Proyecto Fin de Grado
Grado en Ingeniería de las
Tecnologías de Telecomunicación

Detección de malware usando herramientas de big data

Autor: María Valero Campaña

Tutor: Pablo Nebrera Herrera

Dep. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Proyecto Fin de Grado

Grado en Ingeniería de las
Tecnologías de Telecomunicación

Detección de malware usando herramientas de big data

Autor:

María Valero Campaña

Tutor:

Pablo Nebrera Herrera

Profesor Asociado

Dep. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Proyecto Fin de Grado: Detección de malware usando herramientas de big data

Autor: María Valero Campaña
Tutor: Pablo Nebrera Herrera

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

A todo aquel con el que haya compartido alguna cerveza en estos cuatro años

Agradecimientos

Este proyecto tiene mucho que agradecer. En primer lugar, a mi madre y mis hermanas. Sé que, a pesar de la distancia, siempre podré contar con vosotras.

A todos los futuros telecos con los que me he ido formando tanto profesionalmente como a nivel personal. Y agradecer en especial a los grupos "HARDY PART", "iSoftware", "Sssenssualissa" y a mis Marias por amenizarme todos estos años, los mejores y más sufridos de mi vida.

También quiero agradecerle todo el apoyo que me han aportado al equipo de ENEO Tecnología y en especial a Carlos Jiménez, Pablo Casares, Andrés Gómez y Pablo Nebrera. Gracias por estar siempre dispuestos a ayudar. Sin vosotros este trabajo no habría salido adelante.

A Jorge Santamaría y Carlos López por ser los mejores compañeros de piso, amigos y consejeros que una podría desear.

Y por último agradecer a todo el grupo de Salsa Ingenieros por estar siempre dispuestos a alegrar las noches de los jueves :)

María Valero Campaña
Sevilla, 2015

Resumen

En el presente documento se describe el desarrollo del proyecto rb-Malware (perteneciente a la empresa ENEO Tecnología). Este proyecto busca la detección de ficheros maliciosos que entran en una red local. Concretamente, se va a detallar el desarrollo de la parte relacionada con el análisis de malware.

Varios sensores de red se encargan de capturar los ficheros y almacenarlos en rb-S3. Posteriormente, la aplicación rb-sequence-oozie toma estos ficheros y los manda a un sistema de detección de malware. Este sistema de detección realiza un procesamiento por lotes y exporta los resultados a Apache Kafka. Finalmente, estos datos son leídos y mostrados por la interfaz web de rb-Malware.

Este proyecto está basado en el proyecto BinaryPig, diseñado por la empresa Endgame en 2013. El sistema de detección de BinaryPig se desarrolló sobre un cluster de Hadoop para realizar un análisis escalable.

Las tareas desarrolladas durante el procesamiento por lotes, ejecutan algunas herramientas usadas para la detección de malware. Estas herramientas de detección son YARA, VirusTotal, Kaspersky, Metascan y ClamAV.

Finalmente, se han realizado pruebas para la valoración de la release 0.4 del proyecto rb-Malware.

Abstract

This document detail the rb-Malware Project development (from ENEO Tecnología). It will specify the malware analysis part. The main objective of this project is to detect malware that comes into a local network.

Some network sensors store sniffed files into rb-S3. Then, rb-sequence-oozie application take this files and send it to the Malware Detection System. This Malware Detection System do a batch processing and transfer the results to Apache Kafka. Finally, this data is showed by rb-Malware Web Interface.

This project try to be a continuation of the project BinaryPig, which was designed by Endgame in 2013. BinaryPig use a Hadoop cluster to develop the Malware Detection System, in order to avoid scalability problems.

The jobs of batch processing used the malware tools YARA, VirusTotal, Kaspersky, Metascan and ClamAV.

Finally, I have made some test to value the release 0.4 of rb-Malware.

Índice Abreviado

Resumen V
Abstract VII
Índice Abreviado IX

1 Introducción **1**
1.1 Motivación 1
1.2 Objetivos 1
1.3 Estructura del trabajo 2

2 Herramientas utilizadas **3**
2.1 Hadoop 3
2.2 Kafka 6
2.3 Zookeeper 7
2.4 rB-S3 8

3 Introducción al análisis de malware **9**
3.1 Análisis estático 9
3.2 Análisis dinámico 10
3.3 Herramientas de análisis de malware 10

4 Implementación del Sistema **13**
4.1 Estado del arte 13
4.2 Diseño 14
4.3 Implementación de rb-oozie 16
4.4 Funciones de detección 19

5 Pruebas **25**
5.1 Escenario 25
5.2 Casos de uso 26
5.3 Conclusiones 27

6 Presupuesto **33**

7 Conclusiones **35**
7.1 Conclusión 35
7.2 Líneas futuras 35

8 Anexos **37**

Fair Scheduler **38**

Mejora de la eficiencia en Hadoop **40**

Workflow **44**

Casos de uso **50**

8.1 Caso de uso 1 - Detección de e-mails con ficheros adjuntos con Malware 50

8.2 Caso de uso 2 - Detección de Malware en tráfico web 54

KasperskyLoader **58**

KafkaStorage **59**

Bibliografía **63**

Índice de Figuras 65

Índice de Tablas 67

Índice de Códigos 69

Índice alfabético 71

Glosario 71

Índice

Resumen V
Abstract VII
Índice Abreviado IX

1 Introducción **1**
1.1 Motivación 1
1.2 Objetivos 1
1.3 Estructura del trabajo 2

2 Herramientas utilizadas **3**
2.1 Hadoop 3
2.1.1 HDFS 3
2.1.2 MapReduce 4
2.1.3 YARN 4
2.1.4 Pig 4
2.1.5 Oozie 5
2.1.6 Colas de prioridad 5
2.2 Kafka 6
2.3 Zookeeper 7
2.4 rB-S3 8

3 Introducción al análisis de malware **9**
3.1 Análisis estático 9
3.2 Análisis dinámico 10
3.3 Herramientas de análisis de malware 10
3.3.1 Yara 10
3.3.2 ClamAV 11
3.3.3 VirusTotal 11
3.3.4 Metascan 11
3.3.5 Kaspersky 11
3.3.6 Cuckoo 11
3.3.7 Otras herramientas de análisis contempladas 11
Fuzzy Hash 12
Malware Information Sharing Platform (MISP) 12
Whitelist 12

4 Implementación del Sistema **13**

4.1 Estado del arte 13
4.1.1 Arquitectura de BinaryPig 13
4.2 Diseño 14
4.2.1 Funcionamiento de rb-sequence-oozie 14
4.3 Implementación de rb-oozie 16
4.3.1 OozieLeader 16
4.3.2 OozieManager 17
4.3.3 OoziePeon 18
4.4 Funciones de detección 19
4.4.1 Clases de Pig 19
KafkaStorage 19
KasperskyLoader 22
4.4.2 Workflow 22

5 Pruebas 25
5.1 Escenario 25
5.2 Casos de uso 26
5.2.1 Caso de uso 1 - Detección de e-mails con ficheros adjuntos con Malware 26
5.2.2 Caso de uso 2 - Detección de Malware en tráfico web 26
5.3 Conclusiones 27

6 Presupuesto 33

7 Conclusiones 35
7.1 Conclusión 35
7.2 Líneas futuras 35

8 Anexos 37

Fair Scheduler 38

Mejora de la eficiencia en Hadoop 40
TeraSort 40
Factores que afectan a la ejecución de tareas MapReduce 40
mapred-site.xml 41
yarn-site.xml 42
hdfs-site.xml 42
core-site.xml 42

Workflow 44

Casos de uso 50
8.1 Caso de uso 1 - Detección de e-mails con ficheros adjuntos con Malware 50
8.2 Caso de uso 2 - Detección de Malware en tráfico web 54

KasperskyLoader 58

KafkaStorage 59

Bibliografía 63

Índice de Figuras 65
Índice de Tablas 67
Índice de Códigos 69
Índice alfabético 71

Glosario 71

1 Introducción

The problem of viruses is temporary and will be solved in two years.

John McAfee (fundador de McAfee Antivirus), 1988

1.1 Motivación

El uso de internet ha aumentado enormemente en los últimos años, tanto a nivel doméstico como a nivel empresarial[27]. Esto también ha originado que haya aumentado el número de ataques e intrusiones en los sistemas informáticos de todo el mundo.

El término malware (malicious software) hace referencia a todo aquel software que perjudica a un dispositivo. Puede tratarse de un virus, un Troyano, una puerta trasera (backdoor), un programa espía (spyware) o un gusano[11].

Según Securelist[13] en el segundo cuarto de 2015 Kaspersky Lab solutions ha detectado y repelido un total de 379,972,834 ataques. Además, se han detectado un total de 291,887 de nuevas aplicaciones móviles maliciosas.

Para evitar ser infectados por ficheros malware, que tienen como objetivo obtener recursos, información o dañar el dispositivo sin consentimiento del propietario, debemos proteger nuestros sistemas. Esto se ha hecho típicamente con el uso de antivirus.

Los anti-virus o anti-malware son aplicaciones que buscan prevenir, detectar y eliminar malware. El método más usado para identificar posibles virus, es el basado en firmas. Para identificar los ficheros maliciosos, se compara su resumen o hash con una base de datos de firmas malware. Esto conlleva dos grandes problemas. El primero es el de la escalabilidad[1], ya que no es posible manejar tal cantidad de muestras[2] únicas de malware. El segundo inconveniente es la detección del malware que no se encuentra recogido en ninguna base de datos de firmas de malware.

1.2 Objetivos

En este proyecto, se busca desarrollar un sistema que detecte los ficheros maliciosos que entran en una red local.

[1] Entendiéndose escalabilidad como la propiedad deseable de un sistema, una red o un proceso, que indica su habilidad para reaccionar y adaptarse sin perder calidad, o bien manejar el crecimiento continuo de trabajo de manera fluida, o bien para estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos. *(Fuente: Wikipedia)*

[2] A lo largo del proyecto, muestra se va a considerar como un archivo de malware, ya sea virus, gusano o troyano.

Estos ficheros van a llegar a un sistema de almacenamiento (rb-S3) a través de sensores colocados en distintos puntos de la red local. La aplicación rb-sequence-oozie, tomará estos ficheros y los mandará a un sistema de detección. Este sistema de detección realizará un procesado por lotes.

Las tareas realizadas durante este procesamiento por lotes usan métodos basados en firmas, análisis estático (mediante el estudio del código) y análisis dinámico (observando el comportamiento durante la ejecución del fichero) para realizar el análisis de los ficheros. Para llevar a cabo este análisis, se ha decidido usar una serie de frameworks de análisis de ficheros y detección de malware, estos son: YARA, Virustotal, Metascan, ClamAV, Kaspersky y Cuckoo.

Para abordar el problema de la escalabilidad, el sistema de detección se ha desarrollado haciendo uso de aplicaciones de big data. Para ello se ha definido una serie de funciones que permiten conectar la aplicación Hadoop con los frameworks de análisis de ficheros y detección de malware.

1.3 Estructura del trabajo

En el presente capítulo se ha introducido el tema del trabajo y algunos conceptos como malware, análisis estático, análisis dinámico y escalabilidad. Además se ha expuesto el objetivo del trabajo, que es la implementación de una herramienta de detección de malware de forma escalable.

En los siguientes capítulos se tratará:

- Capítulo 2: Herramientas utilizadas. En este apartado se definen las principales herramientas utilizadas durante el desarrollo del sistema. Estas son Apache Hadoop, Apache Pig, Apache Oozie, Kafka, Zookeeper y rb-S3.
- Capítulo 3: Introducción al análisis de malware. En este apartado se hace una pequeña introducción al análisis estático y dinámico de malware. Posteriormente, se presentan las herramientas de análisis usadas para este proyecto, estas son ClamAV, VirusTotal, Metascan, Yara, Kaspersky y Cuckoo. También se habla un poco sobre otras herramientas de análisis que son Fuzzy Hash, MISP y listas blancas de software.
- Capítulo 4: Implementación del sistema. En este apartado se empieza hablando del sistema BinaryPig de Endgame como estado del arte en el que está basado este proyecto. Posteriormente se habla sobre el diseño y desarrollo de rb-sequence-oozie y las funciones desarrolladas para la detección de malware.
- Capítulo 5: Pruebas. En este capítulo se presenta el escenario de prueba y los casos de uso que se han probado.
- Capítulo 6: Conclusiones. En este apartado se presentan tanto las conclusiones como las líneas futuras que puede ser de interés para la mejora de la aplicación.

Por último, encontramos los anexos. Estos que contienen información relacionada con el proyecto pero no son necesarios para su entendimiento. Sin embargo, reflejan parte del trabajo realizado y puede ser útiles e interesantes para el lector.

2 Herramientas utilizadas

Hiding within those mounds of data is knowledge that could change the life of a patient, or change the world

Richard A. Clarke (Investigador en biomedicina informática), 2012

A continuación, se va a realizar una breve descripción de los componentes utilizados en el proyecto rB Malware.

2.1 Hadoop

Hadoop es una aplicación open source que se ha convertido en un estándar de facto en la industria del procesamiento de grandes volúmenes de datos. Es la plataforma más usada para guardar y analizar datos[14].

Hadoop nace a raíz de dos papers publicados por Google en 2003 y 2004. El primero de ellos definía el sistema de ficheros de Google y el otro explicaba un modelo de cómputo para clústeres [1] llamado MapReduce. Doug Cutting y Mike Cafarella crearon Hadoop en 2005 basándose en estas tecnologías.

Hadoop tiene tres partes bien diferenciadas:

- El sistema de ficheros distribuido de Hadoop (HDFS por sus siglas en inglés).
- La plataforma de computación MapReduce.
- El ecosistema de Hadoop, que es un conjunto de herramientas o módulos que usan HDFS y MapReduce para ayudar a la gestión y configuración del cluster, programación de tareas y gestión de datos en Hadoop. Concretamente, para este trabajo, se han usado las herramientas Pig y Oozie.

2.1.1 HDFS

Es el encargado de almacenar los datos. Tiene como objetivo proporcionar una alta capacidad de almacenamiento, robusta y económica[14]. Es un sistema de ficheros de tipo "escribe una vez y lee muchas"[2].

[1] El término clúster (del inglés cluster, "grupo" o "raíz") se aplica a los conjuntos o conglomerados de computadoras unidos entre sí normalmente por una red de alta velocidad y que se comportan como si fuesen una única computadora.*(Fuente: Wikipedia)*

[2] La API (Application Programming Interface) de HDFS permite modificar los ficheros pero no sobreescribirlos.

Los ficheros que se guardan en HDFS se dividen en bloques de 64MB o más y se distribuyen a los largo de los nodos del cluster[7]. Por defecto, se hacen 3 copias de cada bloque y son alojadas en diferentes nodos del cluster. Así se prevé de que un fallo en un servidor conlleve a una pérdida de datos.

Los nodos que se encargan de guardar la información se llaman datanodes. El nodo que se encarga de conocer la ubicación de los bloques pertenecientes a cada fichero es el namenode.

2.1.2 MapReduce

Fue el primer framework de programación para el desarrollo de aplicaciones en Hadoop[7].

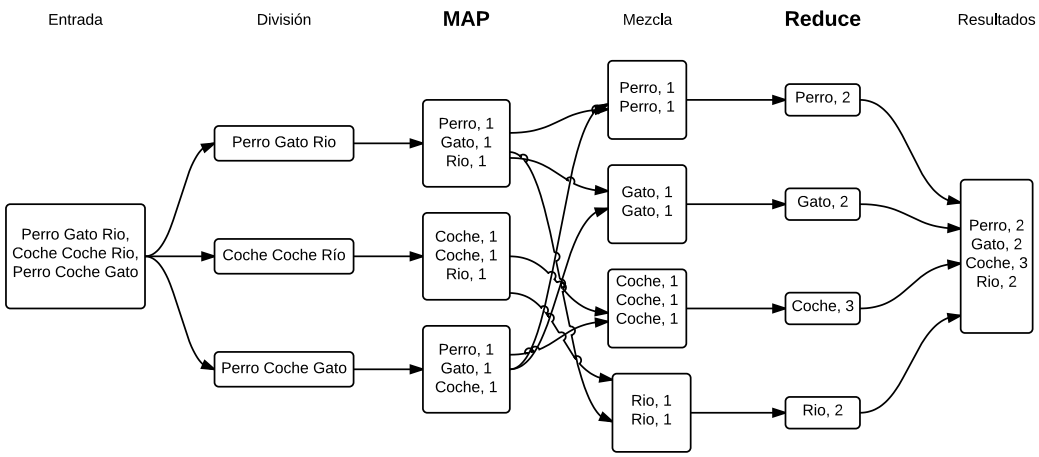

Figura 2.1 Ejemplo de la ejecución de trabajos MapReduce.

Los trabajos que definen las tareas MapReduce tienen dos partes fundamentales, "Map" y "Reduce". Para la primera tarea, el nodo gestor de tareas (Jobtracker) les da a los nodos esclavo (tasktracker) un trozo del fichero que se quiere analizar. Cada tasktracker analiza una parte del fichero y exporta los resultados a Hadoop, generando como salida una lista (clave,valor). En la parte de "Reduce" se agregan los resultados.

Las tareas MapReduce se programan en Java. Sin embargo, se han definido lenguajes de más alto nivel, como Pig o Hive, que permiten realizar tareas MapReduce.

2.1.3 YARN

En mayo de 2012, se lanzó la versión 2.0 de Hadoop, que traía YARN incorporado.

YARN[14] es una capa que se encuentra entre HDFS y MapReduce. Esta capa permite a otras herramientas que estaban fuera del sistema de Hadoop (como Spark o Giraph) a existir nativamente en el cluster de Hadoop. De esta forma, YARN provee de una interfaz que puede ser usada por distintas herramientas para ejecutar tareas.

2.1.4 Pig

Latin Pig[14][8] es un lenguaje de alto nivel para el procesamiento de datos. Nos permite programar tareas MapReduce de forma más rápida y más fácil de mantener que la programación en Java. Esto se debe en a su sintaxis, similar a SQL, que permite que programas de 100 líneas de código en Java se transformen en scripts de 10 líneas de código en Latin Pig.

Originalmente desarrollado por Yahoo en el 2006 fue adoptado por la Apache Software Foundation un año después.

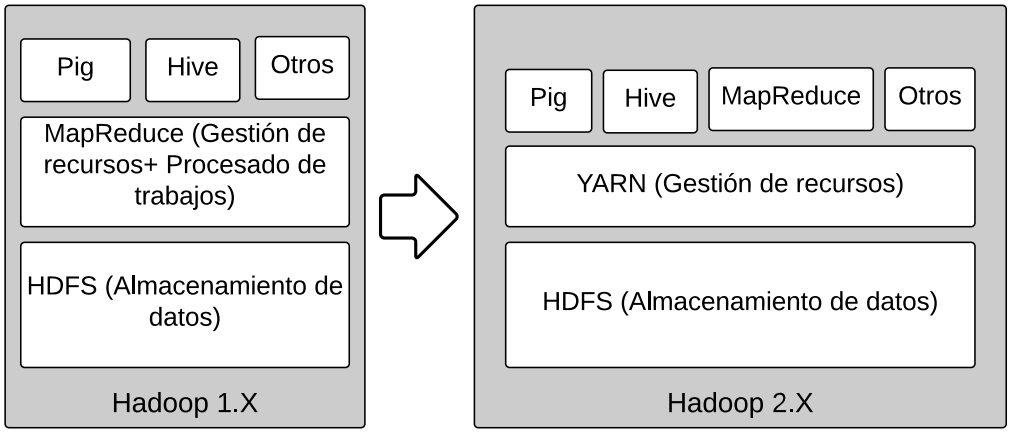

Figura 2.2 Evolución de la estructura de Hadoop.

Su eslogan "Pig eats everything" muestra su capacidad para consumir cualquier tipo de datos. En cierta forma, Pig es una gran herramienta ETL (carga, transforma y almacena de datos). Estas funciones de carga y almacenamiento se pueden realizar definiendo funciones UDF. Hay muchas de estas funciones compartidas en Piggy Bank[8][15].

También se ha elegido la plataforma Apache Pig frente a otras como Hive o Java ya que permite trabajar con datos tanto estructurados como no estructurados.

2.1.5 Oozie

Oozie[14][16] es un planificador de workflows(flujos de trabajo) usado para gestionar trabajos de Hadoop. Se suele usar para tareas complejas que deben ser ejecutadas periódicamente o que tienen varias partes o scripts de ejecución.

Oozie se encarga de ejecutar acciones (trabajos o tareas). Las acciones que realiza pueden ser trabajos MapReduce, Hadoop hdfs, Pig, SSH, HTTP, eMail y sub-flujos-de-trabajo de Oozie. También se pueden añadir extensiones para soportar otros tipos de acciones.

Estas acciones se ejecutan tal y como están definidas en una gráfica DAG que describe qué acciones se deben ejecutar y en qué orden. Esto se define en un fichero XML (concretamente hPDL, lenguaje de definición de procesos de hadoop).

Oozie está diseñado para mandar trabajos a un cluster donde se esté ejecutando Hadoop y gestione la ejecución de estos trabajos. Cuando un trabajo se complete, el sistema remoto avisa a Oozie y este puede proceder a ejecutar la siguiente acción del workflow.

Para este trabajo se han definido ficheros workflow con los script de Pig a ejecutar. Estos trabajos se ejecutan de forma secuencial. Por tanto, un trabajo no puede empezar hasta que el trabajo anterior haya terminado.

Para la definición de un workflow se usan etiquetas para el control de flujo y etiquetas donde se definen las acciones o trabajos a ejecutar:

- Etiquetas de control de flujo: definen el inicio y final del flujo de trabajo y ofrece el mecanismo de controlar el path de ejecución del flujo de trabajo.
- Etiquetas de acción es donde el flujo de trabajo ejecuta los trabajos.

2.1.6 Colas de prioridad

Para poder ejecutar varias tareas en hadoop sin que los trabajos de rB Malware monopolicen todo el sistema, se ha decidido usar una cola de prioridad[17]. El uso de una cola de prioridad permite compartir los recursos del cluster de forma justa.

Las principales colas de prioridad que se usan son:

- Capacity Scheduler: es un scheduler para tareas MapReduce que permite que varios usuarios compartan el cluster de Hadoop de forma segura. Asigna los recursos de la forma más adecuada cumpliendo con las limitaciones de las capacidades asignadas.
- Fair Scheduler: es un scheduler para Hadoop que permite que aplicaciones YARN compartan los recursos en un cluster.

Dado que para este trabajo se usa la versión 2.4.0 de Hadoop, se ha implementado el Fair Scheduler. Este está descrito en mayor detalle en el anexo *Fair Scheduler*.

También se han descrito otras formas de mejorar la eficiencia de Hadoop en el anexo *Mejora de la eficiencia de Hadoop*.

2.2 Kafka

Apache Kafka[18] es un sistema de mensajería Publicador/Suscriptor distribuido, particionado y open source. La publicación de mensajes permite conectar varias aplicaciones con la ayuda del enrutamento de mensajes.

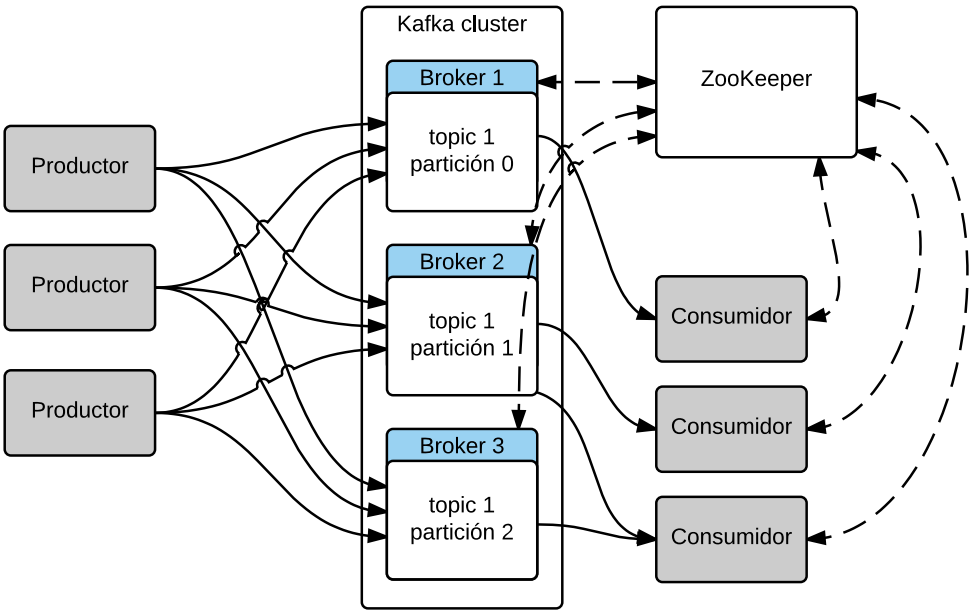

Figura 2.3 Arquitectura de un cluster de Kafka.

Un cluster de Kafka tiene principalmente 5 componentes:

- Topic. Un topic es el nombre que se le da a una lista de mensajes publicados. En Kafka los topics están particionados y cada partición tiene una secuencia de mensajes ordenados. Cada mensaje en la partición asignada tiene asignado un ID único llamado offset.
  El cluster de Kafka mantiene un log de los mensajes publicados en cada partición por un período de tiempo configurable. Estos mensajes se logean tanto si son consumidos como si no. Después, de este tiempo los mensajes son borrados y se libera la memoria que estaba reservada.

- Broker. Un broker es un proceso servidor que está corriendo en los servidores del cluster de Kafka. Cada servidor puede tener uno o varios brokers ejecutandose. Los brokers, por defecto, escuchan los mensajes en el puerto 9092.
- Zookeeper. Proporciona una interfaz de comunicación entre los brokers de Kafka y los consumidores de Kafka.
- Productores. Los productores publican los mensajes a un determinado topic. El productor es responsable de elegir la partición adecuada en la que publicar el mensaje.
- Consumidores. Los consumidores de Kafka pueden tener instancias en distintos procesos o máquinas, cada consumidor de Kafka tienen un identificador de grupo (lo que permite el balanceo de carga).

2.3 Zookeeper

Zookeeper[19] es una plataforma que permite realizar tareas coordinadas entre sistemas distribuidos. Una tarea coordinada es una tarea que involucra a varios procesos.

Estas tareas pueden ser de coordinación o de contención. La coordinación implica la necesidad de los procesos de comunicarse para realizar una tarea de forma conjunta. La contención se refiere a una situación en la que dos procesos no pueden ejecutarse de forma concurrente y tienen que esperarse el uno al otro.

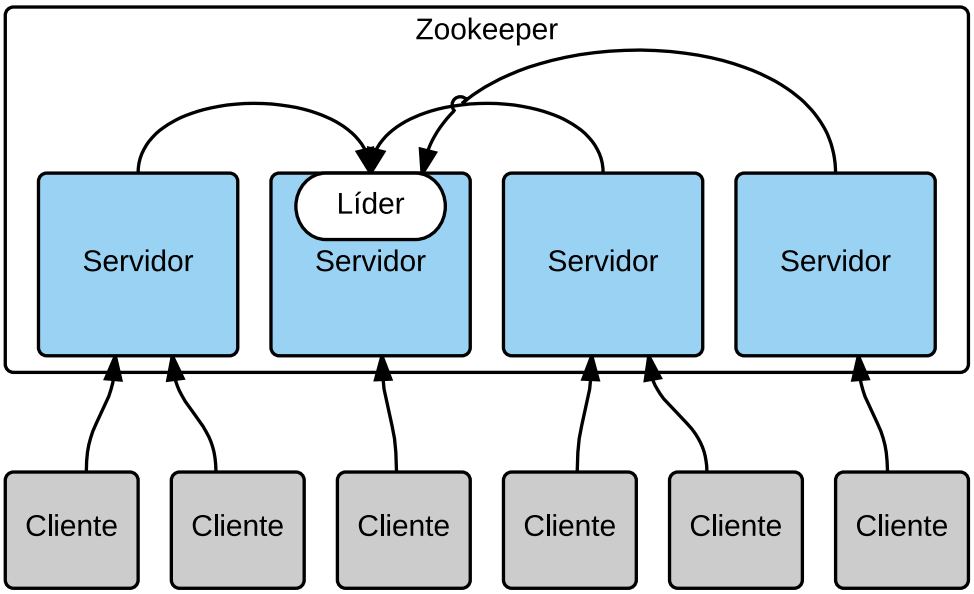

Figura 2.4 Arquitectura de un cluster de Zookeeper.

Define maestros y clientes. Proporciona redundancia, pudiendo existir varios maestros de ZooKeeper [3], permitiendo a los clientes poder acceder a la información en cualquiera de ellos.

Los nodos que se comunican con ZooKeeper, guardan sus datos en un espacio de nombres jerárquico como hace un sistema de archivos. Los clientes pueden leer y escribir en los nodos y de esta forma tienen un servicio de comunicación compartido.

[3] A pesar de haber varios maestros, sólo el nodo "leader" puede escribir en el disco duro para guardar datos.

Este servicio será usado por la aplicación rb-sequence-oozie. Además, ZooKeeper es usado por Apache Kafka y Apache Hadoop.

2.4 rB-S3

rB-S3 es una base de datos mantenida por el equipo de Redborder. Esta implementa el sistema Riak.

Riak[20] es un sistema de ficheros de tipo clave-valor, NoSQL y distribuido. Las características principales de Riak son las siguientes.

- Alta disponibilidad. Permite leer y escribir de varios servidores.
- Simplicidad operacional. Añadir nuevas máquinas a un cluster de Riak es una tarea relativamente sencilla.
- Tolerancia a fallos. Riak, por defecto, replica tres veces los datos a lo largo del cluster.
- Escalabilidad. Riak distribuye los datos a lo largo de un cluster y aumenta el rendimiento de forma casi lineal a medida que se añade capacidad.
- No necesita de nodo maestro. Las peticiones pueden ser atendidas por cualquier nodo del cluster. Esto lo convierte en un sistema muy robusto.

3 Introducción al análisis de malware

I think computer viruses should count as life. I think it says something about human nature that the only form of life we have created so far is purely destructive. We've created life in our own image

Stephen Hawking (físico teórico), 1994

Cualquier software que cause daños al usuario, dispositivo o red (como troyanos, virus, gusanos, rootkits o spyware) puede ser considerado como malware.

El malware puede aparecer de distintas formas. Las herramientas de análisis deben ser capaces de detectarlos, determinar el daño causado y detectar si se han infectado otros ficheros. Una vez que se localiza un fichero malicioso, se añade a una base de datos de firmas de malware para asegurar que el mismo fichero no vuelve a entrar en la red o dispositivo.

Hay dos formas fundamentales de abordar el análisis de malware[3]: análisis estático y dinámico. El análisis estático examina el malware sin llegar a ejecutarlo. El análisis dinámico observa lo que ocurre en el sistema cuando es ejecutado (ficheros modificados, conexión con máquinas remotas,...).

3.1 Análisis estático

Consiste en examinar el fichero y proveer información sobre su estructura y funcionalidad. Este análisis es más básico y seguro (ya que no estamos ejecutando código malicioso) pero no es tan efectivo contra ficheros malware que están ofuscados.

El método más usado para identificar malware es basado en firmas. Cuando un fichero sospechoso entra en el sistema, se genera su hash y se compara con una base de datos de firmas de malware. El problema reside en que los autores de malware pueden modificar fácilmente su código y así evadir este tipo de análisis.

Otro método de análisis, es las búsqueda de cadenas de caracteres en el programa. Es posible que en el código haya alguna URL o IP que nos dé una pista del comportamiento de la aplicación. Sin embargo, muchas veces las herramientas de detección de cadenas de caracteres interpretan erróneamente datos o direcciones de memoria como cadenas de caracteres.

El malware empaquetado u ofuscado contiene muy pocas cadenas de caracteres. Esto nos dificulta el conocer la funcionalidad del fichero. Sin embargo, si encontramos un fichero que contiene pocas cadenas de caracteres significativas, nos sugiere que puede estar ofuscado.

En el caso de los ficheros PE (Portable Executable), lo que más información nos aportan es las librerías que importan. Estas están definidas en la cabecera del fichero.

3.2 Análisis dinámico

Se basa en técnicas que observan el comportamiento de la ejecución de malware en el sistema. Este tipo de análisis nos permite detectar el funcionamiento del malware (aunque esté ofuscado).

Para llevar a cabo un análisis dinámico de malware, se requiere de un ambiente seguro donde se pueda ejecutar cualquier tipo de fichero malicioso sin que haya riesgo de dañar al dispositivo o la red. Estos análisis se suelen llevar a cabo en máquinas virtuales o sandboxes.

Una sandbox es una máquina virtual con software preinstalado para analizar la ejecución de malware. Suele realizar acciones como generar una red virtual para observar la interacción del malware con la red.

Los problemas de una sandbox, es que si el malware requiere opciones por línea de comandos no se va llegar a ejecutar ningún código ya que el programa se va a quedar esperando una entrada.

Otro problema, es que hay software programado para ejecutarse a una determinada hora del día. Por ejemplo, si analizamos el malware por el día y este sólo se ejecuta por la noche, no se va a detectar ninguna actividad sospechosa.

También hay malware capaz de detectar si están corriendo en una máquina virtual y son capaces de cambiar su comportamiento. Esto lo hace para evitar el ser detectados como aplicaciones maliciosas en caso de estar ejecutandose en una sandbox.

Otro problema es que la sandbox debe tener un Sistema operativo donde el malware pueda ejecutarse. Por ejemplo, un fichero puede servir para dañar Windows XP pero no ejecutarse correctamente en Windows 7.

3.3 Herramientas de análisis de malware

A continuación, se van a definir brevemente las herramientas de análisis de malware[1] usadas para este proyecto.

3.3.1 Yara

YARA[1] es un sistema multiplataforma disponible para Windows, Linux y Mac OS. Es una herramienta que ayuda a identificar y clasificar familias de malware. Con YARA, se describen familias de malware basadas en información binaria o textual de los ficheros. Estas descripciones, se guardan en reglas que son aplicadas a los ficheros para determinar si pertenece a una clase o no.

Por ejemplo, podemos crear una regla donde se detecta aquellos ficheros que acceden a una de dos posibles URL maliciosas.

Código 3.1 Regla de Yara.

```
rule BadBoy
{
    strings:
          $a = "http://foo.com/badfile1.exe"
          $b = "http://bar.com/badfile2.exe"
    condition:
          $b or $c
}
```

[1] Para obtener maś información sobre herramientas de análisis malware, se aconseja leer el libro Malware Analyst's Coockbook[6].

Las reglas de yara permiten buscar cadenas de caracteres, expresiones regulares o patrones binarios.

3.3.2 ClamAV

ClamAV[2] es un antivirus, que tiene como objetivo detectar troyanos, virus, malware y otras amenazas. Se ha convertido en un estandar de facto para el análisis de software en servidores de email. Escanea archivos y ficheros comprimidos (ZIP, RAR, ARJ,TAR,...) y además soporta ficheros especiales como HTML, RTF o PDF.

La aplicación incluye un escáner demonio (clamd) y una aplicación de análisis por línea de comandos (clamscan). También contiene una herramienta para actualizar la base de datos de firmas llamada freshclam.

3.3.3 VirusTotal

VirusTotal[21] es un sitio web desarrollado por Hispasec que permite el análisis online de ficheros. Este sistema usa un total de 54 antivirus para analizar los ficheros en busca de malware.

VirusTotal provee de una API para Java que permite automatizar procesos de análisis. Sin embargo, para su uso requiere de una clave que se consigue al registrarse en la comunidad de VirusTotal. Está limitado a 4 peticiones por minuto.

3.3.4 Metascan

Metascan[22] es una herramienta online gratuita de análisis de malware. Este sistema una el escaneo de ficheros con varios motores de análisis (como los antivirus de ESET, AVG o MacAfee).

Al igual que VirusTotal, posee una API para Java y requiere de una clave conseguida el registrarse en el portal OPSWAT. Tiene un límite de 1500 peticiones de búsqueda de hash y 25 peticiones de análisis de ficheros por hora.

3.3.5 Kaspersky

Es una empresa internacional de seguridad que ofrece una gran variedad de productos de detección de malware. Ofrece herramientas de análisis de ficheros para windows, Internet Security para Windows, Android y Mac, un gestor de contraseñas y Endpoint Security. Esta última es una plataforma que permite a las empresas monitorizar dispositivos.

Para este trabajo, se ha usado la herramienta Kaspersky Anti-Virus 8.0 para Servidores de Ficheros de Linux[23]. Sin embargo, esta parte no se ha incluido en las pruebas ya que forma parte de la release 0.8 del proyecto rB Malware.

3.3.6 Cuckoo

Cuckoo[4] es un sistema de código libre que automatiza el análisis dinámico de malware. Es uno de los sandbox más usados actualmente. Se encarga de ejecutar y analizar ficheros en tiempo real. El sandbox Cuckoo consiste en un software central de gestión que maneja la ejecución y análisis de ficheros. Cada análisis se ejecuta en una máquina virtual aislada.

La infraestructura de Cuckoo está compuesta por una máquina Host (con el software de gestión) y un conjunto de máquinas huésped (máquinas virtuales que ejecutan los ficheros). El Host ejecuta todo el proceso de análisis mientras que cada huésped se encarga de ejecutar los ficheros de forma segura.

3.3.7 Otras herramientas de análisis contempladas

A continuación, se van a describir un conjunto de herramientas que no se han llegado a desarrollar para el trabajo pero que son bastante útiles en el análisis de malware.

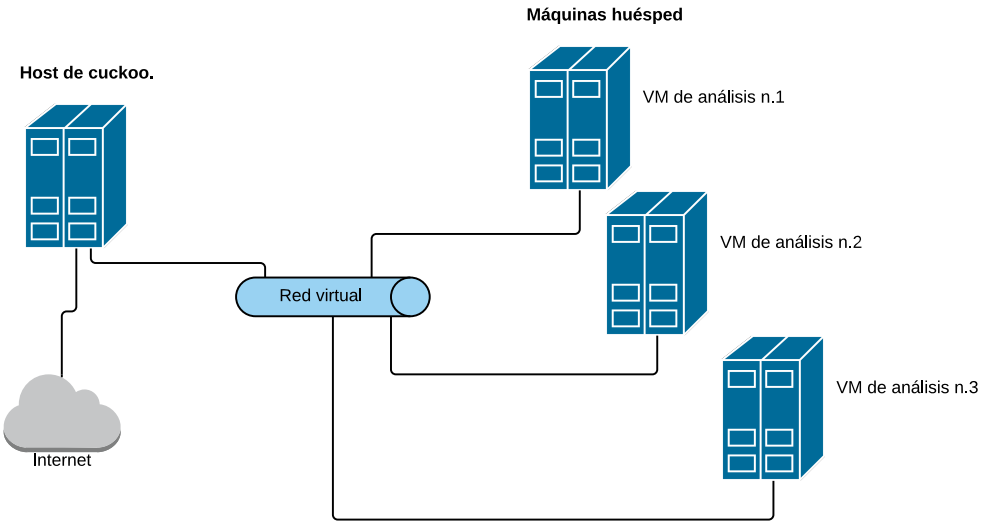

Figura 3.1 Ejemplo de arquitectura de Cuckoo.

Fuzzy Hash

Los algoritmos de hashing buscan identificar un fichero de forma única. Si nos basamos únicamente en esto para detectar malware, es muy fácil para los autores de malware modificar su código y evadir los sistemas de detección.

Lo que busca el fuzzy hash[24] es detectar cómo de parecidos son dos ficheros. De forma que es capaz de detectar si un software es una modificación de otro software comparando el fuzzy hash de los ficheros.

Malware Information Sharing Platform (MISP)

La plataforma MISP[25] permite compartir, almacenar y relacionar información de malware y ataques en tiempo real.

Whitelist

Consiste en una base de datos con hashes de software que se conoce que no es malicioso. Esta es una muy buena forma de evitar los falsos positivos en el análisis de malware.

Varias empresas de detección como VirusTotal, poseen una base de datos privada de whitelist. Kaspersky tiene la Application Advisor[26] que es una versión beta de una whitelist de malware (aún no posee ninguna API pública).

4 Implementación del Sistema

You can't defend. You can't prevent. The only thing you can do is detect and respond

Bruce Schneier (criptógrafo)

El diseño de la arquitectura del proyecto está basado en BinaryPig. BinaryPig es un proyecto desarrollado en Hadoop que busca la extracción de datos binarios de ficheros malware. Con esto, busca dar solución al problema de la detección del malware de forma escalable.

En este apartado, se estudiará la arquitectura del sistema BinaryPig y posteriormente se explicará el diseño del sistema rb-sequence-oozie y las funciones de detección para Hadoop desarrolladas.

4.1 Estado del arte

En general, la detección de ficheros malware se consigue con la comparación del hash de un fichero con la de una base de datos de hashes de muestras malware. El hecho de coleccionar estas muestras "únicas" de malware [1] y el análisis de ficheros está empezando a provocar un problema de escalabilidad. El ejemplo de McAfee, que recibió alrededor de 100.000 muestras de malware por día en 2013, da a entender que debemos buscar soluciones escalables para el análisis de ficheros.

BinaryPig[9] busca abordar este problema usando Apache Hadoop y Apache Pig combinadas con herramientas de análisis existentes para lograr un análisis archivos en tiempo real.

4.1.1 Arquitectura de BinaryPig

Hadoop es una herramienta de análisis que es menos eficiente en el momento de analizar ficheros que ocupan poca memoria. Por lo tanto, BinaryPig pasa las muestras de entrada a Hadoop en forma de ficheros secuenciales. Esta herramienta (llamada SequenceFile y creada para BinaryPig) genera una colección de pares clave/valor donde se representa el ID o hash del fichero y el contenido del mismo.

Para realizar el análisis, se toma el fichero secuencial y se realiza un procesamiento por lotes. Para la programación de las tareas, se han usado scripts de Pig. Estos usan cargadores definidos en Java para Pig (funciones UDF) con los que se leen los ficheros secuenciales y se almacenan los datos en HDFS. Los cargadores definidos en BinaryPig son los siguientes.

- **Generic Script Loader**. Ejecuta un script/programa que esté especificado en el path.

[1] Los autores de malware utilizan múltiples métodos con los que ofuscar sus archivos, lo que modifica el hash del fichero y genera más muestras de malware

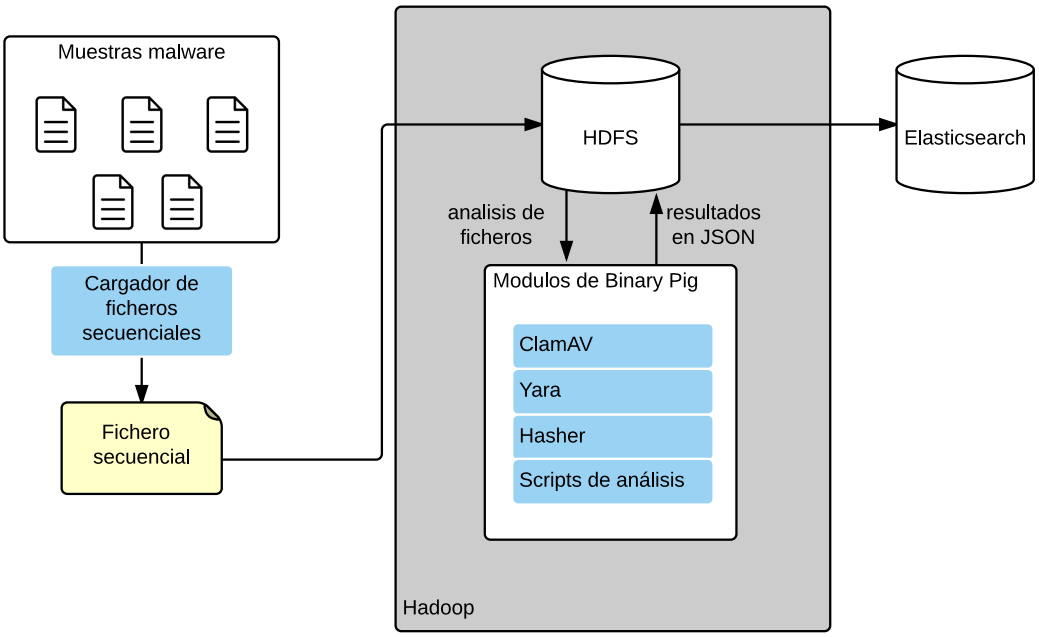

Figura 4.1 Arquitectura del sistema BinaryPig.

- **Generic Daemon Loader**. Escribe binarios en un path y le da esos caminos a un proceso demonio de análisis que está corriendo en la máquina.
- **ClamAV Loader**. Se definen dos cargadores para ClamAV, uno para el análisis usando el script de la aplicación y otro que llama al proceso demonio que está corriendo en la máquina.
- **Yara Loader**. Ejecuta un script en Python que ejecuta Yara para el análisis del fichero en función a unas reglas definidas.
- **Hashing Loader**. Realiza un conjunto de hash al fichero (md5, sha1, sha256, sha512 y pe_hash) y los devuelve.

Los resultados de la ejecución de estas herramientas se almacenan en HDFS. Para mostrar los resultados se usa el motor de Elasticsearch (junto con Wonderdog para exportar los datos desde HDFS). Así el usuario puede realizar búsquedas y ver el resultado de análisis a nivel de aplicación. Sin embargo, se pueden ver los resultados usando otras herramientas (no es obligatorio la implementación de Elasticsearch).

4.2 Diseño

En esta parte se engloban todas las funcionalidades correspondientes a la parte de procesado por lotes. Se va a detallar el desarrollo de rb-sequence-oozie, algunos de los cargadores definidos para rb-Malware y el diseño del workflow de Oozie.

4.2.1 Funcionamiento de rb-sequence-oozie

Tal y como se puede observar en la Figura 4.2, rb-sequence-oozie es una aplicación distribuida que consta de dos partes. La primera parte, llamada rb-sequence, se encarga de obtener los ficheros de rb-S3 y escribirlos en Hadoop como un único fichero secuencial. La segunda parte, llamada rb-oozie, ejecuta los trabajos de análisis a través del servicio Oozie.

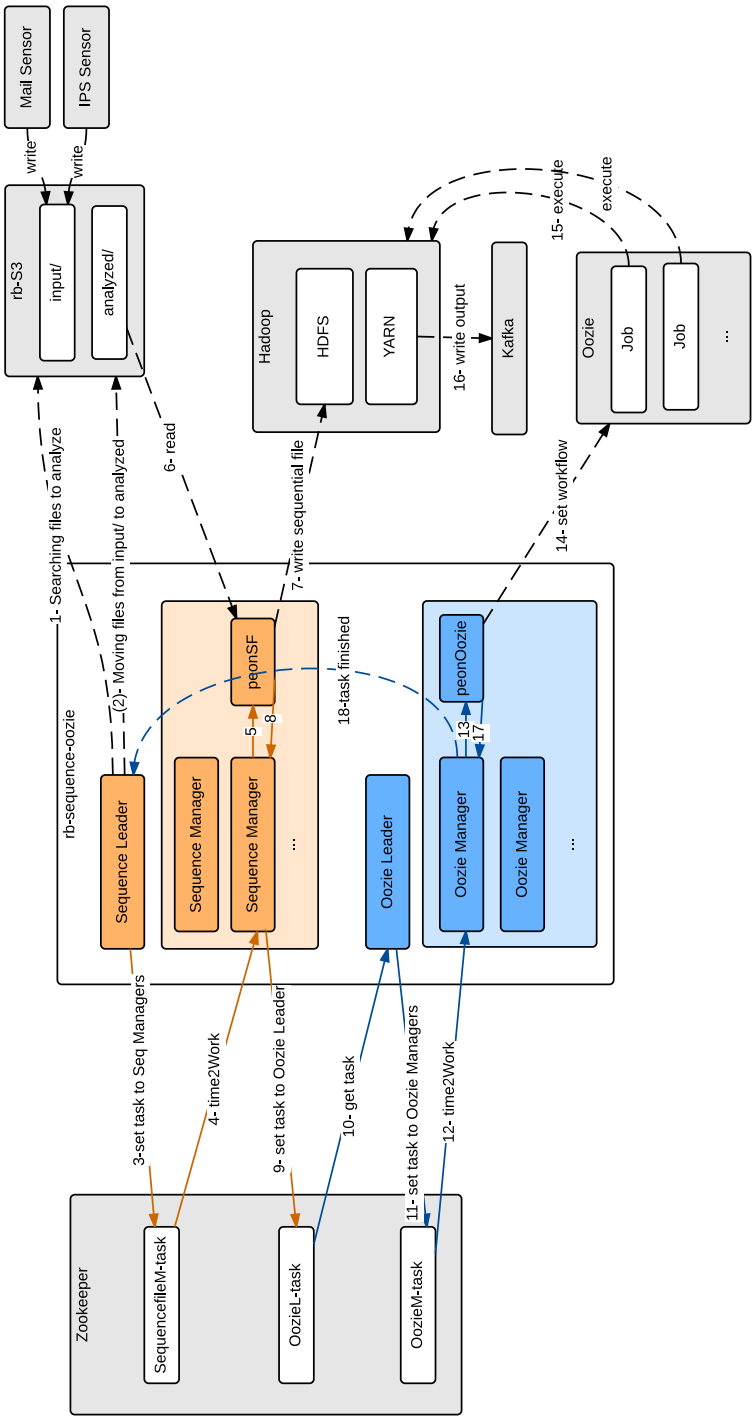

Figura 4.2 Diagrama de funcionamiento de rb-sequenze-oozie.

Ambas partes están compuestas de tres elementos diferenciados: líder, manager y peones. En un clúster de rb-sequence-oozie existe un líder de rb-sequence y un líder de rb-oozie. Se dispone de tantos Managers como instancias del servicio, que lanzarán peones para crear/analizar ficheros en HDFS.

En cuanto al funcionamiento del servicio, los ficheros llegan a la ruta de rb-S3: “s3:// malware/mdata/input”, provenientes del sensor IDS o del sensor Mail Gateway.

El Sequence Leader revisa continuamente las rutas de rb-S3: s3://malware/mdata/input/ y s3:// malware/mdata/analyzed/ buscando si hay ficheros para analizar (dándole prioridad al análisis de los ficheros de la carpeta "input/"). Si hay ficheros en la carpeta "input/" (y se seleccionan para ser analizados), el Sequence Leader pasa estos a la carpeta "analyzed/" [2].

El Sequence Leader seleccionará como máximo el número de ficheros (definido en la configuración). Una vez los ha seleccionado escribe una tarea en zookeeper indicando qué archivos han de ser analizados; esta tarea es asignada a un Sequece Manager que ejecutará un Sequence Peon.

Dicho Peon descarga los ficheros de rb-S3 que el Sequence Leader seleccionó, los fusiona creando un único fichero secuencial y los sube a HDFS de Hadoop. El Sequence Manager escribe entonces una tarea en Zookeeper, que será recibida por la segunda parte del programa, el Oozie Leader, para que se analice el nuevo fichero secuencial. Aquí acaba la primera parte del servicio rb-sequence-oozie.

Llegados a este punto comienza la segunda parte. El Oozie Leader pasa dicha tarea a un Oozie-Manager apoyándose de nuevo en ZooKeper.

El Oozie Manager crea un peonOozie, el cual implementa un cliente del servicio Oozie. Este cliente es el encargado de realizar el análisis (determinado por el workflow a ejecutar). El peón manda dicho trabajo a Oozie y una vez se recibe por parte del servidor de Oozie, va ejecutando las tareas en el YARN de Hadoop.

Cuando termina el análisis, se libera un semáforo del Sequence Leader para que éste vuelva a seleccionar nuevos ficheros para que sean analizados y se repite todo el proceso anteriormente mencionado.

4.3 Implementación de rb-oozie

En este proyecto, se ha desarrollado la parte encargada de ejecutar trabajos en Oozie. Esta se ha desarrollado usando el lenguaje de programación Java y consta de tres clases principales que se definen a continuación

4.3.1 OozieLeader

Hay un único OozieLeader ejecutándose [3] en el sistema. Este está continuamente esperando tareas a través de la ruta "/rb_malware/oozie/tasks" de Zookeeper.

Cuando se recibe una tarea, esta se escribe en /rb_malware/oozie/peontasks. Estas tareas son leidas por los OozieManagers a través de Zookeeper. El objetivo, es que de esta forma Zookeeper decide quién va a ejecutar la tarea y así se reparte la carga de trabajo.

[2] El motivo por el cual se ha decidido mantener este "histórico de ficheros" en rb-S3 es el de poder volver a analizar antiguos ficheros. Pongamos por ejemplo, que llega al sistema un fichero que está ofuscado y es capaz de modificar su comportamiento si se está ejecutando en una máquina virtual. Además, en el momento de ser capturado no se encuentra en ninguna base de datos de firmas de malware. Para este caso, el va a pasar sin ser detectado. Lo único que podemos hacer para remediar el daño, es dejar una copia en el sistema. De esta forma, este fichero se va a analizar cada cierto tiempo y es posible que cuando se vuelva a analizar, alguno de los cargadores de positivo ya que se ha introducido en alguna base de datos de firmas malware.

[3] Si el programa se está ejecutando en un cluster, únicamente habrá un OozieLeader despierto, el resto estarán dormidos y no se usarán.

Código 4.1 Bucle principal del programa OozieLeader.

```
if (oozieTasksHandler.isLeader()) {
  leader = Leader.LEADER;
  try {
    String seqFile;
    while ((seqFile = incomingSeqFiles.poll()) != null) {
      log.info("New File: " + seqFile + " -- Waiting Files: " +
          incomingSeqFiles.size());

      String taskName = UUID.randomUUID().toString();
      log.info("New Task set in '/rb_malware/oozie/peontasks/" +
          taskName + "' ");
      client.create().forPath("/rb_malware/oozie/peontasks/" + taskName,
           seqFile.getBytes());
      oozieTasksHandler.goToWork(true);
    }

    if (status.equals(Status.RUNNING)) {
      waitTasks();
    } else if (status.equals(Status.CLOSING)) {
      client.close();
      status = Status.CLOSE;
      log.info("Status [" + status.name() + "]");
    }
  } catch (Exception e) {
    e.printStackTrace();
  }

} else {
  try {
    leader = Leader.NOT_LEADER;
    log.info("Sleeping... I'm not a leader!");
    Thread.sleep(ONE_MINUTE);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}
```

4.3.2 OozieManager

Habrá un OozieManager por cada instancia de rb-sequence-oozie que se esté ejecutando. Cuando OozieLeader escribe una tarea, Zookeeper se la manda a uno de los OozieManager disponibles. Este OozieManager lee una tarea y ejecuta un OoziePeon para que se encarge de enviar a analizar el fichero secuencial especificado.

Cuando un OoziePeon ha terminado de ejecutarse, se libera el semáforo del SequenceLeader.

Código 4.2 Función ejecutada por uno de los OozieManager cuando Zookeeper lo manda a trabajar.

```
public void time2Work() {
```

```
  if(status.equals(Status.RUNNING)) {
    try {
      mutex.acquire();

      List<String> childrens = client.getChildren().forPath("/rb_malware
          /oozie/peontasks");
      if (!childrens.isEmpty()) {
        String children = childrens.get(0);
        log.info("Children is: " + children);
        byte[] zkData = client.getData().forPath("/rb_malware/oozie/
            peontasks/" + children);
        String file = new String(zkData, "UTF-8");
        log.info("The new file is: " + file);
        client.delete().forPath("/rb_malware/oozie/peontasks/" +
            children);

        OoziePeon peonWork = new OoziePeon(file);
        peonWork.start();
        peons.add(peonWork);

        synchronized (monitor) {
          monitor.notifyAll();
        }
      }
      mutex.release();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

4.3.3 OoziePeon

Se encarga de configurar las propiedades del workflow. Define algunas propiedades como las IPs de las máquinas donde se están ejecutando el Namenode, Jobtracker y Cuckoo, la cola de hadoop en la que se va a ejecutar el trabajo, topics de kafka, brokers de zookeeper, ruta de la librería de malware, api key de VirusTotal y metascan, ruta en la que se encuentran las reglas de Yara, ruta de los scripts.

Una vez tenemos definidas las propiedades del workflow a ejecutar, usamos un cliente oozie para que envíe el trabajo al servidor de Oozie.

Código 4.3 Ejemplo de configuración y envío de un trabajo al servidor de Oozie.

```
OozieClient client = new OozieClient(OOZIE_SERVER);
Properties conf = client.createConfiguration();

// Definimos las propiedades
conf.setProperty("name_node", NAME_NODE);
conf.setProperty("yara_rules", YARA_PATH);
[...]
```

```
// Mandamos el trabajo al servidor de Oozie
idjob = client.run(conf);

// Esperamos a que el workflow termine de ejecutarse
while (client.getJobInfo(idjob).getStatus().equals(WorkflowJob.Status.
    RUNNING))
{
  log.info("Job [" + idjob +"] - Status ["+ client.getJobInfo(idjob).
     getStatus().name() + "]");
  Thread.sleep(15*1000);
}
```

Después, el proceso espera hasta que finalice la ejecución comprobando cada 15 segundos el estado del workflow.

4.4 Funciones de detección

4.4.1 Clases de Pig

Para la programación de los scripts de pig, se han usado varias funciones UDF(User Defined Function) escritas en Java. Las funciones usadas por rb-sequence-oozie son: AbstractExecutingLoader, AbstractFileDropingLoader, ExecutingJsonLoader, ExecutingTextLoader, ClamScanDaemonLoader, CuckooLoader, KasperskyLoader, MetascanOnlineLoader, HashingLoader, VirusTotalLoader, YaraLoader y KafkaStorage.

Concretamente, las clases que se han desarrollado en el proyecto han sido KasperskyLoader (que hereda de ExecutingJsonLoader) y KafkaStorage.

KafkaStorage

Extiende de la clase abstracta StoreFunc, perteneciente a la librería de Apache Pig.

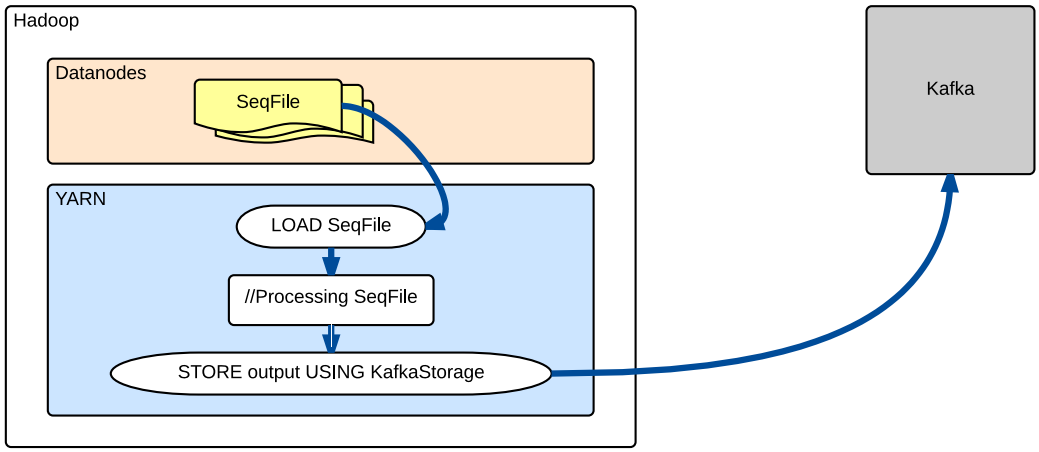

Figura 4.3 Ejecución de los script usando KafkaStorage.

Para crear una función de Almacenamiento para Pig, debemos implementar la interfaz de StoreFunc. Lo primero que debemos implementar es el constructor de la clase (en nuestro caso "KafkaStorage") con los parámetros de entrada. Para comunicarnos con Kafka necesitamos conocer el topic

en el que se va a escribir y el servidor al que se va a mandar la salida. Por tanto los parámetros de entrada de nuestra función serán "topic" y "kafkaServer".

También debe definir la función getOutputFormat() que informa sobre el formato de salida. En nuestro caso, será NullOutputFormat() ya que no se va a devolver la salida a Hadoop. Se define también el método setStoreLocation() pero se dejará vacía por el mismo motivo.

Los objetos usados para escribir se inicializan en prepareToWrite(). En este método definimos el productor de kafka con todas las propiedades (código 4.4).

Código 4.4 Método prepareToWrite() de KafkaStorage.

```
@Override
public void prepareToWrite(org.apache.hadoop.mapreduce.RecordWriter
    recordWriter) throws IOException {
   // We set the kafka URL and topic
   final Properties props = new Properties();
   props.put("metadata.broker.list", kafkaServer);
   props.put("serializer.class", "kafka.serializer.StringEncoder");
   props.put("request.required.acks", "1");
   props.put("message.send.max.retries", "60");
   props.put("retry.backoff.ms", "1000");
   props.put("producer.type", "sync");
   props.put("queue.buffering.max.messages", "500");
   props.put("queue.buffering.max.ms", "250");
   kafkaProducer = new Producer<String, String>(new ProducerConfig(props
       ));
}
```

Y por último, definimos putNext(), que va leyendo tupla a tupla y mandando el resultado a Kafka en formato Json. Este método hace uso de putField(), que es el que lee cada campo de la tupla y lo transforma a Json (código 4.5 y 4.6).

Código 4.5 Método putNext() de KafkaStorage.

```
@Override
public void putNext(Tuple tuple) throws IOException {

   if (tuple.size() > 0) {
       JSONObject jsonObj;
       jsonObj = new JSONObject();
       // We store a string from tuple
       for (int i = 0; i < tuple.size(); i++) {
           Object field;
           try {
               field = tuple.get(i);
           } catch (ExecException ee) {
               throw ee;
           }

           putField(field, i, jsonObj);
       }
```

```
        kafkaProducer.send(new KeyedMessage<String, String>(topic,
            jsonObj.toString()));
        jsonObj = new JSONObject();

    } else{
        log.warning("Tuple size is 0.");
    }
}
```

Código 4.6 Método putField() usado por putNext().

```
private void putField(Object field, int i, JSONObject jsonObj) throws
    IOException {

    try {
        switch (DataType.findType(field)) {
            case DataType.NULL:
                jsonObj.put(String.valueOf(i), "NULL");
                break;
            case DataType.BOOLEAN:
                jsonObj.put(String.valueOf(i), (boolean) field);
                break;
            case DataType.INTEGER:
                jsonObj.put(String.valueOf(i), (int) field);
                break;
            case DataType.LONG:
                jsonObj.put(String.valueOf(i), (long) field);
                break;
            case DataType.FLOAT:
                jsonObj.put(String.valueOf(i), (float) field);
                break;
            case DataType.DOUBLE:
                jsonObj.put(String.valueOf(i), (double) field);
                break;
            case DataType.BYTEARRAY:
                byte[] b = ((DataByteArray) field).get();
                jsonObj.put(String.valueOf(i), b);
                break;
            case DataType.CHARARRAY:
                jsonObj.put(String.valueOf(i), (String) field);
                break;
            case DataType.BYTE:
                jsonObj.put(String.valueOf(i), (byte) field);
                break;

            case DataType.MAP:
                boolean mapHasNext = false;
```

```
                Map<String, Object> m = (Map<String, Object>) field;
                for (Map.Entry<String, Object> e : m.entrySet()) {
                    jsonObj.put(e.getKey(), e.getValue());
                }
                break;

            case DataType.TUPLE:
                Tuple t = (Tuple) field;
                for (int n = 0; n < t.size(); ++n) {
                    try {
                        putField(t.get(n), i, jsonObj);
                    } catch (ExecException ee) {
                        throw ee;
                    }
                }
                break;

            case DataType.BAG:
                Iterator<Tuple> tupleIter = ((DataBag) field).iterator();
                while (tupleIter.hasNext()) {
                    putField(tupleIter.next(), i, jsonObj);
                }
                break;

            default:
                throw new RuntimeException("Unknown datatype " + DataType.
                    findType(field));
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

El código completo de esta función UDF se encuentra en el anexo *KafkaStorage*.

KasperskyLoader

Este Loader se ha hecho de forma similar a la función de carga ClamScanLoader definida en BinaryPig. Se ha usado la clase ExecutingJsonLoader que se encarga de ejecutar un fichero y devolver la salida en formato Json.

Para que se ejecute, escribimos en un script los comandos para que kaspersky analice el fichero.

Código 4.7 Script para la ejecución de kaspersky.

```
if (/opt/kaspersky/kav4fs/bin/kav4fs-control --scan-file $@ | grep "
    Threats found" | grep -q "0"); then echo OK; else echo MALWARE; fi
```

De esta forma se devolverá OK para ficheros limpios y MALWARE para ficheros con malware. La explicación completa y el código de esta función UDF está en el anexo *KasperskyLoader*.

4.4.2 Workflow

El workflow usado para la versión 0.4 de rB Malware es el que se muestra en la figura 4.4.

Primero se ejecuta un análisis estático con ClamAV ya que es un servicio que se ejecuta en local. Después, se mandan peticiones a los servicios en la nube, Metascan y Virustotal. Primero se envía a Metascan ya que el límite de peticiones por hora es mayor. Después, se analiza con Yara ya que las reglas para la versión 0.4 de rB Malware originan un gran número de falsos positivos.

Por último, se realiza un análisis dinámico enviando el fichero a Cuckoo. Cuckoo dispone de firmas de detección de comportamiento anómalo y de indicadores de compromiso que proporcionan información sobre eventos sospechosos generados por la muestra analizada en la máquina bajo estudio. Las alertas generadas nos permitirán tomar una decisión sobre si el fichero bajo análisis es o no un malware.

Una vez que el análisis del fichero ha finalizado, Cuckoo genera como salida una serie de resultados que son escritos en Hadoop. Para poder escribir los resultados en Hadoop, desde redBorder se ha creado un módulo de "reporting" específico para esta tarea [4].

La definición del workflow usada para las pruebas de la versión 0.4 está en el anexo *Workflow*.

[4] El procesado de los resultados generados por Cuckoo no está implementado en la release 0.4 de redBorder Malware. De hecho, en esta release a través de la interfaz web de redBorder Malware no es posible gestionar las máquinas virtuales que son cargadas en Cuckoo. Por tanto, este proceso debe realizarse a bajo nivel, accediendo a través de ssh al servidor.

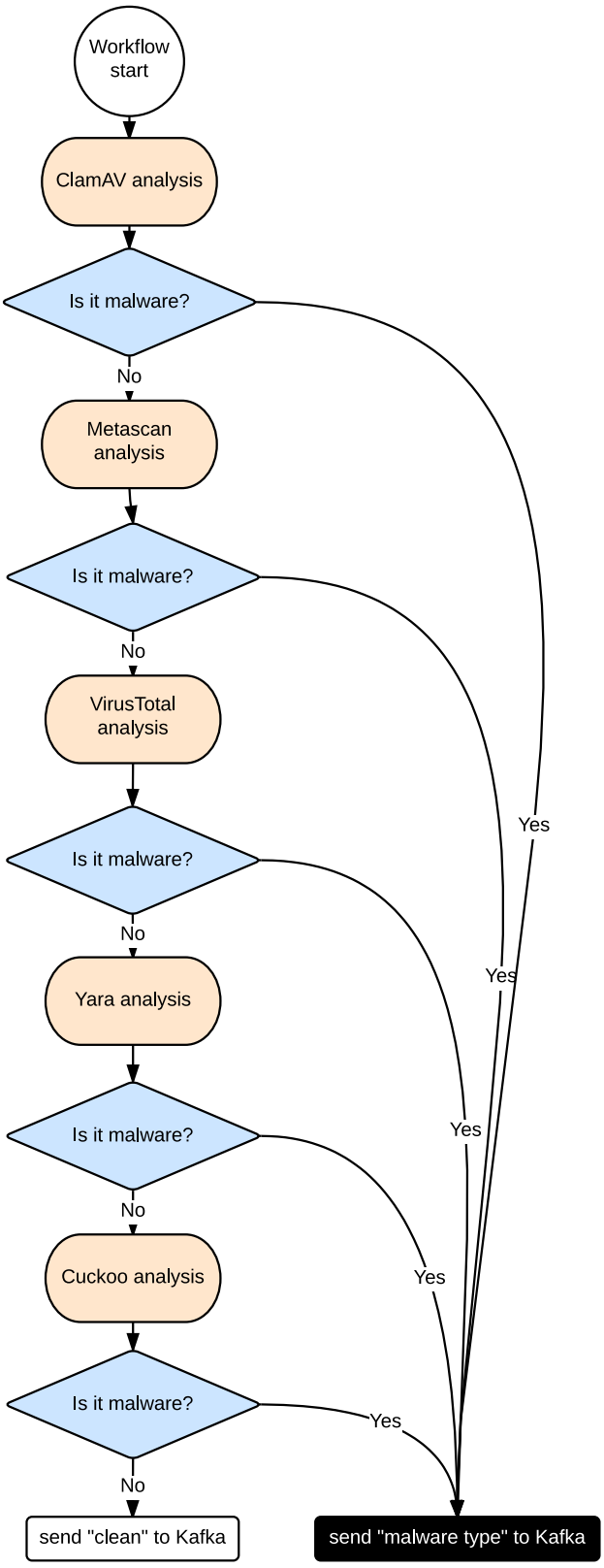

Figura 4.4 Workflow usado en la versión 0.4 de rB Malware.

5 Pruebas

We worried for decades about weapons of mass destruction. Now it is time to worry about a new kind of WMD, weapons of mass disruption.

John Mariotti (Periodista tecnológico y CEO de 'The Enterprise Group')

En el presente apartado se pretende definir las pruebas realizadas para la release 0.4. Primero presentaremos el escenario y los casos de uso contemplados. Posteriormente mostrarán los resultados y las conclusiones de las pruebas.

5.1 Escenario

En el siguiente apartado se describe el entorno de laboratorio utilizado para las pruebas. Se trata de un escenario sencillo. No presenta mecanismos de alta disponibilidad, balanceo, routing, etc., pero permite validar una serie de casos de uso de aplicación real.

Tal y como se muestra en la siguiente imagen, el escenario de esta primera release está formado por una red conmutada con 5 elementos importantes:

- rb-Manager. Elemento central en el cual se recibe, se trata y se visualiza toda la información emitida por los diferentes sensores. Desde este punto centralizado se pueden configurar ciertos aspectos de los sensores.
- rb-IDS. Sensor por el cual pasa el tráfico de red a analizar y, funcionando como un IDS, es capaz de detectar patrones de tráfico correspondientes a posibles ataques y/o malware. Una vez registrado con un Manager, le envía los resultados, así como los ficheros detectados. De igual forma, las reglas utilizadas en el análisis también son gestionadas desde el rb-Manager.
- rb-Mail Gateway. Sensor por el cual pasa el tráfico de correo (SMTP). Funciona como un sencillo MTA (Mail Transfer Agent), ya que cada vez que recibe un e-mail lo reenvía al servidor de correo correspondiente. Este realiza una copia de los e-mails con adjuntos, y lo reenvía también al rb-Manager, junto con datos adicionales del e-mail (p. ej. Dirección remitente/destinatario...). Al igual que el IDS, también se asocia/registra con un rb-Manager.
- LAB Mail Server. Se trata de un servidor de correo en el cual simplemente se recepcionan todos los correos generados para las pruebas y reenviados por el rb-Mail Gateway.
- Testing Group. Formado por un grupo de equipos y usuarios. Desde aquí es desde donde se envían los e-mails con malware y/o desde donde se acceda a páginas web que permitan la descarga de ficheros con código malicioso.

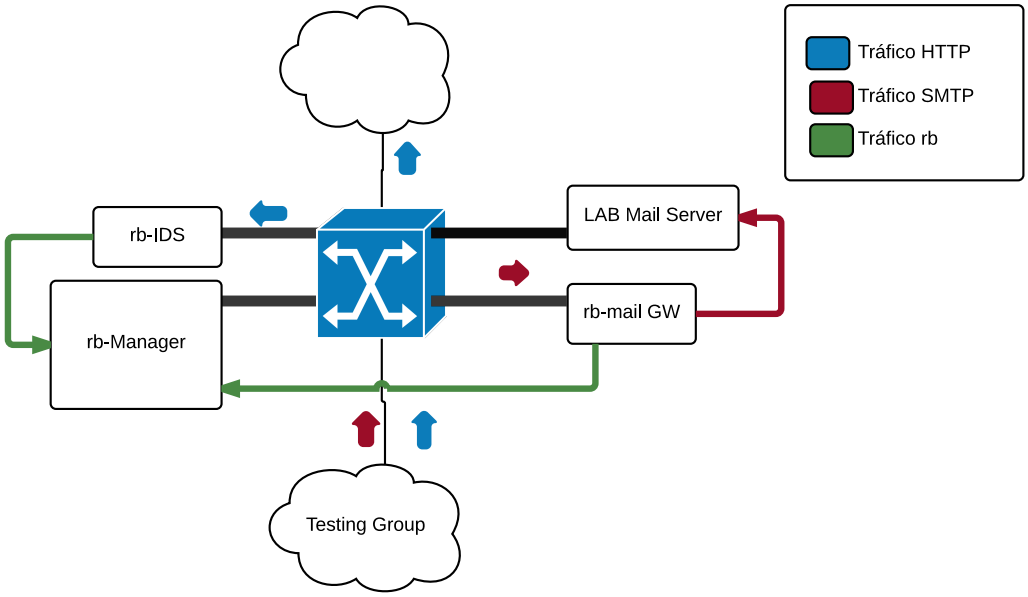

Figura 5.1 Escenario de laboratorio.

5.2 Casos de uso

En esta sección se muestran varios de los casos de uso sobre los que se ha trabajado de cara a garantizar que, en un entorno controlado y de laboratorio, se puedan cumplir y tener así un primer punto de partida para la release 0.4.

A continuación se van a mostrar los resultados de las pruebas. Sin embargo, todas las capturas de pantalla realizadas durante la realización de las pruebas se muestran en el anexo *Casos de uso*, detallando más detenidamente la ejecución de las pruebas.

En términos generales, los casos de uso tratados muestran elementos propios de configuración, así como de operativa del Mail Gateway y del IDS, procesado con las tareas definidas en el workflow y la interacción con las vistas web.

5.2.1 Caso de uso 1 - Detección de e-mails con ficheros adjuntos con Malware

En este caso, se ha intentado mostrar la capacidad de detección de ficheros adjuntos con malware en diversos e-mails enviados. De esta forma, se ha puesto a prueba una de las puertas de entrada para el código malicioso, como es el correo electrónico.

Para ello, se han enviado múltiples e-mails hacia un destinatario de correo electrónico alojado en el entorno de laboratorio. La Figura 5.2 ejemplifica el escenario.

En las tablas 5.1, 5.2 y 5.3 se encuentra los resultados de todas las pruebas ejecutadas para este caso de uso.

5.2.2 Caso de uso 2 - Detección de Malware en tráfico web

Para este caso de uso, nos descargamos un fichero desde un terminal Linux. Este fichero será capturado por el IPS de Redborder y analizado por el sistema rb-sequence-oozie.

En las tablas 5.4 y 5.5 se encuentra los resultados de todas las pruebas ejecutadas para el caso de uso 2.

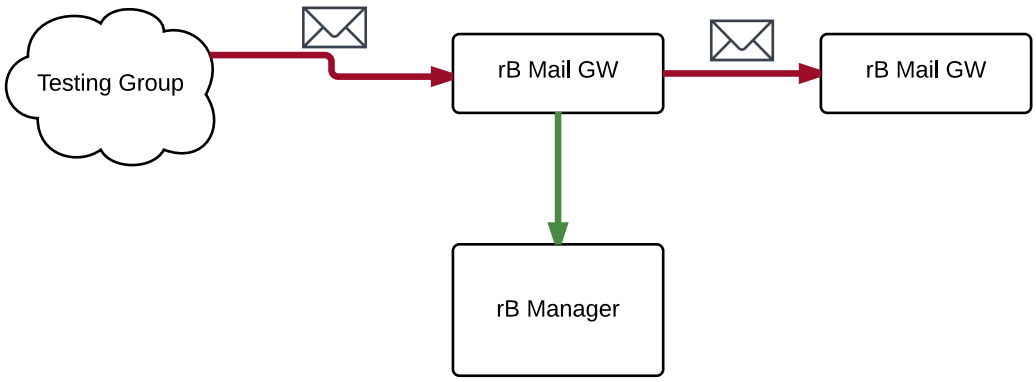

Figura 5.2 Flujo de entrada de correo electrónico.

5.3 Conclusiones

En los treinta y ocho ficheros analizados en las pruebas, veintitrés de estos eran ficheros que contenían malware. De estos veintitrés archivos, tan sólo uno de ellos no ha sido detectado por el sistema. Esto nos da un total de un 4.34 % de ficheros no detectados.

En cuanto a los falsos positivos, de los quince ficheros sin malware analizados, todos han dado falso positivo con alguna regla de Yara. Este problema se podría solucionar de dos formas: revisando las reglas o dando un porcentaje de fiabilidad a algunas de las reglas ya definidas. La primera opción, consiste en modificar las reglas de Yara. Actualmente, se están usando las mismas reglas que las definidas en el proyecto BinaryPig.

La otra forma de mejora, consiste en revisar las reglas actuales y dar un porcentaje que nos indique lo "sospechoso" que es un fichero. De forma que a partir de cierto umbral se considere que el fichero es malicioso.

Otro detalle importante, es que la mayoría de los ficheros maliciosos comprimidos analizados por el sistema han sido detectados por Yara. En total ha sido un 62.5 % los ficheros comprimidos detectados. Estos nos da la idea de que Yara es una buena herramienta para detectar ficheros comprimidos. También se podría pensar usar reglas basadas en la herramienta PEiD (herramienta para windows que detecta comprimidos). PEiD es una aplicación bastante usada en la detección de malware y hay herramientas que permiten portar las reglas definidas en PEiD a Yara.

Tabla 5.1 Caso de uso 1 - parte 1.

Nºiter.	Tamaño del fichero	Tipo de fichero	SHA256	¿Fichero con malware?	¿Detectado correctamente?	ClamAV	Metascan	VirusTotal	Yara	Cuckoo	Observaciones
1	2.7M	.exe	1154535130d546eaa3 3bbc9051a9cb91e2 b0e3a3991286c3d5b 0a708110c9aa7	si	Detectado correctamente	Win.Trojan. Scar-40	-	-	-	-	-
2.1	43K	.dll	ae79d6e52e9eb8ad 4bd0f9a0fe230f1cd be53f077ecda0d159 49945532541d80	Si	Detectado correctamente	Win.Trojan. PlugX-110	-	-	-	-	-
2.2	50K	.tar.gz	3faa16ff914821e4d31 f96755a1b6d04406 fa6f012563d6c125 b5793e289efac	Si	Detectado correctamente	Win.Trojan. PlugX-110	-	-	-	-	Mismo fichero que el anterior pero comprimido
2.3	21K	.zip	e3b956b1561eceba0 6494117333da2799 faedc943988b8e5b0 6a74b917ef5494	Si	Detectado correctamente	Win.Trojan. PlugX-110	-	-	-	-	Mismo fichero que el anterior pero comprimido
3	341K	.zip	2aa64f27057a9ac092e 999b2d7f6df6639b 7e3ba9bd897c378e 65c23d178de70	Si	Detectado correctamente	-	-	-	Cumple 4 reglas	-	Fichero comprimido con contraseña que contiene varios ficheros con malware
4	102K	.zip	055b91fdb33432d29 14191cbeff3c58fa 9a2a69653608f44d8 97ddc1de253890	Si	Detectado correctamente	-	-	-	Cumple 3 reglas	-	Fichero comprimido sin contraseña
5	577K	.apk	2b29d93e58ab328a0 c9418001d7748dc4c50 b514c8074f4c0055 6b890df31644	Si	Detectado correctamente	-	Agent	-	-	-	-
6.1	72K	.dll	17b7ad3434a9ce3ce3 1978a6822be271a 7fe0b45bff47adc6b1e 2d04238ec4be	Si	Detectado correctamente	-	Trojan.Sirefef !TFFRoqWd4Ww	-	-	-	-

Tabla 5.2 Caso de uso 1 - parte 2.

Nºiter.	Tamaño del fichero	Tipo de fichero	SHA256	¿Fichero con malware?	¿Detectado correctamente?	ClamAV	Metascan	VirusTotal	Yara	Cuckoo	Observaciones
6.2	80K	.tar.gz	3b37f9a9c51ed59b ab384d4d3c1979b7 99bb710da1bd31df63 cd589068b11dd6	Si	Detectado correctamente	-	-	-	Cumple 10 reglas	-	Mismo fichero que el anterior pero comprimido
6.3	24K	.zip	8211b1dcf1afd663a0a5 c033e2b860f6afa ccd54d1533efd3e65 244bce68c31c	Si	Detectado correctamente	-	-	-	Cumple 3 reglas	-	Mismo fichero que el anterior pero comprimido
7	247K	.exe	69e966e730557fde8fd 84317cdef1ece0 0a8bb3470c0b58f323 1e170168af169	Si	Detectado correctamente	-	Backdoor.Zaccess !UkzQ0/sevQU	-	-	-	-
8	41M	.dll	ef32516eb5658c6529 9cb1c9a0f7ec5522 16f4b9d2975d074ff3 1eacb3003a19	Si	No detectado	-	-	-	-	-	-
9	59M	.mp4	5b9bcfc27b6ae8fa c9147f1a4a7dd3c4 04392ad8e076d71e0 07eb1225fa3e409	No	Falso positivo	-	-	-	Cumple 16 reglas	-	-
10	9.3M	.mp3	6bc5111b201a47c1 00d22c4add46fbf ca99702c0fb86e478 dc20bea98eef00a1	No	Falso positivo	-	-	-	Cumple 10 reglas	-	-
11	199K	.pdf	607dcdc8d91ea034 9c2d2c8123f21e58 38405688c15f8e1cb 1094223e997824d	No	Falso positivo	-	-	-	Cumple 4 reglas	-	-

Tabla 5.3 Caso de uso 1 - parte 3.

Nºiter.	Tamaño del fichero	Tipo de fichero	SHA256	¿Fichero con malware?	¿Detectado correctamente?	ClamAV	Metascan	VirusTotal	Yara	Cuckoo	Observaciones
12	1.9K	.mp3	cb6cff6eebdd9ac9f1 fe8e0f27c8ae7f2 e8b7e7c5ad5ba4b55 0e3027db5585bb	No	Falso positivo	-	-	-	Cumple 6 reglas	-	-
13	52M	.exe	a8a39a08542ed858a 5fc30a436e28d4 4638cabf3dab43651 0481d6afb045204d	No	Falso positivo	-	-	-	Cumple 13 reglas	-	-
14	2.2K	.sh	684b8508b5a7828b 659b2e62ed99ee92 b01adcf0f01846081 4df280a34c07cb1	No	Falso positivo	-	-	-	Cumple 1 regla	-	-
15	1.4M	.jpg	c91c94a01b510dba5 986cfbd27ba43e7 8553d7ad3b536a4d7 965f1a25d56103f	No	Falso positivo	-	-	-	Cumple 4 reglas	-	-
16	208K	.exe	7bf1692feae03bfc0 321b4ae0aa2947ee 1c1441be571a97c27 b763d9de89dc73	No	Falso positivo	-	-	-	Cumple 14 reglas	-	Ejecutable que abre un servidor HTTP
17	2.9M	.exe	b965933dba084d0f4 f866eb8808dfc19 9f6bf7d4ba3f58909 342d93305517b19	No	Falso positivo	-	-	-	Cumple 19 reglas	-	Ejecutable que abre un servidor TFTP
18	2.8M	.msi	200a64f329f859d7 7c5f60cd197df2b 40f2604a374efefef 53114a00ac630205	No	Falso positivo	-	-	-	Cumple 11 reglas	-	Ejecutable que abre un servidor SMTP

Tabla 5.4 Caso de uso 2 - parte 1.

Nºiter.	Tamaño del fichero	Tipo de fichero	SHA256	¿Fichero con malware?	¿Detectado correctamente?	ClamAV	Metascan	VirusTotal	Yara	Cuckoo	Observaciones
1.1	20K	.exe	efc94fdac875345 1e7070f0cccb1b8e2ba2 ce9e6edd3378a7ac 412a359a256e4	si	Detectado correctamente	Trojan. Rustock-40	-	-	-	-	Nombre del fichero: Malware.exe
1.2	20K	.exe	efc94fdac875345 1e7070f0cccb1b8e2ba2 ce9e6edd3378a7ac 412a359a256e4	si	Detectado correctamente	Trojan. Rustock-40	-	-	-	-	Nombre del fichero: Malware.mp3
2	227K	.zip	32f43843c74e8 fa16b0e88fa63921 f3b81751b05ffe8d80 7711318bda333321d	si	Detectado correctamente	-	-	Trojan .Win32.SdBot .coongn	-	-	Fichero comprimido con contraseña
3	77K	.dll	137e17ed0c693f5ba 23c3f3bf252f7e dc29548d97f426625 a4e0c5fea0558e45	si	Detectado correctamente	Osx.Trojan .Netweird	-	-	-	-	-
4	31K	.exe	59979d3bc3d64500 898f3c1fda833cc0f 87db36b65f1bb4631 e2ac1b232c8aad	si	Detectado correctamente	-	-	W32.MiadheardLTL. Trojan	-	-	-
5	80K	.tar.gz	f9a9c51ed59bab3 84d4d3c1979b79 9bb710da1bd31df6 3cd589068b11dd6	si	Detectado correctamente	-	-	-	Cumple 10 reglas	-	-
6	241K	.bin	45317968759d3e3 7282ceb75149f627 d648534c5b4685f6d a3966d8f6fca662d	si	Detectado correctamente	-	Trojan. Cryptodef!	-	-	-	-
7	2.7M	.exe	1154535130d546eaa 33bbc9051a9cb91e 2b0e3a3991286c3d 5b0a708110c9aa7	si	Detectado correctamente	Win.Trojan. Scar-40	-	-	-	-	-

Tabla 5.5 Caso de uso 2 - parte 2.

Nºiter.	Tamaño del fichero	Tipo de fichero	SHA256	¿Fichero con malware?	¿Detectado correctamente?	ClamAV	Metascan	VirusTotal	Yara	Cuckoo	Observaciones
8.1	55K	.exe	8abb47ca7c0c4871c 28b89aa0e75493e5 eb01e403272888c1 1fef9e53d633ffe	si	Detectado correctamente	Trojan.Agent! sRcCsxKhOnY	-	-	-	-	-
8.2	53K	.zip	55f64106ffbe61f4 a361d4622a2843b8 bcb2e33f9d09d514 6e6e416f196e54c8	si	Detectado correctamente	-	-	-	Cumple 4 reglas	-	Mismo fichero que el anterior pero comprimido
9	5.2M	.zip	69d8a15aa67c9cdfc 0bec6206405a5f4 4f969988ebe6115f2 8a13d91e8a2b5f5	si	Detectado correctamente	-	-	Trojan.Win32. Explosive.dpzrss	-	-	-
10	454K	.jar	8de583ee28079a25f1 b67fe356fcee11d 666af1385172b218 77ae3ead8c731a9	No	Falso positivo	-	-	-	Cumple 6 reglas	-	-
11	1.9M	.mp3	cb6cff6eebdd9ac9f 1fe8e0f27c8ae7f2 e8b7e7c5ad5ba4b5 50e3027db5585bb	No	Falso positivo	-	-	-	Cumple 6 reglas	-	-
12	199K	.pdf	607dcdc8d91ea0349c 2d2c8123f21e583 8405688c15f8e1cb1 094223e997824d	No	Falso positivo	-	-	-	Cumple 8 reglas	-	-
13	2.9M	.exe	b965933dba084d0f 4f866eb8808dfc19 9f6bf7d4ba3f58909 342d93305517b19	No	Falso positivo	-	-	-	Cumple 19 reglas	-	Servicio que crea un servidor tftp
15	668M	.png	26aa214d17c37d19 2887c8e35c60e8e3 3a0f6aff852faea89 c3942d704858d3e	No	Falso positivo	-	-	-	Cumple 5 reglas	-	-

6 Presupuesto

If you spend more on coffee than on IT security, you will be hacked. What's more, you deserve to be hacked

Richard A. Clarke, 2002

En este apartado se calcula el precio estimado correspondiente a la realización la parte de rb-Malware descrita en esta memoria.

Tabla 6.1 Presupuesto para el desarrollo.

Periodo de trabajo	Número de horas	Precio/Hora	Total
Junio 2014-Febrero2015	40 horas	40 €/hora	1600€
Periodo de prácticas de empresa	220 horas	40 €/hora	8800€
Mayo 2015-Junio2015	80 horas	40 €/hora	3200€
Total	340€	40€	13600€

Tabla 6.2 Presupuesto para el equipamiento.

Equipamiento	Cantidad	Precio	Total
Anfitrión para VMware ESXi (Entorno de virtualización de servidores).Portátil para el desarrollo de código	1	5000€	5000€
Portátil para el desarrollo del software	1	300€	300€
Total			5300€

Esto nos da un total de 18900€ de presupuesto para llevar a cabo el desarrollo descrito en la memoria.

7 Conclusiones

If a machine is expected to be infallible, it cannot also be intelligent

Alan Turing, 1947

7.1 Conclusión

En este proyecto se ha diseñado un sistema que busca la detección de malware que entra en una red local.

Para ello se ha realizado una pequeña introducción al mundo del análisis de malware. Primero, realizando un un estudio teórico sobre el análisis estático y análisis dinámico. Y posteriormente se han recopilado varias de las herramientas más usadas en este campo como lo son Yara o ClamAV.

También se ha estudiado el análisis de grandes cantidades de datos. En nuestro caso, este estudio se ha basado en la herramienta Apache Hadoop. Con ella hemos podido ejecutar el análisis de varios ficheros en casi tiempo real [1].

7.2 Líneas futuras

En este proyecto se puede avanzar en dos frentes. El primero añadir herramientas que mejoren la detección de malware y análisis de ficheros. Alguna de estas (ya comentadas en el capítulo de Introducción al análisis de malware) pueden ser Fuzzy Hash para detectar ficheros ofuscados, MISP para mejorar la detección de nuevos ficheros malware y el uso de whitelist para descartar ficheros limpios.

Otra línea en la que se puede avanzar en este proyecto es la de la escalabilidad. Este proyecto se ha realizado con Apache Hadoop porque se ha partido del proyecto de BinaryPig. Sin embargo, está cobrando más fama en el mundo del big data la aplicación Apache Spark debido a su alta velocidad de análisis de ficheros. Por tanto podría ser un avance el mejorar el análisis migrando el sistema desarrollado a Apache Spark.

[1] El análisis de un fichero secuencial tarda del orden de quince minutos

8 Anexos

Fair Scheduler

Fair Scheduler organiza los trabajos en grupos y divide de manera justa los recursos entre estos grupos. Por defecto, hay un grupo por cada usuario, así los usuarios consiguen una compartición justa del cluster.

También permite asegurar un mínimo de recursos para cada grupo, así asegura que las aplicaciones siempre consiguen un mínimo de recursos. Si un grupo no usa los recursos mínimos asignados durante un periodo de tiempo, el Fair Scheduler prioriza las tareas de otros grupos. Si en algún momento se necesitan estos recursos, el scheduler matará alguno de los procesos ejecutados recientemente para así malgastar el menor tiempo de computaicón posible. Esto no origina que los trabajos fallen, si no que estas tareas se vuelven a ejecutar cuando haya recursos disponibles.

Para hacer que hadoop use el FairScheduler, se debe poner la propiedad que aparece en el Código 7.1 en el fichero yarn-site.xml.

Código 8.1 Activación del Fair Scheduler.

```
  <property>
    <name>yarn.resourcemanager.scheduler.class</name>
    <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.
        FairScheduler</value>
  </property>
```

La configuración usada (definida en el fichero fair-scheduler.xml) para el cluster de Hadoop que se ha montado, es:

Código 8.2 fair-scheduler.xml.

```
<?xml version="1.0"?>
<allocations>
  <queue name="others">
      <minResources>200 mb,1 vcores</minResources>
      <maxResources>6000 mb,10 vcores</maxResources>
      <maxRunningApps>15</maxRunningApps>
      <weight>1.0</weight>
      <minSharePreemptionTimeout>2</minSharePreemptionTimeout>
      <schedulingPolicy>fair</schedulingPolicy>
  </queue>

  <queue name="default">
    <minResources>1024 mb,5 vcores</minResources>
```

```
    <maxResources>10000 mb,16 vcores</maxResources>
    <weight>2.0</weight>
    <schedulingMode>fair</schedulingMode>
    <minSharePreemptionTimeout>2</minSharePreemptionTimeout>
  </queue>

  <user name="oozie">
      <maxRunningApps>50</maxRunningApps>
  </user>

  <userMaxJobsDefault>5</userMaxJobsDefault>

</allocations>
```

En oozie, se establece el grupo al que pertenece un trabajo añadiendo la propiedad mapred.job.queue.name en la definición del workflow.

Código 8.3 Configurar la cola de prioridad en el Workflow de Oozie.

```
    <property>
      <name>mapred.job.queue.name</name>
      <value>${queue_name}</value>
    </property>
```

Mejora de la eficiencia en Hadoop

La única forma verdadera de optimizar un cluster de hadoop es definiendo contadores y cambiando la configuración[5]. Así podemos comparar los resultados en cada ejecución de un mismo trabajo MapReduce. Este proceso se debe repetir hasta que se minimice el tiempo lo máximo posible.

Se aconseja partir de un punto de referencia inicial (típicamente se usan los parámetros por defecto en hadoop). Para ello, se recomienda usar una tabla e ir realizando pruebas.

Tabla 8.1 Ejemplo de fichero a rellenar para realizar las pruebas.

	Punto de partida	Prueba 1	...	Prueba M
Propiedad 1	valor	valor	...	valor
...	...	...	...	...
Propiedad N	valor	valor	...	valor

TeraSort

Para este trabajo, se ha usado la herramienta TeraSort que incluye Hadoop. Lo primero es crear un ejemplo de datos de entrada.

Código 8.4 Crear datos de entrada para Hadoop usando Terasort.

```
hadoop dfs -rmr [hdfs-path]
hadoop jar $HADOOP_PREFIX/hadoop-*examples*.jar teragen [tamaño]
    [hdfs-path]
```

Y después ejecutamos TeraSort.

Código 8.5 Análisis de datos en Hadoop usando Terasort.

```
hadoop jar $HADOOP_PREFIX/hadoop-*examples*.jar terasort [hdfs-
   path] /data/output
```

Factores que afectan a la ejecución de tareas MapReduce

El tiempo de procesamiento de los datos de entrada por MapReduce puede verse afectado por varios parámetros:

- Algoritmo o aplicación usado.
- Hardware (o recursos) como el reloj de la CPU, tareas de entrada/salida, ancho de banda de la red, y tamaño de la memoria.

- El subyacente sistema de almacenamiento.
- El tamaño de los datos de entrada y de salida están muy relacionados con el tiempo de ejecución del trabajo.
- Modo de entrada/salida de datos. Puede ser directo (leer directamente del disco local) o streaming (leer de otro proceso que está ejecutándose).
- Parseado de los datos de entrada: es la conversión de los datos en pares clave/valor. Los datos de entrada pueden ser mapeados a objetos Java (u otros). A los objetos que pueden ser modificados se les llama mutable objects y a los que no pueden ser alterados, se les llama immutable objects. Procesar objetos inmutables es significativamente más lento ya que pueden producir un gran número de immutable objects.
- Configuración de Hadoop: Hadoop tiene muchos componentes, algunos conectados en serie y otros en paralelo. Una mala configuración puede impactar en la coordinación entre las tareas

Los parámetros de Hadoop permiten modificar cuatro principales componentes: Uso de CPU, espacio en memoria, entradas/salidas, y tráfico en la red.

Los ficheros de configuración de hadoop se dividen en dos categorías: ficheros sólo de lectura de la configuración por defecto (core-default.xml, hdfs-default.xml, y mapred-default.xml) y ficheros para la configuración de un elemento específico (core-site.xml, hdfs-site.xml, y mapred-site.xml).

mapred-site.xml

Define muchas características, entre ellas el uso de CPU.

- mapred.tasktracker.map.tasks.maximum (por defecto: 2). Número máximo de tareas map que el TaskTraker correrá de manera simultánea.
- mapred.tasktracker.reduce.tasks.maximum (por defecto: 2). Número máximo de tareas reduce que el TaskTracker correrá de manera simultánea. Se suele poner que la suma de ambos valores sea: (nº de nodos en el cluster) * (Cores de la CPU) - 2

También permite definir operaciones de entrada y salida.

- mapred.compress.map.output (por defecto: false). Si está a true, la salida de las tareas map será comprimida usando un códec de compresión en ficheros de secuencia antes de enviarlo a la red.
- mapred.output.compress (por defecto: false). Habilita la comprensión de la salida del trabajo.
- mapred.map.output.compression.codec (por defecto: org.apache.hadoop.io.compress.DefaultCodec). Define el códec de compresión que se usará para la salida de map.

Comprimir la salida mejora el tiempo de escritura en el disco. También hay parámetros para configurar los directorios de salida de hadoop, estos son mapred.local.dir y dfs.data.dir

También se pueden definir parámetros relacionados con la memoria.

- mapred.child.java.opts (por defecto: -Xmx200m). Controla la memoria disponible en cada tarea JVM. El valor por defecto reserva 200MB de memoria para trabajos MapReduce.
- mapred.child.ulimit. Controla el límite de memoria virtual que será dado a un trabajo mapreduce. No tiene valor por defecto, pero se recomienda que sea al menos el doble de mapred.child.java.opts.

Los siguientes valores, si se aumentan, disminuye el tiempo de ejecución de las aplicaciones ya que se reduce el tiempo de I/O, pero aumenta la cantidad de memoria requerida

- io.sort.mb (por defecto: 100). Determina la cantidad de la caché en Mebabytes que será usada cuando se manejan flujos de datos.

- io.sort.factor (por defecto: 10). Determina el número de particiones de salida de map a unir .
- mapred.job.reduce.input.buffer.percent (por defecto: 0.0). Porcentaje de memoria relativa al tamaño máximo e pila para guardar la salida de map durante la fase de Reduce.
- mapred.reduce.parallel.copies (por defecto: 5). Número de transferencias paralelas para almacenar la salida de map durante la fase de shuffle. Se aconseja aumentar este valor sólo si la salida de las tareas ocupa mucho espacio.

yarn-site.xml

Para definir la memoria usando yarn, hay que tener en cuenta que un container es la unidad básica de la capacidad de procesamiento en YARN, como un encapsulamiento de los recursos (CPU,, memoria,...).

Los valores aconsejados para los siguientesparámetros son:

- yarn.nodemanager.resource.memory-mb = containers * RAM-per-container
- yarn.scheduler.minimum-allocation-mb = RAM-per-container
- yarn.scheduler.maximum-allocation-mb = containers * RAM-per-container

hdfs-site.xml

De los parámetros más importantes es el tamaño de partición de los ficheros (dfs.block.size, que por defecto vale 64MB). Si se quieren poner grandes ficheros de entrada, se pueden aumentar este número a 128MB o 256MB

- dfs.access.time.precision (por defecto es una hora). Precisión de los tiempos de acceso en milisegundos.
- dfs.balance.bandwidthPerSec (por defecto 1048576b/s). Máximo ancho de banda para cada datanode.
- dfs.block.size (por defecto 67108864). Tamaño por defecto de los bloques en los que se van a dividir los ficheros de entrada.
- dfs.data.dir (por defecto: $hadoop.tmp.dir/dfs/data). Determina la localización del disco donde el datanode debe guardar los bloques.
- dfs.datanode.du.reserved (por defecto: 0.0). Cantidad de espacio que se debe dejar libre en cada localización.
- dfs.datanode.handler.count (por defecto: 3). Número de servidores que manejan las peticiones de manejo de bloques.
- fs.max.objects (por defecto: 0). Indica el máximo número de objetos (ficheros, directorios y bloques) que pueden ser almacenados (0 indica que no hay límite).
- dfs.name.dir (por defecto: $hadoop.tmp.dir/dfs/name). Directorios usados para guardar bloques.
- dfs.name.edits.dir (por defecto: $dfs.name.dir). Indica dónde se va a guardar el fichero de transacciones del namenode.
- dfs.namenode.handler.count (por defecto 10). Especifica el número de hilos para cada Namenode. Se debe incrementar si se tiene un cluster largo y ocupado.
- dfs.replication (por defecto: 3). Número de veces que se replica un fichero a lo largo de un cluster.
- dfs.replication.considerLoad (por defecto: true). Decide cargar un datanode cuando se elige la localización para la replicación.

core-site.xml

- s.default.name (por defecto file:///): URI del sistema.

- hadoop.tmp.dir (Por defecto: /tmp/hadoop-$user.name). Determina la ruta para guardar ficheros temporales.
- io.file.buffer.size (por defecto 4096). Determina la cantidad de datos almacenados durante las operaciones de lectura/escritura del disco.

Workflow

La configuración usada para llevar a cabo las pruebas ha sido:

Código 8.6 workflow.xml usado para las pruebas.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><workflow-app
    xmlns="uri:oozie:workflow:0.3" name="malware-workflow">

    <start to="clamscan"/>

    <action name="virustotal">
        <pig>
            <job-tracker>${job_tracker}</job-tracker>
            <name-node>${name_node}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queue_name}</value>
                </property>
                <property>
                    <name>mapred.compress.map.output</name>
                    <value>true</value>
                </property>
            </configuration>
            <script>scripts/virustotal.pig</script>
            <param>ZKHOSTS='${zk_hosts}'</param>
            <param>INPUT='${filesequence_path}'</param>
            <param>TIMEOUT_MS='${timeout_ms}'</param>
            <param>USE_DEVSHM='${use_devshm}'</param>
            <param>API_KEY='${virustotal_api_key}'</param>
            <param>TOPIC='${kafka_topic}'</param>
            <param>KAFKAHOSTS='${kafka_brokers}'</param>
            <file>${malware_library}#malwarepig.jar</file>
            <file>hdfs://hadoopnamenode.redborder.cluster:8020/user/oozie
                /cache.yml#cache.yml</file>
        </pig>
        <ok to="yara"/>
        <error to="delete2"/>
```

```
    </action>

    <action name="metascan">
        <pig>
            <job-tracker>${job_tracker}</job-tracker>
            <name-node>${name_node}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queue_name}</value>
                </property>
                <property>
                    <name>mapred.compress.map.output</name>
                    <value>true</value>
                </property>
            </configuration>
            <script>scripts/metascan.pig</script>
            <param>ZKHOSTS='${zk_hosts}'</param>
            <param>INPUT='${filesequence_path}'</param>
            <param>TIMEOUT_MS='${timeout_ms}'</param>
            <param>USE_DEVSHM='${use_devshm}'</param>
            <param>API_KEY='${metascan_api_key}'</param>
            <param>TOPIC='${kafka_topic}'</param>
            <param>KAFKAHOSTS='${kafka_brokers}'</param>
            <file>${malware_library}#malwarepig.jar</file>
            <file>hdfs://hadoopnamenode.redborder.cluster:8020/user/oozie
                /cache.yml#cache.yml</file>
        </pig>
        <ok to="virustotal"/>
        <error to="delete2"/>
    </action>

    <action name="clamscan">
        <pig>
            <job-tracker>${job_tracker}</job-tracker>
            <name-node>${name_node}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queue_name}</value>
                </property>
                <property>
                    <name>mapred.compress.map.output</name>
                    <value>true</value>
                </property>
            </configuration>
            <script>scripts/clamscan.pig</script>
            <param>ZKHOSTS='${zk_hosts}'</param>
            <param>INPUT='${filesequence_path}'</param>
            <param>TIMEOUT_MS='${timeout_ms}'</param>
```

```
            <param>USE_DEVSHM='${use_devshm}'</param>
            <param>TOPIC='${kafka_topic}'</param>
            <param>KAFKAHOSTS='${kafka_brokers}'</param>
            <file>${malware_library}#malwarepig.jar</file>
            <file>hdfs://hadoopnamenode.redborder.cluster:8020/user/oozie
                /cache.yml#cache.yml</file>
        </pig>
        <ok to="metascan"/>
        <error to="delete2"/>
    </action>

    <action name="hasher">
        <pig>
            <job-tracker>${job_tracker}</job-tracker>
            <name-node>${name_node}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queue_name}</value>
                </property>
                <property>
                    <name>mapred.compress.map.output</name>
                    <value>true</value>
                </property>
            </configuration>
            <script>scripts/hasher.pig</script>
            <param>ZKHOSTS='${zk_hosts}'</param>
            <param>INPUT='${filesequence_path}'</param>
            <param>TIMEOUT_MS='${timeout_ms}'</param>
            <param>USE_DEVSHM='${use_devshm}'</param>
            <param>TOPIC='${kafka_topic}'</param>
            <param>KAFKAHOSTS='${kafka_brokers}'</param>
            <file>${lib_scripts}#scripts</file>
            <file>${malware_library}#malwarepig.jar</file>
            <file>${scripts_path}#scripts</file>
            <file>hdfs://hadoopnamenode.redborder.cluster:8020/user/oozie
                /cache.yml#cache.yml</file>
        </pig>
        <ok to="clamscan"/>
        <error to="delete2"/>
    </action>

    <action name="kaspersky">
        <pig>
            <job-tracker>${job_tracker}</job-tracker>
            <name-node>${name_node}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queue_name}</value>
```

```
            </property>
            <property>
                <name>mapred.compress.map.output</name>
                <value>true</value>
            </property>
        </configuration>
        <script>scripts/kaspersky.pig</script>
        <param>ZKHOSTS='${zk_hosts}'</param>
        <param>INPUT='${filesequence_path}'</param>
        <param>TIMEOUT_MS='${timeout_ms}'</param>
        <param>USE_DEVSHM='${use_devshm}'</param>
        <param>TOPIC='${kafka_topic}'</param>
        <param>KAFKAHOSTS='${kafka_brokers}'</param>
        <file>${malware_library}#malwarepig.jar</file>
        <file>${scripts_path}#scripts</file>
        <file>hdfs://hadoopnamenode.redborder.cluster:8020/user/oozie
            /cache.yml#cache.yml</file>
    </pig>
    <ok to="clamscan"/>
    <error to="delete2"/>
</action>

<action name="yara">
    <pig>
        <job-tracker>${job_tracker}</job-tracker>
        <name-node>${name_node}</name-node>
        <configuration>
            <property>
                <name>mapred.job.queue.name</name>
                <value>${queue_name}</value>
            </property>
            <property>
                <name>mapred.compress.map.output</name>
                <value>true</value>
            </property>
        </configuration>
        <script>scripts/yara.pig</script>
        <param>ZKHOSTS='${zk_hosts}'</param>
        <param>INPUT='${filesequence_path}'</param>
        <param>TIMEOUT_MS='${timeout_ms}'</param>
        <param>USE_DEVSHM='${use_devshm}'</param>
        <param>TOPIC='${kafka_topic}'</param>
        <param>KAFKAHOSTS='${kafka_brokers}'</param>
        <file>${malware_library}#malwarepig.jar</file>
        <file>${lib_scripts}#scripts</file>
        <file>${yara_rules}#yara_rules</file>
        <file>hdfs://hadoopnamenode.redborder.cluster:8020/user/oozie
            /cache.yml#cache.yml</file>
    </pig>
    <ok to="cuckoo"/>
```

```
        <error to="delete2"/>
    </action>

    <action name="cuckoo">
        <pig>
            <job-tracker>${job_tracker}</job-tracker>
            <name-node>${name_node}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queue_name}</value>
                </property>
                <property>
                    <name>mapred.compress.map.output</name>
                    <value>true</value>
                </property>
            </configuration>
            <script>scripts/cuckoo.pig</script>
            <param>ZKHOSTS='${zk_hosts}'</param>
            <param>INPUT='${filesequence_path}'</param>
            <param>TIMEOUT_MS='${timeout_ms}'</param>
            <param>USE_DEVSHM='${use_devshm}'</param>
            <param>MAX_RETRIES='${cuckoo_max_retries}'</param>
            <param>TOPIC='${kafka_topic}'</param>
            <param>CUCKOO_SERVER='${cuckoo_server}'</param>
            <param>KAFKAHOSTS='${kafka_brokers}'</param>
            <file>${malware_library}#malwarepig.jar</file>
            <file>hdfs://hadoopnamenode.redborder.cluster:8020/user/oozie
                /cache.yml#cache.yml</file>
        </pig>
        <ok to="delete"/>
        <error to="delete2"/>
    </action>

    <action name="delete">
        <fs>
            <delete path='${filesequence_path}'/>
        </fs>
        <ok to="end"/>
        <error to="fail"/>
    </action>

    <action name="delete2">
        <fs>
            <delete path='${filesequence_path}'/>
        </fs>
        <ok to="fail"/>
        <error to="fail"/>
    </action>
```

```
    <kill name="fail">
        <message>Pig failed, error message[${wf:errorMessage(wf:
            lastErrorNode())}]</message>
    </kill>

    <end name="end"/>

</workflow-app>
```

Casos de uso

8.1 Caso de uso 1 - Detección de e-mails con ficheros adjuntos con Malware

El objetivo de esta prueba es identificar el usuario que ha enviado un fichero con malware. Esto se hará desde la interfaz web de rb-Malware.

Para enviar el correo, se ha usado un script escrito en ruby.

Código 8.7 mailsender.rb.

```
#!/usr/bin/env ruby -w

require 'mail'

Mail.defaults do
  delivery_method :smtp, address: "10.0.203.55", port: 25
end

mail = Mail.new do
    from   'mvalero.ext@redborder.net'
    to     'malware@redborder.net'
    subject 'This is a test email'
    body   "test"
    ARGV.each do |file|
        add_file file
    end
end

mail.deliver
```

Si accedemos a la interfaz web, podemos ver los tipos de malware que han sido detectados en la última hora.

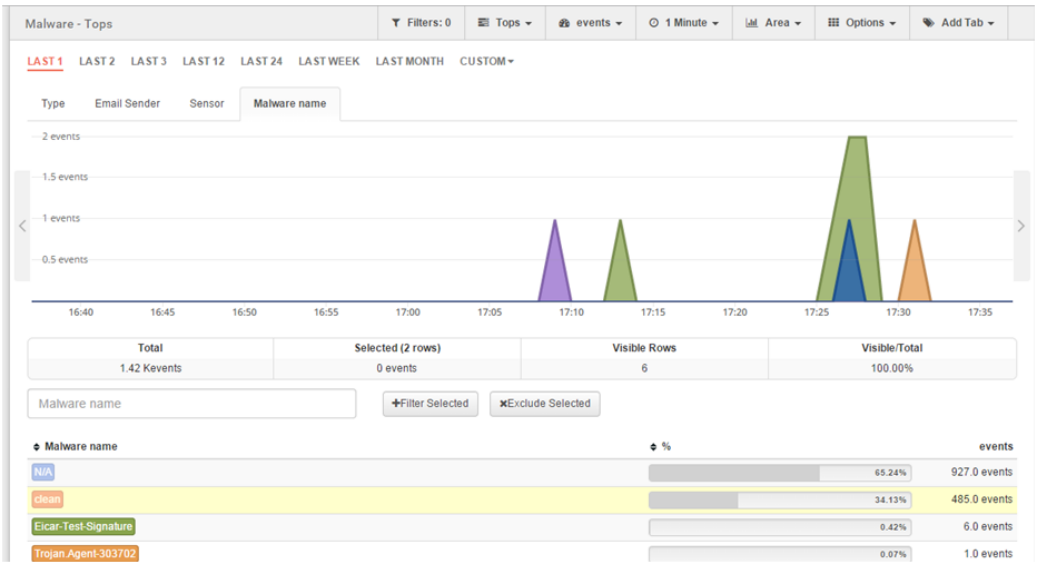

Figura 8.1 Nombres de malware detectados por el sistema en la última hora.

Seleccionamos como filtro los troyanos “Trojan Agent-303702” y “Win.Trojan.Scar-40” para ver los ficheros que los han provocado.

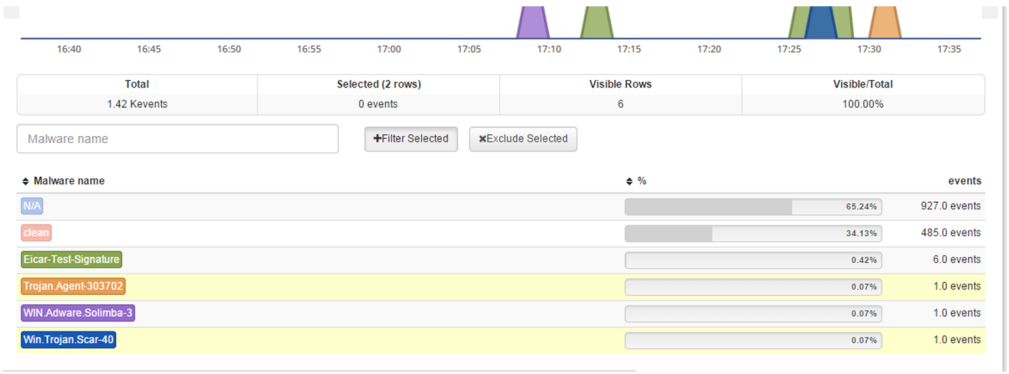

Figura 8.2 Filtrado.

Vemos el SHA de los ficheros

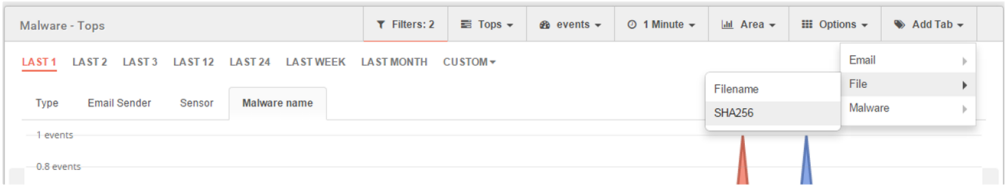

Figura 8.3 Selección de SHA256.

Filtramos por ficheros

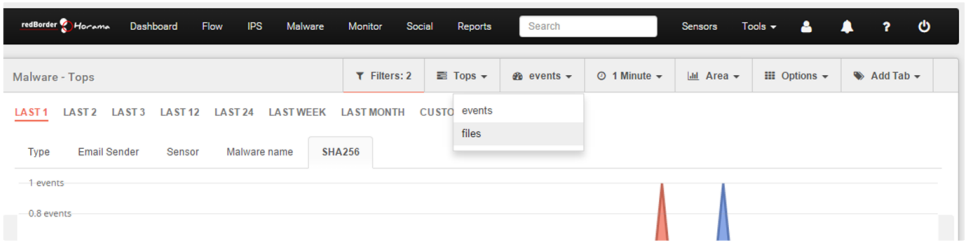

Figura 8.4 Filtro por ficheros.

Vemos que hay dos ficheros en los que se han encontrado troyanos en la última hora.

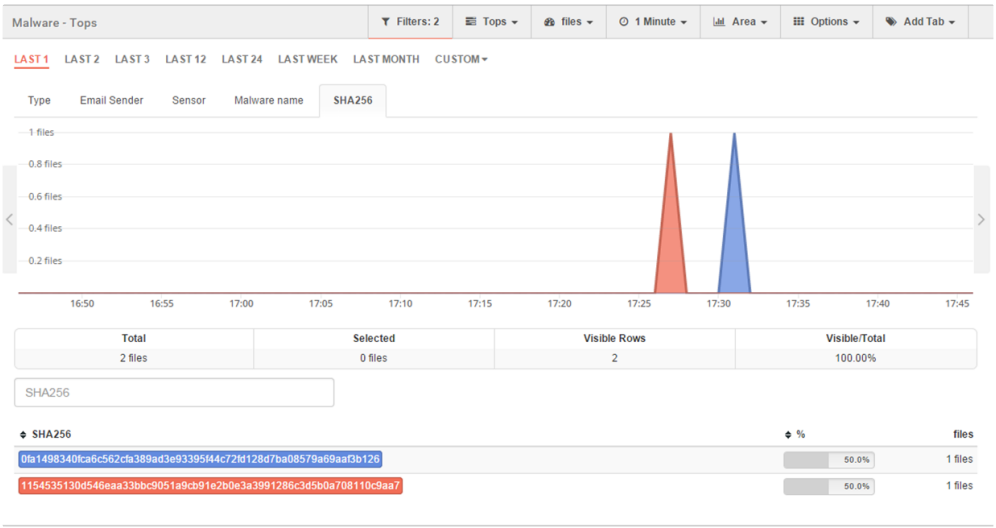

Figura 8.5 SHA256 de los ficheros detectados.

Seleccionamos un fichero y filtramos por este.

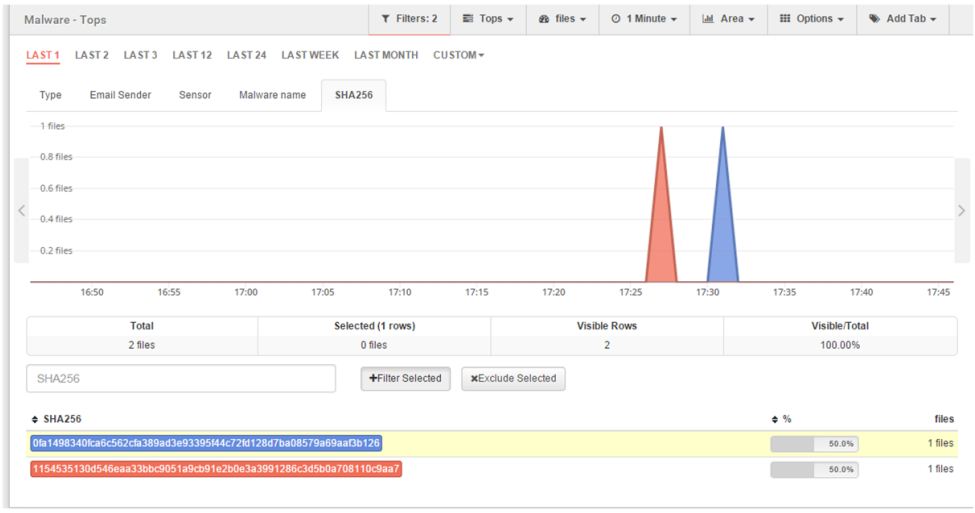

Figura 8.6 Selección de un fichero.

Y quitamos los otros filtros.

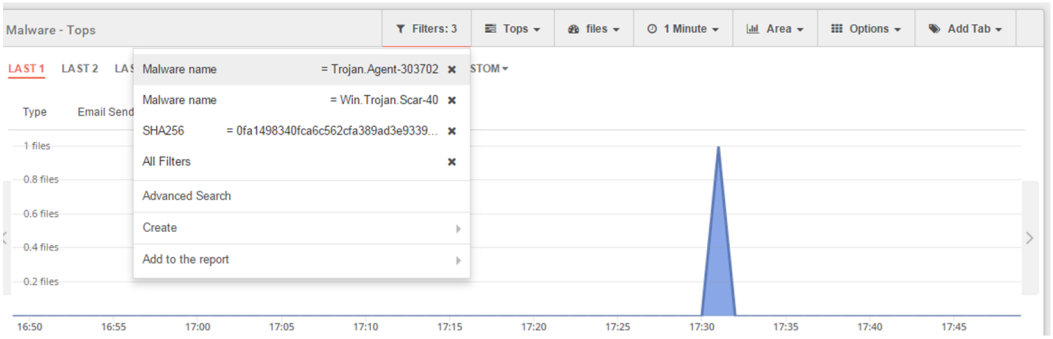

Figura 8.7 Supresión de filtros innecesarios.

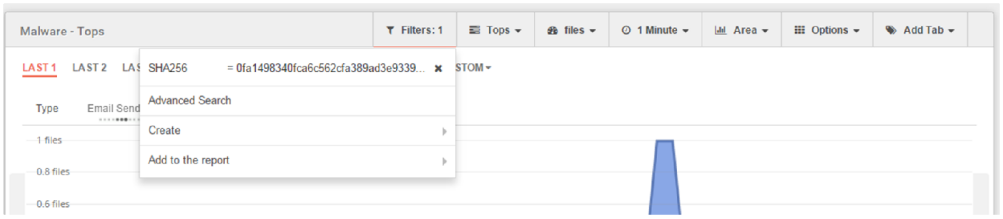

Figura 8.8 Filtros actuales.

Y ahora podemos ver quién ha enviado ese fichero.

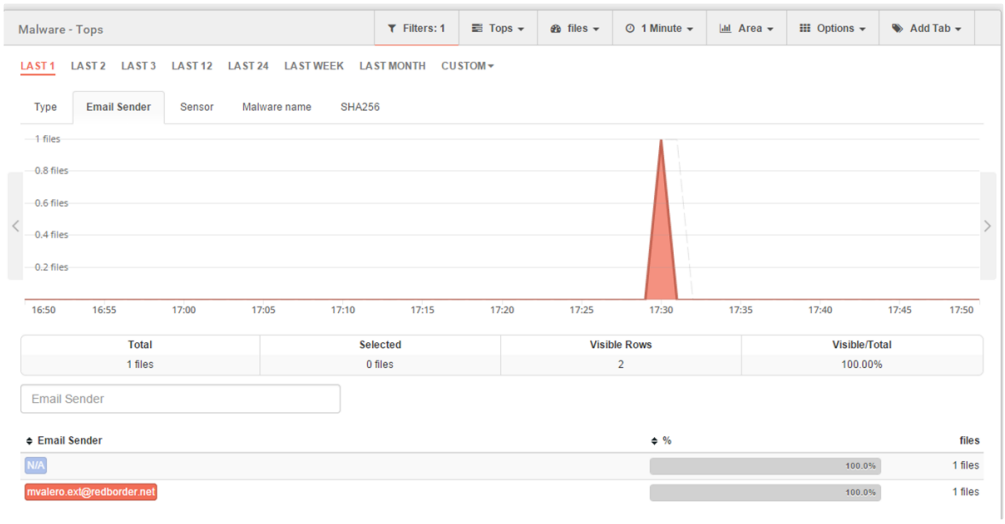

Figura 8.9 Usuario que ha enviado el email con malware adjunto.

8.2 Caso de uso 2 - Detección de Malware en tráfico web

El objetivo de esta prueba es identificar la detección de un fichero malicioso que se descarga desde una máquina que se encuentra en la red local. Esto se hará desde la interfaz web de rb-Malware.

Nos descargamos el archivo “malware.mp3” desde la terminal en Linux.

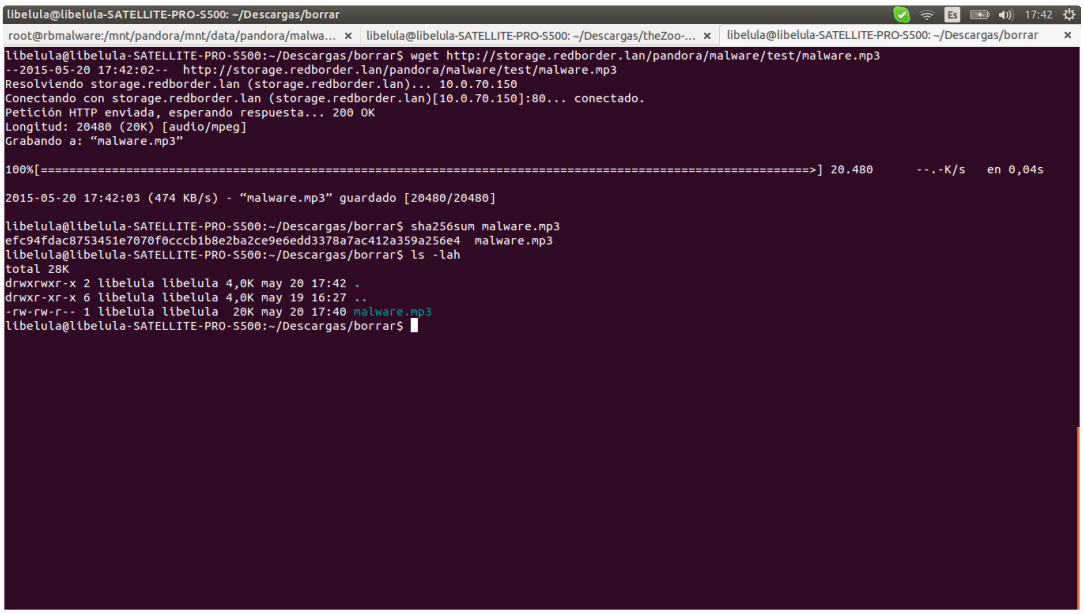

Figura 8.10 Descarga de malware.

Buscamos los primeros dígitos del sha256 de “malware.mp3” en la web

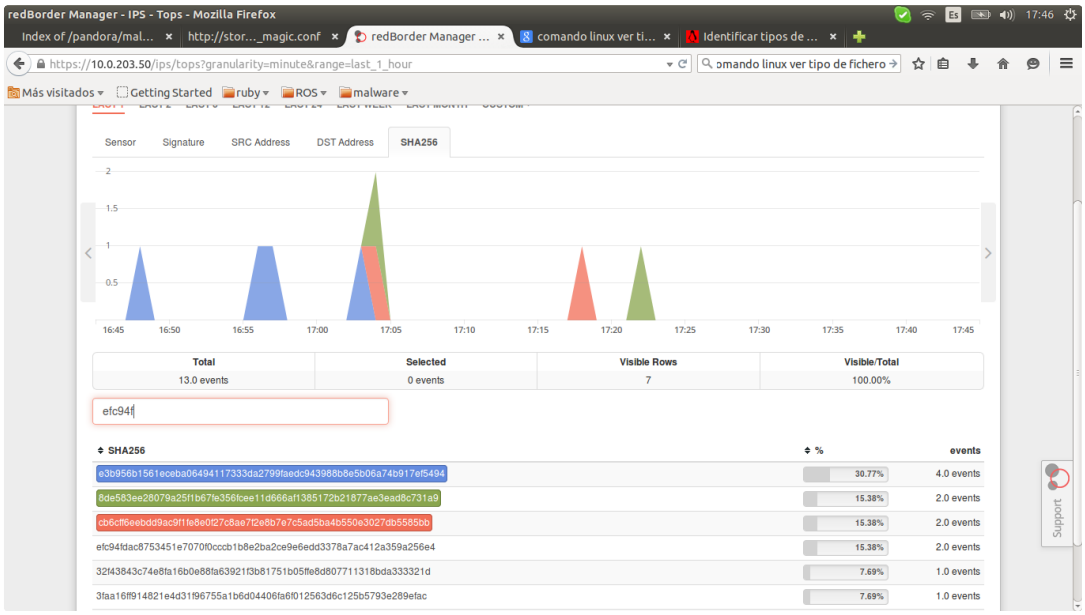

Figura 8.11 Búsqueda del sha256 del fichero.

Añadimos como filtro el sha256 correspondiente del fichero como.

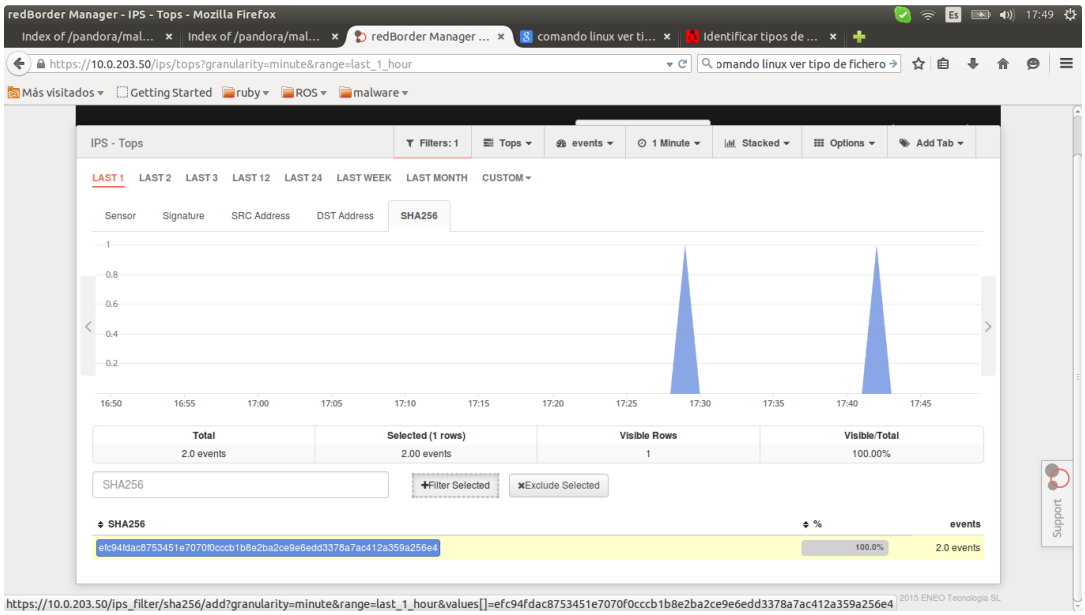

Figura 8.12 Filtrado por sha256.

Quitamos el filtro con la primera parte del sha256

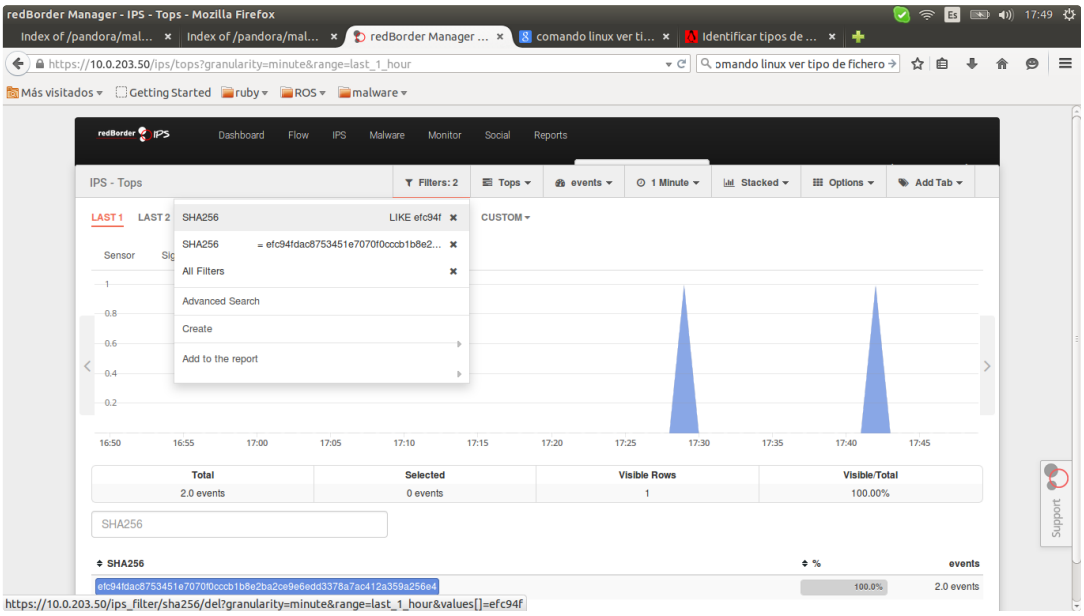

Figura 8.13 Filtrado por sha256.

Vamos a la parte del malware y comprobamos que lo ha detectado ClamAV.

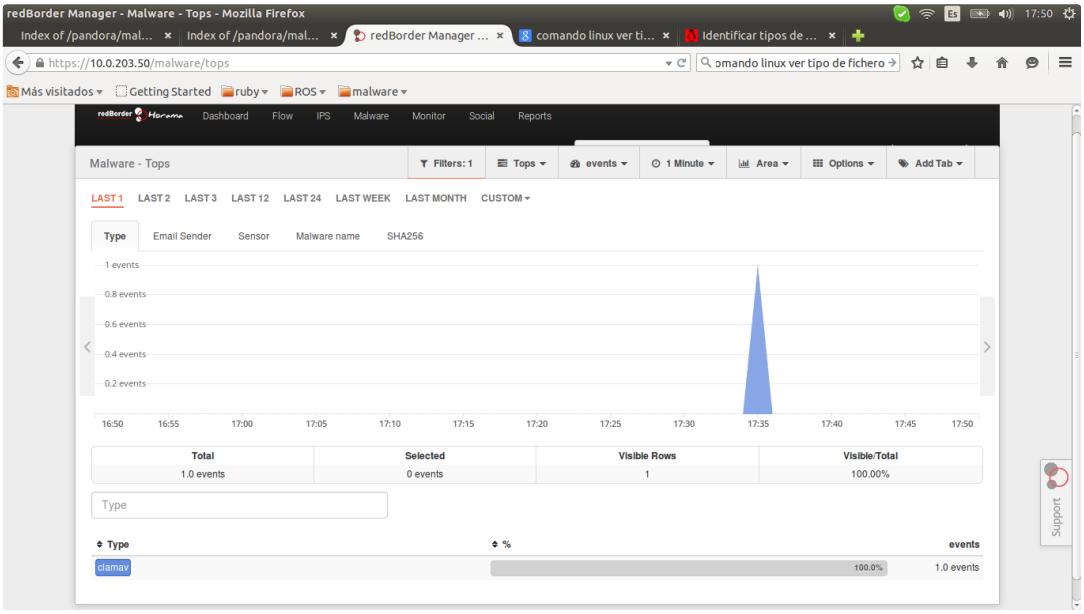

Figura 8.14 Detección del fichero.

Vemos el nombre del malware del fichero “malware.mp3”.

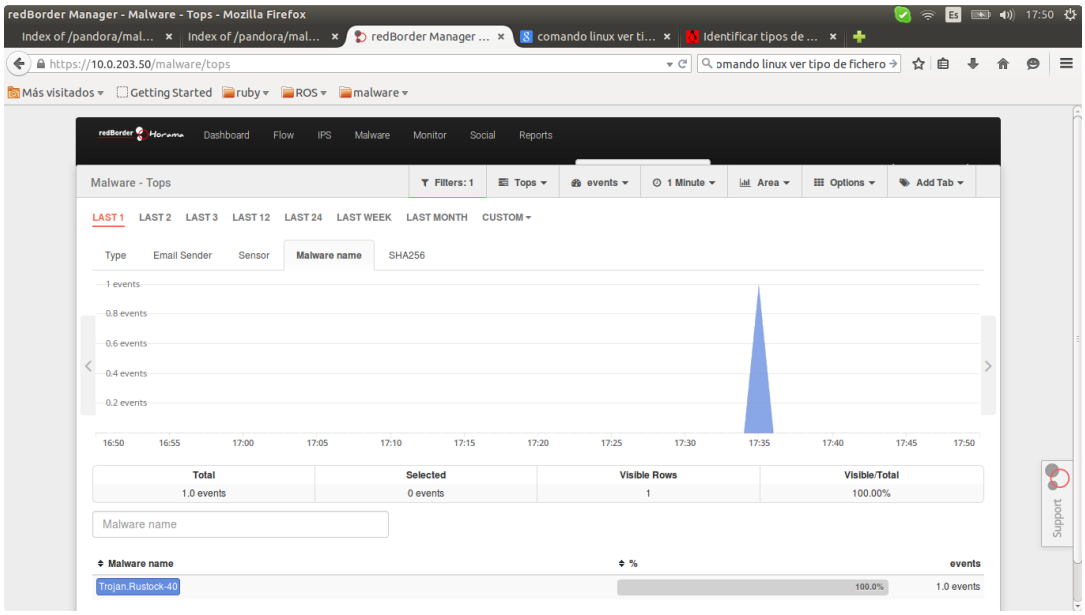

Figura 8.15 Nombre del fichero.

KasperskyLoader

Código desarrollado para la definición de la función de carga de Kaspersky para Pig.

Código 8.8 Script de pig para el análisis de ficheros usando Kaspersky.

```
REGISTER malwarepig.jar;

data = LOAD '$INPUT' USING net.redborder.malware.loaders.kaspersky.
    KasperskyLoader('$ZKHOSTS', '$TIMEOUT_MS', '$USE_DEVSHM');
STORE data INTO 'this-is-ignored' USING net.redborder.malware.storage.
    KafkaStorage('$TOPIC', '$KAFKAHOSTS');
```

Código 8.9 Código de la función UDF KasperskyLoader.

```
public class KasperskyLoader extends ExecutingJsonLoader {
    public KasperskyLoader(String zkHost) {
        super(zkHost, SCRIPT_KASPERSKY,"kaspersky");
    }

    public KasperskyLoader(String zkHost, String timeoutMS) {
        super(zkHost, SCRIPT_KASPERSKY, timeoutMS,"kaspersky");
    }

    public KasperskyLoader(String zkHost, String timeoutMS, String
        useDevShm) {
        super(zkHost, SCRIPT_KASPERSKY, timeoutMS, useDevShm,"kaspersky")
            ;
    }
}
```

Código 8.10 Script de ejecución de Kaspersky.

```
if (/opt/kaspersky/kav4fs/bin/kav4fs-control --scan-file $@ | grep "
    Threats found" | grep -q "0"); then echo OK; else echo MALWARE; fi
```

KafkaStorage

Código desarrollado para la definición de la función de almacenamiento de Kafka para Pig.

Código 8.11 Código de la función UDF KafkaStorage.

```
public class KafkaStorage extends StoreFunc {
    private Producer<String, String> kafkaProducer;
    private String topic;
    String kafkaServer;
    ByteArrayOutputStream mOut;
    Logger log;


    public KafkaStorage(String topic, String kafkaServer) {
        this.topic = topic;
        this.kafkaServer = kafkaServer;
        log = RbLogger.getLogger(KafkaStorage.class.getName());
    }

    @Override
    public org.apache.hadoop.mapreduce.OutputFormat getOutputFormat()
        throws IOException {

        return new NullOutputFormat();
    }

    @Override
    public void setStoreLocation(String s, org.apache.hadoop.mapreduce.
        Job job) throws IOException {
    }

    @Override
    public void prepareToWrite(org.apache.hadoop.mapreduce.RecordWriter
        recordWriter) throws IOException {
        final Properties props = new Properties();
        props.put("metadata.broker.list", kafkaServer);
        props.put("serializer.class", "kafka.serializer.StringEncoder");
        props.put("request.required.acks", "1");
```

```
        props.put("message.send.max.retries", "60");
        props.put("retry.backoff.ms", "1000");
        props.put("producer.type", "sync");
        props.put("queue.buffering.max.messages", "500");
        props.put("queue.buffering.max.ms", "250");
        kafkaProducer = new Producer<String, String>(new ProducerConfig(
            props));
    }

    @Override
    public void putNext(Tuple tuple) throws IOException {

        if (tuple.size() > 0) {
            JSONObject jsonObj;
            jsonObj = new JSONObject();
            for (int i = 0; i < tuple.size(); i++) {
                Object field;
                try {
                    field = tuple.get(i);
                } catch (ExecException ee) {
                    throw ee;
                }

                putField(field, i, jsonObj);
            }

            kafkaProducer.send(new KeyedMessage<String, String>(topic,
                jsonObj.toString()));
            jsonObj = new JSONObject();

        } else{
            log.warning("Tuple size is 0.");
        }
    }

    @SuppressWarnings("unchecked")
    private void putField(Object field, int i, JSONObject jsonObj)
        throws IOException {

        try {
            switch (DataType.findType(field)) {
                case DataType.NULL:
                    jsonObj.put(String.valueOf(i), "NULL");
                    break;
                case DataType.BOOLEAN:
                    jsonObj.put(String.valueOf(i), (boolean) field);
                    break;
                case DataType.INTEGER:
                    jsonObj.put(String.valueOf(i), (int) field);
                    break;
```

```
                case DataType.LONG:
                    jsonObj.put(String.valueOf(i), (long) field);
                    break;
                case DataType.FLOAT:
                    jsonObj.put(String.valueOf(i), (float) field);
                    break;
                case DataType.DOUBLE:
                    jsonObj.put(String.valueOf(i), (double) field);
                    break;
                case DataType.BYTEARRAY:
                    byte[] b = ((DataByteArray) field).get();
                    jsonObj.put(String.valueOf(i), b);
                    break;
                case DataType.CHARARRAY:
                    jsonObj.put(String.valueOf(i), (String) field);
                    break;
                case DataType.BYTE:
                    jsonObj.put(String.valueOf(i), (byte) field);
                    break;

                case DataType.MAP:
                    boolean mapHasNext = false;
                    Map<String, Object> m = (Map<String, Object>) field;
                    for (Map.Entry<String, Object> e : m.entrySet()) {
                        jsonObj.put(e.getKey(), e.getValue());
                    }
                    break;

                case DataType.TUPLE:
                    Tuple t = (Tuple) field;
                    for (int n = 0; n < t.size(); ++n) {
                        try {
                            putField(t.get(n), i, jsonObj);
                        } catch (ExecException ee) {
                            throw ee;
                        }
                    }
                    break;

                case DataType.BAG:
                    Iterator<Tuple> tupleIter = ((DataBag) field).iterator
                        ();
                    while (tupleIter.hasNext()) {
                        putField(tupleIter.next(), i, jsonObj);
                    }
                    break;

                default:
                    throw new RuntimeException("Unknown datatype " +
                        DataType.findType(field));
```

```
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}
```

Código 8.12 Ejemplo de uso de KafkaStorage en un script de Pig.

```
REGISTER malwarepig.jar;

data = LOAD '$INPUT' USING net.redborder.malware.loaders.av.
    ClamScanDaemonLoader('$ZKHOSTS', '$TIMEOUT_MS', '$USE_DEVSHM');
STORE data INTO 'this-is-ignored' USING net.redborder.malware.storage.
    KafkaStorage('$TOPIC', '$KAFKAHOSTS');
```

Bibliografía

[1] Documentación de YARA (yara-project.googlecode.com/files/YARA %20User's %20 Manual %201. 6.pdf)

[2] Documentación de ClamAV (github.com/vrtadmin/clamav-faq/raw/master/manual/ clamdoc.pdf)

[3] Practical Malware Analysis. Autores: Michael Sikorski y Andrew Honig

[4] Cuckoo Malware Analysis. Autores: Digit Oktavianto y Iqbal Muhardianto

[5] Optimizing Hadoop for MapReduce. Autor: Khaled Tannir

[6] Malware Analyst's Cookbook. Autores: Michael Hale Ligh, Steven Adair, Blake Hartstein and Matthew Richard

[7] Hadoop: The Definitive Guide. Autor: Tom White

[8] Pig in Action. Autor: Chuck Lam

[9] BinaryPig: Scalable Static Binary Analysis Over Hadoop. Autores: Zachary Hanif, Telvis Calhoun y Jason Trost

[10] Sistema de ofuscación de malware para la evasión de NIDS, 2013. Autores: Roberto Carrasco de la Fuente, Sergio Gumiel Erena y Adrián Vizcaíno González

[11] Análisis de características estáticas de ficheros ejecutables para la clasificación de malware, 2014. Autor: Richard Rivera Guevara

[12] Clasificación de malware mediante clusterización, 2012. Autores: González Medina Lilia Elena y Salazar Rubio José Miguel.

[13] Securelist. IT threat evolution in Q2 2015, July 2015. Autores: David Emm, Maria Garnaeva, Anton Ivanov, Denis Makrushin y Roman Unuchek.

[14] Field Guide to Hadoop, 2015. Autores: Kevin Sitto y Marshall Presser.

[15] Descripción de PiggyBank en la página oficial de Apache Pig (cwiki.apache.org/confluence/ display/PIG/PiggyBank).

[16] Página Oficial de Oozie (oozie.apache.org/)

[17] Página Oficial de Hadoop (hadoop.apache.org/)

[18] Learning Apache Kafka, 2015. Autor: Nishant Garg

[19] Zookeeper, 2013. Autor: Flavio Junqueira y Benjamin Reed

[20] Página de Riak en Github (github.com/basho/riak)

[21] Página oficial de VirusTotal (www.virustotal.com/)

[22] Página oficial de Metascan (www.metascan-online.com/)

[23] Descripción del servicio Security for File Server de Kaspersky (www.kaspersky.co.uk/business-security/file-server)

[24] Fuzzy Hashing Techniques in Applied Malware Analysis, 2012. Autores: David French, William Casey

[25] Página oficial de MISP en Github (github.com/MISP/MISP)

[26] Página de Application Advisor (whitelist.kaspersky.ru/advisor)

[27] Encuesta sobre Equipamiento y Uso de Tecnologías de Información y Comunicación en los Hogares, Año 2014. Instituto Nacional de estadística

Índice de Figuras

2.1	Ejemplo de la ejecución de trabajos MapReduce	4
2.2	Evolución de la estructura de Hadoop	5
2.3	Arquitectura de un cluster de Kafka	6
2.4	Arquitectura de un cluster de Zookeeper	7
3.1	Ejemplo de arquitectura de Cuckoo	12
4.1	Arquitectura del sistema BinaryPig	14
4.2	Diagrama de funcionamiento de rb-sequenze-oozie	15
4.3	Ejecución de los script usando KafkaStorage	19
4.4	Workflow usado en la versión 0.4 de rB Malware	24
5.1	Escenario de laboratorio	26
5.2	Flujo de entrada de correo electrónico	27
8.1	Nombres de malware detectados por el sistema en la última hora	51
8.2	Filtrado	51
8.3	Selección de SHA256	51
8.4	Filtro por ficheros	52
8.5	SHA256 de los ficheros detectados	52
8.6	Selección de un fichero	53
8.7	Supresión de filtros innecesarios	53
8.8	Filtros actuales	53
8.9	Usuario que ha enviado el email con malware adjunto	54
8.10	Descarga de malware	54
8.11	Búsqueda del sha256 del fichero	55
8.12	Filtrado por sha256	55
8.13	Filtrado por sha256	56
8.14	Detección del fichero	56
8.15	Nombre del fichero	57

Índice de Tablas

5.1	Caso de uso 1 - parte 1	28
5.2	Caso de uso 1 - parte 2	29
5.3	Caso de uso 1 - parte 3	30
5.4	Caso de uso 2 - parte 1	31
5.5	Caso de uso 2 - parte 2	32
6.1	Presupuesto para el desarrollo	33
6.2	Presupuesto para el equipamiento	33
8.1	Ejemplo de fichero a rellenar para realizar las pruebas	40

Índice de Códigos

3.1	Regla de Yara	10
4.1	Bucle principal del programa OozieLeader	16
4.2	Función ejecutada por uno de los OozieManager cuando Zookeeper lo manda a trabajar	17
4.3	Ejemplo de configuración y envío de un trabajo al servidor de Oozie	18
4.4	Método prepareToWrite() de KafkaStorage	20
4.5	Método putNext() de KafkaStorage	20
4.6	Método putField() usado por putNext()	21
4.7	Script para la ejecución de kaspersky	22
8.1	Activación del Fair Scheduler	38
8.2	fair-scheduler.xml	38
8.3	Configurar la cola de prioridad en el Workflow de Oozie	39
8.4	Crear datos de entrada para Hadoop usando Terasort	40
8.5	Análisis de datos en Hadoop usando Terasort	40
8.6	workflow.xml usado para las pruebas	44
8.7	mailsender.rb	50
8.8	Script de pig para el análisis de ficheros usando Kaspersky	58
8.9	Código de la función UDF KasperskyLoader	58
8.10	Script de ejecución de Kaspersky	58
8.11	Código de la función UDF KafkaStorage	59
8.12	Ejemplo de uso de KafkaStorage en un script de Pig	62