Universidad Autónoma de Madrid
Escuela Politécnica Superior

Doble Máster Universitario en Ingeniería Informática (ing.INF) y en Innovación en TIC (i2-TIC)

DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN WEB PARA EL ANÁLISIS CENTRALIZADO DE LOGS DE SEGURIDAD

Grado en Ingeniera Informática

Carlos López Cruces

Enero de 2016

DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN WEB PARA EL ANÁLISIS CENTRALIZADO DE LOGS DE SEGURIDAD

AUTOR: Carlos López Cruces
TUTOR: David Arroyo Guardeño

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Enero de 2016

Agradecimientos

A mis padres por haberme apoyado y dado ánimos en todo momento, incluso aunque no me los mereciera.

A mis amigos Rafa y Guada por interesarse por mi trabajo.

A mi hermano por esas extrañas discusiones que tenemos a veces y que me permite liberar tensiones.

A David Arroyo por las enormes charlas que tuvimos en las tutorías del proyecto y las numerosas ideas aportadas para sacar adelante este trabajo.

Y en menor medida, a mi ordenador portátil por soportar unos cinco formateos de disco desde el inicio del proyecto y muchas de esas preguntas de "¿por qué no funcionas?" cuando la respuesta era más que evidente. Cualquiera habría salido quemado.

Resumen

Resumen

Durante los últimos años, ha habido un incremento significativo del uso de tecnologías de la información y los dispositivos móviles. En consecuencia, el aumento de la demanda de las aplicaciones software ha hecho que el proceso de desarrollo, en muchas ocasiones, sea incompleto debido a que no se implementan los controles de seguridad necesarios. Dicha proliferación de aplicaciones y la movilidad de los usuarios finales han supuesto un gran incremento de la complejidad del flujo de información en las redes de comunicación contemporáneas. Con esta enorme cantidad de información, los mecanismos de protección tradicionales corren el riesgo de quedar obsoletos y es esencial investigar y desarrollar nuevos métodos en la situación actual de la seguridad informática.

La mayoría de aplicaciones guarda registros de actividad, *logs*. Estos registros facilitan la corrección de errores y, en seguridad informática, pueden utilizarse para la detección de ataques y la auditoría del sistema. Los *logs* pueden registrar una gran cantidad de información y pueden tener una gran variedad de formatos ya que no existe un formato común. Es por ello que una herramienta que permita centralizar y procesar esta información puede ser interesante.

El propósito de este trabajo es realizar una primera fase de investigación, diseño e implementación de un proyecto llamado *LogExplorer*. Este proyecto trata de diseñar y desarrollar una aplicación web que servirá de herramienta para centralizar la información de *logs*. Proporciona métodos para recopilar, almacenar, analizar y visualizar los datos de *logs*. Este proyecto está basado en *J2EE* y *Spring Framework*. Uno de los mayores retos de diseño en este proyecto es dotarla de suficiente escalabilidad. Ya que la seguridad necesita cambiar continuamente.

Para realizar el análisis de esta herramienta, se ha estudiado el desafío *VAST Challenge 2012 MC2*. Este ejemplo muestra una situación simulada en la que se detecta el ataque de una *botnet* y su comportamiento a través de los registros del *cortafuegos* y del *IDS*.

Esta herramienta se basa en el patrón modelo-vista-controlador y utiliza *Maven* para gestionar las dependencias y bibliotecas. La base de datos se realiza mediante *PostgreSQL* y la aplicación se conecta a ella a través de *Hibernate*. Para la implementación de métodos de aprendizaje automático, como el algoritmo *K-means* o el Análisis de Componentes Principales (*PCA*), *LogExplorer* utiliza *Weka*. Para implementar la autenticación y autorización de usuarios se ha hecho uso de la extensión de *Spring*, *Spring Security*. La interfaz web de la aplicación se realizó a través de una plantilla, llamada *AdminLTE*. Esta plantilla utiliza *Bootstrap*, un popular *framework* de HTML, CSS y *JavaScript* para proyectos web.

Palabras Clave

Seguridad Informática, Informática forense, Análisis de *logs*, *IDS*, *SIEM*, Aplicación Web, *J2EE*, *Spring framework*

Abstract

Over the last few years, there has been a significant increase in the use of information technology and mobile devices. Consequently, the increase in demand for software applications has determined incomplete development processes, in many occassions due to flawness in the concretion of security controls. This increase of applications and the mobility of end users have result in a much more complex information flow between the components of modern communication networks. With this huge amount of information, traditional protection mechanisms could be obsolete and thus it is essential to research and develop new methods adequate for the current status of computer security. Most applications record activity in files, the so-called log files. These logs are key components to detect and correct system errors, and in computer security could be used to detect attacks and system audit. Logs can record a large amount of information and may have a variety of formats, because there is not a common format. That is why a tool to centralize and process this information may be of interest.

The purpose of this papper is to perform a first phase of research, design and developing of a project named LogExplorer. This project involves the design and the implementation of a web application to centralize the information extracted from log files with diversity in location and format. It provides methods to collect, store, analyze and visualize log data. This project is based on J2EE and the Spring Framework. One of the most challenging design problems in this project is implement with enough scalability. Because the security needs to change continuously.

For the analysis of this tool, it has been studied the challenge VAST Challenge 2012 MC2. This example shows an simulated situation in which the attack of a botnet and its behavior is detected by analyzing the firewall and IDS logs.

This tool is based on model-view-controller pattern and it uses Apache Maven to manage dependencies and libraries. The Database is made by PostgreSQL and the application connects to it through Hibernate. For the implementation of data mining methods, like K-means or Principal Components Analysis (PCA), LogExplorer use Weka. To implement user authentication and authorization has made use of Spring extension, Spring Security. The web interface of the application was performed using a template, called AdminLTE. This template uses Bootstrap, a popular HTML, CSS and JavaScript framework for web projects.

Key Words

Computer Security, Computer forensics, Log analysis, *IDS*, *SIEM*, Web application, *J2EE*, *Spring framework*

Índice general

Índice de figuras

Índice de tablas

Glosario

Antivirus Los antivirus son programas informáticos que tiene el propósito de detectar y eliminar virus u otros programas perjudiciales antes o después de que ingresen al sistema. 8

Botnet Es una red de robots informáticos o bots, que se ejecutan de manera autónoma y automática. El artífice de la botnet puede controlar todos los ordenadores/servidores infectados de forma remota. Es creado por y para requerimientos de capacidad de cálculo y se usan para diversas actividades criminales y también para investigaciones científicas. Existen algunos botnets legales tanto como ilegales. 17, 19–22, 69

Cortafuegos (En inglés *Firewall*), es un dispositivo de una red diseñado para bloquear el acceso no autorizado y permitir el acceso autorizado. El cortafuegos configurado correctamente es una protección necesaria en la seguridad informática, pero nunca una protección suficiente. 2, 9, 16–19, 21, 22, 69

CSS (Del inglés Cascading Style Sheets) Hoja de estilo en cascada es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML. 67

CSV (Siglas del inglés comma-separated values) Es un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas. 12, 23, 60, 64

Curva ROC (Del inglés *Receiver Operating Characteristic*, o Característica Operativa del Receptor) Es una representación gráfica de la eficacia de un clasificador binario. 8

DAO (Del inglés Data Access Object) Es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo. 48, 65, 66

DoS (Del inglés, Denial of Service) Ataque de denegación de servicio. Es un ataque a un sistema de ordenadores o una red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos. Normalmente provoca la pérdida de la conectividad de la red por el consumo del ancho de banda de la red de la víctima o sobrecarga de los recursos computacionales del sistema. 6

Exfiltración La exfiltración de datos es la transferencia no autorizada de los datos de un ordenador. Es decir, la capacidad que tienen algunos programas malignos para obtener datos privados y sensibles de ordenadores y poder mandarlos, generalmente, mediante canales ocultos a otras personas que no tendrían acceso a dicha información de manera convencional. También conocida como la extrusión de datos. 17, 20–22

Firma digital Es un método criptográfico que permite determinar que el mensaje o la información no ha sido alterado desde que fue firmado por el creador y así verificar su integridad.. 9

Framework En español marco de trabajo, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software. 15, 47, 59, 69

FTP (En inglés File Transfer Protocol) Protocolo de Transferencia de Archivos. Es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP, basado en la arquitectura cliente-servidor. 19–21

Función hash Una función Hash es una función resumen que sirve de representación compacta de una cadena de entrada. 68

Host Un Host se refiere a las computadoras conectadas a una red, que proveen y utilizan servicios de ella. 6, 9, 11, 23, 24

HTML (Del inglés HyperText Markup Language) Es el lenguaje estándar para la elaboración de páginas web. 62

HTTP (Del inglés Hypertext Transfer Protocol) es el protocolo usado en cada transacción de la World Wide Web. 61

HTTPS (Del inglés Hypertext Transfer Protocol Secure) es un protocolo de aplicación para la transferencia de datos, comúnmente la versión segura de HTTP. Se basa en crear un canal cifrado entre el cliente y el servidor para que un atacante que esté escuchando el canal no pueda entender la información transmitida. 68, 70

IDS (Del inglés Intrusion Detection System) Sistemas de Detección de Intrusos, es un programa que detecta el acceso no autorizados a un computador o a una red. 2, 9–11, 16–19, 69

IoC (en inglés *Inversion of Control*) Inversión de control es un método de programación en el que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales. Tradicionalmente el programador especifica la secuencia de decisiones y procedimientos. En la inversión de control se especifican respuestas deseadas a sucesos o solicitudes. 47

IP Una dirección IP es una etiqueta numérica que identifica, de manera lógica y jerárquica, a una interfaz (elemento de comunicación/conexión) de un dispositivo (habitualmente una computadora) dentro de una red que utilice el protocolo IP (Internet Protocol). 18, 19, 22–24, 51, 60

IRC (Del inglés Internet Relay Chat) Es un protocolo de comunicación en tiempo real basado en texto, que permite comunicarse dos o más personas. 19, 21

J2EE (Del inglés Java Platform, Enterprise Edition) es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java. 2, 15, 47

JavaScript JavaScript es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. 52, 60, 62, 67, 69

JDBC (Del inglés Java Database Connectivity) es una interfaz que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, utilizando el dialecto SQL del modelo de base de datos que se utilice. 48

JSON (Del inglés JavaScript Object Notation) es un formato abierto estándar derivado de JavaScript que por su simplicidad se ha masificado su uso en internet y las aplicaciones web como medio de intercambio de datos. Este formato se describe en dos estándares: RFC 7159 y ECMA-404. 12, 23, 60, 62, 63

JSP (Del inglés JavaServer Pages) Es una tecnología que facilita a desarrolladores a crear páginas web dinámicas basadas en HTML, XML, entre otros tipos de documentos. Similar a PHP, pero usa el lenguaje de programación Java. 60, 61, 67, 69

JSTL (Del inglés JavaServer Pages Standard Tag Library) Es un componente que proporciona bibliotecas de etiquetas para el desarrollo de páginas web dinámicas. Estas bibliotecas de etiquetas extienden de la especificación de JSP. 60

K-means Es un método de agrupamiento en la minería de datos que busca realizar particiones de un conjunto de datos de n observaciones en k grupos. 8

Log Equivale a bitácora en español. En informática es un registro oficial de eventos durante un rango de tiempo. Se usa para registrar datos o información de un evento que ocurre en un dispositivo en particular o en una aplicación. 1, 2, 8, 9, 11, 12, 15, 16, 18, 19, 23, 24, 26–29, 32, 35–38, 40, 42, 49, 55–57, 63–65, 68–70

MAC (Del inglés Media Access Control) Una dirección MAC es un identificador de 75 bits que corresponde de forma única a una tarjeta o dispositivo de red. 18

MD5 (DEl inglés Message-Digest Algorithm 5) Es una implementación de una función hash ampliamente usado. 68

MVC (Del inglés model-view-controller) Modelo–vista–controlador, es un patrón de arquitectura de software que separa una aplicación en tres componentes distintos para distinguir entre la representación de la información y la interacción del usuario. 45

ORM (Del inglés Object-Relational mapping) mapeo objeto-relacional es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional. 47

Outliers En español valores atípicos, son observaciones que son numéricamente distantes del resto de los datos. 6

PCA (Del inglés, Principal component analysis) Análisis de componentes principales. Es una técnica utilizada para reducir la dimensionalidad de un conjunto de datos. Busca la proyección según la cual los datos queden mejor representados en términos de mínimos cuadrados. 8, 51, 60

POM (Del inglés Project Object Model) es un archivo XML que contiene la información sobre un proyecto software y su configuración usando Apache Maven. 46

Router También conocido como enrutador o encaminador de paquetes, y españolizado como rúter. Es un dispositivo que proporciona conectividad a nivel de red. 2

Servidor Proxy Un servidor Proxy es un servidor, que hace de intermediario en las peticiones de recursos que realiza un cliente a otro servidor distinto. Puede dar soporte a varias opciones: control de acceso, caché, mejorar el rendimiento. . . . 60

SHA-2 Es un conjunto de funciones hash criptográficas (SHA-224, SHA-256, SHA-384, SHA-512) diseñadas por la Agencia de Seguridad Nacional (NSA). 68

SIEM Es una combinación entre la gestión de información de seguridad (Security Information Management, SIM) y la gestión de eventos de seguridad (Security Event Management, SEM). 11, 12, 23

SQL (Del inglés Structured Query Language) Lenguaje de consulta estructurado, es un lenguaje declarativo que se usa en las bases de datos relacionales. 23, 65, 70

SSH (Del inglés Secure Shell) Es un protocolo para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos. 20–22

Tor (Siglas en inglés de The onion router, el router cebolla) es un protocolo de anonimato que permite a sus usuarios no revelar su dirección IP (anonimato a nivel de red) y que, además, mantiene la integridad y el secreto de la información. 60

URL (Del inglés Uniform Resource Locator) es un identificador de recursos uniforme cuyos recursos pueden variar en el tiempo. Están formados por una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que designa recursos en una red, como Internet. 48, 61

XML (Del inglés eXtensible Markup Language) es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. 12, 46, 47

Capítulo 1

Introducción

En los últimos años se ha producido una proliferación de tecnologías de la información y de una mayor conectividad de redes inalámbricas, así como de dispositivos móviles. Como consecuencia de ello se han desarrollado una gran cantidad de aplicaciones software con pequeñas y grandes diferencias entre sí. Esto supone que muchas de estas aplicaciones aparezcan con errores de programación o sin implementar métodos necesarios para protegerse de ataques informáticos [1].

En lo que respecta a la seguridad informática, factores como la mayor cantidad y variedad de usuarios de una aplicación, la mayor disponibilidad de sus servicios y la mayor variedad de dispositivos con los que se puede acceder a la misma, pueden suponer un desafío a la hora de monitorizar y gestionar esa gran cantidad de información. Al cambiar el paradigma en el uso de internet, también ha cambiado el modelo de atacante. Los usuarios maliciosos pueden disponer de herramientas con ataques de un alto nivel de complejidad sin necesitar conocimiento experto para usarlas. El lucro obtenido de dichos ataques puede ser muy elevado y, por ello, se ha incrementado el número de organizaciones criminales y gubernamentales que se dedican exclusivamente a los ataques informáticos. Esto supone un incremento necesario en los gastos en seguridad informática, tanto en personal como en métodos [2].

La seguridad informática tiene que preservar la confidencialidad, la integridad y la disponibilidad de la información del usuario. Para poder evaluar dicha seguridad los sistemas necesitan ser auditados. Una de las más importantes fuentes de información para la auditoría de seguridad son los *logs*, i.e., los registros de actividad [3]. Estos ficheros permiten reconstruir ataques y, además, hacen factible anticiparlos. El principal problema es la variedad de *logs* en formato y en localización, además de la gran cantidad de información o la frecuencia con la que se actualizan [4]. Ahora bien, hoy en día se cuenta con tecnologías que permiten un rápido procesamiento de grandes volúmenes de información. Consecuentemente, se puede concluir que en los tiempos actuales se cuenta con una alta capacidad de cómputo para detectar los ataques [5].

Entre otras aplicaciones de los *logs*, está mejorar la gestión y monitorización de la seguridad en las aplicaciones. Para ello conviene centralizar la información de los *logs*. Estos frecuentemente son ficheros, por lo general en texto plano, que las aplicaciones utilizan para almacenar eventos relativos a la interacción usuario-aplicación. De modo general un *log* guarda en cada línea el instante temporal en el que se realizó una actividad, el usuario que la llevo a cabo y el tipo de actividad efectuada en el marco de una cierta aplicación. Así, se puede extraer de los *logs* información del comportamiento de los atacantes y facilitar su detección [6].

1.1. Objetivos

Este trabajo tiene como objetivo la realización de la primera fase de un proyecto software, llamado *LogExplorer*. Este proyecto busca diseñar e implementar una aplicación web destinada a la gestión centralizada de *logs*. Su funcionalidad permitirá una exploración, procesamiento y visualización de los datos para poder detectar posibles amenazas o ataques en el ámbito de la seguridad informática, y en concreto en su auditoría. Para ello es necesario enfrentarse al hecho de las diferentes codificaciones que tiene cada tipo de *log* y que por tanto es necesario un proceso de captura y *parseado* que transforme la información a un formato común y la guarde en la base de datos para su posterior procesamiento o filtrado. Para ello, el proyecto *LogExplorer* se divide en tres fases:

1. **Primera fase**: es la fase inicial que comprende el estudio y la investigación del problema en cuestión y de las herramientas similares, el análisis de la funcionalidad, el diseño de la aplicación y una implementación inicial que permita comprender más específicamente el problema a desarrollar.

2. **Segunda fase**: es la fase de desarrollo final de la herramienta que comprenderá la implementación lógica y gráfica de la herramienta, así como de la documentación técnica y de usuario.

3. **Tercera fase**: es la fase de validación que comprobará el correcto funcionamiento de la herramienta así como el cumplimiento de los requisitos de usuario. También será la fase de mantenimiento en el caso de que se lleve a producción.

El objetivo de este trabajo de fin de máster es la realización de la primera fase del proyecto de *LogExplorer*. Para el diseño y la implementación, entre las posibles opciones se ha optado por la plataforma *J2EE* mediante el *Spring* [7]. De esta forma a lo largo del presente trabajo fin de máster se ha profundizado en elementos de la seguridad de sistemas, y de modo simultáneo se ha completado el currículo en competencias de desarrollo web ya que tanto la especialización en seguridad como las competencias web son altamente demandadas [8]. También se usa la librería de *Weka* [9] para la implementación de métodos basados en aprendizaje automático y, en la parte gráfica de la aplicación, se ha elegido una plantilla que hace uso de *Bootstrap* [10] llamada *AdminLTE* [11] de *Almsaeed Studio* [12].

El principal problema que presenta una aplicación como esta es el requisito de escalabilidad. Una aplicación dedicada a la seguridad informática requiere una continua actualización. Otro problema es la variedad de formatos de *logs* y la falta de convenciones o estándares. También cabe destacar que se trata de un sistema interactivo y por tanto se necesita capacitar a la herramienta de la suficiente personalización de los métodos usados para garantizar una mayor usabilidad.

Para dar una mayor versatilidad a la herramienta se ha decidido que esta sea en un entorno web, ya que, en muchas ocasiones una red de ordenadores tiene un servidor propio donde guardar los *logs* de múltiples máquinas o dispositivos, (*IDS*, *Routers*, *cortafuegos*...) y el acceso a esa información de forma remota puede suponer beneficio extra a los administradores del sistema informático.

Para explicar este trabajo se ha seguido la siguiente estructura. En el capítulo 2, Estado del Arte, se hace una pequeña introducción a la seguridad informática actual, las herramientas y métodos utilizados, y, al final, se define los *logs* y los ataques en la seguridad informática. En el capítulo 3, Análisis, se realiza un primer análisis del uso de *logs* en la informática forense mediante el desafío de *VAST Challenge 2012 MC2* [13]. Este desafío presenta una situación

simulada del ataque informático a la red de un banco. Se mostrará tanto su descripción como su resultado con el fin de identificar las necesidades a por una herramienta como la que se pretende implementar. Además del estudio de los requisitos de la aplicación a desarrollar. En el capítulo 4, Diseño, se muestra la arquitectura de la herramienta y la especificación del diseño escogido para la misma. En el capítulo 5, Implementación, se expone las herramientas utilizadas para el desarrollo y la codificación utilizada. Por último, en el capítulo 6, Conclusiones, se resume el trabajo aportado y sus beneficios además de un pequeño apartado para el posible trabajo futuro.

Capítulo 2

Estado del arte

Sin lugar a dudas, en el contexto tecnológico actual la seguridad informática es clave tanto en la vida personal de los usuarios como en la mayoría de los negocios [14]. Ésta se define como la protección proporcionada a un sistema de información para alcanzar los objetivos de preservar la integridad, disponibilidad y confidencialidad de los recursos del sistema de información, incluyendo *software*, *hardware*, *firmware*, datos/información y telecomunicaciones [15].

Recientemente el uso de las tecnologías de información se ha disparado. Se debe a un incremento exponencial del número de usuarios como consecuencia de la mejora en la disponibilidad y al mayor acceso a redes inalámbricas públicas y privadas [16]. Este acceso se entiende por el aumento del uso de los nuevos dispositivos móviles y de otros dispositivos inteligentes. Además de la popularización del uso de las redes sociales y de los negocios de venta *online*, cada vez más habituales, que acaban moviendo una gran cantidad de dinero constituyendo un objetivo de interés para los *cibercriminales*.

Este aumento de la disponibilidad, la accesibilidad y la funcionalidad puede tener un efecto negativo en la seguridad, ya que puede incrementar la variedad y forma de los ataques. Hay que destacar que una amenaza no tiene por qué ser un ataque malintencionado. Puede ser un fallo del sistema que suponga un mal funcionamiento del servicio. Resolver estos fallos también puede ser considerado dentro del ámbito de la seguridad informática ya que los atacantes se nutren de los errores y fallos en la programación [17]. El exceso de seguridad puede tener también un efecto negativo al negocio, no permitiendo a usuarios legítimos el acceso. Es necesario un equilibrio entre usabilidad y seguridad [18]. Para explicarlo de una forma gráfica se puede considerar la siguiente cita traducida:

> Es fácil ejecutar un sistema de ordenadores seguro. Solo tienes que desconectar todas las conexiones a Internet y permitir únicamente acceso directo a las terminales, poner la máquina y las terminales en una sala blindada, y poner un guardia en la puerta.
>
> F.T. Gramp and R.H. Morris UNIX Operating System Security [19]

Los ataques directos de usuarios maliciosos tratan de obtener un beneficio ya sea el robo de activos directos (dinero, ventas, servicios...), el robo de información, el control de los ordenadores infectados; y así poder vender su capacidad de procesamiento para fines fraudulentos (envío de *spam*, ataques distribuidos...) y, últimamente, para fines políticos denegando el uso a usuarios legítimos o simplemente por vandalismo [20].

El modelo de estos usuarios maliciosos ha dejado de ser únicamente de personas puntuales con un alto conocimiento informático. Ahora, los ataques se realizan con herramientas muy

sofisticadas que permiten a usuarios con un nivel medio realizar ataques muy complejos. Es el ejemplo de la distribución *Kali Linux* [21], una plataforma que contiene multitud de herramientas para realizar pruebas de penetración. Además de este incremento de la complejidad de los ataques, también se realizan otros de forma coordinada por organizaciones criminales o por agrupaciones de *ciberactivismo* [22].

Todas estas amenazas suponen una necesidad inequívoca de desarrollar nuevas técnicas en la seguridad informática que permitan una mayor protección sin que ello suponga una reducción brusca de la usabilidad y funcionalidad alcanzada. Una de las más estudiadas es la aplicación de la Detección de Anomalías para la detección de ataques en tráfico web y de *malware*.

2.1. Detección de Anomalías

La detección de anomalías se refiere al problema de encontrar patrones en los datos que no se ajustan al comportamiento esperado [23]. Estos datos se suelen nombrar como anomalías, *outliers*, excepciones... También se suele asumir que el comportamiento anómalo es mucho menos frecuente que el comportamiento normal, aunque esto no siempre es así. Por ejemplo, en el tráfico web es posible que un ataque de Denegación de Servicio, *DoS*, genere más paquetes que los generados por el tráfico normal.

Sus aplicaciones y dominios son varios, entre ellos están los siguientes:

- **Sistemas de Detección de Intrusos**: los sistemas de detección de intrusos son una herramienta generalmente opcional de la seguridad informática de un *host* o una red de ordenadores que permite una identificar posibles ataques de usuarios maliciosos [24]. Presuponiendo que los ataques son mucho más infrecuentes que el comportamiento normal, la detección de anomalías puede ser una herramienta muy útil para estos sistemas. En el punto 2.3 se da una explicación más precisa.

- **Detección de Fraude**: la detección del fraude es una aplicación que permite identificar posibles actividades fraudulentas en comercios, bancos, compañías, aseguradoras... Estas actividades pueden costear importantes sumas de dinero a una empresa y la detección de fraude mediante técnicas de detección de anomalías pueden prevenir la mayoría de esas pérdidas. Ejemplos de la detección misma es la detección del fraude en las operaciones de tarjetas de crédito [25, 26], en el uso de teléfonos [27], en las reclamaciones de seguros [28] o en el abuso de información privilegiada [29].

- **Detección de Anomalías Médicas**: la detección de anomalías en el campo médico tiene varias razones, como la detección de errores en la medición de los distintos instrumentos médicos o la detección de anormalidades en la condición de un paciente. Puesto que este dominio es crítico se requiere de un alto nivel de precisión. Existen ejemplo de detección de anomalías en electrocardiogramas [30], en detección de valores atípicos en datos médicos [31] o para detectar brotes de enfermedades [32].

- **Detección de Daño industrial**: la detección de anomalías en este campo puede prevenir fallos en cascada, pérdidas económicas, daño a personas... Se suele dividir en la detección de fallos en unidades mecánicas y la detección de defectos estructurales. La primera tiene ejemplos como los fallos en una caja de cambios [33] o detección de cortocircuitos en turbinas [34]. La segunda tiene ejemplos como la medición de la estabilidad [35, 36] o la salud de los fuselajes de los aviones [37].

- **Detección de anomalías mediante procesamiento de imágenes**: la detección de anomalías en el procesamiento de imágenes puede servir en la detección del movimiento

en un vídeo o en la identificación de regiones de una imagen que sean anormales en esta. Hay ejemplos como en la detección de dígitos en una imagen [38], en la espectrometría [39] o en la vídeo-vigilancia [40].

- **Detección de anomalías en texto**: la detección de anomalías en este campo suele estar relacionado con obtener noticias o nuevos temas en artículos o en colecciones de documentos. también se relaciona con la identificación de los cambios en los documentos que pertenecen a una categoría. Existen ejemplos como la clasificación de artículos [41]
- **Sensores de redes**: al igual que en los sistemas de detección de intrusos, la detección de anomalías en los sensores de redes puede detectar intrusiones o ataques a la red observada. Pero este campo está más enfocado al análisis de datos y a la detección de errores o fallos en la medición. Este campo cuenta con las dificultades de que se trata de un modelo *online* y en tiempo real, por lo que ha de ser rápido en su proceso, además se tiene que proceder de forma distribuida ya que los tipos de datos son muy distintos; continuo, discreto, binario, audio, vídeo... y también puede que estos sensores tengan ruido, dificultando diferenciar entre ruido y anomalía. Este campo no solo analiza datos en internet, también hay casos como en sensores de meteorología [42]. Otros casos se centran en redes inalámbricas [43].

Buscar modelos que distingan entre el comportamiento normal y el comportamiento anómalo no carece de dificultades, siendo las más representativas las siguientes:

- Poder definir un comportamiento normal es difícil. En general se realiza una aproximación. La mayoría de las veces, un comportamiento que se aproxima a lo considerado normal y a lo considerado anomalía está en cualquiera de las dos. Es posible dar un nivel de ambos, por ejemplo un porcentaje.
- En el caso de que las anomalías se refieran a acciones malintencionadas, con el tiempo los atacantes tratarán de dificultar su detección o hacerlas pasar por comportamiento normal.
- Modelar el comportamiento normal o anormal en un momento del tiempo en un sistema que está continuamente evolucionando, puede no ser representativo en un futuro.
- Entre distintos dominios, las anomalías pueden tener un significado muy distinto y por tanto las técnicas tienen que ser específicas para ese dominio.
- Poder etiquetar los datos para el entrenamiento y validación de los modelos es, por lo general, difícil y suele requerir de expertos en el dominio.
- El ruido de los datos puede ser similar a las anomalías reales y por tanto es difícil de eliminar. El ruido puede ser un error en la recepción de los datos, interferencias en la señal...

Para implementar todos los campos anteriormente mencionados y poder subsanar o reducir los problemas, actualmente el enfoque más común en la investigación es el uso de técnicas de minería de datos.

2.2. Minería de datos

La minería de datos es el conjunto de técnicas y algoritmos que permite extraer patrones o estructuras comprensibles de grandes volúmenes de datos. Para el campo de la detección de anomalías, y en especial en el campo de la seguridad informática, la minería de datos permite de forma metodológica extraer información y modelos estadísticos que faciliten el objetivo final:

identificar las anomalías y extraer patrones que detecten posibles ataques sin haberlos tendido con anterioridad.

Dependiendo del problema en cuestión, los resultados y los algoritmos utilizados pueden variar bastante. Aun así se suele realizar una serie de fases comunes [44]:

1. **Obtención de información**: en primer lugar se recoge toda la información posible que pueda ser determinante o simplemente ser relevante para la solución del problema. En el caso de la seguridad informática suelen ser registros de actividad (*logs*), tráfico de red, archivos de configuración, eventos de *antivirus* u otros programas de seguridad...
2. **Auditoría de datos**: en segundo lugar se realiza un estudio preliminar para poder identificar las características y transiciones de la información obtenida y evaluar qué información es realmente relevante y qué información podría llevarnos a una conclusión equivocada. Esta fase se divide en:
   a) **Análisis inicial de los datos**: donde se identifica cada atributo y sus posibles valores. Además de si tiene una distribución especifica, existe ruido en la medición y de qué tipo, tiene cotas como máximos o mínimos definidos, si faltan valores (en inglés, *missing values*)...
   b) **Selección de atributos**: donde se analiza la relevancia de cada atributo para la resolución del problema. También donde se localiza y se eliminan del proceso, si existen, los conocidos falsos predictores, atributos que resuelven de forma engañosa el problema y que pueden llevar a conclusiones erróneas del mismo. En el caso de que la información sea demasiada se recurre a la reducción de dimensionalidad, ya sea eliminando atributos poco relevantes o utilizando algoritmos como el Análisis de Componentes Principales (en inglés, *Principal Component Analysis*, *PCA*).
   c) **Preprocesado**: en algunos casos en vez de tratar con los datos en bruto estos necesitan transformarse para que el proceso de modelado se pueda llevar a cabo. Estas transformaciones pueden ser, por ejemplo; dar un valor a aquellos valores que falten (la media o la moda comúnmente), normalizar los datos para que su media sea igual a cero o para que su desviación típica sea uno, extraer los segundos de una fecha, transformar una enumeración de opciones en un vector numérico...
3. **Modelado**: en la tercera fase se construye el patrón o la estructura de la información analizada previamente. El modelado depende de varios factores aunque se suele clasificar entre:
   a) **Supervisado**: aquellos que necesitan previamente que uno o varios expertos hayan clasificado las instancias de la información en cuestión, es decir hayan aportado un nuevo atributo, la clase, que resuelve el problema. A partir de un subconjunto de la información junto con la clase se desarrolla el modelo. Algunos ejemplos son: Redes neuronales, Redes Bayesianas, Máquinas de vectores de soporte (*Support Vector Machines*, *SVM*), Reglas de asociación...
   b) **No supervisado**: aquellos que no necesitan de esta clase y que dividen la información dependiendo, por ejemplo de una función de distancia. Algunos ejemplos son: Vecino más próximo (kNN k^{th}, *Nearest Neighbor*, que no siempre es no supervisado), Densidad Relativa, *K-means*, árboles de decisión...
4. **Evaluación**: la cuarta fase es donde se mide la eficacia del modelo realizado para poder realizar una comparación veraz con otros posibles modelos. Esta medida se puede realizar con distintas y múltiples métricas y con distintos fines. La más común en estos casos es la *Curva ROC* (*Receiver Operating Characteristic*), una representación gráfica de la eficacia de un clasificador binario.

2.3. Sistemas de Detección de Intrusos - IDS

Los Sistemas de Detección de Intrusos (del inglés *Intrusion Detection System*, *IDS*) principalmente tratan de detectar (y posteriormente, en algunos casos, de actuar) de forma automática ante un posible ataque o intrusión en un sistema informático [24]. Suele constituir como un complemento de seguridad para una aplicación o un servicio en constante ejecución con una función menos agresiva que el *cortafuegos*. Por lo general, se supone que el *IDS* debe evitar los mismos ataques que intercepta y bloquea el *cortafuegos*. El *IDS* posee la ventaja respecto al *cortafuegos* de que su correcto funcionamiento no depende de la implementación de la aplicación en cuestión y, por tanto, no es necesaria una actualización por cada cambio de la aplicación. Uno de los *IDS* más utilizado es el *open-source* y gratuito *Snort* [45].

Los *IDS* nacieron de la necesidad de realizar auditorías internas de los sistemas informáticos para observar el comportamiento de los usuarios y el funcionamiento del sistema [46]. A diferencia de los *cortafuegos*, los *IDS* abarcan una gran variedad de posibles implementaciones y pueden diferir mucho unos de otros. Se suelen dividir en varios tipos [47]:

- **Según su fuente de información**:

  1. **Basado en *host* (*Host-based IDS*, *HIDS*)**: Los *IDS* basados en un solo *host* suelen recoger información de *logs* a nivel del Sistema Operativo. Estos *IDS* asumen que la intrusión dejará un rastro digital en los *logs* aunque se debe de aclarar que es posible que la intrusión sea lo suficientemente inteligente como para borrar su propio rastro en los *logs*.
  2. **Basado en Red (*Network-based IDS*, *NIDS*)**: Los IDS basados en Red suelen recoger información del tráfico de paquetes de la red con un dispositivo del programa que actúa como *sniffer*, es decir, en modo promiscuo.
  3. **Basado en aplicación**: Los *IDS* basados en aplicación pueden recoger la información de *logs* de eventos de la propia aplicación, de análisis del rendimiento, de los recursos de memoria, del acceso a los ficheros o incluso de la propia implementación de la aplicación, buscando para las entradas de datos caracteres extraños o prohibidos [48].
  4. **Basado en el objetivo**: Estos *IDS* generan su propia información mediante la *firma digital* de objetos (ficheros) para detectar la modificación de los mismos y verificar que se cumpla con la política de seguridad, por ejemplo, con la política de permisos y de acceso.

- **Según su tipo de análisis**:

  1. **Detección de uso indebido**: Este es el tipo de análisis más común y más utilizado en los *IDS* comerciales. Se debe a que es el más rápido y sencillo. Pero, como se ha comentado anteriormente, el aumento del uso de las tecnologías de la información, su disponibilidad y la multitud de soportes hardware móviles han posibilitado nuevos tipos ataques y amenazas mucho más sofisticadas. Esto llega a imposibilitar el uso exclusivo de este tipo de análisis ya que se requiere un proceso de actualización continua muy caro en recursos y en tiempo. Este tipo de análisis se puede diferenciar en dos clases:

     a) **Basado en conocimiento**: Que engloba a los *IDS* que son construidos por expertos. Por ejemplo, están los que utilizan análisis de firmas: que se basa en guardar patrones de cadenas sospechosas y en comparar con técnicas de *pattern-matching* cada consulta con esta base de datos para comprobar si se produce o

no un ataque. Entre los *IDS* de este tipo está, por ejemplo el conocido programa *Snort* [49]. También los sistemas expertos que funcionan con reglas de condicionantes, es decir, la regla se activa una vez que cumple todas sus condiciones. Y los sistemas de transición de estados: que funciona con una máquina de estados finitos. Alguno de ellos también buscan en eventos de actividad o de rendimiento y detectan el ataque según un límite predefinido.

   b) **Aprendizaje Automático**: Se basa en, mediante información de ataques y de intrusiones en sistema, utilizar técnicas de minería de datos que proporcionen información o modelos que permitan generar automáticamente patrones, reglas o firmas para detectar posteriormente ataques del mismo tipo. Este tipo de análisis tiene la ventaja de que no supone un coste tan grande como en los anteriores apartados aunque bien, sigue siendo necesario expertos que identifiquen ataques ya producidos.

2. **Detección de anomalías**: Estos sistemas se basan en buscar anomalías en el funcionamiento de la aplicación. Se define como anomalía aquel comportamiento que no está reconocido como normal. Un comportamiento anómalo de un usuario no tiene por que ser un ataque, puede ser un error o una operación no muy usada. Y no todos los ataques son comportamientos anómalos, una de las desventajas de este método es que un usuario con tiempo y paciencia suficientes puede convertir gradualmente un comportamiento anómalo en comportamiento normal. Aun así, analizando el comportamiento general de los usuarios se observa una tendencia regular y, por tanto, se estima que un uso anómalo está muy ligado a ser un ataque. Esto nos permite anticipar un ataque aunque este no se haya dado nunca, es decir, poder detectar nuevos ataques o ataques de día-cero (en inglés *zero-day attack* o *0-day attack*). Estos ataques son aquellos que nunca se habían dado antes pero que es posible que se parezcan o que guarden relación con otros ataques registrados.

   a) **Sistemas expertos**: Al igual que en los *IDS* de detección de uso indebido basados en conocimiento, este tipo de análisis se realiza por expertos que recogen patrones y reglas propias de anomalías y ataques guardados en registros de auditorías del sistema.

   b) **Métodos estadísticos**: Este método funciona detectando desviaciones de modelos estadísticos predefinidos que representarían el comportamiento normal de los usuarios. Suelen funcionar con perfiles de actividad de usuarios en los que se registra actividades en ficheros, programas, servicios...

   c) **Aprendizaje Automático**: Este método es el actualmente más estudiado. Se basa en modelar el comportamiento normal de los usuarios. Para ello se parte de información o bien real o bien autogenerada de la que se extrae reglas, estadísticas o modelos mediante técnicas de minería de datos. Se puede dividir en dos: supervisado y no supervisado. En el supervisado, expertos previamente han etiquetado la información clasificando entre normal y anómalo. El no supervisado se basa en tratar de dividir en grupos la información según sus características para posteriormente identificar cada grupo.

3. **Basado en combinación de ambos**: Utilizar ambos tipos de análisis nos permite recoger las ventajas de ambos pudiendo así detectar nuevos ataques al mismo tiempo que rechazar ataques ya producidos y prevenir ataques prolongados en el tiempo. Este tipo de análisis tiene la desventaja del coste en tiempo de ejecución y en el proceso de actualización.

- **Según su temporización**:

1. **Basado en intervalos (*Batch*)**: Cuando no se tiene la suficiente capacidad o ancho

de banda, se recurre a este método. En cada periodo de tiempo, predefinido con anterioridad, se registra en un fichero las intrusiones detectadas en forma de auditoría.

2. **Basado en tiempo real (*Real-Time*)**: Cada vez que se produce una incidencia, esta es comunicada permitiendo así una respuesta rápida y en algunos casos automática.

- **Según su respuesta**:

1. **Basado en respuesta pasiva**: Esta respuesta solamente trata de avisar, notificar o registrar una intrusión, bien comunicándolo por pantalla, mensajes por teléfono, correo electrónico, *log*, etc. Esta es la respuesta por defecto de los *IDS*. Muchas veces estas notificaciones son desbordantes debido a que es posible que algunas sean falsos positivos y otras sean varias de una sola. En algunos casos se suelen clasificar por riesgo.
2. **Basado en respuesta activa**: Una respuesta activa o respuesta automática es aquella que reacciona ante un supuesto ataque o intrusión con la intención de evitar o reducir los daños, o de averiguar más acerca de la autoría o del método de la intrusión. Las acciones son muy variadas, están por ejemplo cerrar canales de comunicación, denegar el acceso a un usuario, aumentar el tiempo de espera a un servicio, activar un rastreo automático para conocer la procedencia del ataque, etc. Cuando se trata de un *IDS* en red (*NIDS*) se suele seguir un modelo basado en coordinar las respuestas entre distintos *host*. Las respuestas activas pueden suponer un coste muy elevado; en primer lugar debido al alto porcentaje de falsos positivos que puede tener un *IDS* y en segundo lugar debido a que las consecuencias de la respuesta pueden suponer un daño mayor que el daño causado por el ataque. Por ello, muchos de los *IDS* que permiten una respuesta activa dejan decidir al usuario si desea o no dichas respuestas.

Con el paso del tiempo, los *IDS* convencionales se han quedado cortos en satisfacer las necesidades de los actuales sistemas en seguridad. Eso ha derivado en el surgimiento de los *SIEM*, sistemas que tiene un objetivo más amplio y generalista en su análisis.

2.4. Gestión de Información y de Eventos de Seguridad - *SIEM*

Un sistema *SIEM* es una combinación entre la gestión de información de seguridad (*Security Information Management*, *SIM*) y la gestión de eventos de seguridad (*Security Event Management*, *SEM*) [50]. La primera se centra en el análisis de los datos de *logs* y del almacenamiento a largo plazo. El segundo se centra en la monitorización y las notificaciones. Se unen también al *SIEM*, el análisis en tiempo real y el análisis de correlación de los datos. Otro aspecto que se añade a algunos sistemas, es el análisis del uso proporcionando, la oportunidad de mejorar u optimizar el sistema monitorizado, de hay que algunos pasen a ser *SIEOM* (O de "*opportunity*").

La mayoría de *SIEM* no solo pueden recoger, analizar o priorizar eventos de seguridad dentro de una red de ordenadores, también facilitan la labor a la hora de configurar alarmas y notificaciones de seguridad de forma automática, pudiendo ser del tipo alerta ante hechos concretos o simplemente un informe periódico que resuma la situación de la red.

Otras capacidades pueden ser mostrar la información en gráficas o en tablas informativas para ayudar a los administradores a localizar patrones o identificar más fácilmente una actividad determinada. Estos sistemas cumplen ciertas competencias en la seguridad informática que ayudan a la comprensión, el análisis y la monitorización de sistemas críticos. También son usados en el análisis forense ya que aportan funcionalidades para una búsqueda más exacta

y proporcionan un sistema para guardar relaciones entre distintos dispositivos o datos. Los *SIEM* también ayudan a implantar acuerdos y políticas de seguridad, y permiten evaluar el cumplimiento de las normativas y estándares de seguridad.

2.5. Registros de actividad - *Logs*

Existen distintas definiciones de *log*. No existe una única convención que determine qué información debe reflejar un *log*. Idealmente los *logs* no solamente tienen que identificar los problemas del sistema, también deberían poder ser usados para cuantificar la calidad del funcionamiento del sistema o detectar un posible fallo futuro [3].

En primer lugar, se define un evento como un acontecimiento en un entorno que, por lo general, implica un intento de cambio de estado. Un evento suele incluir una noción del tiempo, el acontecimiento y cualquier detalle que, perteneciendo explícitamente al caso bajo consideración o a su entorno, puedan ayudar a explicar o entender las causas o los efectos del evento [51]. Y, por tanto, se define el *log* como una colección de registros de estos eventos.

Los *log* se pueden dividir en cuatro categorías según su formato [52]:

1. **Semi-estructurado**: Los *logs* que pertenecen a esta categoría son aquellos cuyas entradas no tienen una distinción clara entre los valores de cada campo, los delimitadores de los campos y los datos de ruido. Un ejemplo serían los típicos *logs* de *Linux* como *Syslog*.

2. **Débilmente estructurado**: Los *logs* que pertenecen a esta categoría son aquellos que necesitan información adicional para entender sus campos, por ejemplo, los basados en el formato *CSV*.

3. **Fuertemente estructurado o completamente estructurado**: Los *logs* que pertenecen a esta categoría son aquellos en los que está guardada toda la información en la estructura del formato, es decir; tipo de dato, nombre del campo, y valor del dato. Algunos posibles formatos pueden son los basados en *XML* y *JSON* entre otros.

4. **Fuertemente estructurado basado en un perfil de validación**: Los *logs* que pertenecen a esta categoría son aquellos que siendo fuertemente estructurados se proporciona una forma de comprobar los nombres de los campos, los valores de los datos y la semántica usada. Esto sirve para enriquecer los *logs* fuertemente estructurados y evitar problemas como por ejemplo:

   - **Nombres de campo sin definir**: es decir, tener nombres como "*ip*", "*ip-address*" o "*ipaddress*".
   - **Formato sin definir**: es decir, tener un formato llamado "*date*" (traducido, fecha) pero no concretar si se define mediante el estándar *rfc3164* (Enero 7 13:17:53) o mediante el estándar *rfc3339/ISO8601* (2015-01-07T13:17:53.24+02:00).
   - **Presencia de distintos tipos de datos para un formato dado**: es decir, cuando se tiene en formato "*host*" y si en este puede contenerse los datos del tipo direcciones *IP* o también los nombres del *host* ("*hostname*") o ambos.

2.6. Ataques en Seguridad Informática

Los *logs* pueden recoger, ya sea de forma expresa o implícita, ataques o amenazas de seguridad. Como ya se ha explicado anteriormente, la seguridad de un sistema de información se

define como la protección que permite preservar una serie de propiedades. Suponiendo que en un sistema de información existe un flujo de información de una fuente a un destino a través de un canal de comunicación, comúnmente se suelen clasificar estas propiedades como [53, 54, 55]:

- **Confidencialidad**: esta propiedad es la que preserva la información para no ser revelada a personas no autorizadas. Además del contenido, la confidencialidad debe resistir los análisis de tráfico que puedan ser realizados en el flujo de información y debe garantizar la privacidad, es decir, el control de cada usuario a decidir cómo, por dónde, por quién y a quién puede ser revelada su información.
- **Integridad**: esta propiedad protege a la información transmitida contra modificaciones o contra su destrucción no autorizada. Además de proteger la información, la integridad también debe asegurar que el sistema que está siendo ejecutado utiliza la funcionalidad prevista, sin cambios no autorizados.
- **Disponibilidad**: esta propiedad asegura que la información pueda ser accedida de forma y en tiempo confiable. Es decir, un sistema disponible es aquel cuyos recursos están siempre listos para ser usados sin demoras y nunca negar el acceso a usuarios autorizados.
- **Autenticación**: esta propiedad se preocupa de asegurar que la información es auténtica. Es decir, que el sistema ejecutado asegura al destinatario de que la información transmitida es de la fuente que dice ser.
- **No repudio**: esta propiedad asegurar evitar que la fuente o el destinatario puedan negar que un mensaje ha sido transmitido. Es decir; una vez un mensaje se ha transferido, la fuente del mismo puede probar que ha sido recibido por el destinatario, del mismo modo que el destinatario puede probar que el mensaje ha sido enviado por la fuente.

Por tanto los ataques en un sistema de información se pueden clasificar de la siguiente manera [56]:

1. **Interrupción**: este tipo de ataque se da cuando un recurso del sistema se destruye o no está disponible. Es decir evita que la información llegue a su objetivo previsto. Un ejemplo son los ataques de denegación de servicio (*DoS*, del inglés *Denial of Service*).
2. **Interceptación**: este tipo de ataque se da cuando un tercer usuario no autorizado puede acceder a la información mediante escuchas en el canal de comunicación.
3. **Modificación**: este tipo de ataque se da cuando la información no sólo se intercepta, sino que se modifica por una parte no autorizada durante el tránsito desde el origen al destino.
4. **Fabricación**: este tipo de ataque se da cuando un atacante inserta información falsificada en el sistema sin que emisor haga nada. Cuando se inserta un objeto previamente interceptado, se denomina "*replaying*", reproducir. Cuando el atacante se hace pasar por la fuente legítima e inserta su información deseada, el ataque se llama "*masquerading*", enmascaramiento.

Para mostrar de forma más técnica los posibles ataques en la seguridad informática, se observan los diez ataques más frecuentes en el caso particular de las aplicaciones web del año 2013 según la organización de seguridad informática *OWASP* [57] (*The Open Web Application Security Project*):

1. **Injection**: en español, inyección. Se produce cuando el atacante envía datos hostiles para engañar al intérprete del sistema para que ejecute comandos no deseados o acceder a los datos sin la debida autorización.

2. **Broken Authentication and Session Management**: en español, autenticación y gestión de la sesión rota. Se produce cuando el atacante puede comprometer contraseñas o identificadores y así poder asumir las identidades de otro usuario.

3. **Cross-Site Scripting (XSS)**: se produce cuando la aplicación toma datos no fiables que envía a un navegador web sin validar adecuadamente. Permite a los atacantes ejecutar *scripts* en el navegador de la víctima con los que pueden utilizar sus sesiones, modificar los sitios web o redirigirlos a páginas maliciosas.

4. **Insecure Direct Object References**: en español, referencias inseguras a objetos directos. Se refiere a cuando un desarrollador expone una implementación interna permitiendo a un atacante acceder a datos no autorizados saltándose el debido control de acceso.

5. **Security Misconfiguration**: en español, mala configuración de seguridad. Se produce cuando el atacante se aprovecha de una aplicación que no tiene o no mantiene la requerida seguridad al día.

6. **Sensitive Data Exposure**: en español, exposición de datos sensibles. Se produce cuando un atacante pueden sustraer o modificar información débilmente protegida para llevar a cabo delitos como fraudes de tarjetas de crédito o robo de identidad.

7. **Missing Function Level Access Control**: en español, falta de función de nivel de control de acceso. Se da cuando un atacante es capaz de crear solicitudes con el fin de acceder a funcionalidades sin la necesaria autorización debido a que el sistema no valida correctamente el nivel de acceso.

8. **Cross-Site Request Forgery (CSRF)**: este ataque obliga al navegador de la víctima a enviar una solicitud incluyendo en ella la información de autenticación de la víctima a una aplicación vulnerable para que piense que son solicitudes legítimas de la víctima.

9. **Using Components with Known Vulnerabilities**: en español, uso de componentes con vulnerabilidades conocidas. Algunas bibliotecas o *frameworks* pueden utilizar plenos privilegios. Por lo que los atacantes aprovechan vulnerabilidades de estos componentes para hacerse con el control del sistema completo.

10. **Unvalidated Redirects and Forwards**: en español, redireccionamientos y reenvíos sin validar. Se produce cuando una aplicación usa datos sin validar para determinar una reedirección o un reenvío. Esto permite a un atacante redirigir a los usuarios a páginas que no son de confianza.

Capítulo 3

Análisis

En este capítulo se estudian las necesidades y requisitos que tendrá que satisfacer la aplicación a desarrollar. Para ello, este capítulo se divide en una primera parte de descripción general de la aplicación, una segunda parte en el que se muestra un ejemplo de uso de *logs* con el desafío de *VAST Challenge 2012 MC2* [13], una tercera parte en el que se realiza un análisis competitivo de aplicaciones semejantes, una cuarta parte dedicada a definir los roles de usuario, una quinta parte de educción de requisitos, tanto funcionales como no funcionales y una sexta y última parte dedicada a los casos de uso.

3.1. Descripción general

En la seguridad informática, sobre todo en el campo de la informática forense, los registros de actividad o *logs* son tremendamente útiles para obtener las causas, los daños y los culpables de un ataque informático ocurrido en un ordenador o una red de ordenadores. Por ello, una aplicación que aproveche dicha información y que permita procesarla y presentarla de una forma fácil y ágil, es una gran ventaja para usarse en esta rama de la informática [58, 59]. *LogExplorer* será una aplicación web para la centralización de *logs*, es decir, de registros de actividad, que permita a cualquier usuario, guardar, procesar y mostrar de forma gráfica, información de *logs* de distinto tipo. Esta aplicación hará uso del *framework* J2EE *Spring* [7] para agilizar y facilitar el desarrollo de la herramienta, además de ganar aprendizaje y experiencia con dicho *framework*.

Los datos de los *logs* en la aplicación se pueden someter a las siguientes operaciones según la preferencia del usuario:

1. ***Parseado***: el *parseado* es un análisis sintáctico que permitirá al sistema traducir la información de cada *log* a un formato que el sistema pueda entender y procesar.

2. **Almacenamiento**: es decir, el emplazamiento en un sistema de archivos de cada *log*.

3. **Filtrado**: un filtro es cualquier tipo de condición elegida por el usuario para no mostrar la información que no considere relevante.

4. **Procesamiento**: cada procesamiento son aquellas transformaciones que se realicen a los datos, ya sea para extraer información o para deducir información nueva a partir de esta.

5. **Visualización**: la visualización es la representación gráfica de los datos para poder observar de forma rápida una gran cantidad de datos al mismo tiempo y poder así llevar a cabo decisiones más fácilmente [60, pp. 18, 193].

En la figura 3.1 se muestra de modo visual la transformación de los datos *logs*, de acuerdo con un diagrama simplificado de etapas por las que pasan los datos de en la aplicación según la configuración de un usuario.

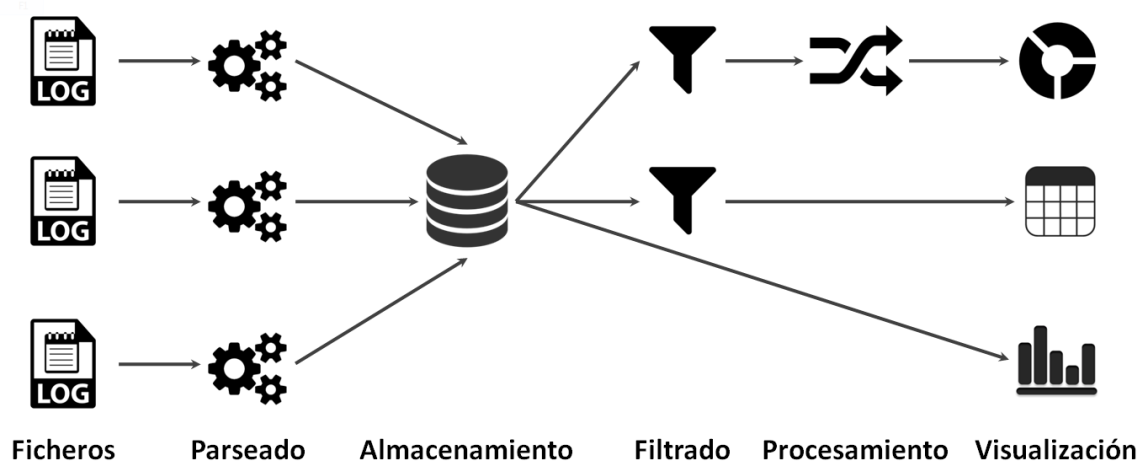

Figura 3.1: Diagrama de etapas de datos.

3.2. Ejemplo de uso de *logs*

En este apartado se mostrará un ejemplo en el que se usan registros de actividad, *logs*, en la auditoría de datos. Este ejemplo usa los datos proporcionados por el *VAST Challenge 2012 MC2* [13]. Este desafío aporta tanto el conjunto de datos de los *logs*, como información contextual y la solución al desafío. Este apartado no trata de resolver el objetivo real del desafío, desarrollar nuevas herramientas gráficas de análisis para grandes volúmenes de datos. A lo largo de este apartado se analizarán diversos procedimientos para emplear la información contenida en los *logs* de cara a esclarecer una situación compleja. Todo ello con objetivo de identificar las necesidades que tiene que cubrir la herramienta a desarrollar.

Para ello, en la primera parte se presentará una pequeña introducción del problema planteado, la segunda parte se enumerarán los objetivos definidos por el desafío, en la tercera parte se describirán las características principales de los datos aportados, en la cuarta parte se mostrarán las soluciones a los objetivos dadas por el desafío para comprender la importancia del análisis de *logs*, y, por último, se destacarán los tres trabajos que alcanzaron mejores resultados en este desafío.

3.2.1. Introducción al *VAST Challenge 2012 MC2*

El desafío utilizado en este capítulo de pruebas se llama *VAST Challenge 2012 MC2*. Este desafío está orientado al tema de la informática forense, tratando de esclarecer una situación simulada de una empresa simulada. La empresa se trata de un banco internacional llamado *Banck Of Money*. Una región dentro de *Banck Of Money* está experimentando una serie de dificultades operativas, vinculadas a la seguridad informática de su sistema.

Para esclarecer los motivos de dichas dificultades y así poder solucionar el problema, se proporcionan los registros del *cortafuegos* y del *IDS* de la red del banco de aproximadamente 5.000 ordenadores, que se describirán posteriormente. También se proporciona una descripción

de la red de la empresa y el contexto general de la situación. La arquitectura de la red se puede ver en la figura 3.2.

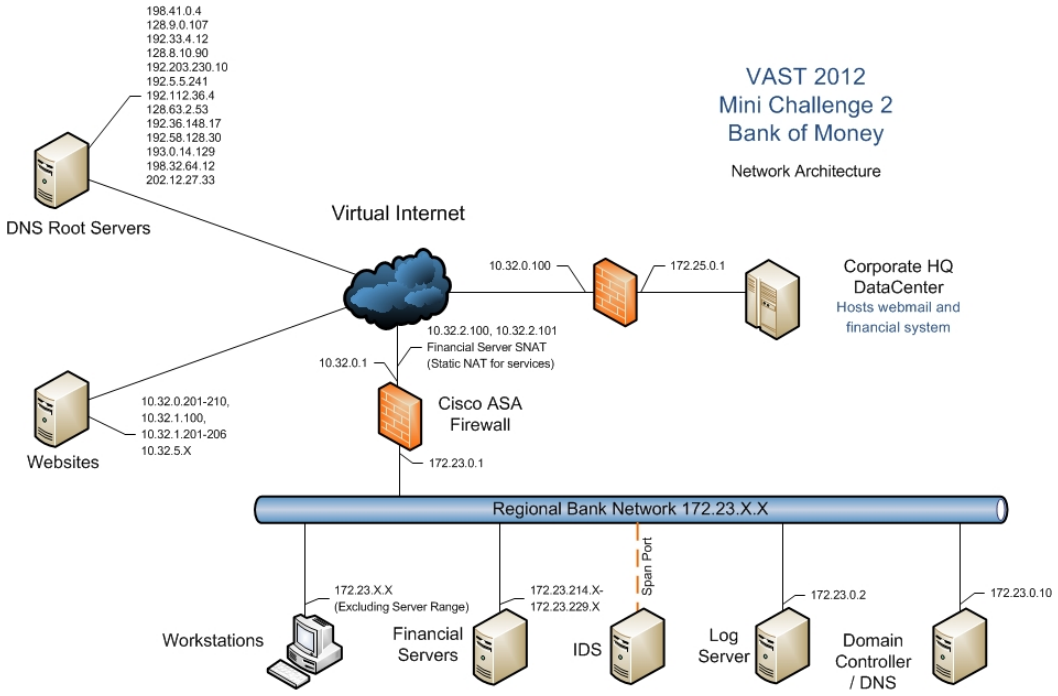

Figura 3.2: Arquitectura de la red de *Bank of Money* del *VAST Challenge 2012 MC2*.

La causa de dichas dificultades dadas por la solución del desafío, es la infección en la red de ordenadores de una *botnet*. Una *botnet* es, por lo general, un conjunto de programas malignos que tiene dos partes diferenciadas, los cliente o *bots* y el servidor que los controla, el servidor *Command & Control*. Aquí se hablará de *botnet* a cada instancia de los *bots* y del servidor que los controla será el servidor C&C. Esta *botnet* tiene por objetivo la *exfiltración* de información de la empresa y la propagación a otros ordenadores de la red. La *exfiltración* es el proceso de envío de datos sensibles a un tercero, en este caso el servidor C&C, que no tienen acceso a ellos. En la solución se describirá más claramente el funcionamiento de ambas partes.

3.2.2. Objetivos

El objetivo real del desafío *VAST Challenge 2012 MC2* es proporcionar una situación parecida a la realidad para que investigadores en seguridad informática desarrollen nuevos métodos de representación gráfica y de detección de ataques con grandes volúmenes de datos. Al final, los métodos deben responder a las siguientes cuestiones:

- **MC 2.1**: Con herramientas de análisis visual, identifica eventos destacables que tuvieron lugar durante el período de tiempo cubierto en los registros del *cortafuegos* e *IDS*. Proporcionar capturas de pantalla de las herramientas analíticas visuales que resalten los cinco eventos más destacados del problema de seguridad, junto con explicaciones de cada evento.
- **MC 2.2**: Mostrar la tendencia de seguridad de los registros del *cortafuegos* e *IDS* en el transcurso de los dos días incluidos.
- **MC 2.3**: ¿Qué sospecha usted que es la causa de los eventos identificados en el primer objetivo? Entendiendo que no se puede apagar la red corporativa o desconectarlo de la

Internet, ¿qué acciones pueden tomar los administradores de red para mitigar la causa principal del problema?

3.2.3. Características de los datos

Se proporcionan dos tipos de ficheros de registro (*log*): registro de la actividad del *cortafuegos* y registro de incidencias detectadas por el *IDS*, que en este caso se trata de *Snort* [45]. Cada *log* se divide en dos partes, siendo la primera parte el primer día y la segunda parte el segundo día. En la tabla 3.1 se muestran los atributos y su descripción de los ficheros de *log* del *cortafuegos*. Y en la tabla 3.2 se muestran los atributos y su descripción de los ficheros de *log* del *IDS*.

Atributo	**Descripción**
Date/Time	Fecha y tiempo de la captura de la actividad.
Logging Device	Dirección *IP* del *log* del Cortafuegos.
Syslog priority	Prioridad del mensaje del *log*.
Operation	Tipo de actividad capturada.
Message code	Código asociado con el mensaje.
Protocol	Tipo de protocolo de conexión.
Source *IP*	Dirección *IP* de origen relacionada con la actividad. Este campo puede estar vacío para algunos mensajes del *log*.
Source *MAC* Address	Dirección *MAC* asignada a la máquina asociada con la actividad.
Destination *IP*	Dirección *IP* de destino relacionada con la actividad. Este campo puede estar vacío para algunos mensajes del *log*.
Source port	Puerto asociado con la *IP* de origen en esta actividad. Este campo puede estar vacío para algunos mensajes del *log*.
Destination port	Puerto asociado con la *IP* de destino en esta actividad. Este campo puede estar vacío para algunos mensajes del *log*.
Source side	Con valor "*inside*" cuando el origen sea interno u "*outside*" cuando sea externo.
Destination side	Con valor "*inside*" cuando el destino sea interno u "*outside*" cuando sea externo.
Destination service	Nombre del servicio asociado con el puerto de destino.
Interface	Este campo está vacío en este conjunto de datos.
Direction	Con valor "*inbound*" cuando sea dirección de entrada o "*outbound*" cuando sea dirección de salida.

Tabla 3.1: Atributos del *log* del *cortafuegos* de *VAST Challenge 2012*

Atributo	**Descripción**
time	Fecha y tiempo de la captura de la actividad.
sourceIP	Dirección *IP* de origen.
sourcePort	Puerto de origen.
destIP	Dirección *IP* de destino.
destPort	Puerto de destino.
classification	Clasificación de la alerta del *IDS*.
priority	Prioridad de la alerta del *IDS*.
label	Texto de la regla violada específica del *IDS*.
packet info	Contiene información de nivel de paquete del *IDS*.
packet info cont'd	Contiene información de nivel de paquete del *IDS*.
xref	Algunos registros contienen un bloque de referencia.

Tabla 3.2: Atributos del *log* del *IDS* de *VAST Challenge 2012*

El *IDS* tiene un comportamiento bastante básico, se trata de un *IDS* que detecta ciertos tipos de actividad según un conjunto de reglas definidas. Al final el *IDS* puede detectar:

1. Si un ordenador está realizando un escaneado de puertos a la red.

2. Si ha empezado una sesión *FTP*.

3. Si ha comenzado una sesión *IRC*.

El *cortafuegos* registra el tráfico que pasa desde la interfaz externa a la interfaz interna, o que pasa de la interfaz interna a la interfaz externa. Es decir, por ejemplo se registra navegación web, actualizaciones de los registros financieros en el servidor corporativo de la sede, la actividad *botnet*, o el *chat IRC*. El *cortafuegos* no registra los movimientos entre dispositivos de la misma interfaz. Por ejemplo, el tráfico entre el servidor dentro de la oficina de la sede regional y una estación de trabajo en la oficina de la sede regional. Ambos dispositivos residen en la red interna y el tráfico no atraviesa la interfaz exterior.

Hay que tener en cuenta la cantidad abrumadora de líneas en cada *log*. Uno de los puntos más importantes de este desafío es la capacidad de procesar grandes cantidades de información. En la tabla 3.3 se puede observar la cantidad de lineas por fichero *log*.

Fichero	**Líneas**
Cortafuegos día 1	13208233.
Cortafuegos día 2	10503108.
IDS día 1	32643
IDS día 2	18430

Tabla 3.3: Número de líneas de los ficheros de *log* de *Cortafuegos* e *IDS*

3.2.4. Solución

Para mostrar la solución, esta se dividirá en cada pregunta del desafío.

MC 2.1: Eventos destacables

Los eventos destacables son aquellos que detectan el comportamiento de la *botnet*. Dicho comportamiento se divide en dos ya que la *botnet* depende también de los servidores C&C.

Los servidores C&C son un número total de 10 y su comportamiento es el siguiente:

1. Se ponen en línea para comunicarse con los clientes *botnet*. Permanecen a la espera hasta que una conexión de una *botnet* se hace de un ordenador.
2. El servidor C&C pide a la *botnet* que determine el tipo de ordenador infectado, consultando el hardware de la computadora.
3. En el caso de que el servidor C&C es detectado por la *botnet*, se envían instrucciones adicionales a esta por el servidor C&C para buscar datos sensibles.
4. Si se detectan datos sensibles en el servidor, el servidor C&C indica a la *botnet* que intente realizar *exfiltración* los datos utilizando los protocolos *FTP* o *SSH*.
5. Si otro ordenador es detectado por la *botnet*, el servidor C&C le indica que trate de replicar por escaneo de puertos a otros equipos de la red que son vulnerables a una particular vulnerabilidad.

Después, el comportamiento de la *botnet* es el siguiente:

1. Después de que la *botnet* esté instalada en un ordenador, esta intenta ponerse en contacto con uno de los 10 de los servidores C&C.
2. Si se conecta, la *botnet* pide al servidor C&C su primera instrucción.
3. Esta instrucción es determinar qué tipo de ordenador que se ha infectado. La *botnet* envía esta información al servidor C&C.
4. La *botnet* es instruida por el servidor C&C para buscar datos confidenciales, como una base de datos o una unidad de red compartida.
5. Si se detecta dicha información sensible, la *botnet* envía los datos que se ha encontrado al servidor C&C.
6. Si el servidor C&C quiere los archivos detectados por la *botnet*, esta es instruida para realizar exfiltración de los datos a cualquier servidor C&C a través de *FTP*.
7. Si la conexión *FTP* falla, la *botnet* espera una duración aleatoria y luego intenta enviarlo por *SSH*.
8. Si otro ordenador es detectado por la *botnet*, esta es instruida por el servidor C&C para encontrar otros equipos de la red para infectar.

MC 2.2: Tendencia

La tendencia se recoge en la linea temporal descrita en la tabla 3.4.

Tiempo	**Evento**
04/05/2012 17:51	El fichero comienza y hay tráfico normal. Este incluye comunicación entre las sedes regionales ordenadores y el servidor financiero de la sede corporativa, navegación web del usuario, y *chat* por *IRC*.
04/05/2012 20:25	Un equipo de limpieza inserta un *USB* en la estación de trabajo de un oficial de préstamo que infecta el ordenador con una *botnet* (no visible en los datos). Esta se replica a otras máquinas de la red. Escanea puertos para detectar vulnerabilidades y se comunicará con uno de los 10 servidores de C&C.
04/05/2012 20:26	La *botnet* infecta otras máquinas y localiza datos sensibles. Intentando su *exfiltración* mediante *FTP*.
04/05/2012 22:16	La *botnet* comienza a recibir instrucción de los servidores C&C que mostrarán anuncios en los ordenadores infectados.
04/05/2012 22:21	El tráfico *FTP* de la *botnet* es bloqueado por el Cortafuegos. Se crean numerosos eventos de denegación y se envían al servidor de registro.
-	La *botnet* que tienen datos para su *exfiltración* son instruidos por un servidor C&C para usar *SSH* en lugar de *FTP*. Funciona y los datos se envían fuera de la red.
04/05/2012 22:40	El departamento de informática es alertado por una gran cantidad de conexiones *FTP* fallidas y se empieza a solucionar el problema (no detectable en los datos). Se parchean y se reinician algunas estaciones de trabajo infectadas para tratar de deshacerse de la *botnet*. Algo del tráfico de la *botnet* ya no es visible, pero la mayoría está presente.
04/05/2012 22:41	Debido a todo el tráfico de la *botnet*, la red se hace más lenta y los usuarios empiezan a notar que sus estaciones de trabajo funcionan más lento de lo habitual (comportamiento de los usuarios no es detectable en los datos).
04/06/2012 17:21	Final del primer fichero.
04/06/2012 17:40	Inicio del segundo fichero.
04/06/2012 17:41	En este momento, el 10 % de las máquinas están infectadas. 5 % de las máquinas infectadas se comunican con los servidores de publicidad (*adware*), y los ordenadores se reinician varias veces. Los datos muestran cierta caída en la *exfiltración*, pero esta no se detiene.
-	El departamento de informática de *The Bank of Money* es informado por un empleado en la sede regional de que su máquina está mostrando una cantidad excesiva de ventanas emergentes con anuncios. El empleado toma nota de esto no ocurrió el día anterior. El empleado informa al departamento de informática que primero trataron de reiniciar su ordenador varias veces antes de llamar. El departamento de informática no estaba al tanto de que la *botnet* se había propagado tan rápido y sigue examinando la cuestión. (No detectable en los datos - explicación escenario)
4/6/2012 18:11	En este momento, el 15 % de las máquinas están infectadas y la *exfiltración* continúa. Las nuevas *botnet* intentan realizar *exfiltración* por *FTP*.

Tabla 3.4: Linea temporal del *VAST Challenge 2012 MC2*

MC 2.3: Causas principales y posibles soluciones

La causa principal de la situación presentada por el desafío es evidentemente la *botnet*, ya que es ella la que satura las estaciones de trabajo y realiza la *exfiltración* de información confidencial. Las posibles soluciones a dicha situación son varias; se podría prohibir el acceso en el *cortafuegos* a las direcciones *IP* de los servidores C&C. Controlar el flujo de información por protocolo *SSH* al menos mitigaría parte de la *exfiltración* de información.

3.2.5. Ganadores del desafío

A continuación se muestran los tres mejores trabajos sobre este desafío:

1. "**BANKSAFE: A Visual Situational Awareness Tool for Large-Scale Computer Networks** [61]": que muestra el desarrollo de una herramienta escalable, distribuida y web. Con visualizaciones de mapas de regiones y mapas de reloj.
2. "**Dynamic Analysis of Large Datasets with Animated and Correlated Views** [62]": que nos muestra una herramienta de análisis visual acelerada mediante GPU.
3. "**Investigating Network Traffic Through Compressed Graph Visualization** [63]": que muestra un análisis visual mediante grafos de gran tamaño.

3.3. Análisis competitivo

Este análisis competitivo se ha realizado para poder identificar funcionalidades y requisitos recomendables, así como ventajas que pueden llevarse a cabo en la herramienta a desarrollar. Para ello se han comparado cuatro sistemas diferentes resaltando sus puntos positivos, sus puntos negativos y aquellos puntos de interés que se podrían contemplar para el proyecto:

1. elastic

   - **Nombre**: *ELK = ElasticSearch + Logstash + Kibana*
   - **Enlace**: elastic.co
   - **Aspectos positivos**:
     a) Todo unido: colector, indexador/buscador, y visualizador gráfico.
     b) Interfaz web.
     c) Se puede actualizar para poder usar *Hadoop* y otro tipo de fuentes de datos.
     d) Puede relacionar tipos distintos de datos y detectarlos automáticamente.
   - **Aspectos negativos**:
     a) Sin usuarios.
     b) Tres partes diferenciadas, difícil de configurar.
     c) No tiene aprendizaje automático.
   - **Aspectos de interés**:
     a) Los filtros y los paneles de gráficos son muy personalizables y usables.
     b) La interfaz es muy flexible.

2. 

   - **Nombre**: Apache Drill
   - **Enlace**: drill.apache.org
   - **Aspectos positivos**:
     a) Distintos tipos de formato para los ficheros *log* (*Avro*, JSON, CSV...).
     b) Transforma los datos en Modelo Relacional.
     c) Ejecuta consultas *SQL*.
     d) No-Relacional y *Big Data*.
   - **Aspectos negativos**:
     a) Solamente es recolector de datos. La visualización o el aprendizaje automático mediante otros software.
   - **Aspectos de interés**:
     a) Las consultas en *SQL* pueden ser flexibles en el análisis y en la relación entre distintos datos.

3. 

   - **Nombre**: AlienVault Unified Security Management
   - **Enlace**: alienvault.com
   - **Aspectos positivos**:
     a) Todo unido: colector, indexador/buscador, y visualizador gráfico.
     b) Encuentra dispositivos automáticamente, *host* y sus vulnerabilidades.
     c) Permite crear alarmas.
     d) Permite crear reportes.
     e) Permite crear reglas de *Snort*.
     f) Detección automática de ataques y anomalías.
     g) *SIEM* y *Open Source Security Information Management* (*OSSIM*).
   - **Aspectos negativos**:
     a) Funciona en una máquina virtual.
     b) No es muy personalizable.
     c) No funciona en *Big Data*.
   - **Aspectos de interés**:
     a) La detección de dispositivos y vulnerabilidades es muy recomendable para un análisis más específico.
     b) También recoge datos de componentes del ordenador (*CPU*, *Memoria*, *Tarjeta gráfica*...).
     c) Permite la geolocalización de direcciones *IP* y se puede mostrar en mapas o utilizando las banderas de los países en las tablas.

4. splunk>

   - **Nombre**: Splunk
   - **Enlace**: splunk.com
   - **Aspectos positivos**:
     a) Todo unido: colector, indexador/buscador, y visualizador gráfico.
     b) *Big Data*.
     c) Encuentra dispositivos automáticamente, *host* y sus vulnerabilidades.
     d) Identificación y edición de usuarios registrados, *logins* y gestión de sus accesos, privilegios y comportamiento.
     e) Posibilidad de recoger datos de tráfico web.
     f) Se pueden editar los eventos y enriquecerlos con comentarios.
     g) Detección automática de intrusiones y ataques *malware*.
     h) Búsqueda de acceso y eventos según un usuario o dirección *IP*.
   - **Aspectos negativos**:
     a) Difícil de configurar.
     b) Funciona por máquina virtual.
   - **Aspectos de interés**:
     a) Análisis por accesos de usuarios facilita la detección de accesos no autorizados.
     b) Análisis directamente por tráfico web, no solo por *logs*.

3.4. Roles de usuarios

Con objetivo de identificar los agentes a los que se hace referencia en los posteriores requisitos de usuario, este apartado muestra los diferentes tipos de usuario que diferencia la aplicación:

- **Usuario no registrado**: el usuario no registrado es aquel que accede por primera vez al sistema. Puesto que no está registrado solo puede acceder al *login* y a la página de registro, el resto está restringido a los usuarios registrados.

- **Usuario registrado**: el usuario registrado es aquel que ha sido dado de alta en el sistema guardando su información en la base de datos de la aplicación mediante la página de registro. Para utilizar el sistema, el usuario registrado tiene que acceder mediante la página de *login* insertando su *email* y contraseña. En este grupo de usuarios existen dos divisiones según sus roles o privilegios:

  - **Usuario final**: el usuario final solamente tendrá acceso a su propia información negando por tanto la visualización, la modificación y/o la eliminación de aquello que este usuario no haya creado o no sea propietario.
  - **Administrador**: el administrador tiene acceso a toda la información del sistema y a todas las acciones que tendría cualquier usuario final sin necesariamente ser el propietario. Además tendrá acceso a las cuentas de los otros usuarios y podrá prohibir el login de cualquier cuenta o eliminarla por completo.

3.5. Requisitos

Los requisitos se dividirán en funcionales "RF" y no funcionales "RNF" añadiendo un número identificativo, y seguidos por su nombre y una descripción del mismo. En el caso de los funcionales se añadirán los roles de usuarios que se ven afectados a este requisito.

3.5.1. Requisitos funcionales

Los requisitos funcionales son aquellos que definen una función de la aplicación web, es decir, una operación en los datos o un comportamiento específico del sistema que puede utilizar el usuario. Por ello cada requisito tendrá su representación en los casos de uso. Los requisitos funcionales son los siguientes:

Requisito funcional 1	
ID:	RF01
Nombre:	Registrarse en el sistema
Rol de usuario:	Usuarios no registrados
Descripción	
Cualquier usuario no registrado puede darse de alta en el sistema. Para ello aportará su información personal a la aplicación en la página de registro.	

Tabla 3.5: Requisito funcional 1 - Registrarse en el sistema

Requisito funcional 2	
ID:	RF02
Nombre:	Entrar en el sistema
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado puede entrar en el sistema y tener una sesión propia e individual.	

Tabla 3.6: Requisito funcional 2 - Entrar en el sistema

Requisito funcional 3	
ID:	RF03
Nombre:	Desconectarse del sistema
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede cerrar su sesión de forma segura.	

Tabla 3.7: Requisito funcional 3 - Desconectarse del sistema

Requisito funcional 4	
ID:	RF04
Nombre:	Crear un nuevo formato de *log*
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede crear un nuevo formato que recoja la información para poder leer un *log* subido posteriormente. Este formato se refiere a la información del tipo de fichero, el nombre de las columnas, el tipo de dato de cada columna...	

Tabla 3.8: Requisito funcional 4 - Crear un nuevo formato

Requisito funcional 5	
ID:	RF05
Nombre:	Eliminar un formato creado
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede eliminar un formato creado por él.	

Tabla 3.9: Requisito funcional 5 - Eliminar un formato creado

Requisito funcional 6	
ID:	RF06
Nombre:	Listar todos los formatos creados
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede listar todos los formatos creados por él.	

Tabla 3.10: Requisito funcional 6 - Listar todos los formatos creados

Requisito funcional 7	
ID:	RF07
Nombre:	Subir un *log*
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede subir un *log* al sistema.	

Tabla 3.11: Requisito funcional 7 - Subir un *log*

Requisito funcional 8	
ID:	RF08
Nombre:	Eliminar un *log* subido
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede eliminar un *log* del sistema previamente subido por él.	

Tabla 3.12: Requisito funcional 8 - Eliminar un *log* subido

Requisito funcional 9	
ID:	RF09
Nombre:	Crear un nuevo filtro
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede crear un filtro nuevo. Eso supondría que el filtro recogería la información de un *log* o el resultado de otro filtro (ambos necesariamente creados por el usuario) aplicaría el filtro y guardaría su resultado en la base de datos.	

Tabla 3.13: Requisito funcional 9 - Crear un nuevo filtro

Requisito funcional 10	
ID:	RF10
Nombre:	Eliminar un filtro creado
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede eliminar un filtro creado por él. Lo que supondría eliminar los datos resultado de este filtro de la base de datos.	

Tabla 3.14: Requisito funcional 10 - Eliminar un filtro creado

Requisito funcional 11	
ID:	RF11
Nombre:	Listar todos los filtros creados
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede listar todos los filtros creados por él.	

Tabla 3.15: Requisito funcional 11 - Listar todos los filtros creados

Requisito funcional 12	
ID:	RF12
Nombre:	Crear un nuevo procesamiento
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede crear un procesamiento nuevo. Eso supondría que el filtro recogería la información de un *log* o el resultado de otro filtro (ambos necesariamente creados por el usuario) aplicaría el procesamiento y guardaría su resultado en la base de datos.	

Tabla 3.16: Requisito funcional 12 - Crear un nuevo procesamiento

Requisito funcional 13	
ID:	RF13
Nombre:	Eliminar un procesamiento creado
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede eliminar un procesamiento creado por él. Lo que supondría eliminar los datos resultado de este filtro de la base de datos.	

Tabla 3.17: Requisito funcional 13 - Eliminar un procesamiento creado

Requisito funcional 14	
ID:	RF14
Nombre:	Listar todos los procesamientos creados
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede listar todos los procesamientos creados por él.	

Tabla 3.18: Requisito funcional 14 - Listar todos los procesamientos creados

Requisito funcional 15	
ID:	RF15
Nombre:	Crear un nuevo panel de monitorización
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede crear un nuevo panel de monitorización. Este panel el usuario podrá vincular los gráficos que quiera.	

Tabla 3.19: Requisito funcional 15 - Crear un nuevo panel de monitorización

Requisito funcional 16	
ID:	RF16
Nombre:	Eliminar un panel de monitorización
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede eliminar un panel de monitorización que el usuario haya creado previamente. Eliminando los gráficos que estén vinculados a este.	

Tabla 3.20: Requisito funcional 16 - Eliminar un panel de monitorización

Requisito funcional 17	
ID:	RF17
Nombre:	Crear un nuevo gráfico
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede crear un nuevo gráfico. Esto supone que el gráfico recoja la información, o bien de los datos de un *log* o de los datos resultado de un filtro o un procesamiento, todos necesariamente creados por el usuario, y muestre en el panel de monitorización de la aplicación que él elija el resultado de forma gráfica.	

Tabla 3.21: Requisito funcional 17 - Crear un nuevo gráfico

Requisito funcional 18	
ID:	RF18
Nombre:	Eliminar un gráfico creado
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede eliminar un gráfico creado por él, lo que supondrá eliminar su forma gráfica de la aplicación.	

Tabla 3.22: Requisito funcional 18 - Eliminar un gráfico creado

Requisito funcional 19	
ID:	RF19
Nombre:	Listar todos los gráfico creados
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede listar todos los gráficos creados por él.	

Tabla 3.23: Requisito funcional 19 - Listar todos los gráfico creados

Requisito funcional 20	
ID:	RF20
Nombre:	Visualizar todos los gráficos creados
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede visualizar de forma gráfica en la página de monitorización, los gráficos que haya creado él.	

Tabla 3.24: Requisito funcional 20 - Visualizar todos los gráficos creados

Requisito funcional 21	
ID:	RF21
Nombre:	Visualizar el historial de tareas
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede visualizar de forma gráfica las tareas que haya realizado a lo largo del tiempo.	

Tabla 3.25: Requisito funcional 21 - Visualizar el historial de tareas

Requisito funcional 22	
ID:	RF22
Nombre:	Eliminar el historial de tareas
Rol de usuario:	Usuarios registrados
Descripción	
Cualquier usuario registrado con una sesión abierta puede eliminar del historial aquellas tareas finalizadas que desee.	

Tabla 3.26: Requisito funcional 22 - Eliminar el historial de tareas

Requisito funcional 23	
ID:	RF23
Nombre:	Listar las cuentas de los usuarios
Rol de usuario:	Usuarios administradores
Descripción	
Cualquier usuario administrador con una sesión abierta puede listar las cuentas de todos los usuarios registrados en el sistema.	

Tabla 3.27: Requisito funcional 23 - Listar las cuentas de los usuarios

Requisito funcional 24	
ID:	RF24
Nombre:	Eliminar una cuenta de usuario o prohibir su acceso
Rol de usuario:	Usuarios administradores
Descripción	
Cualquier usuario administrador con una sesión abierta puede eliminar una cuenta de un usuario o prohibir su acceso.	

Tabla 3.28: Requisito funcional 24 - Eliminar una cuenta de usuario o prohibir su acceso

3.5.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que definen la calidad y se usan para juzgar al sistema basándose en cómo realiza una operación. Los requisitos no funcionales son los siguientes:

Requisito no funcional 1	
ID:	RNF01
Nombre:	Escalabilidad
Descripción	
El diseño del sistema en la medida de lo posible tiene que contemplar posibles ampliaciones y cambios para poder adaptarse sin perder calidad de manera fluida. Dichos cambios no tienen que suponer un coste excesivo en el desarrollo y en el funcionamiento de la página.	

Tabla 3.29: Requisito no funcional 1 - Escalabilidad

Requisito no funcional 2	
ID:	RNF02
Nombre:	Concurrencia
Descripción	
La aplicación tendrá que funcionar con uno o varios usuarios al mismo tiempo.	

Tabla 3.30: Requisito no funcional 2 - Concurrencia

Requisito no funcional 3	
ID:	RNF03
Nombre:	Seguridad
Descripción	
La aplicación en la medida de lo posible tendrá que preservar la integridad, disponibilidad y confidencialidad de los datos de cada usuario.	

Tabla 3.31: Requisito no funcional 3 - Seguridad

Requisito no funcional 4	
ID:	RNF04
Nombre:	Rendimiento
Descripción	
Para hacer una navegación más interactiva para el usuario se hará uso de la tecnología multihilo (*multithreading*) para realizar las operaciones más costosas temporalmente, como la lectura de *logs* y la escritura en la base de datos del resultado de la lectura y de los resultados de los filtros.	

Tabla 3.32: Requisito no funcional 4 - Rendimiento

3.6. Casos de uso

Los casos de uso son cada uno de ellos una descripción inicial en actividades que necesariamente se tendrán que realizar para concluir con un proceso u objetivo de la funcionalidad de la aplicación. Se registrarán también los actores de dichas actividades y las condiciones previas y posteriores. Los casos de uso son los siguientes:

Caso de uso 1	
ID:	CU01
Nombre:	Registrarse en el sistema.
Actores:	Usuario no registrado.
Objetivo:	El objetivo es guardar los datos iniciales de un usuario para que este pueda usar el sistema.
Precondiciones:	Ninguna.
Postcondiciones:	La información del usuario queda registrada en la base de datos.
Escenario básico	
1. El usuario entra en la página de registro. 2. Introduce la información requerida, entre ellas su *e-mail* (identificador de usuario) y su contraseña. 3. El sistema comprueba si es correcta *a*) Si es incorrecta o está incompleta, el sistema advierte con un mensaje de error, se vuelve al primer paso. *b*) Si es correcta, el usuario se registra en el sistema y automáticamente pasa a la página de inicio, realizando el proceso automáticamente de *login* (caso de uso CU02).	

Tabla 3.33: Caso de uso 1 - Registrarse en el sistema

Caso de uso 2	
ID:	CU02
Nombre:	*Loguearse* en el sistema.
Actores:	Usuario registrado.
Objetivo:	El objetivo es iniciar una nueva sesión con un usuario.
Precondiciones:	El usuario debe estar registrado, caso de uso CU01.
Postcondiciones:	Nueva sesión del usuario hasta que esta se cierre o se acabe el tiempo de sesión.
Escenario básico	
1. El usuario entra en la página de *login*. 2. Introduce su *e-mail* y contraseña. 3. El sistema comprueba si son correctas *a*) Si es incorrecta, el sistema advierte con un mensaje de error, se vuelve al primer paso. *b*) Si es correcta, se inicia una nueva sesión y se envía a la página de inicio.	

Tabla 3.34: Caso de uso 2 - *Loguearse* en el sistema

Caso de uso 3	
ID:	CU03
Nombre:	Ver/Editar la información del perfil de usuario
Actores:	Usuario registrado.
Objetivo:	El objetivo es modificar o añadir nueva información del perfil del usuario.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02.
Postcondiciones:	Modificación de la información del perfil en la base de datos.
Escenario básico	
1. El usuario entra en la página del perfil. 2. Introduce la información que quiere añadir o modificar. 3. El usuario pide guardar los cambios. 4. El sistema comprueba si son correctas. *a*) Si es incorrecta el sistema advierte con un mensaje de error, se vuelve al primer paso. *b*) Si es correcta se guardan los cambios en la base de datos y se vuelve a la página de inicio.	

Tabla 3.35: Caso de uso 3 - Ver/Editar la información del perfil de usuario

Caso de uso 4	
ID:	CU04
Nombre:	Prohibir/Eliminar la cuenta de un usuario
Actores:	Usuario administrador.
Objetivo:	El objetivo es que el usuario administrador prohíba el acceso o elimine la cuenta de un usuario registrado.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02.
Postcondiciones:	Se modifica/elimina la información de la cuenta del usuario en la base de datos.
Escenario básico	
1. El administrador entra en la página del listado de usuarios. 2. El administrador selecciona una cuenta de la lista y decide si: *a*) El administrador prohíbe la cuenta o, en el caso de que estuviera prohibida, le permite de nuevo el acceso. *b*) El administrador elimina la cuenta del usuario, eliminando por tanto toda la información relacionada con el del sistema.	

Tabla 3.36: Caso de uso 4 - Prohibir/Eliminar la cuenta de un usuario

Caso de uso 5	
ID:	CU05
Nombre:	Añadir un nuevo panel el sistema
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario añada un nuevo panel en el que posteriormente pueda rellenar con distintos elementos, como gráficos, tablas...
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02.
Postcondiciones:	La información del nuevo panel se registra en la base de datos.
Escenario básico	
1. El usuario entra en el menú principal de la aplicación. 2. El usuario elige la opción de añadir un nuevo panel. 3. El usuario introduce el nombre de este nuevo panel. 4. Se guarda dicha información en la base de datos y se recarga la página principal.	

Tabla 3.37: Caso de uso 5 - Añadir un nuevo panel el sistema

Caso de uso 6	
ID:	CU06
Nombre:	Eliminar un panel del sistema
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario elimine un panel previamente creado y la información relacionada a este.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02, y haber creado el panel a eliminar, caso de uso C05.
Postcondiciones:	La información del panel se elimina de la base de datos y, al igual que su información relacionada.
Escenario básico	

1. El usuario entra en el menú principal de la aplicación.
2. El usuario selecciona el panel a eliminar y elige la opción de eliminar ese panel.
3. Se proporciona un mensaje de comprobación para confirmar la eliminación.
   a) Si se confirma, se elimina la información del panel de la base de datos y se redirige a la página de inicio.
   b) Si se cancela, la acción no se produce ningún efecto.

Tabla 3.38: Caso de uso 6 - Eliminar un panel del sistema

Caso de uso 7	
ID:	CU07
Nombre:	Definir un nuevo tipo de formato de *log*
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario pueda crear un formato para poder leer un *log* que posteriormente el usuario suba al sistema.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02.
Postcondiciones:	La información de este nuevo formato se registra en la base de datos.
Escenario básico	

1. El usuario entra en el menú principal de la aplicación.
2. El usuario elige la opción de añadir un nuevo formato.
3. El usuario introduce la información correspondiente dependiendo del tipo de formato.
4. El sistema valida la información introducida.
   a) Si es errónea o está incompleta, el sistema advierte al usuario con un mensaje de error y se vuelve al primer paso.
   b) Si es correcta, se guarda en la base de datos y se redirige al menú principal.

Tabla 3.39: Caso de uso 7 - Definir un nuevo tipo de formato de *log*

Caso de uso 8	
ID:	CU08
Nombre:	Editar/Eliminar un tipo de formato de *log* registrado
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario pueda editar o eliminar un formato de *log* creado previamente.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02, y haber creado un formato de *log*, caso de uso CU07.
Postcondiciones:	Se modifica/elimina la información del formato en la base de datos.
Escenario básico	

1. El usuario entra en el menú principal de la aplicación.
2. El usuario selecciona el formato de *log*.
3. El usuario elige si:
   - La eliminación del formato de *log*:
     a) Se proporciona un mensaje de comprobación para confirmar la eliminación.
        1) Si se confirma, se elimina la información del formato de la base de datos y se redirige a la página de inicio.
        2) Si se cancela, la acción no se produce ningún efecto.
   - La edición del formato de *log*:
     a) El usuario modifica la información correspondiente dependiendo del tipo de formato.
     b) El sistema valida la información introducida.
        1) Si es errónea o está incompleta, el sistema advierte al usuario con un mensaje de error y se vuelve al primer paso.
        2) Si es correcta, se guarda en la base de datos y se redirige al menú principal.

Tabla 3.40: Caso de uso 8 - Editar/Eliminar un tipo de formato de *log* registrado

Caso de uso 9	
ID:	CU09
Nombre:	Subir un *log* al sistema
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario pueda subir un *Log* al sistema.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02, y haber creado un formato de *log*, caso de uso CU07.
Postcondiciones:	Se guarda el *log* en el sistema de archivos del servidor y su información en la base de datos.
Escenario básico	
1. El usuario entra el la página de añadir un nuevo *log*. 2. El usuario introduce la información del *log* en el formulario, como el formato del mismo, y selecciona el fichero *log* de su ordenador que quiere subir: 3. El sistema valida la información introducida. *a*) Si es errónea o está incompleta, el sistema advierte al usuario con un mensaje de error y se vuelve al primer paso. *b*) Si es correcta, se guarda en la base de datos y se redirige al menú principal.	

Tabla 3.41: Caso de uso 9 - Subir un *log* al sistema

Caso de uso 10	
ID:	CU10
Nombre:	Eliminar un *log* del sistema
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario pueda eliminar un *log* subido anteriormente en el sistema.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02, y haber subido un *log* con anterioridad, caso de uso CU09.
Postcondiciones:	Se elimina el *log* del sistema de archivos del servidor y su información de la base de datos.
Escenario básico	
1. El usuario entra en el menú principal de la aplicación. 2. El usuario selecciona el *log* y elije la opción de eliminarlo. 3. Se proporciona un mensaje de comprobación para confirmar la eliminación. *a*) Si se confirma, se elimina el *log* y se redirige a la página de inicio. *b*) Si se cancela, la acción no se produce ningún efecto.	

Tabla 3.42: Caso de uso 10 - Eliminar un *log* del sistema

Caso de uso 11	
ID:	CU11
Nombre:	Definir un nuevo filtro en el sistema
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario pueda crear un filtro para poder limpiar información no relevante de un *log* que el usuario haya subido al sistema.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02, y haber subido un *log* con anterioridad, caso de uso CU09.
Postcondiciones:	La información de este nuevo filtro se registra en la base de datos, junto con su resultado en el sistema de archivos.
Escenario básico	
1. El usuario entra en el menú principal de la aplicación. 2. El usuario elige la opción de añadir un nuevo filtro. 3. El usuario introduce la información correspondiente dependiendo del tipo de filtro. 4. El sistema valida la información introducida. *a*) Si es errónea o está incompleta, el sistema advierte al usuario con un mensaje de error y se vuelve al primer paso. *b*) Si es correcta, se guarda en la base de datos y se redirige al menú principal.	

Tabla 3.43: Caso de uso 11 - Definir un nuevo filtro en el sistema

Caso de uso 12	
ID:	CU12
Nombre:	Editar/Eliminar un filtro registrado
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario pueda editar o eliminar un filtro creado previamente.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02, y haber creado un filtro, caso de uso CU11.
Postcondiciones:	Se modifica/elimina la información del filtro en la base de datos y se modifica/elimina su resultado en el sistema de archivos.
Escenario básico	

1. El usuario entra en el menú principal de la aplicación.
2. El usuario selecciona el filtro.
3. El usuario elige si:
   - La eliminación del filtro:
     a) Se proporciona un mensaje de comprobación para confirmar la eliminación.
        1) Si se confirma, se elimina la información del filtro de la base de datos y se redirige a la página de inicio.
        2) Si se cancela, la acción no se produce ningún efecto.
   - La edición del filtro:
     a) El usuario modifica la información correspondiente dependiendo del tipo de filtro.
     b) El sistema valida la información introducida.
        1) Si es errónea o está incompleta, el sistema advierte al usuario con un mensaje de error y se vuelve al primer paso.
        2) Si es correcta, se guarda en la base de datos y se redirige al menú principal.

Tabla 3.44: Caso de uso 12 - Editar/Eliminar un filtro registrado

Caso de uso 13	
ID:	CU13
Nombre:	Definir un nuevo procesamiento en el sistema
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario pueda crear un procesamiento para transformar o deducir información de un *log* que el usuario haya subido al sistema.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02, y haber subido un *log* con anterioridad, caso de uso CU09.
Postcondiciones:	La información de este nuevo procesamiento se registra en la base de datos, junto con su resultado en el sistema de archivos.
Escenario básico	

1. El usuario entra en el menú principal de la aplicación.
2. El usuario elige la opción de añadir un nuevo procesamiento.
3. El usuario introduce la información correspondiente dependiendo del tipo de procesamiento.
4. El sistema valida la información introducida.
   a) Si es errónea o está incompleta, el sistema advierte al usuario con un mensaje de error y se vuelve al primer paso.
   b) Si es correcta, se guarda en la base de datos y se redirige al menú principal.

Tabla 3.45: Caso de uso 13 - Definir un nuevo procesamiento en el sistema

Caso de uso 14	
ID:	CU14
Nombre:	Editar/Eliminar un procesamiento registrado
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario pueda editar o eliminar un procesamiento creado previamente.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02, y haber creado un procesamiento, caso de uso CU13.
Postcondiciones:	Se modifica/elimina la información del procesamiento en la base de datos y se modifica/elimina su resultado en el sistema de archivos.
Escenario básico	

1. El usuario entra en el menú principal de la aplicación.
2. El usuario selecciona el procesamiento.
3. El usuario elige si:
   - La eliminación del procesamiento:
     a) Se proporciona un mensaje de comprobación para confirmar la eliminación.
        1) Si se confirma, se elimina la información del procesamiento de la base de datos y se redirige a la página de inicio.
        2) Si se cancela, la acción no se produce ningún efecto.
   - La edición del procesamiento:
     a) El usuario modifica la información correspondiente dependiendo del tipo de procesamiento.
     b) El sistema valida la información introducida.
        1) Si es errónea o está incompleta, el sistema advierte al usuario con un mensaje de error y se vuelve al primer paso.
        2) Si es correcta, se guarda en la base de datos y se redirige al menú principal.

Tabla 3.46: Caso de uso 14 - Editar/Eliminar un procesamiento registrado

Caso de uso 15	
ID:	CU15
Nombre:	Definir un nuevo gráfico en el sistema
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario pueda crear un gráfico para visualizar la información de un *log* que el usuario haya subido al sistema.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02, y haber subido un *log* con anterioridad, caso de uso CU09.
Postcondiciones:	La información de este nuevo gráfico se registra en la base de datos, y al entrar al panel vinculado se mostrará gráficamente su resultado.
Escenario básico	

1. El usuario entra en el menú principal de la aplicación.
2. El usuario elige la opción de añadir un nuevo gráfico.
3. El usuario introduce la información correspondiente dependiendo del tipo de gráfico.
4. El sistema valida la información introducida.
   a) Si es errónea o está incompleta, el sistema advierte al usuario con un mensaje de error y se vuelve al primer paso.
   b) Si es correcta, se guarda en la base de datos y se redirige al menú principal.

Tabla 3.47: Caso de uso 15 - Definir un nuevo gráfico en el sistema

Caso de uso 16	
ID:	CU16
Nombre:	Editar/Eliminar un gráfico registrado
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario pueda editar o eliminar un gráfico creado previamente.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02, y haber creado un gráfico, caso de uso CU15.
Postcondiciones:	Se modifica/elimina la información del gráfico en la base de datos y se modifica/elimina su resultado en el panel vinculado.
Escenario básico	

1. El usuario entra en el menú principal de la aplicación.
2. El usuario selecciona el gráfico.
3. El usuario elige si:
   - La eliminación del gráfico:
     a) Se proporciona un mensaje de comprobación para confirmar la eliminación.
        1) Si se confirma, se elimina la información del gráfico de la base de datos y se redirige a la página de inicio.
        2) Si se cancela, la acción no se produce ningún efecto.
   - La edición del gráfico:
     a) El usuario modifica la información correspondiente dependiendo del tipo de gráfico.
     b) El sistema valida la información introducida.
        1) Si es errónea o está incompleta, el sistema advierte al usuario con un mensaje de error y se vuelve al primer paso.
        2) Si es correcta, se guarda en la base de datos y se redirige al menú principal.

Tabla 3.48: Caso de uso 16 - Editar/Eliminar un gráfico registrado

Caso de uso 17	
ID:	CU17
Nombre:	Ver el historial de tareas del usuario
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario pueda ver la sucesión de tareas que estén ejecutándose o que hayan finalizado así como su progreso o los posibles errores que hayan tenido.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02.
Postcondiciones:	Ninguna.
Escenario básico	
1. El usuario entra en el menú principal de la aplicación. 2. El usuario elige ir al historial de tareas. 3. El usuario puede ver las tareas, la fecha/hora en el que fueron creadas, su progreso y los posibles errores que hayan tenido.	

Tabla 3.49: Caso de uso 17 - Ver el historial de tareas del usuario

Caso de uso 18	
ID:	CU18
Nombre:	Eliminar todas las tareas finalizadas del historial
Actores:	Usuario registrado.
Objetivo:	El objetivo es que el usuario pueda eliminar las tareas que hayan finalizado su ejecución en esos instantes.
Precondiciones:	El usuario debe estar *logueado*, caso de uso CU02.
Postcondiciones:	Se elimina la información de todas las tareas no pendientes en la base de datos.
Escenario básico	
1. El usuario entra en el menú principal de la aplicación. 2. El usuario elige ir al historial de tareas. 3. El usuario elije eliminar todas las tareas finalizadas: 4. Se proporciona un mensaje de comprobación para confirmar la eliminación. *a*) Si se confirma, se elimina la información del gráfico de la base de datos y se redirige a la página de inicio. *b*) Si se cancela, la acción no se produce ningún efecto.	

Tabla 3.50: Caso de uso 18 - Eliminar todas las tareas finalizadas del historial

Capítulo 4

Diseño

En este capítulo se define la aplicación a desarrollar, así como sus componentes y su arquitectura. En la primera parte se define la arquitectura de la aplicación, en la segunda parte se divide el sistema en una serie de subsistemas que implementan cada uno parte de la funcionalidad, el la cuarta parte se muestra el modelo de la base de datos y, por último, se realiza el diseño de la interfaz y se muestran ejemplos de la plantilla utilizada.

4.1. Arquitectura del sistema

La arquitectura del sistema es el diseño a más alto nivel de la aplicación. Consiste en determinar el conjunto de partes del sistema y mostrar la interacción que tienen unas con otras. En primer lugar se hablará del patrón de diseño utilizado, Modelo vista controlador, en segundo lugar se hablará sobre la descarga de las dependencias y librerías mediante *Apache Maven* [64], y por último sobre la arquitectura de la aplicación web como resultado de usar *Spring Framework* [7].

4.1.1. Modelo vista controlador

El modelo vista controlador (*MVC*, siglas en inglés de *Model-View-Controller*) es un patrón de diseño para separar la información real o el modelo de la representación de la información que el usuario utiliza. Utilizado para aplicaciones con interfaces gráficas complejas y, sobre todo, en aplicaciones web como es el caso. Podemos ver la interacción entre las tres partes en el siguiente diagrama 4.1:

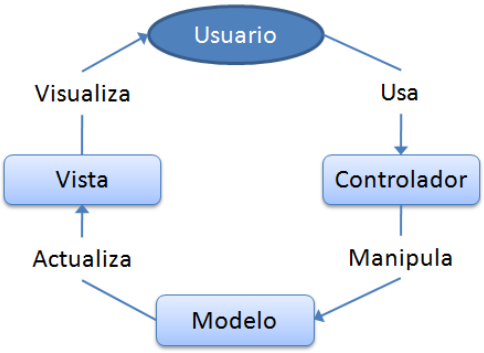

Figura 4.1: Modelo vista controlador.

4.1.2. Maven

Apache Maven [64] es una herramienta de gestión de proyectos software. Está basado en un modelo de objetos de proyecto (en inglés *Project Object Model*, *POM*). El *POM* es un archivo *XML* que define cosas como el nombre del proyecto, su versión, los repositorios usados, las dependencias necesarias... Es muy útil a la hora de implementar una aplicación web ya que muchas herramientas de desarrollo como *NetBeans* o *Eclipse* ya la tienen integrada y buscar y descargar librerías para nuestra aplicación se hace mucho más sencillo.

Se puede observar parte del archivo de configuración de *Maven*, *POM*, llamado *pom.xml* en la aplicación *LogExplorer* en el siguiente código:

Código 4.1: Parte del archivo de configuración de *Maven*.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.uam</groupId>
  <artifactId>LogExplorer</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>LogExplorer</name>
  <properties>
    <endorsed.dir>
      ${project.build.directory}/endorsed
    </endorsed.dir>
    <project.build.sourceEncoding>
      UTF-8
    </project.build.sourceEncoding>
    <spring.version>4.1.7.RELEASE</spring.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
```

```
      <artifactId>junit</artifactId>
      <version>4.12</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>${spring.version}</version>
    </dependency>
        ...
  </dependencies>
    ...
</project>
```

4.1.3. Spring Framework

El *framework* de *Spring* [7] permite el desarrollo y configuración de una aplicación web en *Java* de forma más fácil y rápida. Se basa en *Java* y *J2EE*. Su uso consiste en una serie de anotaciones en el mismo código *Java*, lo que lo hace mucho más fácil de entender y se puede desarrollar rápidamente. *Spring* es un *framework* ligero de inversión de control (en inglés *lightweight Inversion of Control*, *IoC*), es decir, en vez de especificar la secuencia de decisiones y procedimientos que pueden darse, en *Spring* se programan respuestas deseadas a solicitudes, dejando que el *Dispatcher* de *Spring* lleve a cabo las acciones de control que se requieran en el orden necesario. Este *Dispatcher* se configura, por lo general, mediante un fichero *XML*. En este proyecto el fichero tiene el nombre de *"SpringDispatcher-servlet.xml"*.

En este archivo también se configura *Hibernate* [65]. *Hibernate* es una herramienta de Mapeo objeto-relacional (en inglés *Object-Relational mapping*, *ORM*) que nos permite convertir fácilmente de los datos relacionales de una base de datos a orientados a objetos, como tenemos en una clase *Java*. Esto nos facilita almacenar y recuperar datos de la base de datos en forma de los objetos. Para visualizar mejor la forma de configuración el código 4.2 nos muestra parte del fichero de configuración de *Spring*.

Código 4.2: Configuración de *Hibernate*.

```
<bean id="dataSource" class="
org.springframework.jdbc.datasource.DriverManagerDataSource
">
 <property name="driverClassName"
    value="${jdbc.driverClassName}" />
 <property name="url" value="${jdbc.url}" />
 <property name="username" value="${jdbc.username}" />
 <property name="password" value="${jdbc.password}" />
</bean>

<bean id="sessionFactory" class="
org.springframework.orm.hibernate4.LocalSessionFactoryBean
">
 <property name="dataSource" ref="dataSource" />
 <property name="annotatedClasses">
  <list>
    <value>org.uam.logexplorer.model.UserAuthentication</value>
    ...
```

```
  </list>
 </property>
 <property name="hibernateProperties">
  <props>
   <prop key="hibernate.dialect">${hibernate.dialect}</prop>
   <prop key="hibernate.show_sql">${hibernate.show_sql}</prop>
  </props>
 </property>
</bean>
```

Para una mayor claridad se ha recogido los datos necesarios para la conexión con la base de datos en un fichero llamado *hibernate.properties*, dicho fichero tiene valores para poder conectar con la *JDBC* mediante la *URL* de la base de datos y las credenciales, usuario y contraseña.

Para entender el proceso de la aplicación web la figura 4.2 muestra una arquitectura simplificada donde se señala los tipos de clase más importantes en la aplicación. Se muestra en primer lugar la clase *Controller* que se dedica a implementar las respuestas a consultas realizadas a la aplicación. Esta clase utiliza de la interfaz *Service* implementada por la clase *ServiceImpl* que da la funcionalidad de cada servicio de la aplicación. La interfaz *DAO*, a su vez, implementada por la clase *DAOImpl* es la que proporciona el acceso a la información de la base de datos, en este caso realizadas con consultas de *Hibernate* a la base de datos de *PostgreSQL*.

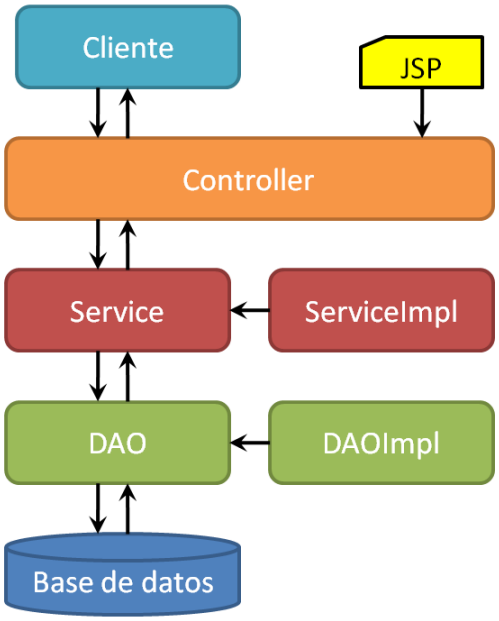

Figura 4.2: Arquitectura simplificada de la aplicación basada en *Spring*.

4.2. Subsitemas

Para facilitar el desarrollo de la aplicación se ha dividido en cuatro subsistemas según su funcionalidad. Estos son:

4.2.1. Subsistema de Usuario

El subsistema de usuario es el encargado de la autenticación, autorización y la gestión de la información de todos los usuarios en el sistema, así como del registro de un nuevo usuario. Por cada usuario se contemplan cuatro partes:

1. **La información de la autenticación**: que contendrá la información necesaria para el login del usuario, es decir, el nombre único del usuario en el sistema (representado con su *e-mail*) y su contraseña.
2. **Los roles del usuario**: es decir, la autorización, que especificará si el usuario es administrador o no.
3. **La información del perfil del usuario**: que contendrá el resto de información del usuario como; su nombre y apellidos, su cumpleaños, su foto de perfil, la fecha de registro...

4.2.2. Subsistema de Gestión de Datos

El subsistema de gestión de datos centra en gestionar la información de todos los colectores y de *parsear*, es decir, traducir y guardar la información leída de un *log* en la base de datos. Este subsistema es muy importantes ya que su correcto diseño supondrá un análisis y tratamiento flexible de la información. La figura 4.3 muestra el diagrama de clases de la gestión de datos del proyecto. Se puede observar en ella el uso del patrón de diseño *Proxy* [66] para diferenciar aquellas colecciones de datos recogidas en un fichero *log* de las almacenadas en memoria.

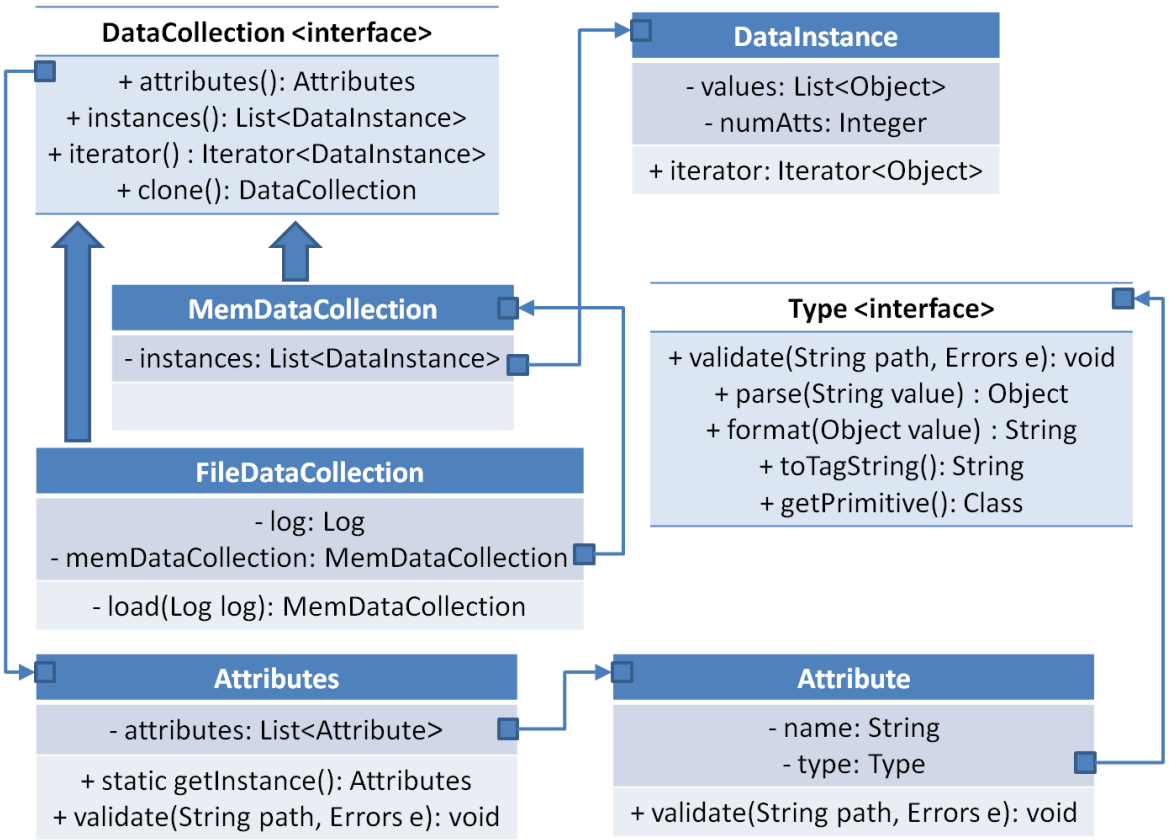

Figura 4.3: Diagrama de clases de la Gestión de Datos.

4.2.3. Subsistema de Tareas

El subsistema de Tareas será el encargado de gestionar de los hilo de ejecución y la información que se recogerá en un historial de tareas, definiendo una tarea como una acción llevada por un usuario. La figura 4.4 muestra el diagrama de clases del subsistema de tareas. Para que el usuario conozca el estado de cada hilo de ejecución mientras está procesándose, se usa el patrón de diseño *Observer* [67].

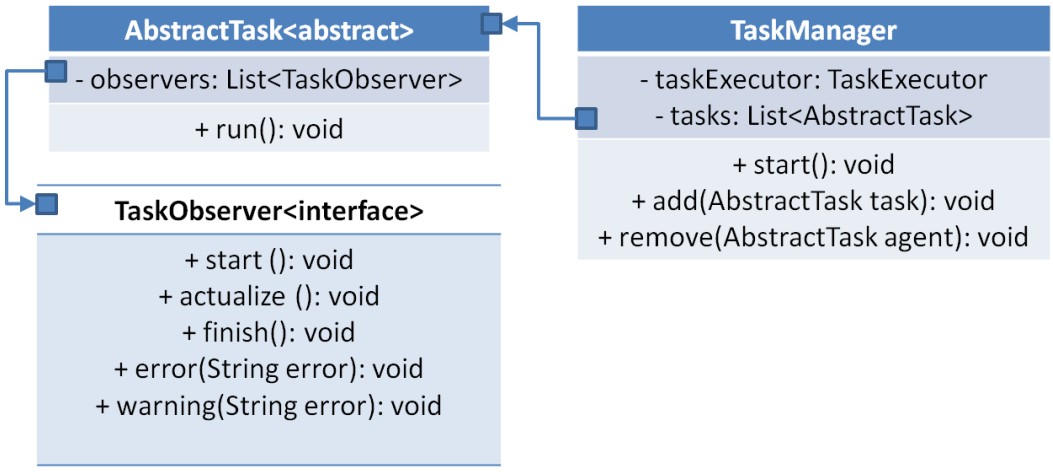

Figura 4.4: Diagrama de clases del Subsistema de Tareas.

4.2.4. Subsistema de Filtros

El subsistema de filtro se encarga de gestionar la información de los filtros creados. Estos filtros tienen por objetivo eliminar aquellas instancias de una colección que no cumplan una condición determinada. Algunos de ellos son:

- **Rango númerico**: dado un atributo numérico, el mínimo y/o el máximo de un rango, el filtro permitirá solo aquellas instancias cuyo atributo esté en dicho rango.
- **Comparación**: dado un atributo, una opción de comparación (igual que, mayor que, menor que...) que dependa del tipo de atributo y otro valor, el filtro permitirá aquellas instancias cuyo atributo cumpla la comparación.
- **Expresión regular**: dado un atributo y una expresión regular, el filtro admitirá solo aquellas instancias cuyo atributo coincidan con la expresión regular.
- **Rango de longitud de cadena**: dado un atributo de tipo cadena de caracteres, un mínimo y/o un máximo del número de caracteres permitidos, el filtro permitirá solo aquellas instancias cuyo atributo cumpla con dicho rango de longitud.
- **Rango de fechas**: dado un atributo de tipo fecha, un mínimo y/o un máximo de rango de fecha, el filtro permitirá solo aquellas instancias cuyo atributo esté en dicho rango.

4.2.5. Subsistema de Procesamiento

El subsistema de procesamiento será el encargado de gestionar los procesamientos creados y de aplicar dichos procesamientos. Los procesamientos tienen por objetivo transformar o deducir información de una colección de datos. Algunos procesamientos son:

- **Obtener información de una fecha**: este procesamiento, dado atributo de tipo fecha, obtiene parte de de dicha fecha como el año, los segundos, las horas....
- **Conversión de fechas**: este procesamiento convierte una fecha completa a segundos, minutos, horas...
- **Normalizar**: este procesamiento transforma un atributo numérico para que tenga una determinada media y/o desviación típica.
- **Seleccionar atributos**: este procesamiento elige los atributos de una colección y el resto los elimina.
- **Seleccionar instancias**: este procesamiento elige las instancias de una colección y el resto las elimina.
- **Contar instancias según un intervalo de tiempo**: este atributo, dado un atributo de tipo fecha y un intervalo de tiempo, cuenta el número de instancias desde el mínimo del atributo, intervalo por intervalo, hasta su máximo.
- **Contar valores únicos**: este procesamiento cuenta el número de instancias por cada valor único de un atributo dado.
- **Localización direcciones *IP***: este procesamiento, dado un atributo de tipo dirección *IP*, devuelve el lugar o el país donde se encuentra el dispositivo al que hace referencia.
- **Ordenar**: este procesamiento ordena de menor a mayor o de mayor a menor según un atributo dado.
- **Aplicar un algoritmo de Aprendizaje Automático**: este procesamiento utilizaría un algoritmo de aprendizaje automático y su respectivas entradas y devolvería o bien una salida completa o bien añadiría un nuevo atributo resultado a la colección dada en la entrada. Como ejemplos podrían ser *PCA* y *K-means*.

4.2.6. Subsistema de Gráficos

El subsistema de gráficos se encargará de gestionar la información de los gráficos creados y de representar los datos leídos de la base de datos. También será el encargado de gestionar los paneles, es decir, de las pantallas en las que se contendrán dichos gráficos ordenadas que se visualizarán de una sola vez. Algunos gráficos son:

- **Tabla**: es el más simple, que muestra en una tabla los datos que se hayan elegido de una colección.
- **Lineal**: este gráfico necesitará un atributo numérico como eje horizontal y uno o varios como cada una de las líneas a representar.
- **Área**: este gráfico necesitará un atributo numérico como eje horizontal y uno o varios como cada una de las áreas a representar.

- **De barras**: este gráfico necesitará un atributo como el eje horizontal y uno o varios como cada una de las barras a representar.
- **Circular**: este gráfico necesitará un atributo para representar en forma circular el porcentaje del valor de dicho atributo para todos los valores posibles.
- **Mapa mundial**: este gráfico necesitará un atributo de tipo cadena de caracteres específico cuyo valor sea el nombre del país al que hace referencia. El gráfico mostrará un mapa mundial donde el color de cada país indique el número de instancias por cada valor del atributo.

4.3. Base de datos

La base de datos que utiliza la aplicación es *PostgreSQL* [68]. Se ha decidido esta herramienta ya que ya se había usado con anterioridad en proyectos antiguos. La base de datos proporciona la base de la aplicación, contiene nueve tablas donde se guarda la información de carácter persistente. La siguiente figura 4.5 muestra el esquema relacional de la base de datos:

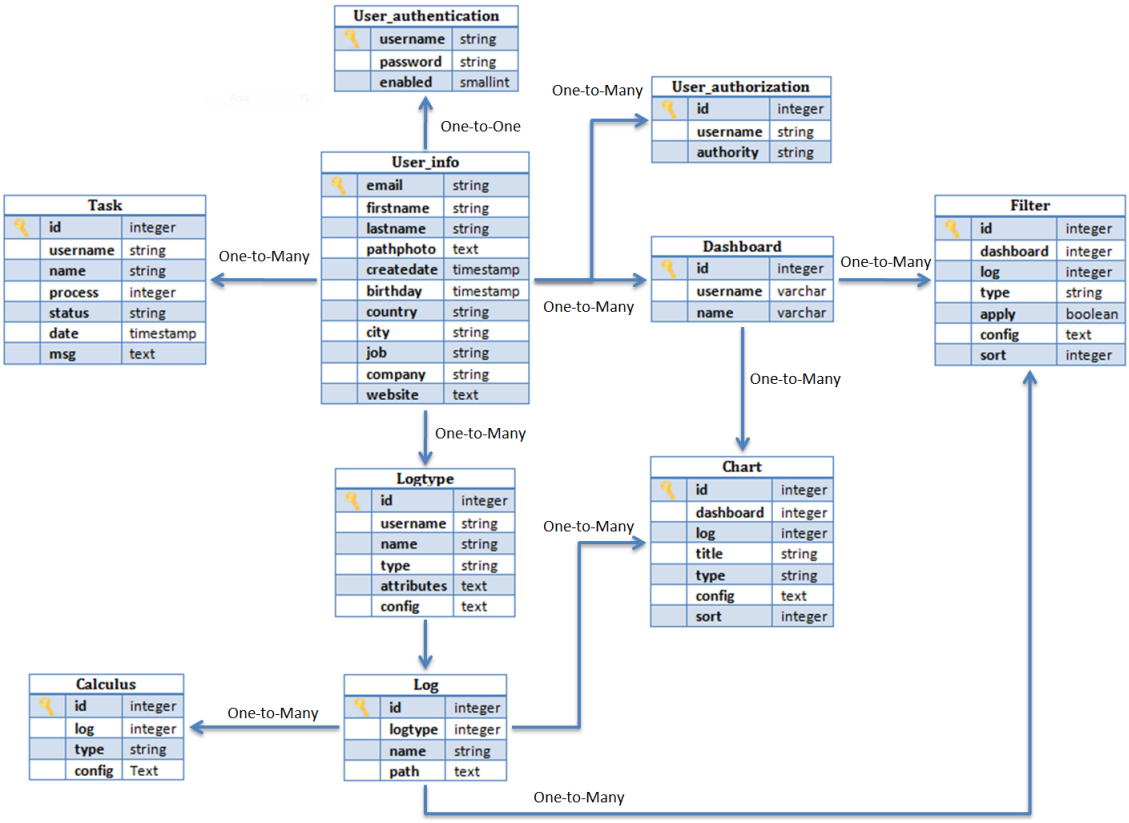

Figura 4.5: Esquema relacional de la Base de Datos del proyecto *LogExplorer*.

4.4. Diseño de la interfaz

Para la interfaz web se usará una plantilla llamada *AdminLTE-2.2.0* [11] de *Almsaeed Studio* [12] que hace uso de la biblioteca *JavaScript Bootstrap* [10]. Este apartado se divide en dos

partes, un diagrama de navegación y algunas imágenes que muestran la plantilla y sus posibles usos en la aplicación.

4.4.1. Diagrama de navegación

La figura 4.6 muestra el diagrama de navegación. En el se observa como la mayor parte de las páginas, ha excepción de la página de registro y la página de login, contienen un menú principal que permite la navegación prácticamente completa de la aplicación web.

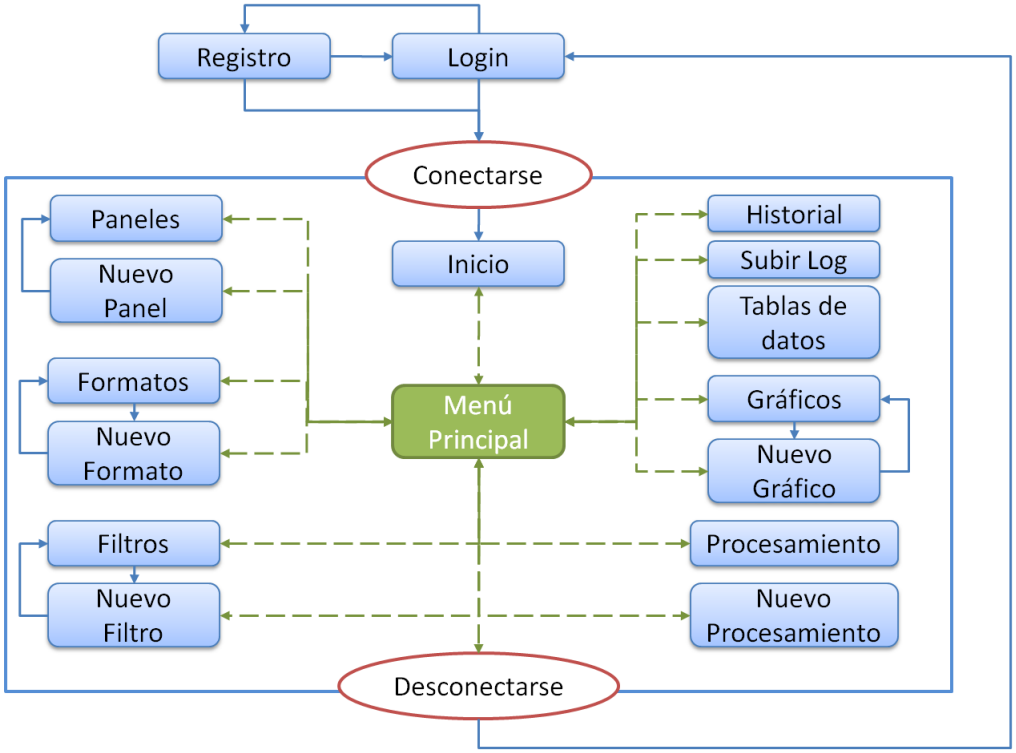

Figura 4.6: Mapa de navegación del proyecto *LogExplorer*.

4.4.2. Plantilla

En este apartado se mostrarán algunos ejemplos de la plantilla que pueden proporcionar la funcionalidad gráfica descrita anteriormente.

Login

La página de login se encarga del primer formulario donde se autentica un usuario usuario registrado. La aplicación tendrá que validar la información del *email* o la contraseña, en caso de introducirse erróneamente, la página deberá advertir al usuario. Debajo del formulario hay un enlace que lleva a la página de registro. La figura 4.7 muestra el ejemplo dado por la plantilla.

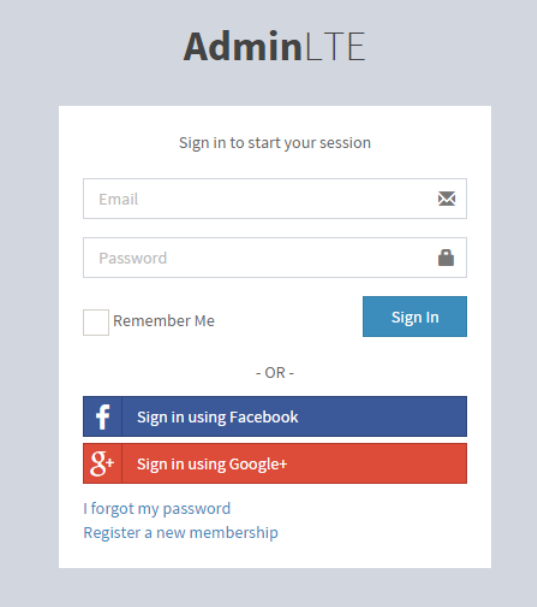

Figura 4.7: Página de Login.

Registro

La página de registro es un formulario en el que se rellena la información básica de un nuevo usuario del sistema. Cualquier usuario puede registrase, no es necesario la aprobación del administrador del sistema, aunque este puede paralizar cualquier cuenta. Al igual que en el caso del login, es necesario validar la información y advertir al usuario si esta es errónea. Si se rellena correctamente, el usuario quedará registrado en el sistema y se redirigirá automáticamente a la página de inicio, creando una nueva sesión con ese nuevo usuario. La figura 4.8 muestra el ejemplo de la página de registro.

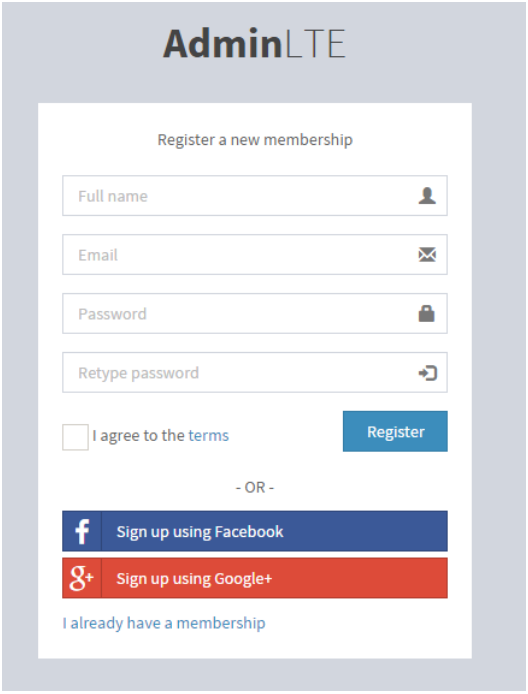

Figura 4.8: Página de Registro.

Paneles

Los paneles de monitorización son los que contendrán las gráficas creadas por los usuarios donde podrán ver visualmente la información de sus *logs*. La figura 4.9 muestra el ejemplo de la plantilla. Esta imagen también muestra el menú principal a la derecha y el menú de cabecera arriba, parte esencial de la navegación de la aplicación.

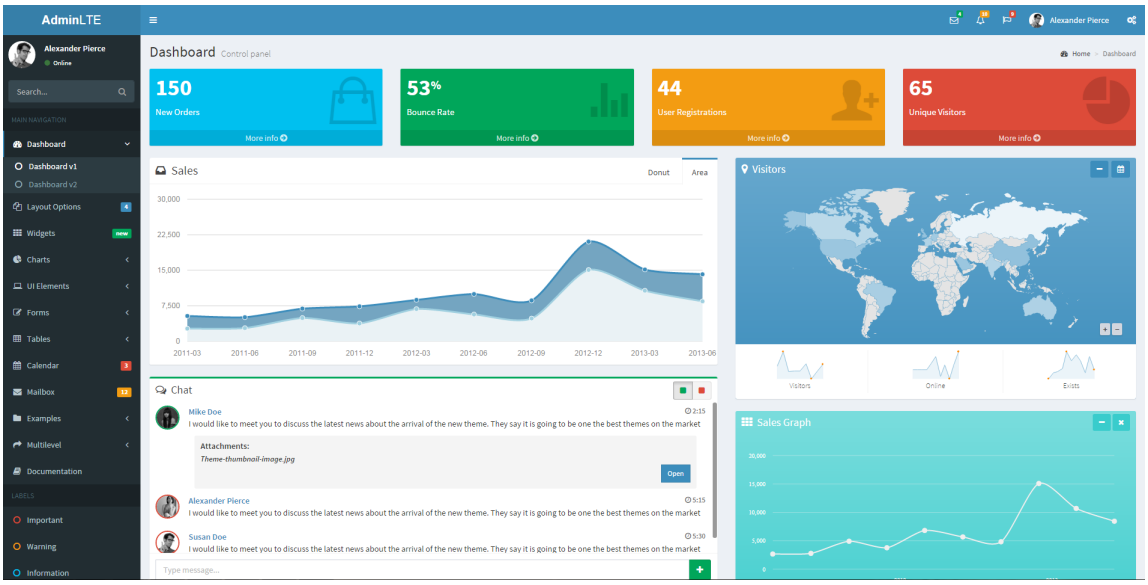

Figura 4.9: Panel de monitorización.

Formularios

Los formularios son elementos gráficos que permiten al usuario introduzca información para que esta sea procesada por el sistema. Un ejemplo sería el formulario dedicado a la subida de un *log*. La figura 4.10 muestra un ejemplo de formulario para la subida de ficheros. La figura 4.11 muestra un ejemplo de formulario más avanzado para que el usuario pueda escoger entre un rango de tiempo, fechas y horas, que puede ser útil para la configuración de filtros o procesamientos.

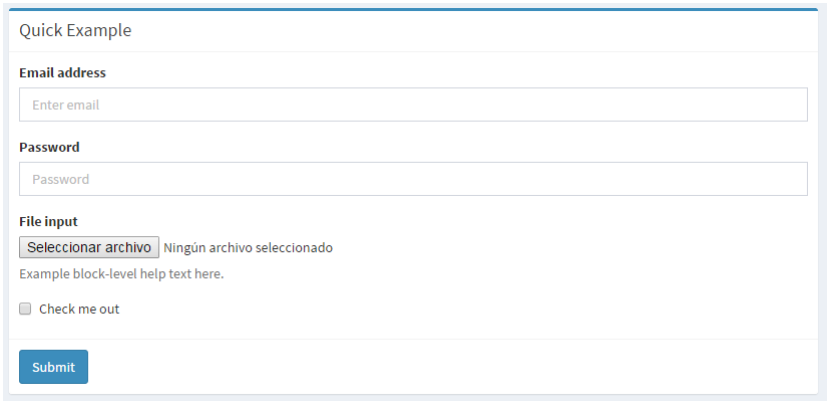

Figura 4.10: Ejemplo de formulario para la subida de ficheros.

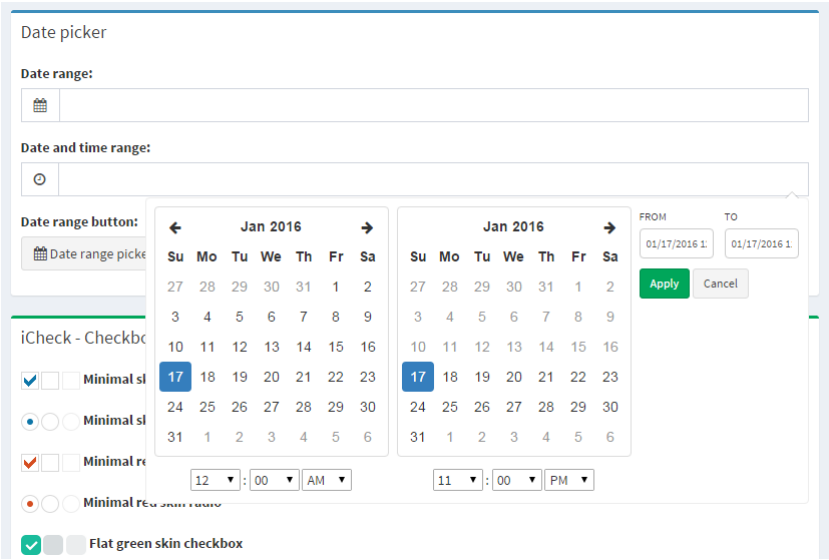

Figura 4.11: Ejemplo de formulario de rango temporal.

Tablas de datos

Las tablas de datos son fundamentales para mostrar la información de forma estructurada al usuario. Dependiendo de que tipo de información se quiera representar en estas tablas y del tamaño de los datos, es necesario pensar en las características y la configuración que tienen que tener. La figura 4.12 muestra un ejemplo de tabla simple que puede ser muy útil en el caso de visualizar las tareas del historial y su progreso. Para datos más complejos y de mayor tamaño

como es el caso de la visualización de un *log* completo, la figura 4.13 muestra una posible solución que permite funcionalidad como la búsqueda en la tabla o la ordenación por columnas.

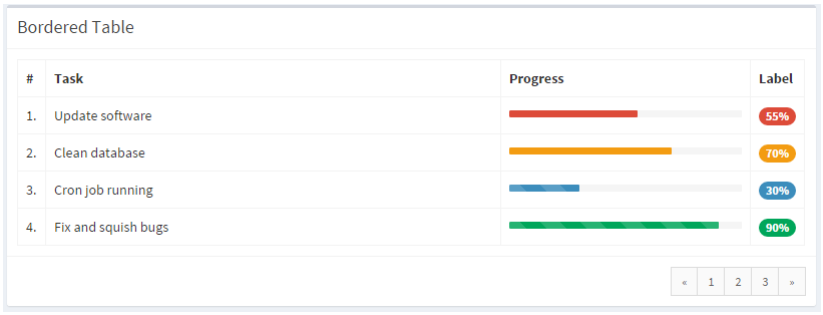

Figura 4.12: Ejemplo de tabla simple.

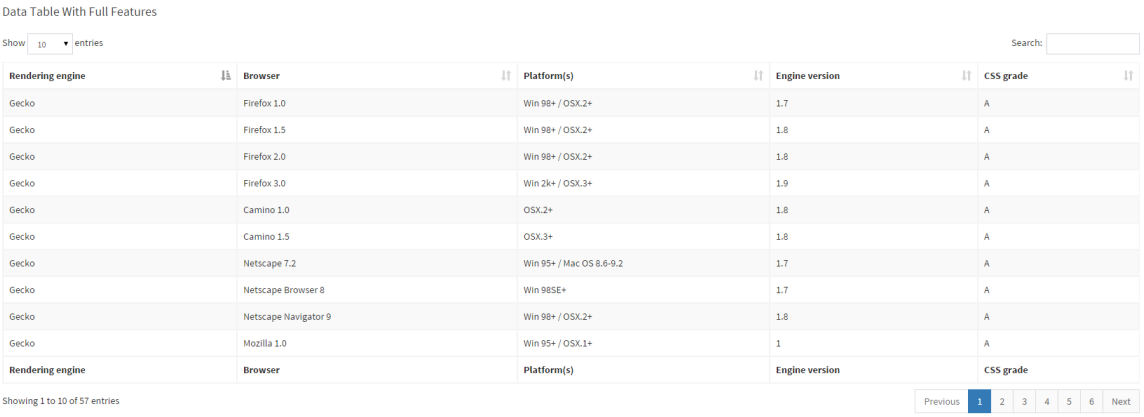

Figura 4.13: Ejemplo de tabla compleja.

Excepciones

El tratamiento de excepciones y de errores en la aplicación debe de ser informativo para que el usuario conozca las causas de ese error, e idealmente debe proporcionarle soluciones viables de forma automática o bien la navegación hacia el menú de ayuda. La figura 4.14 muestra un ejemplo de visualización de un error.

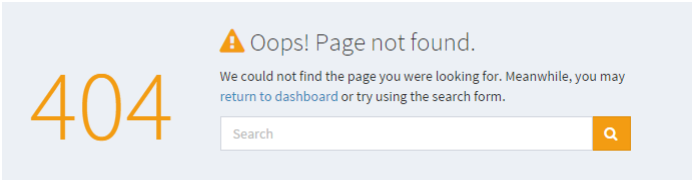

Figura 4.14: Ejemplo de visualización de una excepción.

Capítulo 5

Implementación

En este capítulo se describirá el proceso seguido para el desarrollo de la aplicación web *LogExplorer*. En primer lugar se mostrará el hardware usado para dicho desarrollo y en segundo lugar el software necesario. En tercer lugar se describirá el desarrollo del código *Java* dividido por paquetes. En cuarto lugar se mostrará el árbol de carpetas del proyecto. Y en quinto y último lugar se describirá la interfaz web desarrollada.

5.1. Hardware

El equipo *hardware* para el desarrollo y la realización de las pruebas ha sido un portátil de la marca *Toshiba* de modelo *Satellite L850-1UZ* con las siguientes características:

- **Procesador**: *Intel Core* i7-3630QM 2400 MHz.
- **Memoria RAM**: 8GB DDR3.
- **Disco duro**: 750 GB a 5400 rpm.
- **Sistema Operativo**: *Ubuntu* 14.04 LTS.
- **Arquitectura**: 64 bits.

5.2. Software

El software utilizado para este proyecto son los siguientes:

- **Herramientas de desarrollo**: La herramienta software principal para el desarrollo del proyecto *LogExplorer* es *NetBeans* [69]. Para la ejecución de pruebas en local se ha usado el servidor *GlassFish* [70]
- **Gestión de librerías**: La aplicación web es un proyecto *Maven* ya que, como hemos dicho anteriormente, se utiliza para definir las dependencias y librerías usadas por el proyecto.
- ***framework* principal**: El *framework* principal del proyecto es *Spring*, este tiene extensiones como *Spring Security*, de la que se habla más adelante, para la implementación de la autenticación y autorización de los usuarios.

- **Interfaz gráfica**: Para la interfaz web el proyecto usa páginas *JSP* y librerías *JavaScript* como *Bootstrap*. Se basa en una plantilla llamada *AdminLTE-2.2.0* de *Almsaeed Studio* para el menú, los iconos y en general el estilo completo de la interfaz. Para las pruebas de la interfaz web se ha utilizado el navegador *Google Chrome*.

- **Parseado**: El parseado se refiere a un analizador sintáctico que generalmente traduce un formato de texto a instancias de objetos de la aplicación y viceversa. Los formatos traducidos son los siguientes:

  - ***JSON***: para *parsear* datos en formato *JSON* se han utilizado dos librerías:
    - *JSON.simple* [71], que permite el acceso del los datos *JSON* como si se tratase de un árbol
    - *GSON* [72] de *Google*, que permite acceder convertir directamente el texto *JSON* a un objeto *Java* normal.
  - ***CSV***: para *parsear* el formato *CSV* se ha utilizado la librería *Super CSV* [73] que permite múltiples configuraciones y opciones de lectura y escritura de ficheros en este formato.

- **Procesamiento**: en el subsistema de procesamiento existen dos tipos complejos que requieren el uso de librerías externas. Estos son:

  - **Aprendizaje Automático**: en aquellos procesamientos que busquen aplicar aprendizaje automático, como *K-means* y *PCA*, se usa la librería gratuita *Weka* [9].
  - **Geolocalización**: Para geolocalizar direcciones *IP* se usa un programa gratuito llamado *GeoIP2* [74] de la desarrolladora *MaxMind* que utiliza una base de datos gratuita para la *geolocalización* de direcciones *IP*. Este programa devuelve, en caso de que funcione satisfactoriamente, el país donde reside dicha IP. Es posible que esta localización no sirva para conocer el país de un posible atacante ya que muchos de ellos utilizan servidores *Servidor Proxy* para ocultar su procedencia o bien desde redes de anonimato como *Tor*.

5.3. Plataformas

La complejidad de la aplicación web se demuestra al enumerar el conjunto de lenguajes de programación distintos que utiliza

- ***Java***: usada para la realización del código fuente de la aplicacción web.

- ***PostgreSQL***: usada para el desarrollo de la base de datos.

- ***JSP***: (siglas en inglés de *JavaServer Page*) usadas para realizar la interfaz de usuario. Estas páginas utilizan *JSTL* (siglas en inglés de *JavaServer Pages Standard Tag Library*) para mostrar la información llegada de los controladores de *Spring*.

- ***JavaScript***: usado en la realización de alguna funcionalidad por parte de la interfaz de usuario y de la plantilla usada, *AdminLTE-2.2.0*. Por ejemplo, en validación en el lado del cliente o para definir la configuración de los gráficos.

5.4. Codificación

Este apartado describe principalmente las partes desarrolladas del sistema. Por una parte el código *Java* divido en paquetes, por otra parte las especificaciones de seguridad mediante la configuración de *Spring Security* y por último el desarrollo de la interfaz web.

5.4.1. Paquetes del proyecto

En cada paquete java del proyecto se realizará una funcionalidad específica del sistema de la aplicación.

org.uam.logexplorer.controller

Los controladores son clases que se ocupan de responder a consultas HTTP. En *Spring* los controladores se anotan como "*@Controller*" y contienen los servicios auto-conectados mediante la anotación "*@Autowired*". La mayor parte de las funciones de los controladores están anotados como "*@RequestMapping*" que se dedican a la dar respuesta a consultas según los argumentos recogidos en esta anotación, como el valor de la *URL*, el método *HTTP* (*GET* o *POST*)... Como retorno de la función se devuelve el nombre del archivo *JSP* al que se quiere ir.

En el caso de que se quieran añadir nuevas variables que después se encargará de utilizar el *JSP*, existe un argumento del tipo "*ModelMap*" para la función, con un método "*addAttribute*". En el caso en el que el controlador se dedique a procesar un formulario para crear un nuevo modelo y tengan un validador que no sea el por defecto, es decir, una clase que implemente la interfaz *Validator* de *Spring* se tiene que definir explícitamente. Para ello se tiene que usar la notación "*@Qualifier*" para recoger el *Bean* definido en el fichero de configuración de *Spring*, *SpringDispatcher-servlet.xml* y el método "initBinder" para definir dicho validador. Como ejemplo podemos ver el código del *RegisterController* 5.1 dedicado al registro de un nuevo usuario.

Código 5.1: Código del RegisterController.

```
@Controller
public class RegisterController {

    @Autowired
    private RegisterService registerService;

    @Autowired
    @Qualifier("infoRegisterValidator")
    private Validator validator;

    @InitBinder
    private void initBinder(WebDataBinder binder) {
        binder.setValidator(validator);
    }

    @RequestMapping(value = "/register", method =
       RequestMethod.GET)
    public String addEmployee(ModelMap model) {
        InfoRegister user = new InfoRegister();
        model.addAttribute("infoRegister", user);
```

```
        return "register";
    }

    @RequestMapping(value = "/register", method =
       RequestMethod.POST)
    public String registerCont(@ModelAttribute("infoRegister")
       @Validated InfoRegister user, BindingResult result,
       ModelMap model) {

        if (result.hasErrors()) {
            return "register";
        } else if
           (this.registerService.isRegistered(user.getEmail())
           == true) {
            result.rejectValue("email", "user.email.exist");
            return "register";
        } else {
            this.registerService.register(user);
        }
        return "login";
    }
}
```

org.uam.logexplorer.core.calculus

El paquete de *calculus* es el que contiene las clases usadas para realizar los procesamientos nombrados en el subsistema de procesamiento. A estos procesamientos se les proporcionan la colección de datos y la configuración del procesamiento en formato *JSON*. Esta configuración se recogerá en la clase *CalculusManager* que instanciará cada un de los tipos de procesamiento que implementan la interfaz *CalculusInterface*. Una vez instanciado el procesamiento se ejecuta y su resultado se guarda en otra colección de datos.

org.uam.logexplorer.core.chart

El paquete de *chart* es el que contiene las clases que suministran la funcionalidad específica para la representación gráfica de los elementos gráficos creados. El funcionamiento es parecido al del paquete *calculus*. A los gráficos se les proporcionan también la colección de datos y la configuración del gráfico en formato *JSON*. Además del identificador del panel en el que se van a mostrar los resultados. La configuración se recogerá en la clase *ChartManager* que instanciará cada un de los tipos de gráfico que implementan la interfaz *ChartInterface*. Una vez instanciado el gráfico se ejecuta y su resultado se guarda en una clase llamada *ViewResult* que contiene el código *HTML* y *JavaScript* que se interpretará en el navegador web.

org.uam.logexplorer.core.exception

El paquete exception recoge las excepciones y errores propios de *Java*. Estas excepciones se manejan en *Spring* mediante un controlador especial. En el caso de que se lancen estas excepciones, el controlador se hará cargo de dar la respuesta *HTML* correspondiente. En el código 5.2 se muestra un ejemplo con la excepción *NotFoundException*.

Código 5.2: Código del RegisterController.

```
@ControllerAdvice
public class NotFoundExceptionController {

    private static final Logger logger =
       Logger.getLogger(NotFoundExceptionController.class);

    @ExceptionHandler(NotFoundException.class)
    public ModelAndView
       handleNotFoundException(HttpServletRequest req,
       NotFoundException exception){
        ModelAndView mav = new ModelAndView();
        mav.setViewName("notfound");
        return mav;
    }
}
```

org.uam.logexplorer.core.filter

Este paquete contiene las clases que proporcionan la funcionalidad descrita en el subsistema de filtros. Este paquete tiene la misma estructura que el paquete *chart*. A los filtros se les proporcionan también la colección de datos, la configuración del gráfico en formato *JSON* y el panel donde van a ser ejecutados. La configuración se recogerá en la clase *FilterManager* que instanciará cada un de los tipos de filtro que implementan la interfaz *FilterInterface*. Una vez instanciado el filtro se ejecuta y su resultado será una nueva colección de datos que podrán utilizar los gráficos o los procesamientos posteriores.

org.uam.logexplorer.core.model

Este paquete se contiene las clases del subsistema de la gestión de datos. Tales como la información de los atributos, las instancias de datos y las colecciones.

org.uam.logexplorer.core.parser

En este paquete se encuentran las clases que permiten *parsear* los distintos formatos de *logs* que la aplicación puede soportar. Cada clase se dedica a un formato exclusivamente e implementan una interfaz de este paquete llamada *Parser*. Para poder obtener la clase de parser correcta, se utiliza la clase *ParserManager* que tiene un constructor estático que depende del formato de fichero. Los parser pueden ser del tipo:

- **Pattern**: usando la librería de Java *java.util.regex*, dedicada a la implementación de expresiones regulares, podemos *parsear* la mayor parte de información de *logs* de tipo semi-estructurado describiéndolas mediante una cadena de caracteres [75]. Algunos ejemplos de *logs* que los usuarios pueden definir mediante este formato pueden ser:
  - **Dmesg**: en *Linux*, es el *log* que contiene mensajes importantes generados durante el arranque del sistema y durante la depuración de aplicaciones.
  - **Syslog**: es a la vez un *log* y un estándar de facto de *Linux* para el envío de mensajes de registro en una red informática. En el se suele registrar intentos fallidos de

autenticación con contraseñas equivocadas, accesos correctos, alertas o anomalías en el funcionamiento del sistema, actividades del sistema operativo...

- **Apache Log**: en la mayoría de aplicaciones web se usa el sistema de *Apache* de *log* [76]. La mayoría se registra en su forma semi-estructurada, aunque la librería se puede configurar para cosas como: qué se registra, cómo se registra, en qué ficheros se guarda...

- **CSV**: como se ha explicado anteriormente se utiliza la librería *Super CSV* [73], donde el usuario tiene que especificar la configuración que quiere utilizar, por ejemplo; si la primera línea son los nombres de los atributos, si el separador es una coma, un tabulador o un punto y coma...

org.uam.logexplorer.core.task

En este paquete se implementan las clases cuya funcionalidad es la de proporcionar la gestión de los hilos de ejecución. Existen tres clases en este paquete, la clase abstracta *AbstractTask*, que implementa la interfaz de *Java Runnable*. Estos hilos pueden ser de dos tipos: realizar una subida de fichero y realizar una ejecución de procesamiento.

Puesto que los ficheros se suben por internet es necesario añadir la librería *commons-fileupload* que contiene la clase *MultipartFile* En el código 5.3 se puede observar la configuración de la subida de ficheros guardada en *"SpringDispatcher-servlet.xml"*, donde se puede decidir si hay tamaño máximo de fichero, tamaño mínimo, tipo de fichero obligado...

Código 5.3: Configuración de la subida de fichero.

```
<!-- Subida de archivos -->
 <bean id="multipartResolver" class="
org.springframework.web.multipart.commons.CommonsMultipartResolver
">
 <!-- one of the properties available; the maximum file size in
    bytes
 <property name="maxUploadSize" value="100000"/> -->
 </bean>
```

La clase *TaskManager* que se ocupa de gestionar los hilos de ejecución. Para poder definir los hilos en *Spring* es necesario añadir una parte en su fichero de configuración *"SpringDispatcher-servlet.xml"* tal como se muestra en el código 5.4.

Código 5.4: Configuración de los hilos.

```
<!-- TaskExecutor MULTI-THREAD -->
<bean id="taskExecutor" class="
 org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor
 ">
 <property name="corePoolSize" value="5"/>
 <property name="maxPoolSize" value="8"/>
 <property name="queueCapacity" value="30"/>
</bean>

<bean id="taskManager"
 class="org.uam.logexplorer.core.task.TaskManager">
 <constructor-arg ref="taskExecutor" />
</bean>
```

org.uam.logexplorer.core.type

En este paquete se guardan los tipos de datos utilizados en la aplicación. Estos tipos de datos son tales como: una fecha, un número, un valor de sí o no, un texto... Sirven para parsear los valores leídos de los ficheros de *log* y todos implementan la interfaz *Type* mencionada en el subsistema de gestión de datos.

org.uam.logexplorer.core.utils

Este paquete contiene algunas clases que pueden considerarse herramientas útiles para facilitar y distribuir mejor la funcionalidad de la aplicación. Unas tienen conversores como la clase *ColorHTMLConvert* que proporciona métodos para generar colores aleatorios a los gráficos y convertir cadenas de caracteres en colores y viceversa. Y otras contienen funcionalidades en común, como es el caso de métodos estadísticos básicos para el cálculo de mínimos, máximos y desviaciones estándar o el cálculo de tendencias en series temporales aplicando el método de los mínimos cuadrados.

org.uam.logexplorer.dao

Los *DAO* (siglas en inglés de *Data Access Object*, Objeto de acceso de datos) son componentes que se ocupan de suministrar una interfaz común para el acceso de los datos ya sea para ficheros o para base de datos. Cada *DAO* tiene una interfaz cuyo nombre es el modelo al que involucran y que acaba en "*Dao*", y una clase que implementa esta interfaz, con el nombre del modelo al que involucran más "*DaoImpl*". Las clases que implementan los *DAO* tienen la anotación de *Spring* "*@Repository*" Que los identifica y permite autoconectarlos a los servicios. Cada *DAO* hace referencia al *SessionFactory*, definido en el *SpringDispatcher-servlet.xml*, que se dedica a gestionar las transacciones a la base de datos para que no haya problemas de luchas por el acceso a la base de datos. Los *DAO* tratan las consultas y los comandos de *Hibernate SQL* que posteriormente se traducen a consultas y comandos *SQL* a la base de datos de *PostgreSQL* de la aplicación. Los *DAO* tienen métodos como:

- **insert**: guarda un modelo en la base de datos.
- **delete**: elimina un modelo de la base de datos.
- **search**: busca un modelo en la base de datos según su clave primaria, devolviendo *null* en caso de que no exista.
- **update**: actualiza un modelo en la base de datos.
- **list**: Si no tiene argumentos lista todos los modelos de la base de datos. Si sí tiene, realiza la consulta lista su resultado. Frecuentemente se utiliza el argumento del identificador del usuario, ya que los usuarios solamente pueden acceder a su propio contenido.

org.uam.logexplorer.model

Este paquete contiene las clases modelo o entidad. Existe un modelo por cada tabla de la base de datos. Estos modelos son clases en las que se guarda la información de los atributos de cada instancia de su relativa tabla de la base de datos. También pueden contener restricciones de validación como tamaño máximo o modificadores de la base de datos como "*not null*" o claves externas. En el caso de que una tabla tenga una clave primaria compuesta, es decir, una clave primaria de más de un atributo, se genera otra clase con el nombre del modelo y que finaliza en "PK". Todos los modelos en *Spring* llevan la anotación "*@Entity*".

org.uam.logexplorer.service

Este paquete contiene las clases e interfaces que implementan los servicios de la aplicación, que serán usados por los hilos de ejecución de filtros y colectores, y por los controladores. Estos servicios están relacionados con la gestión final de los gráficos, los filtros y los colectores. También existen tres servicios distintos de estos: el primero es *DataService*, que implementa los métodos para mostrar los datos en las tablas, el segundo es *RegisterService* que implementa los métodos usados para registrar un nuevo usuario, y el último es *MenuService* un servicio que implementa los métodos necesarios para la construcción del menú de la aplicación que la mayor parte de las páginas utiliza. Los servicios, parecido a los *DAO*, tiene una interfaz cuyo nombre acababa en "*Service*" y su implementación, acabada en "*ServiceImpl*". Las implementaciones tienen una anotación de *Spring* tienen una anotación "*@Service*" con las que se puede autoconectar a los controladores.

org.uam.logexplorer.validator

Este paquete contiene las clases que periten la validación de los datos que posteriormente se volcarán a la base de datos. El el paquete hay dos tipos de clases, aquellas cuyo nombre termina en "*Validator*" lo que hace referencia a que es una clase validadora, y las que no, que son clases que contienen los valores dados por formularios por los usuarios y que utilizarán las clases validadoras para comprobar si son correctas. Esto es necesario para asegurar un correcto funcionamiento de la aplicación, estas clases validadoras comprueban cada argumento y su valor implementando la interfaz de *Spring*, *Validator* y guardan los posibles errores en una clase para mostrarse posteriormente en la página en cuestión. Para poder ser usados por los controladores, estas clases validadoras se configuran en el archivo de configuración de *Spring* (*"SpringDispatcher-servlet.xml"*). En el código 5.5 se muestra la configuración en cuestión.

Código 5.5: Configuración de los validadores.

```
<!-- Validator -->
<bean id="infoRegisterValidator"
   class="org.uam.logexplorer.validator.InfoRegisterValidator"/>
<bean id="collectorValidator"
   class="org.uam.logexplorer.validator.CollectorValidator"/>
<bean id="infoNewFilterValidator"
   class="org.uam.logexplorer.validator.InfoNewFilterValidator"/>
<bean id="infoNewChartValidator"
   class="org.uam.logexplorer.validator.InfoNewChartValidator"/>
```

5.4.2. Árbol de carpetas

Una de las partes clásicas en el desarrollo de una aplicación web es la definición de la organización en carpetas del proyecto. La figura 5.1 muestra el árbol de carpetas del proyecto *LogExplorer* según la visión de *NetBeans*. Cabe destacar dos partes diferenciadas, la carpeta *Web Pages* donde se guarda el contenido de la interfaz junto con varios archivos de configuración, y la carpeta llamada *Source Packages*, que contiene todos los paquetes Java definidos anteriormente. En la carpeta *Web Pages* se observa otra partición en dos sub-carpetas, *WEB-INF* y *resources*. En *WEB-INF* se guardan los archivos de configuración y las páginas *JSP* guardadas en la carpeta *views*. En la carpeta de *resources* se guardan los recursos como librerías de *JavaScript*, archivos de estilo *CSS*, imágenes, archivos *properties* como en la carpeta de *messages* donde se guardan los mensajes de error que muestra la aplicación en caso de que el usuario rellene erróneamente un formulario...

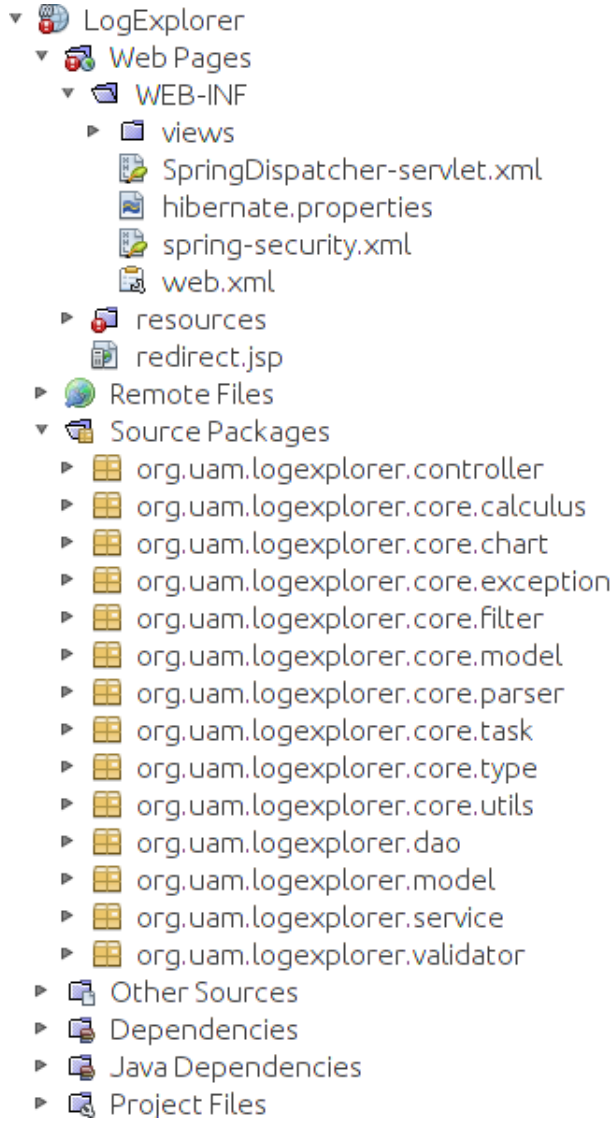

Figura 5.1: Árbol de carpetas del proyecto *LogExplorer* en *NetBeans*.

5.5. Seguridad implementada

Mediante la extensión del *Spring Framework* llamada *Spring Security* la aplicación web tiene un control de la autenticación y autorización de los usuarios del sistema. *Spring Security* añade una capa más a la aplicación y filtra según la sesión. Para ello utiliza un archivo de configuración llamado *spring-security.xml* que se muestra parte en el código 5.6. En él se puede observar el procedimiento de la autenticación, y el proceso de autorización conectando con la base de datos. Dicho archivo, además, incluye la declaración del formulario de *login* y de las páginas referenciadas según se realice una autenticación correcta, una desconexión del sistema...

Código 5.6: Configuración de *Spring Security*.

```
<http auto-config="true">
 <intercept-url pattern="/home" access="hasRole('ROLE_USER')" />
 <form-login login-processing-url="/login" login-page="/login"
  username-parameter="username" password-parameter="password"
    default-target-url="/home"
    authentication-failure-url="/login?authfail" />
   <logout logout-url="/login?logout"
      logout-success-url="/login?logout" />
   <csrf disabled="true" />
 </http>
 <authentication-manager>
  <authentication-provider>
   <password-encoder hash="sha-256"/>
    <jdbc-user-service data-source-ref="dataSource"
     users-by-username-query="select username, password,
        enabled from user_authentication where username=?"
      authorities-by-username-query="select username, authority
         from user_authorization where username =?"/>
    </authentication-provider>
   </authentication-manager>
```

Para que en la base de datos no se guarde la contraseña en texto plano, se realiza una *función hash* de la contraseña. Una *función hash* es una función resumen que sirve de representación compacta de una cadena de entrada. En una primera versión se utilizó la *función hash MD5* . Esta es una función insegura, se han detectado colisiones de *MD5* [77], lo que permitiría a un atacante acceder sin necesariamente saber la contraseña. Se sustituye por *SHA-2*. Cuando se realiza el *login* se compara con este valor de *función hash* con la base de datos. Para el registro se recoge la contraseña y se procesa la *función hash* mediante la librería *Java Security*. Es necesario destacar que este proceso también es inseguro, ya que la autenticación de usuario y contraseña se envían sin cifrar. Eso quiere decir que el cliente envía su identificador y su clave sin procesar y después es el servidor el encargado de realizar la *función hash*. Esto permite a un atacante que esté escuchando la comunicación entre servidor y cliente, saber el usuario y la contraseña y, por tanto poder hacerse pasar por él. Para resolver esto y otras inseguridades del sistema, en versiones posteriores se implementará el protocolo *HTTPS* que además proporcionaría seguridad en el envío de los ficheros *log*.

Capítulo 6

Conclusiones

6.1. Conclusiones

Como primera fase del proyecto *LogExplorer* se ha realizado una investigación de las posibles herramientas similares así como de la situación actual. Se ha realizado un análisis y un diseño, además de un inicio del desarrollo para entender el funcionamiento y los retos en la realización completa del mismo.

Primero se ha analizado el desafío *VAST Challenge 2012 MC2* donde se muestra una situación cercana a la realidad en la que, mediante ficheros *log* del *cortafuegos* y del *IDS* se consigue analizar y detectar el ataque y el comportamiento de una *botnet*. Este desafío ha servido para entender la importancia de la información de los *logs*, en la detección de ataques por parte de usuarios maliciosos y en la auditoría de la seguridad informática.

Se ha diseñado una aplicación web, *LogExplorer*, que permite centralizar la información de *logs*, procesarla y presentarla de forma gráfica para poder utilizarla en la detección de ataques informáticos. Con el fin de ganar competencias en el desarrollo de aplicaciones web, este diseño ha utilizado *Spring Framework*, un framework nunca visto hasta ahora. Se ha hecho uso de *PostgreSQL* para la realización de la base de datos que la aplicación web usa y se ha conectado con *Spring* mediante *Hibernate*. También se ha hecho uso de la extensión *Spring Security* para proporcionar los métodos propios de seguridad como la autenticación y autorización de los usuarios. En lo relativo al apartado gráfico se ha usado una plantilla gratuita llamada *AdminLTE* que hace uso de librerías en JavaScript como *Bootstrap*. Mediante la tecnología *JSP*, las páginas web mostrarían los datos y recogerían los formularios de la aplicación.

La complejidad de una herramienta de estas características es notable. Es necesario tener en cuenta en un primer lugar la falta de consenso en el formato de los *logs*. Al tener una forma de codificación distinta en cada sistema, es necesario dotar a una herramienta de métodos que puedan leer la información según qué formato y guardarla en un formato único. Otro desafío es el de dotar a la herramienta de una escalabilidad que permita añadir, de forma fácil y sin tener la necesidad de realizar grandes cambios, nuevos métodos, filtros y procesamientos, que permitan procesar la información como más convenga al usuario. Al ser una aplicación web no es posible ignorar la usabilidad y seguridad. Es necesario facilitar la interactividad así como dotar la herramienta de menús de ayuda.

6.2. Trabajo futuro

Evidentemente, el trabajo futuro empezaría tratando de realizar la segunda y tercera fase del proyecto. Estas son el desarrollo final y la validación y documentación de la herramienta. También, como se comenta en el capítulo de Implementación, existen algunas mejoras respecto a la seguridad del sistema como la implementación del protocolo *HTTPS*. Es necesario destacar algunas mejoras en el diseño que podrían suponer ventajas adicionales a una herramienta de este tipo. Debido a la gran cantidad de información y a la gran variedad de formatos que pueden tener los *logs* tiene sentido utilizar la reciente tecnología conocida como *Big Data*. En la mayoría de propuestas existentes dentro de este dominio, la solución utilizada con más frecuencia es reducir de la dimensionalidad de los datos para poder realizar un procesado de la información o extraer características que puedan facilitar al usuario la toma de decisiones [78, 79]. Dentro de *Big Data* la arquitectura más conocida es *Apache Hadoop*, que integra un sistema de ficheros distribuido *HDFS*, un paradigma de programación llamado *Map-Reduce* y un sistema de gestión de aplicaciones *YARN*. Otra extensión en la misma arquitectura podría ser el paradigma *NoSQL* que aportan un procesamiento parecido a *SQL* pero con las ventajas de distintos formatos. Uno de los más conocidos es *Apache Hive*. Esta arquitectura ya se está investigando incluso aumentando las posibles entradas, no solamente *logs*. Uno de los nombres utilizados para estos sistemas es *Security Data Lake* [80]. Otra de las mejoras a esta aplicación es implementarla para que las operaciones se realicen en tiempo real y así cubrir el marco de la monitorización de sistemas. Además de un aprendizaje automático mucho más desarrollado. En el caso de aplicarse en *Big Data* una posibilidad es *Apache Spark* que permite aprendizaje automático y tiene módulos para procesamiento en *streaming*. Uno de los puntos a mejorar en un trabajo futuro sería automatizar la recogida y *parseado* de los ficheros *logs*. Una posibilidad en este respecto es aplicar la herramienta *Apache Flume* que permite recogida de datos en *streaming*.

Bibliografía

[1] David McCandless. World's biggest data breaches. http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/, 2015.

[2] Babak Akhgar, Andrew Staniforth, and Francesca Bosco. *Cyber Crime and Cyber Terrorism Investigator's Handbook*. Syngress, 2014.

[3] Anton Chuvakin, Kevin Schmidt, and Chris Phillips. *Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management*. Syngress Publishing, 2013.

[4] Daniel Gonçalves, Joao Bota, and Miguel Correia. Big data analytics for detecting host misbehavior in large logs. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 1, pages 238–245. IEEE, 2015.

[5] Sam Curry, Engin Kirda, Eddie Schwartz, William H Stewart, and Amit Yoran. Big data fuels intelligence-driven security. *RSA Security Brief, January*, 2013.

[6] Karen Kent and Souppaya M. Guide to computer security log management. *Special Publication 800- 92 (National Institute of Standards and Technology)*, 2007.

[7] Pivotal Software. Spring. http://spring.io/.

[8] Computerworld. 10 hottest it skills for 2015, 2015. http://www.computerworld.com/article/2844020/it-careers/10-hottest-it-skills-for-2015.html.

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[10] Bootstrap. http://getbootstrap.com/.

[11] Almsaeed Studio. Adminlte control panel template, 2015. https://github.com/almasaeed2010/AdminLTE.

[12] Almsaeed Studio. https://almsaeedstudio.com/.

[13] VAST Challenge 2012. http://www.vacommunity.org/VAST+Challenge+2012.

[14] Propuesta a los Grupos Parlamentarios. Agenda digital para españa, 2012. http://www.minetur.gob.es/telecomunicaciones/es-ES/Novedades/Documents/Agenda_Digital_para_Espa~na_Propuesta_Grupos_Parlamentarios.pdf.

[15] Deborah Russell and G. T. Gangemi, Sr. *Computer Security Basics*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1991.

[16] Instituto Nacional de Estadística. Población que usa internet (en los últimos tres meses). `http://tinyurl.com/hsxce4a`.

[17] Michael Howard and David LeBlanc. *Writing secure code*. Pearson Education, 2003.

[18] Simson Garfinkel Lorrie Faith Cranor. *Security and Usability: Designing Secure Systems that People Can Use*. O'Reilly Media, Inc. `http://shop.oreilly.com/product/9780596008277.do`.

[19] F. T. Grampp and R. H. Morris. The UNIX system: UNIX operating system security. *AT&T Bell Laboratories Technical Journal*, 63(8):1649–1672, 1984.

[20] Wikipedia. Operation payback. `https://en.wikipedia.org/wiki/Operation_Payback`.

[21] Offensive Security. Kali linux. `https://www.kali.org/`.

[22] Phil Williams. Organized crime and cybercrime: Synergies, trends, and responses. *Global Issues*, 6(2):22–26, 2001.

[23] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[24] Rebecca Gurley Bace. *Intrusion detection*. Sams Publishing, 2000.

[25] J. R. Dorronsoro, F. Ginel, C. Sánchez, and C. S. Cruz. Neural fraud detection in credit card operations. *Trans. Neur. Netw.*, 8(4):827–834, July 1997.

[26] E. Aleskerov, B. Freisleben, and B. Rao. Cardwatch: a neural network based database mining system for credit card fraud detection. In *Computational Intelligence for Financial Engineering (CIFEr), 1997., Proceedings of the IEEE/IAFE 1997*, pages 220–226, Mar 1997.

[27] Steven L. Scott. Detecting network intrusion using a markov modulated nonhomogeneous poisson process. *Journal of the American Statistical Association (submitted*, 2000.

[28] Patrick L Brockett, Xiaohua Xia, and Richard A Derrig. Using Kohonen's self-organizing feature map to uncover automobile bodily injury claims fraud. *Journal of Risk and Insurance*, pages 245–274, 1998.

[29] Steve Donoho. Early detection of insider trading in option markets. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM, 2004.

[30] Jessica Lin, Eamonn Keogh, Ada Fu, and Helga Van Herle. Approximations to magic: Finding unusual medical time series. In *Computer-Based Medical Systems, 2005. Proceedings. 18th IEEE Symposium on*, pages 329–334. IEEE, 2005.

[31] Jorma Laurikkala, Martti Juhola, Erna Kentala, N Lavrac, S Miksch, and B Kavsek. Informal identification of outliers in medical data. In *Fifth International Workshop on Intelligent Data Analysis in Medicine and Pharmacology*, pages 20–24. Citeseer, 2000.

[32] Weng-Keen Wong, Andrew Moore, Gregory Cooper, and Michael Wagner. Rule-based anomaly pattern detection for detecting disease outbreaks. In *AAAI/IAAI*, pages 217–223, 2002.

[33] MJ Desforges, PJ Jacob, and JE Cooper. Applications of probability density estimation to the detection of abnormal conditions in engineering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 212(8):687–703, 1998.

[34] Robert J Streifel, RJ Marks II, MA El-Sharkawi, and I Kerszenbaum. Detection of shorted-turns in the field winding of turbine-generator rotors using novelty detectors-development and field test. *IEEE transactions on energy conversion*, 11(2):312–317, 1996.

[35] Graeme Manson, Gareth Pierce, and Keith Worden. On the long-term stability of normal condition for damage detection in a composite panel. *Key Engineering Materials*, 204:359–370, 2001.

[36] Graeme Manson, S Gareth Pierce, Keith Worden, Thomas Monnier, Philippe Guy, and Kathryn Atherton. Long-term stability of normal condition data for novelty detection. In *SPIE's 7th Annual International Symposium on Smart Structures and Materials*, pages 323–334. International Society for Optics and Photonics, 2000.

[37] Simon J Hickinbotham and James Austin. Novelty detection in airframe strain data. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 536–539. IEEE, 2000.

[38] B Boser Le Cun, John S Denker, D Henderson, Richard E Howard, W Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. Citeseer, 1990.

[39] Da Chen, Xueguang Shao, Bin Hu, and Qingde Su. Simultaneous wavelength selection and outlier detection in multivariate regression of near-infrared spectra. *Analytical Sciences*, 21(2):161–166, 2005.

[40] Christopher P Diehl, JB Hampshire, et al. Real-time object classification and novelty detection for collaborative video surveillance. In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, volume 3, pages 2620–2625. IEEE, 2002.

[41] L Douglas Baker, Thomas Hofmann, Andrew McCallum, and Yiming Yang. A hierarchical probabilistic model for novelty detection in text. In *Proceedings of International Conference on Machine Learning*. Citeseer, 1999.

[42] Vasilis Chatzigiannakis, Symeon Papavassiliou, Mary Grammatikou, and B Maglaris. Hierarchical anomaly detection in distributed large-scale sensor networks. In *Computers and Communications, 2006. ISCC'06. Proceedings. 11th IEEE Symposium on*, pages 761–767. IEEE, 2006.

[43] D Janakiram, V Adi Mallikarjuna Reddy, and AVU Phani Kumar. Outlier detection in wireless sensor networks using bayesian belief networks. In *Communication System Software and Middleware, 2006. Comsware 2006. First International Conference on*, pages 1–6. IEEE, 2006.

[44] Michael Collins. *Network Security Through Data Analysis: Building Situational Awareness*. O'Reilly Media, Inc., 2014.

[45] Snort. `https://www.snort.org/`.

[46] James P. Anderson. Computer security threat monitoring and surveillance. Technical report, Fort Washington, Pennsylvania, 1980.

[47] Urko Zurutuza Ortega. Sistemas de detección de intrusos. *Universidad de Mondragón*, pages 1–47, 2004.

[48] Carmen Torrano-Gimenez, Alejandro Perez-Villegas, and Gonzalo Alvarez. A self-learning anomaly-based web application firewall. 63:85–92, 2009.

[49] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[50] Sander Dorigo. Security information and event management, 2012.

[51] The CEE Editorial Board. Common event expression. `http://cee.mitre.org/docs/CEE_Architecture_Overview-v0.5.pdf`.

[52] Jens Kühnel. Centralized and structured log file analysis with open source and free software tools, 2013.

[53] George F Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems: concepts and design*. pearson education, 2005.

[54] Stephen Northcutt. *Network Intrusion Detection: An Analyst's Handbook*. New Riders Publishing, Thousand Oaks, CA, USA, 1999.

[55] Christopher Kruegel. *Intrusion Detection and Correlation: Challenges and Solutions*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2004.

[56] William Stallings. Network security essentials: Applications and standards (2000).

[57] OWASP. Top 10 2013. `https://www.owasp.org/index.php/Top_10_2013-Top_10`.

[58] Wei Peng, Tao Li, and Sheng Ma. Mining logs files for data-driven system management. *ACM SIGKDD Explorations Newsletter*, 7(1):44–51, 2005.

[59] Xia Ning, Guofei Jiang, Haifeng Chen, and Kenji Yoshihira. Heterogeneous log analysis, October 1 2014. US Patent App. 14/503,549.

[60] Raffael Marty. *Applied security visualization*. Addison-Wesley Upper Saddle River, 2009.

[61] Fabian Fischer, Johannes Fuchs, Florian Mansmann, and Daniel A Keim. Banksafe: A visual situational awareness tool for large-scale computer networks: Vast 2012 challenge award: Outstanding comprehensive submission, including multiple vizes. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 257–258. IEEE, 2012.

[62] Yong Cao, Reese Moore, Peng Mi, Alex Endert, Chris North, and Randy Marchany. Dynamic analysis of large datasets with animated and correlated views: Vast 2012 mini challenge# award: Honorable mention for good use of coordinated displays. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 283–284. IEEE, 2012.

[63] Chunxin Yang, Qi Liao, and Lei Shi. Investigating network traffic through compressed graph visualization: Vast 2012 mini challenge 2 award:"good adaptation of graph analysis techniques". In *2012 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 279–280. IEEE, 2012.

[64] Apache maven. `https://maven.apache.org/`.

[65] Hibernate. `http://hibernate.org/`.

[66] Wikipedia. Proxy (patrón de diseño). https://es.wikipedia.org/wiki/Proxy.

[67] Wikipedia. Observer (patrón de diseño). https://es.wikipedia.org/wiki/Observer.

[68] Postgresql. `http://www.postgresql.org.es/`.

[69] Netbeans. `https://netbeans.org/`.

[70] Glassfish. `https://glassfish.java.net/`.

[71] Json.simple. `https://code.google.com/p/json-simple/`.

[72] Google. google-gson. `https://github.com/google/gson`.

[73] Super csv. `http://super-csv.github.io/super-csv/index.html`.

[74] MaxMind. Geoip2. `http://dev.maxmind.com/#GeoIP2`.

[75] Oracle. Pattern. `https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html`.

[76] apache. Apache log files. `https://httpd.apache.org/docs/1.3/logs.html`.

[77] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In *Advances in Cryptology–EUROCRYPT 2005*, pages 19–35. Springer, 2005.

[78] Jorge Camacho, Gabriel Macia-Fernandez, Jesus Diaz-Verdejo, and Pedro Garcia-Teodoro. Tackling the big data 4 vs for anomaly detection. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 500–505. IEEE, 2014.

[79] Tieming Chen, Xu Zhang, Shichao Jin, and Okhee Kim. Efficient classification using parallel and scalable compressed model and its application on intrusion detection. *Expert Systems with Applications*, 41(13):5972–5983, 2014.

[80] Raffael Marty. *The Security Data Lake,Leveraging Big Data Technologies to Build a Common Data Repository for Security.* O'Reilly Media, Inc., Sebastopol, CA, USA, 2015.