Universidad de Valladolid

Grado en Ingeniería de Tecnologías de Telecomunicación

2018-2019

Trabajo Fin de Grado

Monitorización de paquetes a su paso a través de un cortafuegos basado en el proyecto Netfilter

Luis Ruiz Mayorga

Tutor

Federico Simmross Wattenberg

Septiembre de 2019

«Yo sólo puedo mostrarte la puerta. Tú eres quien debe atravesarla.»

Morfeo *en la película* "Matrix"

RESUMEN

Durante los últimos años, el constante crecimiento de las redes y los sistemas ha propiciado el aumento de la importancia del uso de sistemas de monitorización. A su vez, el aumento de ataques cibernéticos ha vuelto imprescindible el uso de cortafuegos de red como primera línea de defensa de estas redes y sistemas. El cortafuegos del *kernel* de Linux, IPTables, desarrollado por el proyecto Netfilter, es uno de los más extendidos y usados en redes pequeñas y medianas. Pese a sus ventajas como la enorme flexibilidad o su gratuidad, presenta una dificultad inherente a su diseño, pues resulta complejo realizar un seguimiento de los paquetes de red a través de sus diversas tablas, reglas y cadenas.

Por ello, en este Trabajo de Fin de Grado se va a implementar un sistema de monitorización que permita realizar de forma gráfica un seguimiento de estos paquetes y permita visualizar la configuración del cortafuegos. Para su implementación, se utilizarán las distintas herramientas que conforman la pila Elastic: Beats, Logstash, Elasticsearch y Kibana.

Palabras clave: Netfitler, IPTables, cortafuegos, Elasticsearch, Logstash, Kibana, Beats, Elastic Stack, Filebeat, monitorización, registro, *log*, regla, cadena, tabla, *script*.

DEDICATORIA

A mi tutor, Federico, por todo lo que me has enseñado en la carrera y por haberme dado la oportunidad de realizar este Trabajo de Fin de Grado.

A mi familia, por haberme dado la oportunidad de irme fuera a estudiar, y por lo que queda. Por el apoyo y la ayuda que me habéis prestado siempre.

Y a mis amigos, porque siempre habéis estado ahí, por todos los buenos momentos.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Este primer capítulo pretende explicar y dar una perspectiva global del presente Trabajo de Fin de Grado. Para ello, se dividirá en cuatro apartados en los cuales se desarrollarán las motivaciones del trabajo, los objetivos que persigue, así como la metodología y la estructura del resto de la memoria.

1.1. Motivación

La monitorización de sistemas es una práctica cada vez más presente en todos los ámbitos. El avance de los mismos provoca un incremento tanto en el número de sistemas como de su complejidad. Se hace imprescindible el uso de herramientas de monitorización para poder analizar el estado de la infraestructura y los diversos sistemas que la componen. Este conocimiento del estado de los sistemas permite reaccionar de forma eficiente ante los errores que puedan producirse. En la actualidad, rastrear un error a partir de una lista corta de posibles causas es inviable. La causa del error puede tener tantos orígenes y motivos diferentes por culpa de esta complejidad creciente de los sistemas. El error puede estar provocado por un problema en el sistema operativo, la configuración de red, el código de algún programa mal escrito, etc.

La monitorización se ha convertido en un paraguas bajo el cual se incluyen muchos significados que cambian enormemente según el contexto. En términos generales, se refiere al proceso de toma de conciencia sobre el estado de un sistema [1]. Esto puede realizarse de dos maneras: de forma proactiva o de forma reactiva. La primera implica la observación de indicadores visuales, como series temporales y paneles, y es lo que a veces los administradores entienden como monitorización. La última involucra formas automatizadas de enviar notificaciones a operadores con el objetivo de avisarles sobre un cambio grave en el estado del sistema. Usualmente es lo que se conoce como *alerta*.

Sin embargo, esta ambigüedad no termina aquí. El término monitorizar es empleado erróneamente como el simple hecho de realizar mediciones, sin involucrar necesariamente ninguna interacción humana. Es por ello que en este punto, resulta útil dar una definición más formal del concepto de monitorización.

“La monitorización es el proceso de mantener bajo supervisión la existencia y magnitud tanto de los cambios de estado como de los flujos de datos en un sistema. La monitorización trata de identificar fallos y ayudar en su posterior eliminación. Las técnicas utilizadas en la monitorización de los sistemas de la información combinan el procesamiento en tiempo real, el análisis de los datos y la obtención de estadísticas. Al conjunto de herramientas de software utilizadas para la adquisición de datos, procesado y representación se les llama sistema de monitorización.” [1]

Dentro de los numerosos sistemas que se pueden monitorizar, en este Trabajo de Fin de Grado se va a monitorizar el cortafuegos de Linux, haciendo especial énfasis en la ruta que siguen los distintos paquetes que lo atraviesan.

Los cortafuegos son piezas de software o hardware que filtran todo el tráfico de red entre un equipo, red doméstica o red corporativa y el resto de Internet. Tienen un funcionamiento similar al de un cortafuegos de un edificio. Al igual que estos protegen una parte del edificio, los cortafuegos de red garantizan que si algún problema ocurre en una lado del cortafuegos, no afecte a los equipos del otro lado. A diferencia de los cortafuegos de los edificios, los cuales protegen contra una amenaza muy específica (el fuego), los cortafuegos de red aportan protección contra muchos tipos distintos de amenazas: virus, gusanos, ataques de denegación de servicio, *hacking*, intrusiones, etc. [2]

El crecimiento de estos ataques procedentes de Internet contra nuestras redes y equipos se ha disparado en las últimas dos décadas. Informes como *2019 Data Breach Investigations* de *Verizon* [3] o *Evolution of Cybercrime* de *Trend Micro* [4], que aunque puedan estar tendenciosamente desarrollados al tratarse de empresas que prestan servicios de ciberseguridad, nos muestran claramente una realidad en la que los ciberataques son cada vez más numerosos y dañinos. Contra estos, la primera línea de defensa son los cortafuegos.

Dentro de los cortafuegos disponibles en la actualidad, destaca especialmente el del proyecto *Netfilter* [5], presente en prácticamente todas las distribuciones Linux de forma predeterminada. Su código abierto y su extrema flexibilidad hacen de este cortafuegos un candidato muy importante cuando se trata de la protección de redes domésticas o de tamaño medio. Sin embargo, este cortafuegos presenta una mayor complejidad y dificultad de administración que sus alternativas comerciales [6].

Surge aquí la motivación que fundamenta este Trabajo de Fin de Grado. El cortafuegos desarrollado por el proyecto *Netfilter* presenta un alto grado de penetración en las redes por estar integrado en el núcleo Linux. Por su gran versatilidad y costo gratuito, es ampliamente utilizado como cortafuegos para redes medianas y pequeñas. No obstante, presenta ciertas limitaciones, pues el grado de complejidad en su configuración es alto y los errores de configuración son, en ocasiones, difíciles de detectar, por lo que se convierten en difíciles de corregir. Este Trabajo de Fin de Grado subsanará esta deficiencia, con la implementación de un sistema que permita monitorizar los paquetes a su paso por el cortafuegos.

1.2. Objetivos

El presente Trabajo de Fin de Grado tiene por objetivo desarrollar un sistema de monitorización que permita visualizar de forma gráfica los paquetes que atraviesan el cortafuegos. Deberá ser capaz de monitorizar exactamente qué reglas atraviesa cada paquete. También permitirá visualizar de forma gráfica la configuración existente en el cortafuegos

en un instante de tiempo determinado. Se utilizará de forma combinada un sistema de registro de *logs*, al cual llamaremos de ahora en adelante simplemente sistema de registro, y otro de monitorización. El sistema de registro no deberá interferir con el funcionamiento del cortafuegos a analizar. También se optará por soluciones de monitorización lo más extendidas posible y de software abierto, que hagan posible su integración en sistemas ya existentes. El último objetivo es permitir determinar en qué puntos del cortafuegos se producen descartes de paquetes, así como el porqué de los mismos.

1.3. Objetivos específicos

Para cumplir con los objetivos explicados en la sección 1.2, se establecen los siguientes objetivos específicos.

- Desarrollar un sistema que permita visualizar la configuración completa del cortafuegos en un instante de tiempo determinado.
- Desarrollar un sistema que permita visualizar los paquetes que atraviesan el cortafuegos de forma gráfica y ordenada cronológicamente. No deberá perderse información del intervalo de tiempo analizado.
- Desarrollar un sistema que permita almacenar los datos de monitorización así como realizar búsquedas y filtrar los resultados.
- Desarrollar un sistema que permita el cambio en la configuración del cortafuegos mientras se esté monitorizando y que consuma una mínima cantidad de recursos del mismo
- Desarrollar un sistema que permita que el agente monitorizado y el sistema de visualización se encuentren en máquinas distintas, así como integrarse en soluciones de monitorización ya existentes

1.4. Metodología

Para implementar un sistema que cumpla con los objetivos detallados en las secciones 1.2 y 1.3 se ha seguido la línea de trabajo del método de la ingeniería. Este método se compone de cuatro fases: analizar las posibles soluciones al problema, desarrollar un sistema que cumpla con los objetivos establecidos, implementar dicho sistema, y por último, comprobar su funcionamiento [7]. Así pues, en primer lugar se analizó de forma concisa y detallada el funcionamiento del cortafuegos, para posteriormente, buscar un sistema de monitorización adecuado. Como sistema de monitorización se decidió utilizar la pila Elastic, ya que como se analizará en el capítulo 3, garantiza la suficiente flexibilidad para poder cumplir con los objetivos establecidos.

De esta forma, se divide el trabajo en tres fases, las cuales pueden realizarse de forma independiente, pero que una vez completadas funcionarán de forma integrada.

- **Fase de obtención de datos.** Esta primera fase engloba todos los aspectos relacionados con el cortafuegos monitorizado. Incluye todas las tareas de documentación y búsqueda de soluciones que permitan obtener los datos necesarios del cortafuegos.
- **Fase de transferencia y transformación de los datos**. Esta segunda fase abarca todo aquello relacionado con la transferencia desde el agente hasta nuestro sistema de monitorización de todos estos datos. Así como de las transformaciones necesarias para poder visualizarlos. Para ello, es necesario documentarse y buscar la solución óptima que cumpla todos los objetivos especificados.
- **Fase de visualización de los datos**. En esta tercera fase se abordará todo lo relacionado con la visualización final de estos datos que se han obtenido y transformado en las dos fases anteriores.

1.5. Estructura del resto de la memoria

A continuación se indica el contenido del resto de los capítulos de la memoria:

- **Capítulo 2 - Cortafuegos**: se explica que es un cortafuegos en el contexto de las redes TCP/IP, así como sus características y tipos. Se analiza también el cortafuegos del proyecto Neftilter IPTables.
- **Capítulo 3 - Elastic Stack**: de forma similar a la del capítulo 2, se describe el conjunto de herramientas que componen la pila Elastic. Se discute también la idoneidad de su uso.
- **Capítulo 4 - Sistema de monitorización**: se explica de forma detallada los pasos seguidos para desarrollar el sistema de monitorización así como su despliegue.
- **Capítulo 5 - Resultados y validación del sistema**: en este capítulo se muestra el resultado del funcionamiento del sistema completo mediante la realización de pruebas.
- **Capítulo 6 - Conclusiones y futuras líneas de trabajo**: por último, se concluye este TFG exponiendo las conclusiones obtenidas del desarrollo del sistema. Se detallan también las principales líneas de trabajo que podrían seguirse para continuar su desarrollo.

2. CORTAFUEGOS

En este capítulo se explica qué es un cortafuegos de red. Se analizará de forma general la historia de los mismos y los tipos existentes en la actualidad, para al final ahondar en el cortafuegos implicado en el presente Trabajo de Fin de Grado, el desarrollado por el proyecto Netfilter.

2.1. Origen del cortafuegos de red

El término cortafuegos, en inglés *firewall*, empezó a utilizarse a finales de la década de 1980. En esta época Internet aún era una tecnología relativamente nueva, que presentaba interconexiones de carácter limitado, pues el número de equipos que conformaban la red era bajo. La función de estos cortafuegos era realizada por los routers, encargados de separar redes entre sí.

La necesidad de usar cortafuegos surge con el cambio de concepción de Internet a finales de la década de los 80, este deja de ser una comunidad pequeña formada por usuarios que valoraban la libertad para colaborar e intercambiar información. El rápido incremento del número de incidentes y ataques tuvo una consecuencia clara: no todo el mundo era confiable. Muchos usuarios eran no fiables o maliciosos. Por lo tanto, era necesario establecer unos niveles de confianza en las conexiones de redes, en el sentido de que, por ejemplo, una organización confía en que sus propios equipos y usuarios no son maliciosos, pero no así el resto de Internet [8].

2.2. ¿Qué es un cortafuegos?

Para definir formalmente qué es un cortafuegos, en primer lugar nos remitiremos a la definición formal que hace Cisco [9]. “Un cortafuegos es un dispositivo de seguridad de red que monitoriza el tráfico entrante y saliente y decide si debe permitir o bloquear un tráfico específico en función de un conjunto de restricciones de seguridad ya definidas”.

Aunque la primera idea que se viene a la cabeza es la de que los cortafuegos son dispositivos que controlan el tráfico entre dos redes distintas, también es posible implementarlos directamente en un equipo. Es decir, mientras que en el primer caso estaríamos hablando de un cortafuegos de red, en el segundo estaríamos ante un cortafuegos de equipo. El objetivo en ambos casos es el mismo: proporcionar un método para cumplir una política de control de acceso para, en el caso de los cortafuegos de red, permitir que el tráfico lo atraviese o, en caso de los cortafuegos de equipo, que acceda al sistema protegido [6].

2.3. Tipos de cortafuegos

Los cortafuegos existentes en la actualidad pueden diferenciarse según el tipo de tecnologías subyacentes que utilizan [6]. No son excluyentes entre sí: un cortafuegos puede combinar varias tecnologías.

Cortafuegos personales

Diseñados para proteger un único equipo, pueden ser vistos como una coraza alrededor del sistema, ya sea un servidor, equipo de escritorio o cualquier dispositivo. Comúnmente estos cortafuegos personales asumen que el tráfico saliente es permitido e inspeccionan el entrante. El cortafuegos IPTables, además de cortafuegos de red, también está incluido en esta categoría [6].

Cortafuegos de filtrado de paquetes

Son dispositivos de red que filtran el tráfico basándose simplemente en las características del paquete analizado. Para permitir tráfico en ambas direcciones deben estar configurados para permitir el tráfico de retorno. No tienen capacidad de analizar el estado de las conexiones, no pueden generar con el tráfico saliente reglas entrantes para las respuestas de forma dinámica. El tráfico se bloquea a nivel de red.

Cortafuegos de traducción de direcciones de red (NAT)

En la actualizad se encuentran integrados en prácticamente todo el resto de cortafuegos. La función principal de estos cortafuegos NAT es multiplexar tráfico de una red interna y presentarlo a una red más grande como si el origen fuese una sola, o un rango de direcciones IP [10].

Cortafuegos de nivel de circuito

Estos cortafuegos funcionan en la capa de sesión del modelo OSI, y monitorizan el establecimiento de conexiones para determinar si es legítimo. La información que atraviesa un cortafuegos de este tipo es modificada para que parezca estar originada en el mismo. Por lo que también protege la red por ocultamiento. No filtran paquetes de forma individual.

Cortafuegos *proxy*

Actúan como intermediarios entre dos equipos de forma similar a los cortafuegos de nivel de circuito. Sin embargo, la interacción se controla en la capa de aplicación, forzando

a que ambos extremos se comuniquen a través del mismo.

Cortafuegos con estado

Combinan aspectos de los cortafuegos de traducción de direcciones de red, de nivel de circuito y *proxy*. Estos cortafuegos inspeccionan los paquetes basándose en sus características como los cortafuegos de filtrado de paquetes, pero también comprueban las sesiones para determinar si el tráfico es permitido. Permiten también la inspección de la capa de aplicación para determinados servicios. Casi todos los cortafuegos modernos se engloban en este grupo.

Cortafuegos transparentes

También conocidos como cortafuegos puente, son un subconjunto dentro de los cortafuegos con estado. Mientras que casi todos los cortafuegos operan en la capa de red y superiores, estos se ubican en la capa de enlace monitorizando las superiores. De esta forma, se pueden aplicar las reglas de filtrado como en un cortafuegos con estado pero de forma invisible para usuarios finales. No requieren cambios en las redes subyacentes.

Cortafuegos de nueva generación (NGFW)

Mejoran características del resto de cortafuegos, permitiendo analizar también la carga de los paquetes para así realizar descifrado de paquetes, detección de intrusiones, funciones de antivirus, etc.

El cortafuegos desarrollado por el proyecto Netfilter, IPTables, puede situarse en el grupo de cortafuegos con estado.

2.4. El cortafuegos del proyecto Netfilter, IPTables

2.4.1. Historia del proyecto Netfilter

Aunque hay contribuciones al código del proyecto anteriores, no fue hasta noviembre de 1999 en el que Marc Boucher y Rusty Russel, decidieron formar el núcleo del proyecto, con el objetivo de consolidar la comunidad [11]. Este fue el momento del nacimiento de los módulos de *iptables filter*, *nat* y *mangle*, así como de la primera herramienta, *ipnatctl*, que dotaba al cortafuegos de funciones NAT.

En primavera del año 2000, netflter/IPTables se integran por primera vez en el *kernel* Linux.

En la *Sydney Linux Expo*, James Morris se une al núcleo del proyecto. En este punto,

los esfuerzos del equipo se centraron en integrar el proyecto en la versión 2.4 del *kernel*. Este fue la tercera implementación de un cortafuegos distinto en Linux. Harald Welte, un programador del *kernel* de Linux, es añadido al proyecto.

Este fue el encargado de desarrollar el módulo de seguimiento de conexiones, así como de dotar de compatibilidad del cortafuegos para IPv6. También destacamos la creación del *target* ULOG y de la primera versión del demonio ulogd, así como de la creación de un tipo de interfaz (nfnetlink), para la comunicación entre el espacio de usuario y el *kernel*

En los años posteriores, otros muchos miembros como Martin Josefsson, Patrick McHardy, Yasuyuki Kozakai o Pablo Neira, fueron añadidos al proyecto. Este último es el jefe del mismo en la actualidad [8].

2.4.2. El cortafuegos *IPTables*

Como ha sido visto anteriormente, la llegada del *kernel* 2.4 de Linux en 2001 trajo consigo un nuevo cortafuegos, mucho más maduro. Incluía características propias de los cortafuegos con estado, con capacidades mejoradas de inspección de tráfico y de *logging*. Al ser un cortafuegos de Linux, tiene la gran ventaja de que puede ser instalado en cualquier hardware a un coste mínimo [6].

En las siguientes secciones se detalla el funcionamiento del cortafuegos, así como de su principal herramienta, IPTables, término que será usado de ahora en adelante para referirse a cortafuegos.

2.4.3. Características

El cortafuegos IPTables tiene una serie de características que se detallarán a continuación [5]:

- Filtrado de paquetes sin estado para IPv4 e IPv6.
- Filtrado de paquetes con estado para el tráfico IPv4 e IPv6.
- Todo tipo de direccionamiento de red, incluido la traducción de direcciones de red (NAT) y traducción de puertos de direcciones de red (NAPT) tanto para IPv4 como IPv6.
- Infraestructura flexible y extensible.
- Múltiples capas de interfaces de programación de aplicaciones (API) para extensiones de terceros.
- Gran cantidad de módulos.
- Soporte de múltiples interfaces de red.

Otro beneficio importante es que es de código abierto.

2.4.4. Funcionamiento

Los primeros conceptos que deben tenerse en cuenta sobre el funcionamiento del cortafuegos IPTables son los de tabla, cadena y regla. Las tablas se utilizan para proporcionar ciertos tipos de funcionalidades y están compuestas por cadenas. Las cadenas definen rutas las cuales tienen que atravesar los paquetes. Están formadas por una lista de reglas, que definen que acción tomar en paquetes que coincidan con la regla.

Los paquetes que atraviesan una cadena se comparan con una lista de reglas siguiendo un cierto orden, desde el inicio al final. Cuando una regla coincide, se aplica la acción asociada a esta. Como es difícil que una cadena cubra de forma explícita todas las posibles condiciones a no ser que se cree específicamente una regla "caza todo", estas tienen una política por defecto, que se aplica si ninguna de las reglas coincidió.

Las tablas que componen el cortafuegos IPTables son actualmente cinco, las cuales no tienen que estar todas presentes, pues dependen de la configuración del *kernel* [12].

- **La tabla Filter.** Es la tabla por defecto, proporciona la mayor parte de las funcionalidades cuando pensamos en un cortafuegos. Contiene las cadenas prefijadas INPUT (paquetes destinados a *sockets* locales), FORWARD (para paquetes que son reenviados), OUTPUT (para paquetes generados localmente).

- **La tabla NAT.** Esta tabla es consultada cuando se encuentra un paquete que crea una nueva conexión. Se utiliza para implementar las reglas de traducción de direcciones de red (NAT). Consiste en tres cadenas predefinidas: PREROUTING (para alterar paquetes en cuanto llegan), OUTPUT (para alterar paquetes generados localmente antes del enrutamiento) y POSTROUTING (para alterar paquetes justo antes de salir).

- **La tabla Mangle.** Esta tabla se usa para alteraciones de paquetes específicas.Permite tanto alterar las cabeceras IP de los paquetes de la forma que se desee, como asignar marcas a los mismos. Estas marcas no alteran el paquete; sólo afectan a la representación interna en el kernel y son usadas para procesarlos en otras reglas. Tiene dos cadenas predefinidas: PREROUTING (para alterar paquetes entrantes antes del enrutado), OUTPUT (para alterar paquetes generados localmente antes del enrutado), INPUT (para alterar paquetes con destino local), FORWARD (para alterar paquetes que son reenviados), y POSTROUTING (para alterar paquetes antes de ser enviados).

- **La tabla Raw.** Esta tabla es utilizada principalmente para configurar exenciones al seguimiento de conexiones. Es comprobada antes que cualquier otra tabla y del seguimiento de conexiones. Tiene dos cadenas predefinidas: PREROUTING (para los paquetes entrantes por cualquier interfaz de red) y OUTPUT (para los paquetes generados por los procesos locales).

- **La tabla Security** Esta tabla se usa para establecer permisos MAC (*Mandatory Access Control*), como SELinux, a los paquetes. Se comprueba después de la tabla *Filter* permitiendo que estas reglas DAC (*Discretionary Access Control*) sean comprobadas en primer lugar. Contiene las cadenas prefijadas INPUT (paquetes destinados a *sockets* locales), FORWARD (para paquetes que son reenviados), OUTPUT (para paquetes generados localmente).

Analizando las cadenas de forma independiente, hay tres posibles rutas en función del tipo de tráfico:

- **Paquetes entrantes con destino el sistema local.**
  Siguen la ruta PREROUTING → INPUT
- **Paquetes entrantes con destino otro *host*.**
  Siguen la ruta PREROUTING → FORWARD → POSTROUTING
- **Paquetes salientes con destino otro *host*.**
  Siguen la ruta OUTPUT → POSTROUTING

La figura 2.1 condensa lo explicado en esta sección del cortafuegos IPTables y permite ver cualquier posible ruta que pueda seguir un paquete en el interior del cortafuegos.

Además de estas cadenas predefinidas, existe la posibilidad de crear otras definidas por el usuario.

2.4.5. Reglas IPTables

Las reglas del cortafuegos IPTables tienen dos partes fundamentales: el *match*, o condición que debe cumplir el paquete y el *target* o acción de la regla, que se realiza cuando el paquete coincide con el *match*.

Match

Esta parte de la regla especifica qué condiciones debe cumplir el paquete para aplicar la acción. La cantidad de opciones es enorme. Además, puede expandirse con extensiones. Algunos de los *matches* más comunes se detallan en la tabla 2.1 [14] [12].

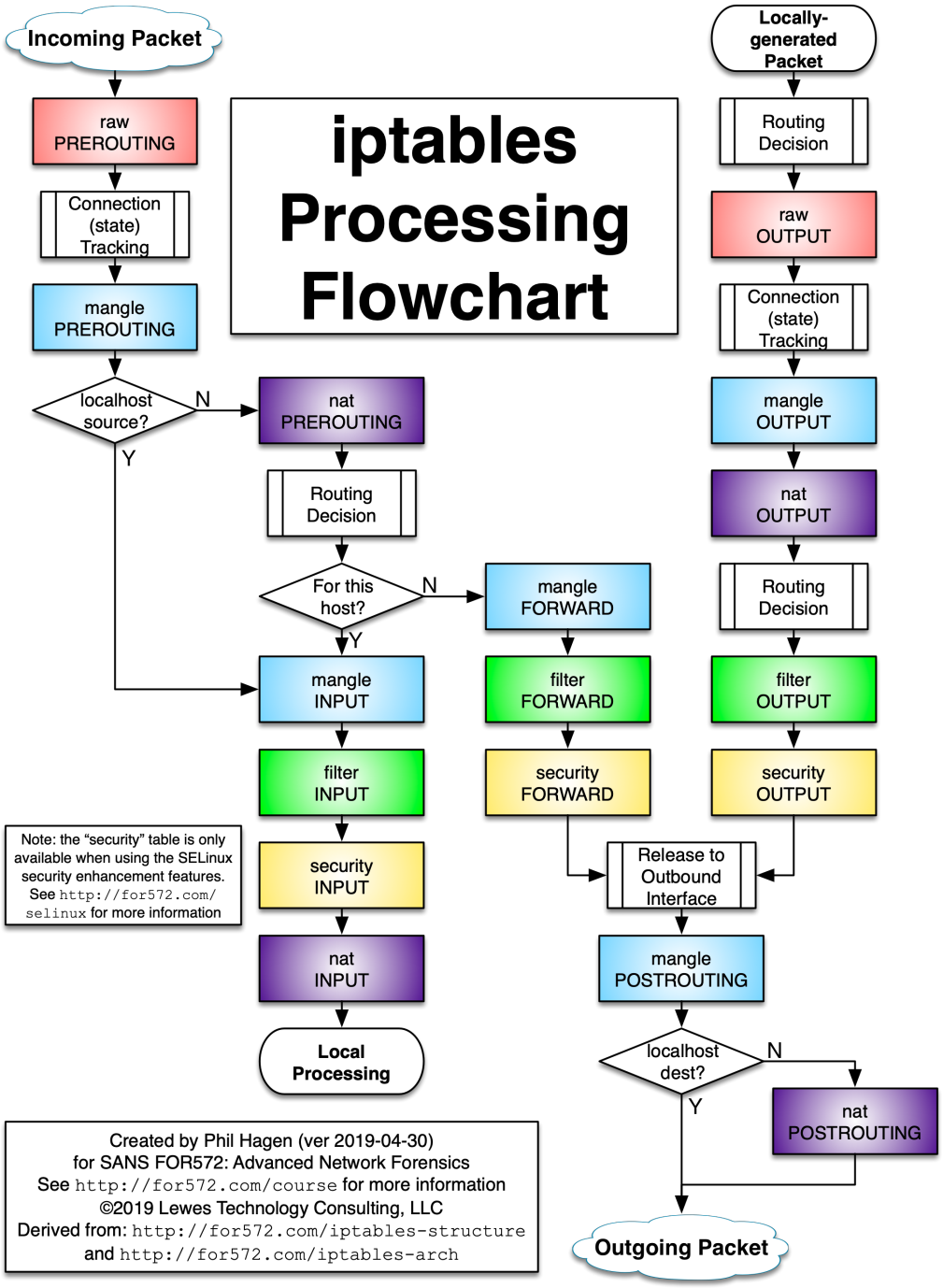


Fig. 2.1. Diagrama de flujo del cortafuegos IPTables [13]

TABLA 2.1. IPTABLES *MATCHES*

Match	Descripción
–protocol protocol	Protocolo del paquete a comprobar. Puede ser uno de los siguientes: tcp, udp, udplite, icmp, icmpv6, esp, ah, sctp, mh ... (o el número del protocolo IP), o all para todos.
–source address[/mask][,...]	Origen del paquete. Puede especificarse un nombre de red, un *hostname* o una dirección de red con su correspondiente máscara o una IP plana. Permite especificar múltiples direcciones, pero se expanden en múltiples reglas.
–destination address[/mask][,...]	Idéntico a *source* pero con la dirección de destino.
–in-interface name	Nombre de la interfaz por la que el paquete ha sido recibido. Solo puede ser usada en las cadenas INPUT, FORWARD y PREROUTING.
–out-interface name	Nombre de la interfaz por la que el paquete va a ser enviado. solo puede usarse en las cadenas FORWARD, OUTPUT y POSTROUTING.
–fragment	Para fragmentos de un paquete IP, a excepción del primero.

Target

Una vez que se comprueba si un paquete coincide con el *match* o los *matches*, la segunda parte de la regla especifica la acción a realizar. Puede ser uno de los valores especiales ACCEPT, DROP o RETURN, así como alguno de los *targets* incluidos con las extensiones de IPTables o el nombre de una cadena definida por el usuario.

Distinguimos dos tipos de *target*:

- *Target* terminal: no se siguen comprobando el resto de reglas de la cadena después de realizar la acción.
- *Target* no terminal: se realiza la acción pero se siguen comprobando el resto de reglas de la cadena.

Algunos de los *target* más comunes se muestran en la tabla 2.2.

TABLA 2.2. IPTABLES *TARGETS*

Target	Descripción
ACCEPT	Se permite al paquete atravesar la cadena. *Target* terminal.
DROP	Se descarta el paquete. *Target* terminal.
RETURN	En una cadena definida por el usuario, se deja de recorrer y se vuelve a la que la llamó. En una cadena predefinida, se aplica la política de la cadena al paquete. *Target* terminal.
REJECT	Se descarta el paquete, pero se envía un mensaje ICMP como respuesta. *Target* terminal.
LOG	Se crea una línea de registro del paquete desde el kernel. *Target* no terminal.
SNAT	Sólo puede ser utilizado en la tabla nat y la cadena POSTROUTING. Nos permiten realizar NAT en origen, alterando la IP de origen y/o el puerto, los cuales son recordados durante toda la conexión. *Target* no terminal.
MASQUERADE	Similar a SNAT pero como IP de origen se fija la de la interfaz de salida. *Target* no terminal.
DNAT	Solo puede ser utilizado en la tabla nat y en la cadena PREROUTING. Nos permite realizar NAT en destino, alternado la IP y/o puertos de destino de la conexión. *Target* no terminal.

3. ELASTIC STACK

En este capítulo se introducirá el grupo de herramientas que conforman Elastic Stack, para finalmente comprobar cómo se adaptan perfectamente a los objetivos establecidos en la sección 1.2.

3.1. ¿Qué es Elastic Stack?

"Built on an open source foundation, the Elastic Stack lets you reliably and securely take data from any source, in any format, and search, analyze, and visualize it in real time." [15]

Esta frase sintetiza de forma muy acertada la funcionalidad del conjunto de las cuatro herramientas que conforman el proyecto: Beats, Logstash, Elasticsearch y Kibana, las cuales se describen en posteriores secciones. La pila Elastic, así será llamada de ahora en adelante, aporta un marco ideal en el que desarrollar el sistema de monitorización [16]:

- **Beats**: son clientes ligeros que permiten enviar datos de una máquina a otra con un consumo mínimo de recursos.
- **Logstash**: herramienta que permite transformar datos procedentes de múltiples fuentes y almacenarlos en múltiples bases de datos.
- **Elasticsearch**: es una base de datos no relacional [17] que permite la realización de búsquedas avanzadas, soporta replicación y un alto nivel de escalabilidad.
- **Kibana**: herramienta que permite visualizar y buscar información almacenada en Elasticsearch.

Además, la pila Elastic es proyecto de Código Abierto [18], en el cual la mayor parte del código se encuentra bajo la licencia Apache 2.0 [19] y la licencia Elastic [20].

3.2. Origen de la pila Elastic

Los orígenes de este proyecto se remontan a principios de los años 2000, en Londres, donde Shay Banon comenzó la creación de una herramienta de búsquedas de recetas. La primera versión tuvo el nombre de Compass, la segunda fue Elasticsearch, de la cual publicó su código [21].
Con el crecimiento de la comunidad de usuarios, Shay Banon, junto con Steven Schuurman, Uri Boness y Simon Willnauer, fundaron la compañía Elasticsearch Inc. Durante esta etapa surgen dos nuevos proyectos de código abierto. Jordan Sissel desarrolla la herramienta Logstash, una herramienta de ingesta de *logs* que permitía conectarse a la base

de datos que el usuario eligiese, entre ellas, Elasticsearch. Rashid Khan a su vez crea la UI (interfaz de usuario) Kibana. Ambos decidieron formar equipo y surgió el ELK Stack, Elasticsearch, Logstash, y Kibana Stack.
A lo largo de los siguientes años el proyecto continuó ampliándose con nuevos *plugins* y extensiones. En el año 2015 aparece Packetbeat, un cliente ligero que permitía el envío de datos de red a Elasticsearch, el primero de la familia de Beats.

En octubre de 2015, con el lanzamiento de la versión 2.0 del ELK Stack, se produce un cambio importante en la forma de integración de estas herramientas. Las actualizaciones son lanzadas de forma simultánea, y son completamente compatibles entre sí. Un mes más tarde, Beats 1.0 es lanzado. Un año después, es lanzada la versión 5.0, la cual pasa a llamarse Elastic Stack, y en la que continua mejorándose la integración de las herramientas. Se agrupan las funcionalidades no disponibles bajo la licencia Apache en el X-Pack (seguridad, monitorización, alertas, *machine learning*).

En los años posteriores las versiones 6.0 y 7.0 fueron publicadas, siendo esta última la presente en la actualidad. El X-Pack fue incluido también como código abierto bajo la licencia Elastic [20] y Elastic Cloud conformó la oferta de SaaS (software como servicio) [21] [22].

3.3. Logstash

Logstash es una herramienta *Open Source* de recolección y procesamiento de datos en tiempo real basada en tuberías. Puede unificar datos de multitud de fuentes y normalizarlos para su envío a múltiples destinos diferentes. Esto permite hacer un análisis y visualización posterior de los datos.

Originalmente trajo innovación a la recolección de *logs*. Sus capacidades pueden extenderse a prácticamente cualquier caso de uso. Cualquier evento puede ser transformado y enriquecido con la multitud de *plugins* de entrada, filtrado y salida existentes, y con los *codecs* existentes que simplifican el proceso de entrada de datos [23].

3.3.1. Principales funcionalidades de Logstash

- Herramienta de procesamiento de datos con capacidad de escalado horizontal y una fuerte integración con Elasticsearch, Kibana y Beats.
- Arquitectura de tuberías. Permite combinar y mezclar diferentes entradas, filtros y salidas.
- Ecosistema de *plugins* extenso. Con facilidad para desarrollar nuevos, lo que dota a la herramienta de una gran flexibilidad

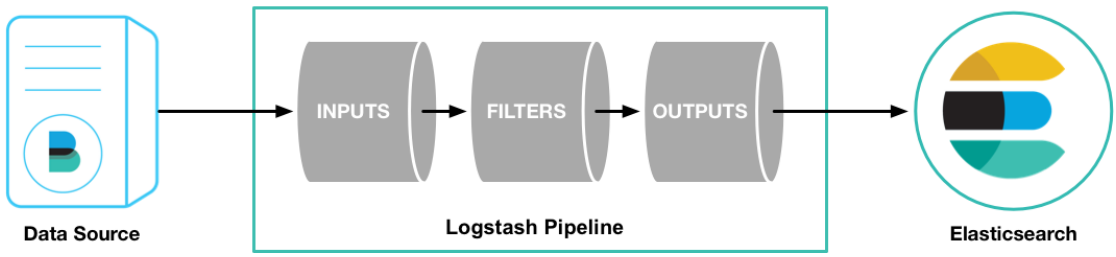


Fig. 3.1. Tubería de Logstash [24].

3.3.2. Fuentes de datos para Logstash

Aunque originalmente Logstash solo fue diseñado para recopilar datos procedentes de ficheros de *logs*, actualmente cuenta con muchas otras fuentes de datos.

- ***Logs* y métricas**
  - Cualquier tipo de *logs*: *logs* web de Apache, *logs* de aplicaciones como log4j, syslog, *logs* de red y cortafuegos, etc.
  - Soporta el reenvío de ficheros de *logs* gracias a Filebeat.
  - Recolección de métricas de Ganglia, collectd, NetFlow, JMX y multitud de otras infraestructuras y aplicaciones sobre TCP y UDP.
- **La Web**
  - Permite transformar peticiones HTTP en eventos, Twitter, GitHub, HipChar, JIRA, etc.
  - Generar eventos a partir de respuesta de peticiones HTTP, perfecto para escenarios en los que se prefiere sondear.
- **Datos almacenados y flujos**
  Procesado de datos ya existentes en bases de datos o en colas de datos como Kafka, RabbitMQ, etc.
- **Sensores e IoT (Internet de las Cosas)**
  Una amplia gama de otros tipos de datos diferentes. Algunos ejemplos se detallan en la sección 3.3.3.

3.3.3. Funcionamiento de una tubería de Logstash

El concepto de tubería implementado en Logstash permite cualquier combinación de *plugins* de entrada, filtrado y salida [24]. En la figura 3.1 puede verse de forma gráfica el concepto de tubería.

Plugins de entrada

Los datos se encuentran dispersos en numerosos sistemas y formatos. Logstash nos permite elegir entre una gran variedad de entradas que extraen eventos de fuentes comunes. En la tabla 3.1 se encuentran listados algunos de los *plugins* más comunes de entrada y su descripción [25].

TABLA 3.1. *PLUGINS* DE ENTRADA LOGSTASH

Plugin	Descripción
beats	Recibe eventos desde el *framework* de Elastic Beats.
elasticsearch	Lee resultados de consultas a un *cluster* de Elasticsearch.
exec	Captura la salida de un comando de *shell* en forma de evento.
file	Transmite eventos desde ficheros.
http	Recibe eventos sobre HTTP o HTTPS.
imap	Lee correos de un servidor IMAP.
kafka	Lee eventos de un servidor Kafka.
snmp	Consulta a dispositivos de red mediante el protocolo SNMP y crea eventos.
snmptrap	Crea eventos en función de Traps SNMP recibidos.
stdin	Lee eventos de la entrada estándar del sistema.
syslog	Lee mensajes de syslog como eventos.
tcp	Lee eventos de un *socket* TCP.
udp	Lee eventos por UDP.
unix	Lee eventos de un *socket* UNIX.

Plugins de filtrado

Una vez recibidos los datos, los filtros de Logstash analizan cada evento, identificando campos y asignándoles un nombre para darles un formato común que permita su análisis [24].

Hay numerosos *plugins* de filtrado que nos permiten transformar los eventos sin importar la complejidad de los mismos [26]. Los más comunes pueden verse en la tabla 3.2.

TABLA 3.2. *PLUGINS* DE FILTRADO LOGSTASH

Plugin	Descripción
aggregate	Agrupa información de varios eventos originados por la misma tarea.
alter	Realiza alteraciones generales de los campos que el filtro mutate no puede.
bytes	Transforma representaciones con forma de cadena de tamaños de almacenamiento del equipo, como “123 MB” o “5.6gb”, en su valor numérico en bytes.
clone	Duplica el evento.
date	Parsea los campos de fecha para ser usados como *timestamp* (marca temporal) del evento.
dissect	Extrae datos de eventos no estructurados en campos utilizando delimitadores.
dns	Realiza búsquedas DNS tanto directas como inversas
drop	Descarta todos los eventos.
elasticsearch	Copia campos de eventos de *logs* anteriores en Elasticsearch a eventos actuales.
fingerprint	“Firma” (*fingerprints*) campos reemplazando valores por un *hash*.
geoip	Añade información geográfica de una dirección IP.
grok	Parsea los datos de eventos no estructurados en campos.
json	Parsea eventos con formato JSON.
kv	Parsea datos estructurados como parejas clave-valor.
mutate	Realiza transformaciones en campos.
ruby	Ejecuta código Ruby arbitrario.
urldecode	Parsea URLs.
xml	Parsea datos con formato XML en campos.
uuid	Añade identificadores únicos universales a eventos.

Plugins de salida

Una vez se han enriquecido y transformado los datos, Logstash ofrece gran cantidad de opciones para enviar datos [24] [27].

Alguno de los *plugins* de salida se describen en la tabla 3.3.

TABLA 3.3. *PLUGINS* DE SALIDA LOGSTASH

Plugin	Descripción
csv	Escribe eventos en el disco con un formato de delimitado concreto.
elasticsearch	Almacena *logs* en Elasticsearch.
email	Envía un *email* a una dirección específica.
exec	Ejecuta un comando en el sistema.
file	Escribe los eventos en archivos.
http	Envía eventos a nodos HTTP o HTTPS.
kafka	Envía eventos a un servidor Kafka.
mongodb	Escribe los eventos en MongoDB.
nagios	Envía resultados de chequeos pasivos a Nagios.
pipe	Tubería hacia la entrada estándar de otro programa.
stdout	Escribe los eventos en la salida estándar.
syslog	Envía los eventos a un servidor syslog.
tcp	Envía los eventos a través de un *socket* TCP.
udp	Envía los eventos a través de UDP.
zabbix	Envía los eventos a un servidor Zabbix.

3.4. Elasticsearch

Elasticsearch es el motor de búsqueda y análisis distribuido situado en el centro de la pila Elastic. Logstash y Beats facilitan la recopilación, agregación y enriquecimiento de datos con su posterior almacenamiento en Elasticsearch. Kibana por su parte permite explorar, visualizar y compartir de manera interactiva información extraída de estos datos. En Elasticsearch es donde se produce el proceso de indexación, búsqueda y análisis.

Elastisearch proporciona búsquedas y análisis en tiempo real para todo tipo de datos. Ya sea texto estructurado o no estructurado, datos numéricos o geoespaciales, se almacenan e indexan de forma eficiente para que sea posible realizar búsquedas rápidas. También es posible ir mas allá de la simple recuperación de datos y realizar agregaciones para descubrir tendencias y patrones en los datos. Si el volumen de datos y búsquedas crece, la naturaleza distribuida del motor Elasticsearch permite un escalado simultáneo [28].

Conceptos básicos

Hay una serie de conceptos que resulta fundamental entender sobre Elasticsearch [29].

- **Tiempo real**
  Elasticsearch es una plataforma de búsquedas en tiempo real, lo que significa que

hay una ligera latencia de aproximadamente un segundo desde el momento que un documento es indexado hasta que está disponible para búsquedas.

- ***Cluster***
  Un *cluster* es un grupo de uno o más nodos (servidores) que, en conjunto, conservan todos los datos y proporcionan indexado y capacidades de búsqueda a través de todos ellos. Los *clusters* se identifican con un nombre único. Es perfectamente válido tener un *cluster* formado por un único nodo.

- **Nodo**
  Un nodo es un servidor que forma parte de un *cluster*, almacena datos y participa en las tareas de búsqueda e indexado del *cluster*. Como con los *clusters*, un nodo es identificado por su nombre. Cada nodo puede ser configurado para que se una a un *cluster* determinado. En un *cluster* puede haber todos los nodos que sean necesarios.

- **Índice**
  Un índice es una colección de documentos que comparten características similares. Cada índice se identifica con un nombre que es usado para referirse a él en operaciones de indexación, búsqueda, actualización y eliminación en los documentos que contiene. En un *cluster* se pueden definir tantos índices como se necesite.

- **Documento**
  Un documento es la unidad básica de información que puede ser indexada. Estos documentos están en formato JSON (JavaScript *Object Notation*) [30], un formato de texto sencillo para el intercambio de datos. Dentro de un índice pueden almacenarse tantos documentos como se desee.

- **Fragmentos (*shards*) y réplicas**
  Un índice puede en potencia almacenar una gran cantidad de datos, lo que puede llegar a exceder los límites del hardware de un nodo. Para solventar este problema, Elasticsearch brinda la capacidad de subdividir un índice en múltiples partes llamadas fragmentos. Cada fragmento es en sí mismo un "índice" completamente funcional e independiente que puede estar alojado en cualquier nodo del *cluster*. El *sharding* (fragmentación) es importante por dos principales motivos:

  - Permite dividir y escalar horizontalmente el volumen de contenido.
  - Permite distribuir y paralelizar operaciones entre distintos *shards* (que pueden estar en diferentes nodos), aumentando así el rendimiento.

  Elasticsearch gestiona de forma transparente al usuario la mecánica de cómo se distribuyen los fragmentos, así como la forma en que se agregan los documentos procedentes de búsquedas.

  Para prevenir el riesgo de fallos, es importante tener un mecanismo de conmutación en caso de que un fragmento o nodo se desconecte o desaparezca por algún motivo.

Para este fin, Elasticsearch permite hacer una o más copias de los *shards* de los índices. Esto es importante por dos motivos principalmente:

- Proporciona alta disponibilidad en caso de que un nodo/fragmento falle, para lo que es necesario no tener alojadas las réplicas de los fragmentos en el mismo nodo que el fragmento original.
- Permite escalar el rendimiento de las búsquedas, ya que estas se pueden realizar en todas las réplicas de forma paralela.

Como resumen, un índice puede dividirse en varios fragmentos, además de poder replicarse cero o más veces. Cada réplica tendrá fragmentos primarios y fragmentos réplica.

El número de fragmentos y réplicas se define en el momento que se crea el índice, aunque pueden ser modificados una vez creados. De forma predeterminada un índice en Elasticsearch tiene asignado un fragmento primario y una réplica. Por lo que si hay dos nodos en el *cluster*, uno alojará el fragmento primario y el otro el fragmento réplica.

3.4.1. Elasticsearch como base de datos

Elasticsearch es una base de datos documental distribuida [31]. En lugar de almacenar la información como filas de columnas de datos, almacena estructuras de datos complejas en formato JSON [30]. Si un *cluster* tiene múltiples nodos, los documentos almacenados se distribuirán a través del mismo, manteniéndose accesibles desde cualquiera de los nodos [32].

Cuando un documento es almacenado, es indexado en tiempo real y, en aproximadamente un segundo, se encuentra disponible para realizar búsquedas. Elasticsearch utiliza una estructura de datos conocida como de índice invertido, que admite búsquedas de texto muy rápidas. En un índice invertido, se lista cada palabra única que aparece en cualquier documento y se identifican todos los documentos en los que aparece.

Un índice puede considerarse como una colección optimizada de documentos, en la que cada documento es una colección de campos, que son los pares clave-valor que contienen los datos. Por defecto, Elasticsearch indexa todos los datos de todos los campos, y cada campo indexado tiene su propia estructura de datos optimizada. Por ejemplo, los campos de texto se indexan con índices invertidos mientras que los numéricos y los de geolocalización se almacenan en árboles BKD [33]. Esta capacidad de usar estructuras de datos por campo hace que las búsquedas en Elasticsearch sean más rápidas.

Elasticsearch también tiene la capacidad de indexar documentos sin haber especificado previamente los distintos tipos de campos que puede haber. Si el mapeo dinámico de Elasticsearch se encuentra activado, este automáticamente detecta y añade nuevos campos al índice. De forma predeterminada Elasticsearch también es capaz de detectar y asignar

a booleanos, números de punto flotante y enteros, fechas y cadenas de texto los tipos de datos apropiados.
Sin embargo, también pueden definirse reglas para controlar la asignación dinámica, o explícitamente definir asignaciones para controlar totalmente como se almacenan e indexan los campos. Definir estas asignaciones permite:

- Distinguir entre campos que contienen cadenas de textos completos o simplemente un valor exacto.
- Realizar un análisis de texto específico del idioma.
- Optimizar campos para coincidencias parciales en las búsquedas.
- Usar formatos de fecha personalizados.
- Utilizar distintos tipos de datos que no pueden ser detectados automáticamente.

3.4.2. Elasticsearch como motor de búsqueda y análisis

Aunque Elasticsearch puede usarse simplemente para almacenar y recuperar documentos y sus datos, también proporciona un conjunto de aplicaciones de búsqueda creadas sobre el motor de Apache Lucene [34].

Elasticsearch proporciona una serie de interfaces de programación de aplicaciones (API) de tipo REST [35], que permiten manejar el *cluster* e indexar y buscar datos. Para desarrollar aplicaciones hay disponibles clientes oficiales en los siguientes lenguajes de programación: Java, JavaScript, Go, .NET, PHP, Perl, Python or Ruby [36]. Y de forma no oficial, desarrollados por la comunidad: B4J, C++, Clojure, ColdFusion (CFML), Erlang, Go, Haskell, Java, JavaScript, Kotlin, Lua, .NET, Perl, PHP, Python, R, Ruby, Rust, Scala, Smalltalk, Vert.x [37].

Búsqueda

Las API REST de Elasticsearch soportan tanto consultas estructuradas, como de texto, o consultas más complejas combinando ambas. Las consultas estructuradas son similares a las que pueden construirse en SQL. Por ejemplo, puedes buscarse en los campos de *genero* y *edad* en un índice de *empleados*, y ordenar las coincidencias por el campo *fecha_contratacion*. Por otro lado, las consultas de texto buscan todos los documentos que coinciden con la cadena de texto, devolviendo las coincidencias ordenadas por relevancia (cómo de bien coinciden con los términos consultados). No solo se pueden buscar términos individuales, también se puede realizar búsquedas de frases, por similitud o de prefijos, así como obtener sugerencias de autocompletado.

Para utilizar estas características de búsqueda, pueden utilizarse dos tipos de lenguajes de consultas:

- Utilizando el lenguaje de consulta de Elasticsearch basado en el estilo JSON (consultas DSL) [38].
- Utilizando consultas en formato SQL, permitiendo realizar búsquedas y agregaciones de forma nativa en Elasticsearch [39].

Análisis

Las agregaciones de Elasticsearch permiten construir complejos informes de los datos para obtener información sobre métricas, patrones o tendencias. Estas aprovechan las mismas estructuras de datos que las búsquedas, por lo pueden realizarse en tiempo casi real. Estas agregaciones pueden realizarse de forma simultánea a las búsquedas, de modo que con una sola consulta puede realizarse una búsqueda de documentos, filtrado de resultados y análisis. De esta forma solo se analizan los resultados coincidentes con los parámetros de la búsqueda.

También se encuentran disponibles funciones de aprendizaje automático (*machine learnig*), que permiten la detección de patrones anómalos en los datos [40].

3.5. Kibana

Kibana es una plataforma de análisis y visualización de código abierto diseñada para trabajar con Elasticsearch. Kibana permite ver, buscar e interactuar con los datos almacenados en los índices de Elasticsearch. De forma sencilla permite realizar búsquedas de datos avanzadas y visualizarlos en una variedad de gráficos, tablas y mapas.

Kibana facilita la interpretación de grandes volúmenes de datos. Con una interfaz accesible vía web, permite crear y compartir tableros (*dashboards*) que muestran los datos de Elasticsearch en tiempo real. No se necesita programar ni utilizar infraestructura adicional para utilizar Kibana [41].

Conexión de Kibana con Elasticsearch

Antes de comenzar a utilizar Kibana, es necesario configurar qué índices de Elasticsearch se van a explorar. Para ello se definen los *index patterns*, que pueden coincidir con uno o más índices de Elasticsearch [42].

Al acceder a la aplicación web Kibana, se dispone de distintas secciones que ofrecen multitud de funcionalidades diferentes.

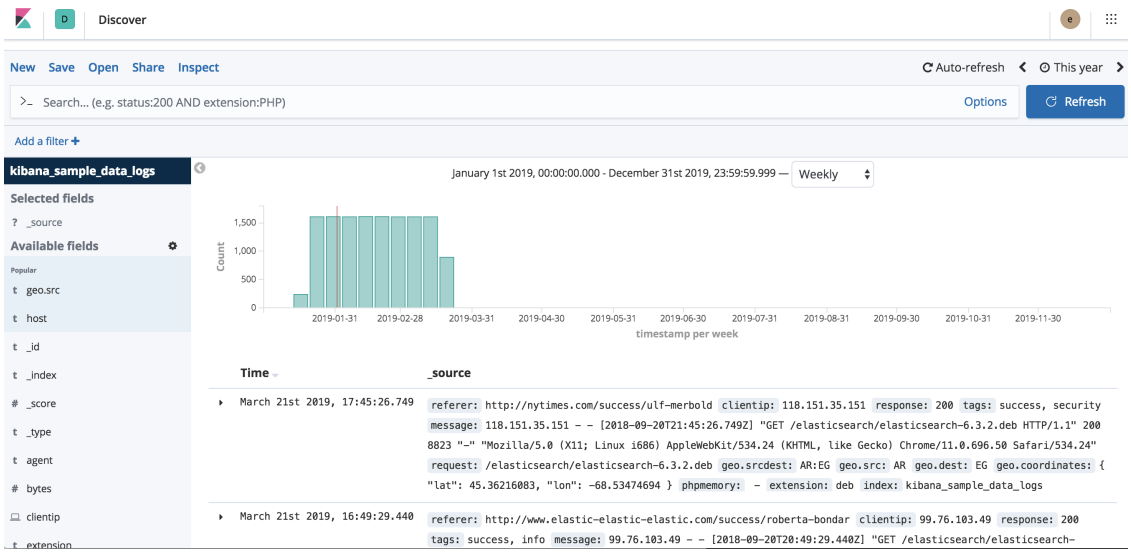


Fig. 3.2. Ejemplo de la sección *Discover* [43].

3.5.1. *Discover*

La sección Descubrir (*Discover*) permite explorar los datos con diversas funciones disponibles en Kibana. Permite acceder a todos documentos de cualquier *index pattern*. Pueden realizarse búsquedas, filtrarlas y ver los datos de los documentos. También puede verse el número de documentos que coinciden con una consulta y obtener estadísticas del valor del campo. Si está configurado el campo de tiempo en el *index pattern*, se muestra un histograma con la distribución de los documentos en la parte superior de la página [43]. Puede verse una captura de pantalla de esta sección en la figura 3.2.

3.5.2. *Visualize*

Esta sección permite la creación de visualizaciones de los datos indexados por Elasticsearch, que serán utilizadas en los *dashboards*. Las visualizaciones de Kibana están basadas en las consultas de Elasticsearch, usando agregaciones para extraer y procesar los datos. Pueden crearse multitud de visualizaciones distintas [44] [45] como puede verse de forma gráfica en la figura 3.4 y de forma detallada en la tabla 3.4.

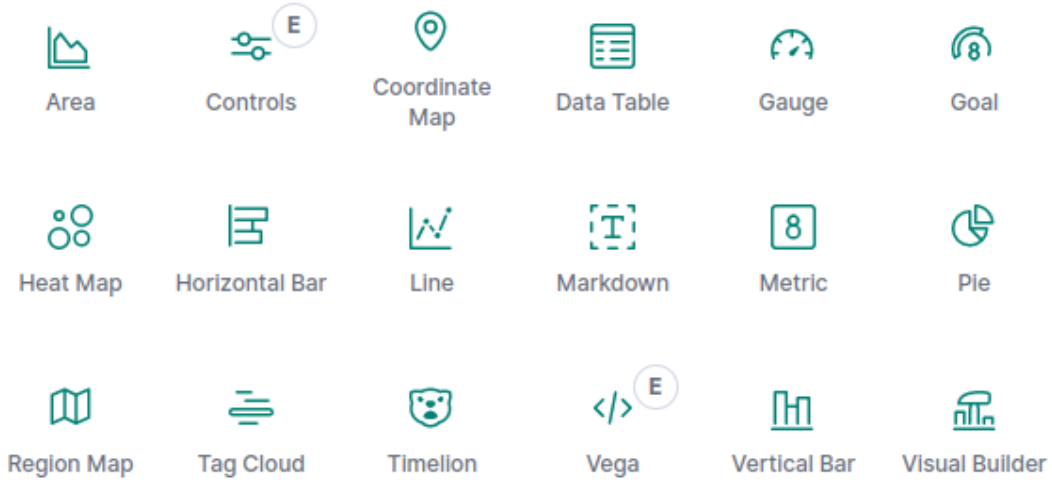


Fig. 3.3. Distintos tipos de visualizaciones disponibles en Kibana 7.0 [45].

TABLA 3.4. DISTINTOS TIPOS DE VISUALIZACIONES EN KIBANA 7.0.

Nombre visualización	Descripción
Gráficos de líneas, áreas y barras	Compara diferentes series de datos en tablas X/Y.
Mapas de calor	Sombrea células dentro de una matriz.
Gráfico circular	Muestra la contribución de cada fuente a un total.
Tabla de datos	Muestra los datos en bruto de una agregación compuesta.
Métrica	Muestra un solo número.
Meta y progreso	Muestra un indicador.
Mapa de coordenadas	Asocia los resultados de un agregación con ubicaciones geográficas.
Mapa de regiones	Mapas temáticos donde la intensidad del color depende del valor de los datos.
Timelion	Procesa y combina datos de múltiples series temporales.
Constructor visual de series temporales	Visualiza datos de series temporales utilizando múltiples agregaciones.
Controles	Permite añadir entradas interactivas a los *dashboards*.
Widget en formato *Markdown*	Muestra información o instrucciones utilizando el lenguaje Markdown [46].
Nube de etiquetas	Muestra nubes de palabras en las que el tamaño de cada una depende de su importancia.
Gráfico Vega	Soporta gráficos definidos por el usuario en formato Vega [47], con fuentes externas, imágenes y permitiendo interacción.

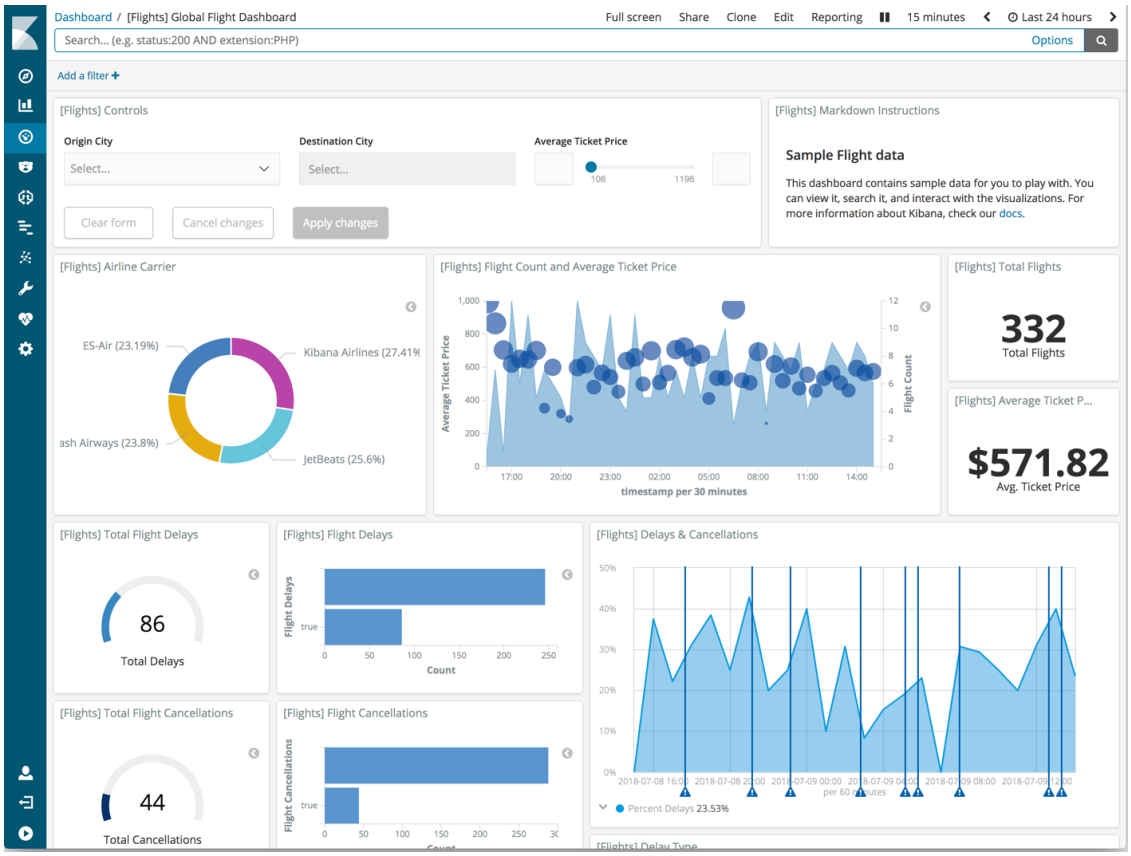


Fig. 3.4. Ejemplo de un *dashboard* [48].

3.5.3. *Dashboard*

Los *dashboards* de Kibana muestran una colección de visualizaciones y búsquedas, dotando de una total libertad para organizarlo, cambiar de tamaño, editar el contenido y compartirlo [48]. Puede verse una captura de pantalla de un *dashboards* en la figura 3.4.

3.5.4. *Canvas*

Canvas es un aplicación que permite combinar datos con colores, formas y texto. Permite construir visualizaciones dinámicas multipágina visualmente muy superiores y de mayor complejidad que los *dashboards* [49], como puede verse en la figura 3.5. Se dispone de una colección muy completa de funciones para programar en *Canvas* [50].

3.5.5. Otras secciones

Muchas otras aplicaciones están disponibles en Kibana: *Machine Learning*, *Maps*, *Infraestructure*, *Logs*, *Elastic Application Performance Monitoring (APM)*, *Uptime*, *Dev Tools*, etc. [51]

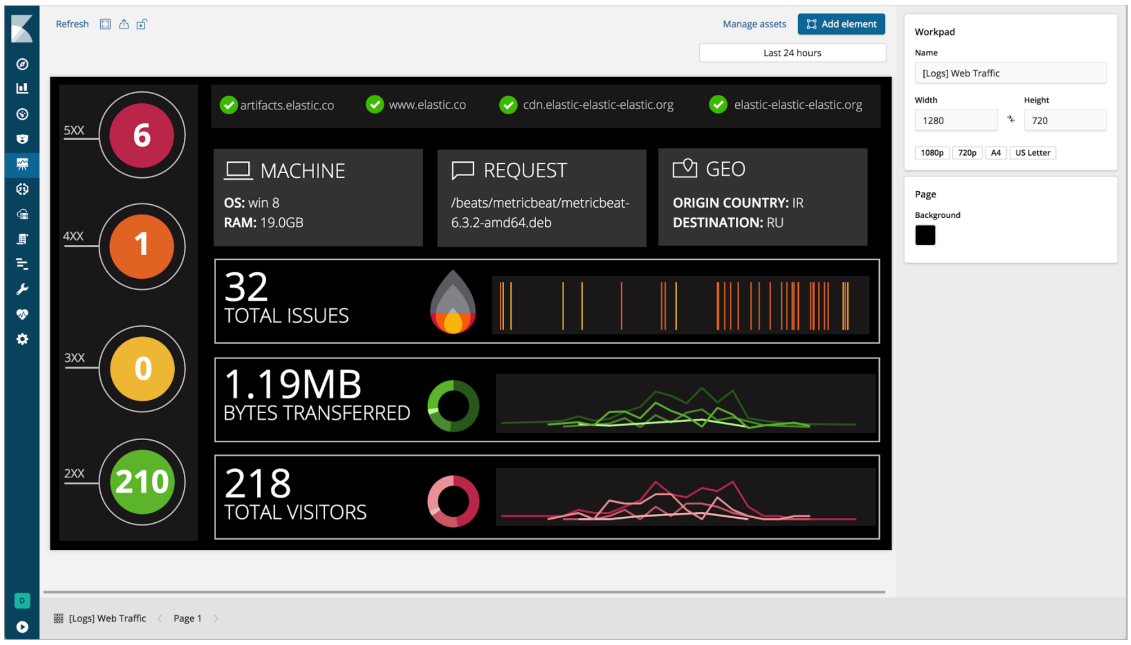


Fig. 3.5. Ejemplo de un *Canvas* [49].

3.6. Beats

Los Beats son clientes de código abierto que se instalan en los agentes para enviar datos directamente a Elasticsearch. También permiten el envío a través de Logstash para poder procesar y enriquecer los datos antes de visualizarlos en Kibana [52]. Todos los beats están programados sobre la biblioteca `libbeat`, existen multitud de beats desarrollados por la comunidad además de los oficiales. Puede verse una abstracción gráfica del funcionamiento de Beats en la figura 3.6.

3.6.1. Beats oficiales

Los Beats oficiales en la versión 7.0 de la pila Elastic son:

Auditbeat

Auditbeat permite monitorizar la actividad y los procesos de los usuarios, así como de otros eventos sin utilizar auditd [53]. Auditbeat se comunica directamente con el *framework* de auditoría de Linux, usando las reglas ya existentes de auditoría, y envía los datos a la pila Elastic [54].

Filebeat

Filebeat permite, de forma ligera, reenviar y centralizar *logs* y ficheros. Dispone de multitud de módulos que simplifican la recopilación, análisis y la visualización de forma-

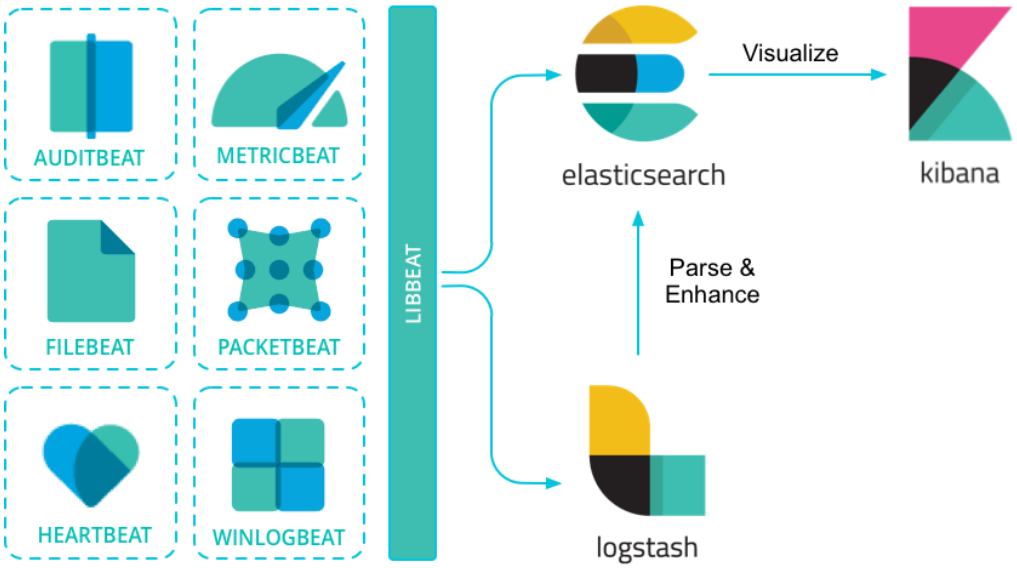


Fig. 3.6. Funcionamiento de Beats [52].

tos de *logs* comunes (auditd, Apache, NGINX, System, MySQL...) de forma centralizada. También dispone de un protocolo *backpressure-sensitive* que permite reducir la velocidad de envío de datos en caso de que Logstash o Elasticsearch se encuentren congestionados o incluso caídos. Una vez que se soluciona el problema, Filebeat recupera el ritmo de transmisión original [55].

Functionbeat

Desplegado como una función en plataformas FaaS (*Functions as a Service*) permite recopilar, enviar y monitorizar datos de servicios en la nube [56].

Heartbeat

Permite supervisar la disponibilidad de servicios mediante sondeos activos, ya sean del mismo *host* o a través de Internet. Soporta sondeos via ICMP, TCP y HTTPS, con soporte para TLS, autentificación y *proxys* [57].

Metricbeat

Metricbeat permite monitorizar sistemas Windows, Linux y Mac, desde el uso de la CPU del sistema, la memoria, el sistema de ficheros, la lectura y escritura del disco... hasta cualquier proceso ejecutándose en el sistema [58]. Dispone de multitud de módulos para monitorizar servicios como Apache, Jolokia, NGINX, MongoDB, MySQL, PostgreSQL

o Prometheus, así como la posibilidad de desarrollar nuevos módulos [59].

Packetbeat

Packetbeat es un monitorizador de paquetes de red que permite analizar el tráfico en tiempo real y enviarlo a Logstash o Elasticsearch. Permite ver los protocolos de red, los errores la latencia de aplicaciones, tiempos de espera, así como tendencias, patrones, etc [60].

Winlogbeat

Winlogbeat permite transferir *logs* de eventos de Windows de forma ligera a Elasticsearch o Logstash. Puede ser configurado para leer de cualquier canal de eventos [61].

3.6.2. Beats comunitarios

Algunos de los Beats de código abierto desarrollados por la comunidad [62] pueden verse en la tabla 3.5.

TABLA 3.5. BEATS DESARROLADOS POR LA COMUNIDAD

Nombre del Beat	Descripción
apachebeat	Lee el estado de un servidor web Apache.
cloudfrontbeat	Lee *logs* de Amazon CloudFront.
dockbeat	Lee estadísticas de contenedores Docker.
execbeat	Ejecuta de forma periódica comandos *shell* y envía sus salidas estándar.
githubbeat	Monitoriza la actividad en un repositorio de GitHub.
httpbeat	Realiza peticiones a nodos HTTP(S) y envía los datos.
journalbeat	Envía *logs* de sistemas Linux basados en systemd/journal.
mongobeat	Monitoriza bases de datos MongoDB, también permite el envío de documentos.
mysqlbeat	Realiza un consulta en MySQL y envía los resultados.
nagioscheckbeat	Envía verificaciones de Nagios y datos de rendimiento.
nginxbeat	Lee el estado de un servidor web Nginx.
packagebeat	Recopila información de los gestores de paquetes sobre los paquetes instalados en el sistema.
serialbeat	Lee de un puerto serie.
tracebeat	Lee la salida de un *traceroute* y envía los resultados.

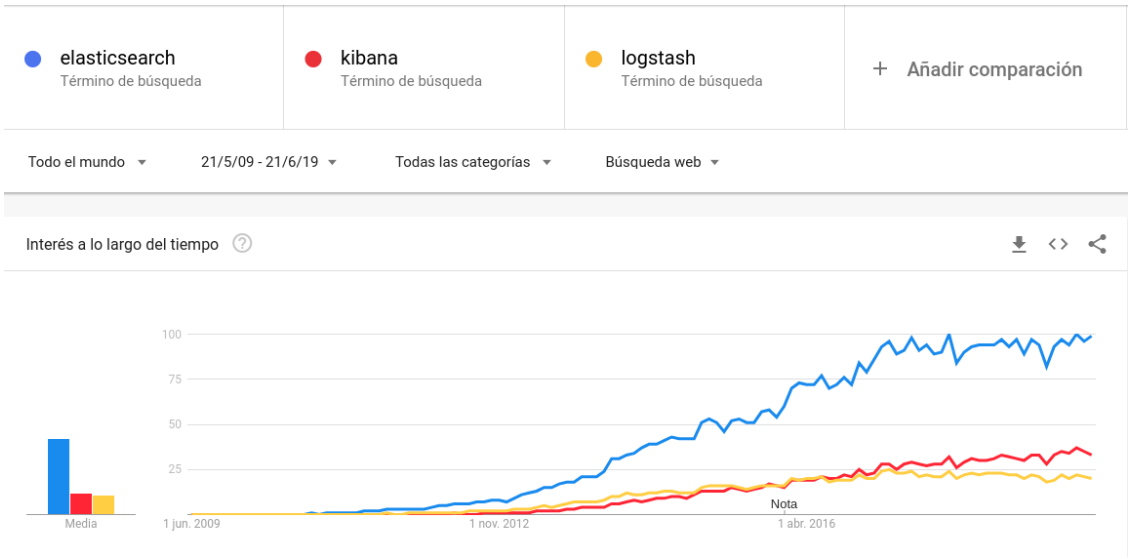


Fig. 3.7. Crecimiento de la pila Elastic, Google Trends.

3.7. Discusión

A lo largo del capítulo se ha ido introduciendo y desarrollando los distintos elementos que componen la pila Elastic y sus funcionalidades.

En primer lugar, cumple con el requisito de tratarse de software gratuito, ya que ambas licencias, tanto la Apache2.0 [19], como la de Elastic [20], ambas de código abierto, permiten utilizar el software en la elaboración del sistema previsto.

La existencia de los clientes ligeros Beats cumple a la perfección con el objetivo de interferir lo mínimo posible en el cortafuegos monitorizado, siendo Filebeat una alternativa perfecta, ya que garantiza que no se perderán datos en su transmisión al sistema de monitorización.

También cumple con los requisitos de visualización establecidos. La existencia de Kibana y su integración con Elasticsearch permite implementar toda la parte gráfica con una curva de aprendizaje relativamente corta.

Respecto a la necesidad de transformación de datos previa a su visualización final, ha quedado demostrado en este capítulo que Logstash cumple con las necesidades del sistema a desarrollar.

Por último, esta solución permite trabajar con un proyecto relativamente joven, pero que ha sufrido un fuerte crecimiento estos últimos años, como puede verse en la figura 3.7. A pesar de ello, existen hoy en día pocos trabajos académicos en los que se utilice esta herramienta [63], algo que contrasta con el amplio número de organizaciones que sí la utilizan [64].

4. SISTEMA DE MONITORIZACIÓN

A lo largo de este capítulo se expondrá la solución planteada para monitorizar el cortafuegos y los paquetes que lo atraviesan. En primer lugar se explica de forma global la arquitectura que tiene el sistema de monitorización, para posteriormente detallar paso a paso el proceso de implementación.

4.1. Arquitectura planteada

Como se introdujo brevemente en el capítulo 1, la metodología ingenieril establece una serie de etapas, de las cuales abordaremos la segunda, consistente en desarrollar un sistema que cumpla con los objetivos establecidos en la sección 1.2.

La arquitectura propuesta, que hace uso de las herramientas y del cortafuegos explicados en los capítulos 2 y 3 sigue en todo momento el flujo de los datos de nuestro sistema, es decir, desde el cortafuegos hacia el sistema de monitorización.

En un primer lugar en este flujo de datos se encuentra el cortafuegos a monitorizar, el cual generará *logs* en texto plano. Para ello se hace uso del *target* TRACE de IPTables y de dos *scripts* de Python. En un fichero de *logs* se encontrarán los datos de monitorización del cortafuegos y en el otro de los paquetes que lo atraviesan.

En un segundo lugar, el sistema de monitorización debe introducir estos ficheros de *log* en la pila Elastic, trasladándolos a la máquina desde la cual se va a monitorizar. Para ello se usarán las herramientas Beats, Logstash y Elasticsearch descritas en el capítulo 3.

Por último, el sistema de monitorización dispone de un *front-end*, el cual muestra los datos de monitorización al usuario del sistema de una forma gráfica. Para ello, se explotará la herramienta Kibana descrita en el capítulo 3.

Es por esto que se decide separar la implementación del sistema en tres etapas. Cada una de ellas corresponde a una parte de la arquitectura descrita: etapa de registro, etapa de transferencia y transformación de *logs* y etapa de visualización.

4.2. Etapa de registro

Tal y como se ha explicado en el capítulo 1, el cortafuegos empleado en este TFG es el presente en los *kernels* de Linux modernos, Netfilter/IPTables. En el capítulo 2 se detalló el funcionamiento y arquitectura del mismo así como de sus reglas.

En primer lugar, se va a profundizar algo más en el proceso de extracción de *logs* del mismo, para lo que el *target* TRACE se vuelve imprescindible.

4.2.1. *Target TRACE* del cortafuegos

Del manual de Linux se obtiene el funcionamiento de este *target* [65]:

```
TRACE
 This target marks packets so that the kernel will log every rule
    which match the packets as those traverse the tables, chains,
    rules.

 A logging backend, such as ip(6)t_LOG or nfnetlink_log, must be
    loaded for this to be visible. The packets are logged with the
    string prefix: "TRACE: tablename:chainname:type:rulenum " where
     type can be "rule" for plain rule, "return" for implicit rule
    at the end of a user defined chain and "policy" for the policy
    of the built in chains.
 It can only be used in the raw table.
```

Es decir, si un paquete es marcado con este *Target*, el *kernel* nos va a generar un *log* cada vez que coincida con alguna regla del cortafuegos, lo que se ajusta completamente a las necesidades de nuestro sistema.

Como se muestra en la figura 2.1, y como indica el manual, dicha regla irá colocada en la tabla raw en sus cadenas PREROUTING y POSTROUTING. Será la primera de todas las reglas de dichas cadenas. Por lo tanto, quedan definidos dos *scripts*, los cuales permiten añadir y eliminar estas reglas. Hacen falta permisos de superusuario para su ejecución.

```
#!/bin/bash

iptables -t raw -I PREROUTING ! -i lo -j TRACE
iptables -t raw -I OUTPUT ! -o lo -j TRACE
```

Este *script* inserta ambas reglas en la primera posición de las cadenas. Se excluye voluntariamente el marcaje de paquetes con destino la interfaz de *loopback*.

```
#!/bin/bash

iptables -t raw -D PREROUTING ! -i lo -j TRACE
iptables -t raw -D OUTPUT ! -o lo -j TRACE
```

Este por el contrario elimina ambas reglas de las tablas del cortafuegos. Resulta recomendable eliminarlas especificando la regla completa en lugar de su orden, para evitar, por ejemplo, borrar dos veces la primera regla por error.

Lo que hace este *logging backend* es hacer una copia de y enviarlos al espacio del usuario a través de un *socket* de tipo *netlink* [66]. Los procesos de este espacio de usuario pueden suscribirse al grupo para recibir los paquetes, los cuales contienen por encima de

la cabecera IP otra NFLOG. Uno de los campos de esta cabecera NFLOG es *Trace*, que contiene exactamente lo que indica el *target* en el manual.

4.2.2. Registro de los paquetes

Una vez completado el proceso de envío de los paquetes desde el espacio del *kernel* al espacio de usuario, se pasará a construir un *script* que permita completar esta fase de desarrollo del sistema. El objetivo será el registro en texto plano de los paquetes, que tiene como principal ventaja el ser almacenable y legible por humanos.

La solución adoptada ha sido programar un *script* en Python3, que permita generar un *log* personalizado con los datos que se necesitan. Para ello se utilizan principalmente dos módulos: pyshark y logging.

Módulo pyshark

El módulo de Python pyshark [67] es una implementación para Python de la herramienta `tshark` [68], que a su vez es la versión de terminal de Wireshark [69]. Wireshark y Tshark son analizadores de protocolos de red que nos van a permitir analizar los paquetes. Con pyshark se podrá extraer cualquier campo de las cabeceras de los paquetes.

Módulo logging

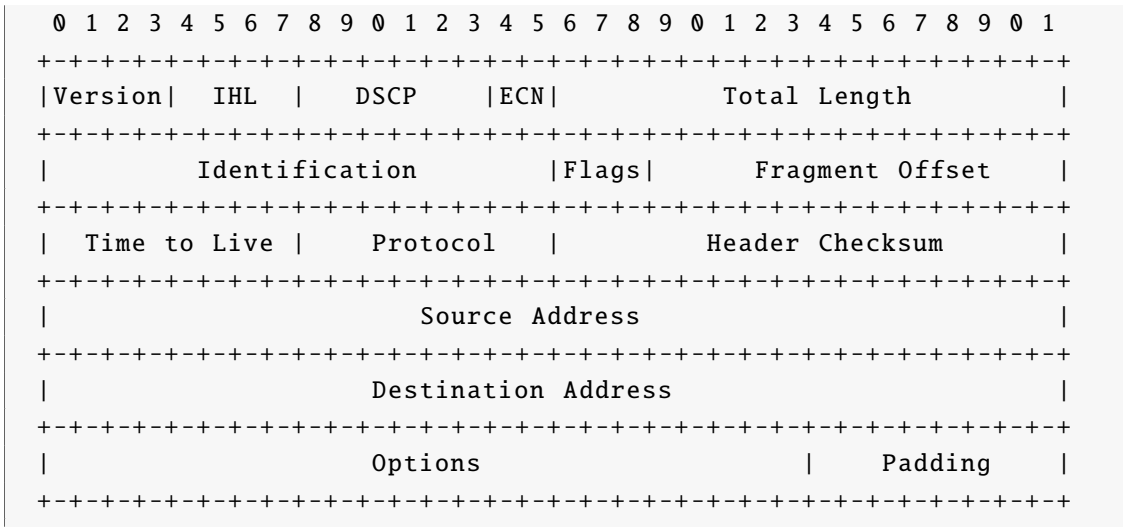
El modulo de Python logging [70], por su parte, permite simplificar el proceso de escritura de los *logs* en el disco y dispone de una documentación oficial muy extensa [71] [72]. Otra función a usar de dicho módulo es la de rotado de *logs*, que permite controlar el tamaño máximo que van a ocupar los ficheros en el disco.

Como en Internet hay una gran variedad y diversidad de protocolos de nivel de aplicación, resulta imprescindible limitar las funciones de nuestro sistema de monitorización únicamente a las cabeceras de capa de red y transporte, donde con IP, TCP, UDP e ICMP se puede analizar de forma común cualquier tipo de protocolo de nivel superior que opere sobre ellos. Por ello, en primer lugar va a realizarse un análisis de las cabeceras de estos para ver que datos se van a extraer. El fin último de esto es permitir identificar inequívocamente un paquete.

Cabecera IP

Como se especifica en el RFC 791 [73] y sus posteriores actualizaciones, la cabecera IPv4 tiene la siguiente forma:

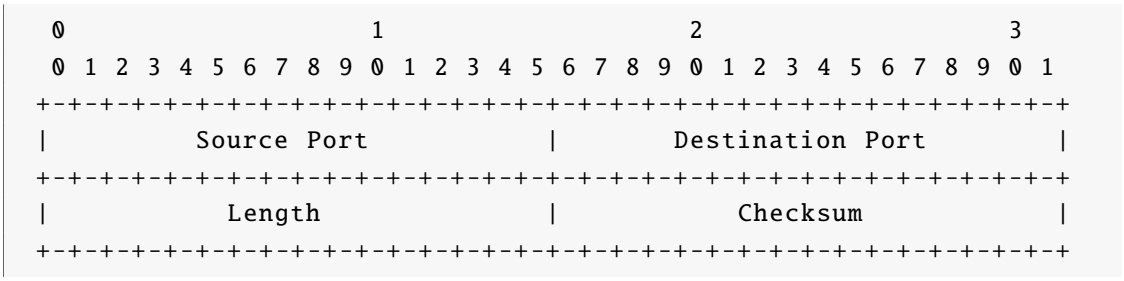
```
1  0                   1                   2                   3
```



- **Version**. Versión del protocolo IP utilizado.
- **IHL**. Tamaño de la cabecera IP.
- **DSCP**. Servicios diferenciados.
- **ECN**. Notificación de congestión explícita.
- **Total Length**. Tamaño total del paquete IP.
- **Identification**. Permite identificar fragmentos de un mismo datagrama IP.
- **Flags**. Tres bits para identificación y control de fragmentos
  - **Reserved**. Sin uso
  - **DF**. Prohibido fragmentar el paquete.
  - **MF**. Existen más fragmentos del paquete.
- **Fragment Offset**. Desfase de un fragmento respecto al principal.
- **Time to Live**. Número de saltos restantes antes de ser descartado el paquete.
- **Protocol**. Protocolo encapsulado en el paquete IP.
- **Header Checksum**. Suma de comprobación de la cabecera.
- **Source Address**. Dirección IP de origen.
- **Destination Address**. Dirección IP de destino.

Cabecera UDP

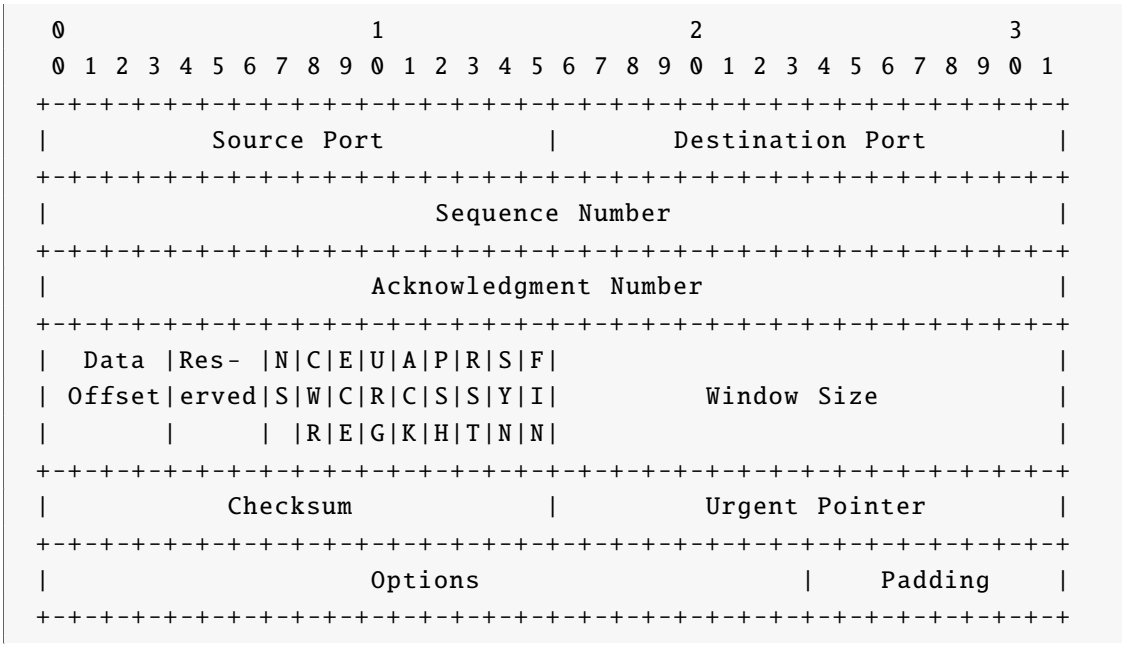
Como se indica en el RFC 768 [74], la cabecera UDP tiene la siguiente forma:



- **Source Port**. Puerto de origen del paquete.
- **Destination Port**. Puerto destino del paquete.
- **Length**. Tamaño total del paquete UDP.
- **Checksu**. Suma de comprobación.

Cabecera TCP

Definida en el RFC 793 [75] y sucesivas actualizaciones, la cabecera TCP tiene la siguiente forma:



- **Source Port**. Puerto de origen del paquete.
- **Destination Port**. Puerto destino del paquete.

- **Sequence Number**. Número de secuencia de la sesión TCP.
- **Acknowledgment Number**. Número de secuencia de la sesión TCP confirmado.
- **Data Offset**. Tamaño de la cabecera TCP.
- **Reserved**. Sin uso.
- **Flags**. 9 bits con distintos usos:
  - **NS**. Uso experimental para protección de ocultamiento.
  - **CWR**. Notifica de que se ha recibido un segmento TCP con el bit ECE, y que se ha respondido con mecanismos de control de congestión.
  - **ECE**. Sirve para indicar congestión.
  - **URG**. Indica que el campo Urgent Pointer contiene un valor significativo.
  - **ACK**. Indica que el campo Acknowledgment Number contiene un valor significativo.
  - **PSH**. Indica que se deben pasar los datos almacenados en el *buffer* a la capa de aplicación.
  - **RST**. Resetea la conexión.
  - **SYN**. Sincroniza los números de secuencia.
  - **FIN**. Último paquete enviado.
- **Window Size**. Tamaño de la ventana de recepción.
- **Checksum**. Suma de comprobación del paquete TCP.
- **Urgent Pointer**. Indica el desfase respecto al último número de secuencia urgente.

Cabecera ICMP

Se especifica en el RFC 792 [76] y en posteriores actualizaciones, el formato de la cabecera depende del tipo de tipo de mensaje ICMP. La parte común a todos ellos tiene la siguiente forma:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Rest of Header                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- **Type**. Tipo de mensaje ICMP.

- **Code**. Código del mensaje ICMP.
- **Checksum**. Suma de comprobación.
- **Rest of Header**. Varía en función del tipo de paquete ICMP y el código.

Formato del *log*

Antes de pasar a la fase de creación del *script*, hay que decidir qué forma van a tener los *logs*. Para ello se distinguen cuatro casos y se eliminan los campos de *checksum* y longitudes de las cabeceras ya que no aportan información adicional. Se respeta en todo momento el orden de los elementos en las cabeceras del protocolo correspondiente.

- **Parte común**
  "marca temporal" + "prefijo NFLOG" + "protocolo de transporte" + "protocolo superior" +
- **Cabecera IP**
  + "DSCP" + "ECN" + "Identification" + "DF" + "MF" + "Fragment Offset" + "Time to Live" + "Source Address" + "Destination Address" +
- **Cabecera transporte**
  - **Cabecera UDP**
    + "Source Port" + "Destination Port"
  - **Cabecera TCP**
    + "Source Port" + "Destination Port" + "Sequence Number" + "Acknowledgment Number" + "NS" + "CWR" + "ECE" + "URG" + "ACK" + "PSH" + "RST" + "SYN" + "FIN" + "Window Size" + "Urgent Pointer"
  - **Cabecera ICMP**
    + "Type" + "Code"
  - **Cabecera otros protocolos**

Para la marca temporal se utiliza la notación estándar ISO 8601 [77].

Script de registro de paquetes

Con las especificaciones para el *script* definidas, se crea el siguiente código de Python:

```
1 #!/usr/bin/python3
2
3 import pyshark
4 import logging
5 import logging.handlers as handlers
```

```
6
7   logger = logging.getLogger('packets')
8   logger.setLevel(logging.INFO)
9
10  formatter = logging.Formatter('%(message)s')
11  logHandler = handlers.RotatingFileHandler('/var/log/tfg/packets.log'
        , maxBytes=1000000000, backupCount=5)
12  logHandler.setLevel(logging.INFO)
13  logHandler.setFormatter(formatter)
14  logger.addHandler(logHandler)
15
16  capture = pyshark.LiveCapture(interface='nflog')
17
18  for pkt in capture.sniff_continuously():
19      prefix = pkt.nflog.prefix.split(' ')[1]
20      timestamp = pkt.sniff_time.isoformat()
21      if hasattr(pkt, 'tcp'):
22          logger.info(f'{timestamp} {prefix} tcp {pkt.highest_layer} {
                pkt.ip.dsfield_dscp} {pkt.ip.dsfield_ecn} {pkt.ip.id} {
                pkt.ip.flags_df} {pkt.ip.flags_mf} {pkt.ip.frag_offset}
                {pkt.ip.ttl} {pkt.ip.src} {pkt.ip.dst} {pkt.tcp.srcport}
                 {pkt.tcp.dstport} {pkt.tcp.seq} {pkt.tcp.ack} {pkt.tcp.
                flags_ns} {pkt.tcp.flags_cwr} {pkt.tcp.flags_ecn} {pkt.
                tcp.flags_urg} {pkt.tcp.flags_ack} {pkt.tcp.flags_push}
                {pkt.tcp.flags_reset} {pkt.tcp.flags_syn} {pkt.tcp.
                flags_fin} {pkt.tcp.window_size} {pkt.tcp.urgent_pointer
                }')
23      elif hasattr(pkt, 'udp'):
24          logger.info(f'{timestamp} {prefix} udp {pkt.highest_layer} {
                pkt.ip.dsfield_dscp} {pkt.ip.dsfield_ecn} {pkt.ip.id} {
                pkt.ip.flags_df} {pkt.ip.flags_mf} {pkt.ip.frag_offset}
                {pkt.ip.ttl} {pkt.ip.src} {pkt.ip.dst} {pkt.udp.srcport}
                 {pkt.udp.dstport}')
25      elif hasattr(pkt, 'icmp'):
26          logger.info(f'{timestamp} {prefix} icmp {pkt.highest_layer}
                {pkt.ip.dsfield_dscp} {pkt.ip.dsfield_ecn} {pkt.ip.id} {
                pkt.ip.flags_df} {pkt.ip.flags_mf} {pkt.ip.frag_offset}
                {pkt.ip.ttl} {pkt.ip.src} {pkt.ip.dst} {pkt.icmp.type} {
                pkt.icmp.code}')
27      else:
28          logger.info(f'{timestamp} {prefix} other {pkt.highest_layer}
                 {pkt.ip.dsfield_dscp} {pkt.ip.dsfield_ecn} {pkt.ip.id}
                {pkt.ip.flags_df} {pkt.ip.flags_mf} {pkt.ip.frag_offset}
                 {pkt.ip.ttl} {pkt.ip.src} {pkt.ip.dst}')
```

Para su construcción se ha seguido la documentación oficial [67] [70]. Mediante el uso de *RotatingFileHandler* [78], se especifica el número de ficheros de *logs* que se van a almacenar y rotar; en este caso, 5 ficheros de 1GB cada uno, que seguirán el estándar, por lo que se renombrarán siguiendo la forma *app.log*, *app.log.1*, *app.log.2*, hasta *app.log.5*.

También se ha especificado que los *logs* se almacenen en `/var/log/tfg/packets.log`, siguiendo el estándar de jerarquía del sistema de ficheros de The Linux Foundation [79]

Los *logs* resultantes quedan de la siguiente forma:

```
1 2019-06-22T13:29:37.258328 filter:OUTPUT:rule:1 tcp TCP 0 0 0
     x0000dfe1 1 0 0 64 10.133.140.252 216.58.201.163 36208 443 1 1 0
      0 0 0 1 0 0 0 0 255 0
2 2019-06-22T13:29:34.879120 security:OUTPUT:policy:2 udp DNS 0 0 0
     x0000d0cb 1 0 0 64 10.133.140.252 157.88.18.190 36013 53
3 2019-06-22T13:28:25.604815 mangle:OUTPUT:rule:1 icmp ICMP 0 0 0
     x000073ec 1 0 0 64 10.133.140.252 216.58.214.163 8 0
```

4.2.3. Registro de la configuración del cortafuegos

Para completar los objetivos de esta primera fase de nuestro sistema de monitorización especificados en la sección 4.1, el sistema debe almacenar la configuración completa del cortafuegos en un instante determinado. Para ello la solución adoptada es idéntica a la de la sección 4.2.2, usando el mismo módulo de *logging* [70] junto con el módulo *python-iptables* [80]. Se ha optado por este módulo en lugar de otros por ser el más y mejor documentado.

Formato del *log*

Se especifica nuevamente el formato del *log*, en el cual se diferencia el caso en el que se trate de la configuración de una regla, con el de una política de cadena.

- **Parte común**
  "Marca Temporal" + "Tabla" + "Cadena" + "Número de regla" +
  - **Regla de una cadena**
    + "*Matches* + *Target*" + "*Target*"
  - **Política de una cadena**
    + "Política"

Para facilitar el tratamiento posterior de los *logs*, se le asigna a la política de cada cadena un numero de regla correspondiente al número total de reglas de la cadena más uno. De nuevo, para la marca temporal se sigue el estándar ISO 8601 [77]. Por simplicidad se duplica la información del *target* en las reglas. En el caso de las cadenas definidas, al no tener política, se establece por defecto *return*.

Script de registro de reglas

Con la documentación oficial de ambos módulos de *logging* [70] y *python-iptables* [80], el *script* resultante adquiere la siguiente forma:

```
#!/usr/bin/python3

import logging
import logging.handlers as handlers
import iptc
import time
from datetime import datetime
import sys

logger = logging.getLogger('firewall')
logger.setLevel(logging.INFO)

formatter = logging.Formatter('%(message)s')
logHandler = handlers.RotatingFileHandler('/var/log/tfg/firewall.log', maxBytes=10000000, backupCount=5)
logHandler.setLevel(logging.INFO)
logHandler.setFormatter(formatter)
logger.addHandler(logHandler)

timeinterval = 30

if len(sys.argv) >= 2:
    timeinterval = int(sys.argv[1])

while True:
    now = datetime.now()
    timestamp = now.isoformat(timespec='seconds')
    for thing in [iptc.Table.FILTER, iptc.Table.RAW, iptc.Table.NAT, iptc.Table.MANGLE, iptc.Table.SECURITY]:
        table = iptc.Table(thing)
        for chain in table.chains:
            rulenum = 1
            for rule in chain.rules:
                logger.info(f'{timestamp} Table: {table.name} Chain: {chain.name} Rule: {rulenum} {iptc.easy.decode_iptc_rule(rule)} Target: {rule.target.name}')
                rulenum += 1
            try:
                policy = chain.get_policy().name
            except:
                policy = "return"
            logger.info(f'{timestamp} Table: {table.name} Chain: {chain.name} Rule: {rulenum} Policy: {policy}')
    time.sleep(timeinterval)
```

Este *script* tiene la particularidad de que permite introducir el intervalo de tiempo en segundos que va a esperar entre distintas muestras de la configuración total del cortafuegos. Si no se introduce ningún valor, se realiza cada 30 segundos. Para poder acceder a la configuración del cortafuegos, de nuevo son necesarios permisos de superusuario. Se ha considerado el caso del uso de cadenas personalizadas, en las que no tiene por qué haber política por defecto. Nuevamente, se realiza rotación de *logs*.

Los *logs* resultantes quedan de la siguiente forma:

```
2019-06-23T14:03:14 Table: raw Chain: PREROUTING Rule: 1 {'in-
    interface': '!lo', 'target': 'TRACE'} Target: TRACE
2019-06-23T14:03:14 Table: raw Chain: OUTPUT Rule: 2 Policy: ACCEPT
2019-06-23T14:03:14 Table: filter Chain: pruebacadena Rule: 1 Policy
    : return
```

4.3. Etapa de transferencia y transformación de *logs*

Una vez completada la primera etapa de la arquitectura propuesta en la sección 4.1, en esta segunda se abordará el envío y transformación de *logs* hacia la máquina monitorizadora.

Para ello comienza la integración en las distintas herramientas de la pila *Elastic*. Como ya se analizó en el capítulo 3, dispone de un cliente ligero, llamado Filebeat que cumple a la perfección las necesidades de transmisión de estos *logs*.

4.3.1. Envío de *logs* con Filebeat

La solución escogida para instalar Filebeat es la de repositorio APT [81]. Existen muchas otras alternativas, desde instalar directamente los paquetes deb o rmp, msi para sistemas Windows, brew para macOS, docker hasta instalarse desde archivo en Linux [82].

El proceso seguido para su instalación desde de los repositorios desde terminal es el siguiente:

```
sudo apt install apt-transport-https
echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main"
    | sudo tee -a /etc/apt/sources.list.d/elastic-7.x.list
sudo apt update
```

La instalación de Filebeat se realizaría simplemente con el comando `apt install`. Para configurar Filebeat, el fichero de configuración */etc/filebeat/filebeat.yml* deberá tener, al menos, las siguientes lineas.

```
filebeat.inputs:
```

```
  - type: log
      - /var/log/tfg/packets.log
      tags: ["packet"]
  - type: log
      - /var/log/tfg/firewall.log
      tags: ["firewall"]

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

De esta forma, se indica qué dos ficheros de *logs* debe procesar, así como que añada una etiqueta distinta a cada uno de ellos para poder diferenciarlos con mayor facilidad en Logstash [83]. Como salida, se ha configurado una de tipo Logstash, en la que deberemos asignar la IP de nuestra maquina monitorizadora [84].

4.3.2. Recepción y transformación de *logs* con Logstash

Como segundo apartado de esta etapa, se trabajará ahora en la maquina monitorizadora, en la cual se va a realizar la recepción, transformación de *logs* y almacenaje en base de datos. Para ello, se combinan las herramientas de la pila Elastic: Logstash y Elasticsearch. En primer lugar se instalan las herramientas logstash y elasticsearch de forma equivalente a la realizada en la sección 4.3.1

Se comenzará ahora a configurar la tubería de Logstash, la cual se introdujo en el capítulo 3. En primer lugar, se analiza el funcionamiento de los ficheros y directorios de configuración de Logstash, pues esta tubería deberá, cumpliendo con los objetivos establecidos en la sección 1.2, poder integrarse en soluciones de monitorización ya existentes.

Tubería de Logstash

Para poder añadir la tubería a Logstash sin alterar otras existentes que pudiese haber ya configuradas, al fichero de configuración */etc/logsatsh/pipelines.yml* se le añaden las siguiente líneas: [85].

```
  - pipeline.id: my-pipeline_1
    path.config: "/etc/logstash/p1.config"
```

Por lo que será en esta ruta donde se creará el fichero de configuración de la tubería `/etc/logstash/p1.config`.

Consideraciones generales sobre la tubería de Logstash:

- La tubería deberá distinguir entre dos tipos de *logs*, diferenciados por las etiquetas *packet* y *firewall*.

- En ambos casos, se asigna la marca temporal del *log* a la del documento de Elasticsearch.
- Consideración especial a la marca temporal de los paquetes, ya que de forma nativa Elasticsearch solo soporta marcas temporales de precisión milisegundos. Se deberá en este caso recortar la marca temporal y almacenarla también en microsegundos.
- Se deben convertir todos los campos a su tipo de datos correspondiente: *string*, *integer*, *float* o *boolean*.
- Para cada paquete se construirá un identificador que permita reconocerlo a su paso por las distintas reglas del cortafuegos. Para ello se va a crear una firma de tipo no criptográfica con todos los campos de las cabeceras de los que se dispone, a excepción del campo *Time To Live*, que es alterado en el momento del reenvío de paquetes.
- Para cada configuración del cortafuegos, se crea un identificador en función del instante en el que se leyó la configuración.
- Se deberán indexar los documentos en el índice correspondiente de Elasticsearch: *index-firewall* o *index-packets*.

Plugins de entrada

Antes de abordar la configuración del *plugin* de entrada en la tubería, se diferencian dos casos. En el primero, el entorno real de desarrollo del TFG: el cortafuegos monitorizado se sitúa en una máquina distinta de la que monitoriza. En el segundo, y debido a las limitaciones de hardware de este TFG, ambos se encuentran en la misma máquina.

Para el primer escenario, se continua con la configuración basándose en el cliente Beats, por lo que el *plugin* de entrada será Beats [86].

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

En el segundo, el *plugin* de entrada será el fichero de *log* directamente, por lo que se usa el *plugin* de entrada File [87].

```
input {
  file {
    path => ["/var/log/tfg/firewall.log"]
    tags => ["firewall"]
    start_position => "end"
  }
```

```
    file {
      path => ["/var/log/tfg/packets.log"]
      tags => ["packet"]
      start_position => "end"
    }
  }
```

Este segundo caso permite visualizar de forma explícita un comportamiento que comparten ambos, los ficheros, al arrancar tanto Losgstash como Filebeat, se leen desde el final. Es decir, las nuevas lineas que se escriban a partir de ese momento.

Plugins de filtrado

Como se ha especificado en la sección 4.3.2, se van filtrar en la misma tubería dos tipos de eventos totalmente diferentes.

- **Eventos de configuración del cortafuegos**
  Se pueden diferenciar fácilmente porque tienen una etiqueta *firewall* en el evento. Tienen la estructura descrita en la sección 4.2.3. El esquema de filtrado en este caso se compone a rasgos generales de:
  - **Filtro Grok** [88]: permite diferenciar los casos en los que la línea de *log* sea una regla o una política. Extrae a su vez la información en campos.
  - **Filtro Date** [89]: convierte un campo con marca temporal en la marca temporal del evento.
  - **Filtro Fingerprint** [90]: permite crear una huella en función de campos de datos. Se crea una huella única para todos los eventos que compartan marca temporal. No es necesario usar algoritmos criptográficos, por lo que se opta por el algoritmo de *hash* MURMUR3.
- **Eventos de paquetes**
  En este caso, la etiqueta que diferencia este tipo será *packet*, y tendrán la estructura descrita en la sección 4.2.2. El esquema de filtrado es el siguiente:
  - **Filtro Dissect** [91]: separa la parte común a todos los *logs* mediante el uso de delimitadores. En este caso el delimitador será el espacio.
  - **Filtro Ruby** [92]: se utiliza código Ruby [93] en la generación de nuestra marca temporal en microsegundos.

```
t = Time.parse(event.get('sniff_timestamp'))
event.set('timestamp_us', t.to_i * 1000000 + t.usec)
```

  - **Filtro Date**: de la misma manera que en caso de la configuración del cortafuegos, se asigna a la marca temporal del evento la del *log*.

- **Filtro Dissect**: nuevamente se aplica este filtro con la parte del *log* diferenciado en cada tipo de protocolo.
- **Filtro Fingerprint**: se crea una huella que hace las funciones de identificador de cada paquete distinto.

Para completar todo el proceso de transformación, se aplican los filtros Mutate [94] necesarios para convertir los campos a su tipo adecuado y darles un nombre coherente. Puede verse el resultado en el Anexo 2.

Plugins de salida

Como se va a usar Elasticsearch como base de datos, se utiliza el *plugin* Elasticsearch [95] como salida. El *host* seleccionado en este caso será *localhost*, ya que tanto Logstash como Elasticsearch están desplegados en la misma máquina. Se diferencia nuevamente en función de la etiqueta del evento en que índice se almacenarán los documentos.

```
output {
#  stdout {
#    codec => rubydebug { metadata => true }
#  }

  if "packet" in [tags] {
    elasticsearch {
      hosts => [ "localhost:9200" ]
      index => "index-packets"
    }
  } else if "firewall" in [tags] {
    elasticsearch {
      hosts => [ "localhost:9200" ]
      index => "index-firewall"
    }
  }
}
```

De esta forma, la segunda etapa en el desarrollo del sistema de monitorización queda completada.

4.4. Etapa de visualización

Para abordar la ultima etapa descrita en la sección 4.1, se utilizará la herramienta Kibana para visualizar los datos de monitorización obtenidos en las dos etapas anteriores.

El proceso de instalación de la herramienta Kibana de nuevo se realiza de la forma escogida en la sección 4.3.1.

En este momento Kibana ya se encontrará accesible desde cualquier navegador web accediendo a `http://localhost:5601`.

Planteamiento de la interfaz gráfica

Para configurar *Kibana* con el objetivo de que muestre los datos de monitorización de forma gráfica, hay que tener en cuenta en todo momento las posibilidades y limitaciones vistas en el capítulo 3. Diseñaremos la interfaz de la forma descrita a continuación.

Se crearán 2 *dashboards*, uno para monitorizar los paquetes a su paso por el cortafuegos, y el otro para la configuración del mismo.

- ***Dashboard* de monitorización de la configuración del cortafuegos.**
  Este estará compuesto por:

  - Una visualización de tipo gráfico de barras verticales [96], que permita desplazarse temporalmente con mayor facilidad distintas entre configuraciones del cortafuegos.
  - Una visualización de control [97], que permita seleccionar en primer lugar la tabla y después la cadena de la cual ver su configuración.
  - Una búsqueda [98] en la que, ordenadas de forma descendente por numero de regla, podrán verse las reglas de la cadena seleccionada.

  Puede verse el *dashboard* completo en la figura 4.1.

- ***Dashboard* de monitorización de los paquetes**
  Estará compuesto por:

  - Una visualización de tipo gráfico de barras verticales, que permite desplazarnos temporalmente con facilidad entre todos los paquetes.
  - Tres visualizaciones de tipo gráfico circular [99], que permitan filtrar de forma rápida por IP origen y destino, puerto TCP origen y destino y puerto UDP origen y destino.
  - Una búsqueda en la que, ordenadas de forma descendente por marca temporal en microsegundos, podrán verse las reglas por las que ha pasado el paquete. Se añade también la columna con la huella para filtrar una vez encontrado el paquete específico el cual se monitoriza.

  Puede verse este segundo *dashboard* completo en la figura 4.2.

Para la creación de estos dos *dashboards* en primer lugar, una vez están creados los índices en Elasticsearch (*index-firewall* e *index-packets*), se crean dos *index patterns* en Kibana. Esto puede hacerse desde el menú *Management*. A la hora de crearlos especificaremos que se use como marca temporal el campo *@timestamp* de los documentos.

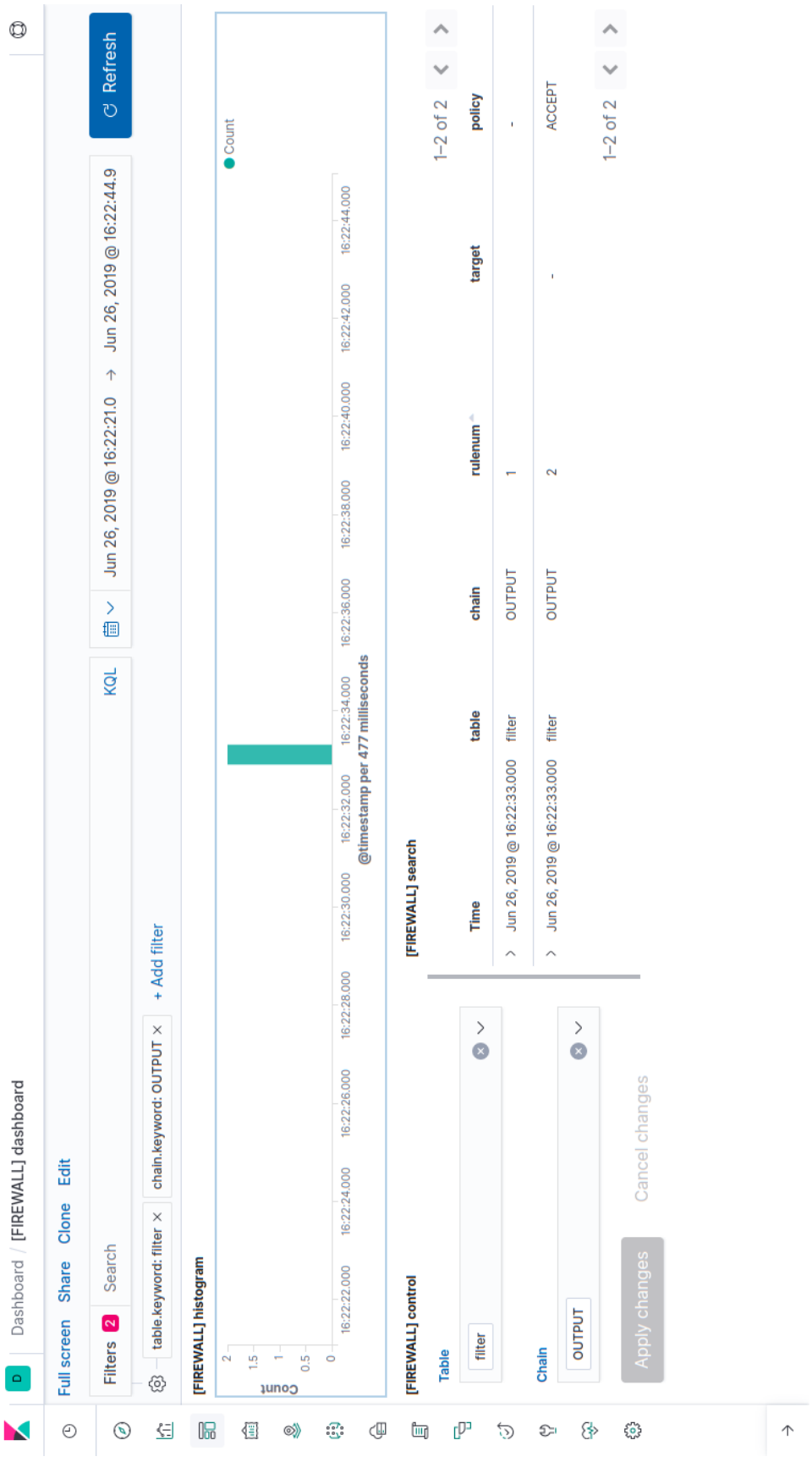


Fig. 4.1. *Dashboard* de monitorización de la configuración del cortafuegos.

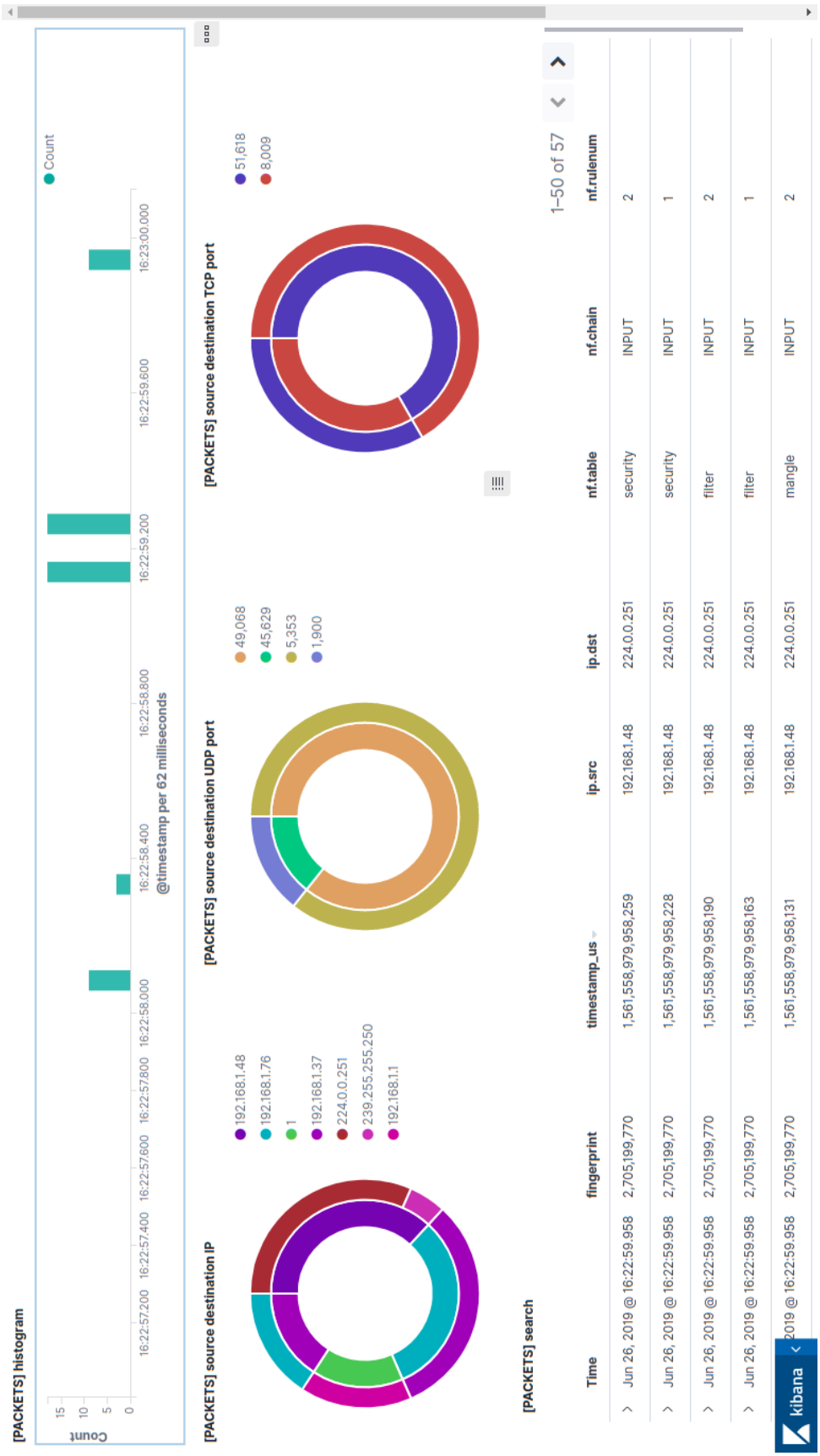


Fig. 4.2. *Dashboard* de monitorización de los paquetes (a pantalla completa).

Posteriormente hay que crear todas las visualizaciones y búsquedas que se van a añadir a los *dashboards*, en los menús de *Visualize* y *Discover* respectivamente. Al momento de guardar las búsquedas, debemos ordenar previamente las columnas con los campos que aparecerán por defecto en los *dashboards*.

La creación de los *dashboards* se realiza desde el correspondiente menú de *Dashboard*, las visualizaciones y búsquedas se ordenarán de la forma que sea conveniente.

La configuración que se ha descrito en esta sección de *Kibana* no deja de ser una de las múltiples opciones posibles. Pueden añadirse visualizaciones, *dashboards* y búsquedas de la forma que sea conveniente para unas necesidades de monitorización que no tienen por qué ser las mismas en todos los casos.

Con todo esto quedaría completado el despliegue de los sistemas de monitorización de paquetes y configuración del cortafuegos. Para iniciar su funcionamiento, se deben arrancar los servicios correspondientes: elasticsearch, filebeat, logstash y kibana. Una vez arrancados, se ejecutaran los dos *scripts* de Python.

```
sudo systemctl start filebeat
sudo systemctl start logstash
sudo systemctl start elasticsearch
sudo systemctl start kibana
sudo python3 logPackets.py
sudo python3 logFirewall.py 30
```

5. RESULTADOS Y VALIDACIÓN DEL SISTEMA

En este capítulo se explicarán las pruebas realizadas sobre el sistema completo, junto con los resultados obtenidos.

5.1. Hardware y software empleados

En la implementación del sistema se ha usado el siguiente hardware:

- Ordenador portátil *MSI GP60 2PE-422XES* con procesador *intel core* i5-4210H de 4 núcleos a 2.90GHz, 6GB de memoria RAM y 250GB de disco duro en estado sólido.

Las herramientas de software utilizadas se enumeran a continuación:

- Ubuntu versión 18.04.2 LTS.
- Python versión 3.6.8.
  - Módulo pyshark versión 0.4.2.3.
  - Módulo python-iptables versión 0.14.0.
- Logstash versión 7.1.1.
- Kibana versión 7.1.1.
- Elasticseach versión 7.1.1.
- Filebeat versión 7.1.1.
- Nmap versión 7.60.
- IPTables versión 1.6.1.

5.2. Validación del sistema de monitorización completo

Para validar el sistema desarrollado, el proceso se ha separado en dos partes.

En primer lugar, se validará el funcionamiento de la monitorización de la configuración del cortafuegos. Se comprobarán los siguientes casos, derivados de los objetivos detallados en la sección 1.3:

- Permite ver la configuración completa de cualquier regla en cualquier tabla.

- Permite ver las reglas de tablas personalizadas por el usuario.

Por último, se realizaran pruebas de la monitorización de paquetes. Para ello se comprobarán los siguientes casos.

- Se comprobará que se es capaz de realizar un seguimiento de paquetes tanto ICMP, UDP como TCP cuando tienen destino o salida el equipo con el cortafuegos. Se añadirán las reglas de prueba necesarias.
- Se comprobará que se puede realizar el seguimiento cuando el paquete es reenviado a otra máquina. Se añadirán las reglas de prueba necesarias.
- Se colocarán reglas que descartarán distintos tipos de paquetes y se comprobará que se es capaz de identificar exactamente qué regla provoca el descarte en cada caso.

5.2.1. Validación del sistema de monitorización de la configuración del cortafuegos

Para abordar esta primera parte del proceso de validación, debe plantearse primero una estructura de reglas de prueba, las cuales se cargarán en el cortafuegos para poder comprobar si se monitorizan correctamente y toda su configuración es accesible desde la interfaz Kibana. La configuración de prueba propuesta es la siguiente:

```
# Define chain to allow particular source addresses
iptables -N chain-incoming-ssh
iptables -A chain-incoming-ssh -s 192.168.1.148 -j ACCEPT
iptables -A chain-incoming-ssh -s 192.168.1.149 -j ACCEPT
iptables -A chain-incoming-ssh -j DROP

# Define chain to allow particular services
iptables -N chain-outgoing-services
iptables -A chain-outgoing-services -p tcp --dport 53  -j ACCEPT
iptables -A chain-outgoing-services -p udp --dport 53  -j ACCEPT
iptables -A chain-outgoing-services -p tcp --dport 123 -j ACCEPT
iptables -A chain-outgoing-services -p udp --dport 123 -j ACCEPT
iptables -A chain-outgoing-services -p tcp --dport 80  -j ACCEPT
iptables -A chain-outgoing-services -p tcp --dport 443 -j ACCEPT
iptables -A chain-outgoing-services -p tcp --dport 22  -j ACCEPT
iptables -A chain-outgoing-services -p icmp            -j ACCEPT
iptables -A chain-outgoing-services -j DROP

# Define chain to allow established connections
iptables -N chain-states
iptables -A chain-states -p tcp  -m conntrack --ctstate ESTABLISHED,
    RELATED -j ACCEPT
iptables -A chain-states -p udp  -m conntrack --ctstate ESTABLISHED,
    RELATED -j ACCEPT
iptables -A chain-states -p icmp -m conntrack --ctstate ESTABLISHED,
    RELATED -j ACCEPT
```

```
iptables -A chain-states -j RETURN

# Drop invalid packets
iptables -A INPUT -m conntrack --ctstate INVALID -j DROP

# Accept everthing on loopback
iptables -A INPUT  -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# Accept incoming/outgoing packets for established connections
iptables -A INPUT  -j chain-states
iptables -A OUTPUT -j chain-states

# Accept incoming ICMP
iptables -A INPUT -p icmp -j ACCEPT

# Accept incoming SSH
iptables -A INPUT -p tcp --dport 22 -j chain-incoming-ssh

# Accept outgoing
iptables -A OUTPUT -j chain-outgoing-services

## Drop everything else
iptables -P INPUT   DROP
iptables -P FORWARD DROP
iptables -P OUTPUT  DROP
```

Se comprueba que la configuración se aplica correctamente con *sudo iptables -L -v*:

```
Chain INPUT (policy DROP 10 packets, 926 bytes)
 pkts bytes target     prot opt in     out     source               destination
   27  2580 DROP       all  --  any    any     anywhere             anywhere            ctstate INVALID
   12  1181 ACCEPT     all  --  lo     any     anywhere             anywhere
  147 28624 chain-states  all  --  any    any     anywhere             anywhere
    0     0 ACCEPT     icmp --  any    any     anywhere             anywhere
    0     0 chain-incoming-ssh  tcp  --  any    any     anywhere             anywhere            tcp dpt:ssh

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

```
14     12  1181 ACCEPT     all  --  any    lo      anywhere
           anywhere
15     134 18053 chain-states  all  --  any    any     anywhere
                   anywhere
16     26  1876 chain-outgoing-services  all  --  any    any
           anywhere             anywhere
17
18   Chain chain-incoming-ssh (1 references)
19    pkts bytes target     prot opt in     out     source
       destination
20       0     0 ACCEPT     all  --  any    any     192.168.1.148
           anywhere
21       0     0 ACCEPT     all  --  any    any     192.168.1.149
           anywhere
22       0     0 DROP       all  --  any    any     anywhere
           anywhere
23
24   Chain chain-outgoing-services (1 references)
25    pkts bytes target     prot opt in     out     source
       destination
26       0     0 ACCEPT     tcp  --  any    any     anywhere
           anywhere             tcp dpt:domain
27       5   372 ACCEPT     udp  --  any    any     anywhere
           anywhere             udp dpt:domain
28       0     0 ACCEPT     tcp  --  any    any     anywhere
           anywhere             tcp dpt:ntp
29       4   304 ACCEPT     udp  --  any    any     anywhere
           anywhere             udp dpt:ntp
30       1    60 ACCEPT     tcp  --  any    any     anywhere
           anywhere             tcp dpt:http
31      14   740 ACCEPT     tcp  --  any    any     anywhere
          anywhere            tcp dpt:https
32       0     0 ACCEPT     tcp  --  any    any     anywhere
           anywhere             tcp dpt:ssh
33       0     0 ACCEPT     icmp --  any    any     anywhere
           anywhere
34       2   400 DROP       all  --  any    any     anywhere
           anywhere
35
36   Chain chain-states (2 references)
37    pkts bytes target     prot opt in     out     source
       destination
38     240 43238 ACCEPT     tcp  --  any    any     anywhere
          anywhere            ctstate RELATED,ESTABLISHED
39       5   637 ACCEPT     udp  --  any    any     anywhere
           anywhere             ctstate RELATED,ESTABLISHED
40       0     0 ACCEPT     icmp --  any    any     anywhere
           anywhere             ctstate RELATED,ESTABLISHED
41      36  2802 RETURN     all  --  any    any     anywhere
           anywhere
```

El resultado de la monitorización en Kibana puede verse en las figuras 5.1, 5.2, 5.3 y 5.4.

Se puede comprobar que el resultado es satisfactorio, pues cumple con los objetivos establecidos en la sección 1.2.

5.2.2. Validación del sistema de monitorización de paquetes

Para realizar las pruebas de funcionamiento correspondientes, en primer lugar se definen una serie de reglas que van a descartar unos tipos de tráfico especificados. Para generar este tráfico de prueba, se usan las herramientas *ping* y *nping* [100]. La primera permite crear tráfico ICMP de forma sencilla y la segunda se usará para TCP y UDP. Se tiene en este caso especial cuidado en seguir el diagrama de flujo de la imagen 2.1 a la hora de elaborar las reglas de prueba.

Tráfico ICMP

La primera prueba con tráfico ICMP se hará desde *localhost* hacia una dirección IP externa, por lo que los paquetes seguirán la siguiente ruta:

```
raw OUTPUT -> mangle OUTPUT -> nat OUTPUT -> nat OUTPUT -> filter
    OUTPUT -> security OUTPUT -> mangle POSTROUTING -> nat
    POSTROUTING
```

Se crean las siguientes reglas. Los paquetes ICMP serán descartados en *mangle POSTROUTING*:

```
iptables -t raw -A OUTPUT ! -o lo -j TRACE
iptables -t raw -A OUTPUT -p ICMP
iptables -t mangle -A OUTPUT -p ICMP
iptables -t nat -A OUTPUT -p ICMP
iptables -t filter -A OUTPUT -p ICMP
iptables -t security -A OUTPUT -p ICMP
iptables -t mangle -A POSTROUTING -p ICMP -j DROP
iptables -t nat -A POSTROUTING -p ICMP
```

Se comprueba que la configuración del cortafuegos es la correcta para la prueba.

```
root@host:~$ iptables -t raw -L OUTPUT
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
TRACE      all  --  anywhere             anywhere
           icmp --  anywhere             anywhere

root@host:~$ iptables -t mangle -L OUTPUT
Chain OUTPUT (policy ACCEPT)
```

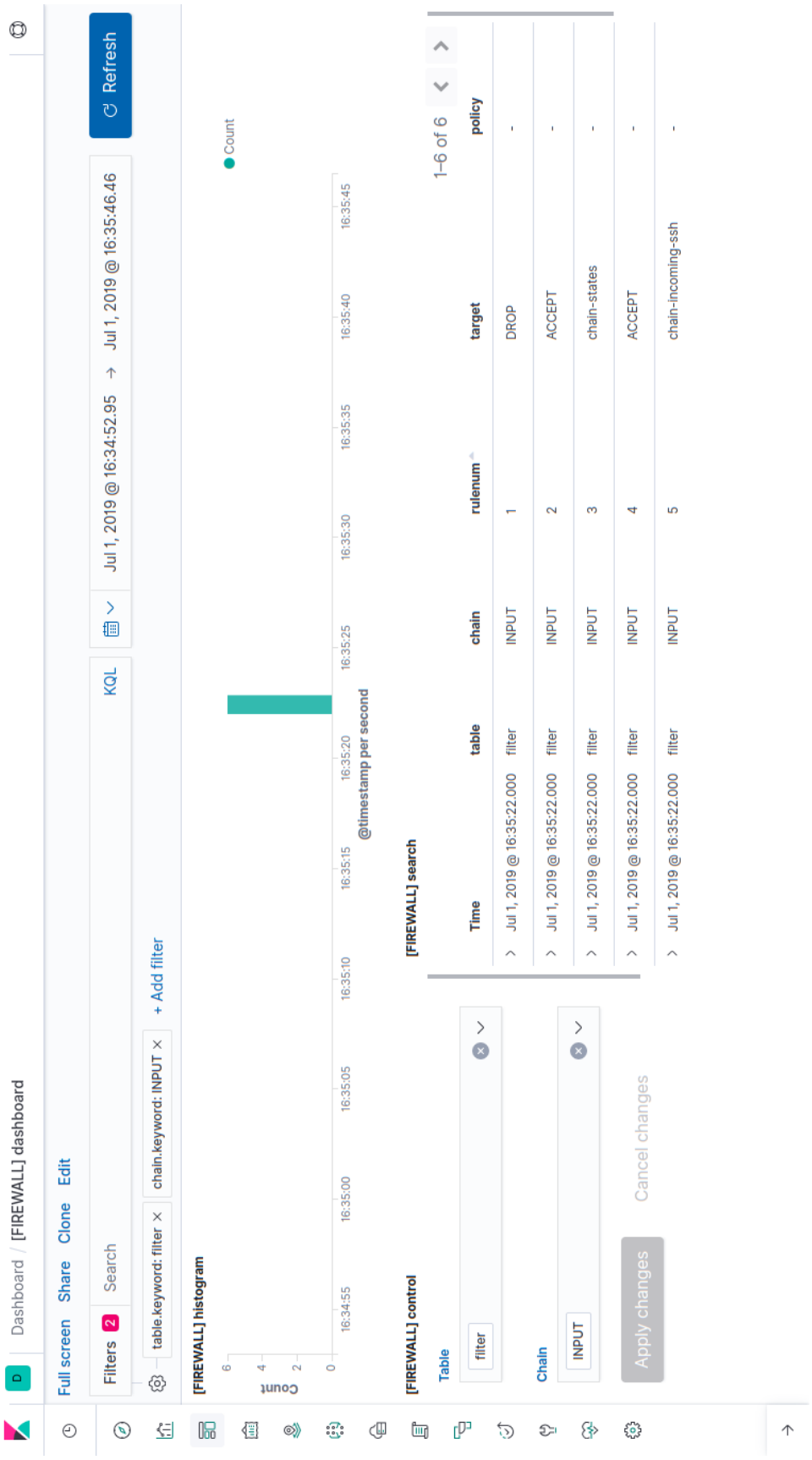


Fig. 5.1. Configuración de una cadena predefinida.

Fig. 5.2. Configuración de una regla en una cadena predefinida.

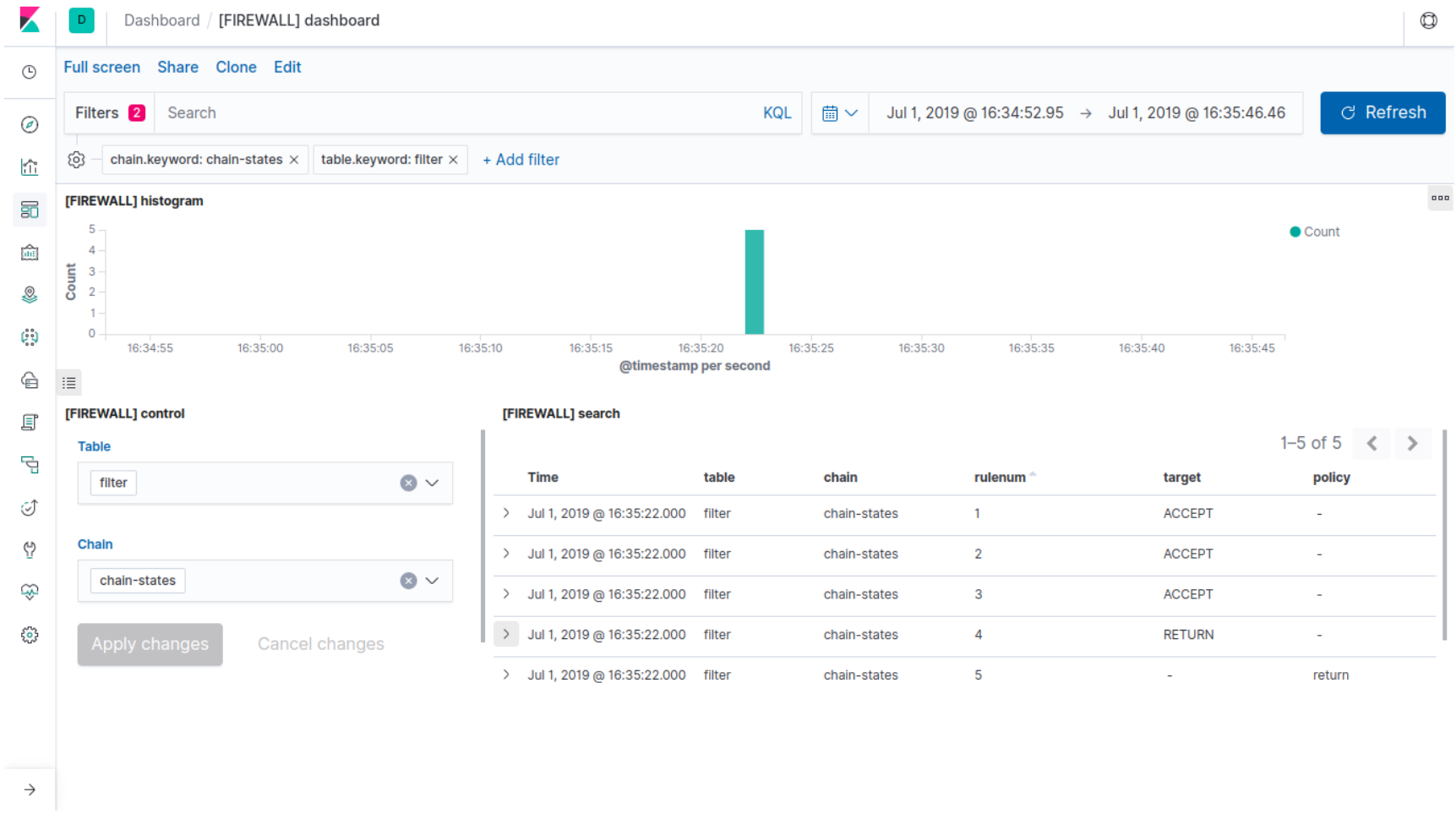


Fig. 5.3. Configuración de una cadena personalizada.

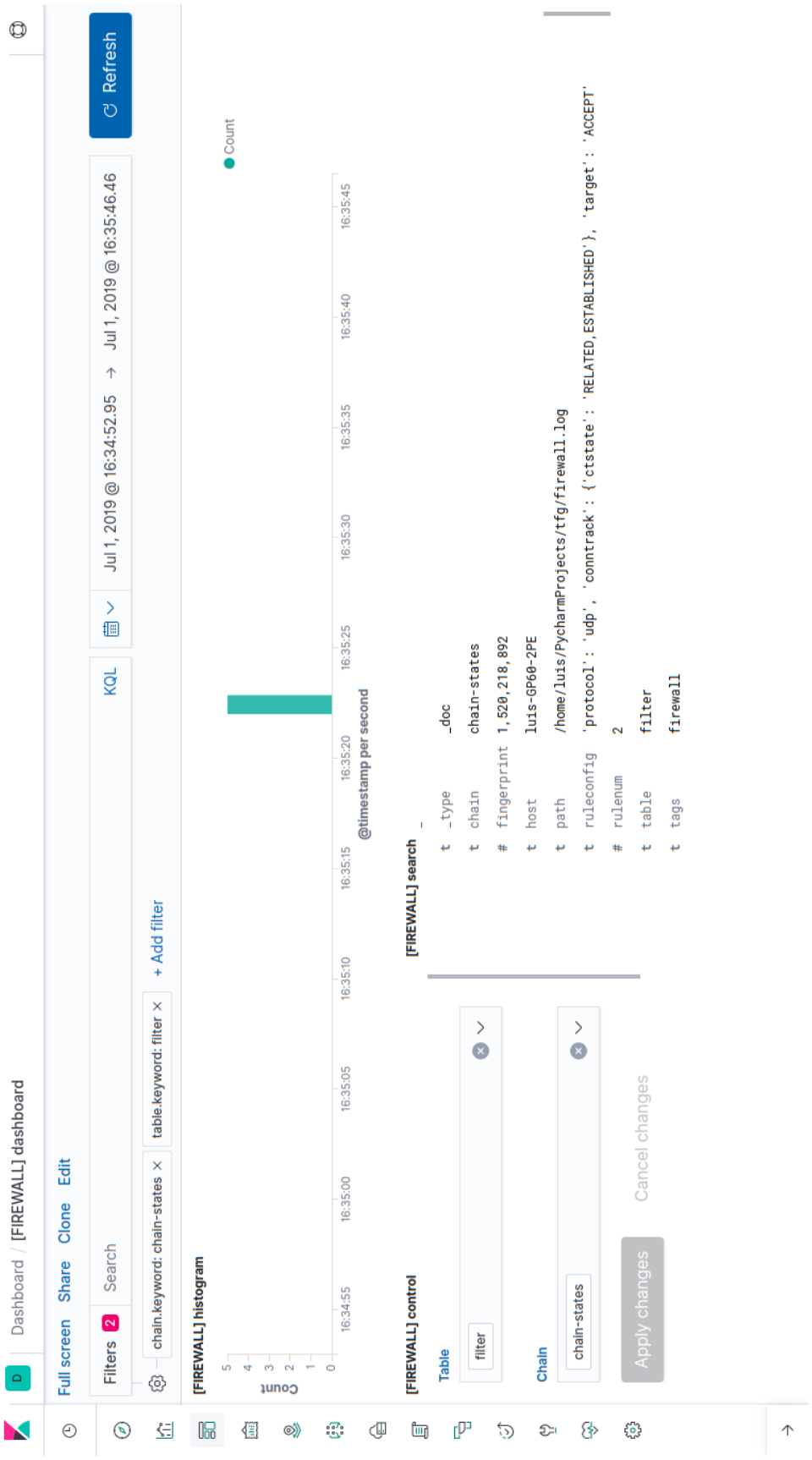


Fig. 5.4. Configuración de una regla en una cadena personalizada.

```
target     prot opt source               destination
           icmp --  anywhere             anywhere

root@host:~$ iptables -t nat -L POSTROUTING
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
           icmp --  anywhere             anywhere

root@host:~$ iptables -t filter -L OUTPUT
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
           icmp --  anywhere             anywhere

root@host:~$ iptables -t security -L OUTPUT
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
           icmp --  anywhere             anywhere

root@host:~$ iptables -t mangle -L POSTROUTING
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  anywhere             anywhere

root@host:~$ iptables -t nat -L POSTROUTING
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
           icmp --  anywhere             anywhere
```

Se ejecuta el sistema completo y se lanza un *ping sudo ping -c 1 www.tel.uva.es*. Desde Kibana, en el *dashboard* de monitorización de paquetes, filtrando por IP de destino a 157.88.129.91 en un intervalo de tiempo lo suficiente corto, se puede observar perfectamente la ruta seguida por el paquete ICMP a través de las tablas, representándose de forma ascendente. Como se comprueba en la figura 5.5, la última regla en la que se tiene constancia del paquete es en *mangle POSTROUTING* 1, la que se ha añadido con el *target* DROP.

Tráfico TCP

La segunda prueba va a realizarse con tráfico TCP, con origen y destino *localhost* y puerto de destino 2000. Por lo tanto, en este caso la ruta de los paquetes será:

```
raw OUTPUT -> mangle OUTPUT -> nat OUTPUT -> nat OUTPUT -> filter
    OUTPUT -> security OUTPUT -> mangle POSTROUTING -> raw
    PREROUTING -> mangle PREROUTING -> mangle INPUT -> filter INPUT
    -> security INPUT -> nat INPUT
```

Mediante la creación de las siguientes reglas, el descartado de tráfico se producirá en

Fig. 5.5. Seguimiento de un paquete ICMP por el cortafuegos.

mangle PREROUTING:

```
iptables -t raw -A PREROUTING -j TRACE
iptables -t raw -A OUTPUT -j TRACE
iptables -t raw -A OUTPUT -p tcp --dport 2000
iptables -t nat -A OUTPUT -p tcp --dport 2000
iptables -t mangle -A POSTROUTING -p tcp --dport 2000
iptables -t raw -A PREROUTING -p tcp --dport 2000
iptables -t mangle -A PREROUTING -p tcp --dport 2000 -j DROP
iptables -t mangle -A INPUT -p tcp --dport 2000
```

La configuración del cortafuegos queda correctamente configurada de la siguiente manera:

```
root@host:~$ iptables -t raw -L OUTPUT -n
Chain OUTPUT (policy ACCEPT)
target      prot opt source         destination
TRACE       all  --  0.0.0.0/0      0.0.0.0/0
            tcp  --  0.0.0.0/0      0.0.0.0/0       tcp dpt:2000

root@host:~$ iptables -t nat -L OUTPUT -n
Chain OUTPUT (policy ACCEPT)
target      prot opt source         destination
            tcp  --  0.0.0.0/0      0.0.0.0/0       tcp dpt:2000

root@host:~$ iptables -t mangle -L POSTROUTING -n
Chain POSTROUTING (policy ACCEPT)
target      prot opt source         destination
            tcp  --  0.0.0.0/0      0.0.0.0/0       tcp dpt:2000

root@host:~$ iptables -t raw -L PREROUTING -n
Chain PREROUTING (policy ACCEPT)
target      prot opt source         destination
TRACE       all  --  0.0.0.0/0      0.0.0.0/0
            tcp  --  0.0.0.0/0      0.0.0.0/0       tcp dpt:2000

root@host:~$ iptables -t mangle -L PREROUTING -n
Chain PREROUTING (policy ACCEPT)
target      prot opt source         destination
DROP        tcp  --  0.0.0.0/0      0.0.0.0/0       tcp dpt:2000

root@host:~$ iptables -t mangle -L INPUT -n
Chain INPUT (policy ACCEPT)
target      prot opt source         destination
            tcp  --  0.0.0.0/0      0.0.0.0/0       tcp dpt:2000
```

Para generar el tráfico UDP en este caso se utiliza el comando *nping sudo nping -c 1 –TCP -p 2000 localhost*.

De nuevo, en la figura 5.6 se puede seguir el paquete, ordenado de abajo a arriba, y

comprobar dónde se ha descartado.

Tráfico UDP

Para la última prueba el tráfico será UDP, el cual se genera desde otro equipo que tendrá configurado la IP del equipo en pruebas como router predeterminado. El destino será el puerto 10000 y una IP externa para que se produzca reenvío.

La ruta que van a seguir estos paquetes es:

```
raw PREROUTING -> mangle PREROUTING -> nat PREROUTING -> mangle
   FORWARD -> filter FORWARD -> security FORWARD -> mangle
   POSTROUTING -> nat POSTROUTING
```

Con la creación de las siguientes reglas, el tráfico se descartará en *mangle POSTROUTING*:

```
iptables -t raw -A PREROUTING ! -i lo -j TRACE
iptables -t raw -A PREROUTING -p udp --dport 10000
iptables -t nat -A PREROUTING -p udp --dport 10000
iptables -t mangle -A FORWARD -p udp --dport 10000
iptables -t filter -A FORWARD -p udp --dport 10000
iptables -t mangle -A POSTROUTING -p udp --dport 10000 -j DROP
iptables -t nat -A POSTROUTING -p udp --dport 10000
```

Nuevamente se comprueba que la configuración del cortafuegos es correcta.

```
root@host:~$ iptables -t raw -L PREROUTING -n
Chain PREROUTING (policy ACCEPT)
target      prot opt source         destination
TRACE       all  --  0.0.0.0/0      0.0.0.0/0
            udp  --  0.0.0.0/0      0.0.0.0/0       udp dpt:10000

root@host:~$ iptables -t nat -L PREROUTING -n
Chain PREROUTING (policy ACCEPT)
target      prot opt source         destination
            udp  --  0.0.0.0/0      0.0.0.0/0       udp dpt:10000

root@host:~$ iptables -t mangle -L FORWARD -n
Chain FORWARD (policy ACCEPT)
target      prot opt source         destination
            udp  --  0.0.0.0/0      0.0.0.0/0       udp dpt:10000

root@host:~$ iptables -t filter -L FORWARD -n
Chain FORWARD (policy ACCEPT)
target      prot opt source         destination
            udp  --  0.0.0.0/0      0.0.0.0/0       udp dpt:10000

```

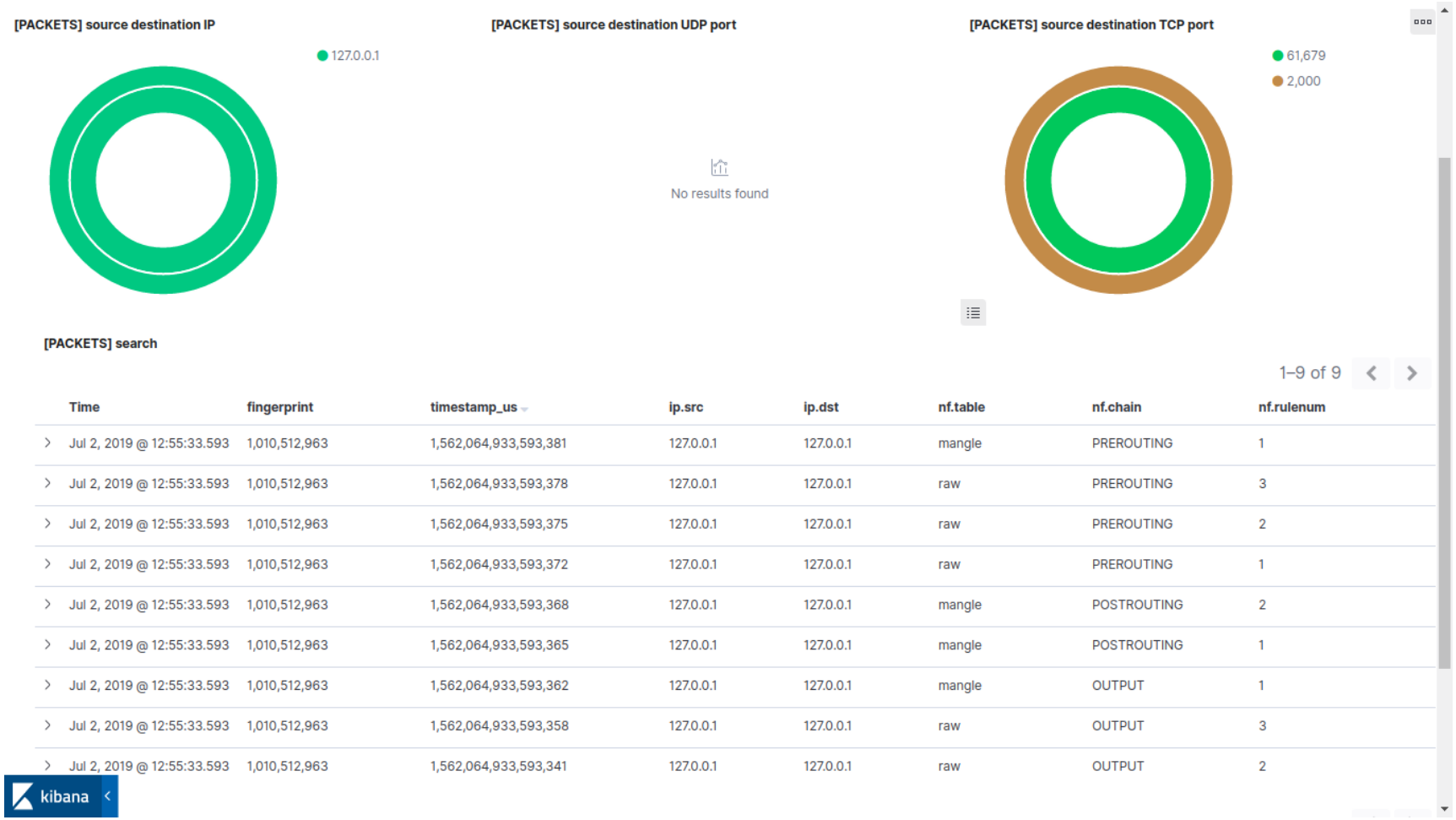


Fig. 5.6. Seguimiento de un paquete TCP por el cortafuegos.

```
root@host:~$ iptables -t mangle -L POSTROUTING -n
Chain POSTROUTING (policy ACCEPT)
target      prot opt source         destination
DROP        udp  --  0.0.0.0/0      0.0.0.0/0       udp dpt:10000

root@host:~$ iptables -t nat -L POSTROUTING -n
Chain POSTROUTING (policy ACCEPT)
target      prot opt source         destination
            udp  --  0.0.0.0/0      0.0.0.0/0       udp dpt:10000
```

El tráfico se genera con el comando *nping –udp -p 10000 -c 1 –dest-mac d0:7e:35:f7:93:ec www.tel.uva.es*, realizándose en *Kibana* un filtrado por puerto UDP de destino. Una vez más, puede verse en la figura 5.7 ordenadas de forma ascendete, con qué tablas y reglas ha coincidido el paquete y que, efectivamente, se descarta en la regla 1 de *mangle POSTROUTING*. Cabe destacar que en la tabla *mangle FORWARD* se añadió una regla que luego se eliminó, motivo por el cual aparece su política a la hora de monitorizar. Esto es debido a que las tablas constituyen módulos independientes del *kernel*, y en caso de que no se hayan usado desde el arranque del sistema, no se cargan.

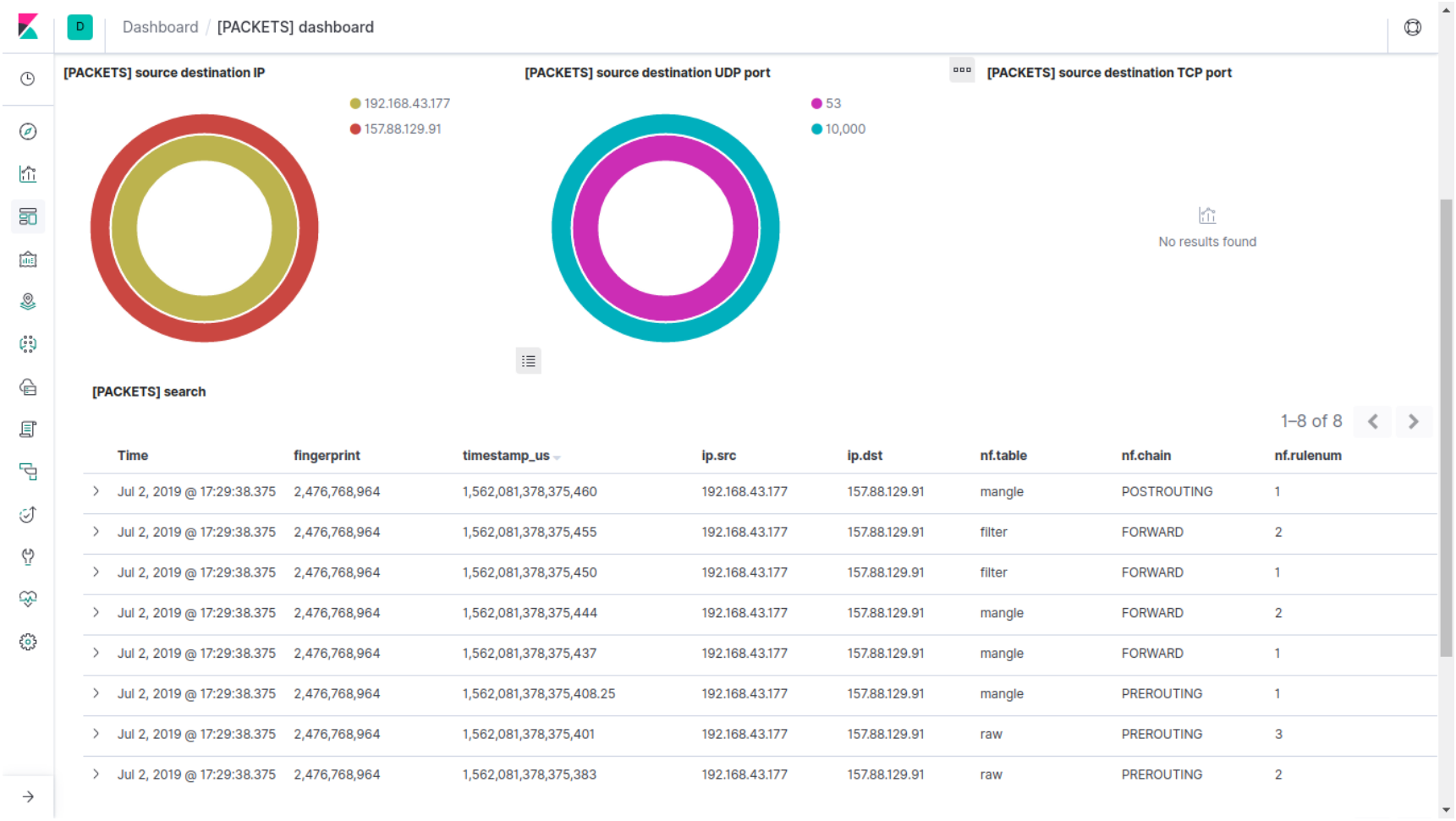


Fig. 5.7. Seguimiento de un paquete UDP por el cortafuegos.

6. CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO

A lo largo de este capítulo se comentarán las conclusiones extraídas de la realización del Trabajo de Fin de Grado, tanto del sistema de monitorización del cortafuegos como del sistema de monitorización de paquetes. Se analizarán posteriormente los principales logros y puntos de mejora del sistema para finalizar proponiendo algunas posibles líneas con las que continuar el desarrollo de ambos sistemas.

6.1. Conclusiones

La realización de este trabajo ha supuesto un reto en diversos aspectos, desde la dificultad que supone abordar un proyecto de forma individual, hasta el empleo de nuevas tecnologías y su correspondiente aprendizaje.

En primer lugar, las labores de investigación previas realizadas antes de abordar el proyecto han resultado de gran utilidad. Desde la búsqueda de soluciones para un problema existente, así como la capacidad de predecir futuras complicaciones y errores que puedan derivarse de las soluciones escogidas. La elección y uso de la pila Elastic para la realización de este Trabajo de Fin de Grado ha resultado un gran acierto, ya que ha aportado la suficiente flexibilidad como para poder ir afrontando las distintas complicaciones que han aparecido a lo largo del desarrollo sin tener que cambiar de herramienta. Asimismo, Elastic ha permitido orientar el trabajo desde un punto de vista menos centrado en la ingeniería de software y más dirigido a administración de sistemas.

Otro gran aspecto extraído de la realización de este Trabajo de Fin de Grado ha sido la necesidad de búsqueda de soluciones acordes a la cantidad de tiempo disponible para la realización de un proyecto, motivo por el cual no ha sido viable abordar una posible modificación del *kernel* de Linux, así como la elección del lenguaje de programación Python por su corta curva de aprendizaje. La pila Elastic también ha demostrado ser una herramienta que aporta soluciones efectivas con un tiempo de implementación corto.

Por último, gracias al uso del cortafuegos IPTables presente en el núcleo de Linux, se han adquirido una serie de conocimientos bastante avanzados tanto a nivel técnico, sobre su funcionamiento y sus características, como a nivel cultural, señalando la serie de acontecimientos que han surgido a lo largo de la historia en el campo de administración de sistemas.

6.2. Logros y puntos débiles

Se analizan en este apartado tanto los aspectos positivos como los que resulta necesario mejorar en ambos sistemas de monitorización.

Logros

- El sistema de monitorización del cortafuegos permite ver la configuración de cualquier regla, independientemente del número de *matches* y el *target* que tenga.
- El sistema de monitorización del cortafuegos permite ver las reglas de cadenas personalizadas.
- El sistema de monitorización de paquetes permite rastrear el paso de los paquetes por cualquier regla de cualquier cadena, tanto predeterminada como personalizada.
- El sistema de monitorización de paquetes permite identificar de forma única los paquetes con el contenido de sus cabeceras.
- Ambos sistemas funcionan utilizando la pila Elastic, por lo que pueden ser desplegados sobre cualquier instalación existente de la misma o sobre la nube.
- Ambos sistemas permiten funcionar en máquinas distintas a la que se monitoriza.
- El sistema de monitorización del cortafuegos permite complementar al sistema de monitorización de paquetes en el caso de que una regla altere información de las cabeceras del paquete y cambie su identificador único.
- Ambos sistemas permiten seleccionar en todo momento el intervalo de tiempo que se desee analizar.
- Ambos sistemas permiten almacenar los datos para poder ser analizados posteriormente.

Puntos débiles

- El sistema de monitorización de paquetes no permite distinguir de forma satisfactoria dos paquetes con todos sus campos de las cabeceras idénticos si se producen en el mismo instante de tiempo.
- Ambos sistemas están limitados por las especificaciones tanto de procesamiento como de velocidad de escritura en disco de la máquina a monitorizar.
- El sistema de monitorización de paquetes presenta limitaciones en el análisis de protocolos de nivel de transporte distintos de ICMP, TCP y UDP.

6.3. Futuras líneas de trabajo

Existen ciertas líneas futuras que podrían explorarse para seguir con el desarrollo de los sistemas creados en este Trabajo de Fin de Grado.

Traslado de los sistemas a un entorno real

Sería interesante desplegar ambos sistemas en un entorno real, en el que exista un cortafuegos IPTables configurado con multitud de reglas y tablas y un cierto volumen de tráfico. Permitiría comprobar el funcionamiento y utilidad de ambos.

Pruebas de rendimiento

Convendría realizar pruebas de estrés al sistema de monitorización de paquetes para determinar de forma cuantitativa el volumen de tráfico que es capaz de monitorizar. De esta forma podrían abordarse soluciones que permitan mejorarlo.

Nuevas funcionalidades de los sistemas

Podrían establecerse nuevos objetivos y funcionalidades para ambos sistemas como, por ejemplo, permitir ver en el sistema de monitorización de paquetes también la configuración de las reglas que pasan los paquetes. También podrían implementarse nuevas visualizaciones que permitan simplificar el proceso de filtrado de paquetes. En último lugar, se podría abandonar el uso de Kibana si no cumple con los requisitos, y desarrollar una interfaz personalizada utilizando las APIs de Elasticsearch.

Creación de un Beat comunitario

Con la construcción de un Beat comunitario como se vio en la sección 3.6.2, podría simplificarse todo el proceso de instalación en el cortafuegos a monitorizar, así como el despliegue en nuevos cortafuegos. De esta forma, el sistema de monitorización se integraría aún más en la pila Elastic.

Actualización a NFTables

Con el paso del tiempo el cortafuegos predeterminado del *Kernel* de Linux se irá sustituyendo paulatinamente por el nuevo NFTables, también del proyecto Netfilter [101]. Esta nueva generación del cortafuegos ya empieza a ser la predeterminada en distribuciones GNU/Linux como Debian 10 [102]. Es por ello que acabará siendo necesario la actualización de ambos sistemas de monitorización para hacerlos compatibles con NFTables.

BIBLIOGRAFÍA

[1] S. Ligus, *Effective Monitoring and Alerting*. O'Reilly Media, Inc., 2013.

[2] Brian Komar, Ronald Beekelaar, Joern Wettern, *Firewalls For Dummies*. Wiley Publishing, Inc, 2003.

[3] Verizon, «2019 Data Breach Investigations Report», 2019. [En línea]. Disponible en: https://enterprise.verizon.com/resources/reports/dbir/.

[4] Ed Cabrera, Robert McArdle, and the U.S. Secret Service, «The Evolution of Cybercrime and Cyberdefense», 2019. [En línea]. Disponible en: https://www.trendmicro.com/vinfo/nl/security/news/cybercrime-and-digital-threats/evolution-of-cybercrime#downloadPDF.

[5] The netfilter.org project. (1 de ago. de 2011). What is netfilter.org?, [En línea]. Disponible en: https://www.netfilter.org/.

[6] Wes Noonan, Ido Dubrawsky, *Firewall Fundamentals*. Cisco Press, 2006.

[7] W. Richards Adrion, *Research Methodology in Software Engineering*. ACM Software Engineering Notes, SIGSoft, 1993.

[8] Ingham, Kenneth; Forrest, Stephanie. (1 de ago. de 2011). A History and Survey of Network Firewalls, [En línea]. Disponible en: https://www.cs.unm.edu/~treport/tr/02-12/firewall.pdf.

[9] Cisco Systems Inc. (1 de ago. de 2011). ¿Qué es un firewall?, [En línea]. Disponible en: https://www.cisco.com/c/es_es/products/security/firewalls.

[10] P. Srisuresh y K. Egevang, «Traditional IP Network Address Translator (Traditional NAT)», RFC Editor, RFC 3022, ene. de 2001.

[11] The netfilter.org project. (15 de ago. de 2011). About the netfilter/iptables project, [En línea]. Disponible en: https://netfilter.org/about.html#history.

[12] Linux Man. (1 de ago. de 2011). iptables(8) - Linux man page, [En línea]. Disponible en: https://linux.die.net/man/8/iptables.

[13] Phil Hagen. (23 de jul. de 2019). iptables Processing Flowchart, [En línea]. Disponible en: https://stuffphilwrites.com/2014/09/iptables-processing-flowchart/.

[14] The netfilter.org project. (1 de ago. de 2011). Packet Filtering HOWTO, [En línea]. Disponible en: https://netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.txt.

[15] Elastic NV. (14 de jun. de 2019). The Elastic Stack, [En línea]. Disponible en: https://www.elastic.co/es/products/.

[16] ——, (14 de jun. de 2019). LOGGING Solutions, [En línea]. Disponible en: https://www.elastic.co/es/solutions/logging.

[17] MongoDB, Inc. (23 de jul. de 2019). What Is A Non Relational Database, [En línea]. Disponible en: https://www.mongodb.com/scale/what-is-a-non-relational-database.

[18] Open Source Initiative. (14 de jun. de 2019). The Open Source Definition, [En línea]. Disponible en: https://opensource.org/osd.

[19] The Apache Software Foundation. (16 de jun. de 2019). APACHE LICENSES, [En línea]. Disponible en: http://www.apache.org/licenses/.

[20] Elastic NV. (16 de jun. de 2019). ELASTIC LICENSE AGREEMENT, [En línea]. Disponible en: https://github.com/elastic/elasticsearch/blob/master/licenses/ELASTIC-LICENSE.txt.

[21] ——, (16 de jun. de 2019). Our Story, [En línea]. Disponible en: https://www.elastic.co/es/about/history-of-elasticsearch.

[22] ——, (16 de jun. de 2019). ELASTIC CLOUD, [En línea]. Disponible en: https://www.elastic.co/es/cloud/.

[23] ——, (16 de jun. de 2019). Logstash Introduction, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/introduction.html.

[24] ——, (14 de jun. de 2019). LOGSTASH The Elastic Stack, [En línea]. Disponible en: https://www.elastic.co/es/products/logstash.

[25] ——, (16 de jun. de 2019). Logstash Input plugins, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/input-plugins.html.

[26] ——, (16 de jun. de 2019). Logstash Filter plugins, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/filter-plugins.html.

[27] ——, (16 de jun. de 2019). Logstash Output plugins, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/output-plugins.html.

[28] ——, (18 de jun. de 2019). You know, for search (and analysis), [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html#elasticsearch-intro.

[29] ——, (18 de jun. de 2019). Basic Concepts, [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started-concepts.html.

[30] json.org. (18 de jun. de 2019). Introducing JSON, [En línea]. Disponible en: https://www.json.org/.

[31] MongoDB, Inc. (23 de jul. de 2019). Document Databases, [En línea]. Disponible en: https://www.mongodb.com/document-databases?lang=en-en.

[32] Elastic NV. (18 de jun. de 2019). Data in: documents and indices, [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html.

[33] John Robinson, *The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes*. SIGMOD '81 Proceedings of the 1981 ACM SIGMOD international conference on Management of data, 1981.

[34] The Apache Software Foundation. (18 de jun. de 2019). Welcome to Apache Lucene, [En línea]. Disponible en: https://lucene.apache.org/.

[35] Roy Thomas Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine, 2000. [En línea]. Disponible en: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[36] Elastic NV. (18 de jun. de 2019). Elasticsearch Clients, [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/client/index.html.

[37] ——, (18 de jun. de 2019). Community Contributed Clients, [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/client/community/current/index.html.

[38] ——, (18 de jun. de 2019). Query DSL, [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html.

[39] ——, (18 de jun. de 2019). SQL access, [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/current/xpack-sql.html.

[40] ——, (18 de jun. de 2019). Machine learning in the Elastic Stack, [En línea]. Disponible en: https://www.elastic.co/guide/en/elastic-stack-overview/7.1/xpack-ml.html.

[41] ——, (19 de jun. de 2019). Kibana Introduction, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/introduction.html.

[42] ——, (19 de jun. de 2019). Connect Kibana with Elasticsearch, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/connect-to-elasticsearch.html.

[43] ——, (19 de jun. de 2019). Kibana Discover, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/discover.html.

[44] ——, (19 de jun. de 2019). Kibana Visualize, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/visualize.html.

[45] ——, (19 de jun. de 2019). Kibana Creating a Visualization, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/createvis.html.

[46] D. Fireball. (19 de jun. de 2019). Markdown, [En línea]. Disponible en: https://daringfireball.net/projects/markdown/.

[47] UW Interactive Data Lab. (19 de jun. de 2019). Vega and Vega-Lite, [En línea]. Disponible en: https://vega.github.io/.

[48] Elastic NV. (19 de jun. de 2019). Kibana Dashboard, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/dashboard.html.

[49] ——, (19 de jun. de 2019). Kibana Canvas, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/canvas.html.

[50] ——, (19 de jun. de 2019). Canvas Common Functions, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/canvas-common-functions.html.

[51] ——, (19 de jun. de 2019). Kibana User Guide, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/index.html.

[52] ——, (20 de jun. de 2019). Beats Overview, [En línea]. Disponible en: https://www.elastic.co/guide/en/beats/libbeat/current/beats-reference.html.

[53] Linux Man. (20 de jun. de 2019). auditd(8) - Linux man page, [En línea]. Disponible en: https://linux.die.net/man/8/auditdt.

[54] Elastic NV. (20 de jun. de 2019). Auditbeat, [En línea]. Disponible en: https://www.elastic.co/es/products/beats/auditbeat.

[55] ——, (20 de jun. de 2019). Filebeat, [En línea]. Disponible en: https://www.elastic.co/es/products/beats/filebeat.

[56] ——, (20 de jun. de 2019). Functionbeat, [En línea]. Disponible en: https://www.elastic.co/es/products/beats/functionbeat.

[57] ——, (20 de jun. de 2019). Heartbeat, [En línea]. Disponible en: https://www.elastic.co/es/products/beats/heartbeat.

[58] ——, (20 de jun. de 2019). Metricbeat, [En línea]. Disponible en: https://www.elastic.co/es/products/beats/metricbeat.

[59] ——, (20 de jun. de 2019). Metricbeat Modules, [En línea]. Disponible en: https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-modules.html.

[60] ——, (20 de jun. de 2019). Packetbeat, [En línea]. Disponible en: https://www.elastic.co/es/products/beats/packetbeat.

[61] ——, (20 de jun. de 2019). Winlogbeat, [En línea]. Disponible en: https://www.elastic.co/es/products/beats/winlogbeat.

[62] ——, (20 de jun. de 2019). Community Beats, [En línea]. Disponible en: https://www.elastic.co/guide/en/beats/libbeat/current/community-beats.html.

[63] Universidad de Valladolid. (23 de jul. de 2019). UVaDOC Repositorio Documental de la Universidad de Valladolid, [En línea]. Disponible en: https://uvadoc.uva.es/.

[64] Elastic NV. (23 de jul. de 2019). Stories from Users Like You, [En línea]. Disponible en: https://www.elastic.co/use-cases/.

[65] Linux Man. (8 de jul. de 2019). iptables-extensions, [En línea]. Disponible en: http://ipset.netfilter.org/iptables-extensions.man.html.

[66] ——, (25 de jul. de 2019). netlink - communication between kernel and user space (AF NETLINK), [En línea]. Disponible en: http://man7.org/linux/man-pages/man7/netlink.7.html.

[67] KimiNewt. (21 de jun. de 2019). PyShark. Python packet parser using wireshark's tshark, [En línea]. Disponible en: http://kiminewt.github.io/pyshark/.

[68] Wireshark Foundation. (21 de jun. de 2019). tshark - Dump and analyze network traffic, [En línea]. Disponible en: https://www.wireshark.org/docs/man-pages/tshark.html/.

[69] ——, (21 de jun. de 2019). Wireshark, [En línea]. Disponible en: https://www.wireshark.org/.

[70] Python Software Foundation. (21 de jun. de 2019). logging — Logging facility for Python, [En línea]. Disponible en: https://docs.python.org/3/library/logging.html.

[71] ——, (21 de jun. de 2019). Logging HOWTO, [En línea]. Disponible en: https://docs.python.org/3/howto/logging.html#logging-basic-tutorial.

[72] ——, (21 de jun. de 2019). Logging Cookbook, [En línea]. Disponible en: https://docs.python.org/3/howto/logging-cookbook.html#logging-cookbook.

[73] J. Postel, «Internet Protocol», RFC Editor, STD 5, sep. de 1981, http://www.rfc-editor.org/rfc/rfc791.txt. [En línea]. Disponible en: http://www.rfc-editor.org/rfc/rfc791.txt.

[74] J. Postel, «User Datagram Protocol», RFC Editor, STD 6, ago. de 1980, http://www.rfc-editor.org/rfc/rfc768.txt. [En línea]. Disponible en: http://www.rfc-editor.org/rfc/rfc768.txt.

[75] J. Postel, «Transmission Control Protocol», RFC Editor, STD 7, sep. de 1981, http://www.rfc-editor.org/rfc/rfc793.txt. [En línea]. Disponible en: http://www.rfc-editor.org/rfc/rfc793.txt.

[76] J. Postel, «Internet Control Message Protocol», RFC Editor, STD 5, sep. de 1981, http://www.rfc-editor.org/rfc/rfc792.txt. [En línea]. Disponible en: http://www.rfc-editor.org/rfc/rfc792.txt.

[77] International Organization for Standardization. (23 de jun. de 2019). Date and time format - ISO 8601, [En línea]. Disponible en: https://www.iso.org/iso-8601-date-and-time-format.html.

[78] Python Software Foundation. (23 de jun. de 2019). RotatingFileHandler, [En línea]. Disponible en: https://docs.python.org/3/library/logging.handlers.html#logging.handlers.RotatingFileHandler.

[79] LSB Workgroup, The Linux Foundation. (23 de jun. de 2019). Filesystem Hierarchy Standard, [En línea]. Disponible en: https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html.

[80] Vilmos Nebehaj. (23 de jun. de 2019). Welcome to python-iptables's documentation!, [En línea]. Disponible en: http://ldx.github.io/python-iptables/index.html.

[81] Elastic NV. (23 de jun. de 2019). Install Elasticsearch with Debian Package, [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/7.1/deb.html#deb-repo.

[82] ——, (23 de jun. de 2019). Installing Elasticsearch Yourself, [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/7.1/install-elasticsearch.html.

[83] ——, (23 de jun. de 2019). Filebat Configure inputs, [En línea]. Disponible en: https://www.elastic.co/guide/en/beats/filebeat/current/configuration-filebeat-options.html.

[84] ——, (23 de jun. de 2019). Configure the Logstash output, [En línea]. Disponible en: https://www.elastic.co/guide/en/beats/filebeat/current/logstash-output.html.

[85] ——, (26 de jun. de 2019). Logastash Multiple Pipelines, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/multiple-pipelines.html.

[86] ——, (23 de jun. de 2019). Logastash Beats input plugin, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-inputs-beats.html.

[87] ——, (). Logastash File input plugin, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-inputs-file.htmll.

[88] ——, (25 de jun. de 2019). Logastash Grok filter plugin, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html.

[89] ——, (25 de jun. de 2019). Logastash Date filter plugin, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-date.html.

[90] ——, (25 de jun. de 2019). Logastash Fingerprint filter plugin, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-fingerprint.html.

[91] ——, (25 de jun. de 2019). Logastash Dissect filter plugin, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-dissect.html.

[92] ——, (25 de jun. de 2019). Logastash Ruby filter plugin, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-ruby.html.

[93] Ruby community. (25 de jun. de 2019). Ruby Programming Language, [En línea]. Disponible en: https://www.ruby-lang.org/en/.

[94] Elastic NV. (25 de jun. de 2019). Logastash Mutate filter plugin, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-mutate.html.

[95] ——, (25 de jun. de 2019). Logastash Elasticsearch filter plugin, [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-outputs-elasticsearch.html.

[96] ——, (26 de jun. de 2019). Kibana Line, Area, and Bar charts, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/xy-chart.html.

[97] ——, (26 de jun. de 2019). Kibana Controls Visualization, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/controls.html.

[98] ——, (26 de jun. de 2019). Kibana Saving and Opening Searches, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/save-open-search.html.

[99] ——, (26 de jun. de 2019). Kibana Pie Charts, [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/pie-chart.html.

[100] NMAP.org. (1 de jul. de 2019). Nping - Network packet generation tool / ping utiliy - Nmap, [En línea]. Disponible en: https://nmap.org/nping/.

[101] The netfilter.org projec. (8 de jul. de 2019). The netfilter.org "nftables"project, [En línea]. Disponible en: https://netfilter.org/projects/nftables/index.html.

[102] Debian. (8 de jul. de 2019). Release Notes for Debian 10 (buster), 64-bit PC, [En línea]. Disponible en: https://www.debian.org/releases/stable/amd64/release-notes/index.en.html.

A. ANEXO 1

Fichero de configuración de una tubería en Logstash

```
input {
#  stdin {
#    tags => ["packet"]
#    tags => ["firewall"]
#  }
#  file {
#    path => ["/home/luis/PycharmProjects/tfg/tablas.log"]
#    tags => ["firewall"]
#    start_position => "end"
#  }
#  file {
#    path => ["/home/luis/PycharmProjects/tfg/paquetes.log"]
#    tags => ["packet"]
#    start_position => "end"
#  }

  beats {
    port => 5044
  }

}

filter {

  if "packet" in [tags] {

  dissect {
    mapping => {
      "message" => "%{sniff_timestamp} %{nflog_prefix} %{
          transport_protocol} %{highest_layer} %{[ip][dsfield][dscp
          ]} %{[ip][dsfield][ecn]} %{[ip][id]} %{[ip][flags][df]}
          %{[ip][flags][mf]} %{[ip][frag]} %{[ip][ttl]} %{[ip][src]}
           %{[ip][dst]} %{rest}"
    }
    remove_field => "message"
  }

  ruby {
#    init => "require 'time'"
    code => "
      begin
        t = Time.parse(event.get('sniff_timestamp'))
        event.set('timestamp_us', t.to_i * 1000000 + t.usec)
```

```
      rescue Exception => e
        event.set('ruby_exception', e.message)
      end
      "
  }

#  ruby {
#    code => "
#      t = Time.parse(event['sniff_timestamp'], '%Y-%m-%d %H:%M:%S.%L')
#      event['timestamp_us'] = t.to_i * 1000000 + t.usec
#    "
#  }

  mutate {
    gsub => ["sniff_timestamp","\d\d\d$",""]
  }

  date {
    match => [ "sniff_timestamp", "ISO8601" ]
    remove_field => [ "sniff_timestamp" ]
  }


  mutate {
    convert => {
      "timestamp_us" => "integer"
      "[ip][dsfield][dscp]" => "integer"
      "[ip][dsfield][ecn]" => "integer"

      "[ip][flags][df]" => "boolean"
      "[ip][flags][mf]" => "boolean"
      "[ip][frag]" => "integer"
      "[ip][ttl]" => "integer"
    }
  }

  dissect {
    mapping => {
      "nflog_prefix" => "%{[nf][table]}:%{[nf][chain]}:%{[nf][match]}:%{[nf][rulenum]}"
    }
  }
  if [nf][match] == 'policy' {
    mutate {
      add_field => { "[nf][policy]" => "true" }
      convert => { "[nf][policy]" => "boolean" }
    }
  }
  mutate {
    remove_field => [ "nflog_prefix", "[nf][match]" ]
```

```
89        convert => { "[nf][rulenum]" => "integer" }
90      }
91
92      if [transport_protocol] == 'tcp' {
93
94        dissect {
95          mapping => {
96            "rest" => "%{[tcp][srcport]} %{[tcp][dstport]} %{[tcp][seq]}
                  %{[tcp][ack]} %{[tcp][flags][ns]} %{[tcp][flags][cwr]}
                 %{[tcp][flags][ece]} %{[tcp][flags][urg]} %{[tcp][flags
                 ][ack]} %{[tcp][flags][psh]} %{[tcp][flags][rst]} %{[tcp
                 ][flags][syn]} %{[tcp][flags][fin]} %{[tcp][window_size
                 ]} %{[tcp][urgent_pinter]}"
97          }
98          remove_field => "rest"
99        }
100
101       mutate {
102         convert => {
103           "[tcp][srcport]" => "integer"
104           "[tcp][dstport]" => "integer"
105           "[tcp][seq]" => "integer"
106           "[tcp][ack]" => "integer"
107           "[tcp][flags][ns]" => "boolean"
108           "[tcp][flags][cwr]" => "boolean"
109           "[tcp][flags][ece]" => "boolean"
110           "[tcp][flags][urg]" => "boolean"
111           "[tcp][flags][ack]" => "boolean"
112           "[tcp][flags][psh]" => "boolean"
113           "[tcp][flags][rst]" => "boolean"
114           "[tcp][flags][syn]" => "boolean"
115           "[tcp][flags][fin]" => "boolean"
116           "[tcp][window_size]" => "integer"
117           "[tcp][urgent_pinter]" => "integer"
118         }
119       }
120
121       fingerprint {
122         source => [ "transport_protocol", "[ip][dsfield]", "[ip][id]",
                "[ip][flags]", "[ip][frag]", "[ip][src]", "[ip][dst]", "[
               tcp]" ]
123         target => "fingerprint"
124         method => "MURMUR3"
125         concatenate_sources => true
126       }
127
128
129     } else if [transport_protocol] == 'udp' {
130
131       dissect {
132         mapping => {
```

```
133           "rest" => "%{[udp][srcport]} %{[udp][dstport]}"
134         }
135         remove_field => "rest"
136       }
137
138       mutate {
139         convert => {
140           "[udp][srcport]" => "integer"
141           "[udp][dstport]" => "integer"
142         }
143       }
144
145       fingerprint {
146         source => [ "transport_protocol", "[ip][dsfield]", "[ip][id]",
                "[ip][flags]", "[ip][frag]", "[ip][src]", "[ip][dst]", "[
               udp]" ]
147         target => "fingerprint"
148         method => "MURMUR3"
149         concatenate_sources => true
150       }
151
152     } else if [transport_protocol] == 'icmp' {
153
154       dissect {
155         mapping => {
156           "rest" => "%{[icmp][type]} %{[icmp][code]}"
157         }
158         remove_field => "rest"
159       }
160
161       mutate {
162         convert => {
163           "[icmp][type]" => "integer"
164           "[icmp][code]" => "integer"
165         }
166       }
167
168
169       fingerprint {
170         source => [ "transport_protocol", "[ip][dsfield]", "[ip][id]",
                "[ip][flags]", "[ip][frag]", "[ip][src]", "[ip][dst]", "[
               icmp]" ]
171         target => "fingerprint"
172         method => "MURMUR3"
173         concatenate_sources => true
174       }
175
176     } else if [transport_protocol] == 'other' {
177
178       mutate {
179         remove_field => "rest"
```

```
180       }
181
182       fingerprint {
183         source => [ "transport_protocol", "[ip][dsfield]", "[ip][id]",
                "[ip][flags]", "[ip][frag]", "[ip][src]", "[ip][dst]" ]
184         target => "fingerprint"
185         method => "MURMUR3"
186         concatenate_sources => true
187       }
188
189     }
190
191
192     } else if "firewall" in [tags] {#fin seccion paquetes
193
194     grok {
195       match => {
196         "message" => [
197           "%{NOTSPACE:timestamp} Table: %{NOTSPACE:table} Chain: %{
                  NOTSPACE:chain} Rule: %{NUMBER:rulenum:int} Policy: %{
                  GREEDYDATA:policy}",
198           "%{NOTSPACE:timestamp} Table: %{NOTSPACE:table} Chain: %{
                  NOTSPACE:chain} Rule: %{NUMBER:rulenum:int} {%{
                  GREEDYDATA:ruleconfig}} Target: %{GREEDYDATA:target}"
199         ]
200       }
201       remove_field => "message"
202     }
203
204     if ![policy] {
205       if ![target] {
206         mutate { add_field => { "target" => "" } }
207       }
208     }
209
210     fingerprint {
211       source => "timestamp"
212       target => "[fingerprint]"
213       method => "MURMUR3"
214     }
215
216     date {
217       match => [ "timestamp", "ISO8601" ]
218       remove_field => [ "timestamp" ]
219     }
220
221     } #fin seccion tablas
222
223   }
224   output {
225
```

```
#  stdout {
#    codec => rubydebug { metadata => true }
#  }

  if "packet" in [tags] {
    elasticsearch {
      hosts => [ "localhost:9200" ]
      index => "index-packets"
    }
  } else if "firewall" in [tags] {
    elasticsearch {
      hosts => [ "localhost:9200" ]
      index => "index-firewall"
    }
  }

}
```

B. ANEXO 2

Resultado de la transformación de *logs* con Logstash

Log de configuración del cortafuegos

```
{
           "chain" => "PREROUTING",
      "@timestamp" => 2019-06-23T12:03:14.000Z,
         "rulenum" => 1,
        "@version" => "1",
            "host" => "luis-GP60-2PE",
     "fingerprint" => 773454870,
           "table" => "raw",
          "target" => "TRACE",
            "tags" => [
         [0] "firewall"
     ]
}
```

Log de paquete TCP

```
{
                   "tcp" => {
           "window_size" => 255,
               "dstport" => 443,
                   "ack" => 1,
                 "flags" => {
             "rst" => false,
             "psh" => false,
              "ns" => false,
             "ece" => false,
             "ack" => true,
             "syn" => false,
             "fin" => false,
             "urg" => false,
             "cwr" => false
         },
               "srcport" => 36208,
                   "seq" => 1,
         "urgent_pinter" => 0
     },
      "highest_layer" => "TCP",
       "timestamp_us" => 1561202977258328,
```

```
              "@timestamp" => 2019-06-22T11:29:37.258Z,
                     "ip" => {
         "dsfield" => {
              "ecn" => 0,
             "dscp" => 0
         },
            "frag" => 0,
             "dst" => "216.58.201.163",
             "src" => "10.133.140.252",
           "flags" => {
             "df" => true,
             "mf" => false
         },
              "id" => "0x0000dfe1",
             "ttl" => 64
     },
               "@version" => "1",
            "fingerprint" => 31537748,
                     "nf" => {
           "chain" => "OUTPUT",
         "rulenum" => 1,
           "table" => "filter"
     },
                  "host" => "luis-GP60-2PE",
     "transport_protocol" => "tcp",
                  "tags" => [
         [0] "packet"
     ]
 }
```

Log de paquete UDP

```
 {
                    "udp" => {
         "dstport" => 53,
         "srcport" => 36013
     },
          "highest_layer" => "DNS",
           "timestamp_us" => 1561202974879120,
             "@timestamp" => 2019-06-22T11:29:34.879Z,
                    "ip" => {
         "dsfield" => {
              "ecn" => 0,
             "dscp" => 0
         },
            "frag" => 0,
             "dst" => "157.88.18.190",
             "src" => "10.133.140.252",
```

```
            "flags" => {
              "df" => true,
              "mf" => false
          },
               "id" => "0x0000d0cb",
              "ttl" => 64
      },
                "@version" => "1",
           "fingerprint" => 3678516420,
                      "nf" => {
            "chain" => "OUTPUT",
          "rulenum" => 2,
            "table" => "security",
         "policy" => "true"
      },
                  "host" => "luis-GP60-2PE",
      "transport_protocol" => "udp",
                  "tags" => [
          [0] "packet"
      ]
  }
```

Log de paquete ICMP

```
  {
           "highest_layer" => "ICMP",
            "timestamp_us" => 1561202905604815,
              "@timestamp" => 2019-06-22T11:28:25.604Z,
                     "ip" => {
          "dsfield" => {
               "ecn" => 0,
              "dscp" => 0
          },
             "frag" => 0,
              "dst" => "216.58.214.163",
              "src" => "10.133.140.252",
            "flags" => {
              "df" => true,
              "mf" => false
          },
               "id" => "0x000073ec",
              "ttl" => 64
      },
                "@version" => "1",
           "fingerprint" => 1140346290,
                      "nf" => {
            "chain" => "OUTPUT",
          "rulenum" => 1,
```

```
            "table" => "mangle"
    },
                "host" => "luis-GP60-2PE",
    "transport_protocol" => "icmp",
                "icmp" => {
        "code" => 0,
        "type" => 8
    },
                "tags" => [
        [0] "packet"
    ]
}
```