento de los diferentes tipos de datos.

- Ha de estar orientada al tratamiento de grandes conjuntos de datos (Big Data).

Aunque todos los requisitos son igual de importantes, se pone especial énfasis en que la solución permita analizar múltiples fuentes de datos, que esté orientada al tratamiento de grandes conjuntos de datos, y se pueda integrar en el entorno actual, sin substituirlo. También es especialmente relevante que cumpla con la filosofía de Software Libre y Código Abierto (FOSS).

2.2.2. Posibles Soluciones

Una vez conocido el entorno y recogidos los requisitos necesarios se inicia la búsqueda, intentando encontrar el mayor número de herramientas con el fin de tener un amplio abanico de soluciones, representado en la siguiente lista:

- Splunk [35]
- Loggly [24]
- Loom [26]
- Graylog [17]
- PandoraFMS [30]
- Logcheck [22]
- Logwatch [25]
- Fail2Ban [16]
- Apache Metron [6]
- Apache Spot [11]

Se ha estudiado cada una de las herramientas, teniendo en cuenta sus características y como se adecuan al caso de estudio. Inicialmente se descartan las soluciones propietarias y/o de pago, como Splunk, Loggly o Loom, así como aplicaciones que implican el despliegue de una solución completa que obligaría a sustituir el entorno actual, como Graylog o PandoraFMS.

También se descartan las soluciones más básicas que apenas aportan funcionalidades, con las que no sería posible conseguir el objetivo planteado: Logcheck, Logwatch y Fail2Ban.

Finalmente, la lista queda reducida a dos soluciones que podrían resultar adecuadas: Apache Metron y Apache Spot. Las dos son herramientas de código abierto muy potentes, enfocadas al análisis de grandes conjuntos de datos, que siguen la filosofía de trabajo de la Apache Software Foundation (ASF) y que permiten tanto soluciones completas como parciales, es decir, permiten integrar la herramienta en un entorno ya existente sin sustituirlo.

2.2.3. Apache Metron

Apache Metron [6] es una herramienta que permite centralizar las tareas de seguridad, análisis y monitorización. Para ello integra diferentes tecnologías de código abierto enfocadas al análisis de grandes cantidades de datos, ofreciendo la capacidad de agregación de registros, indexado de paquetes capturados, almacenamiento, análisis de comportamiento y enriquecimiento de datos en la misma plataforma.

Metron no nace como un proyecto propio, si no a partir de OpenSOC, desarrollado por Cisco [46]. En abril de 2015 OpenSOC comienza la producción para diferentes clientes, y al poco tiempo adquiere tecnologías de terceros, convirtiendo el proyecto en una versión propietaria basada en hardware.

En diciembre de ese mismo año Metron es aceptado en la incubadora de Apache, dejando de llamarse OpenSOC, y se empieza a construir una arquitectura abierta que integra gran cantidad de herramientas usadas en diferentes entornos, haciéndolo fácilmente adaptable a los casos de uso de

las diferentes organizaciones. Finalmente, en abril de 2016, sale a la luz la primera versión oficial de Apache Metron (0.1).

Funcionalidades

Metron tiene cuatro funcionalidades principales:

- Capturar, almacenar y normalizar cualquier tipo de métrica de seguridad a gran velocidad, ya que estos datos están siendo generados continuamente y deben ser reenviados a las diferentes unidades de procesamiento con la menor latencia posible.
- Procesar y enriquecer los datos recibidos para conocer el contexto y urgencia de los mismos y así poder investigarlos.
- Almacenar eficientemente la información dependiendo de como vaya a ser usada:
  - Registros de datos y métricas se almacenan de manera que su análisis sea lo más eficiente posible.
  - Paquetes extraíbles y totalmente reconstruíbles para facilitar su análisis.
  - Almacenamiento a largo plazo que permite realizar análisis más avanzados mediante técnicas de aprendizaje automático que crean modelos de información contra los cuales se pueden comparar los datos futuros.
- Presentar una interfaz que ofrece una visión centralizada de los datos y alertas del sistema, con resúmenes, búsqueda avanzada y herramientas de extracción de paquetes para que el analista no necesite acudir a herramientas adicionales.

Se puede decir, por lo tanto, que el funcionamiento de Metron se divide en las fases de: captura, procesamiento, almacenamiento y visionado.

Requerimientos

Apache Metron ofrece gran flexibilidad a la hora de ser desplegado en diferentes sistemas, pero precisa gran cantidad de memoria ya que hace un uso bastante intensivo de la misma. Además, se recomienda el uso de diferentes nodos con el fin de paralelizar los diferentes pasos del análisis de datos.

En el caso de instalar únicamente el núcleo de Metron sería necesario tener instalado un clúster de Hadoop con los siguientes servicios:

- HBase
- HDFS
- Kafka [5]
- Storm [12]
- Zookeeper [13]

Además, se deberá instalar *ElasticSearch* [15], un servidor de búsqueda desarrollado en Java bajo licencia Apache, que provee un motor de búsqueda de texto completo, distribuido, con una interfaz web y documentos JSON, para indexar los datos analizados. También *Kibana* [21], una plataforma de código abierto diseñada para analizar y visualizar los datos almacenados en ElasticSearch. Aunque existen otras soluciones para realizar este indexado y visionado, Apache Metron recomienda estas dos herramientas.

Existe una solución que permite crear el clúster de Hadoop junto a todos los servicios necesarios cuando se procede a instalar Apache Metron, lo que minimiza el esfuerzo.

Arquitectura

Metron es, en esencia, una arquitectura Kappa [27] [20] con Apache Storm como componente de procesamiento y Kafka como bus de datos. Es decir, una plataforma que utiliza *streaming* de datos mediante Kafka para pasar la información a los diferentes componentes de procesamiento de Metron, que

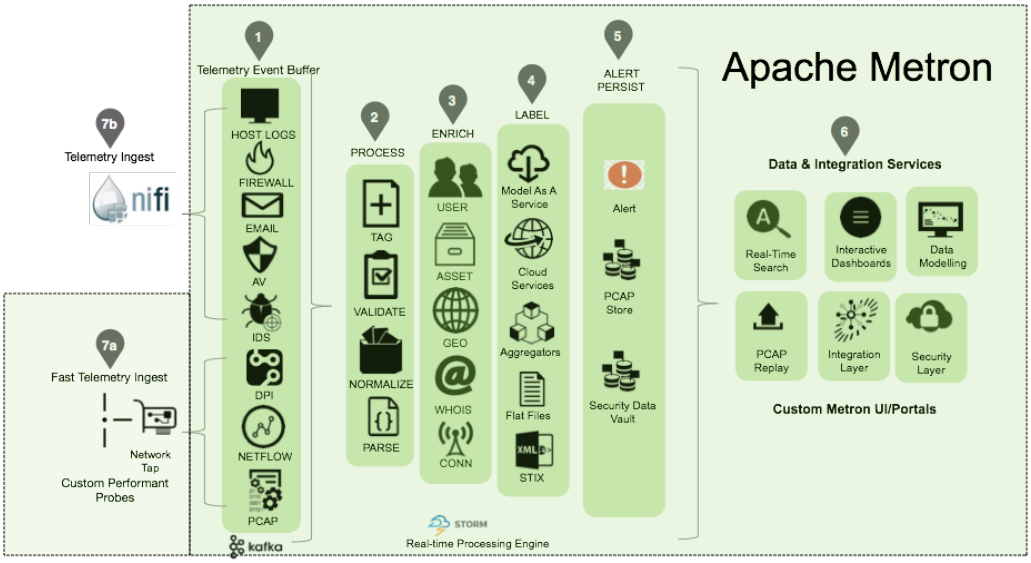

Figura 2.2: Arquitectura de Metron

gestiona Apache Storm. Este flujo permite no tener que almacenar los datos según se van recolectando, y leerlos para su análisis, reduciendo la latencia del proceso.

En la figura 2.2 se pueden observar las diferentes fases de esta arquitectura, explicadas a continuación.

1. **Buffer de métricas:** todos los eventos capturados por Ni-Fi u otros sistemas de captura personalizados se envían a través de su propio flujo (*topic*, usando su terminología) en Kafka, dando comienzo al procesamiento.

2. **Procesamiento:** dividido en parseo, normalización, validación y etiquetado, cada evento se convierte en una estructura JSON estándar para que el sistema pueda correlacionar los diferentes mensajes a partir de estos campos. En esta fase también se pueden añadir metadatos para ser usados en el procesamiento posterior.

3. **Enriquecimiento:** en este paso se añaden datos como la geolocalización de la IP o información sobre el dueño de la misma con el fin de

aumentar la información disponible a la hora de analizar las capturas.

4. **Búsqueda de amenazas:** en este punto las métricas se comparan con las aportadas por fuentes de datos de información sobre amenazas como Soltra [34] o Stix/Taxii [37], entre otros. En esta fase se etiquetan las métricas como amenazas si existen coincidencias. También se pueden catalogar las métricas a través de modelos analíticos desarrollados por el usuario.

5. **Alertas y persistencia:** en esta fase se pueden generar alertas a partir de eventos, dependiendo de factores como el tipo de evento o si ha sido catalogado previamente como amenaza. En este paso todos los eventos se almacenan en HDFS y ElasticSearch permitiendo acceder a ellos en análisis futuros.

6. **Visionado de datos e integración de servicios:** gracias a la división de tareas en diferentes pasos, Metron permite integrar diferentes servicios de manera efectiva.

    - Búsqueda en tiempo real y centros de control interactivos que permiten a los analistas ver las diferentes alertas y eventos detectados.
    - Modelado de datos, facilitado por la normalización y enriquecimiento de los datos almacenados.
    - Integración y capas de extensión que permiten adaptar el producto a cada usuario según sus necesidades y requisitos. Algunas de ellas son: obtener datos de diferentes fuentes, añadir parseadores, enriquecedores de datos o información de amenazas, integración con motores de flujo de datos empresariales y personalizar los portales y centros de control.

7. **Obtención de métricas:**

a **Ingesta de métricas a alta velocidad:** permite obtener datos directamente de los dispositivos de red que estén realizando mediciones de gran volumen.

b **Ingesta de registros:** para fuentes de datos que utilizan protocolos se usará Apache Ni-Fi.

2.2.4. Apache Spot

Apache Spot [11] es una herramienta de código abierto de análisis de métricas, flujos de datos y paquetes que permite mejorar la visibilidad de la red y detectar diferentes amenazas y ataques. Apache Spot aumenta la capacidad de los usuarios para descubrir conexiones sospechosas y ataques inadvertidos gracias a sus tecnologías de análisis.

Apache Spot nace a partir de un proyecto de Intel conocido como Open Network Insight (ONI), lanzado en febrero de 2016. En septiembre de ese mismo año el proyecto es donado a la Apache Software Foundation (ASF), tomando el nombre de Apache Spot.

Funcionalidades

Apache Spot está formado por 4 componentes:

- **spot-setup:** formado por scripts que generan las rutas de HDFS, tablas de Hive y configuración de Apache Spot.
- **spot-ingest:** que captura los binarios y ficheros de registros para enviarlos al clúster de Hadoop, donde se transforman y almacenan.
- **spot-ml:** que incluye algoritmos de aprendizaje automático para detectar anomalías.
- **spot-oa:** que añade a la salida del aprendizaje automático contextos y heurísticas y permite al usuario visualizar estos datos.

Conociendo esos cuatro componentes se puede dividir la actividad de Apache Spot en 3 fases, ya que el setup únicamente se ejecuta cuando se crean los directorios de HDFS y las bases de datos de *Impala*, un motor de consultas SQL de código abierto de Cloudera para el procesamiento masivo de los datos almacenados en un clúster de Apache Hadoop. Estas fases serían la captura, el análisis mediante aprendizaje automático y la visualización.

Requerimientos

Apache Spot no ofrece una solución completa como Apache Metron, si no que es necesario tener un clúster de Hadoop preinstalado con los siguientes servicios:

- HDFS
- Hive
- Impala
- Kafka
- Spark[1]
- Yarn
- Zookeeper

Cabe destacar, que aunque se pueden instalar juntos, se recomienda que los cuatro módulos que conforman Spot se distribuyan por los diferentes nodos del clúster de Hadoop para mejorar su rendimiento.

Arquitectura

La arquitectura de Apache Spot se divide en múltiples fases [36], conectando los diferentes módulos a través de Kafka y el almacenamiento de HDFS. Esta arquitectura se puede observar en la figura 2.3.

[1]Versión mínima requerida 2.1.0

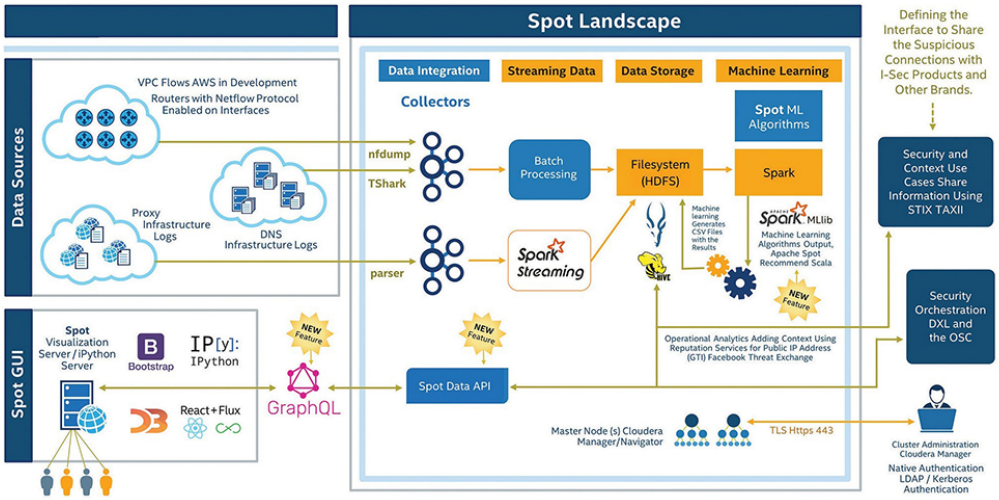

Figura 2.3: Arquitectura de Spot

1. **Fuentes de Datos:** Spot puede obtener directamente datos de los flujos de red (nfdump), DNS (TShark) y Proxies (parser). Estos datos también se pueden obtener a través de un *sistema de gestión eventos e información de seguridad* (*SIEM*) o un servidor de registros de datos. Si fuera necesario obtener datos de otras fuentes, esto se puede realizar a través de un *Open Data Model* (ODM). Para manejar toda esta cantidad de datos, Spot ofrece una solución rápida, distribuida y escalable, con una disponibilidad de prácticamente el 100 % y una pérdida de datos nula.

2. **Almacenamiento:** una vez recogidos los datos, estos se transforman y posteriormente se almacenan tanto en HDFS como en Hive. Cuando se ha recogido suficiente información (3 o 4 horas), esta se analiza usando aprendizaje automático.

3. **Análisis y aprendizaje automático:** encargado de encontrar conexiones sospechosas. Se genera una lista con los eventos considerados menos probables y más extraños. Uno de los métodos utilizados es el *topic modeling*, que permite encontrar comportamientos extraños en las

estructuras semánticas de los registros de las diferentes IPs. Para ello hace uso de *Latent Dirichlet Allocation* (LDA), un modelo generativo que permite asignar una categoría concreta a un valor a partir de unos conjuntos ya determinados. Es decir, dependiendo de la forma del mensaje recibido, este se trata de diferente manera, permitiendo encontrar datos extraños o amenazas. También se infiere un modelo probabilístico para el comportamiento de cada IP, que asigna una puntuación a cada entrada según cómo de sospechoso sea.

4. **Analíticas:** se añade un contexto, así como un enriquecimiento, a los resultados del aprendizaje automático, permitiendo acelerar la detección de indicadores de compromiso.

5. **Visualización:** mediante la interfaz gráfica que proporciona Spot se pueden observar los resultados más sospechosos, aunque también se pueden buscar y crear historiales de amenazas. Esta interfaz aprovecha las últimas tecnologías web para mejorar la experiencia y accesibilidad de la herramienta. Por ejemplo, mediante *IPython notebooks* se pueden crear paneles personalizados, que permitan al administrador escoger el método de visualización y filtrar la información.

2.3. Elección de herramienta

Como se puede observar, tanto Apache Metron como Apache Spot son herramientas de análisis muy potentes, pensadas para infraestructuras informáticas de gran capacidad donde se manejan gran cantidad de datos de eventos, lo que dificulta el análisis de amenazas y ataques.

Ambas herramientas permiten capturar y analizar eventos y registros de diferentes fuentes, hacen uso de aprendizaje automático para acelerar la detección de problemas, e incluyen una potente interfaz gráfica que facilita el trabajo al encargado de la seguridad, además de tener gran cantidad de documentación disponible.

Una de las principales diferencias se observa a la hora de instalar las dos herramientas, pues, aunque ambas dependan de un clúster de Hadoop para funcionar y se puedan integrar en uno fácilmente, solo Apache Metron permite crear este clúster en el momento de la instalación. Mientras, Apache Spot debe ser desplegado en un entorno donde tanto Hadoop como los servicios necesarios ya están instalados.

El momento en el que se almacenan los datos también difiere entre las dos herramientas, pues mientras que Apache Metron lo hace después de realizar todo el análisis, Apache Spot lo hace antes, almacenando los datos capturados, para posteriormente analizarlos.

También hay diferencias a la hora de integrar las herramientas con otros servicios diferentes a los propios del clúster de Hadoop, ya que Apache Metron permite añadir diferentes herramientas al clúster de manera sencilla, permitiendo al usuario incluir herramientas con las que ya está familiarizado. De esta manera la "experiencia" es más personalizada. En cambio, Apache Spot no da esta facilidad, si no que ofrece un único entorno al que el usuario se debe acostumbrar.

Otras diferencias que podríamos apuntar son que Apache Metron es respaldado por Hortonworks [19], mientras que Spot es apoyado por Cloudera [14], y que, a la hora de manejar y visualizar los datos, Apache Metron usa ElasticSearch y Kibana, mientras que Spot usa Impala.

La última versión de Metron es la 0.6.0, de septiembre de 2018, mientras que la última versión de Spot es la 1.0, de agosto de 2017.

Una vez estudiadas las características de ambas herramientas y conociendo el entorno en el que va a ser aplicado, se escoge Apache Metron como herramienta de análisis, ya que se integrará con mayor facilidad en el entorno actual del CESGA, es más estable y permite añadir tanto múltiples fuentes de datos como nuevos servicios.

Capítulo 3

Apache Metron

En este capítulo, se estudian los diferentes módulos de Metron y las herramientas y servicios que intervienen en el proceso de análisis de datos y detección de ciberataques. De esta manera se pretende tener un mayor conocimiento y control de la herramienta con el fin de adaptarla a las necesidades del centro.

3.1. Interfaces y herramientas

El primer paso para conocer Apache Metron es saber como funciona su entorno y como administrarlo. Se divide este estudio en dos categorías, las interfaces y las herramientas de la arquitectura Metron.

En el grupo de interfaces destacan las siguientes:

- **Ambari:** esta interfaz permite controlar y configurar los diferentes componentes del clúster de Hadoop, iniciar y apagar tanto máquinas como servicios, y cambiar la configuración de las herramientas.
- **Metron-UI:** permite controlar y configurar los llamados *sensores*, que especifican como debe tratar Metron una fuente de datos concreta.
- **Kibana:** permite visualizar, de manera gráfica y sencilla, los datos almacenados en ElasticSearch.

En el grupo de herramientas de la arquitectura Metron, las principales son:

- **Storm:** se encarga de gestionar los procesos que conforman los diferentes pasos de una tarea concreta, repartiéndolas entre los nodos encargados de ejecutar estas tareas. Storm llama a este conjunto de procesos topología (*topology*), nombre también utilizado en la arquitectura de Metron, por lo que se usará este término para referirnos a ellos.

  Las topologías se dividen en dos grupos, uno formado por los procesos que reciben datos (llamados *spouts*), escuchando a un flujo de Kafka, y otro formado por los procesos que transforman y redirigen los datos (llamados *bolts*), escribiendo a un nuevo flujo de Kafka o almacenándolos directamente.

- **Kafka:** es el encargado de gestionar el paso de mensajes entre las diferentes fases del procesamiento, desde que se capturan los datos hasta que son almacenados. Para ello usa diferentes flujos (*topics*, en la terminología de Kafka), que sirven para enviar la información al destino correspondiente.

- **Zookeeper:** se encarga de mantener la sincronización entre los diferentes nodos, indicando a todos ellos cual es la configuración actual que deben utilizar. Además, trabaja junto a Apache Kafka controlando los diferentes flujos de datos.

- **HBase:** es la base de datos de Hadoop, ejecutada sobre HDFS, permite almacenar datos y acceder a ellos rápidamente de modo que puedan ser utilizados por Metron para enriquecer las métricas, buscar amenazas o comparar datos.

- **HDFS:** es el sistema de ficheros distribuido de Hadoop y uno de los dos métodos de almacenamiento para los datos analizados.

- **ElasticSearch:** es el segundo método de almacenamiento, por indexado, lo que permite consultar la información a través de la interfaz Kibana de manera rápida y sencilla.

3.2. Procesamiento de datos con Metron

El procesamiento de datos de Apache Metron puede se divide en tres fases:

- Parseado [9]
- Enriquecimiento [7]
- Indexado [8]

Ya que estas fases no tienen dependencias entre ellas, podemos analizarlas de manera independiente. Cada fase funciona en el conjunto como una topología de Storm, que recibe, procesa y redirige los datos por una salida.

3.2.1. Captura de datos

Apache Metron no realiza ningún tipo de captura de datos, si no que depende de otras herramientas para ello. Estas herramientas deben enviar los datos a Metron para que los pueda procesar y analizar. Para ello existen dos herramientas de control de flujo, Ni-Fi y Kafka. En el caso de utilizar Kafka, la herramienta que capture los datos debe de ser capaz de crear un flujo de datos por el cual enviar la información obtenida. Un *sensor* de Apache Metron lee este flujo de datos para su analizar las capturas. Ni-Fi se utiliza para poder conectar con Apache Metron los servicios y herramientas que no pueden escribir en un flujo de datos de Kafka directamente. El funcionamiento es sencillo, en la interfaz web de Ni-Fi se crea un consumidor que escucha en un puerto o que lee de un fichero. Cuando llega información nueva, esta es redirigida por un topic de Kafka hacia Apache Metron.

La lectura y procesamiento de las diferentes capturas se definen en los sensores. En ellos se indica el flujo que se escucha, así como las configuraciones de parseado, enriquecimiento e indexado que se van a aplicar a la información capturada. El sensor es, por lo tanto, la entidad encargada de centralizar las configuraciones y manejar el comportamiento de las topologías que procesan y analizan cada mensaje de una fuente de datos concreta.

3.2.2. Parseado

El parseado es el primer paso en el procesamiento de datos realizado por Apache Metron. La finalidad de este módulo es recibir los datos y convertirlos a un formato que Apache Metron pueda interpretar, una estructura JSON estándar. En esta fase también se realizan transformaciones sencillas de los campos resultantes con el fin de facilitar el procesamiento en fases posteriores.

El parseador depende del tipo de registros y métricas que se van a procesar, por lo tanto no se puede usar una única topología de propósito general. En su lugar se crean diferentes topologías, una para cada sensor, de modo que las diferentes fuentes de datos sean redirigidas a su topología correspondiente.

Estas topologías pueden hacer uso de dos tipos de parseadores, escritos en Java o de propósito general. Los parseadores Java son más complejos, pues están optimizados para topologías que requieren baja latencia, maximizando velocidad y rendimiento. Esta complejidad dificulta los cambios, teniendo que recompilar la topología entera al realizar modificaciones. Por la otra parte, los parseadores de propósito general están diseñados para topologías más lentas, por lo que son más sencillos y permiten una solución rápida a la hora de crear nuevos sensores. Actualmente existen parseadores Grok, que usan expresiones regulares para identificar los diferentes campos de los mensajes, parseadores CSV, que realizan la transformación del formato CSV a JSON, y mapeador JSON, que únicamente redirige el mensaje.

Una vez convertido el mensaje original a formato JSON, se pueden realizar pequeñas transformaciones sobre el mismo. Algunas de estas transformaciones pueden ser añadir o eliminar campos, renombrar campos y modificar o

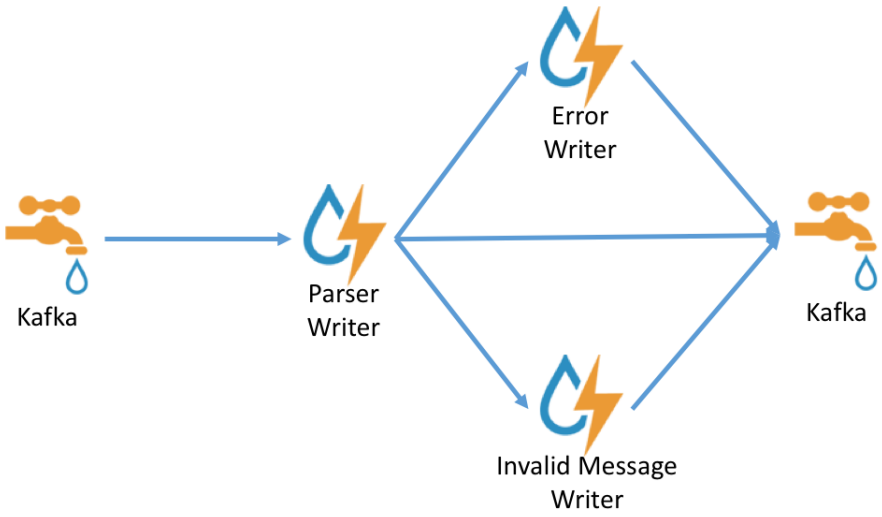

Figura 3.1: Arquitectura de la topología de parseado

añadir contenido.

Si durante el parseado o la transformación surgen errores, o se detecta que el mensaje recibido es inválido, se genera un nuevo mensaje especificando las razones, que se envía directamente a la topología de indexado para que se guarde como un error.

En la imagen 3.1 se puede ver de manera gráfica el funcionamiento de la topología de parseado. El parseador lee un flujo de datos de Kafka, que contiene los mensajes generados por una fuente de datos. Los mensajes se procesan y transforman y el resultado se escribe en un nuevo flujo de Kafka, enviando los datos a la topología de enriquecimiento. En el caso de que surja algún error durante el parseado o la transformación se envía el mensaje a un nuevo proceso (*bolt* en terminología de Storm), dependiendo del tipo de error. Este bolt crea un nuevo mensaje, indicando la razón del error, y lo escribe en un flujo de Kafka dirigido a la topología de indexado.

Las configuraciones de las diferentes topologías de parseado se definen en un documento JSON almacenado en Zookeeper. En este documento se

especifica, entre otras cosas, el parseador que se va a utilizar, que flujo debe escuchar, las transformaciones a realizar o las razones para invalidar un mensaje. Para poder utilizar este parseador y analizar los datos es necesario crear un sensor en el que se introduce la configuración pertinente.

3.2.3. Enriquecimiento

Cuando el mensaje ha sido transformado a un formato JSON que Apache Metron puede interpretar, y los campos han sido convenientemente transformados según las necesidades del usuario, el mensaje pasa a la fase de enriquecimiento. En este módulo, el mensaje se enriquece con datos externos a partir de los ya existentes en sus campos, y, en el caso de existir, se marcan las amenazas detectadas y se les asigna un valor de triaje.

En la figura 3.2 se puede observar la complejidad de esta topología. El funcionamiento de entrada y salida es el mismo que en la topología de parseado, escuchando un flujo de Kafka y reenviando a otro una vez terminado el proceso. Siendo estos los flujos de enriquecimiento e indexado.

Cuando llega un nuevo mensaje, los campos que se van a utilizar para el enriquecimiento se envían a los diferentes *bolts*, dependiendo del tipo de enriquecimiento que se vaya a realizar. Cada campo se puede enviar a uno o varios *bolts*. Una vez enriquecido, el mensaje se ensambla con todos los nuevos datos y modificaciones. Este mensaje se reparte nuevamente entre los *bolts* de búsqueda de amenaza según el análisis que se quiera realizar. Una vez procesado se vuelve a ensamblar el mensaje. En este último paso es donde se asigna al mensaje su puntuación de riesgo. Finalmente el mensaje se envía a un nuevo flujo de Kafka, pasando a la fase de indexado.

En la figura 3.2, se pueden observar tres tipos de enriquecimiento. El enriquecimiento de HBase, que utiliza esta base de datos para añadir datos nuevos en base a los campos del mensaje. El enriquecimiento de *Stellar*, que permite al usuario crear su propias condiciones, utilizando este lenguaje de dominio específico creado por Apache Metron. Y el enriquecimiento por geolocalización, que hace uso de la base de datos de *GeoLite2* para añadir

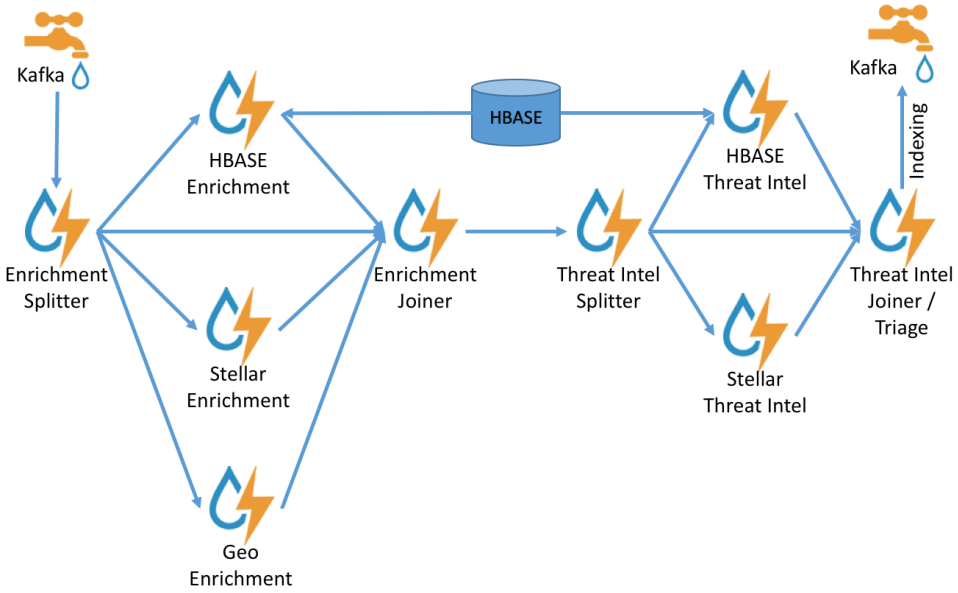

Figura 3.2: Arquitectura de la topología de enriquecimiento

localizaciones a las diferentes IPs del mensaje. En la fase de búsqueda de amenazas, se usa HBase y Stellar únicamente.

Aún con una única topología y una configuración global, es posible crear configuraciones adecuadas a los diferentes sensores. Esto permite crear diferentes enriquecimientos y amenazas para cada fuente de datos, aumentando las capacidades del análisis. Todas estas configuraciones se definen en un documento JSON almacenado en Zookeeper.

La configuración de los sensores se divide en dos partes, enriquecimiento y amenazas. Estas partes se dividen de nuevo, en el caso de enriquecimiento en tres campos:

- *fieldToTypeMap*: indica la clave a utilizar en HBase. Para realizar un enriquecimiento por HBase sencillo se debe indicar el mapeo entre los campos utilizados y el tipo de enriquecimiento a realizar.

- *fieldMap*: indica el enriquecimiento que se aplica junto a la lista de campos utilizados. En el caso de realizar enriquecimientos por Stellar

más complejos habría que indicar las operaciones a realizar.

- *config*: contiene la configuración general del enriquecimiento.

Para las amenazas se utilizan los mismos campos y se añade uno nuevo:

- *triageConfig*: en el que se incluye la configuración del indicador de triaje. Si se detecta una amenaza se asigna al mensaje una puntuación de riesgo basada en esta configuración.

Una vez enriquecido y analizado el mensaje, este es enviado a la siguiente fase, donde es indexado para su consulta.

3.2.4. Indexado

El indexado es el último paso del procesamiento. En él, los datos ya analizados se introducen en HDFS y ElasticSearch, permitiendo su visualización y consulta.

Al igual que el enriquecimiento, el indexado recibe datos de un único flujo de Kafka, pero este se divide en dos topologías, una encargada de escribir en HDFS, y otra encargada de hacerlo en ElasticSearch. El funcionamiento de ambas es idéntico y se pueden estudiar como una única topología.

En la figura 3.3 se puede observar este funcionamiento. La topología recibe datos del flujo de indexado y escribe su contenido en los dos índices, terminando el proceso. En el caso de que ocurra algún error durante el indexado, se genera un nuevo mensaje que se envía al flujo de indexado para su almacenamiento.

La escritura de estos índices se realiza en forma de lotes, que se especifican en las configuraciones de cada sensor. Esta configuración está definida en un documento JSON almacenado en Zookeeper, y en ella se puede modificar el comportamiento de cada escritor.

Cuando los datos han sido almacenados, estos se pueden consultar y visualizar a través de la interfaz de Kibana.

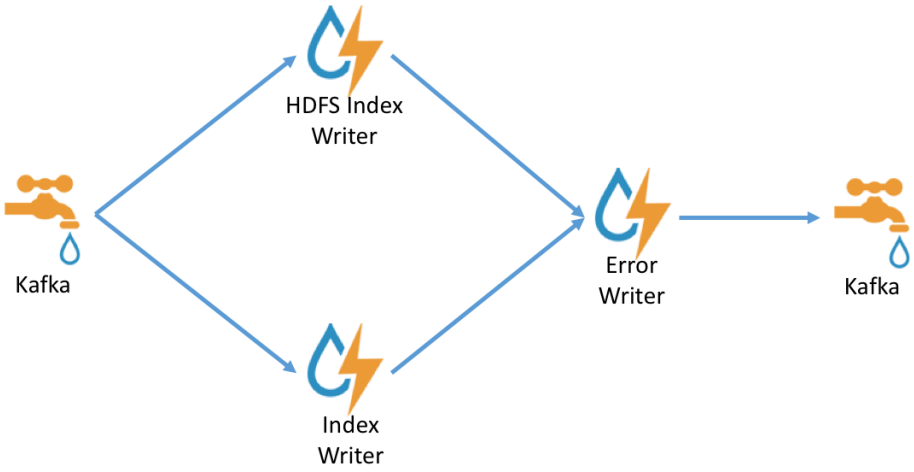

Figura 3.3: Arquitectura de la topología de indexado

Capítulo 4

Despliegue del entorno de pruebas

Una vez escogida la herramienta, se ha procedido a realizar un despliegue completo en un entorno de pruebas lo más parecido al entorno real. Para ello se utilizan máquinas virtuales y se han instalado los servicios y herramientas usados en el entorno final, con las que Apache Metron deberá integrarse.

4.1. Entorno virtual

Antes de crear los nodos virtuales del entorno de pruebas, se estudian los requerimientos mínimos de las diferentes herramientas y aplicaciones, de forma que se minimicen los problemas en el despliegue. Estos requerimientos no son fijos, si no que dependen del entorno y la cantidad de datos que se vayan a manejar. Consultada la documentación oficial y diferentes guías de instalación, también se descubre que existen múltiples vías de instalación de esta herramienta, siendo las más comunes: utilizando Ambari, en una máquina virtual única, en Amazon Web Services (AWS), o mediante instalación manual.

Una vez estudiadas las diferentes opciones se decidió usar la instalación a través de Ambari, una herramienta de código abierto que permite instalar,

Numero de Nodos	**Memoria Disponible**	**Almacenamiento**
1	1024 MB	10 GB
10	1024 MB	20 GB
50	2048 MB	50 GB
100	4096 MB	100 GB
300	4096 MB	100 GB
500	8096 MB	200 GB
1000	12288 MB	200 GB
2000	16384 MB	500 GB

Cuadro 4.1: Requerimientos del host de Ambari Metrics Collector según el número de nodos

manejar y monitorizar un clúster de Hadoop de manera simple y gráfica. De este modo se puede desplegar tanto Apache Metron como su entorno de manera sencilla. Por otra parte, el CESGA cuenta con suficiente poder computacional para albergar Apache Metron en sus propias máquinas, sin necesidad de tener que externalizar servicios, por lo que la opción de usar AWS quedó descartada. También se descartó la opción de instalación en una única máquina virtual ya que serviría únicamente como primer contacto y no se aproximaría a la solución real.

Escogida la opción de Ambari, es necesario tener también en cuenta los requisitos de esta herramienta, especialmente de su recolector de métricas (*Ambari Metrics Collector*), que es el que permite conocer el estado de los nodos, así como el de los diferentes servicios instalados en cada uno de ellos. La tabla 4.1 muestra la cantidad de memoria y almacenamiento necesario por el host de *Ambari Metrics Collector*, dependiendo del número de nodos del clúster. En el caso del *Ambari Server* se recomienda que el servicio disponga de al menos 1GB de RAM y unos 500MB de espacio en disco.

Al buscar los requerimientos de memoria y almacenamiento de Metron se comprobó que no existen unos requerimientos fijos, si no simplemente orien-

Nodo	Núcleos	Memoria	Almacenamiento
Nodo 1	4	12 GB	50 GB
Nodo 2	4	8 GB	20 GB
Nodo 3	4	8 GB	20 GB

Cuadro 4.2: Clúster inicial de Apache Metron

tativos. Debido a que los nodos virtuales que hemos utilizado para las pruebas permiten añadir fácilmente tanto almacenamiento, como memoria disponible se decidió crear un clúster de 3 nodos con las características mostradas en la tabla 4.2.

En todos los nodos se instaló CentOS 7 ya que es uno de los pocos sistemas operativos soportado por Apache Metron, además de ser ampliamente utilizado en las máquinas del CESGA.

4.2. Proceso de instalación

Una vez creados los nodos se inicia la instalación de Apache Metron. Este proceso incluye tanto la preparación de las máquinas como el despliegue de la herramienta a través de Ambari. Esta instalación se explica más detalladamente en el apéndice A, mientras, en este apartado, se hace un pequeño resumen de los pasos más relevantes y problemáticos, con el fin de facilitar el proceso en instalaciones futuras y poder entender el funcionamiento de la herramienta.

Los primeros problemas que se encontraron a la hora del despliegue fueron tanto la falta de documentación actualizada como las inconsistencias entre la información de las diferentes fuentes. Finalmente el despliegue se llevó a cabo usando la información aportada por la Wiki de Apache Metron, así como su página de Github y la disponible en la Web de Hortonworks. El proceso se documentó convenientemente para los técnicos del CESGA, y se puede encontrar en el apéndice A de esta memoria.

4.2.1. Prerrequisitos

El primer paso del despliegue es la comprobación e instalación de los prerrequisitos y dependencias que deben cumplir todos los nodos. Algunos de estos prerrequisitos son:

- Crear un *Repositorio EPEL*
- Establecer conexión SSH sin contraseña entre nodos
- Ajustar el número máximo de ficheros abiertos y procesos
- Desactivar IPv6
- Desactivar *Transparent HugePage*
- Desactivar *SELinux*
- Instalar dependencias de Ambari, Metron, BBDD, etc...

Una vez preparados todos los nodos se debe acondicionar el nodo principal para albergar el servidor de Ambari y poder llevar a cabo la compilación de Apache Metron.

4.2.2. Compilación

Debido a que Metron está en constante desarrollo y es altamente modular no existe un paquete precompilado, si no que es necesario construir los diferentes paquetes a partir del código disponible en su página oficial de GitHub.

Para compilar estos paquetes debemos instalar el siguiente software:

- Maven
- Docker
- npm
- gcc-c++ >4.8

Esta compilación puede dar lugar a errores a la hora de construir algunos módulos con npm, concretamente los relacionados con la interfaz gráfica, por ello se recomienda utilizar un contenedor de Docker con todas las dependencias instaladas como se detalla en el apéndice A.

Una vez obtenidos los paquetes debemos añadir los diferentes *RPMs* al repositorio local de cada nodo, además de añadir los MPack a Ambari. Estos MPack son unos paquetes especiales que permiten añadir funcionalidades a Ambari sin necesidad de actualizar la herramienta. De esta manera se pueden instalar servicios no soportados por defecto, como Metron o ElasticSearch y Kibana.

4.2.3. Ambari

El último paso antes del despliegue del clúster es la instalación y configuración del servidor de Ambari. Primero se activa la sincronización de los tres nodos mediante NTP y se desactivan los cortafuegos para evitar problemas de comunicación. Al ser un entorno controlado este último paso no conlleva ningún riesgo.

Además, debemos indicar los nombres de cada máquina a todos los nodos para que puedan resolver los diferentes *hostnames* y comunicarse entre sí con mayor facilidad.

Una vez preparados los nodos, instalamos el servidor de Ambari en el nodo principal y añadimos el MPack de Metron anteriormente compilado. Ya que vamos a usar ElasticSearch y Kibana, añadimos el MPack de estos servicios, que también se crea al compilar Apache Metron.

Finalmente, iniciamos el servicio y ya podemos acceder a la interfaz web localizada en el puerto 8080 del nodo principal. Desde aquí crearemos el clúster y seleccionaremos los servicios, indicando los nodos donde los queremos instalar, así como los maestros, esclavos y clientes de cada uno.

Como se había comentado anteriormente, esta instalación generará un entorno independiente, por lo que al finalizar la instalación tendremos un clúster de Hadoop (HDP 2.5) con los siguientes servicios:

- HDFS
- HBase
- Zookeeper
- Storm
- Kafka

Además, también estarán instalados otros servicios requeridos por Apache Metron, como ElasticSearch, Kibana, Mapreduce o Hive.

4.2.4. Configuración inicial

El último paso antes de finalizar la instalación del clúster de Hadoop es la configuración de los diferentes componentes para que exista comunicación entre todos los nodos y Apache Metron pueda realizar el análisis de datos.

Primero, debe configurarse de manera manual la base de datos de Metron REST en el nodo en el que se haya instalado este módulo. Esta base de datos permite utilizar la interfaz web de Metron y manipular desde ella los diferentes componentes de la herramienta. Este paso se detalla en el apéndice A.

El resto de la configuración se realiza desde la propia interfaz de Ambari, ya que mantiene una sincronización entre los diferentes nodos. Se han sustituido los valores concretos, incluidos en el apéndice A, por una pequeña explicación de la importancia de cada campo.

Las acciones que deben llevarse a cabo en cada servicio son:

- HDFS: aumentar el tamaño de la pila de Java para poder manejar la gran cantidad de ficheros generados en el análisis.
- ElasticSearch: indicar el nombre del nodo maestro y la red en la que se encuentra.

- Kibana: indicar la dirección del nodo maestro de ElasticSearch para que sepa de donde obtener los datos a visualizar.
- Metron: indicar la dirección del nodo maestro de ElasticSearch para poder indexar los datos analizados.
- Metron-REST: añadir los datos de la base de datos creada anteriormente.

Ambari también nos pedirá introducir una nueva contraseña en diferentes servicios. Inicialmente no es necesario configurar servicios como Storm, Kafka o Zookeeper, aunque sí que será recomendable a posteriori.

Una vez completados estos pasos tendremos instalado un clúster operativo de Hadoop con Apache Metron. El siguiente paso será configurar y administrar la herramienta Apache Metron, así como los diferentes servicios del clúster para una mayor eficiencia del mismo.

Capítulo 5

Implementación de la solución

Una vez estudiado el funcionamiento de la herramienta, y desplegado el entorno de pruebas y todas sus dependencias, se procede a la configuración y administración de Apache Metron, realizando las primeras aproximaciones con datos controlados para conocer las capacidades de la herramienta de manera práctica y comenzar el análisis de datos. En este capítulo se explica como se han creado los diferentes sensores, incluyendo las configuraciones de parseado, enriquecimiento e indexado. También se comentan los problemas encontrados durante la implementación de soluciones más complejas y los cambios en las configuraciones de las herramientas del entorno para mejorar el rendimiento del procesamiento. En el apéndice B se detallan las configuraciones y los pasos concretos a seguir a la hora de crear estos sensores y añadir nuevas métricas.

5.1. Primera aproximación

El primer paso a la hora de utilizar y comprender el procesamiento de Apache Metron es introducir fuentes de datos para su análisis. Para ello se instalan diferentes herramientas de recolección de métricas en el entorno de pruebas. Así como la herramienta Ni-Fi [28] para poder administrar algunas de ellas.

Para facilitar las pruebas, se decide instalar todas las herramientas que se vayan a utilizar en un único nodo. De esta manera solo se debe gestionar una máquina. Debido a que la instalación del clúster de Hadoop ocupa casi en su totalidad el almacenamiento disponible en los diferentes nodos, se crea un nuevo disco de 50GB en el que se almacenan las herramientas instaladas.

5.1.1. Zeek

Como primer contacto se utiliza el monitor de red Zeek [38], anteriormente conocido como Bro [43]. Apache Metron recomienda esta herramienta, que viene soportada por defecto, ya que incluye un parseador específico para los mensajes de Zeek. También existe un plugin, *metron-bro-plugin-kafka* [23], que permite a Zeek escribir en un topic de Kafka y comunicarse con Apache Metron fácilmente, sin la necesidad de utilizar Ni-Fi.

Zeek es una herramienta de código abierto y análisis de red, desarrollada en 1994. Su nombre original era una referencia al *Big Brother* de la famosa obra *1984* de George Orwell. Esta herramienta captura y envía paquetes IP a un analizador de eventos, que los acepta, realizando sobre ellos un análisis a mayor profundidad, o los desecha. Estos analizadores son scripts que el usuario puede modificar a su conveniencia. En este caso, se utiliza la herramienta para capturar paquetes de peticiones DNS y HTTP, sin realizar ningún tipo de análisis previo.

La instalación y configuración inicial de la herramienta y el plugin se realizan siguiendo las instrucciones detalladas en el apartado B.2.1 del apéndice B.

Una vez instalada y configurada la herramienta se configura el sensor de Apache Metron, encargado de procesar las capturas. No es necesario crear este sensor, pues ya que existe por defecto, pero sigue siendo necesario configurarlo para la finalidad del centro. Esta configuración se puede realizar desde la propia interfaz web de Apache Metron. En esta configuración se indica el flujo de datos a escuchar, el parseador que se va a utilizar, incluido por defecto en Metron para esta herramienta, las transformaciones, y el tipo

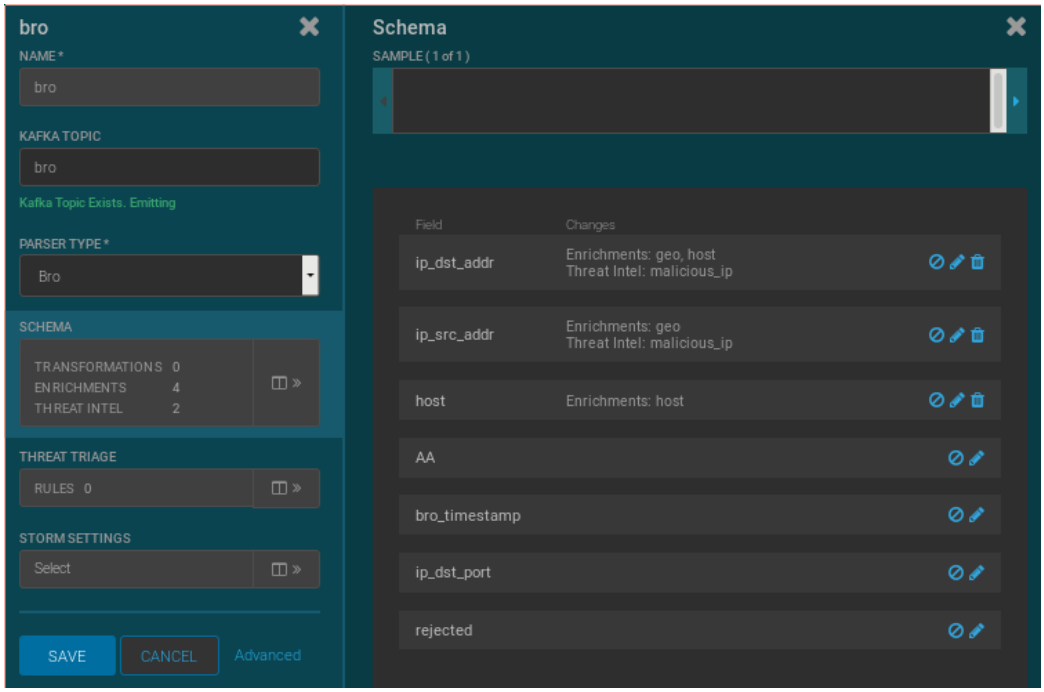

Figura 5.1: Configuración de Zeek en la interfaz web de Metron

de enriquecimiento que se va a aplicar. Para esta fuente de datos únicamente se realiza un enriquecimiento de geolocalización para conocer la posición del servidor DNS consultado. Los diferentes campos de la configuración se pueden observar en la figura 5.1.

Al iniciar el sensor desde la interfaz web el análisis comienza. Se puede comprobar que Metron está procesando los datos consultando las diferentes topologías desde la interfaz web de Storm. También se pueden consultar desde consola los flujos de enriquecimiento e indexado para confirmar que el flujo de datos es correcto. Finalmente, también se comprueba que los datos están siendo indexados en ElasticSearch haciendo una petición GET al servidor:

```
$ curl -XGET 'http://node2:9200/_cat/indices/*'
```

5.1.2. Squid

La siguiente fuente de datos que se va a utilizar es *Squid*, un servidor proxy para web con caché, que guarda las conexiones web realizadas por el cliente. Esta herramienta guarda la información obtenida en logs, y no permite publicar en un flujo de datos de Kafka los registros, por lo que se usa Ni-Fi para leer los ficheros y reenviar los datos capturados por Squid a Apache Metron. Tanto la instalación de Squid como la de Ni-Fi se detallan en el apéndice B.

Una vez se han instalado las dos herramientas se debe configurar Ni-Fi para que lea el registro de Squid y lo envíe a Apache Metron mediante Kafka. Esto se hace desde la interfaz web de la herramienta, que permite crear consumidores y productores de ficheros y flujos de datos de manera gráfica y sencilla. Para el caso de uso actual, se crea un procesador de tipo *TailFile*, encargado de leer el registro de Squid. La ruta al fichero se indica en el campo *File to Tail* de la configuración, como se observa en la figura 5.2. Una vez creado el consumidor, se debe crear un productor. Para ello, se crea un procesador de tipo *PublishKafka*, que se configura indicando el topic a usar y la lista de servidores Kafka a los que enviar los mensajes, como se muestra en la figura 5.3. También se puede modificar el nombre de los procesadores para mejorar su legibilidad.

Una vez creados ambos procesadores, se conectan arrastrando la flecha desde el lector de registros hasta el productor de Kafka. Una vez conectados, como en la figura 5.4, es necesario generar datos para que Ni-Fi envíe algo a Apache Metron. Para crear una fuente estable y continua de peticiones web se crea un pequeño fichero formado por una lista de direcciones web. Un sencillo script, ejecutado en segundo plano, se encarga de hacer peticiones web, mediante el cliente de Squid, a las diferentes direcciones del fichero, de manera aleatoria, y en cortos intervalos de tiempo. Una vez en funcionamiento se comprueba que Ni-Fi esté emitiendo datos por este flujo de datos mediante un consumidor de Kafka.

A diferencia de Zeek, Metron no trae por defecto un parseador para los

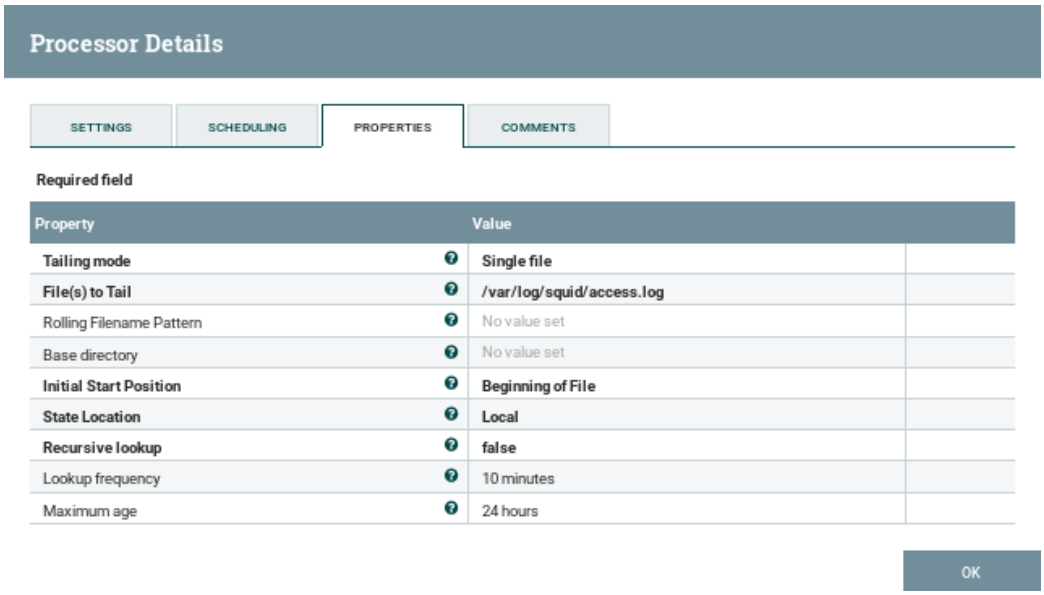

Figura 5.2: Configuración del procesador *TailFile* de Squid

registros de Squid, por lo que se debe de crear uno con el que poder convertir y analizar la información aportada por esta herramienta. Como se ha visto en el capítulo 3, existen dos tipos de parseadores, los escritos en Java y los de propósito general. Dado que, el caso de estudio es una primera aproximación, se espera un volumen de carga bajo, y los mensajes de Squid están bien estructurados, se ha decidido utilizar un parseador Grok. Este parseador se crea a partir de una expresión del filtro Grok, capaz de detectar y dividir los diferentes campos en los mensajes de Squid. Para crear esta expresión se utiliza uno de los múltiples depuradores de Grok, como el *Grok debugger* [18], disponibles en Internet. Este depurador permite validar la expresión con un mensaje real, que se copia del registro de Squid, para comprobar si funciona, antes de ser añadida a Metron. Una vez creada esta expresión, se guarda en un fichero y se mueve a HDFS, para que Metron tenga constancia de este nuevo parseador.

Ahora, se debe crear el sensor y las diferentes configuraciones de parseado, enriquecimiento e indexado. Estos pasos, así como las configuraciones completas, se detallan en el apartado B.3.2 del apéndice B

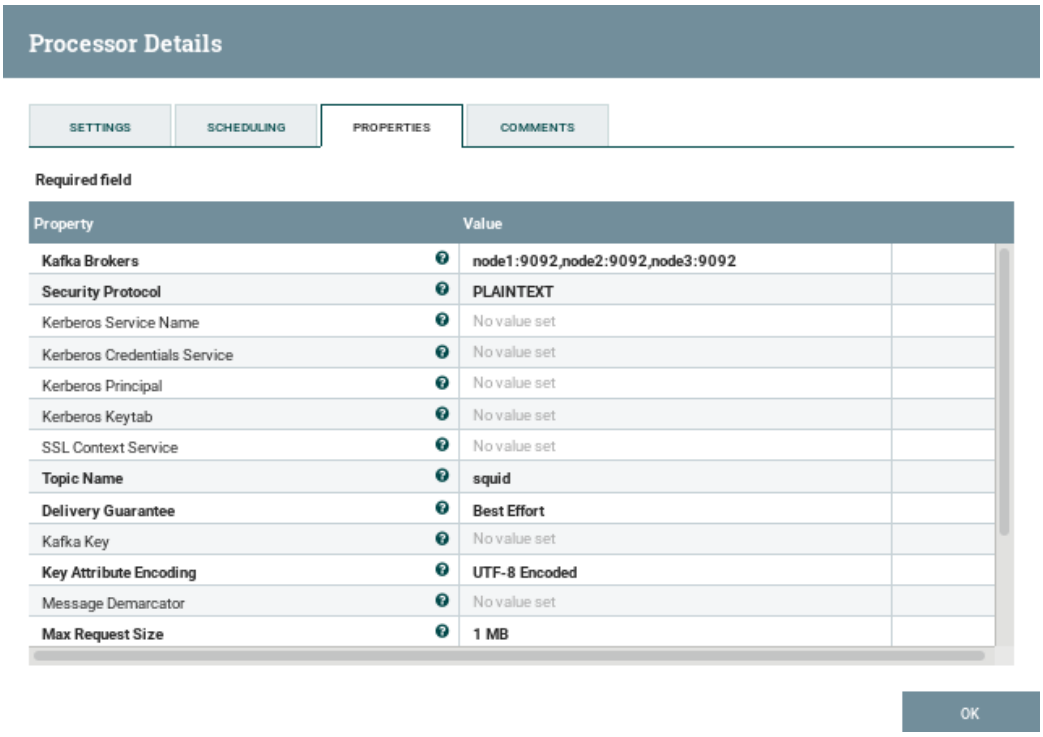

Figura 5.3: Configuración del procesador *PublishKafka*

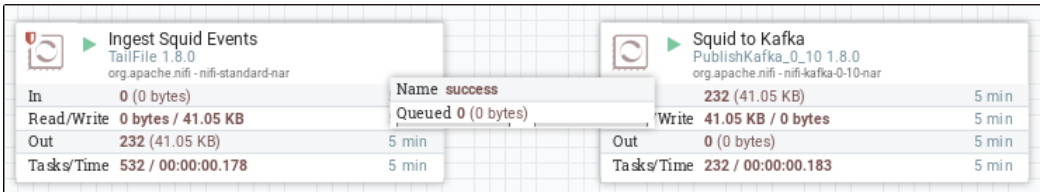

Figura 5.4: Procesos de Ni-Fi conectados

La primera configuración que se define es la de la topología de parseado, que incluye los datos del sensor. Para ello se crea un fichero JSON en el que se establecen el tipo de parseador que se va a utilizar, el nombre del flujo del que se obtienen los datos y la ruta a la expresión Grok. A mayores, también se puede añadir la configuración de transformación. En este caso, como Squid guarda la URL completa de la petición y únicamente interesa el dominio, se convierte el campo URL mediante una expresión Stellar, de modo que queda reducido al dominio web.

Todas las configuraciones de Metron se almacenan en Zookeeper, por lo que al completar la configuración, se debe cargar en el coordinador.

Desde la interfaz web de Apache Metron se puede comprobar que el sensor ha sido creado. Al iniciarse, se confirma que se ha creado una nueva topología en Storm, consultándolo a través de su interfaz web. Para comprobar que los mensajes están siendo parseados correctamente, se consulta el flujo de enriquecimiento, observando que incluya mensajes de Squid.

Ahora se debe establecer la configuración del enriquecedor. El proceso es similar al del parseador: se indica el tipo de enriquecimiento que se va a aplicar en un fichero JSON específico. En este caso, únicamente se añade enriquecimiento de geolocalización, con el fin de conocer las coordenadas de los servidores a los que se realizan las peticiones. Una vez creada la configuración, la cargamos en Zookeeper y comprobamos que funciona consultando el flujo de indexado.

El indexado, como se ha explicado en el capítulo 3, es la última fase del procesamiento de datos. Para su configuración, se crea, nuevamente, un fichero JSON, en el que se indica como se deben indexar los datos tanto en HDFS como en ElasticSearch. Finalizada la configuración, se actualiza el contenido de Zookeeper y se comprueba en ElasticSearch que los datos están siendo indexados correctamente.

5.1.3. Visualización de Datos

Una vez introducidos los datos en ElasticSearch, se pueden visualizar a través de la interfaz web de Kibana. Primero, se definen en Kibana los patrones que se van a utilizar para obtener los datos de los diferentes índices. Esto se puede hacer desde la propia interfaz web, y, en este caso, se crea un patrón general para todos los índices, y uno para cada tipo de índice, Zeek y Squid. Una vez creados los patrones, se pueden consultar los datos guardados desde la página principal de Kibana.

Al mismo tiempo, es necesario crear unas plantillas para los diferentes índices de ElasticSearch, en el que se indican los diferentes campos y el tipo

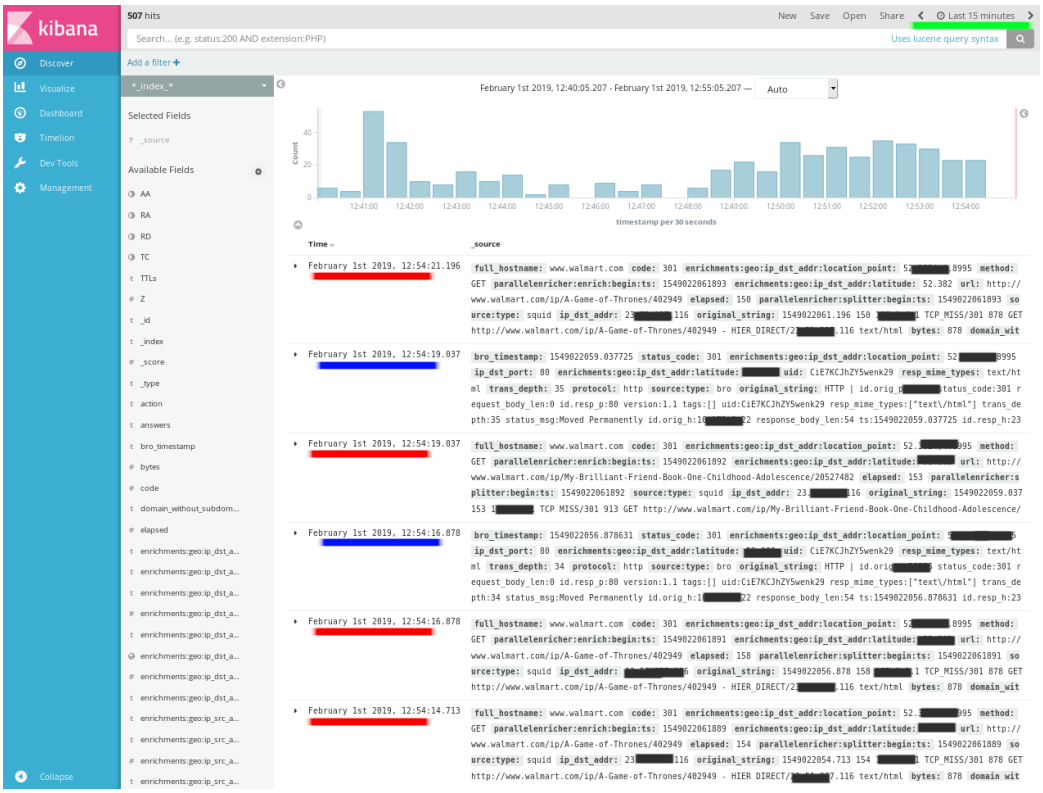

Figura 5.5: Ejemplo de datos almacenados en ElasticSearch

de datos que almacenan, de modo que Kibana pueda manejar dichos datos correctamente. Existen plantillas predeterminadas, pero se recomienda adaptarlas a los campos de cada sensor, especialmente si se han añadido nuevos datos con el enriquecimiento o las transformaciones.

En la figura 5.5 se pueden observar los datos de Squid, subrayados en rojo, y los datos de Zeek (almacenados como Bro), en azul. Encima de estos datos, Kibana presenta un diagrama de barras indicando la cantidad de datos almacenados en diferentes instantes de tiempo. Estos periodos de tiempo se pueden cambiar en el campo subrayado en verde.

Kibana también permite crear *visualizadores*, que permiten obtener información gráfica de los datos almacenados, como contar la cantidad de mensajes que cumplen una condición, mapas con la posición de las conexiones, gráficas circulares, diagramas de barras, etc... Estos visualizadores pueden

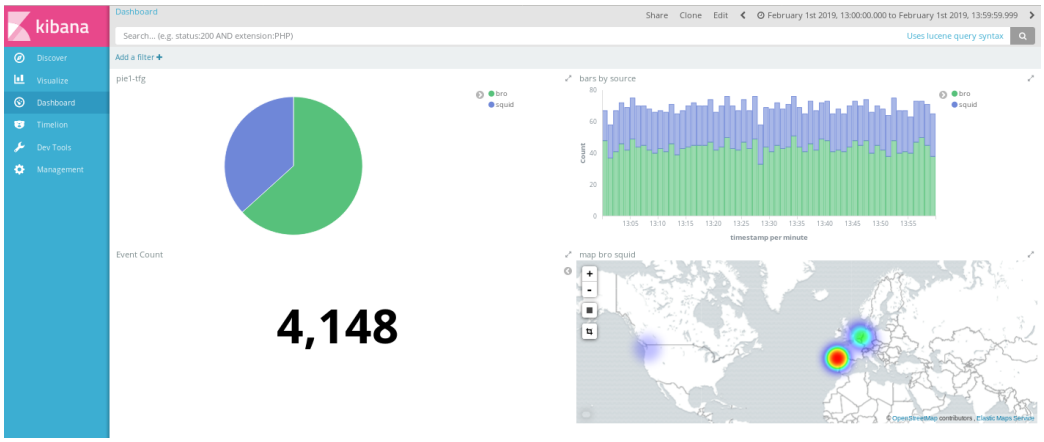

Figura 5.6: Panel de Kibana con datos de Zeek y Squid

ser combinados para crear un panel de administración que facilite el análisis visual de la información, como se puede observar en la figura 5.6. En este panel se muestra la relación de datos de Squid y Zeek en una gráfica circular, la cantidad de datos indexados según el tipo de fuente en un diagrama de barras, el número total de capturas, y un mapa de calor con las posiciones de los servidores a los que se han hecho las peticiones. Existe la posibilidad de crear visualizaciones más complejas, pero al ser esta una primera aproximación no se ha dedicado mucho tiempo a este apartado.

5.1.4. Problemas de almacenamiento

Tras unos días de funcionamiento correcto, algunos servicios del clúster comienzan a fallar, entre ellos Kafka y HDFS, que a su vez producen errores en los servicios que dependen de ellos. La causa principal resulta ser la falta de espacio de almacenamiento. Apache Kafka almacena los mensajes que recibe, tanto de las fuentes de datos como de los procesos de Storm, encargados de procesarlos y escribirlos en nuevos flujos. Estos mensajes se almacenan durante un tiempo configurable, por defecto 1 semana, por lo que, si la cantidad de datos que llegan a través del flujo de entrada es grande, el tamaño del registro de Kafka puede crecer notablemente, llegando a 20GB

sin dar tiempo al administrador a borrar los más antiguos. Por otra parte, el indexado en HDFS dejaba poco espacio para las otras aplicaciones que también necesitaban este almacenamiento.

La solución a este problema consistió en aumentar el almacenamiento en todos los nodos, ofreciendo un mayor espacio a todos los servicios y dando una mayor ventana de tiempo al administrador para liberar ese espacio en caso de que algún servicio se exceda.

5.1.5. Añadiendo fuentes de datos externas

Una vez se han realizado las primeras pruebas, se avanza hacia la configuración del entorno real, integrando datos de fuentes externas al entorno de pruebas con el fin de comprobar la respuesta a un mayor volumen de datos, y al funcionamiento con datos reales.

Añadiendo registros del sistema

Apache Metron tiene un parseador de Syslog incorporado por defecto, que se ha estudiado y configurado apropiadamente para tratar los datos reales. Todas las configuraciones de este apartado están incluidas y detalladas en el apéndice B.

En la primera aproximación se utilizan los registros del sistema de los tres nodos pertenecientes al clúster. Esto permite tener control total sobre los mensajes que son enviados y así poder realizar pruebas en un entorno controlado.

La herramienta utilizada localmente para manejar los registros de cada máquina es *rsyslog*, que permite enviar estos registros a un servidor a través de la red. Existe la posibilidad de usar un plugin, *omkafka* [29], para enviar los registros a través de un flujo de Kafka. Este plugin únicamente funciona a partir de la versión 8.7.0 de rsyslog, por lo que se decide buscar una manera de leer los datos con Ni-Fi. Como se ha visto en el apartado 5.1.2, Ni-Fi permite leer un fichero y reenviar su contenido a través de un flujo de Kafka. Por lo tanto, como primera solución, se podría configurar el nodo donde

está instalado Ni-Fi como servidor de *rsyslog*, y leer los mensajes del fichero de registros. Sin embargo, Ni-Fi también tiene un *procesador* diseñado para funcionar como un servidor de *rsyslog*, recibiendo los registros directamente sin necesidad de leer ningún fichero. En este procesador se indica el puerto y protocolo en el que Ni-Fi escucha. En los diferentes nodos se configura el reenvío de todos los registros capturados al puerto y dirección del procesador de Ni-Fi. Para enviar a Apache Metron simplemente se crea un procesador que publique en un nuevo flujo de datos de Kafka.

Una vez creado el flujo de datos de Kafka es necesario crear el sensor y añadirle el parseador correspondiente. Inicialmente, este parseador no dividía los mensajes en los campos pertinentes. Esto se debía a que el parseador únicamente funciona con el formato de Syslog RFC 5424 [32], por lo que es necesario editar la configuración de los diferentes clientes para que envíen en este formato. Para ello se indica la plantilla que se quiere utilizar para enviar el mensaje junto al destino de los mensajes. Al estar estandarizado, en *rsyslog* existe una plantilla para el formato RFC 5424 y no es necesario crearla.

Una vez modificada la configuración del cliente y reiniciado el servicio, los mensajes son aceptados y procesados por Apache Metron. Al ser esta fase un primer contacto no se añadió ningún tipo de transformación o enriquecimiento a los mensajes, si no que se estudió la forma final de los mismos desde Kibana. Se comprobó que el parseador identifica los diferentes campos del registro, dejando como un campo único la parte que incluye el mensaje. Esto no permite, por lo tanto, extraer datos de ese mensaje de manera sencilla, y es necesario crear algún tipo de parseador para dividir la información de los registros.

Finalmente, se introducen datos de registros de sistemas reales. Para ello, se configura una de las máquinas del CESGA para que envíe datos al proceso de Ni-Fi y poder analizarlos.

Capturas de red

Una vez introducidos los datos de los registros se procede a añadir los datos de red. Para ello se utilizan diferentes conmutadores, que envían capturas a una de las máquinas del clúster a través del protocolo *sFlow* [33]. En el clúster, esta información es recogida por pmacct [31] y reenviada a través de un nuevo flujo de datos de Kafka para que sea analizada por Apache Metron. Los datos de las capturas ya están formateados en JSON, por lo que no es necesario crear un nuevo parseador, si no que se utiliza el mapeador JSON incluido en Apache Metron.

Al comprobar que el nombre de algunos campos, como el de puerto e IP, destino y origen, era diferente al utilizado por el resto de sensores, se decide realizar una transformación en el nombre para así mantener una nomenclatura homogénea entre todos los sensores. Los detalles de esta transformación se pueden ver en el apéndice B, junto al resto de configuraciones de este sensor.

En vista del gran volumen de datos que se van a procesar, para poder estudiar el comportamiento de la solución, se añade únicamente el enriquecimiento por geolocalización, para conocer la posición de las máquinas que tratan de conectarse al CESGA. Además, se cambia la configuración del indexado para que se almacenen los mensajes por lotes.

5.2. Solución de problemas

Una vez añadidos los sensores principales, y antes de pasar a una solución más avanzada, se ha probado exhaustivamente el funcionamiento de la primera aproximación. En esta sección se comentarán los principales problemas encontrados.

5.2.1. Problemas de rendimiento

A causa del gran volumen de datos manejado tras añadir los registros del *Syslog* y las capturas de red, el clúster no tarda en saturarse y los errores

empiezan a aparecer. Tras analizar el entorno y las diferentes herramientas, se comprueba que el problema principal radica en la baja velocidad de indexado, que aumenta la latencia de todo el proceso. Esta latencia se va acumulando a lo largo del tiempo y de las diferentes fases, creando una cola enorme, que Kafka debe controlar. Esto genera unos logs de gran tamaño que rápidamente ocupan todo el espacio de almacenamiento del sistema, haciendo fallar al resto de herramientas que necesitan espacio físico.

Se aumenta el tamaño de los lotes de indexado para evitar que Metron pierda demasiado tiempo intentando indexar un paquete pequeño. Este cambio produce una reducción en la latencia, pero no es suficiente para hacer frente al gran volumen de datos procesado. Al monitorizar los discos usados por el clúster se descubre que el disco en el que se almacena la información tiene una carga de trabajo superior al 80 %, incluso cuando se para la topología de indexado. Esto se debe a que dicho disco está provisto mediante un sistema de almacenamiento distribuido (*Gluster*) que proporciona servicios de almacenamiento adicionales y con un grado de utilización muy elevado. Como solución, se migran las imágenes de los diferentes discos usados por el clúster a una nueva cabina, con discos menos usados. La latencia del indexado se reduce considerablemente. Además, como solo un trabajador ejecutaba la topología de indexado, se decide añadir otro trabajador más a esta tarea y así no saturar a un único nodo.

Otro de los problemas detectados en el indexado era producido por Kafka y un desajuste en los índices del contador, que controlan las posiciones del último mensaje enviado y el último mensaje consumido. Al consultar estos índices se puede conocer el retardo del flujo de datos, permitiendo saber si un consumidor necesita mas recurso para poder procesar los datos recibidos. En este caso la posición del último mensaje consumido era muy superior a la del último mensaje enviado, produciendo un retardo negativo que no permitía al consumidor funcionar correctamente. Este desajuste se había originado al eliminar los registros de Kafka para liberar espacio, ya que el índice del último mensaje enviado por un flujo se reinicia si no encuentra ningún re-

gistro anterior. El índice del último mensaje consumido se mantiene, puesto que este dato es controlado por Zookeeper, el cual coordina los diferentes consumidores y permite repartir los mensajes. Para solucionar este problema se opta por borrar los registros y toda la información de los flujos de datos con retardo negativo y volver a crearlos, ya que no contenían datos importantes. En versiones de Kafka posteriores a la utilizada en este proyecto, se añade la posibilidad de manipular el índice del último mensaje consumido, permitiendo reiniciarlo.

5.2.2. Cambios en el clúster y sus servicios

Una vez resueltos los diferentes problemas, se decide aumentar la memoria disponible de las máquinas del clúster. Esta decisión es motivada por el hecho de que la mayoría de herramientas están escritas en Java y requieren una máquina virtual de java (JVM) propia para ser ejecutadas. Estas JVM tienen asignada una memoria máxima, por lo que están limitadas en la cantidad de recursos que pueden usar. Desde Ambari se puede editar el máximo disponible para cada JVM, por lo que, en vista del aumento de memoria disponible, se dobla el umbral en los servicios principales, como HDFS, Storm o ElasticSearch.

Además, como el parseador de cada sensor necesita una topología de Storm propia, y al tener que añadir más de un trabajador a alguna topología, los 6 trabajadores indicados en la configuración inicial no son suficientes. Al ser estos trabajadores los encargados de ejecutar las tareas no se pueden iniciar nuevas topologías. Para solucionar este problema, se aumenta el número de trabajadores de Storm a 4 por máquina. El aumento de trabajadores también permite a Storm cambiar la topología a trabajadores ociosos en nodos con menos carga, haciendo el proceso más eficiente.

5.3. Enriquecimiento avanzado y búsqueda de amenazas

Una vez solucionados todos los problemas conocidos y en vista del correcto funcionamiento de la herramienta, se decide incorporar el enriquecimiento avanzado, y, una vez en funcionamiento, comprobar las capacidades de la búsqueda de amenazas.

La escasa la documentación ofrecida tanto por Apache Metron como por Hortonworks en este punto nos fuerza a comenzar implementando un simple enriquecimiento sobre los datos procesados en el sensor de Squid. Como primera aproximación se contempla un enriquecimiento sencillo mediante el uso de expresiones *Stellar*, pero se descarta debido a su baja utilidad para analizar los datos de este sensor, usándolo únicamente como prueba de que también se pueden transformar campos en esta fase. Se decide, por lo tanto, utilizar el enriquecimiento por HBase. Este enriquecimiento se realizará sobre el campo de dominio, y agregará a la captura información acerca del propietario de dicho dominio. Esta información la añade el usuario a una tabla de HBase, la cual es consultada cada vez que se realiza un enriquecimiento en el sensor de Squid. En el caso de que la petición se haya realizado a un dominio guardado en la base de datos, el mensaje es enriquecido con la información disponible de ese dominio. Tanto la configuración como un ejemplo de la información introducida está disponible en el apéndice B.

Una vez en funcionamiento el enriquecimiento por HBase del sensor de Squid, se decide crear una nueva tabla, para enriquecer tanto las capturas de Zeek como de sFlow. En esta tabla se añade la lista de páginas más visitadas de Internet junto a su IP. En total 1 millón de entradas. Esta lista es obtenida del top-1m de Alexa [1], y un script añade la IP correspondiente a cada dominio mediante consultas DNS. Este enriquecimiento permite añadir un nombre de dominio a los mensajes que contengan una IP existente en dicha tabla, permitiendo saber, de manera más sencilla de donde provienen y a donde se dirigen las peticiones realizadas al y desde el CESGA.

Una vez finalizada la configuración los enriquecimientos, se pasa a la búsqueda de amenazas. Como se comenta en el capítulo 3, esta búsqueda se puede realizar a través de consultas a HBase y mediante expresiones Stellar. En este apartado se encuentra la oportunidad de centrar la solución en el entorno real, por lo que se decide comparar la IP origen y destino de las diferentes capturas de sFlow con una lista de IPs consideradas malignas por diferentes organizaciones. Esta lista se crea y guarda en el CESGA, lo que permite utilizar datos reales, necesarios para la detección de posibles ataques. Aunque esta lista contiene millones de direcciones, no se aprecia ningún tipo de retardo en el procesamiento, lo que demuestra la eficiencia de HBase y Metron. Esto anima a probar nuevos tipos de búsqueda de amenazas, como por ejemplo, mediante expresiones *Stellar*. Para ello, se prueban las expresiones más interesantes y más próximas a las necesidades del centro, siendo algunas de ellas:

- DOMAIN_REMOVE_SUBDOMAINS: elimina los subdominios de un dominio.
- DOMAIN_REMOVE_TLD: elimina el TLD de un dominio.
- DOMAIN_TO_TLD: obtiene el TLD de un dominio.
- ENDS_WITH: comprueba si un string termina con un sufijo determinado.
- ENRICHMENT_EXISTS: pregunta a una tabla de HBase si existe o no un tipo de enriquecimiento concreto.
- ENRICHMENT_GET: pide a una tabla de HBase los datos de un tipo de enriquecimiento concreto.
- IN_SUBNET: comprueba si una IP pertenece a una subred.
- IS_EMPTY: comprueba si el campo está vacío.
- IS_IP: comprueba si la IP es válida.

- STARTS_WITH: comprueba si una cadena comienza con un prefijo determinado.
- SUBSTRING: obtiene una subcadena contenida en una cadena de texto.
- URL_TO_HOST: obtiene el nombre del host de una URL.

Además de todas estas expresiones, es posible realizar operaciones matemático-lógicas creando configuraciones más complejas. En el caso actual, se comprueba si el dominio presente en el mensaje pertenece a una lista de TLDs determinados, con el fin de detectar las peticiones a y desde dominios extranjeros. Los detalles de todas las configuraciones se encuentran en el apéndice B

Una vez finalizada la búsqueda de amenazas queda añadir la puntuación de triaje a los mensajes en los que se ha encontrado. Para ello se añade en la configuración un campo en el que se evalúan y describen las amenazas detectadas. Esta puntuación va de 0 a 100 y permite clasificar los diferentes mensajes según su factor de riesgo.

5.4. Visualización

En este apartado se mejora la calidad de la información expuesta en el primer panel de control de Kibana tras añadir nuevas fuentes de datos y métodos de enriquecimiento.

En la figura 5.7 se pueden observar los diferentes campos de un mensaje del registro de una máquina del CESGA. Esto permite conocer los valores concretos de un mensaje para tener más información. En la figura 5.8 se muestra parte de un panel de control, incluido el resto en la figura 5.9. Este panel es una modificación del visto anteriormente, con datos de Zeek y Squid. Se han añadido los mensajes de los registros del sistema y las capturas de red a la gráfica circular y al diagrama de barras. También se ha creado un diagrama de barras mostrando el nombre de la máquina de la que provienen los registros. En estas gráficas se puede observar el gran volumen de datos

Figura 5.7: Panel de Kibana con datos de Zeek y Squid

generado por las capturas de red, pues la mayoría de los datos guardados pertenecen a este sensor. En la figura 5.9 se puede ver como se han añadido dos mapas de calor, mostrando la localización de las IPs origen (superior izquierda), y localización de las IPs destino, de las capturas de red.

Finalmente, se muestra un recuento de todas las alertas generadas y un diagrama de barras en el que se dividen las columnas según el tipo de alerta, permitiendo tener una percepción temporal de los intentos de ataque.

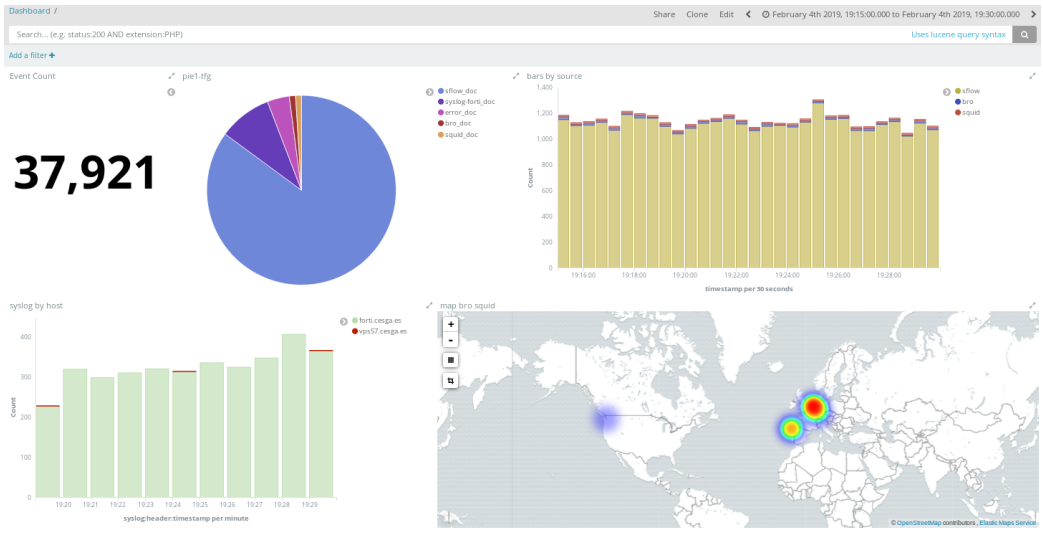

Figura 5.8: Panel de Kibana con diferentes visualizadores

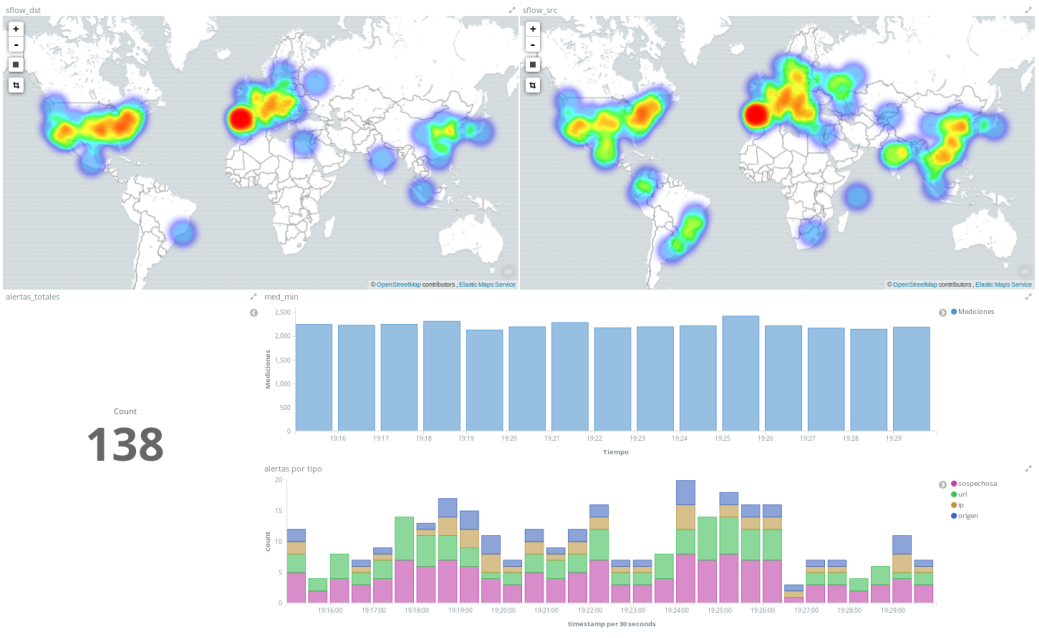

Figura 5.9: Conexiones al CESGA y alertas

Capítulo 6

Conclusiones

En la actualidad, el cibercrimen es uno de los mayores retos a los que se enfrentan no solo las instituciones públicas, sino todas las empresas y organizaciones del mundo. El aumento de dispositivos conectados a la red, así como la cantidad de datos que se deben controlar, dificulta la labor de los encargados de la seguridad de los centros de datos, haciendo necesario implementar sistemas capaces de tratar grandes cantidades de datos y solucionar problemas automáticamente. En el caso del CESGA, los servicios de seguridad y control actuales están implementados sobre un clúster de Hadoop basado en la plataforma de datos de Hortonworks. Esta implementación permite manejar y almacenar gran cantidad de datos haciendo uso de herramientas como Kafka, HBase, HDFS o Hive. Sin embargo, esta solución no permite analizar los datos de manera sencilla, puesto que se utilizan complejos *scripts* programados por los administradores. Es por ello que, en este proyecto, se busca una herramienta que facilite esta tarea.

Se han comparado y analizado diferentes opciones disponibles en el mercado, entre las que destacan Apache Metron y Apache Spot. Se ha realizado un estudio en profundidad de estas dos soluciones, comprobando las diferencias en su arquitectura, métodos de análisis, herramientas utilizadas y capacidad de integración en un entorno existente. Apache Metron parece la solución más apropiada para cumplir los requisitos del centro, por lo que se

opta por crear un entorno de pruebas en el que poder desplegar y configurar la herramienta, para manejar y conocer esta solución antes de introducirla en la plataforma real del CESGA. Apache Metron utiliza, mayoritariamente, herramientas desarrolladas por la fundación Apache, y el mayor contribuyente es Hortonworks, por lo que cumple algunos de los requisitos principales propuestos por el centro, siendo estos: que la herramienta fuera de código abierto, orientada al procesamiento de grandes cantidades de datos, y fácilmente integrable en el entorno real.

Una vez desplegado el entorno y la herramienta se configuraron sus diferentes componentes, divididos en sus tres fases principales: parseado, enriquecimiento e indexado. Estas configuraciones se aplican a cada sensor, que son los encargados de definir como se debe tratar cada fuente de datos.

Otro de los requisitos principales era obtener una herramienta capaz de analizar diferentes fuentes de métricas. Este punto también se ha cubierto, al extender la configuración para poder añadir cualquier tipo de fuente de datos para su análisis en Apache Metron, tanto a través de Ni-Fi como mediante flujos de datos de Kafka. La dificultad de este punto radica en la complejidad del parseador que hay que crear, necesario para transformar las capturas en JSON, pero la posibilidad de utilizar expresiones Grok facilita esta tarea. Este tipo de parseador es sencillo pero lento, por lo que se recomienda utilizarlo únicamente como primera aproximación a la creación de un sensor, y actualizar a un parseador escrito en Java más adelante.

Además de diferentes tipos de parseadores, también se han configurado diferentes métodos de enriquecimiento de datos, como la geolocalización de las IPs mediante la base de datos *GeoLite2*, la búsqueda en tablas de HBase con información introducida por el usuario, y el uso de expresiones del lenguaje Stellar, creado por Apache Metron. De la misma manera, se han utilizado tanto HBase como Stellar en la búsqueda de amenazas.

Una vez analizados los mensajes, estos se almacenan en HDFS y ElasticSearch, permitiendo al administrador consultar las capturas mediante herramientas de visualización como Jupyter o Kibana. Se usa principalmente

Kibana, pues permite crear paneles de control de manera sencilla y gráfica desde su interfaz web. En estos paneles se pueden observar los datos almacenados mediante texto, mapas o gráficas.

A lo largo del despliegue y configuración, tanto del entorno como de la herramienta, se han encontrado numerosos problemas, algunos producidos por fallos del software utilizado, pero la gran mayoría causados por una documentación insuficiente, y, en ocasiones, demasiado vaga para una herramienta tan compleja. De todos modos, conviene destacar que las dificultades de la configuración de la solución provienen del entorno y de la complejidad del funcionamiento interno de Apache Metron, pero no se transmite a las capas superiores de la herramienta. Por lo tanto, aunque el despliegue pueda tener inicialmente un coste superior, el mantenimiento y la adhesión de nuevas funcionalidades resulta relativamente sencillo.

Finalmente, debemos destacar las capacidades de análisis de la solución propuesta y la cantidad de aplicaciones posibles, propiciadas por la versatilidad de Apache Metron. Además, al no funcionar como un bloque inmutable, la solución permite hacer uso de las capacidades de todos los servicios disponibles en un entorno Hadoop para añadir nuevas funcionalidades. Se pueden compartir datos a través del almacenamiento distribuido, las bases de datos o el flujo de datos de Kafka. También es posible modificar las topologías o añadir otras nuevas adaptadas a las necesidades del centro.

6.1. Trabajo futuro

Debido a que Apache Metron es una herramienta en continuo desarrollo y con gran número de aplicaciones, es difícil listar todas sus posibilidades. Aún así, conociendo su funcionamiento, y teniendo en cuenta las necesidades del CESGA, se pueden concretar algunas tareas realizables a corto plazo, como por ejemplo:

- Añadir nuevas fuentes de enriquecimiento y riesgos: esto permite aumentar la información de las capturas y la capacidad de la herramienta

para encontrar y resolver amenazas.

- Uso de Stellar: este lenguaje permite realizar un análisis más avanzado de los diferentes campos de las capturas mediante el uso de expresiones matemático-lógicas, así como de las funciones incluidas.

- Integración en el entorno real: como se ha comprobado a lo largo del proyecto, Apache Metron ofrece una alta compatibilidad con los servicios actualmente desplegados en el entorno Big Data del CESGA, por lo que su integración no debería acarrear ningún problema. Aún así, el paso a producción nunca está exento de riesgos, y ha de realizarse en el momento adecuado para la institución.

- Actualización a Metron 0.7.0: puesto que las nuevas versiones aportan nuevas funcionalidades y mejoran el rendimiento y funcionamiento, solucionando errores y problemas de integración entre servicios, es necesario mantener una versión actualizada de la herramienta.

Bibliografía

[1] Alexa Top 1M. https://github.com/mozilla/cipherscan/tree/master/top1m.

[2] Apache Ambari. https://ambari.apache.org/.

[3] Apache Flume. https://flume.apache.org/.

[4] Apache Hadoop. https://hadoop.apache.org/.

[5] Apache Kafka. http://kafka.apache.org/.

[6] Apache Metron. https://metron.apache.org/.

[7] Apache Metron Enrichment. https://metron.apache.org/current-book/metron-platform/metron-enrichment/index.html.

[8] Apache Metron Indexing. https://metron.apache.org/current-book/metron-platform/metron-indexing/index.html.

[9] Apache Metron Parsing. https://metron.apache.org/current-book/metron-platform/metron-parsers/index.html.

[10] Apache Spark. https://spark.apache.org/.

[11] Apache Spot. http://spot.incubator.apache.org/.

[12] Apache Storm. https://storm.apache.org/.

[13] Apache Zookeeper. https://zookeeper.apache.org/.

[14] Cloudera. https://www.cloudera.com.

[15] ElasticSearch. https://www.elastic.co/.

[16] Fail2Ban. https://www.fail2ban.org/wiki/index.php/Main_Page.

[17] Graylog. https://www.graylog.org/.

[18] Grok Debugger. https://grokdebug.herokuapp.com/.

[19] HortonWorks. https://es.hortonworks.com/.

[20] Kappa Architecture. http://milinda.pathirage.org/kappa-architecture.com/.

[21] Kibana. https://www.elastic.co/products/kibana.

[22] Logcheck. http://logcheck.org/.

[23] Logging Bro Output to Kafka. https://github.com/apache/metron-bro-plugin-kafka.

[24] Loggly. https://www.loggly.com/.

[25] Logwatch. https://wiki.archlinux.org/index.php/Logwatch.

[26] Loom. https://www.loomsystems.com/.

[27] Metron Architecture. https://cwiki.apache.org/confluence/display/METRON/Metron+Architecture.

[28] Ni-Fi. https://nifi.apache.org/.

[29] omkafka. https://www.rsyslog.com/doc/master/configuration/modules/omkafka.html.

[30] PandoraFMS. https://pandorafms.com/.

[31] pmacct. http://www.pmacct.net/.

[32] RFC 5424. https://tools.ietf.org/html/rfc5424.

[33] sFlow. https://sflow.org/.

[34] Soltra Edge. https://www.soltra.com/en/.

[35] Splunk. https://www.splunk.com/.

[36] Spot Architecture. http://spot.incubator.apache.org/get-started/architecture/.

[37] Stix/Taxii. https://oasis-open.github.io/cti-documentation/.

[38] Zeek. https://en.wikipedia.org/wiki/Zeek.

[39] CESGA. https://www.cesga.es/es/cesga.

[40] CESGA Big Data. http://hadoop.cesga.es.

[41] Hiscox. The Hiscox Cyber Readiness Report. *Hiscox*, 2017.

[42] Javier Pagola. Talento en ciberseguridad, entre el déficit actual y el reto de ser profesión del futuro. *ABC*, 2017.

[43] Vern Paxson. Renaming the Bro Project. https://blog.zeek.org/2018/10/renaming-bro-project_11.html, 2018.

[44] Alec Ross. Want job security? Try online security. *Wired*, 2016.

[45] Enigma Software. Top 20 Countries Found to Have the Most Cybercrime. *Enigma Software*, 2012.

[46] George Vetticaden. Evolution of apache metron. *Metron Wiki*, 2016.

Apéndice A

Instalación de Apache Metron 0.6.1 en CentOS 7

En este apéndice se detalla la instalación de Apache Metron 0.6.1 en CentOS 7. En este ejemplo también se cubre la instalación de un clúster de Hadoop, necesario para el correcto funcionamiento de la herramienta.

A.1. Prerrequisitos

Los diferentes nodos del clúster deben disponer de, aproximadamente, 16GB de RAM y 50GB de almacenamiento. En este caso se usará un clúster de 3 nodos.

Además, los nodos deben cumplir los siguientes requisitos antes de iniciar la instalación:

- Distribución CentOS 7 instalada en todas las máquinas
- Conexión SSH sin contraseña entre nodos. Únicamente necesario desde el nodo maestro a los esclavos. Para ello se crean las claves:

```
cat /dev/zero | ssh-keygen -q -N "" 2>/dev/null
cat .ssh/id_rsa.pub >> .ssh/authorized_keys
chmod 400 .ssh/authorized_keys
```

  Y se envian a los nodos esclavo:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub <ip_esclavo>
```

- Añadir el repositorio epel y actualizar el sistema:

```
yum install epel-release -y
yum update -y
```

- Incrementar los límites de ElasticSearch y Storm en el nodo en el que van a ser instalados:

```
echo -e "elasticsearch - memlock unlimited\nstorm - nproc 257597" >> /
    etc/security/limits.conf
```

- Aumentar el número máximo de procesos y ficheros abiertos:

```
ulimit -n 32768
ulimit -u 65536
echo -e "* - nofile 32768\n* - nproc 65536" >> /etc/security/limits.
    conf
```

- Desactivar IPv6 para evitar que los servicios usen estas direcciones:

```
sysctl -w net.ipv6.conf.all.disable_ipv6=1
sysctl -w net.ipv6.conf.default.disable_ipv6=1
echo -e "\n# Disable IPv6\nnet.ipv6.conf.all.disable_ipv6 = 1\nnet.
    ipv6.conf.default.disable_ipv6 = 1" >> /etc/sysctl.conf
```

- Desactivar *Transparent HugePage*. Para ello se crea un servicio del systemd que lo haga en cada reinicio, introduciendo las siguientes líneas en el fichero `/etc/systemd/system/disable-thp.service`:

```
1 [Unit]
2 Description=Disable Transparent Huge Pages (THP)
3
4 [Service]
5 Type=simple
6 ExecStart=/bin/sh -c "echo 'never' > /sys/kernel/mm/
      transparent_hugepage/enabled && echo 'never' > /sys/kernel/mm/
      transparent_hugepage/defrag"
7
8 [Install]
9 WantedBy=multi-user.target
```

Una vez creado, se reinicia systemd y se activa el servicio:

```
systemctl daemon-reload
systemctl start disable-thp
systemctl enable disable-thp
```

- Desactivar SELinux:

```
setenforce 0
sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
```

- Instalar el software necesario de Ambari:

```
yum install git wget curl rpm tar unzip scp bzip2 wget createrepo yum-
    utils ntp python-pip psutils python-psutil ntp libffi-devel gcc
    openssl-devel -y
pip install --upgrade pip
pip install requests==2.6.1
```

Es necesario instalar una versión de *requests* igual o superior a 2.6.1 para el correcto funcionamiento del indexado de Apache Metron.

- Instalar Java en el nodo de Metron e indicar la ruta:

```
yum install java-1.8.0-openjdk java-1.8.0-openjdk-devel -y
export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s_/jre/bin/java___
    ")
echo 'export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s_/jre/bin/
    java___")' > /etc/profile.d/java_18.sh
chmod +x /etc/profile.d/java_18.sh
source /etc/profile.d/java_18.sh
```

- Instalar la base de datos utilizada por Metron REST:

```
yum install -y mariadb-server mysql
yum install mysql-connector-java -y
```

A.2. Compilación

Una vez instalados los requisitos de Metron y Ambari se prepara uno de los nodos para la compilación de los diferentes paquetes. Primero se instala Docker, para ello se deben seguir los siguientes pasos:

- Eliminar las posibles instalaciones antiguas de Docker:

```
yum remove docker docker-client docker-client-latest docker-common
    docker-latest docker-latest-logrotate docker-logrotate docker-
    engine
```

- Instalar las dependencias y añadir los repositorios de Docker:

```
yum install -y yum-utils device-mapper-persistent-data lvm2
yum-config-manager --add-repo https://download.docker.com/linux/centos
    /docker-ce.repo
```

- Instalar Docker:

```
yum install docker-ce docker-ce-cli
  containerd.io
```

- Iniciar el servicio:

```
systemctl start docker
```

- Opcionalmente se puede probar que Docker funciona correctamente usando una imagen de prueba:

```
docker run hello-world
```

Una vez instalado Docker, se puede descargar el paquete de Metron y preparar un contenedor para la compilación:

- El paquete de Metron se puede obtener de su página de GitHub:

```
git clone https://github.com/apache/metron
```

- Desde `metron/metron-deployment/packaging/docker/ansible-docker` se construye y ejecuta el contenedor de Docker:

```
docker build -t ansible-docker:latest .
docker run -it -v `pwd`/../../../..:/root/metron -v ~/.m2:/root/.m2
    ansible-docker:latest bash
```

- Una vez dentro del contenedor se instalan las herramientas de desarrollo devtoolset-6:

```
yum install devtoolset-6 -y
```

- Desde el contenedor se compilan los diferentes paquetes:

  - Metron:

    ```
    cd /root/metron
    mvn clean install package -DskipTests
    ```

  - RPMs:

    ```
    cd metron-deployment
    mvn clean package -Pbuild-rpms -DskipTests
    ```

  - MPack:

    ```
    mvn clean package -DskipTests
    ```

  Estos paquetes se encuentran en las siguientes rutas:

  - RPM: `metron/metron-deployment/packaging/docker/rpm-docker/RPMS/noarch/*.rpm`
  - Metron MPack: `metron/metron-deployment/packaging/ambari/metron-mpack/target/*tar.gz`
  - ElasticSearch MPack: `metron/metron-deployment/packaging/ambari/elasticsearch-mpack/target/*tar.gz`

  Se deben mover todos estos paquetes a un mismo directorio (p.e: /localrepo) en todos los nodos y crear un repositorio local en todos ellos:

```
createrepo /localrepo
```

A.3. Preparando la base de datos

Antes de instalar el clúster se debe configurar la base de datos de Metron REST siguiendo estos pasos:

1. Conectarse a MySQL y crear la base de datos:

```
mysql -u root -p -e "CREATE DATABASE IF NOT EXISTS metronrest;"
```

2. Crear un usuario Metron y darle permisos:

```
CREATE USER 'metron'@'$REST_HOST' IDENTIFIED BY 'passwd';
GRANT ALL PRIVILEGES ON metronrest.* TO 'metron'@'$REST_HOST';
```

3. Crear tablas de usuarios y autoridades:

```
use metronrest;
create table if not exists users(
username varchar(50) not null primary key,
password varchar(50) not null,
enabled boolean not null
);
create table authorities (
username varchar(50) not null,
authority varchar(50) not null,
constraint fk_authorities_users foreign key(username) references
users(username)
);
create unique index ix_auth_username on authorities (username,
    authority);
```

4. Añadir los usuarios a la base de datos:

```
use metronrest;
insert into users (username, password, enabled) values ('metron', '
    passwd',1);
insert into authorities (username, authority) values ('metron', '
    ROLE_USER');
```

5. Salir de MySQL:

```
quit
```

A.4. Instalación

Una vez compilados los paquetes se puede proceder a la instalación. Para ello se instala un servidor Ambari, desde el que se desplegará el clúster y sus servicios. Esta instalación se realiza siguiendo los siguientes pasos:

- Preparar el entorno activando sincronización de tiempo, desactivando el cortafuegos y SELinux en todos los nodos:

```
systemctl enable ntpd
systemctl start ntpd
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -t nat -F
iptables -t mangle -F
iptables -F
iptables -X
iptables-save > /etc/sysconfig/iptables
systemctl stop firewalld
systemctl disable firewalld
setenforce 0
```

- En el caso de estar utilizando Python 2.7.5 o superior se debe desactivar la comprobación de certificados para evitar el siguiente error en la instalación de Ambari:

```
[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c
    :579)
```

Para desactivar esta comprobación:

```
sed -i 's/verify=platform_default/verify=disable/' /etc/python/cert-
    verification.cfg
```

- Es necesario que cada nodo tenga constancia de su nombre y de los del resto de nodos. El fichero `/et/hosts` de todos los nodos debe ser similar al siguiente:

```
1 10.10.10.1 node1
2 10.10.10.2 node2
3 10.10.10.3 node3
```

- En el nodo maestro se descarga y prepara el repositorio de Ambari.

```
wget -nv http://public-repo-1.hortonworks.com/ambari/centos7/2.x/
    updates/2.4.3.0/ambari.repo -O /etc/yum.repos.d/ambari.repo
```

- Se comprueba que el repositorio ha sido añadido correctamente:

```
yum repolist | grep ambari
# Updates-ambari-2.4.3.0    ambari-2.4.3.0
# Updates                                12
```

- Se instala el servidor de Ambari:

```
yum install ambari-server -y
ambari-server setup -s
```

- Se añaden los paquetes MPack de Metron y ElasticSearch:

```
ambari-server install-mpack --mpack=/localrepo/*.tar.gz --verbose
```

- Finalmente se inicia el servicio:

```
ambari-server start
```

Una vez instalado Ambari ya se puede acceder a la interfaz web e iniciar la instalación de Metron desde la siguiente URL: `http://<ip-nodo-maestro>:8080`. El usuario/contraseña por defecto es admin/admin.

Una vez dentro de la interfaz se despliega el clúster con todos los servicios requeridos usando el asistente de instalación y siguiendo las siguientes indicaciones:

- En la página de inicio se pone el nombre que se quiere usar para el clúster.
- En la selección de versión se marca la opción de *Repositorio Público* y se comprueba que el directorio `/localrepo` aparezca.
- En las opciones de instalación se indica el nombre de los nodos que van a formar parte del clúster. Deben ser los mismos que los incluidos en `/etc/hosts`. Además, se añade la clave SSH privada del nodo maestro.
- En este momento Ambari instala el *ambari-agent* en todos los nodos y comprueba que sean válidos. En caso de que algún nodo no pueda ser registrado se deben añadir las siguientes lineas en el fichero `/etc/ambari-agent/conf/ambari-agent.ini`:

```
[security]
[force_https_protocol=PROTOCOL_TLSv1_2
```

- Una vez configurados los agentes, se deben seleccionar los siguientes servicios:
  - HDFS
  - YARN + MapReduce2
  - Tez
  - HBase
  - Pig
  - Zookeeper
  - Storm
  - Flume
  - Ambari Metrics
  - Kafka
  - Elasticsearch
  - Kibana
  - Metron
  - Slider
  - Zeppelin Notebook
  - Hive
- Una vez seleccionados los servicios se deben escoger los maestros de cada uno. *Kafka Broker* debe estar en todos los nodos, y ElasticSearch en un nodo diferente al de el Servidor de Ambari. El resto de servicios se pueden distribuir libremente, a excepción de que el asistente de instalación indique que deben ir juntos.

- Para asignar los esclavos y clientes se selecciona el campo .all. los siguientes servicios:
  - DataNoder
  - NodeManager
  - RegionServer
  - Supervisor
  - Client
- Ahora se deben configurar algunos servicios para su correcto funcionamiento:
  - HDFS ->Configs:

    "NameNode Java heap size"de 1024 MB a 4096 MB como mínimo.
  - ElasticSearch:

    zen_discovery_ping_unicast_hosts a ip/nombre del nodo maestro.

    Advanced elastic-site->network_host a '0.0.0.0'.
  - Kibana:

    kibana_es_url a '`http://<ip-maestro-elasticsearch>:9200`'.
  - Metron:

    Elasticsearch Hosts a la IP del maestro de ElasticSearch.
  - Metron->REST:

    Metron JDBC client path: '`/usr/share/java/mysql-connector-java.jar`'

    Metron JDBC Driver: 'com.mysql.jdbc.Driver'.

    Metron JDBC password: 'passwd'.

    Metron JDBC platform: 'mysql'.

Metron JDBC URL: jdbc: '`mysql://127.0.0.1:3306/metronrest`'.

Metron JDBC username: 'metron'.

- Poner contraseña en los servicios que lo piden.

- Una vez configurados los servicios se inicia su instalación.

Una vez instalados todos los servicios del clúster se puede acceder a las interfaces de las siguientes herramientas[1]:

- Ambari: `http://nodeX:8080`
- Kibana: `http://nodeX:5000`
- Sensor Status (monit): `http://nodeX:2812`
- Elasticsearch: `http://nodeX:9200`
- Storm UI: `http://nodeX:8744`
- Metron REST interface: `http://nodeX:8082/swagger-ui.html`
- Management UI: `http://nodeX:4200`
- Apache Nifi: `http://nodeX:8089/nifi`
- Zookeeper: `http://nodeX:2181`
- Kafka: `http://nodeX:6667`

En el apéndice B se detalla la instalación de herramientas adicionales y la creación y configuración de los diferentes sensores.

[1]Cambiar nodeX por el nombre del nodo correspondiente

Apéndice B

Instalación y configuración de sensores en Apache Metron

En este apéndice se detalla la instalación y configuración de diferentes herramientas de captura de datos, así como de los sensores encargados de analizar dichas métricas.

B.1. Ni-Fi

Ni-Fi se utiliza para poder conectar con Apache Metron los servicios y herramientas que no pueden escribir en un flujo de datos de Kafka directamente.

B.1.1. Instalación

Para instalar esta herramienta únicamente se debe descargar y descomprimir el paquete correspondiente a la versión que se va a utilizar:

```
wget apache.rediris.es/nifi/1.8.0/nifi-1.8.0-bin.tar.gz
tar xf nifi-1.8.0-bin.tar.gz
cd nifi-1.8.0-bin
```

Antes de de ejecutar el script de instalación e iniciar el servicio, se recomienda cambiar el puerto en el que la interfaz web de Ni-Fi escucha para que

no coincida con el del servidor de Ambari. Esto se puede hacer en el fichero `conf/nifi.properties`:

```
nifi.web.http.port=8089
```

Finalmente se instala e inicia el servicio:

```
bin/nifi.sh install nifi
service nifi start
```

B.2. Zeek

Zeek es una herramienta de código abierto y análisis de red, desarrollada en 1994. Esta herramienta captura y envía paquetes IP a un analizador de eventos, que los acepta, realizando sobre ellos un análisis a mayor profundidad, o los desecha.

B.2.1. Instalación

Para instalar la herramienta, se descargan los paquetes ya compilados de la página oficial y se instalan mediante el administrador de paquetes *yum*:

```
cd /etc/yum.repos.d/
wget http://download.opensuse.org/repositories/network:bro/CentOS_7/network:
    bro.repo
yum install -y bro
```

Una vez instalado, se añade el plugin. Para ello es necesario instalar una librería de Kafka adicional, *librdkafka*, una implementación en C y C++ del protocolo Kafka:

```
curl -L https://github.com/edenhill/librdkafka/archive/v0.11.5.tar.gz | tar
    xvz
cd librdkafka-0.11.5/
./configure
make && make install
```

Una vez instalado, se descarga y compila el plugin:

```
git clone https://github.com/apache/metron-bro-plugin-kafka
cd metron-bro-plugin-kafka/
./configure --bro-dist=$BRO_SRC
make && make install
```

Con el siguiente comando se comprueba si la instalación ha sido exitosa:

```
bro -N Apache::Kafka
```

Ahora se debe indicar a Zeek que utilice este plugin, para ello se incluyen las siguientes líneas en el fichero de configuración *local.bro*:

```
1 @load Apache/Kafka/logs-to-kafka.bro
2 redef Kafka::logs_to_send = set(DNS::LOG,HTTP::LOG);
3 redef Kafka::tag_json = T;
4 redef Kafka::topic_name = "bro";
5 redef Kafka::kafka_conf = table(
6     ["metadata.broker.list"] =
7         "node1:9092,node2:9092,node3:9092"
8 );
```

- **@load**: especifica la ruta al módulo que se utiliza.
- **redef Kafka::***: indica al módulo la nueva configuración de los diferentes campos.
  - **logs_to_send**: indica el protocolo de los paquetes que se van a capturar.
  - **tag_json**: si verdadero se añade al JSON enviado el protocolo del paquete.
  - **topic_name**: indica el nombre del topic que se va a utilizar para enviar los mensajes.

- **kafka_conf**: redefine diferentes campos de la configuración de Kafka, en el caso del fichero de configuración indicamos la lista de *brokers* junto al puerto en el que escuchan.

Una vez especificada la nueva configuración se debe recargar el servicio para que Zeek publique las capturas en el topic de Kafka.

```
broctl reload
```

Antes de configurar el procesamiento de Apache Metron, se comprueba que el servicio esté escribiendo en el topic de Kafka. Para ello se utilizan las herramientas de consola de Kafka, que permiten crear consumidores y productores de un topic, así como conocer los topics existentes y manejarlos. Primero, se comprueba que el topic de bro ha sido creado automáticamente, para ello se puede consultar a Kafka por todos los topics existentes actualmente en el clúster.

```
bin/kafka-topics.sh --zookeeper localhost:2181 --list
```

En el caso de no existir, se crea, indicando el nombre del topic, la cantidad de particiones y el número de réplicas.

```
bin/kafka-topics.sh --zookeeper localhost:2181 --create --topic bro --
    partitions 1 --replication-factor 1
```

Se vuelve a comprobar que existe y se crea un consumidor, al cual se le indica la dirección y puerto de Zookeeper así como el topic que se va a consumir.

```
bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic bro
```

Mientras se ejecuta el consumidor se realizan en otra consola algunas operaciones que requieran de resolución DNS, como un *ping* a un dominio cualquiera o la descarga de una página web mediante *wget*.

Al poco deben aparecer, en la consola donde se ejecuta el consumidor, las diferentes peticiones DNS, indicando los diferentes campos, como la IP de origen y destino, el protocolo utilizado, etc...

B.2.2. Configuración

Diferentes ficheros de configuración usados en Apache Metron para cada fase del análisis:

- Parseado: `$METRON_HOME/config/zookeeper/parsers/bro.json`:

```
"parserClassName":"org.apache.metron.parsers.bro.BasicBroParser",
"sensorTopic":"bro",
"parserConfig": {}
```

- Enriquecimiento: `$METRON_HOME/config/zookeeper/enrichments/bro.json`:

```
"enrichment" : {
    "fieldMap": {
    "geo": [
        "ip_dst_addr",
        "ip_src_addr"
    ], "host": [
        "host",
        "ip_dst_addr"
    ]}
}, "threatIntel": {
    "fieldMap": {
        "hbaseThreatIntel": ["ip_src_addr", "ip_dst_addr"]
    }, "fieldToTypeMap": {
        "ip_src_addr" : ["malicious_ip"],
        "ip_dst_addr" : ["malicious_ip"]
    }
}
```

- Indexado: `$METRON_HOME/config/zookeeper/indexing/bro.json`:

```
"hdfs":{
    "index":"bro",
    "batchSize":50,
    "enabled":true
},
"elasticsearch":{
```

```
    "index":"bro",
    "batchSize":50,
    "enabled":true
}
```

B.3. Squid

B.3.1. Instalación

Squid se instala mediante el administrador de paquetes *yum*:

```
yum install -y squid
systemctl enable squid
systemctl start squid
```

Script para crear consultas de Squid.

```
while sleep 2; do cat links.txt | shuf -n 1 | xargs -i squidclient -v {};
    done
```

A la hora de enviar la información registrada en los logs a Metron, se debe utilizar Ni-Fi.

B.3.2. Configuración

Diferentes ficheros de configuración usados en Apache Metron para cada fase del análisis:

- Parseado: `$METRON_HOME/config/zookeeper/parsers/squid.json`:

```
"parserClassName":"org.apache.metron.parsers.GrokParser",
"sensorTopic":"squid",
"parserConfig":{
    "grokPath":
        "/patterns/squid",
        "patternLabel":"SQUID_DELIMITED",
        "timestampField":"timestamp"
    },
"fieldTransformations":[{
    "input":[],
```

```
    "output":[
        "full_hostname",
        "domain_without_subdomains"
    ],
    "transformation":"STELLAR",
    "config":{
        "full_hostname":"URL_TO_HOST(url)",
        "domain_without_subdomains":"DOMAIN_REMOVE_SUBDOMAINS(
    full_hostname)"
    }
}]
```

- Enriquecimiento: `$METRON_HOME/config/zookeeper/enrichments/squid.json`:

```
"enrichment" : {
    "fieldMap" : {
        "geo" : [ "ip_dst_addr" ],
        "hbaseEnrichment" : [ "domain_without_subdomains" ]
    }, "fieldToTypeMap" : {
        "domain_without_subdomains" : [ "whois" ]
    }, "config" : { }
},"threatIntel" : {
    "fieldMap" : {
        "hbaseThreatIntel" : [ "domain_without_subdomains" ],
        "stellar" : {
            "config" : {
                "is_alert" :
                    "(exists(is_alert) and is_alert) or (not(ENDS_WITH
    (domain_without_subdomains, '.com')) or ENRICHMENT_EXISTS('
    zeusList',domain_without_subdomains,'threatintel','t'))"
            }
        }
    }, "fieldToTypeMap" : {
        "domain_without_subdomains" : [ "zeusList" ]
    },
    "config" : { },
    "triageConfig" : {
        "riskLevelRules" : [{
            "name" : "zeusList is alerted",
            "comment" : "Threat intelligence enrichment type zeusList
    is alerted.",
            "rule": "exists(threatintels.hbaseThreatIntel.
    domain_without_subdomains.zeusList)",
            "score" : 5
```

```
        },{
            "name" : "URL sospechosa",
            "comment" : "La URL no termina en .com",
            "rule": "not(ENDS_WITH(domain_without_subdomains, '.com'))
    ",
            "score" : 10
        }],
        "aggregator" : "MAX",
        "aggregationConfig" : { }
    }
}
```

- Indexado: `$METRON_HOME/config/zookeeper/indexing/squid.json`:

```
"hdfs":{
    "index":"squid",
    "batchSize":10,
    "enabled":true
},
"elasticsearch":{
    "index":"squid",
    "batchSize":10,
    "enabled":true
},
"alert" : {
    "type": "nested"
}
```

B.4. Syslog

Diferentes ficheros de configuración usados en Apache Metron para cada fase del análisis:

- Parseado: `$METRON_HOME/config/zookeeper/parsers/syslog.json`:

```
"parserClassName":"org.apache.metron.parsers.syslog.Syslog5424Parser",
"sensorTopic":"syslog_forti",
"parserConfig": {
    "nilPolicy": "DASH"
}
```

- Enriquecimiento: `$METRON_HOME/config/zookeeper/enrichments/syslog.json`:

```
"enrichment":{
    "fieldMap":{},
    "fieldToTypeMap":{},
    "config":{}
},
"threatIntel":{
}
```

- Indexado: `$METRON_HOME/config/zookeeper/indexing/syslog.json`:

```
"hdfs":{
    "index":"syslog",
    "batchSize":50,
    "enabled":true
},
"elasticsearch":{
    "index":"syslog",
    "batchSize":50,
    "enabled":true
}
```

B.5. SFlow

Diferentes ficheros de configuración usados en Apache Metron para cada fase del análisis:

- Parseado: `$METRON_HOME/config/zookeeper/parsers/sflow.json`:

```
"parserClassName":"org.apache.metron.parsers.json.JSONMapParser",
"sensorTopic":"ipfix",
"fieldTransformations" : [{
    "transformation" : "RENAME",
        "config" : {
            "ip_src" : "ip_src_addr",
            "ip_dst" : "ip_dst_addr",
```

```
            "port_src" : "ip_src_port",
            "port_dst" : "ip_dst_port"
        }
}]
```

- Enriquecimiento: `$METRON_HOME/config/zookeeper/enrichments/sflow.json`:

```
"enrichment" : {
    "fieldMap" : {
        "geo" : [ "ip_dst_addr", "ip_src_addr" ],
        "host" : [ "host", "ip_dst_addr" ],
        "stellar" : {
            "config" : {
                "host_dst" : "MAP_GET('host',ENRICHMENT_GET('top-1m',
    ip_dst_addr,'top-1m','t'),'')",
                "host_src" : "MAP_GET('host',ENRICHMENT_GET('top-1m',
    ip_src_addr,'top-1m','t'),'')"
            }
        }
    }
},
"threatIntel" : {
    "fieldMap" : {
        "stellar" : {
            "config" : {
                "bad_dst_ip" : "ENRICHMENT_EXISTS('bad_ip',ip_dst_addr
    ,'my-ips','t')",
                "bad_src_ip" : "ENRICHMENT_EXISTS('bad_ip',ip_src_addr
    ,'my-ips','t')",
                "is_alert" : "(exists(is_alert) and is_alert) or (
    bad_dst_ip or bad_src_ip)"
            }
        }
    },
    "fieldToTypeMap" : { },
    "config" : { },
    "triageConfig" : {
        "riskLevelRules" : [{
            "name" : "IP DESTINO SOSPECHOSA",
            "comment" : "La ip de destino es sospechosa",
            "rule": "bad_dst_ip",
            "score" : 50
        },{
            "name" : "IP ORIGEN SOSPECHOSA",
```

```
            "comment" : "La ip de origen es sospechosa",
            "rule": "bad_src_ip",
            "score" : 25
        }],
        "aggregator" : "MAX",
        "aggregationConfig" : { }
    }
}
```

- Indexado: `$METRON_HOME/config/zookeeper/indexing/sflow.json`:

```
"hdfs" : {
    "index": "sflow",
    "batchSize": 100,
    "enabled" : true
},
"elasticsearch" : {
    "index": "sflow",
    "batchSize": 100,
    "enabled" : true
},
"alert" : {
    "type": "nested"
}
```

Apéndice C

Comandos, herramientas y anotaciones a tener en cuenta

En este apéndice se detallan algunos comandos y herramientas útiles a la hora de manejar Apache Metron.

C.1. Kafka

```
KAFKA_HOME = /usr/hdp/2.5.3.0-37/kafka/
```

- Mostrar todos los topics:

```
bin/kafka-topics.sh --list --zookeeper $ZOOKEEPER_HOST:2181
```

- Mostrar datos de un topic

```
bin/kafka-topics.sh --describe --zookeeper $ZOOKEEPER_HOST:2181 --
    topic indexing
```

- Borrar un topic (la opción *"delete.topic.enable"* debe ser *"true"* en la configuración de Kafka):

```
bin/kafka-topics.sh --delete --zookeeper $ZOOKEEPER_HOST:2181 --topic
    indexing
```

- Si el topic no aparece como borrado proceder de la siguiente manera:

  1. Parar el servidor de Kafka
  2. Borrar el directorio del topic en `/kafka-logs/`
  3. Conectarse al servidor de Zookeper

     ```
     /usr/hdp/2.5.3.0-37/zookeeper/bin/zkCli.sh
     ```

  4. Comprobar que existe el topic y eliminarlo de los siguientes directorios

     ```
     ls /brokers/topics
     rmr /brokers/topics/$TOPIC_NAME
     ls /admin/delete_topics
     rmr /admin/delete_topics/$TOPIC_NAME
     ```

  5. Reiniciar el servidor de Kafka

- Producir mensajes para un topic:

```
bin/kafka-console-producer.sh --broker-list $KAFKA_BROKER_LIST:9092 --
    topic TOPIC_NAME
```

- Consumir mensajes de un topic:

```
bin/kafka-console-consumer.sh --zookeeper $ZOOKEEPER_HOST:2181 --topic
     ipfix
```

- Listar grupos de consumidores:

```
bin/kafka-consumer-groups.sh --new-consumer --bootstrap-server
    $KAFKA_BROKER_LIST:9092 --list
```

- Mostrar datos de un grupo de consumidores:

```
bin/kafka-consumer-groups.sh --new-consumer --bootstrap-server
    $KAFKA_BROKER_LIST:9092 --describe --group GROUP_NAME
```

C.2. Zookeeper

```
METRON_HOME = /usr/metron/0.6.1/
```

- Directorios de configuración de las diferentes fases de los sensores:
  - `$METRON_HOME/config/zookeeper/parsers/SENSOR_NAME.json`
  - `$METRON_HOME/config/zookeeper/enrichments/SENSOR_NAME.json`
  - `$METRON_HOME/config/zookeeper/indexing/SENSOR_NAME.json`
- Directorio de los parseadores GROK:
  - `$METRON_HOME/patterns/SENSOR_NAME`
- Mostrar configuraciones almacenadas en Zookeeper:

```
$METRON_HOME/bin/zk_load_configs.sh -m DUMP -z $ZOOKEEPER_HOST:2181
```

- Subir las configuraciones locales a Zookeeper:

```
$METRON_HOME/bin/zk_load_configs.sh -m PUSH -z $ZOOKEEPER_HOST:2181 -i
    $METRON_HOME/config/zookeeper
```

- Actualizar las configuraciones locales con las almacenadas en Zookeeper:

```
$METRON_HOME/bin/zk_load_configs.sh -m PULL -z $ZOOKEEPER_HOST:2181 -f
    -o $METRON_HOME/config/zookeeper/
```

C.3. ElasticSearch

Desde la interfaz web de Elasticsearch se pueden realizar diferentes consultas:

- Consultar el estado del clúster:

`http://ip_nodo_maestro:9200/_cluster/health`

- Consultar los índices almacenados:

  `http://ip_nodo_maestro:9200/_cat/indices?v`

- Consultar las plantillas:

  `http://ip_nodo_maestro:9200/_template/*`

C.4. Solución de problemas comunes

- En caso de que Zeppelin no se inicie correctamente:

```
ps aux | grep zeppelin
kill <zeppelin_java_pid>
mkdir /var/run/zeppelin
chown zeppelin.hadoop zeppelin/
```

- Tabla *threatintel* no encontrada:

```
echo "create 'threatintel','t'" | hbase shell
echo "scan 'threatintel'" | hbase shell
```