

Correlador de eventos avanzado

Andrés Izquierdo Núñez

Máster universitario de Seguridad de las TIC
M1.830 - TFM-Ad hoc aula 1

Jordi Guijarro Olivares

Victor García Font

06/2018

Copyright © 2018 Andrés Izquierdo Núñez

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU Free Documentation License".

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>Correlador de eventos avanzado</i>
Nombre del autor:	<i>Andrés Izquierdo Núñez</i>
Nombre del consultor/a:	<i>Jordi Guijarro Olivares</i>
Nombre del PRA:	<i>Victor García Font</i>
Fecha de entrega:	06/2018
Titulación:	<i>Máster universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones</i>
Área del Trabajo Final:	<i>M1.830 - TFM-Ad hoc aula 1</i>
Idioma del trabajo:	Castellano
Palabras clave	<i>correlar, distribuido, elastic</i>

Resumen del Trabajo:

La finalidad de este proyecto es obtener un entorno de correlación de eventos que permita distribuir la carga entre diferentes instancias, ofreciendo una gran flexibilidad y escalabilidad.

En el contexto de la seguridad de un sistema informático nos encontramos con un gran ecosistema de dispositivos, cuyos logs y tráfico generado son importantes fuentes de información para detectar comportamientos maliciosos o anómalos. Uno de los problemas más comunes es relacionar toda esta información que se encuentra en diferentes sistemas.

Tras investigar los diferentes SIEM y correladores más conocidos y usados se llegó a la conclusión de que no existía ningún producto open source que nos ofreciera todas las características que se deseaban.

Se decidió diseñar un entorno de correlación formado por diferentes componentes que nos permiten objetivos deseados. Uno de estos componentes fue necesario desarrollarlo porque no existía ningún software que nos proporcionara las funcionalidades requeridas.

La metodología elegida fue el Proceso Unificado (RUP), el cuál nos proporciona un marco de trabajo con unas metodologías para la correcta gestión del proyecto y la planificación del mismo.

De este desarrollo se obtuvo el componente ELKorrelator, una pieza de software que junto a la pila ELK nos proporciona un entorno de correlación estable, rápido y escalable.

La conclusión principal del trabajo es que para obtener una mayor información a partir de los eventos individuales, estos se deben poder normalizar, relacionar y enriquecer mutuamente, obteniendo así un evento más complejo que nos proporciona información más completa.

Abstract:

The goal of this project is to generate an environment of event correlation that allows to distribute the load between different instances, offering great flexibility and scalability.

In the context of the security of a computer system we find a large ecosystem of devices, whose logs and generated traffic are important sources of information to detect malicious or anomalous behavior. One of the most common problems is to relate all this information found in different systems.

After investigating the different SIEMs and correlators more known and used, it was concluded that there was no open source product that offered all the desired characteristics.

It was decided to design a correlation environment formed by different components that allow us desired objectives. One of these components was necessary to develop it because there was no software that would provide us with the required functionalities.

The chosen methodology was the Rational Unified Process (RUP), which provides us with a framework of work with some methodologies for the correct management of the project and its planning.

From this development we obtained the ELKorrelator component, a piece of software that together with the ELK stack provides with a stable, fast and scalable correlation environment.

The main conclusion of the work is that to obtain more information from individual events, these must be able to normalize, relate and enrich each other, thus obtaining a more complex event that provides us with more complete information.

Índice

1. Introducción.....	1
1.1 Contexto y justificación del Trabajo.....	1
1.2 Objetivos del Trabajo.....	2
1.3 Enfoque y método seguido.....	2
1.4 Planificación del Trabajo.....	3
1.5 Breve resumen de productos obtenidos.....	6
1.6 Breve descripción de los otros capítulos de la memoria.....	6
2. Análisis.....	7
2.1. Analizar los participantes en el proyectos.....	7
2.2. Analizar los requisitos.....	7
2.3 Describir los actores.....	11
2.4 Describir los casos de uso.....	11
2.5 Analizar la arquitectura del sistemas.....	13
2.6 Analizar las herramientas a utilizar.....	14
3. Diseño.....	15
3.1 Diseñar el diagrama de clases.....	15
3.2 Diseñar los diagramas de secuencia.....	16
4. Implementación.....	21
4.1 Entorno de desarrollo.....	21
4.2 Comunicación de los componentes.....	22
4.3 Desarrollo de ELKorrelator.....	23
4.4 Interfaz web.....	25
5. Pruebas.....	27
5.1 Ejemplos de alertas generadas.....	27
6. Conclusiones.....	31
6.1 Conclusiones.....	31
6.2 Mejoras futuras.....	32
7. Glosario.....	33
8. Bibliografía.....	35
9. Anexo I Manual de usuario.....	37
10. Anexo II Manual de instalación.....	41

Lista de figuras

Ilustración 1: Estructura del sistema de correlación.....	3
Ilustración 2: Planificación PEC1.....	4
Ilustración 3: Planificación PEC2.....	5
Ilustración 4: Planificación PEC3.....	5
Ilustración 5: Planificación PEC4.....	6
Ilustración 6: Logo ELKorrelator.....	6
Ilustración 7: Diagrama de casos de uso.....	12
Ilustración 8: Diagrama de la arquitectura del sistema.....	13
Ilustración 9: Diagrama de interacción.....	14
Ilustración 10: Diagrama de clases.....	15
Ilustración 11: Diagrama de secuencia de "Parsear eventos".....	16
Ilustración 12: Diagrama de secuencia "Generar contextos".....	17
Ilustración 13: Diagrama de secuencia de "Correlar eventos".....	18
Ilustración 14: Diagrama de secuencia de "Eliminar contextos antiguos".....	19
Ilustración 15: Entorno de desarrollo y arquitectura.....	21
Ilustración 16: Directorio base.....	23
Ilustración 17: directorio ficheros_ELKorrelator.....	24
Ilustración 18: directorio ficheros_elk.....	25
Ilustración 19: Interfaz web de Kibana.....	25
Ilustración 20: Alerta generada al cumplirse R001.....	27
Ilustración 21: Alerta generada al cumplirse R002.....	28
Ilustración 22: Alerta generada al cumplirse R003.....	28
Ilustración 23: Alerta generada al cumplirse R004.....	29
Ilustración 24: Elegimos logs-* como index pattern.....	44
Ilustración 25: Elegimos el campo @timestamp.....	45
Ilustración 26: Importamos la configuración de Kibana.....	45
Ilustración 27: Dashboard con la información de las alertas de correlación.....	45

Lista de tablas

Participante_001.....	7
Participante_002.....	7
Participante_003.....	7
Nivel_de_importancia.....	7
Requisito_funcional_001.....	8
Requisito_funcional_002.....	8
Requisito_funcional_003.....	8
Requisito_funcional_004.....	8
Requisito_funcional_005.....	9
Requisito_funcional_006.....	9
Requisito_funcional_007.....	9
Requisito_funcional_008.....	9
Requisito_no_funcional_001.....	10
Requisito_no_funcional_002.....	10
Requisito_no_funcional_003.....	10
Requisito_no_funcional_004.....	10
Actor_001.....	11
Actor_002.....	11
Actor_003.....	11

1. Introducción

1.1 Contexto y justificación del Trabajo

En el contexto de la seguridad de un sistema informático nos encontramos con un gran ecosistema de dispositivos, cuyos logs y tráfico generado son importantes fuentes de información para detectar comportamientos maliciosos o anómalos.

Esta información es analizada por sistemas de detección que generan alertas, entre los que encontramos:

- sistemas que analizan el tráfico y lo comparan con reglas predefinidas.
- sandbox que detectan ficheros maliciosos analizando su comportamiento.
- sistemas que detectan anomalías en el comportamiento del tráfico o logs.
- escáneres de detección activa y pasiva de vulnerabilidades.
- sistemas endpoint para detectar equipos de usuario infectados.

Uno de los problemas más comunes es relacionar toda esta información que se encuentra en diferentes sistemas, posiblemente de diferentes fabricantes, para obtener información de mayor relevancia. También es interesante ser capaces de contextualizar esta información para descartar falsos positivos.

En estos momentos existen soluciones propietarias como Qradar[1] o Arcsight[2] capaces de correlar información de diferentes eventos de distintas fuentes, se trata de sistemas SIEM compuestos por diferentes componentes entre los que encontramos potentes motores de correlación.

Entre los correladores OpenSource encontramos Prelude-correlator[5], SEC[3] o Drools[4] que nos permiten generar alertas relacionando diferentes eventos. En concreto Prelude[5] permite crear reglas de correlación basadas en caso de uso a partir del lenguaje de programación Python. Tiene los inconvenientes de que su rendimiento baja con grandes cantidades de datos y no tiene la posibilidad de escalar el rendimiento distribuyendo la carga entre diferentes instancias. Normalmente se correlan alertas más simples como las generadas en snort o ossec[6]. Ninguno de los correladores anteriormente mencionados tiene la capacidad de escalar porque no son capaces de distribuir la carga entre diferentes instancias.

Por esta razón, cuando se necesita detectar alertas de seguridad basadas en la correlación de diferentes eventos se suelen utilizar soluciones propietarias. En otras ocasiones el papel de interpretar la información y relacionar diferentes alertas se realiza de manera manual por analistas de seguridad.

Se busca una solución capaz de obtener la información de diferentes fuentes, enriquecer los datos mediante ficheros CMDB o eventos anteriores, almacenar esta información de manera estructurada rápidamente accesible y por último disponer de un sistema capaz de analizar toda esta información y correlarla mediante casos de uso para obtener alertas más complejas. Esta solución debe estar compuesta por diferentes componentes que nos proporcionen un sistema escalable y fácil de administrar.

Ejemplos de casos de uso que nos debe permitir detectar:

- Un equipo descarga un fichero malicioso y en los siguientes cinco minutos se detecta una alerta snort desde la misma IP origen.
- Una IP externa un escaneo de vulnerabilidades y se detecta que en la última semana ya se había detectado otros comportamientos maliciosos de la misma IP.
- Se detecta mediante IDS un ataque SQL Injection contra un portal que no ha sido detenido por el WAF.
- Se detecta un ataque de fuerza bruta exitoso contra un servidor FTP y posteriormente se detecta que el usuario afectado sube un fichero al servidor.

1.2 Objetivos del Trabajo

El objetivo principal es obtener un entorno de correlación que nos proporcione una gran compatibilidad con cualquier formato de log, que sea fácilmente escalable, que sea capaz de evaluar casos de uso complejos y que esté compuesto por diferentes componentes que ofrezcan escalabilidad.

Lo primero es agrupar los objetivos en función de su prioridad. Este nivel de prioridad se tendrá en cuenta tanto para la planificación del trabajo como para tomar las decisiones oportunas a lo largo del desarrollo.

Prioridad alta, el sistema debe ser capaz de:

- Obtener un sistema capaz de integrarse fácilmente con cualquier formato de fuente como CEF, squid, snort o JSON.
- Recoger los logs por syslog, beats o kafka.
- Almacenar estos datos de manera estructurada que permita ser consultados rápidamente.
- Correlar la información de diversas fuentes, evaluando casos de uso en base a consultas de los datos estructurados.
- Escalar de manera sencilla, permitiendo separar los diferentes componentes en distintas máquinas.

Prioridad media, es deseable que el sistema sea capaz de:

- Permitir la creación de detección de nuevos casos de uso por parte del usuario.
- Securitizar las comunicaciones entre los diferentes componentes de la solución.

Prioridad baja, es un añadido que el sistema sea capaz de:

- Enriquecer los eventos con alertas anteriores.
- Enriquecer los eventos con información de CMDB.
- Mostrar visualizaciones con gráficos para consultar fácilmente tanto los datos estructurados como las alertas de correlación generadas.

1.3 Enfoque y método seguido

Para obtener las funcionalidades que deseamos se utilizan los siguientes productos existentes como componentes:

- Logstash nos permite el parseo de los logs, enriquecimiento de datos y filtrado de eventos.
- Elasticsearch nos permite almacenar los datos de manera estructurada y fácilmente accesible mediante consultas. Permite almacenar los datos de manera distribuida en diferentes clusters.
- Kibana nos aporta un interfaz para acceder a los datos de manera sencilla.

También se desarrollan otros componentes:

- ELKorrelator, este componente será el que evaluará las reglas de correlación para detectar comportamientos maliciosos.

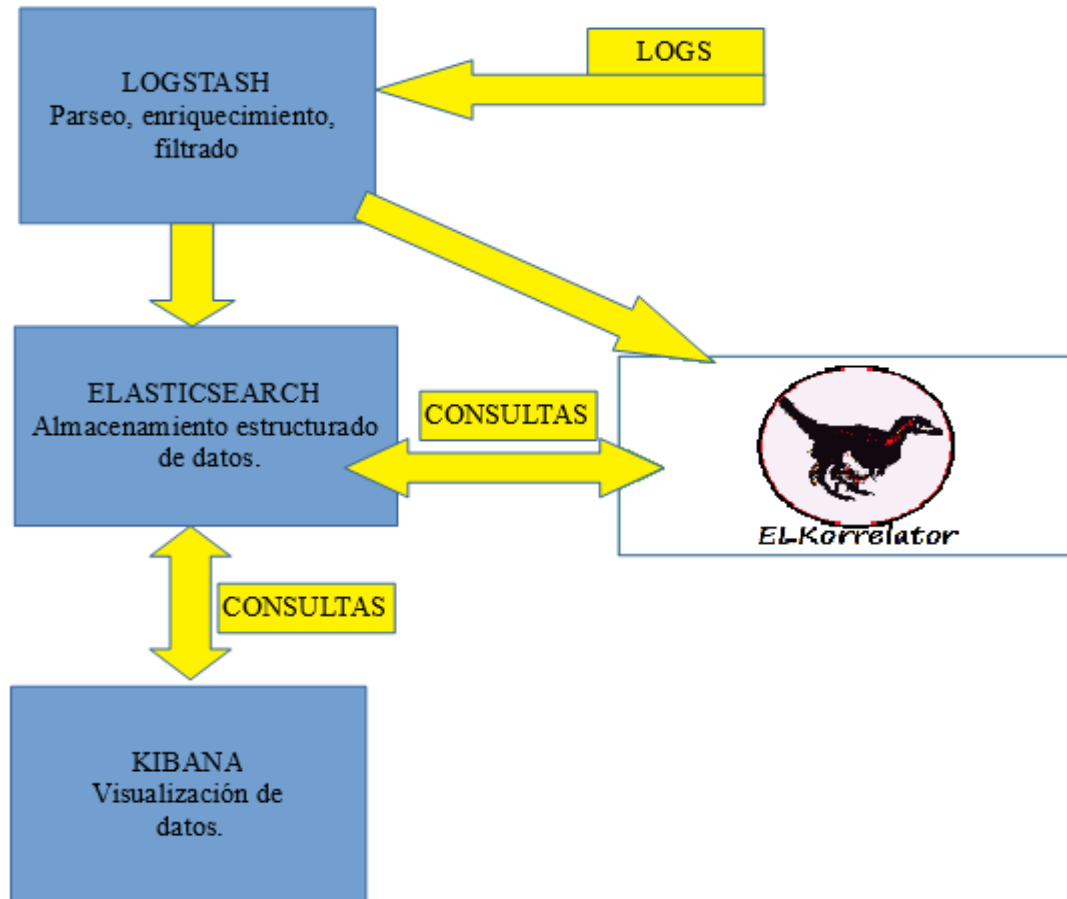


Ilustración 1: Estructura del sistema de correlación

Esta decisión se ha tomado en función de la experiencia del estudiante con dichas herramientas. Se conoce por experiencia que los datos almacenados con la pila ELK [7] son accesibles de manera muy rápida. Además es una solución muy escalable, permitiendo almacenar la base de datos entre diferentes instancias que forman un cluster, obteniendo de esta forma una mejora significativa de rendimiento.

Se ha decidido desarrollar el componente encargado de evaluar los casos de uso en base a consultas a Elasticsearch porque no hay ningún proyecto ni producto que nos proporcione esta funcionalidad. Hay algún proyecto que podría servir para generar alertas simples en base a consultas a Elasticsearch como es Elastalert[8], pero no nos proporciona la correlación que se desea obtener, basada en casos de uso complejos.

1.4 Planificación del Trabajo

El proceso de desarrollo del software elegido es el Proceso Unificado (RUP)[13], el cuál se adapta a proyectos de desarrollo proporcionándonos un marco de trabajo con unas metodologías para la correcta gestión del proyecto y la planificación del mismo.

Se han identificado las diferentes tareas a cumplir en este proyecto teniendo en cuenta que algunas de estas tareas podrían ser realizadas en paralelo en el caso de desarrollarse con más personas. Para la planificación se ha tenido en cuenta que sólo se realizarán tareas los días de diario, con tres horas de dedicación diarias.

En la planificación se distinguen cuatro fases cuyos hitos coinciden con las entregas de las diferentes PEC.

En primer lugar tenemos la planificación de la fase de inicio, que sólo consta de una iteración, cuyas tareas principales son:

- Contextualizar la motivación, para explicar el problema que se intenta resolver.
- Estudiar el estado del arte, para conocer que productos podrían ayudarnos a resolver nuestro problema.
- Idear una arquitectura básica, porque desde los primeros inicios se debe enfocar el entorno de correlación como un entorno distribuido y fácilmente escalable.
- Planificar el proceso de desarrollo, definiendo que componentes pueden ser implementados con productos existentes y cuales tendrán que ser desarrollados.
- Generar documentación PEC1, consiste en generar el entregable de este hito.

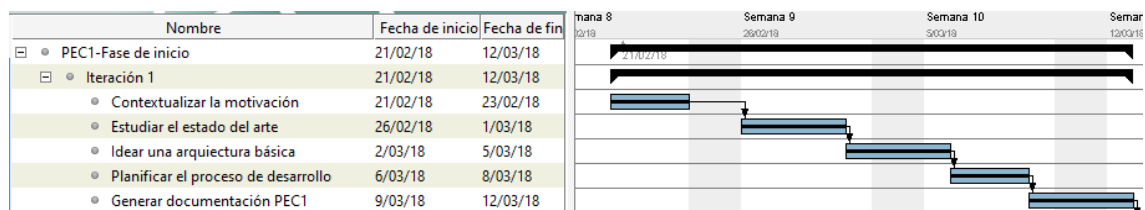


Ilustración 2: Planificación PEC1

La segunda fase o fase de elaboración está formada de una iteración en la que se distinguen las siguientes diez tareas:

- Analizar los participantes del proyecto, consiste en definir todas aquellas personas que tienen relación con el desarrollo del proyecto.
- Analizar los objetivos del proyecto, consiste en encontrar una serie de objetivos que nuestro entorno de correlación debe satisfacer.
- Analizar los requisitos, a partir de los objetivos obtenemos los requisitos funcionales y no funcionales del proyecto.
- Describir actores, que serán todos los entes que interactuarán con el entorno de correlación.
- Describir los casos de uso, que consiste en describir las diferentes funcionalidades que debe permitir el entorno de correlación.
- Analizar la arquitectura del sistema, en este punto se define cuál será la arquitectura en la que se desplegarán los diferentes componentes.
- Diseñar diagrama de clases, enfocado al diseño de la pieza de software que se debe desarrollar.
- Diseñar el diagrama de secuencia, consiste en mostrar visualmente como interactúan los diferentes componentes para completar los diferentes casos de uso.
- Generar documentación PEC2, consiste en generar el entregable de este hito.

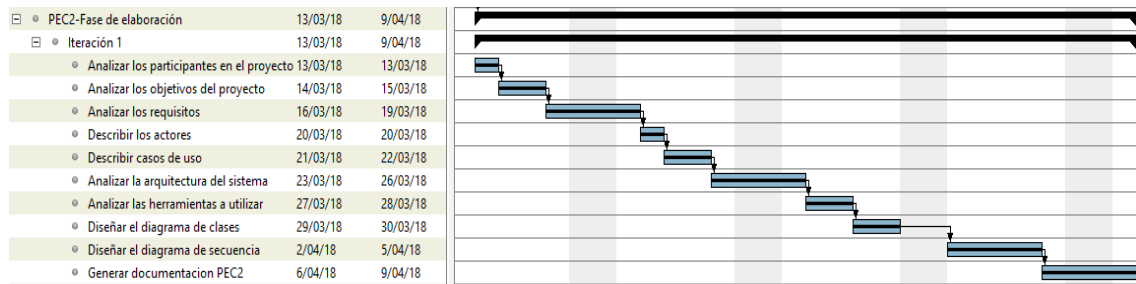


Ilustración 3: Planificación PEC2

La tercera fase es la de construcción, en la que encontramos dos iteraciones distintas, que se deciden separar debido a su distinta naturaleza. La primera iteración se centra en el desarrollo de ELKorrelator, en la que encontramos las siguientes tareas:

- Preparación del entorno, en el cual se prepara el entorno en el que se debe desplegar el entorno de correlación.
- Desarrollo del componente correlador, es decir, desarrollo de ELKorrelator.
- Depuración de bugs-pruebas, en el que se solucionan los problemas encontrados.

En la segunda iteración nos centramos en la integración del componente desarrollado ELKorrelator con los productos Logstash, Elasticsearch y Kibana, en la que encontramos las siguientes tareas:

- Integración de los componentes, configuración de los diferentes productos y componentes que forman el entorno de correlación.
- Depuración de bugs-pruebas, en el que se solucionan los problemas encontrados.
- Generar documentación PEC3, consiste en generar el entregable de este hito.

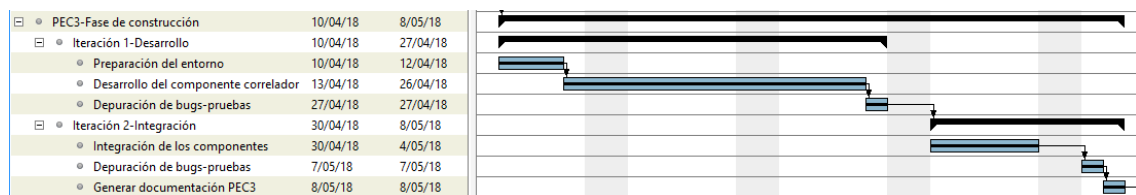


Ilustración 4: Planificación PEC3

Por último tenemos la fase final que se centra en la generación de entregables, formado por una sola iteración, encontramos las siguientes tareas:

- Generación del documento TFM, la generación de la memoria, es decir, este documento.
- Generación del manual de usuario, en el cual se detallan los diferentes parámetros que se pueden utilizar para crear nuevas reglas de correlación. Se encuentra en el Anexo I.
- Generación del manual de instalación, en el que se detallan los pasos a realizar para replicar el entorno de correlación. Se encuentra en el Anexo II.
- Publicación del software en github, esto es del código fuente del componente ELKorrelator y de toda la configuración necesaria para su funcionamiento, se encuentra en <https://github.com/aizquierdonu/ELKorrelator>.
- Preparación de la defensa-presentación, que se entregará en forma de vídeo.

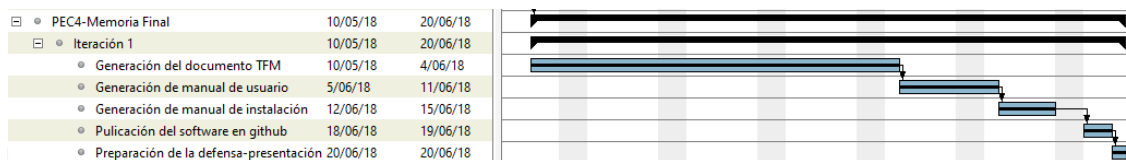


Ilustración 5: Planificación PEC4

Para la generación de los diagramas de Gantt se utiliza el software ganttproject.[9]

1.5 Breve resumen de productos obtenidos



ELKorrelator

Ilustración 6: Logo ELKorrelator

Para obtener un entorno de correlación escalable ha sido necesario el desarrollo del componente ELKorrelator, que se encuentra disponible en github[10]. Este software se explica en mayor detalle en los siguientes capítulos, pero básicamente sus funcionalidades son:

- Identificar eventos que deben provocar el análisis de una regla de correlación, es decir generar un contexto de correlación.
- Gestionar de manera adecuada los contextos de correlación.
- Generar alertas a partir de información consultada a la BBDD elasticsearch.

1.6 Breve descripción de los otros capítulos de la memoria

En la memoria encontramos los siguientes capítulos:

- Análisis, en el que se explica como se aborda el problema y como se obtiene una solución que satisface todos los requisitos deseados.
- Diseño, capítulo en el cual nos centramos en el diseño de los componentes que se deben desarrollar y que conforman el producto ELKorrelator.
- Implementación, nos centramos en explicar las diferentes piezas de software y de configuración que han resultado del desarrollo y que se pueden obtener desde github[10]
- Pruebas, para comprobar el correcto funcionamiento del entorno de correlación se han creado reglas de correlación, que ponen a prueba todos los componentes del sistema.
- Conclusiones, apartado en el que encontramos las conclusiones tras finalizar el proyecto y se definen las mejoras futuras.
- Glosario, apartado en el que se definen algunos de los conceptos más importantes utilizados en este documento.
- Bibliografía.

Además en este mismo documento podemos encontrar los siguientes anexos:

- Manual de usuario, en el que se explican los diferentes campos que componen las reglas de correlación.
- Manual de instalación, en el cual se explican los pasos para replicar el entorno de correlación desarrollado.

2. Análisis

2.1. Analizar los participantes en el proyectos

En las siguientes tablas se presentará a los participantes que han intervenido en el desarrollo del proyecto:

Participante 001	
Nombre	Andrés Izquierdo Núñez
Organización	Estudiante del máster universitario de Seguridad de las TIC de la UOC
Rol	Jefe de proyecto, analista, diseñador, desarrollador y depurador.

Participante 002	
Nombre	Jordi Guijarro Olivares
Organización	Profesor UOC
Rol	Tutor del TFM

Participante 003	
Nombre	Victor García Font
Organización	Profesor UOC
Rol	Responsable de la asignatura TFM

2.2. Analizar los requisitos

En primer lugar definimos los diferentes niveles de importancia que afectan a los diferentes requisitos:

Nivel de importancia	
Alta	Es imprescindible cumplir este objetivo para que el sistema de correlación sea funcional.
Media	Se necesita cumplir este objetivo para ajustarse a las necesidades deseadas por un usuario.
Baja	No es necesario cumplir este objetivo para disponer de un sistema de correlación funcional.

Los requisitos se deben separa en dos categorías diferenciadas, requisitos funcionales y requisitos no funcionales.

2.2.1 Requisitos funcionales

Los requisitos funcionales pueden ser los diferentes cálculos, detalles técnicos o funcionalidades específicas que el sistema debe cumplir.

Requisito funcional 001	
Nombre	Normalización de eventos.
Descripción	El sistema debe ser capaz de parsear diferentes formatos de eventos. Algunos de los formatos que se deben parsear son CEF, squid, snort o JSON.
Importancia	Alta
Estado	Completo

Requisito funcional 002	
Nombre	Centralización de eventos.
Descripción	El sistema debe ser capaz de recibir eventos mediante syslog, filebeat y kafka.
Importancia	Alta
Estado	Completo

Requisito funcional 003	
Nombre	Correlación de eventos.
Descripción	El sistema debe ser capaz de detectar comportamientos maliciosos a partir de la correlación de diferentes eventos. Esta correlación se hará mediante reglas de correlación definidas.
Importancia	Alta
Estado	Completo

Requisito funcional 004	
Nombre	Creación de reglas de correlación
Descripción	El sistema debe permitir que se creen nuevas reglas de correlación para identificar nuevos comportamientos complejos.
Importancia	Media
Estado	Completo

Requisito funcional 005	
Nombre	Securizar el envío de eventos.
Descripción	El sistema debe permitir que el envío de eventos hacia el sistema de correlación sea de mediante conexiones cifradas.
Importancia	Media
Estado	No completada.

Requisito funcional 006	
Nombre	Eficiencia de almacenamiento
Descripción	Se debe controlar el espacio ocupado por los archivos temporales necesarios para el correcto funcionamiento de la plataforma, incluyendo la rotación de logs.
Importancia	Media
Estado	Completo

Requisito funcional 007	
Nombre	Enriquecer eventos con alertas anteriores.
Descripción	Se debe permitir enriquecer los eventos con detecciones anteriores.
Importancia	Baja
Estado	Completo

Requisito funcional 008	
Nombre	Enriquecer eventos con CMDB.
Descripción	Se debe permitir enriquecer los eventos con información conocida.
Importancia	Baja
Estado	Completo

2.2.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos enfocados en el diseño o la implementación, enfocándose en las propiedades que debe cumplir el sistema.

Requisito no funcional 001	
Nombre	Almacenamiento de rápido acceso.
Descripción	El sistema debe ser capaz almacenar estos datos de manera estructurada que permita ser consultados rápidamente.
Importancia	Alta
Estado	Completo

Requisito no funcional 002	
Nombre	Correlación mediante datos estructurados.
Descripción	La evaluación de los casos de uso se realizará mediante consultas de los datos.
Importancia	Alta
Estado	Completo

Requisito no funcional 003	
Nombre	Componentes del sistema de correlación distribuidos.
Descripción	El sistema debe permitir distribuir la carga entre diferentes máquinas y permitir un escalado sencillo, para ello se debe permitir que los diferentes componentes puedan funcionar de manera distribuida.
Importancia	Alta
Estado	Completo

Requisito no funcional 004	
Nombre	Visualización de gráficas.
Descripción	Los eventos y las detecciones se deben estar accesibles en un portal con visualizaciones y gráficas que ayuden al entendimiento de los datos.
Importancia	Baja
Estado	Completo

2.3 Describir los actores

En las siguientes tablas se muestran los actores presentes en los diferentes casos de uso.

Actor 001	
Nombre	Usuario
Descripción	Usuario final que interactúa con el sistema de correlación.
Versión	1.0 - 04/04/2018
Autor	Andrés Izquierdo

Actor 002	
Nombre	Tiempo
Descripción	Algunas funcionalidades se ejecutan de manera periódica. Este comportamiento se representa mediante este actor.
Versión	1.0 - 04/04/2018
Autor	Andrés Izquierdo

Actor 003	
Nombre	Logger
Descripción	Algunas funcionalidades se ejecutan cuando llega un log o evento. Este comportamiento se representa mediante este actor
Versión	1.0 - 04/04/2018
Autor	Andrés Izquierdo

2.4 Describir los casos de uso

A partir de los requisitos funcionales identificados y los actores presentados anteriormente, podemos generar el siguiente diagrama de casos de uso que debe cumplir el sistema.

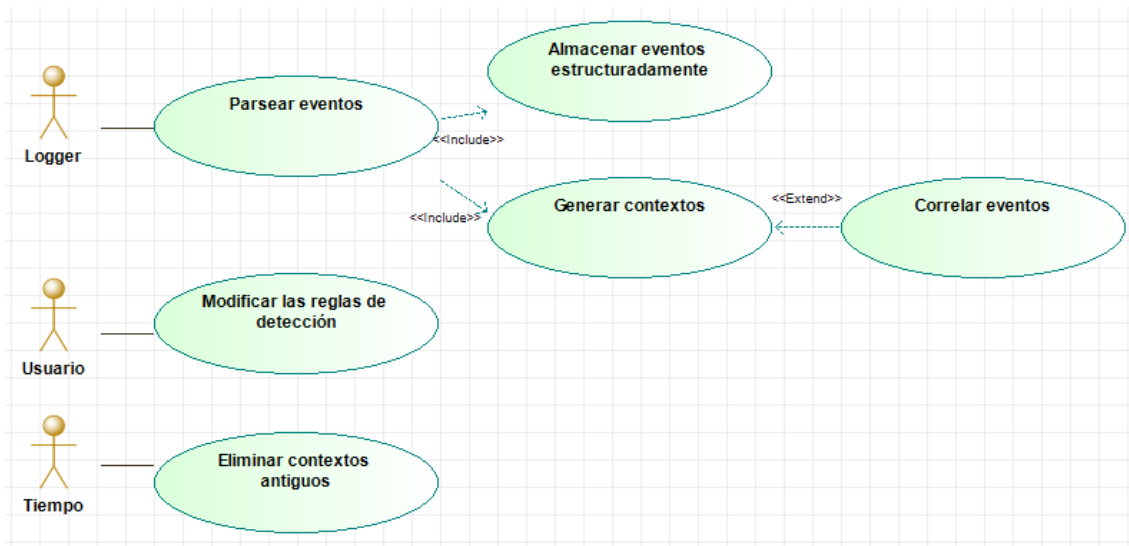


Ilustración 7: Diagrama de casos de uso

El actor Logger, representación de la llegada de eventos al sistema de correlación, provoca que se ejecute el caso de uso “Parsear evento”, que consiste en obtener los diferentes campos que nos interesen del evento, como puede ser la IP origen, el hostname, la URL, etc.

Una vez parseado el evento se ejecutan de manera simultanea los siguientes casos de uso:

- Almacenar eventos estructurados, que consiste en almacenar el evento junto a sus campos parseados en un almacenamiento estructurado.
- Generar contextos, se comprueba si alguno de los eventos parseados debe generar un contexto de correlación. En caso afirmativo se produce la ejecución del caso de uso “Correlar eventos”.

A continuación nos encontramos con el caso de uso “Modificar las reglas de detección”, en el que el actor Usuario puede añadir, borrar o modificar reglas de correlación.

Por último tenemos el caso de uso “Eliminar contextos antiguos”, básicamente consiste en eliminar los ficheros temporales generados al crear contextos de manera periódica.

2.5 Analizar la arquitectura del sistemas

Se va a plantear una arquitectura distribuida en dos máquinas, una encargada del parseo y almacenamiento estructurado de los datos, y otra encargada de gestionar los contextos de correlación y detectar los casos de uso definidos.

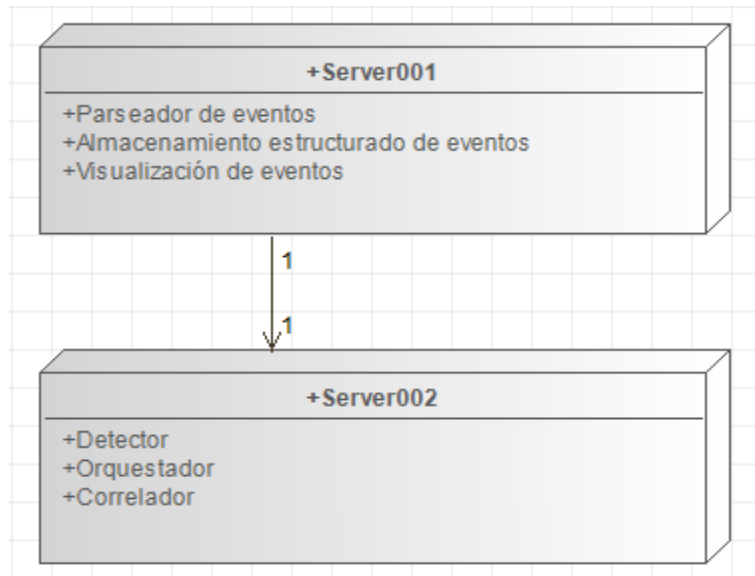


Ilustración 8: Diagrama de la arquitectura del sistema

La solución proporcionará la posibilidad de distribuir los diferentes componentes en la arquitectura que mejor se adapte. Los componentes que nos encontramos son los siguientes:

- Parseador de eventos, se encarga de parsear los eventos.
- Almacenamiento estructurado de eventos, se encarga de almacenar los eventos parseados. Debe proporcionar un mecanismo de búsqueda sencillo.
- Visualización de eventos, debe proporcionar una interfaz simple para acceder a los eventos almacenados.
- Detector, evalúa si un evento debe iniciar un contexto de correlación.
- Orquestador, se encarga de coordinar los diferentes detectores y correladores, una tarea fundamental si se desea distribuir la carga entre diferentes máquinas. El caso más evidente del que se encarga es que no se produzcan duplicidades con los contextos.
- Correlador, mediante consultas a los datos estructurados comprueba si se cumple un caso de uso que debe ser detectado.

Ambas máquinas tendrán sistema operativo Linux. Se decide tomar esta decisión en base a la experiencia del alumno en este SO, porque consume pocos recursos y porque el poco espacio de almacenamiento necesario. En concreto se utilizará la distro Ubuntu Server, pero la solución debe ser compatible con otras distros como Debian o Centos.

2.6 Analizar las herramientas a utilizar

A continuación se muestra el diagrama de interacción de los diferentes componentes.

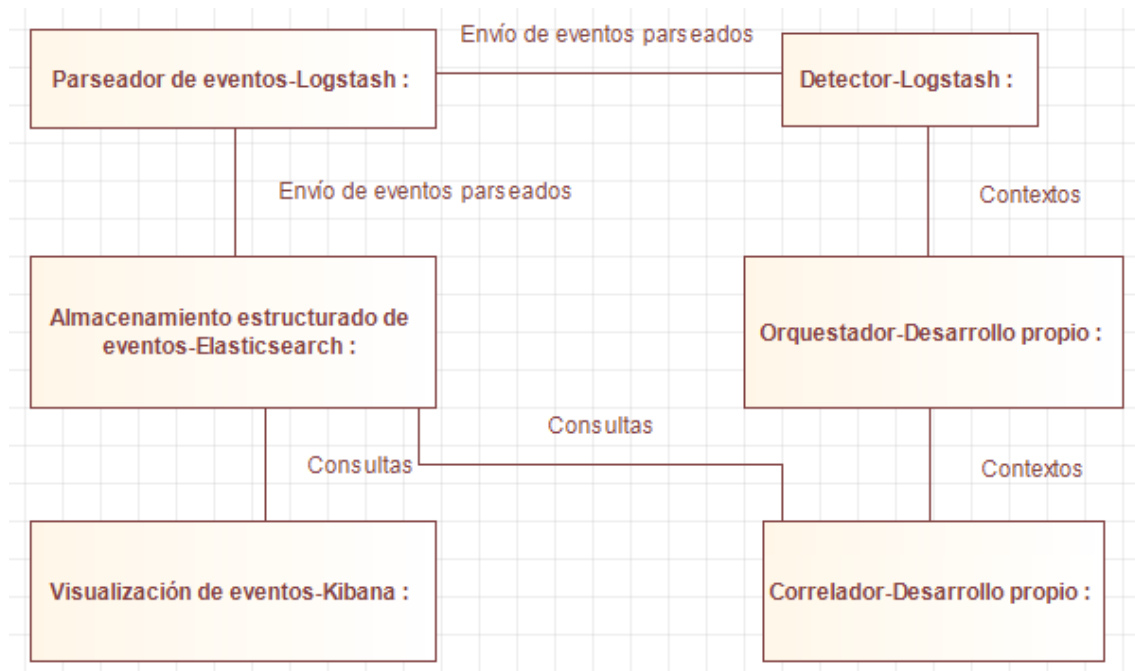


Ilustración 9: Diagrama de interacción

Para el parseo de eventos se utilizará Logstash, que nos permite extraer los diferentes campos del evento y el enriquecimiento de estos.

Para el almacenamiento estructurado se utilizará Elasticsearch. Se trata de una base de datos no relacional que permite indexar grandes cantidades de eventos por segundo. Además ofrece una gran velocidad de búsqueda.

Para la visualización de eventos se utilizará Kibana, que nos permite visualizar los eventos indexados en Elasticsearch.

El detector se implementará con Logstash, que permite definir filtrados de eventos muy avanzados.

El orquestador se desarrollará, es posible que reciba una gran carga de trabajo por lo que se ha decidido desarrollarse en C.

El correlador se desarrollará, en este punto el rendimiento no es tan importante, en primer lugar porque el orquestador filtrará la mayor parte de los contextos y además limitaciones en cuanto a rendimiento tendrán lugar en las consultas a Elasticsearch. Se decide implementarlo en Python.

Tanto Logstash como Elasticsearch necesitan que la máquina en la que se ejecuten tenga instalado Java para funcionar.

Los componentes Detector, Orquestador y Correlador forman ELKorrelator.

3. Diseño

3.1 Diseñar el diagrama de clases

Una vez realizada la etapa de análisis donde se ha analizado los requisitos del sistema, los actores que intervienen y se han definido los casos de uso pasamos a la etapa de diseño, incidiendo en los componentes que tendremos que desarrollar.

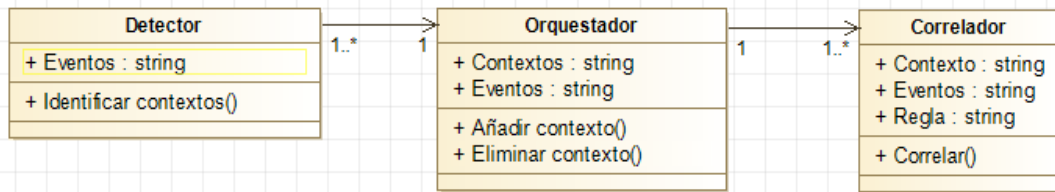


Ilustración 10: Diagrama de clases

La clase Detector representa al componente Detector. Los detectores deben tener asignado un orquestador, pero puede haber uno o más detectores.

La clase Orquestador representa al componente Orquestador. El sistema debe contar con un Orquestador. Es posible utilizar varias instancias de Orquestador siempre y cuando nos aseguremos que gestionan contextos distintos, aunque es recomendable gestionarlo en un sólo punto siempre y cuando el rendimiento de la máquina no se vea comprometido.

Tanto Logstash como el Detector(implementado con este mismo software) contienen tres subcomponentes diferenciados que serán representados como clases:

- Input: Se encarga de recibir eventos
- Filter: Se encarga de parsear, filtrar y enriquecer eventos
- Output: Se encarga de enviar los eventos

Elasticsearch también será representado como clases en los diagramas de secuencia.

3.2 Diseñar los diagramas de secuencia

A continuación se presentan los diagramas de secuencia de diseño teniendo en cuenta todos los puntos analizados, actores, casos de uso, requisitos, arquitectura, diagrama de iteración y diagrama de clases.

3.2.1 Diagrama de secuencia 'Parsear eventos'

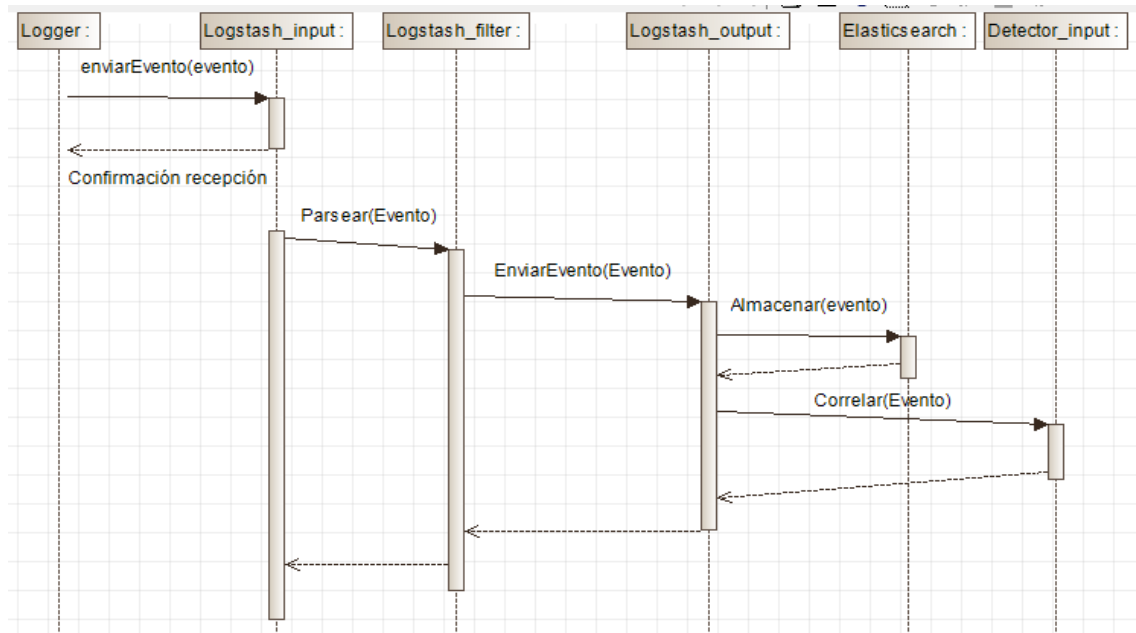


Ilustración 11: Diagrama de secuencia de "Parsear eventos"

Como se puede observar en el diagrama de secuencia, una vez que el evento es parseado, enriquecido y normalizado en Logstash se envía tanto a Elasticsearch para que sea almacenado como al detector, para comenzar el caso de uso "generar contextos" que se presenta a continuación.

3.2.2 Diagrama de secuencia 'Generar contextos'

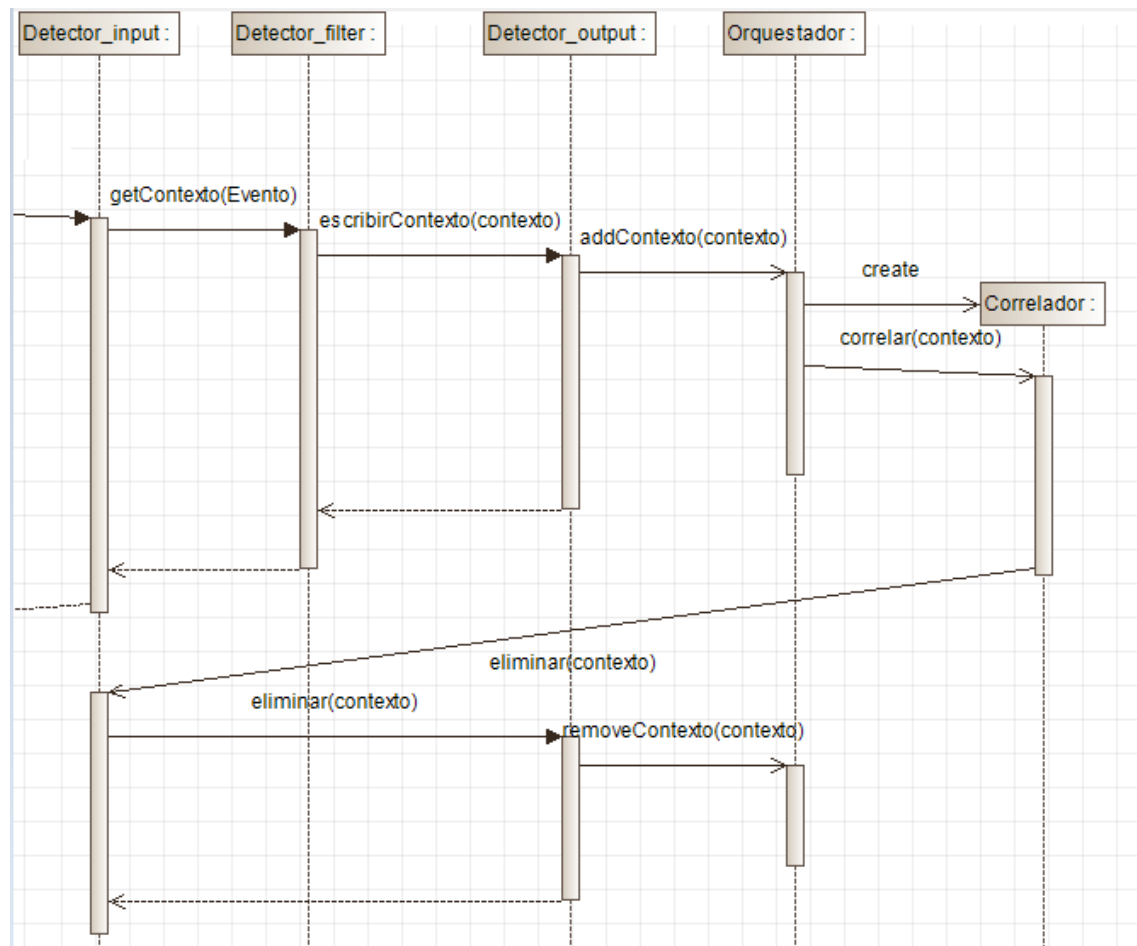


Ilustración 12: Diagrama de secuencia "Generar contextos"

Una vez que llega un evento normalizado al detector se evalúa para comprobar si debe generar algún contexto, un evento puede generar varios contextos. En caso de ser así, se llama (asincronamente) al Orquestador para que se añada el contexto.

En caso de que no se encuentre ya en correlación se añadirá y se creará una instancia Correlador para encargarse de evaluar si se cumple el caso de uso del contexto.

Una vez finalizada la correlación(explicada brevemente en el siguiente diagrama) se debe eliminar el contexto, por lo que el correlador envía el contexto a eliminar al Detector, quien se lo comunica al Orquestador. Se ha decidido que el Orquestador únicamente será accesible desde el detector para evitar problemas de seguridad.

3.2.3 Diagrama de secuencia 'Correlar eventos'

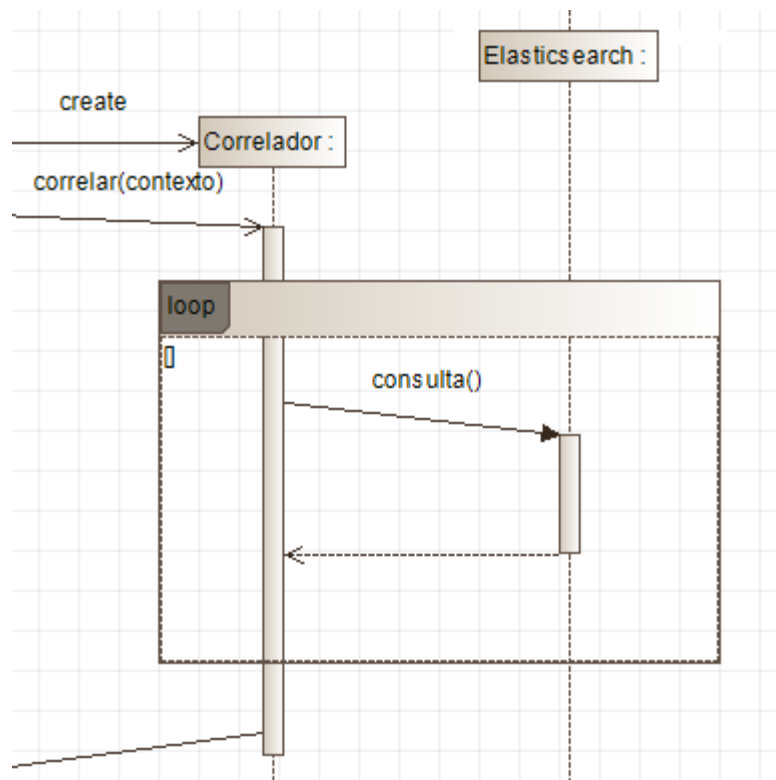


Ilustración 13: Diagrama de secuencia de "Correlar eventos"

En este diagrama observamos el comportamiento que se sigue cuando se debe correlar eventos. Cuando el Orquestador encuentra un contexto que se debe correlar se crea una nueva instancia encargada de ello. La instancia Correlador obtendrá la información necesaria a partir del contexto, la regla de correlación y las consultas contra la base de datos documental Elasticsearch.

3.2.4 Diagrama de secuencia 'Eliminar contextos antiguos'

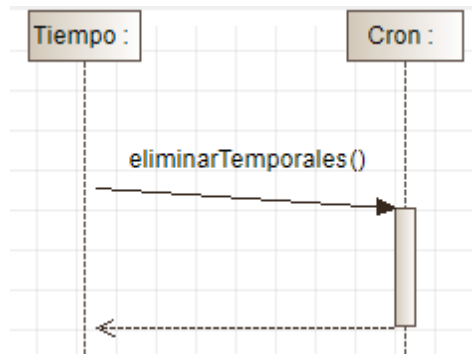


Ilustración 14: Diagrama de secuencia de "Eliminar contextos antiguos"

Básicamente consiste en planificar una tarea en crontab que se encargue de eliminar ficheros temporales antiguos utilizados para la transferencia desde el Detector al Orquestador.

El caso de uso "Modificar las reglas de detección" simplemente sirve para representar que el usuario podrá crear o añadir nuevas reglas. Queda fuera de este proyecto el desarrollar una interfaz desde la que poder realizar esta tarea.

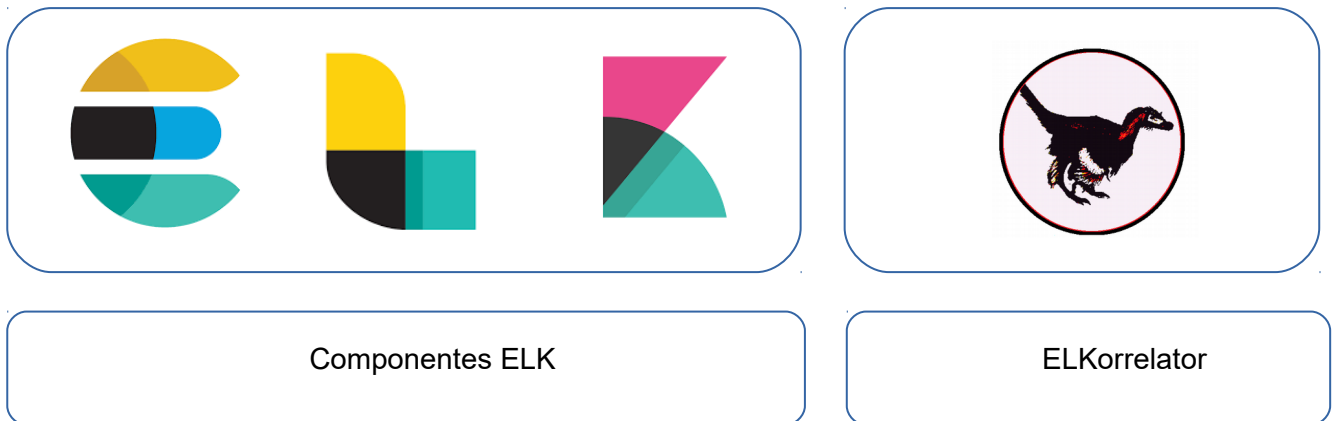
Para la generación de los diagramas UML se ha utilizado el software Modelio 3.7[11]

4. Implementación

4.1 Entorno de desarrollo

En primer lugar se ha preparado el entorno de desarrollo y despliegue, que en este caso ha sido el mismo.

Tal y como se había definido en la fase de diseño se despliega la solución en un entorno distribuido, una máquina con los componentes ELK y otra máquina con el componente ELKorrelator (formado por Detector, Orquestador y Correlador) que tuvieran comunicación directa entre ellas. Se ha decidido virtualizar estas máquinas mediante VirtualBox[12] por tratarse de una herramienta gratuita, estable, y con la que el estudiante trabaja muy a menudo.



El sistema operativo elegido ha sido Ubuntu Server 16.04, pero la solución debería ser compatible con otras distro de Linux como Debian o Fedora. La versión de los componentes de la pila ELK que se ha decidido utilizar es la versión 6.2.3, que se trata de la última disponible al inicio del proyecto. No debería haber ningún problema para utilizar cualquier versión 6.x.

A continuación vemos el diagrama de despliegue, en el que se puede observar como se han desplegado los diferentes componentes y como interactúan entre ellos.

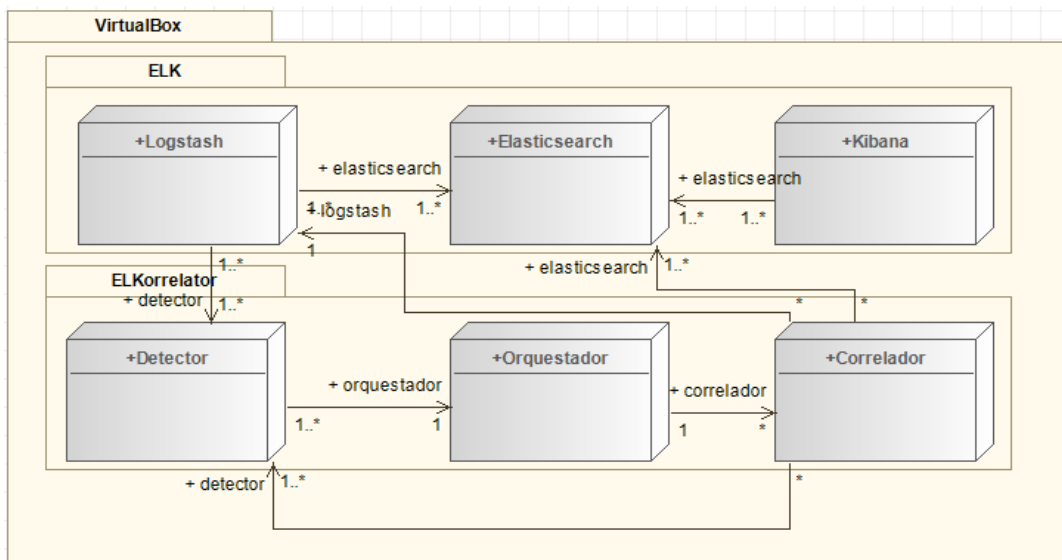


Ilustración 15: Entorno de desarrollo y arquitectura

4.2 Comunicación de los componentes

En este apartado se van a explicar los diferentes mecanismos utilizados para la comunicación entre los distintos componentes. Como se puede observar en el diagrama del “entorno de desarrollo y arquitectura” los componentes se comunican entre sí. En cada caso se han utilizado diferentes mecanismos teniendo en cuenta:

- Los conocimientos del estudiante.
- El tiempo que lleva a cabo el desarrollo de los componentes de comunicación.
- La necesidad de que los componentes se desplieguen en un entorno distribuido y con una arquitectura diferente a la propuesta.

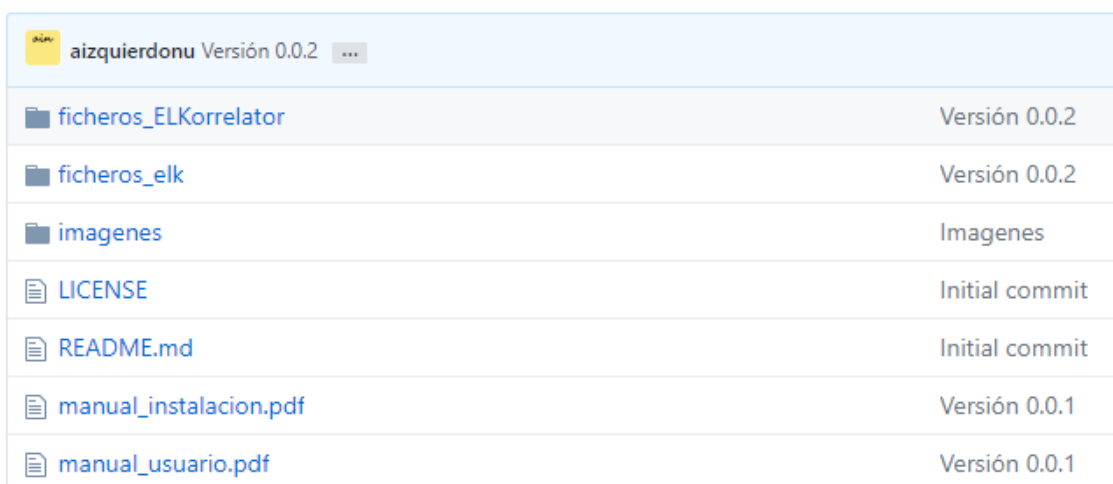
Tras el análisis de las diferentes comunicaciones se obtienen las siguientes conclusiones:

- Comunicación Logstash-Elasticsearch
La mejor opción es utilizar el plugin de output de Logstash ‘elasticsearch’.
- Comunicación Kibana-Elasticsearch
Configuración por defecto, Kibana está desarrollado específicamente para integrarse con Elasticsearch.
- Comunicación Logstash-Detector
Se decidió implementar la comunicación utilizando los plugin de input/output de Logstash ‘tcp’. Existen otras posibilidades pero proporcionan menor rendimiento.
- Comunicación Detector-Orquestador
En este punto se decidió que la comunicación se realice mediante ficheros que se escriben por el Detector y se leen por el Orquestador. En caso de querer separar estos dos componentes es tan sencillo como enviar estos logs a otra máquina (por tcp, syslog, filebeats, etc) y que se escriban en disco en la máquina que se encuentra el Orquestador.
- Comunicación Orquestador-Correlador
Los correladores son instancias que crea el Orquestador mediante una llamada a system, la información se pasa en el momento de creación como un parámetro. No es inmediato distribuir estos dos componentes, habría que modificar el comando de creación para que se realice la llamada mediante ssh a otra máquina.
- Comunicación Correlador-Elasticsearch
Se realizan consultas mediante la función (requests.post) de python 2.7. Elasticsearch responde en formato JSON, por lo que es muy sencillo de parsear.
- Comunicación Correlador-Detector
Esta comunicación es necesaria para avisar de cuando tiene que eliminar un contexto. Se realiza con comunicación TCP mediante la función “sendTCP” de python 2.7.
- Comunicación Correlador-Logstash
Esta comunicación es necesaria para informar de las reglas de correlación que se cumplen. Se realiza con comunicación TCP mediante la función “sendTCP” de python 2.7.

4.3 Desarrollo de ELKorrelator

El desarrollo se puede encontrar en github[10], en el que está disponible el código fuente, ficheros de configuración necesarios para desplegar el componente ELKorrelator junto con la pila ELK para formar un entorno de correlación, manual de usuario, manual de instalación y los logos de ELKorrelator. A continuación se explica la estructura básica de directorios y ficheros.

4.3.1 Directorio base

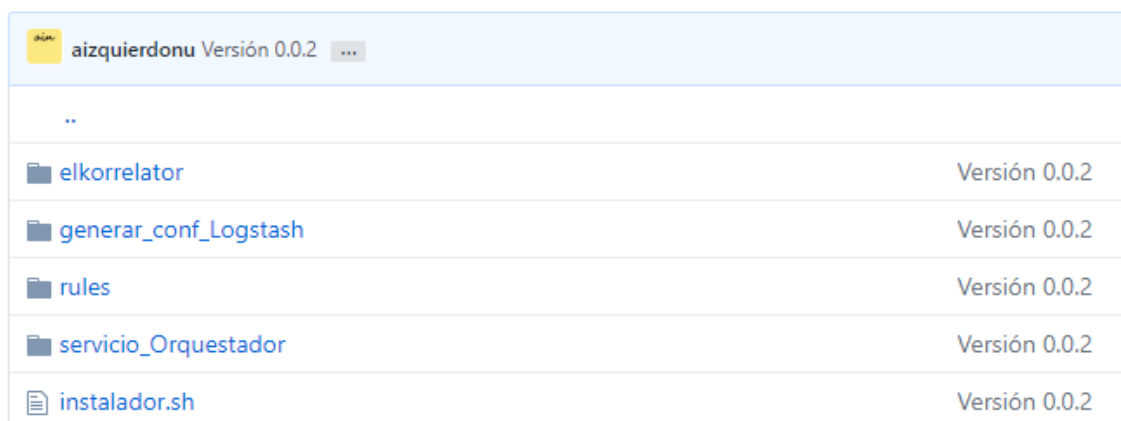


aizquierdonu Versión 0.0.2	
📁 ficheros_ELKorrelator	Versión 0.0.2
📁 ficheros_elk	Versión 0.0.2
📁 imagenes	Imágenes
📄 LICENSE	Initial commit
📄 README.md	Initial commit
📄 manual_instalacion.pdf	Versión 0.0.1
📄 manual_usuario.pdf	Versión 0.0.1

Ilustración 16: Directorio base

- En el directorio base encontramos: ficheros_ELKorrelator, encontramos los ficheros necesarios para la instalación y configuración del componente ELKorrelator.
- ficheros_elk, encontramos los ficheros necesarios para configurar la pila ELK y scripts para probar las reglas de correlación.
- Imágenes, encontramos los logos de ELKorrelator.
- LICENSE, se trata de la licencia del proyecto, se ha decidido utilizar GNU General Public License v3.0
- README.md, contiene una pequeña descripción del proyecto.
- manual_instalacion.pdf, se trata del manual de instalación para replicar el entorno de correlación, también se puede encontrar en el Anexo II.
- manual_usuario.pdf, se trata del manual de usuario en el que se explica como crear nuevas reglas de correlación, también se puede encontrar en el Anexo I.

4.3.2 Directorio ficheros_ELKorrelator



aizquierdonu Versión 0.0.2	
..	
elkorrelator	Versión 0.0.2
generar_conf_Logstash	Versión 0.0.2
rules	Versión 0.0.2
servicio_Orquestador	Versión 0.0.2
instalador.sh	Versión 0.0.2

Ilustración 17: directorio ficheros_ELKorrelator

En el directorio ficheros_ELKorrelator encontramos:

- elkorrelator, en el que se encuentra el código fuente de los subcomponentes Orquestador y Correlador.
- generar_conf_Logstash, que nos permite generar la configuración de Logstash para implementar el subcomponente Detector.
- Rules, reglas de correlación por defecto. Encontramos las siguientes reglas:
 - Descarga de fichero malicioso y detección IDS, identificador R001.
 - Ip realiza un escaneo de vulnerabilidades NISSUS y se detecta que no es la primera vez que lo hace esta semana, identificador R002.
 - Se detecta un ataque SQL Inyección en IDS pero no se bloquea en WAF, identificador R003.
 - Se detecta un ataque de fuerza bruta ftp satisfactorio y a continuación se detecta que el usuario afectado sube un fichero al servidor, identificador R004.
- servicio_Orquestador, fichero en el que se define el servicio Orquestador para poder controlar dicho servicio mediante systemctl.
- Instalador.sh, ejecutable que automatiza la instalación de los subcomponentes de ELKorrelator.

4.3.3 Directorio ficheros_elk

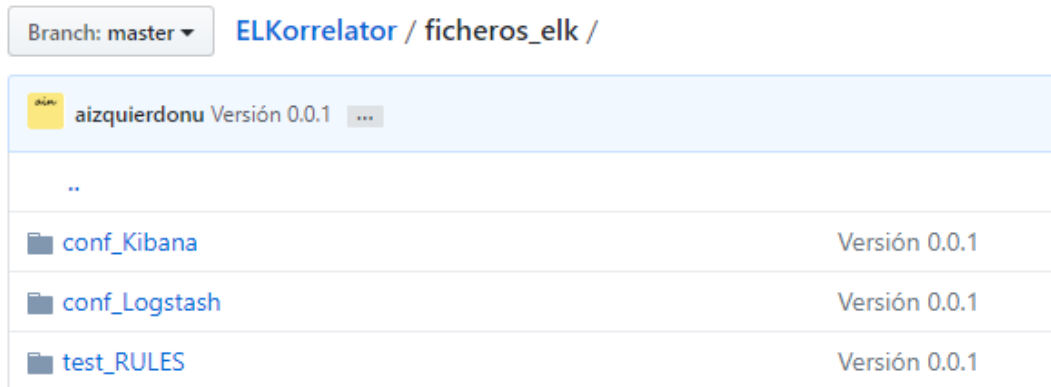


Ilustración 18: directorio ficheros_elk

En el directorio ficheros_elk encontramos:

- conf_Kibana, en el que esta disponible un fichero de configuración con las visualizaciones y Dashboard creados para ver las alertas generadas por el entorno de correlación.
- conf_Logstash, en el que encontramos una configuración básica de Logstash para poder probar las reglas de correlación.
- test_RULES, en el que encontramos diferentes scripts para generar los logs necesarios que nos hacen saltar las reglas definidas:
 - R001_fichero.py, provoca que salte la regla R001
 - R002_fichero.py, provoca que salte la regla R002
 - R003_fichero.py, provoca que salte la regla R003
 - R004_fichero.py, provoca que salte la regla R004

4.4 Interfaz web

Desde la interfaz de Kibana podemos ver las alertas que genera el entorno de correlación y gráficas sobre diferentes campos de estas alertas.



Ilustración 19: Interfaz web de Kibana

5. Pruebas

5.1 Ejemplos de alertas generadas

A continuación se va a mostrar como al ejecutar los scripts anteriormente descritos el entorno de correlación es capaz de generar alertas en base a las reglas de correlación desplegadas.

5.1.1 Regla de correlación R001

En primer lugar se ejecuta el script:

```
python R001_fichero.py >> /var/log/centralizador/in.log
```

Los logs generados se envían tanto a Elasticsearch como al Detector. El correlador detecta que se cumple la R001 y envía la alerta a Logstash que lo almacena en Elasticsearch y visualizamos así desde Kibana:

# port	Q Q [] *	60,520
t regla	Q Q [] *	R001
t regla_descripcion	Q Q [] *	Descarga_de_fichero_malicioso_y_deteccion_IDS
t regla_level	Q Q [] *	HIGH
t regla_nombre	Q Q [] *	Posible_equipo_infectado
t rule.message_n0	Q Q [] *	Jun 03 14:10:52 10.0.1.47 intel: rule_description=Detected_malicious_file_HASH srcip=10.0.1.53 hostname=peliculas.t =http://peliculas.b/descarga/plugin
t rule.message_n1	Q Q [] *	Jun 03 14:10:57 10.0.1.46 snort: [1:620:2] MALWARE Peliculas.b toolbar [Classification: trojan] [Priority: 2] {TCP} 0.1.53:1382 -> 4.4.4.4:80
t rule_description	Q Q [] *	Detected_malicious_file_HASH
t srcip	Q Q [] *	10.0.1.53
t syslog_hostname	Q Q [] *	10.0.1.47
t syslog_programname	Q Q [] *	intel
t syslog_timestamp	Q Q [] *	Jun 03 14:10:52
t tags	Q Q [] *	add_alert

Ilustración 20: Alerta generada al cumplirse R001

Lo más interesante es que se almacena un log por cada nivel de la regla de correlación, en este caso rule.message_n0 y rule.message_n1.

5.1.2 Regla de correlación R002

Ejecutamos el script:

```
python R002_fichero.py >> /var/log/centralizador/in.log
```

De igual modo vemos una alerta generada al cumplirse R002:

t contexto	Q Q [] *	R002_10.0.1.52
t host	Q Q [] *	192.168.1.124
t message	Q Q [] *	Jun 03 14:11:17 10.0.1.46 snort: [1:620:2] Scanner NESSUS [Classification: attack] [Priority: 2] {TCP} 10.0.1.52:13 _ 1.1.1.1:80
# port	Q Q [] *	60,530
t regla	Q Q [] *	R002
t regla_descripcion	Q Q [] *	Ip_realiza_un_escaneo_de_vulnerabilidades_NESSUS_y_se_detecta_que_no_es_la_primera_vez_que_lo_hace_esta_semana
t regla_level	Q Q [] *	HIGH
t regla_nombre	Q Q [] *	Escaneo_NESSUS_reincidente
t rule.message_n0	Q Q [] *	Jun 03 14:11:17 10.0.1.46 snort: [1:620:2] Scanner NESSUS [Classification: attack] [Priority: 2] {TCP} 10.0.1.52:13 _ 1.1.1.1:80
t rule.message_n1	Q Q [] *	Jun 03 11:52:09 10.0.1.46 snort: [1:620:2] Scanner NESSUS [Classification: attack] [Priority: 2] {TCP} 10.0.1.52:13 > 6.6.6.6:80
t syslog_hostname	Q Q [] *	10.0.1.46
t syslog_programname	Q Q [] *	snort
t syslog_timestamp	Q Q [] *	Jun 03 14:11:17

Ilustración 21: Alerta generada al cumplirse R002

Observamos que igual que en la alerta anterior, podemos ver los dos eventos que se han correlado.

5.1.3 Regla de correlación R003

Ejecutamos el script:

```
python R003_fichero.py >> /var/log/centralizador/in.log
```

Vemos la alerta generada al cumplirse R003

t contexto	Q Q [] *	R003_10.0.1.51
t host	Q Q [] *	192.168.1.124
t message	Q Q [] *	Jun 03 14:11:27 10.0.1.46 snort: [1:620:2] SQL Injection [Classification: attack] [Priority: 2] {TCP} 10.0.1.51:1382 5.5.5.5:80
# port	Q Q [] *	60,550
t regla	Q Q [] *	R003
t regla_descripcion	Q Q [] *	Se_detecta_un_ataque_SQL_Injection_en_IDS_pero_no_se_bloquea_en_WAF
t regla_level	Q Q [] *	HIGH
t regla_nombre	Q Q [] *	Amenaza_NO_bloqueada
t rule.message_n0	Q Q [] *	Jun 03 14:11:27 10.0.1.46 snort: [1:620:2] SQL Injection [Classification: attack] [Priority: 2] {TCP} 10.0.1.51:1382 5.5.5.5:80
t rule.message_n1	Q Q [] *	Se cumple que no hay log
t syslog_hostname	Q Q [] *	10.0.1.46
t syslog_programname	Q Q [] *	snort
t syslog_timestamp	Q Q [] *	Jun 03 14:11:27

Ilustración 22: Alerta generada al cumplirse R003

En este caso vemos el primer log de la correlación pero no el segundo, porque no le hay. Esto es porque precisamente lo que correla esta regla es que exista una alerta snort de una IP pero que no exista un log de la misma IP en el WAF.

5.1.4 Regla de correlación R004

Ejecutamos el script:

```
python R004_fichero.py >> /var/log/centralizador/in.log
```

Por último vemos que también se genera la alerta R004:

t host	🔍 📄 📄 *	192.168.1.124
t message	🔍 📄 📄 *	Jun 03 14:11:41 10.0.1.35 ftp: action=Logon srcip=6.6.6.6 resultaction=ok user=pepe
# port	🔍 📄 📄 *	60,542
t regla	🔍 📄 📄 *	R004
t regla_descripcion	🔍 📄 📄 *	Se_detecta_la_subida_de_un_fichero_desde_una_cuenta_robada
t regla_level	🔍 📄 📄 *	CRITICAL
t regla_nombre	🔍 📄 📄 *	Robo_de_cuenta_FTP_y_subida_de_fichero
t resultaction	🔍 📄 📄 *	ok
t rule.message_n0	🔍 📄 📄 *	Jun 03 14:11:41 10.0.1.35 ftp: action=Logon srcip=6.6.6.6 resultaction=ok user=pepe
t rule.message_n1	🔍 📄 📄 *	Jun 03 14:11:38 10.0.1.35 ftp: action=Logon srcip=6.6.6.6 resultaction=fail user=pepe
t rule.message_n2	🔍 📄 📄 *	Jun 03 14:11:46 10.0.1.35 ftp: action=Upload srcip=8.8.8.8 resultaction=ok user=pepe
t srcip	🔍 📄 📄 *	6.6.6.6
t syslog_hostname	🔍 📄 📄 *	10.0.1.35
t syslog_programname	🔍 📄 📄 *	ftp
t syslog_timestamp	🔍 📄 📄 *	Jun 03 14:11:41
t tags	🔍 📄 📄 *	add_alert
t user	🔍 📄 📄 *	pepe

Ilustración 23: Alerta generada al cumplirse R004

Como se puede observar en este caso hay tres mensajes asociados a tres tipos de eventos. Esto se debe a que esta regla tiene tres niveles de correlación.

Aunque en el nivel 1 se tiene como condición que se exista al menos 5 eventos de login fallidos en la alerta visualizada en Kibana solo podemos consultar uno de ellos.

6. Conclusiones

6.1 Conclusiones

Finalizado el proyecto se puede afirmar que se cumplen los principales objetivos propuestos para el entorno de correlación. Se ha obtenido un entorno de correlación que permite enriquecer los eventos, normalizarlos, almacenarlos para ser posteriormente consultados, correlar esos eventos en base a reglas que comprueban casos de uso definidos y visualizar todos estos resultados en gráficas a través de una interfaz web. Todo esto en un entorno que se ha diseñado desde el primer momento pensando en la escalabilidad, la flexibilidad, y la rapidez de búsqueda, que permite la correlación de grandes cantidades de datos utilizando un uso de memoria y procesamiento reducido.

Para el desarrollo de este proyecto se han utilizado procesos propios de la ingeniería del software junto con todos los conceptos adquiridos en las diferentes asignaturas de la asignatura, entendiendo de esta forma que los eventos generados por diferentes dispositivos de un sistema informático como pueden ser servidores web, firewalls, IDS o sistemas WAF están relacionados. Es posible que un ataque deje evidencias en diferentes sistemas, que si se relacionan correctamente, pueden alertarnos de un ataque exitoso.

La lección más importante que he aprendido a lo largo del proyecto es que es muy complicado obtener un entorno escalable, que permita que sus componentes trabajen de manera distribuida. Es un trabajo que debe empezar ya en la fase de análisis y debe consolidarse en la fase de diseño. La comunicación entre los diferentes componentes supone un reto para la fase de implementación, principalmente si se desea que esta sea de manera cifrada. Se trata del único punto que no he conseguido cumplir y se explica en mayor detalle en el apartado de mejoras futuras.

La planificación se ha seguido tal y como estaba previsto, no se han tenido retrasos en ninguna de las tareas definidas y el tiempo asignado a cada una de ellas concuerda con la cantidad de trabajo necesario para cumplirlas.

6.2 Mejoras futuras

A continuación se explican algunos de las mejoras futuras que se deberán implementar para obtener un entorno de correlación más completo.

- Comunicaciones cifradas, se trata de uno de los objetivos que no ha podido ser implementado. Existe un plugin para la pila ELK llamado X-Pack que nos permite cifrar las conexiones entre sus componentes. Por otro lado habría que hacer un desarrollo complementario para conseguir que el correlador pudiera mantener conexiones cifradas con los demás componentes.
- Más operaciones de condición para los niveles de las reglas de correlación. En estos momentos un campo solo se puede comparar con un valor para comprobar si son iguales o si está contenido. Se deben añadir condiciones como mayor o menor que.
- Agregaciones más complejas para el cumplimiento de un nivel. El correlador debe ser capaz de poder agregar los resultados por un campo para, por ejemplo, poder definir condiciones del tipo que aparezcan más de dos usuarios diferentes para una misma IP. Simplemente habría que hacer una agregación por usuario.
- Mayor personalización de los diferentes niveles de una regla.
 - Permitir que cada nivel de una regla pueda ir contra un servidor Elasticsearch diferente.
 - Permitir que cada nivel de una regla pueda ir contra un índice de Elasticsearch diferente.
 - Mostrar, si se desea, el número de resultados obtenidos para cada nivel en la alerta generada.
- Búsquedas más complejas, que permitan detectar anomalías comparando los resultados con el comportamiento habitual de un servicio. Por ejemplo que se detecte por parte de un IDS que se ha lanzado un exploit contra un servidor y a continuación se detecte que hay un usuario más de los que había ayer en el servidor víctima.

7. Glosario

- **Alerta:** Consiste en un nuevo evento que se genera cuando se cumple una regla de correlación. Contiene información de los eventos primarios que han provocado que se genere la alerta.
- **Contexto:** Se trata de una entidad que sirve para identificar aquellos análisis que ya se están correlando. Evita que dos eventos primarios generen dos procesos de correlación iguales, mejorando de esta manera el rendimiento del sistema.
- **ELKorrelator:** Nombre para definir los componentes de correlación desarrollados (Detector, Orquestador y Correlador) que hacen uso de la base de datos elasticsearch para comprobar si se cumplen reglas de correlación.
- **Enriquecer eventos:** Consiste en añadir información en los eventos a partir de valores del propio evento. Por ejemplo obtener el nombre de un equipo de red interna si en el evento original está disponible su IP.
- **Normalizar eventos:** Consiste en almacenar en un formato estándar todos los eventos.
- **Parsear eventos:** Consiste en obtener los diferentes campos contenidos en un log.
- **Regla de correlación:** Consiste en una definición estructurada de un caso de uso que se debe identificar. Para conocer en mayor detalle que estructura deben tener estas reglas es recomendable leer el manual de usuario.

8. Bibliografía

- [1]<https://www.ibm.com/blogs/sweeden/gradar-event-correlation-rules/> visitado el día 12/03/2018
- [2]<https://software.microfocus.com/en-us/products/arcsight-express-siem-appliance/overview> visitado el día 12/03/2018
- [3]<https://simple-evcorr.github.io/> visitado el día 12/03/2018
- [4]https://wiki.opennms.org/wiki/Drools_Correlation_Engine visitado el día 12/03/2018
- [5]<https://github.com/Prelude-SIEM/prelude-correlator> visitado el día 12/03/2018
- [6]<https://github.com/ossec/ossec-hids> visitado el día 12/03/2018
- [7]<https://www.elastic.co/elk-stack> visitado el día 12/03/2018
- [8]<https://github.com/Yelp/elastalert> visitado el día 12/03/2018
- [9]<http://www.ganttproject.biz/> visitado el día 12/03/2018
- [10]<https://github.com/aizquierdonu/ELKorrelator> visitado el día 03/06/2018
- [11]<https://www.modelo.org/> visitado el día 06/05/2018
- [12]<https://www.virtualbox.org/> visitado el día 06/05/2018
- [13]https://es.wikipedia.org/wiki/Proceso_Unificado_de_Rational visitado el día 06/05/2018

9. Anexo I Manual de usuario

En este documento se explican los diferentes parámetros que se pueden utilizar para crear nuevas reglas de correlación.

Campos generales

En primer lugar tenemos los campos generales de la regla, donde **###** es el identificador de la regla:

- **R###_NAME**
 - Descripción: Se trata del nombre que le vamos a dar a la regla de correlación
 - Valores permitidos: Cualquiera
 - Obligatorio: Sí
 - Valor por defecto: No hay
 - Ejemplo:
 - R001_NAME:Posible equipo infectado
- **R###_DESCRIPTION**
 - Descripción: Se trata de la descripción de la regla
 - Valores permitidos: Cualquiera
 - Obligatorio: Sí
 - Valor por defecto: No hay
 - Ejemplo:
 - Descarga de fichero malicioso y deteccion IDS
- **R###_LEVEL**
 - Descripción: Se trata del nivel de severidad de la regla de correlación
 - Valores permitidos: Cualquiera
 - Obligatorio: Sí
 - Valor por defecto: No hay
 - Ejemplo:
 - R001_LEVEL:HIGH
- **R###_CONTEXTO**
 - Descripción: Se trata del campo que se utilizará para identificar el contexto de correlación, de manera que si llegan varios eventos que podrían provocar el análisis de la misma regla y además tienen algún campo en común se puede limitar para que sólo se analice el primer evento
 - Valores permitidos: Cualquier campo del evento escrito entre corchetes
 - Obligatorio: Sí
 - Valor por defecto: No hay
 - Ejemplo:
 - R001_CONTEXTO:[srcip]

Campos del evento disparador, N0

A continuación se deben definir las condiciones que el evento debe cumplir para que se inicialice la regla de correlación, donde **###** es el identificador de la regla y **\$** el identificador de la condición:

- **R###_N0_C\$**
 - Descripción: Se trata de las condiciones que se deben cumplir para que se analice la regla de correlación.

- Valores permitidos: En primer lugar se debe definir YES o NOT para que el resultado de la consulta se devuelva sin alterar o se devuelva invertido. A continuación se define una condición booleana, utilizando el esquema:
 - Nombre del campo entre corchetes
 - Símbolo “==” que define una operación de igualdad o “=~” que define una operación de contener
 - Valor del campo entre comillas dobles
- Obligatorio: Sí
- Valor por defecto: No hay
- Ejemplo:
 - R001_N0_C0:YES [rule_description]==="Detected_malicious_file_HASH"
 - R001_N0_C1:NOT [regla]==="R001"

Campos de los diferentes niveles de correlación

A continuación se deben definir los diferentes eventos con los que se desea correlar el evento inicial teniendo en cuenta que cada tipo de evento se definirá en un nivel distinto con propiedades propias, donde ### es el identificador de la regla y \$ el identificador de la condición y % el nivel. Si se cumplen todos los niveles de la regla de correlación se generará una alerta.

- R###_N%_C\$
 - Descripción: Se trata de las condiciones que se deben cumplir para que se cumpla este nivel de la regla de correlación.
 - Valores permitidos: En primer lugar se debe definir YES o NOT para que el resultado de la consulta se devuelva sin alterar o se devuelva invertido. A continuación se define una condición booleana, utilizando el esquema:
 - Nombre del campo entre corchetes
 - Símbolo “==” que define una operación de igualdad o “=~” que define una operación de contener
 - Valor del campo entre comillas dobles
 Se pueden añadir varias condiciones separadas por la operación OR
 Se pueden referenciar los valores de campos de otros niveles
 - El valor se debe colocar entre corchetes. En primer lugar se define el nivel del que queremos obtener el valor, a continuación “___”, y por último el nombre del campo.
 - Obligatorio: Sí
 - Valor por defecto: No hay
 - Ejemplo:
 - R001_N1_C0:YES [syslog_programname]==="snort"
 - R001_N1_C1:YES [srcip]==[0___srcip]
- R###_N%_SL
 - Descripción: Se trata de un sleep en segundos antes de iniciar la correlación de eventos de ese nivel. Es interesante si los eventos de un tipo sabemos que llegan con cierto retraso.
 - Valores permitidos: cualquier número positivo
 - Obligatorio: No
 - Valor por defecto: 10
 - Ejemplo: R001_N1_SL:1
- R###_N%_SQ
 - Descripción: Se trata de un sleep en segundos entre consultas del mismo nivel.
 - Valores permitidos: cualquier número positivo
 - Obligatorio: No
 - Valor por defecto: 60

- Ejemplo: R001_N1_SQ:6
- R###_N%_Tmin
 - Descripción: Se trata del espacio temporal en segundos a analizar antes del evento del nivel anterior
 - Valores permitidos: cualquier número
 - Obligatorio: No
 - Valor por defecto: -60
 - Ejemplo:
 - R001_N1_Tmin:-1
- R###_N%_Tmax
 - Descripción: Se trata del espacio temporal en segundos a analizar después del evento del nivel anterior
 - Valores permitidos: cualquier número
 - Obligatorio: No
 - Valor por defecto: +1200
 - Ejemplo:
 - R001_N1_Tmax:10
- R###_N%_Rnot
 - Descripción: Se trata de una propiedad que define el comportamiento de la regla
 - Valores permitidos: 0 o 1
 - 0 significa comportamiento normal: Al pasar el valor RyOK la firma salta, al agotarse el tiempo la firma desaparece
 - 1 significa comportamiento inverso: Al pasar el valor RyOK la firma desaparece, al agotarse el tiempo la firma salta
 - Obligatorio: No
 - Valor por defecto: 0
 - Ejemplo:
 - R001_N1_Rnot:0
- R###_N%_RyOK
 - Descripción: Se trata del umbral, el número de eventos para que se cumpla este nivel de la regla(en comportamiento normal) o de que se desestime(en comportamiento inverso)
 - Valores permitidos: Cualquier número positivo
 - Obligatorio: No
 - Valor por defecto: 1
 - Ejemplo:
 - R001_N1_RyOK:1
- R###_N%_Event
 - Descripción: Propiedad que permite elegir con que evento encontrado nos quedamos para el siguiente nivel.
 - Valores permitidos: 0 o 1
 - Obligatorio: No
 - Valor por defecto: 0
 - Ejemplo:
 - R001_N1_Event:0

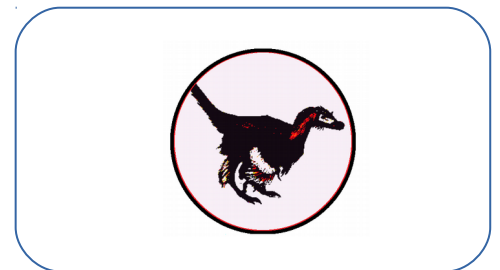
10. Anexo II Manual de instalación

En este documento se va a explicar el proceso de instalación del entorno de correlación, el cual se va a desplegar en dos máquinas distintas.

Por un lado tenemos la máquina ELK, en la cual se instalarán los componentes Logstash, Elasticsearch y Kibana. Estos componentes son los encargados de normalizar, enriquecer y almacenar los eventos. Por otro lado tenemos la máquina ELKorrelator, en la cual se instalará el componente software desarrollado. Será el encargado de correlar los eventos normalizados y enriquecidos a través de consultas a los eventos almacenados en la máquina ELK.



Componentes ELK (192.168.1.123)



ELKorrelator (192.168.1.124)

Instalación de los componentes ELK

En primer lugar vamos a instalar los componentes de la máquina ELK. Se va a comenzar a partir de un Ubuntu Server 16.04 recién instalado.

Lo primero es copiar los ficheros del directorio ficheros_elk a la máquina. En este directorio encontramos una configuración base para Logstash y scripts para generar eventos que prueban el funcionamiento de las reglas de correlación.

Para la correcta instalación se debe utilizar el usuario root.

Si se desean utilizar los scripts de test, se debe tener instalado python2:

```
apt install python-minimal
```

A continuación debemos instalar Java para que puedan funcionar Logstash y Elasticsearch.

```
apt install default-jre
```

Para instalar Elasticsearch ejecutamos:

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-6.2.3.deb  
dpkg -i elasticsearch-6.2.3.deb
```

Configuramos Elasticsearch para que responda a cualquier IP:

```
vi /etc/elasticsearch/elasticsearch.yml
```

En este caso deseamos que se pueda conectar localhost para que Logstash pueda almacenar datos y la IP de la máquina ELKorrelator(en mi caso 192.168.1.124) para

que los Correladores puedan realizar consultas a Elasticsearch. Añado la siguiente línea:

```
network.host: 0.0.0.0
```

Ahora limitamos el acceso con ufw:

```
ufw allow from 192.168.1.124 to any port 9200
ufw enable
```

Si se utilizan conexiones ssh para controlar remotamente el equipo también se debe añadir:

```
ufw allow from any to any port 22
ufw enable
```

Para instalar Logstash ejecutamos:

```
wget https://artifacts.elastic.co/downloads/logstash/logstash-6.2.3.deb
dpkg -i logstash-6.2.3.deb
```

Copiamos la configuración básica, la cual contiene los parseos necesarios para poder probar los scripts de test:

```
cp logstash.conf /etc/logstash/conf.d/
```

Sustituimos la variable "IP_ElKorrelator" por la IP de la máquina ELKorrelador, en este caso:

```
vi /etc/logstash/conf.d/logstash.conf
```

```
:%s/IP_ElKorrelator/192.168.1.124/g
```

Añadimos permisos para que el componente Correlador de ELKorrelator se pueda conectar con el Logstash con ufw:

```
ufw allow from 192.168.1.124 to any port 45555
ufw enable
```

Para instalar Kibana ejecutamos:

```
wget https://artifacts.elastic.co/downloads/kibana/kibana-6.2.3-amd64.deb
dpkg -i kibana-6.2.3-amd64.deb
```

Permitir el acceso a Kibana, en primer lugar editamos el fichero de configuración:

```
vi /etc/kibana/kibana.yml
```

Y añadimos la siguiente línea:

```
server.host: 0.0.0.0
```

También tenemos que abrir el puerto:

```
ufw allow from any to any port 5601
ufw enable
```

Añadimos una línea en el fichero crontab para borrar los ficheros temporales antiguos:

```
vi /etc/crontab
```

```
0 * * * * root find /var/log/elkorrelator/in/ -atime +1 -name "*.log" -exec rm {} \;
```

Por último reiniciamos los servicios y los activamos para que arranquen automáticamente si se reinicia la máquina:

```
systemctl restart kibana
systemctl restart elasticsearch
systemctl restart logstash
systemctl enable logstash
systemctl enable kibana
systemctl enable elasticsearch
```

Instalación los componentes ELKorrelator

Lo primero es copiar los ficheros del directorio ficheros_ELKorrelator a la máquina. En este directorio encontramos el software desarrollado, un instalador y reglas de correlación de ejemplo.

Para la correcta instalación se debe utilizar el usuario root.

En primer lugar se debe tener instalado python2 y requests:

```
apt install python-minimal
apt-get install python-setuptools
sudo easy_install -U requests
```

A continuación debemos instalar Java para que pueda funcionar Logstash(componente detector)

```
apt install default-jre
```

Lo siguiente es instalar "build-essential" para poder compilar el orquestador, escrito en c:

```
apt-get install build-essential
```

Ahora instalamos Logstash, que hará las funciones de detector:

```
wget https://artifacts.elastic.co/downloads/logstash/logstash-6.2.3.deb
dpkg -i logstash-6.2.3.deb
```

Ejecutamos el instalador, que se encargará de instalar el Orquestador y configurar el detector y los Correladores:

```
/bin/bash instalador.sh
```

Configuramos en el fichero base de los Correladores en que IP se encuentra el Elasticsearch al que tienen que consultar(en este caso 192.168.1.123), la IP en la que se encuentra el Logstash de enriquecimiento de normalización de eventos(en este caso 192.168.1.123) y el Detector(en este caso la misma máquina):

```
vi /usr/share/elkorrelator/bin/correlador.py
:%s/IP_ELASTICSEARCH/192.168.1.123/g
:%s/IP_LOGSTASH/192.168.1.123/g
:%s/IP_DETECTOR/127.0.0.1/g
```

Ahora limitamos el acceso al Detector con ufw:

```
ufw allow from 192.168.1.123 to any port 25555
ufw enable
```

Si se utilizan conexiones ssh para controlar remotamente el equipo también se debe añadir:

```
ufw allow from any to any port 22
ufw enable
```

Por último reiniciamos los servicios y los activamos para que arranquen automáticamente si se reinicia la máquina:

```
systemctl restart orquestador.service
systemctl restart logstash
systemctl enable orquestador.service
systemctl enable logstash
```

Testear instalación

Una vez que están todos los componentes instalados probamos desde la máquina ELK si las reglas de correlación funcionan correctamente:

```
mkdir -p /var/log/centralizador/
```

```
touch /var/log/centralizador/in.log
```

```
systemctl restart logstash
```

```
python test_RULES/R001_fichero.py >> /var/log/centralizador/in.log
```

```
python test_RULES/R002_fichero.py >> /var/log/centralizador/in.log
```

```
python test_RULES/R003_fichero.py >> /var/log/centralizador/in.log
```

```
python test_RULES/R004_fichero.py >> /var/log/centralizador/in.log
```

Ahora vamos a la interfaz de Kibana, accesible en el puerto 5601 de la máquina ELK, en este caso <http://192.168.1.123:5601> , y configuramos un “index pattern”:

Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

Step 1 of 2: Define index pattern

Index pattern

Ilustración 24: Elegimos logs- como index pattern*

Y elegimos el campo utilizado como "timestamp":

Step 2 of 2: Configure settings

You've defined **logs-*** as your index pattern. Now you can specify some settings before we create it.

Time Filter field name Refresh

@timestamp ▼

The Time Filter will use this field to filter your data by time.

You can choose not to have a time field, but you will not be able to narrow down your data by a time range.

Ilustración 25: Elegimos el campo @timestamp

A continuación importamos las visualizaciones por defecto de ELKorrelaor para Kibana:

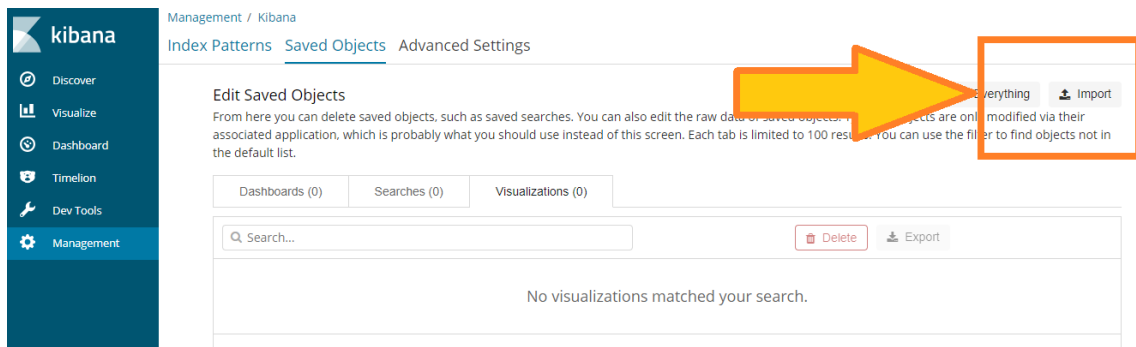


Ilustración 26: Importamos la configuración de Kibana

Por último abrimos el DASHBOARD con nombre "ELKorrelator", en el cual veremos diferentes gráficas generadas a partir de la información de las alertas de correlación:

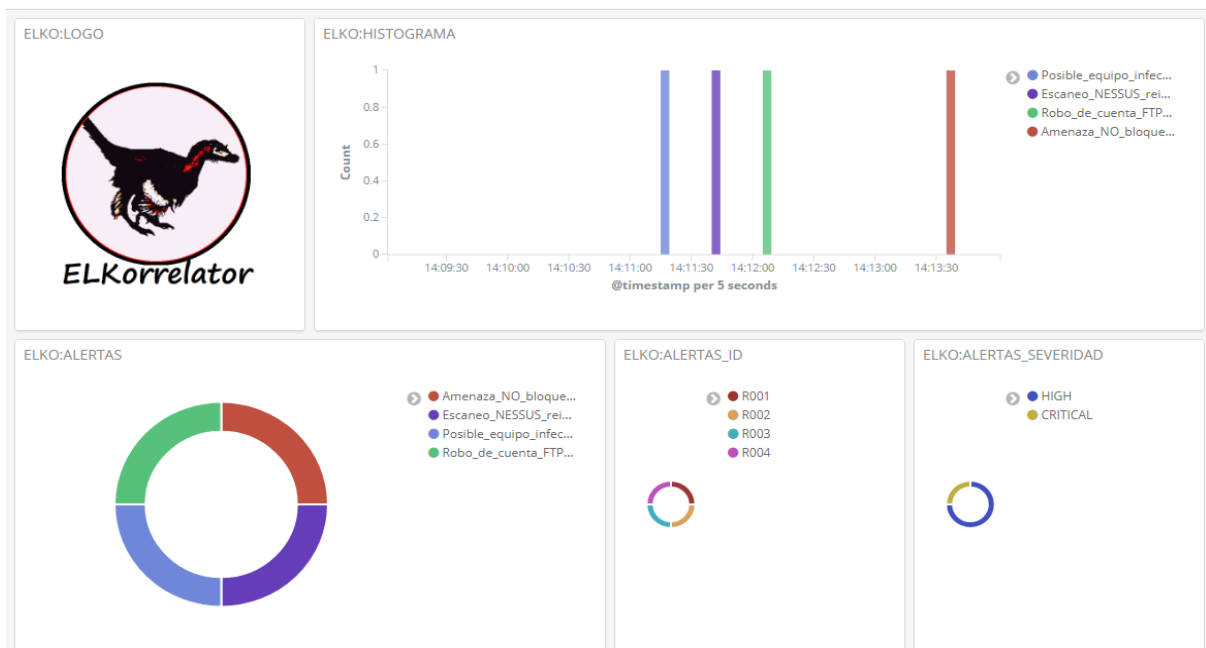


Ilustración 27: Dashboard con la información de las alertas de correlación