



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

Departamento de Teoría de la Señal,
Telemática y Comunicaciones

PROYECTO FIN DE CARRERA

Sensor para recopilación y visualización de información de seguridad en nodos de una red

Moisés Gautier Gómez
Director: Gabriel Maciá Fernández

Granada, 5 de septiembre de 2016

Sensor para recopilación y visualización de información de seguridad en nodos de una red

Moisés Gautier Gómez

Palabras clave

Iptables, Seguridad informática, Paquetes, Eventos, Python, Sonda, Django, Web, Visualización

Resumen

El objetivo principal del proyecto es desarrollar un software que permita recopilar y visualizar la información generada por las aplicaciones de monitorización y control de seguridad que se ejecutan en una máquina.

La motivación del mismo surge fruto de la necesidad de monitorizar un red corporativa a través de un mecanismo de gestión automatizada de eventos (SIEM). Los pasos para la realización de este sistema se han modularizado y dividido en diferentes etapas que se desarrollarán como un todo dentro del proyecto de investigación VERITAS (<http://nesg.ugr.es/veritas/>) del Network Engineering & Security Group (NESG - <http://nesg.ugr.es/>) perteneciente al área de Ingeniería Telemática de la Universidad de Granada.

Para esta finalidad será necesario definir los pasos para la obtención de logs de una fuente de seguridad, configurar la instalación para dicha fuente, realizar un sistema de parseo de logs para extraer la información, almacenarla en un sistema persistente (base de datos) y visualizarla mediante una interfaz web en la sonda desplegada.

Por último, para comprobar la efectividad y analizar el funcionamiento de la solución software, se realizaría una demostración en directo con procesamiento real de eventos para la fuente de seguridad cuyo ámbito tiene este proyecto, que será Iptables.

Sensor for collecting and displaying information security network nodes

Moisés Gautier Gómez

Keywords

Iptables, Computer security, Packets, Events, Python, Probe, Django, Web, Visualization

Abstract

The main objective of the project is to develop a software to collect and display information generated by monitoring applications and security controls running on a machine.

The motivation arises because of the need to monitor a corporate network through a mechanism of automated event management (SIEM). The steps for the implementation of this system are modularized and divided into different stages to be developed as a whole within the research project VERITAS (<http://nesg.ugr.es/veritas/>) of the Network Engineering & Security Group (NESG - <http://nesg.ugr.es/>), that belongs to the area of Telematic Engineering of the University of Granada.

For this purpose it is necessary to define the steps to obtain logs of a security source, configure the installation for that source, perform system parsing logs to extract the information, store the data in a persistent system (database) and visualize these data via a web interface.

Finally, to test the effectiveness and analyze the performance of the software solution, a demonstration is done with actual live events processing from Iptables.

Yo, *Moisés Gautier Gómez*, alumno de la titulación *Ingeniería en Informática* de la *Escuela Superior de Ingenierías Informática y de Telecomunicación* de la *Universidad de Granada*, con DNI, autorizo la ubicación de la siguiente copia de mi Proyecto Fin de Carrera en la biblioteca del centro, para que pueda ser consultada por las personas que lo deseen.

Fdo: Moisés Gautier Gómez

Granada, 1 de septiembre de 2016

D. Gabriel Maciá Fernández, Profesor del *Departamento de Teoría de la Señal, Telemática y Comunicaciones* de la *Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación* de la *Universidad de Granada*, como director del Proyecto Fin de Carrera de *Moisés Gautier Gómez*

INFORMO:

Que el presente proyecto titulado “Sensor para recopilación y visualización de información de seguridad en nodos de una red” ha sido realizado por el mencionado alumno bajo mi supervisión, y que autorizo la defensa de dicho proyecto ante el tribunal que corresponda.

Y para que así conste, expido y firmo el presente informe en Granada a 1 de septiembre de 2016.

Fdo: D. Gabriel Maciá Fernández

Agradecimientos

Este trabajo de fin de carrera significa la finalización de mis estudios de segundo ciclo, por lo que me gustaría agradecérselo a todas las personas que en menor o mayor medida han hecho posible que me encuentre hoy en esta situación.

En primer lugar me gustaría agradecerle a mi familia su ayuda y apoyo durante estos largos años, y el tesón que han demostrado hacia mí en los momentos más difíciles.

También quiero mencionar a mis compañeros de carrera, sin su inestimable ayuda no habría conseguido lograr mi meta con una sonrisa.

Por último quiero dedicarle este proyecto a todos los interesados en la seguridad informática que tantas horas de entretenimiento pueden darte.

Licencia

Este documento ha sido liberado bajo Licencia GFDL 1.3 (GNU Free Documentation License). Se incluyen los términos de la licencia en inglés al final del mismo.

Copyright © 2016 Moisés Gautier Gómez. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Índice general

Índice general	I
Índice de figuras	V
1. Introducción	1
1.1. Contexto: Seguridad informática	2
1.2. Motivación y objetivos del proyecto	2
1.3. Estructura de la memoria	3
2. Estado del arte y tecnologías utilizadas	5
2.1. Estado del arte de herramientas para monitorizar información	5
2.1.1. Lookwise	5
2.1.2. ELK Stack	7
2.1.3. Logstash	7
2.1.4. Elasticsearch	8
2.1.5. Kibana	8
2.1.6. Análisis de la herramienta	9
2.2. SIEM	9
2.2.1. Principales características	10
2.2.2. Análisis	11
2.2.3. Otros SIEM comerciales	11
2.3. Solución tecnológica de este proyecto	12
2.4. Tecnologías utilizadas	12
2.4.1. Python	12
2.4.2. Procesos vs hilos	14
2.4.3. Django	15
2.4.4. D3js	17
2.4.5. C3js	18
2.4.6. L ^A T _E X	18
2.4.7. Reactjs	19
2.4.8. Highcharts	19
2.4.9. SQLite	20
2.4.10. Rsyslog	21
2.4.11. Logrotate	21
2.4.12. Syslog	22
2.4.13. Iptables	23
2.4.14. Django-rest	23
2.4.15. JSON	23
2.4.16. PyCharm	24

2.4.17. Bitbucket	24
2.4.18. Taiga	24
2.4.19. Git	25
2.4.20. Digital Ocean	25
2.4.21. Nginx	26
2.4.22. Dependencias Python	27
2.4.23. Versiones de las herramientas empleadas	28
3. Requisitos	29
3.1. Especificación	29
3.1.1. Requisitos funcionales	29
3.1.2. Requisitos no funcionales	30
4. Diseño	31
4.1. Diseño de la arquitectura	31
4.2. Diseño del software de cada módulo	32
4.3. Diagramas de Secuencia - Operaciones	40
4.3.1. Flujo de ejecución de la aplicación: BackEnd	41
4.3.2. Flujo de ejecución de la aplicación: FrontEnd	42
4.4. Diseño de la vista	43
4.4.1. Flujo de interacción de la vista principal	43
4.4.2. Flujo de interacción de la vista de eventos en tiempo real	44
4.4.3. Flujo de interacción de la vista de estadísticas de los paquetes	44
5. Implementación	49
5.1. Flujo de ejecución BackEnd	49
5.2. Flujo de ejecución FrontEnd	50
5.3. Configuración de la aplicación	50
5.3.1. Chef	51
5.3.2. Ansible	51
5.3.3. Docker	52
5.4. Configuración local	53
5.4.1. Logs	54
5.4.2. Iptables	55
5.4.3. Parser	57
5.4.4. Workflow	59
5.4.5. Visualización de eventos	61
6. Evaluación	63
6.1. Test: PacketEventsInformation	63
6.1.1. Método: setUp	63
6.1.2. Método: test_id_ip_source	65
6.1.3. Método: test_id_ip_dest	65
6.1.4. Método: test_id_source_port	66
6.1.5. Método: test_id_dest_port	66
6.1.6. Método: test_protocol	67
6.1.7. Método: test_id_source_mac	67
6.1.8. Método: test_id_dest_mac	68
6.1.9. Método: test_raw_info	68
6.1.10. Método: test_tag	69

6.1.11. Método: test_id	69
7. Planificación temporal	71
7.1. Tareas	72
7.1.1. Hito 1: 1 Mayo a 1 Julio 2015	72
7.1.2. Hito 2: 1 Julio a 1 Septiembre 2015	72
7.1.3. Hito 3: 1 Septiembre a 1 Noviembre 2015	72
7.1.4. Hito 4: 1 Noviembre 2015 a 1 Enero 2016	72
7.1.5. Hito 5: 1 Enero a 1 Marzo 2016	72
7.1.6. Hito 6: 1 Marzo a 1 Mayo 2016	72
7.1.7. Hito 7: 1 Mayo a 1 Julio 2016	73
7.1.8. Hito 8: 1 Julio a 19 Septiembre 2016	73
7.2. Diagrama de Gantt	74
8. Conclusiones	83
9. Apéndice 1: Instalación	85
9.1. Instalación de Django y VirtualEnv	85
9.2. Rsyslog	86
9.3. LogRotate	87
9.4. /var/log/iptables.log	87
9.5. Offset paquete PygTail	87
9.6. Rsyslog.d	88
9.7. Iptables	88
9.8. Web Server	88
9.8.1. Nginx	89
10. Apéndice 2: Tests	91
10.1. Test: Ips	91
10.1.1. Método: test_ips	91
10.1.2. Método: test_ips_hostname	92
10.1.3. Método: test_ips_tag	92
10.2. Test: Ports	93
10.2.1. Método: test_ports_tag	93
10.3. Test: Tcp	93
10.3.1. Método: test_tcp_service	94
10.3.2. Método: test_tcp_description	94
10.3.3. Método: test_tcp_id	95
10.4. Test: Udp	95
10.4.1. Método: test_udp_service	95
10.4.2. Método: test_udp_description	96
10.4.3. Método: test_udp_id	96
10.5. Test: Tags	97
10.5.1. Método: test_tags_description	97
10.5.2. Método: test_tags_tag	98
10.6. Test: Macs	98
10.6.1. Método: test_macs_mac	98
10.6.2. Método: test_macs_tag	99
10.7. Test: LogSources	99
10.7.1. Método: test_logsources_description	100

10.7.2. Método: test_logsources_type	100
10.7.3. Método: test_logsources_model	101
10.7.4. Método: test_logsources_active	101
10.7.5. Método: test_logsources_software_class	102
10.7.6. Método: test_logsources_path	102
10.8. Test: Historic	103
10.8.1. Método: test_historic_source	103
10.8.2. Método: test_historic_timestamp	104
10.8.3. Método: test_historic_events	104
10.9. Test: Events	105
10.9.1. Método: test_events_timestamp	105
10.9.2. Método: test_events_timestamp_insertion	106
10.9.3. Método: test_events_source	106
10.9.4. Método: test_events_comment	107
10.10. Test: Visualizations	107
10.10.1. Método: test_visualizations_week_month	108
10.10.2. Método: test_visualizations_week_day	108
10.10.3. Método: test_visualizations_name_day	109
10.10.4. Método: test_visualizations_date	109
10.10.5. Método: test_visualizations_hour	109
10.10.6. Método: test_visualizations_source	110
10.10.7. Método: test_visualizations_process_events	110
10.11. Test: Packet Additional Information	111
10.11.1. Método: test_packet_additional_id_tag	112
10.11.2. Método: test_packet_additional_id_packet_events	113
10.11.3. Método: test_packet_additional_value	113
Bibliografía	115
GNU Documentation Free License	121

Índice de figuras

2.1. Funcionalidades Lookwise	6
2.2. ELK Stack [55]	7
2.3. Logstash description	7
2.4. Diagrama de flujo interno de logstash [55]	8
2.5. Sistema SIEM multivariante distribuido - Proyecto VERITAS	10
2.6. Comprobación dentro de la pila de Threads para el objeto de la clase Iptables	15
2.7. Cabecera de la plantilla Master que heredarán el resto de plantillas/vistas de la aplicación	16
2.8. Contenido de la plantilla index que hereda funcionalidades de la plantilla Master.	17
2.9. Ejemplo de D3js	18
2.10. Ejemplo de visualización para la biblioteca Highcharts	20
2.11. Esquema de Rsyslog	21
2.12. Configuración de iptables para Logrotate	22
2.13. Backlog del proyecto siguiendo un SCRUM	25
2.14. Droplet desplegado en digital ocean	26
2.15. Configuración de nginx en digital ocean	27
4.1. Arquitectura interna del software	31
4.2. Arquitectura: Assets	32
4.3. Diagrama de clases: Assets	33
4.4. Arquitectura: Monitor	34
4.5. Diagrama de clases: Monitor	34
4.6. Arquitectura: BD	35
4.7. Diagrama de clases para la BD (usando ORM)	36
4.8. Ejemplo de clase ORM, en concreto PacketEventsInformation	37
4.9. Arquitectura: Visualizaciones	38
4.10. Clase Visualization para el paquete ReactJS	38
4.11. Paquete Visualizations	39
4.12. Paquete Events	39
4.13. Paquete Info	40
4.14. Diagrama de Secuencia para la parte BackEnd	41
4.15. Diagrama de Secuencia para la parte FrontEnd	42
4.16. Diagrama de Secuencia para la vista principal	43
4.17. Diagrama de Secuencia para la vista de eventos en tiempo real	44
4.18. Diagrama de Secuencia para la vista de estadísticas de los paquetes	44
4.19. Vista principal de la Web	45
4.20. Vista principal con la información de eventos del día seleccionado	46
4.21. Vista principal con la información adicional por cada evento seleccionado	47
4.22. Vista principal con las estadísticas de los eventos del día seleccionado	48

5.1.	Diagrama de clases BackEnd	49
5.2.	Flujo de ejecución del FrontEnd - [15]	50
5.3.	Arquitectura de un sistema bajo Chef	51
5.4.	Diagrama de la arquitectura Ansible	52
5.5.	Diagrama de la arquitectura general de Docker	53
5.6.	Configuración de iptables para Rsyslog	53
5.7.	Configuración de iptables para Logrotate	54
5.8.	Configuración de iptables.conf para Rsyslog.d	54
5.9.	Ejemplo de regla iptables	54
5.10.	Configuración reglas iptables	55
5.11.	Evento de ssh localhost en el sistema	56
5.12.	Log capturado y almacenado por rsyslog en /var/log/iptables.log	56
5.13.	Procesamiento del log capturado y almacenado en la bd interna de la aplicación	57
5.14.	Instancia de la clase Pygtail y lectura de las líneas del log	57
5.15.	Permisos de los archivos iptables.log e iptables.log.offset	58
5.16.	Uso del método split sobre la entrada de lineas de log	58
5.17.	Obtenemos la tupla key=>value para cada etiqueta del log	58
5.18.	Vamos asignando cada etiqueta y su valor a su asociado del ORM	59
5.19.	Vista general	61
7.1.	Panel de actividades - Taiga	71
7.2.	Hito 1 - 7.1.1	74
7.3.	Hito 2 - 7.1.2	75
7.4.	Hito 3 - 7.1.3	76
7.5.	Hito 4 - 7.1.4	77
7.6.	Hito 5 - 7.1.5	78
7.7.	Hito 6 - 7.1.6	79
7.8.	Hito 7 - 7.1.7	80
7.9.	Hito 8 - 7.1.8	81
9.1.	Configuración de nuestro entorno virtual de Python	85
9.2.	Instalación de las dependencias del proyecto	85
9.3.	Configuración de la base de datos y creación del super usuario	86
9.4.	Configuración para filtrado de eventos de Iptables	86
9.5.	Permisos a la creación de archivos	86
9.6.	Timestamp más preciso	86
9.7.	Configuración de LogRotate	87
9.8.	Creación del archivo base de iptables.log	87
9.9.	Creación del archivo offset de PygTail	87
9.10.	Creación del archivo de configuración iptables para el demonio Rsyslog	88
9.11.	Reglas de Iptables usadas para el proyecto	88
9.12.	Instalación del paquete Nginx	89
9.13.	Configuración del servidor web en Nginx	89
9.14.	Enlace simbólico a nuestra configuración previa	90
9.15.	Comprobación de sintaxis de Nginx	90
9.16.	Ejecución del servidor de Django	90

Capítulo 1

Introducción

*Estar preparado para la guerra
es uno de los medios más eficaces
para conservar la paz
George Washington*

En el siglo XXI todo pasa por ser digital y si no, ya es que lo era antes de llegar a este punto. Quizás muchas de las tecnologías que hoy conocemos se basen en un sistema informatizado, ya sea en su formato de software o sistema embebido. Y es que todo pasa por ser una herramienta software diseñada para un propósito en concreto: un mecanismo de apertura de puertas mediante tarjetas RFID, procedimientos industriales o aplicados a alguna infraestructura crítica o de bien común, una aplicación del tiempo en tu terminal móvil, etc.

Hay un sinfín de aplicaciones software que hacen nuestro día a día más llevadero y más fácil. Pero hay un punto que no todos conocen y es la necesidad de saber como se ha creado ese producto o cómo funciona realmente por su interior. En ese interior, a veces, podemos encontrar cosas que no estaban predestinadas a tener ese comportamiento y debido a ese comportamiento anómalo o imprevisto se generan situaciones de incertidumbre en las que el ser humano debe estar capacitado para afrontarlas. Dichas situaciones se suelen conocer con el término anglosajón de “bug” y sobre los bugs hay una especial categoría que se denominan fallas de seguridad o *critical/several bugs*.

Estas situaciones no previstas provocan que nuestro sistema, sea el que fuese, actúe de forma inesperada ante un input de información permitido o legítimo permitiendo un uso inadecuado de los recursos a los que se acceden mediante la aplicación. De este concepto o problema, surge en gran medida, el término de seguridad informática el cual intenta abarcar y dar solución a estos problemas que pueden ir desde un simple fallo de desarrollo, a un fallo crítico que comprometa la seguridad o confidencialidad de los documentos de una empresa o gobierno.

Debido a esta problemática, surge la necesidad de analizar, monitorizar y generar sistemas de seguridad perimetral que permita a las empresas ver que tipo de tráfico interno se genera, que tipo de tráfico externo tiene y cómo se hace uso de él (navegación hacia el exterior, tuneles VPN, conexiones remotas a dispositivos, etc). Así pues, se podría decir que para obtener este tipo de eventos sobre protocolos, tráfico, DNS, IPs, VPNs,.. se tienen que configurar dispositivos de seguridad para la recolección de estos tipos de inputs o fuentes.

Una de las fuentes más conocidas dentro del mundo de la informática es el firewall, pero no así de las de un uso más extendido dentro del mundo doméstico sino el comercial o corporativo. Y dentro de los muchos tipos de software enfocados a tráfico (firewall) se encuentra el paquete de las distribuciones GNU/Linux: Iptables (software fruto del proyecto Netfilter para el kernel de GNU/Linux). Con esta herramienta se pueden definir políticas de filtrado de tráfico para cualquier tipo de protocolo TCP/UDP que queramos limitar entre el exterior y nuestra máquina y viceversa. Además, estas políticas nos permiten derivar dicho tráfico a archivos que podemos manipular obteniendo así los eventos que representan al tráfico generado por una máquina conectada a una red, que posteriormente podemos manipular para generar estadísticas o tipos de uso para una red.

1.1. Contexto: Seguridad informática

La seguridad informática es el proceso de mantener un aceptable nivel de percepción frente a un riesgo. Así pues ninguna organización se puede considerar “segura” en cualquier momento, más allá de la última comprobación que se realizó dentro de su política de seguridad.

Un proceso seguro se encuadra dentro de las siguientes 4 etapas: Evaluación, Prevención, Detección y Respuesta:

- Evaluación: es la preparación para las otras 3 etapas. Se considera como una acción separada porque se relaciona con las políticas, procedimientos, leyes, reglamentos, presupuestos y otras funciones de gestión, además de la propia evaluación técnica enfocada a la seguridad. No tener en consideración estos supuestos anteriores, podría dañar las etapas del diseño.
- Prevención: es la aplicación de contramedidas para reducir la probabilidad de tener una situación comprometida.
- Detección: es el proceso de identificación de intrusiones. Una intrusión se puede considerar como una violación de una política de seguridad o como un incidente de seguridad a nivel de software/dispositivo.
- Respuesta: es el proceso de validar los inputs recogidos por la detección para tomar medidas que solucionen las intrusiones. El primer enfoque que debemos realizar consiste en restaurar la funcionalidad dañada y seguir recopilando información para tener claras las evidencias del atacante sobre nuestro sistema y poder así emprender las acciones legales que correspondan.

De estas etapas anteriores haremos hincapié en la etapa de detección que es en la que se enmarca el ámbito de este proyecto. Dado que el principal objetivo es facilitar la monitorización de la información de seguridad.

1.2. Motivación y objetivos del proyecto

El objetivo principal del proyecto es desarrollar un software que permita recopilar y visualizar la información generada por las aplicaciones de monitorización y control de seguridad que se

ejecutan en una máquina.

La motivación del mismo surge fruto de la necesidad de monitorizar una red corporativa a través de un mecanismo de gestión automatizada de eventos. Los pasos para la realización de este sistema se han modularizado y dividido en diferentes etapas que se acometarán como un todo dentro del proyecto de investigación VERITAS (<http://nesg.ugr.es/veritas/>) del Network Engineering & Security Group (NESG - <http://nesg.ugr.es/>) perteneciente al área de Ingeniería Telemática de la Universidad de Granada.

1.3. Estructura de la memoria

En cuanto a la estructura de esta memoria del proyecto final de carrera, tras este capítulo dónde se presentan los objetivos y la visión en general del proyecto, se expone el estado del arte y el análisis de requisitos previos al desarrollo software.

En el capítulo siguiente, veremos la etapa del diseño de software así como una posterior evaluación del mismo.

Finalmente, se presentan las conclusiones generales obtenidas una vez realizado el proyecto, y también la planificación del mismo y estimación de costes.

Además, se presentan las referencias bibliográficas donde se incluyen las fuentes consultadas para la elaboración de este proyecto, un resumen que engloba las generalidades fundamentales de la aplicación, una guía de utilización (manual de usuario), una guía de instalación, un compendio del software utilizado para el desarrollo y otro de los lenguajes de programación, y finalmente, la licencia completa del documento.

Capítulo 2

Estado del arte y tecnologías utilizadas

2.1. Estado del arte de herramientas para monitorizar información

En primer lugar, se realiza un estado del arte de herramientas que pudieran realizar funcionalidades parecidas a las que se van a implementar en este proyecto.

Actualmente existen diversos tipos de herramientas que realizan tareas de recolección, filtrado y gestión de eventos dentro un sistema. Las que se han podido analizar y comprobar han sido las siguientes:

2.1.1. Lookwise



Lookwise es una herramienta corporativa que hace las funciones de SIEM en materia de gestión de seguridad, Big Data y cumplimiento normativo (ISO/LOPD y PCI DSS).

Características

Gestión centralizada

- Interfaz gráfica de administración y operación centralizada.
- Creación y distribución de políticas de forma remota.
- Integración de alertas y explotación de resultados.
- Cuadro de mandos de seguridad.
- Visibilidad en base a roles y permisos.
- Integración con sistemas SIEM.

Comunicaciones

- Autenticadas y cifradas

- Ininterrumpidas
- Comprimidas

Plataformas soportadas

- Familia Windows XP (Windows Kernel 5)
- Familia Windows 7 (Windows Kernel 6)

Arquitectura

- Arquitectura Distribuida.
- Arquitectura Modular.
- Flexible y escalable.
- Balanceo de carga.
- Despliegue remoto de funcionalidades y actualizaciones.

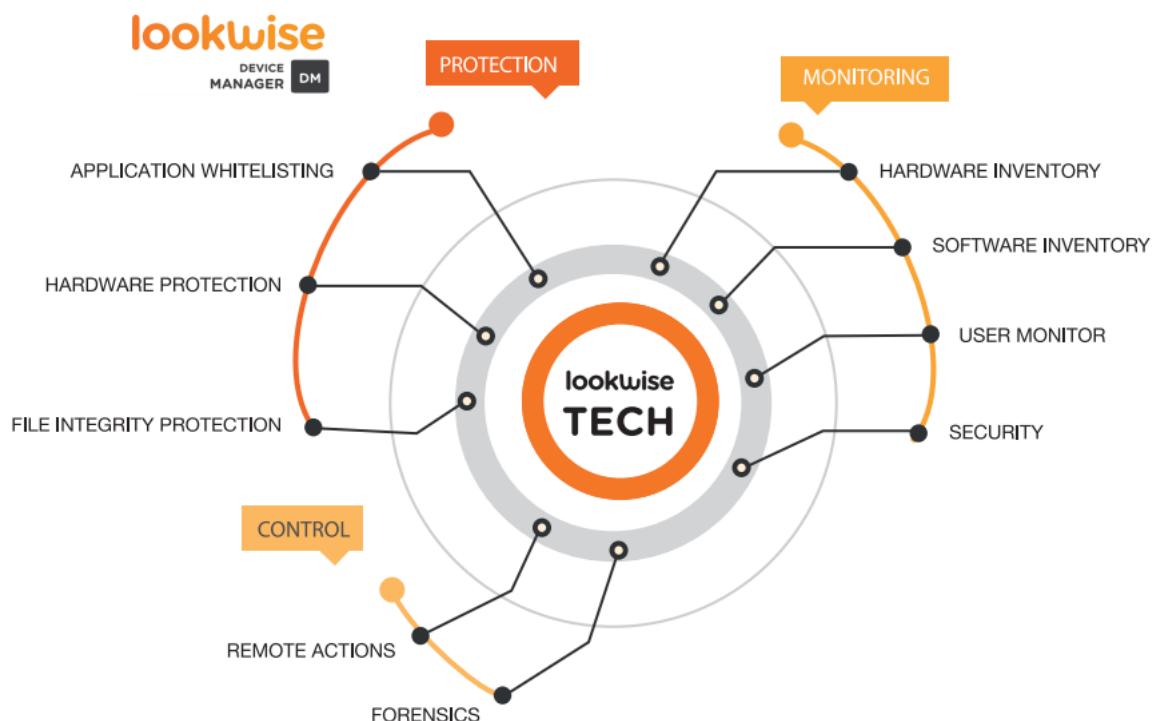


Figura 2.1: Funcionalidades Lookwise

Análisis de la herramienta

La herramienta de análisis de incidencias y vulnerabilidades hace la mismas funcionalidades de un SIEM pero con una capa más enfocada al cumplimiento del estándar de seguridad de datos de cuentas bancarias - PCI DSS (Payment Card Industry Data Security Standard -

<https://es.pcisecuritystandards.org/minisite/en/>) y de infraestructuras críticas. Gestiona eventos del sistema y los correla con alertas predefinidas internamente o que se hayan incluido cómo especificación del cliente. Detecta fallos en el Active Directory y nos genera un sistema de informe con las incidencias más graves de cara a la parte de consultoría de incidentes por parte el equipo interno.

2.1.2. ELK Stack

La pila ELK se basa en una solución open-source de tres productos bien diferenciados que se relacionan entre sí (en resumidas cuentas se trata de un SEM) de la siguiente manera:

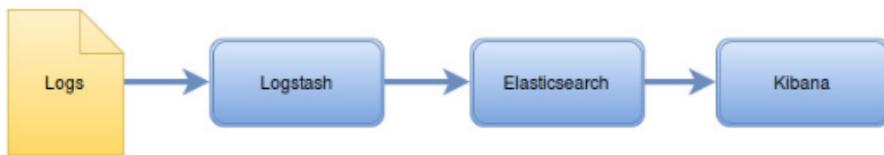


Figura 2.2: ELK Stack [55]

- Logstash para el registro centralizado de logs y su posterior normalización y enriquecimiento de datos.
- Elasticsearch para la búsqueda de datos y análisis en profundidad de un sistema.
- Kibana como herramienta de visualización de los datos recolectados y procesados anteriormente según las especificaciones que queramos para el filtrado.

2.1.3. Logstash

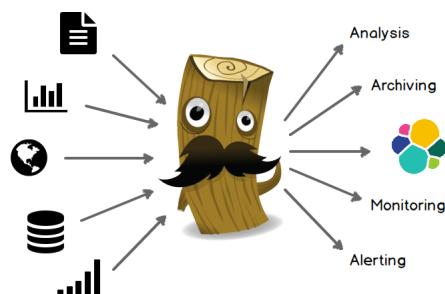


Figura 2.3: Logstash description

Logstash es un motor open-source de recopilación de datos con capacidad de multihebrado en tiempo real. Esta herramienta puede unificar dinámicamente datos de diferentes fuentes y normalizar dichos datos para los outputs de nuestra elección. Además nos permite filtrar y

discretizar todos los datos recolectados para obtener información analítica que pueda ser visualizada.

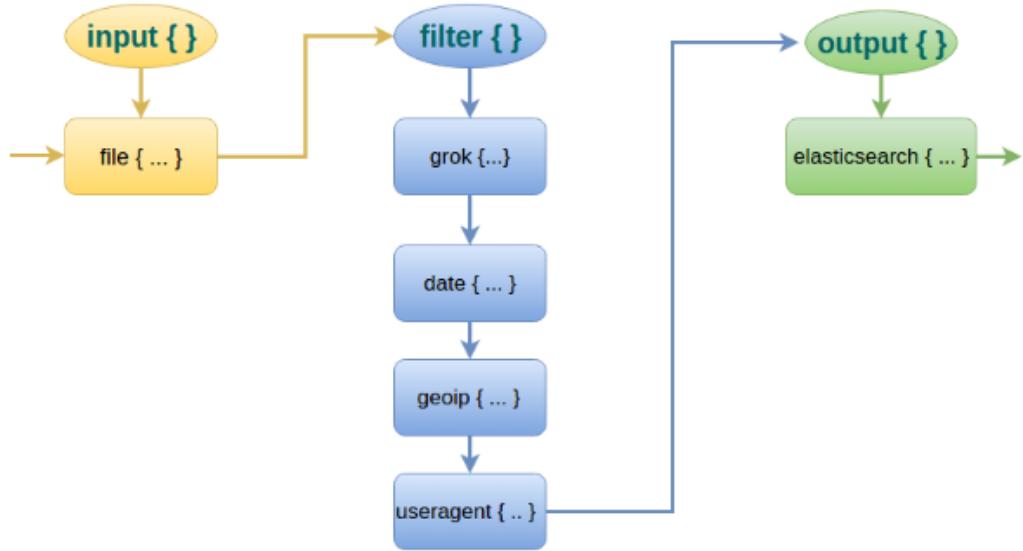


Figura 2.4: Diagrama de flujo interno de logstash [55]

Con logstash, cualquier tipo de evento puede ser enriquecido y transformado con un amplio repertorio de inputs/filtros y los datos de salida, se pueden modificar dependiendo del tipo de software sobre el que queramos introducir esos datos (plugins y codecs).

2.1.4. Elasticsearch



Elasticsearch es un motor open-source de búsqueda y análisis de información de gran escalabilidad. Esta herramienta permite almacenar, buscar y analizar grandes volúmenes de datos de forma rápida y en tiempo real. Se suele utilizar como motor/tecnología subyacente de otras aplicaciones (wrappers), permitiendo así realizar funciones de búsqueda complejas de una manera más ágil.

2.1.5. Kibana



Kibana es una plataforma open-source de análisis y visualización de datos diseñada para trabajar con Elasticsearch. Kibana se utiliza para buscar, ver e interactuar con datos almacenados en

los índices o base de datos de Elasticsearch. De esta forma se puede realizar fácilmente un análisis avanzado de datos que permita visualizarlo en una gran variedad de gráficas, tablas y mapas.

Kibana hace que sea fácil de entender y procesar grandes volúmenes de datos. Mediante su interfaz basada en un cliente web, nos permite crear de forma simple y ágil filtros sobre los datos extraídos mediante consultas a la bd de Elasticsearch en tiempo real.

2.1.6. Análisis de la herramienta

La pila ELK, como bien se ha comentado en los tres puntos anteriores, nos permite recolectar, procesar y correlar cualquier tipo de logs que genere nuestro sistema de una manera visual, ágil y fácil de entender.

El motor de indexación de datos, Elasticsearch, nos permite obtener una ejecución en tiempo real de los logs del sistema así como poder escalar dicho volumen de datos dependiendo de la situación. El único inconveniente de esta herramienta es que el sistema de referenciación de documentos internos es mediante json. Es un formato muy versátil pero incapaz de tener funcionalidad por si sola.

Después tenemos Logstash que es el encargado de hacer de middleware entre elastic y kibana, es decir, la parte de la recogida de muestras/eventos y la parte donde se visualizan esas muestras. Logstash hace de filtro y de motor de normalización de datos entre los diferentes activos que tiene asignados para así poder tener todos los datos unificados según la especificación que queramos dar a cada uno de ellos.

Cada parte de la pila esta desarrollada en su propio lenguaje de desarrollo, siendo Java (o Groovy) el lenguaje de desarrollo de elastic, Ruby el de logstash y Javascript para Kibana. El único inconveniente que se ha podido observar es que la solución ELK está diseñada para trabajar más optimizada en entornos de cloud computing y no de manera local en un servidor. Dado que tendría que usar recursos propios de la máquina y conforme se vayan escalando nuevos recursos estos irán aumentando los de la máquina. Si hay limitación de hardware por los mismos puede llegarse a experimentar un cuello de botella entre elastic y kibana, siendo la carga de datos muy lenta y con tiempos de refresco bastante altos.

2.2. Sistemas de gestión de información y eventos de seguridad: SIEM

Un sistema de gestión de información y eventos de seguridad, es un concepto que se suele usar para referirse a dos tecnologías que unifican el concepto como son SEM y SIM.

SEM (Security Event Management), tiene como principal funcionalidad la de recolectar, procesar y almacenar los eventos de seguridad que ocurren en una máquina; y SIM (Security Information Management) se suele usar para la correlación de eventos y obtención de una normalización de los mismos para un posterior análisis, procesamiento mediante algún tipo de técnica estadística o matemática, generando de informe sobre los datos que se han obtenido en

la etapa del SEM, etc.

Así pues, el término SIEM haría referencia a la recopilación de eventos de seguridad y otros relacionados para su documentación y análisis. La mayoría de estos sistemas trabajan mediante la implementación de varios agentes en una tipología jerárquica en la que cada nivel de la jerarquía cumple con un cometido a menor escala dentro del sistema de seguridad definido (máquinas de usuarios, servidores, equipos en red, firewalls, antivirus, etc). Los agentes reenvian dichos eventos a una consola de gestión centralizada, que realiza inspecciones sobre los mismos y asigna la criticidad correspondiente.

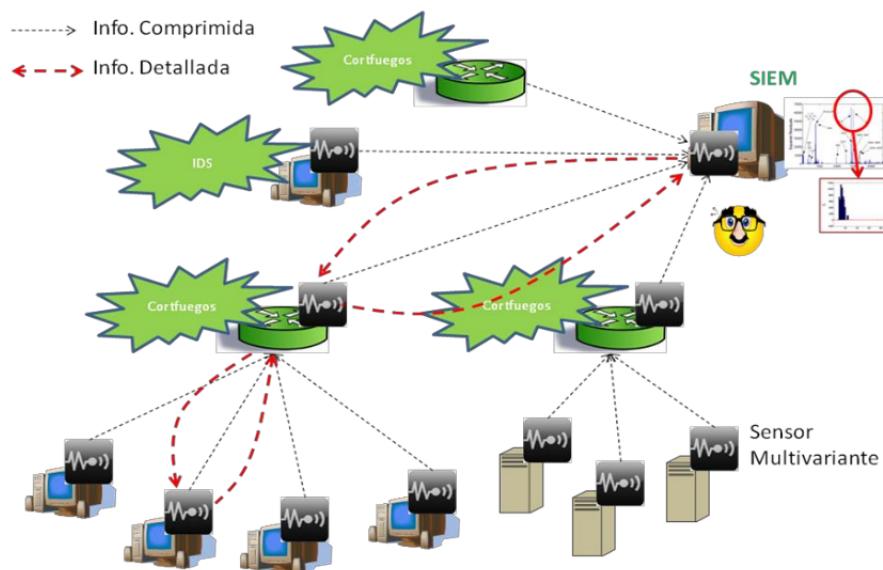


Figura 2.5: Sistema SIEM multivariante distribuido - Proyecto VERITAS

2.2.1. Principales características

Estos son los puntos que principalmente gestiona un SIEM:

- Gestión de parches (actualizaciones) del kernel o software de terceros como adobe, java, etc.
- Antivirus en máquinas de usuarios o en servidor.
- Gestión de cortafuegos.
- Integración con Active Directory (LDAP).
- Sistema de prevención de intrusiones (en red: NIPS / basado en hosts: HIPS)
- Proxy / Filtro de contenidos
- Email: anti-spam / anti-phishing

- Análisis de vulnerabilidades
- Herramientas de seguridad opcionales:
 - ◊ IPS para redes Wifi.
 - ◊ Control de firewall web.
 - ◊ Aplicaciones que monitoricen bases de datos.
 - ◊ Prevención de pérdida de datos.
 - ◊ Gestión de riesgos y herramienta de cumplimiento de políticas

2.2.2. Análisis

Aunque esta herramienta sea como un conglomerado de aplicaciones o herramientas de detección y análisis, su puesta en marcha no es así tan fácil como cabría esperarse dado que cada despliegue requiere de unos activos o fuentes diferentes. Además, cada solución final requerirá de unas especificaciones distintas, con lo que su implantación depende en gran medida de la facilidad de adaptación al entorno y también de que los responsables de dicha herramienta tengan un profundo conocimiento sobre ella.

Una de las palabras más comunes que definen la implementación de un SIEM es: desalentador. A menudo termina costando más de lo previsto, requiere una experiencia sobre la herramienta que a menudo suele ser externalizada (sobre el propio fabricante) y puede llevar un tiempo considerable antes de obtener resultados tangibles.

Los motivos para los que generalmente se introduce este sistema en la red corporativa suelen ser varios, pero entre los más destacados suelen ser: el cumplimiento de una normativa industrial o de un gobierno, la gestión de incidentes recurrentes de seguridad o también que en la licitación de un contrato esté contemplado la puesta en marcha de este sistema para una mayor calidad del servicio prestado por un tercero o por la propia entidad. Y aún así, que la propia empresa ya gestione sus eventos internos o los monitorice, no implica que su migración al SIEM sea inmediata.

Además, la licitación o adquisición de un producto de estas características supone que el entorno sobre el que se va a aplicar contiene dispositivos de seguridad que monitorizar (si no se está realizando dicho control), que se dispongan de herramienta de centralización de datos (físicamente o en cloud) y que se quiera un sistema de gestión 24/7, incluido la monitorización fuera de horario laboral.

Son estas razones por las que a veces un sistema tan complejo y gigantesco puede que genere un sobrecoste o cubra pocas áreas de la red en las que no se tiene necesidad de vigilar. Siendo esto un inconveniente finalmente, dado que hay otras herramientas (de las que se nutre el SIEM) que ya cumplen con dicha funcionalidad a coste de tener un sistema muy pesado que gestionar.

2.2.3. Otros SIEM comerciales

Existen otras muchas más herramientas, comerciales en su gran mayoría, que tratan de abordar el concepto de SIEM a un paso más allá de lo anteriormente analizado. A continuación se van a citar algunas de esas herramientas:

2.3. SOLUCIÓN TECNOLÓGICA DE ESTE PROYECTO

- **OSSIM** - AlienVault: Herramienta SIEM de código abierto.
- **HPE Arsight SIEM** - HP: Herramienta SIEM privativa.
- **IBM Security QRadar SIEM** - IBM
- **Splunk**
- **LogRhythm**

2.3. Solución tecnológica de este proyecto

La principal diferencia que existe entre este proyecto final de carrera y las herramientas anteriormente analizadas es que el desarrollo y gestión de la información no se hace de forma analítica sino que se gestionan cantidades de datos (en la fase de recolección de logs de dispositivos) y estos no reciben ningún tipo de filtro o procesamiento previo. Se transportan desde el dispositivo monitorizado a otro en donde se correlan los distintos logs o recolecciones según patrones definidos por los ingenieros de seguridad de la compañía encargada de detectar anomalías en los mismos.

No existe ningún tipo de mecanismo analítico de esta información dado que el punto de inflexión o la inteligencia, está en las etapas finales y no en las previas del tratamiento de la información. Para dotar de una mayor funcionalidad a estos sistemas hay extensiones o módulos que desarrollan las compañías que amplian la capacidad analítica de la misma por una cantidad económica bastante alta y su uso suele estar supeditado a licencia.

Además, la visualización de los eventos almacenados no siempre es inmediata y necesita de otra aplicación externa o módulo (funcionalidad extra) que explote estos recursos y los haga fácil para la interacción humana.

2.4. Tecnologías utilizadas

A continuación se detallan las diferentes tecnologías/bibliotecas/lenguajes que se han empleado para la elaboración del proyecto y por qué se han escogido por encima de otras posibles soluciones.

2.4.1. Python



Web: <https://www.python.org/>

Python es un lenguaje multiparadigma cuya estructura principal está íntegramente basada en objetos con sus diferentes métodos y atributos internos. Además también posee tipado dinámico, recolector/administrador de la memoria interna y un sistema de referenciado interno de atributos.

Las principales características que han favorecido su elección fueron:

- Es un lenguaje que facilita la implementación de una forma muy ágil y dinámica.
- Su sistema de paquetes es muy intuitivo y ligero. Se puede instalar cualquier dependencia descargandola del repositorio de paquetes PyPi en el que se pueden encontrar infinidad de soluciones software dependiendo de lo que necesites. Si no, siempre se puede definir el propio paquete software y subirlo al repositorio.
- Es de licencia similar a la BSD (Python Software Foundation License) compatible con la GNU GPL. Por lo tanto se puede modificar cualquier aspecto del kernel del lenguaje o mejorar cualquier funcionalidad anteriormente definida.
- Permite la fácil portabilidad entre plataformas, ya que sólo requiere de un intérprete que traduzca dicho código fuente a lenguaje máquina por lo que no existe una pesada fase de compilación por parte del sistema.
- Tiene una amplia comunidad de desarrolladores y es muy fácil de aprender en un corto periodo de tiempo para alguien iniciado.
- Es ampliamente empleado en el mundo de la seguridad informática para ser usado en parsers, procesadores de eventos y usos con la web como principal reclamo (Protocolos, paquetes, tráfico, etc).

Python frente a otras soluciones

En la fase de estudio de tecnologías, se planteo la posibilidad de desarrollar el proyecto con el lenguaje de programación Java pero se descarto por los siguientes aspectos:

- Precisa de unos conocimientos más avanzados sobre el control de procesos e hilos aunque sea más eficiente, también es más pesada la ejecución de los mismos en un sistema concurrente que puede que tenga muchos activos o fuentes que usar.
- Lenguaje fuertemente tipado y muy estructurado.
- Precisa de una compilación previa que pudiera incrementar los costes de empaquetar dicha solución y además solo se puede ejecutar en un entorno donde exista una JVM o java virtual machine.
- Para adaptar la solución obtenida a un resultado web tendría que hacer uso de un framework o IDE que ayudase a la hora de la gestión y diseño de la interfaz web así como la comunicación. En Python frameworks como Django, facilitan la tarea del despliegue en formato web y lo más cercano que se le parece es el lenguaje Groovy sobre el que se basa el software Elasticsearch.

2.4.2. Procesos vs hilos

En el proceso de desarrollo del proyecto hubo varios momentos en los que se valoró y estudió las diferencias entre un desarrollo software basado en procesos, a la hora de cada nuevo input que llegase a la monitorización, frente hilos o threads. A continuación, se hará un breve resumen sobre las diferencias entre ambos:

Procesos

Un proceso, se basa en la parte de un programa que se ejecuta en nuestro sistema, es decir, un conjunto de recursos reservados del sistema.

Hilos

Un hilo o thread, es similar a un proceso dado que hace uso de unos recursos reservados del sistema. Pero con una gran diferencia, porque si los procesos ocupan diferentes espacios de memoria, los hilos comparten ese espacio entre ellos.

Problemas de los hilos

La normal general es que un conjunto de hilos o procesos tiendan a compartir información entre ellos. Por lo que la solución de los hilos a priori parece ser la más adecuada dado que compartir información será mucho más fácil. Sin embargo, la compartición de grandes cantidades de información y haciendo un uso amplio de tareas concurrentes, pueden producir dos situaciones delicadas: el bloqueo mutuo (deadlock) y la condición de carrera (race condition).

Hilos del kernel vs hilos del usuario

Diferentes hilos comparten un mismo PID mientras que diferentes procesos, poseen sus propios PID. Sin embargo, esto no sucede a nivel de kernel. Los hilos del kernel tienen sus propios PID, debido a la forma en la que el kernel es ejecutado.

El kernel (para sistemas GNU) en sí mismo no se ejecuta como un proceso sino que sus tareas se ejecutan como parte de otros procesos. Debido a la gran cantidad de tareas que se ejecutan, en el kernel se realiza una implementación o acción alternativa para operar de forma similar a los procesos (esto es a lo que se le conoce como demonios).

Solución que aplica Python

A la hora de la implementación para comprobar que hilos se están ejecutando a través el hilo padre, se utiliza el método de la clase Thread enumerate.

Lo que se realiza en esa comprobación es que dentro de la pila o lista de thread que se han lanzado haya alguna coincidencia de objetos de la clase Iptables y si la hay ya existe un hilo previo en ejecución que hace las comprobaciones pertinentes.

```

1 for threads in threading.enumerate():
2
3     test = Iptables(
4         args=(1,),
5         source={'T': 'Firewall', 'M': 'iptables', 'P':
6             '/var/log/iptables.log',
7                 'C': './secapp/kernel/conf/iptables-conf.conf'}
8     )
9     if type(threads) == type(test):
10        exist_thread = True
11
12 if not exist_thread:
13     thread_ipTables = Iptables(
14         args=(1,),
15         source={'T': 'Firewall', 'M': 'iptables', 'P':
16             '/var/log/iptables.log',
17                 'C': './secapp/kernel/conf/iptables-conf.conf'}
18     )
19     thread_ipTables.start()

```

Figura 2.6: Comprobación dentro de la pila de Threads para el objeto de la clase Iptables

2.4.3. Django



Web: <https://www.djangoproject.com/>

Django es un framework web de alto nivel para el lenguaje de programación Python. Este framework permite un despliegue de una aplicación de escritorio al entorno web de una forma sencilla, segura y fácilmente escalable.

Su metodología es MVT - Model View Template. A continuación, se explicarán en que consisten cada una de ellas:

Model

Model o Modelo es la parte encargada de la gestión con la base de datos y de abstraer su uso encapsulandola mediante clases que representan a cada una de las instancias de la BD (o tablas).

View

View o Vista es la parte encargada de la gestión entre la base de datos y el template. Puede llevar a confusión esta metodología de desarrollo web, ya que existe una similar: MVC. Pero en MVC (Model-View-Controller) el modelo se encarga de la gestión de la BD, la vista de representar los

datos y el controlador de administrador o motor de contenidos entre la BD y la vista final de la aplicación web.

Así pues la **Vista** en el framework Django se representa al modelo MVC como el controlador.

Template

Template o Plantilla se refiere a la parte encargada de representar la información almacenada en BD, procesada y generada mediante la Vista (View). Al contrario de un modelo MVC, en donde el controlador se encarga de generar las vistas, en este modelo se representan las mismas como una plantilla de muchas otras que se van enrutando a diferentes funciones generadas por la vista.

Dicho esto, siempre habrá una plantilla Master o padre de la que heredarán todas las hijas que vayamos asociando a nuestra web. Estas heredan una estructura o skeleton en formato html enriquecido con lenguaje propio de Django en formato python. Veamos un ejemplo de plantilla Master y una plantilla que se nutre de este skeleton:

```

1  <!DOCTYPE html>
2  {% load staticfiles %}
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width,
7          initial-scale=1, maximum-scale=1"/>
8      <title>Security Sensor | {% block title %}{% endblock %}</title>
9
10     <link rel="stylesheet" href="{% static 'css/main.css' %}"
11         type="text/css"/>
12     <link rel="stylesheet" href="{% static
13         'bower_components/c3/c3.css' %}" type="text/css"/>
14
15 </head>
16 <body>
17
18     {% block content %}
19
20     {% endblock %}
21
22     <script src="{% static 'bower_components/react/react.js',
23         %}"></script>
```

Figura 2.7: Cabecera de la plantilla Master que heredarán el resto de plantillas/vistas de la aplicación

En el fragmento de la plantilla Master incluimos los archivos estáticos del sistema (**load staticfiles**) y además especificamos donde iría el cuerpo de las vistas o plantillas hijas que heredarán este skeleton mediante la especificación de un bloque (**block content** y **endblock**). Y ahora veremos un ejemplo de lo que podría ir dentro de ese bloque.

```

1  {% extends 'MasterPages/MasterPage.html' %} 
2  {% block title %}Home{% endblock %}
3  {% block content %} 
4
5  <div id="content">
6
7  </div>
8  <div id="chart">
9  <svg></svg>
10 </div>
11 <div id="sub-content">
12 </div>
13 <div id="events-info">
14 </div>
15
16 </div>
17 {% if latest_source_list %}
18 <ul>
19     {% for logsource in latest_source_list %}
20         <li><a href="{% url 'secapp:events' logsource.id %}">{{logsource.Type }}</a></li>
21     {% endfor %}
22 </ul>
23 {% else %}
24 <p> No sources are available.</p>
25 {% endif %}
26 {% endblock %}

```

Figura 2.8: Contenido de la plantilla index que hereda funcionalidades de la plantilla Master.

Aquí se observa perfectamente como se heredan las características de la plantilla Master y se define el bloque de contenido que en la plantilla Master ya se especificó (se situa dentro de las etiquetas **block** y **endblock**).

2.4.4. D3js



Web: <https://d3js.org/>

D3.js es una biblioteca javascript para la manipulación de documentos basado en datos. D3 nos ayuda a la hora de dar vida a los datos usando HTML, SVG y CSS de una forma sencilla y visualmente muy espectacular. Tiene su propio lenguaje que hace uso de las funcionalidades de Javascript por debajo y cuya comunidad es muy amplia en dónde podemos obtener multitud de ejemplos visuales que demuestran la capacidad de generación de gráficas y eventos de la biblioteca.

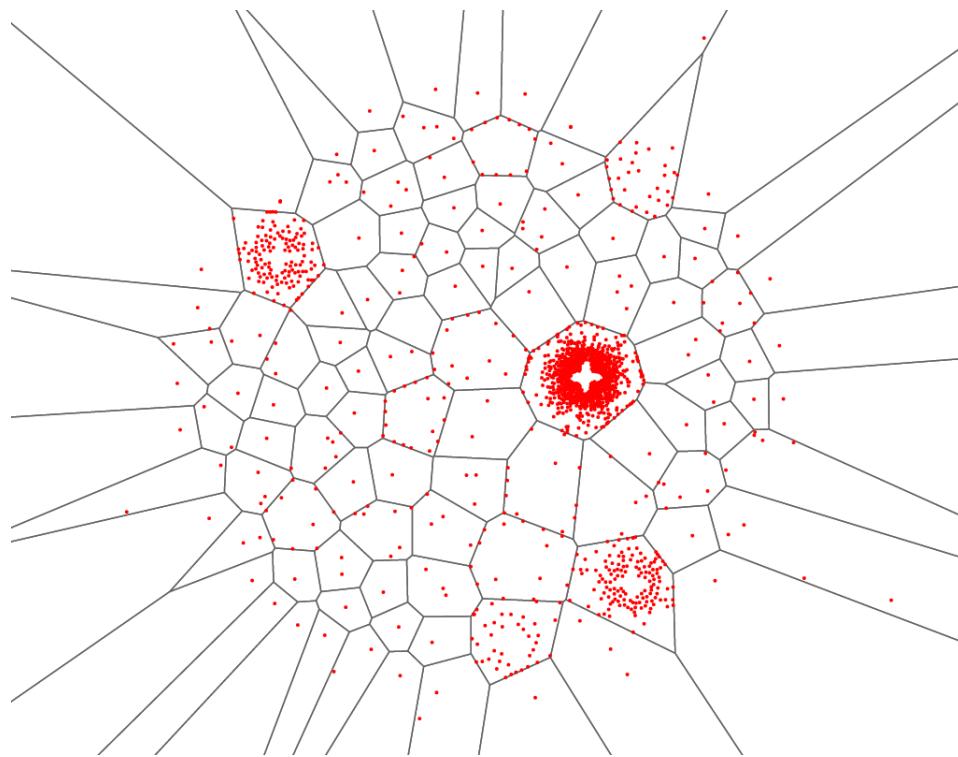
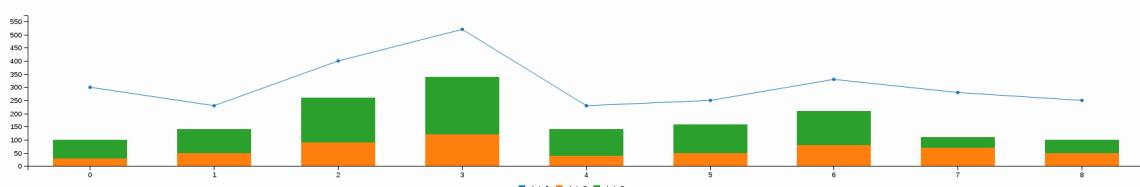


Figura 2.9: Ejemplo de D3js

2.4.5. C3js

Web: <http://c3js.org/>

Es una biblioteca gráfica basada D3js, que es otra biblioteca visual en Javascript. En este caso, la biblioteca hace uso de la funcionalidad para gráficas de D3js adaptando su motor a otras posibles soluciones de implementación.

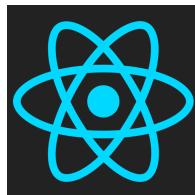


2.4.6. LATEX

Web: <https://www.latex-project.org/>

LATEX es un lenguaje de marcado que sirve para la redacción de documentos científicos o técnicos. Con esta herramienta o lenguaje se ha desarrollado la memoria actual del proyecto de final de carrera.

2.4.7. Reactjs



Web: <https://facebook.github.io/react/>

React es una biblioteca Javascript que permite construir interfaces de usuario en nuestra aplicación web. Existen soluciones similares a React como podría ser JQuery, pero en este caso el proceso de creación de componentes visuales se hace muy intuitivo mediante clases internas que encapsulan el contenido que posteriormente se traducirá a código Javascript nativo.

Para la traducción, React utiliza un componente llamado JSX, pero en nuestra solución se ha usado el paquete Babel (<https://babeljs.io/>) que es similar pero en formato Javascript y no NodeJS.

2.4.8. Highcharts



Web: <http://www.highcharts.com/>

Highcharts es una biblioteca gráfica desarrollada en Javascript que permite construir gráficos de manera muy intuitiva, fácil, elegante y rápida. Dispone de una api documental muy práctica y que ayuda en su aprendizaje.

Con esta biblioteca se ha desarrollado la parte visual de la web en dónde se incluyen gráficas de eventos, eventos en tiempo real y estadísticas generadas por día.

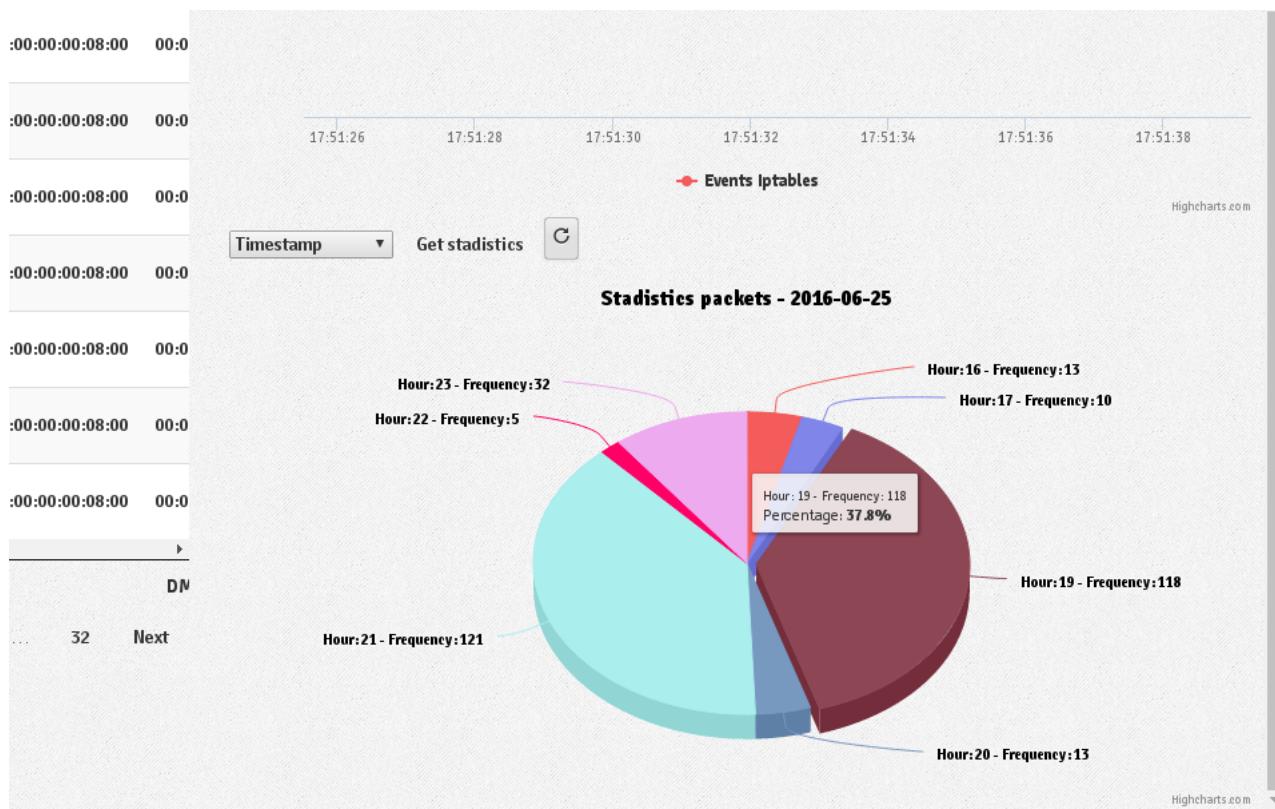


Figura 2.10: Ejemplo de visualización para la biblioteca Highcharts

2.4.9. SQLite



Web: <https://www.sqlite.org/>

SQLite es un biblioteca software que implementa un motor para bases de datos SQL. Sus principales características son las siguientes:

- Las transacciones de datos son atómicas, consistentes, aisladas y duraderas (ACID) incluso después de que el sistema tenga un fallo o se apague inesperadamente.
- No necesita de una administración o configuración previa para su normal funcionamiento.
- Implementación de un sistema SQL íntegro con características más avanzadas como indexación parcial.
- La base de datos, en su totalidad, se almacena como un archivo normal en disco. Esto es útil para ser cargado directamente como un archivo en una aplicación.
- Soporta bases de datos de Terabytes de tamaño y Gigabytes de string o archivos.
- Facilidad de uso mediante su API interna.

- No tiene ninguna dependencia externa.
- Multiplataforma: Android, BSD, iOS, Linux, Mac, Solaris, VxWorks y Windows soportan este tipo de formato de base de datos.
- El código fuente está bajo dominio público para cualquier tipo de uso.

2.4.10. Rsyslog

Web: <http://www.rsyslog.com/>

Rsyslog (Rocket-Fast System for Log Processing), es un sistema de recogida de logs de sistemas UNIX (servicios mediante syslogd) que nos permite manipularlos y exportarlos a un formato más adecuado para su procesado.

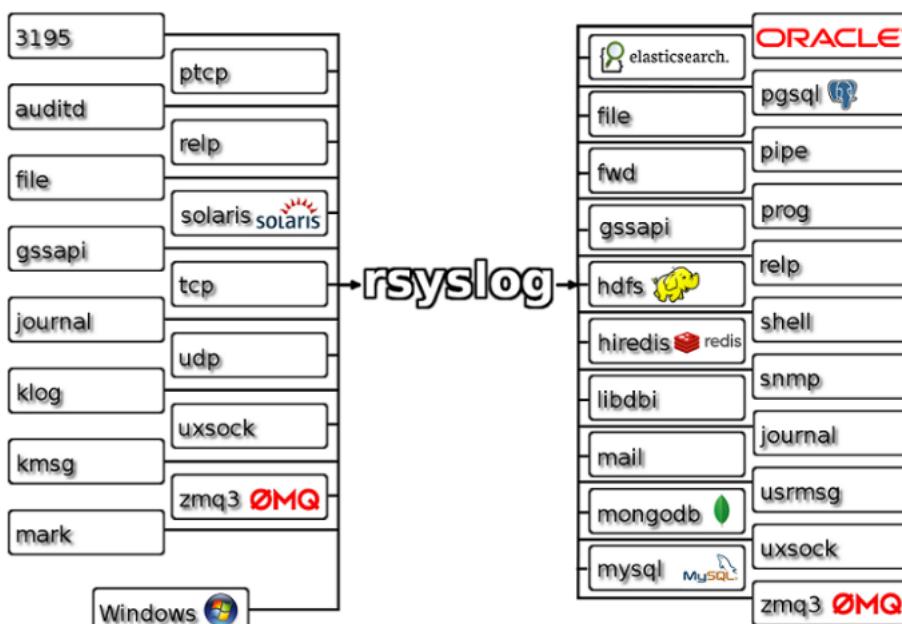


Figura 2.11: Esquema de Rsyslog

- Multihilo
- TCP, SSL, TLS RELP
- Se puede filtrar cualquier parte de los mensajes de syslog
- Se puede configurar el formato de salida de la recolección

2.4.11. Logrotate

Web: <https://github.com/logrotate/logrotate>
man: <http://linux.die.net/man/8/logrotate>

Es una utilidad de los sistemas UNIX que nos permite simplificar la administración de archivos de logs en un sistema en dónde haya logs de muchos tipos de fuentes. Nos permite automatizar la rotación, compresión, eliminación y envío por email de los archivos de logs del sistema. Logrotate se encuentra normalmente corriendo como un proceso cron diario.

```

1 /var/log/iptables.log
2     {
3         rotate 7
4         daily
5         missingok
6         notifempty
7         delaycompress
8         compress
9         postrotate
10            invoke-rc.d rsyslog restart > /dev/null
11        endscript
12    }

```

Figura 2.12: Configuración de iptables para Logrotate

Ahora se realizará una breve explicación de cada configuración que se ha establecido sobre el archivo `iptables.log`:

- **rotate <count>**: Los archivos de log son rotados una cantidad de `<count>` veces antes de ser eliminados o enviados por correo. Si `<count>` es 0, las versiones anteriores son eliminados antes de efectuarse la rotación.
- **daily**: Los archivos de log son rotados diariamente.
- **missingok**: Si el archivo de log no existe, ir al siguiente sin mostrar un mensaje de error.
- **notifempty**: No rotar el log si éste está vacío.
- **delaycompress**: Pospone la compresión de los logs previos para el siguiente ciclo de rotaciones. Esto sólo sucede cuando se combina con la opción **compress**.
- **compress**: Las versiones antiguas de logs son comprimidas con gzip por defecto.
- **postrotate-endscript**: Las líneas que se encuentran entre estas dos palabras en la configuración, son ejecutadas (usando `/bin/sh`) antes de que el archivo de log sea rotado y sólo si el log actual va a ser rotado.

En este caso, el comando que se ejecuta fuerza a `rsyslog` a reabrir el archivo de log para escribir en él. Se suele usar después que logrotate mueva los archivos de logs antiguos, entonces `rsyslog` comienza a escribir en los nuevos.

2.4.12. Syslog

Web (man): <http://linux.die.net/man/3/syslog>

Sirve para el envío de mensajes al sistema de logs interno. `syslog()` genera un mensaje de log, el cuál se distribuye mediante el demonio `syslogd`.

2.4.13. Iptables

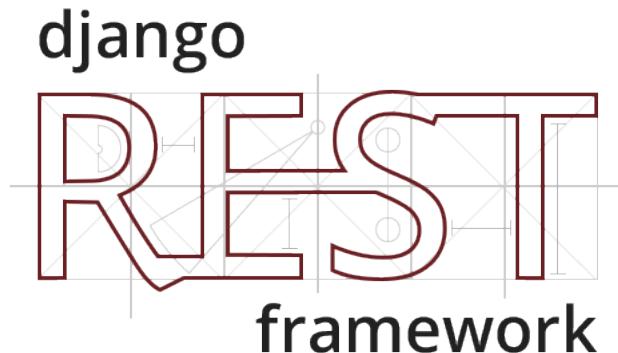
Web: <http://www.netfilter.org/projects/iptables/index.html>

Manual: <http://www.netfilter.org/documentation/HOWTO/es/packet-filtering-HOWTO-7.html>

man: <http://linux.die.net/man/8/iptables>

Herramienta de filtrado de paquetes IPv4/IPv6 y NAT. Iptables es usado para configurar, mantener e inspeccionar tablas de paquetes IP mediante filtros o reglas en el kernel de Linux. Se pueden definir multitud de tablas, en las que cada una pueda contener un número de reglas precargadas o que pueden ser redefinidas por el usuario.

2.4.14. Django-rest



Web: <http://www.django-rest-framework.org/>

Django REST es un framework potente y flexible que permite construir Web APIs. Se usa como un complemento al framework Django para el uso de API Restfull dentro de la propia aplicación.

2.4.15. JSON

Web: <http://www.json.org/>

JSON (Javascript Object Notation) es un formato de intercambio de datos ligero. Se usa para facilitar la legibilidad y escritura para los humanos y además es fácil de interpretar y parsear para una máquina. Está basado como un subconjunto de JavaScript y el Standard ECMA-262 3^a Edición.

JSON está construido sobre dos estructuras:

- Una colección de pares nombre/valores. En varios lenguajes, esto se realiza mediante objetos, registros, estructuras, diccionarios, tablas hash, listas de claves o arrays asociativos.
- Una lista de valores ordenados. En la mayoría de lenguajes, esto se realiza mediante un array, un vector, lista o secuencia.

2.4.16. PyCharm



Web: <https://www.jetbrains.com/pycharm/>

PyCharm es un IDE que permite el trabajo con aplicaciones Python en sus respectivos entornos virtuales (virtualenv) así como la integración con frameworks de desarrollo como es el caso de Django. La aplicación en sí fue desarrollada nativamente mediante un editor de texto (Emacs) pero a la hora de realizar un empaquetado e integración con otra herramientas se optó por este IDE.

2.4.17. Bitbucket

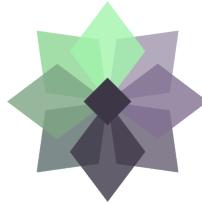


Web: <https://bitbucket.org/>

Repositorio: <https://bitbucket.org/MGautierGomez/securityproject>

Gestor de repositorios Git y Mercurial. Se optó por este gestor para probar su funcionamiento, así como la posibilidad de tener repositorios privados para desarrollar partes del proyecto que necesitasen ser ocultadas. También se encuentra alojado en el repositorio de GitHub: <https://github.com/MGautier/security-sensor>

2.4.18. Taiga



Web: <https://taiga.io/>

Taiga es un gestor de proyectos que nos permite implementar una metodología SCRUM o Kanban sobre nuestro proyecto. Es una herramienta muy intuitiva y colaborativa, que permite definir hitos/tareas/wikis para solucionar cada punto de la fase de desarrollo de un proyecto. Además, permite la integración (WebHooks) con otras plataformas de repositorios de proyectos como GitHub, GitLab o BitBucket.

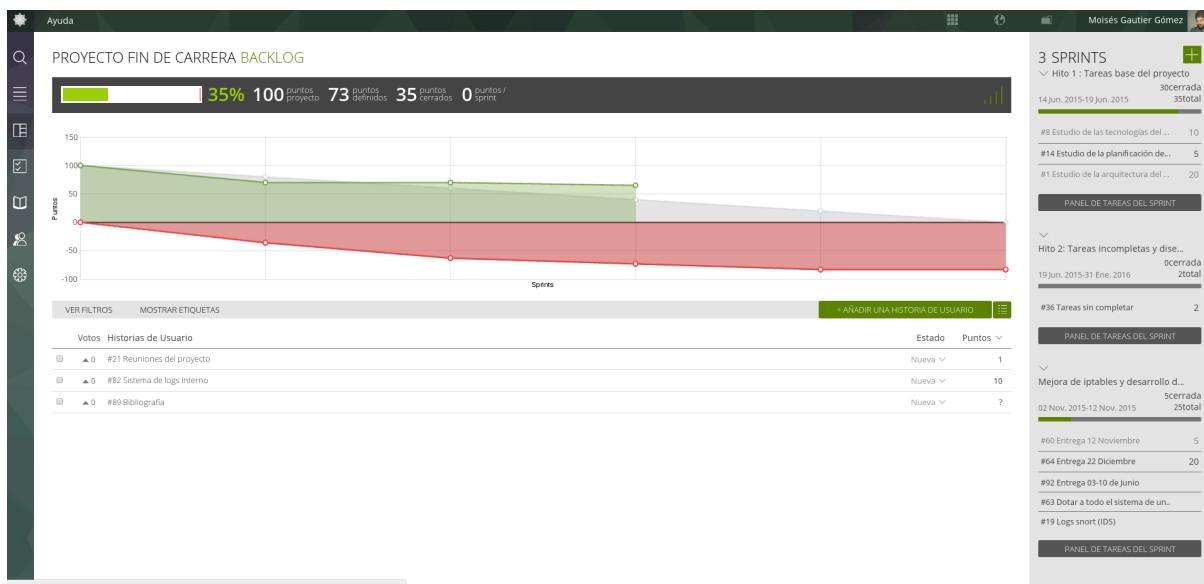


Figura 2.13: Backlog del proyecto siguiendo un SCRUM

2.4.19. Git



Web: <https://git-scm.com/>

Git es un sistema open-source de control de versiones diseñado para manejar integralmente las fases de desarrollo de proyectos, simples y complejos, con velocidad y eficiencia.

2.4.20. Digital Ocean



Web: <https://www.digitalocean.com/>

Servidor web para alojar proyectos en cloud. La ventaja de este servicio de VPS es que te permite desplegar máquinas de cualquier tipo (siempre que sean software libre) de una manera muy fácil y rápida. Además tiene un punto fuerte y es que la información se almacena en discos SSD, con lo que el procesamiento se ve muy mejorado a la hora de computar (en este caso eventos de Iptables).

The screenshot shows the DigitalOcean control panel. At the top, there's a navigation bar with links for 'Droplets', 'Images', 'Networking', 'API', and 'Support'. To the right of the navigation is a green 'Create Droplet' button. Below the navigation, the word 'Droplets' is prominently displayed in large letters. To the right of 'Droplets' is a search bar labeled 'Search By Droplet Name'. Underneath, there are two tabs: 'Droplets' (which is selected and underlined) and 'Volumes'. The main area displays a table with one row of data. The columns are 'Name', 'IP Address', and 'Created'. The first row shows 'debian-512mb-lon1-01' as the name, '139.59.165.31' as the IP address, and '3 months ago' as the creation date. There's also a 'More' link next to the creation date.

Name	IP Address	Created
debian-512mb-lon1-01 512 MB / 20 GB Disk / LON1	139.59.165.31	3 months ago

Figura 2.14: Droplet desplegado en digital ocean

2.4.21. Nginx



Web: <http://nginx.org/>

Características: <http://nginx.org/en/>

Nginx (engine x) es un servidor HTTP y proxy inverso, un servidor proxy de email y un servidor proxy genérico de TCP/UDP. Algunas de sus principales características son las siguientes:

- Sirve archivos estáticos, index y autoindexados.
- Acelera el proceso de proxy inverso con caché: Tolerancia a fallos y carga balanceada de datos.
- Soporta aceleración con cacheo de FastCGI, uwsgi, SCGI y servidores memcached: Tolerancia a fallos y carga balanceada de datos.
- Soporta SSL y TLS SNI.
- Soporta HTTP/2 con dependencia basada en prioridad y balanceo.

Configuración de nginx para el servidor en digital ocean:

```

1 server {
2
3     root /var/www/html;
4
5     # Tipos de archivos index de nuestro sistema
6
7     index index.html index.htm index.nginx-debian.html;
8
9     # Nombre del servidor en local
10
11    server_name 139.59.165.31;
12
13    location /static/ {
14        alias ~/trunk/version-1-0/webapp/secproject/static/;
15        expires 30d;
16    }
17
18    location / {
19        proxy_set_header X-Forwarded-For
20            $proxy_add_x_forwarded_for;
21        proxy_set_header Host $http_host;
22        proxy_redirect off;
23        proxy_pass http://127.0.0.1:8000;
24        proxy_pass_header Server;
25        proxy_set_header X-Real-IP $remote_addr;
26        proxy_connect_timeout 10;
27        proxy_read_timeout 10;
28    }
29}
30

```

Figura 2.15: Configuración de nginx en digital ocean

2.4.22. Dependencias Python

Estas son las dependencias que se necesitan para la fase de desarrollo del proyecto. Para poder hacer uso de ellas, previamente tendremos que tener instalado un entorno virtual o virtualenv desde el cual hacer la instalación de estos paquetes.

Virtualenv

Es una herramienta que nos permite crear entornos aislados de Python. De esta forma podemos realizar pruebas de dependencias o paquetes para un entorno de desarrollo, en dónde las instalaciones o pruebas no afecten a las dependencias internas del sistema.

- Django 1.9.2: <https://pypi.python.org/pypi/Django/1.9.2>
- argparse 1.4.0: <https://docs.python.org/2.7/library/argparse.html>
- dnspython 1.12.0: <https://pypi.python.org/pypi/dnspython/1.12.0> - Se usa para hacer resolución directa e inversa de DNS.

- gunicorn 19.4.5: <https://pypi.python.org/pypi/gunicorn> - Sirve para desplegar un servidor WSGI (Web Server Gateway Interface) HTTP en entornos Unix.
- optional-django 0.3.0: <https://pypi.python.org/pypi/optional-django/> - Otras utilidades del framework Django
- pygtail 0.6.1: <https://pypi.python.org/pypi/pygtail> - Sirve para leer archivos de log en los cuales haya registros del mismo sin leer. Similar al comando tail -f en sistemas Unix.
- python-dateutil 2.4.2: <https://pypi.python.org/pypi/python-dateutil/2.4.2> - Extensión al módulo principal de Python datetime.
- react 2.0.2: <https://pypi.python.org/pypi/react/2.0.2> - Módulo que ejecuta un servidor de datos para los componentes react de la aplicación en Python.
- requests 2.9.1: <https://pypi.python.org/pypi/requests/> - Es una biblioteca Python para poder consumir recursos HTTP de forma segura y controlada.
- setuptools 20.1.1: <https://pypi.python.org/pypi/setuptools> - Herramienta para la descarga, compilación, instalación, desinstalación y actualización de todos los paquetes Python. El resultado de la misma se puede usar mediante pip que se nutre el repositorio de paquetes PyPi.
- six 1.10.0: <https://pypi.python.org/pypi/six> - Es una biblioteca de compatibilidad entre Python 2 y 3.
- wheel 0.29.0: <https://pypi.python.org/pypi/wheel> - Es un gestor o generador de paquetes en Python.
- wsgiref 0.1.2: <https://pypi.python.org/pypi/wsgiref> - Es una biblioteca que sirve como soporte de validación para WSGI 1.0.1 para versiones de Python inferiores a la 3.2.
- configparser 3.5.0: <https://docs.python.org/2/library/configparser.html> - Es un xsmódulo o clase que implementa un lenguaje muy sencillo para el parseo de archivos de configuración, siguiendo una estructura similar a como se describen y procesan los archivos INI en sistemas Microsoft Windows.

2.4.23. Versiones de las herramientas empleadas

Las versiones de las herramientas anteriormente mencionadas no difieren mucho el comportamiento unas de otras, para el caso de herramientas softwares que no sean dependencias directas de Python.

Para el caso de las dependencias de Python, las especificadas en los requerimientos de instalación dentro del entorno virtual, son las que se han definido en la sección anterior. De igual manera, no afecta que sea de una versión superior o no, simplemente se empezó a desarrollar el proyecto en esas versiones.

Capítulo 3

Especificación y análisis de requisitos

Para el caso que nos concierte en el proyecto, dentro del marco de investigación que define la totalidad de la infraestructura, la funcionalidad principal del mismo será:

- Definir los pasos para obtener recolección de logs de una fuente de seguridad. Configuración de rsyslog, logrotate y supervisord (este último en el caso de que sea necesario).
- Una vez configurada la máquina, configurar la instalación para cada fuente en particular. El ámbito del proyecto se enfoca sobre iptables.
- Realizar un sistema de parseo de logs para extraer la información necesaria para cada evento que se registre en el sistema.
- Sistema de gestión de base de datos en donde se encuentren los datos en crudo recogidos, los procesados y los dispuestos para su visualización.
- Panel de control donde visualizar la información de ese nodo con total detalle de la información.

3.1. Especificación de los requisitos

En esta etapa del modelado de requisitos se captura el propósito general del sistema:

- Se analiza qué debe hacer el sistema.
- Se obtiene una versión contextualizada del sistema.
- Identifica y delimita el sistema.
- Se determinan las características, cualidades y restricciones que debe satisfacer el sistema.

3.1.1. Requisitos funcionales

Los requisitos funcionales que se han obtenido en el sistema son los siguientes:

- Ser una herramienta multiplataforma y que permita a cualquier usuario definir sus propias interfaces de gestión de eventos.

- Dotar de funcionalidad gráfica que permita extraer información en tiempo real con gráficas o mecanismos visuales (en web) del sistema de base de datos que ha procesado los inputs de las fuentes para las que ha sido configurada.
- Dotar de una api interna que nos permita extraer información en tiempo real en un formato uniforme para la web o para que cualquier usuario pueda usar la funcionalidad del proyecto para su propio beneficio usando herramientas generadas en el back-end para otro tipo de aplicaciones.
- Ser parte de un todo, en el que el todo sea un SIEM capaz de obtener información de las diferentes sondas o módulos, que en este caso, sería la solución desarrollada.
- Desarrollar una sonda para procesar eventos logs de firewall.
- Extracción y parseo de eventos de firewall Iptables.

3.1.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que describen cualidades o restricciones del sistema que no se relacionan de forma directa con el comportamiento funcional del mismo. A continuación se especifican los más importantes del sistema:

- No requiere un conocimiento específico del sistema una vez puesto en funcionamiento.
- La aplicación tendrá manual de uso.
- La base de datos estará implementada en un lenguaje objeto relacional como SQLite.
- La aplicación estará realizada en el lenguaje de programación Python.
- La interfaz debe reflejar claramente la distinción entre las distintas partes del sistema.
- La documentación del código fuente será llevada a cabo mediante la aplicación Sphinx.
- El sistema se desplegará sobre una versión GNU Linux Debian 8 Jessie.
- El código fuente de la aplicación seguirá un estilo uniforme y normalizado para todos los módulos del mismo.
- El formato de las fechas será dd/mm/yyyy a excepción de la utilizada para la funcionalidad api/events/day/<source>/yyyy-mm-dd.

Capítulo 4

Diseño

A continuación pasaremos a detallar cada una de las partes del diseño del sistema.

4.1. Diseño de la arquitectura

La arquitectura sobre la cual se ha basado el proyecto se representa en el siguiente diagrama correspondiente con una sonda o nodo de recolección de información para el proyecto VERITAS:

Arquitectura del sensor PCA

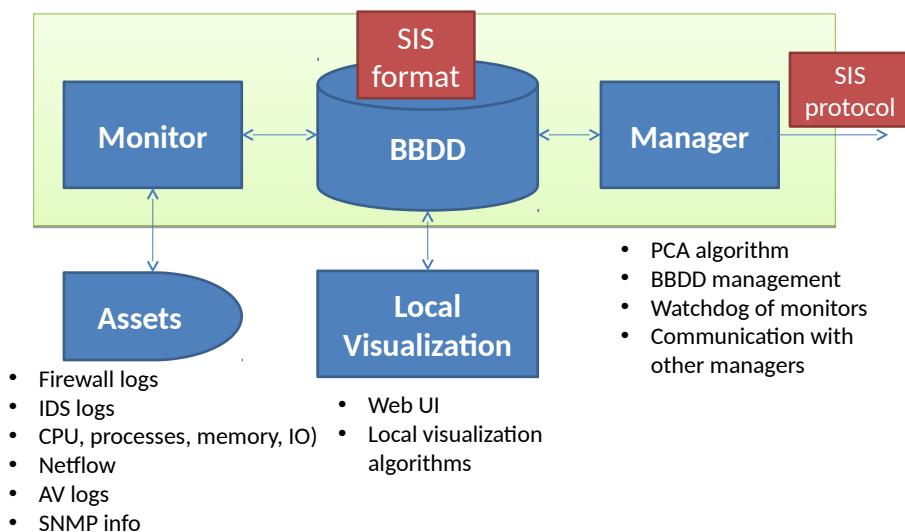


Figura 4.1: Arquitectura interna del software

En el esquema podemos ver toda la arquitectura que tendría una sonda para el proyecto VERITAS, aunque el ámbito de éste proyecto no se engloba en su totalidad, sino en unas partes en concreto del esquema que en el siguiente punto se explicarán en detalle. Puntualizar que la interacción de la sonda con la fuente de seguridad es independiente de la interacción base de

datos con la visualización de los datos. Si bien se hace uso del mismo ORM, proporcionado por el framework Django, estos se ejecutan por separado a la hora de procesar las fuentes.

4.2. Diseño del software de cada módulo

Assets

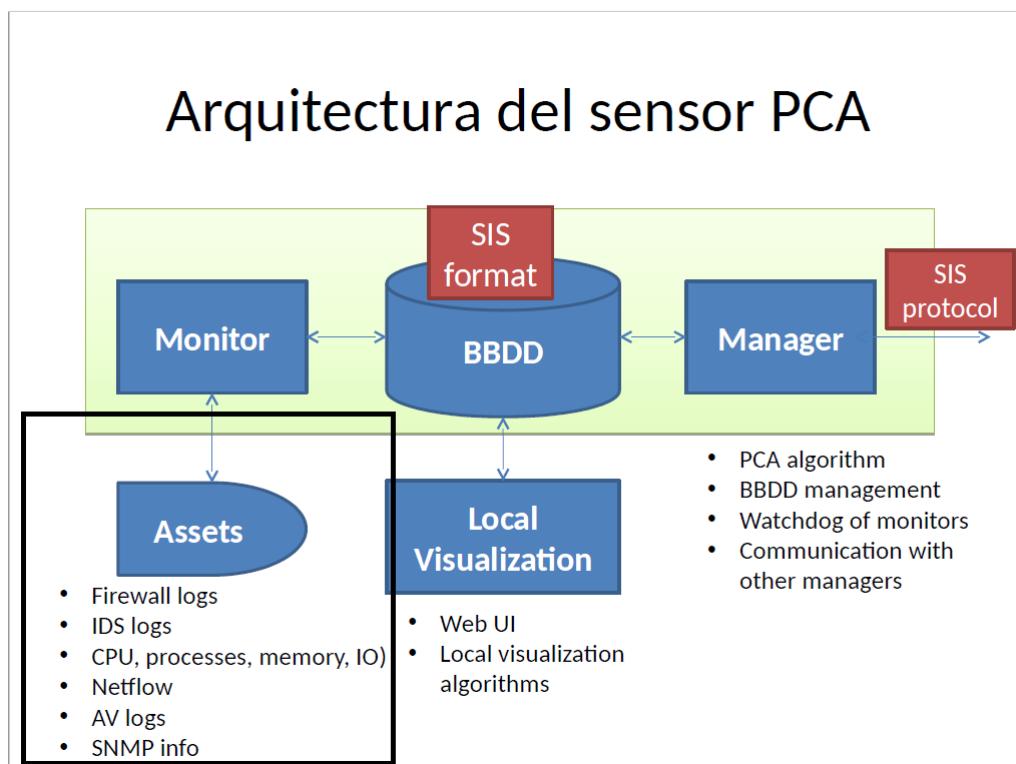


Figura 4.2: Arquitectura: Assets

En esta sección de la arquitectura es donde se define la parte de las fuentes de seguridad que se van a gestionar desde la sonda, es decir, implementar, recolección y pasar el control final a su clase superior: Monitor. La jerarquía de clases sería de la siguiente manera:

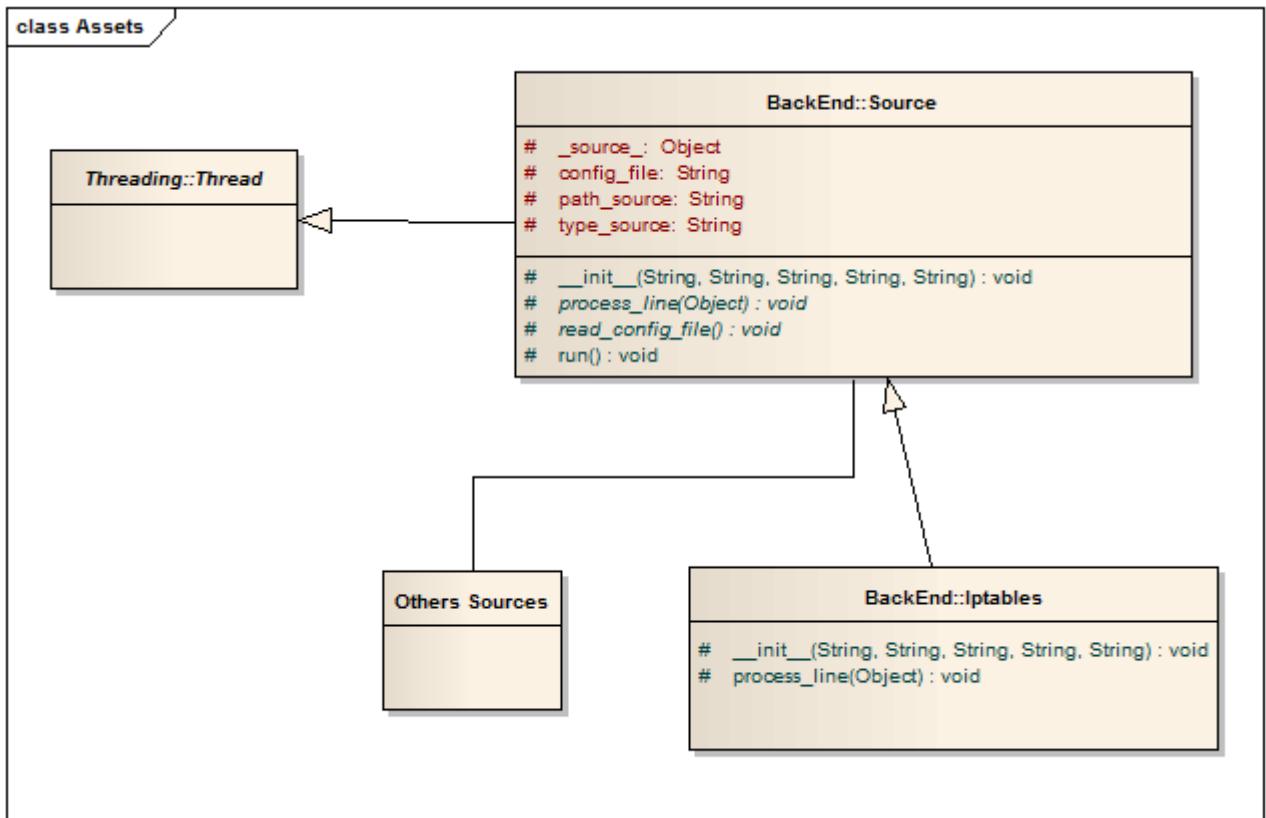


Figura 4.3: Diagrama de clases: Assets

La clase base que sería Source, hereda del comportamiento de la clase Thread (de la biblioteca threading de Python) y a su vez toda fuente de seguridad que se decida implementar heredará de ella los métodos previstos.

Estas fuentes de seguridad podrían ser varias (Iptables, Snort, Syslog, etc), pero cada una de ellas irá asociada a una instancia de Source, es decir, a una hebra que se encargará de monitorear su log. Además, el uso de la clase Source, permite instanciar al resto de clases heredadas de manera interna abstrayendo su uso para el usuario final.

Monitor

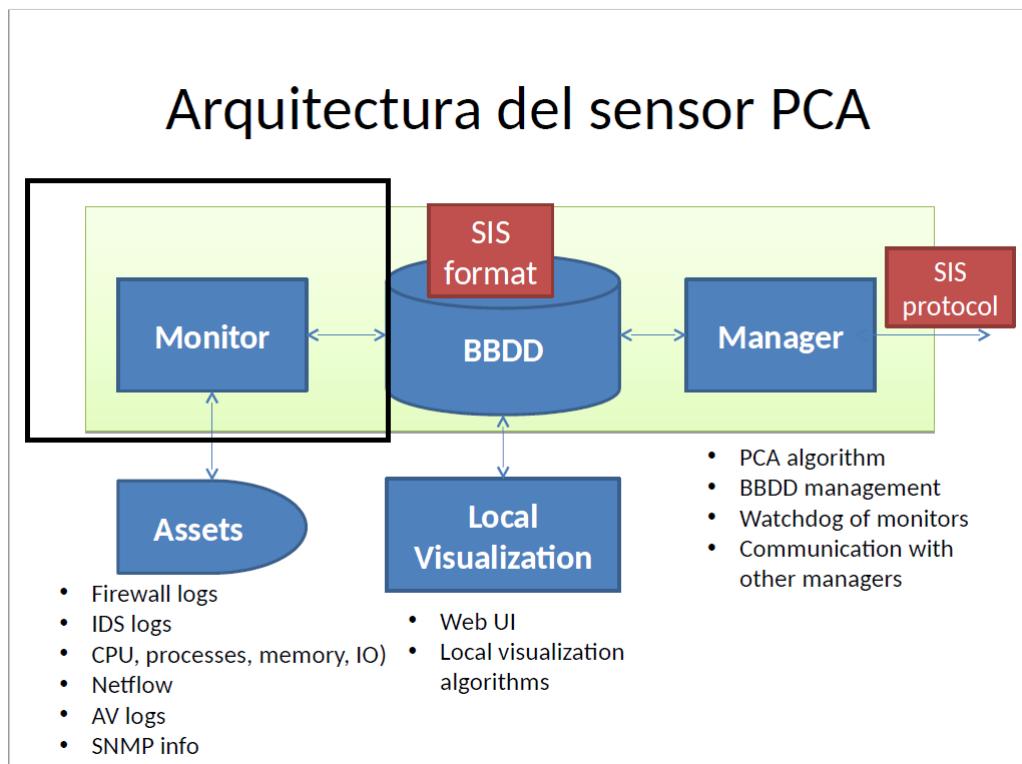


Figura 4.4: Arquitectura: Monitor

En esta sección de la arquitectura es donde se define la parte del control y ejecución (hilos) de las fuentes de seguridad implementadas. A su vez, estas pasan dicha información a la base de datos usando el modelo ORM del framework Django. La jerarquía de clases sería de la siguiente manera:

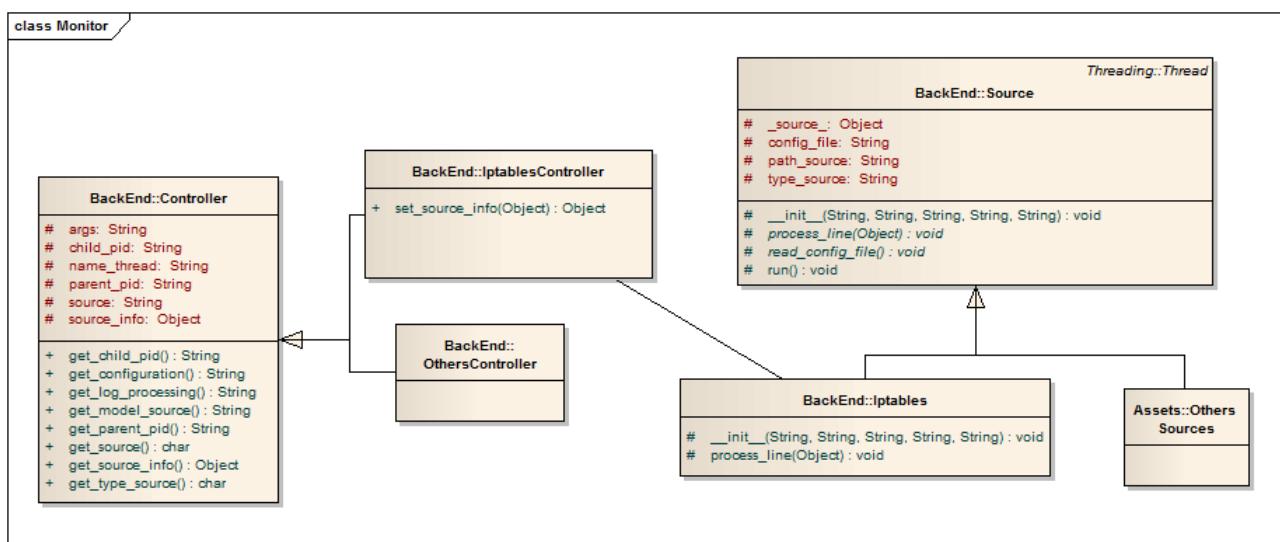


Figura 4.5: Diagrama de clases: Monitor

Nuestra arquitectura monitor se representa con la clase Controller que se encarga de dotar de funcionalidad de ejecución de la fuente (hilo) para unos determinados parámetros de configuración.

BBDD

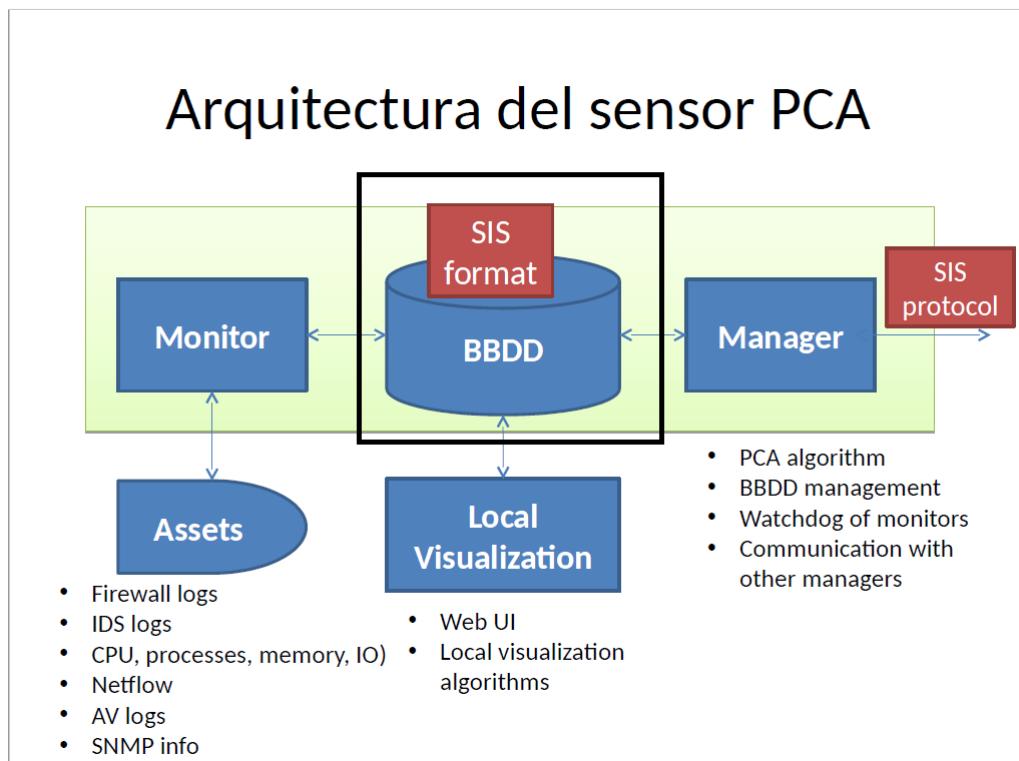


Figura 4.6: Arquitectura: BD

En esta sección de la arquitectura es dónde se define la parte de la interacción con la base de datos y que tablas (clases del modelo ORM) se han definido y con que relaciones. La jerarquía de clases sería de la siguiente manera:

4.2. DISEÑO DEL SOFTWARE DE CADA MÓDULO

CAPÍTULO 4. DISEÑO

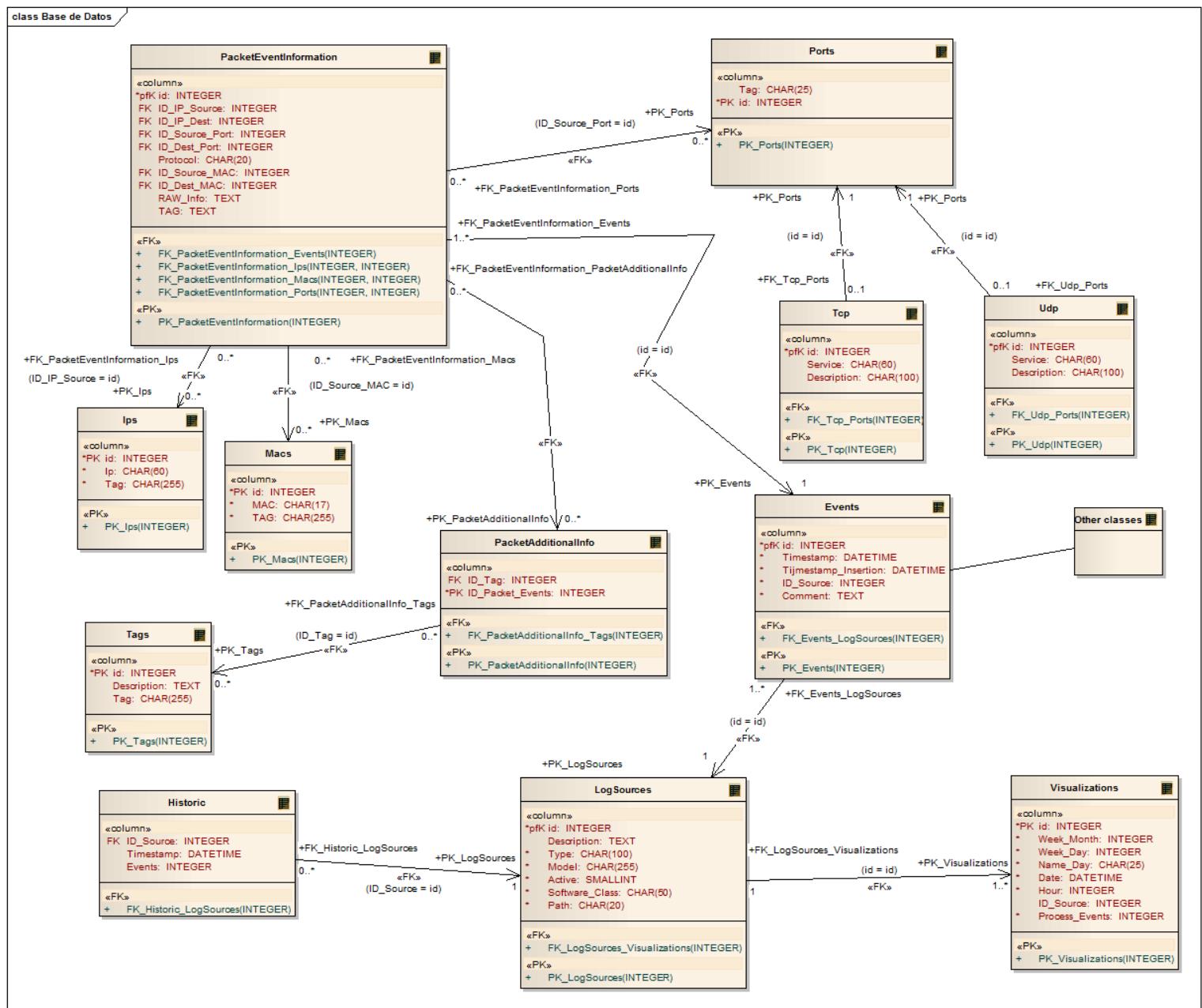


Figura 4.7: Diagrama de clases para la BD (usando ORM)

La clase o tabla que contendrá el peso de toda la jerarquía de base de datos, será **PacketEventsInformation**. Esta tabla sólo contendrá referencias externas o “foreign keys” a cada tabla que haga participe en su definición, es decir:

- Ips
 - Macs
 - Ports
 - Events
 - PacketAdditionalInfo

Como podemos observar en el diagrama anterior, por ejemplo, para la clase PacketEventInformation, tenemos su traducción a formato ORM del motor proporcionado por Django:

```

1 # Clase que alberga la informacion relacionada con el paquete extraido
2 # mediante el log. La gran mayoria de los campos
3 # son identificadores o claves foraneas a otros
4 # objetos/instancias de la base de datos.
5
6 class PacketEventsInformation(models.Model):
7     ID_IP_Source = models.ForeignKey(Ips, models.SET_NULL, blank=True,
8                                         null=True,
9                                         related_name="ip_source")
10    ID_IP_Dest = models.ForeignKey(Ips, models.SET_NULL, blank=True,
11                                    null=True,
12                                    related_name="ip_dest")
13    ID_Source_Port = models.ForeignKey(Ports, models.SET_NULL,
14                                         blank=True,
15                                         null=True,
16                                         related_name="port_source")
17    ID_Dest_Port = models.ForeignKey(Ports, models.SET_NULL,
18                                     blank=True,
19                                     null=True,
20                                     related_name="port_dest")
21    Protocol = models.CharField(max_length=20, default='-')
22    ID_Source_MAC = models.ForeignKey(Macs, models.SET_NULL,
23                                       blank=True,
24                                       null=True,
25                                       related_name="mac_source")
26    ID_Dest_MAC = models.ForeignKey(Macs, models.SET_NULL, blank=True,
27                                    null=True,
28                                    related_name="mac_dest")
29    RAW_Info = models.TextField(default='-')
30    TAG = models.CharField(max_length=255, default='-')
31    id = models.OneToOneField(Events, on_delete=models.CASCADE,
32                             primary_key=True)
33
34    def __str__(self):
35        return '%s' % self.id

```

Figura 4.8: Ejemplo de clase ORM, en concreto PacketEventsInformation

El formato normalizado de la base de datos para visualizar dicha información será mediante JSON, ya que para hacer uso de la información la implementación consumirá dichos datos de la api que se proporciona con la aplicación.

Visualizations

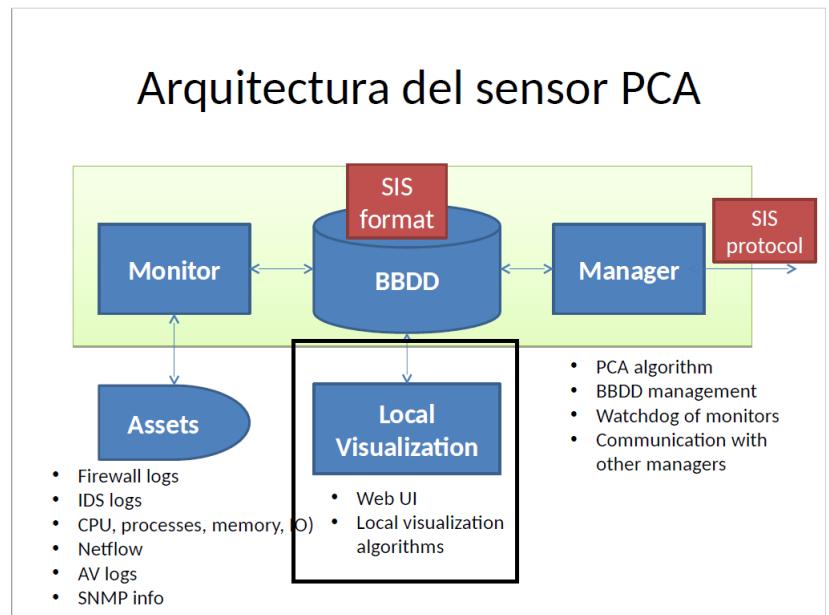


Figura 4.9: Arquitectura: Visualizaciones

En esta sección de la arquitectura es donde se define la parte de la interacción con la base de datos y la visualización de los datos en la interfaz web. La jerarquía de clases sería de la siguiente manera:

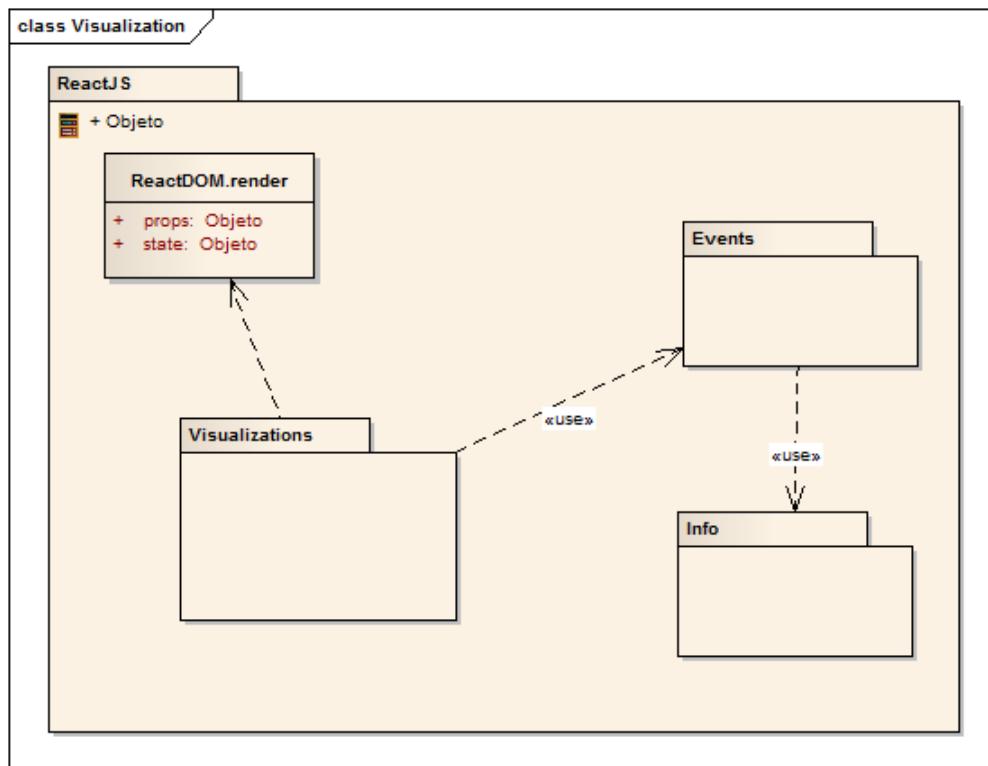


Figura 4.10: Clase Visualization para el paquete ReactJS

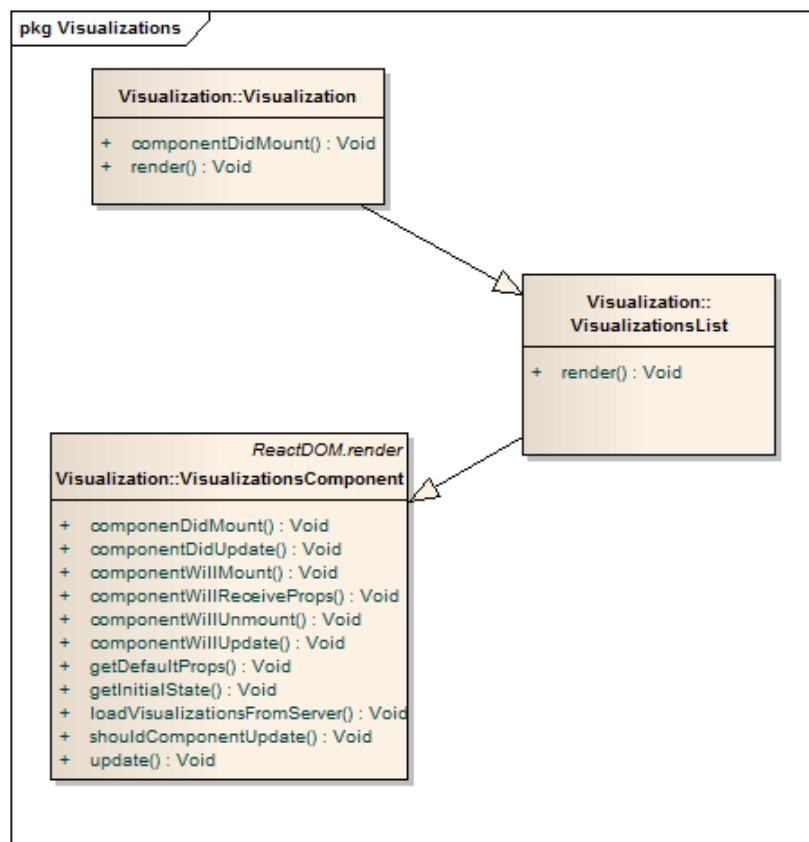


Figura 4.11: Paquete Visualizations

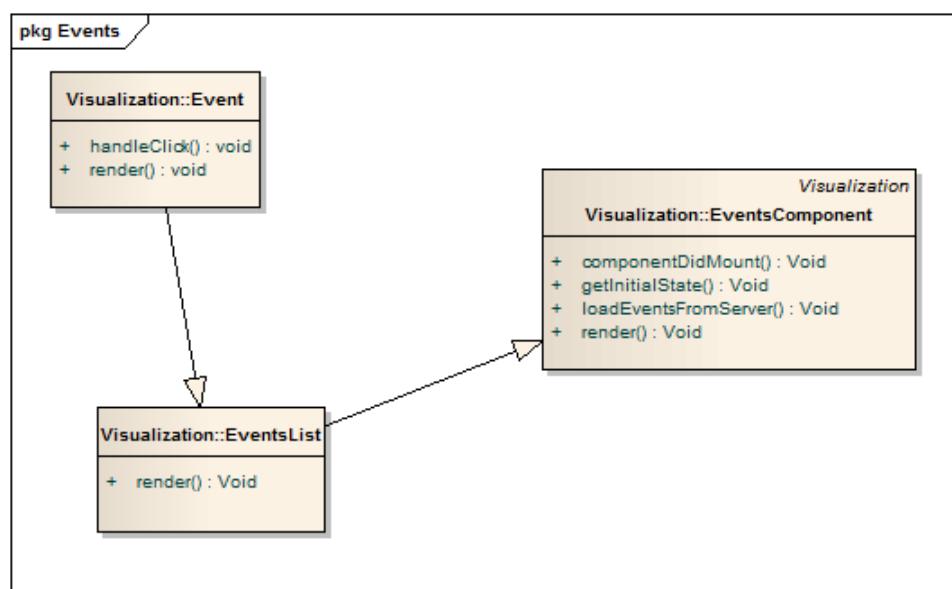


Figura 4.12: Paquete Events

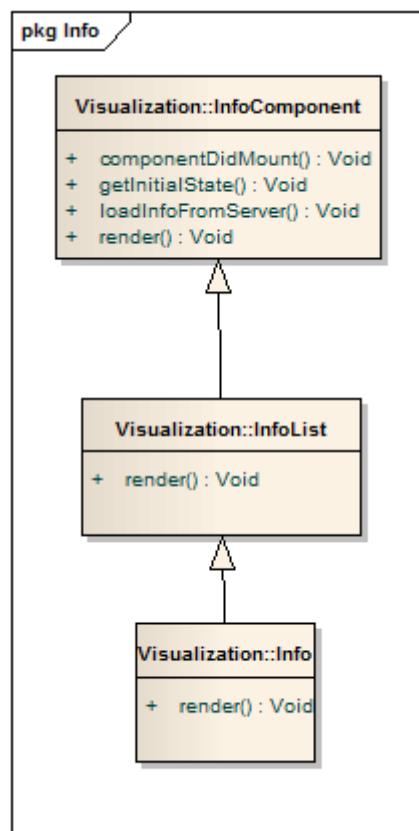


Figura 4.13: Paquete Info

4.3. Diagramas de Secuencia - Operaciones

En esta sección vamos a describir los diagramas de secuencia de operaciones tales como:

- Ejecución principal de la sonda.
- Ejecución principal del servidor web que sirve los datos a la interfaz web.

Ahora definiremos la parte de diagramas de secuencias para la interacción entre el usuario y el procesamiento de fuentes (Back); y el usuario y la visualización de los datos en la web de la aplicación (Front).

4.3.1. Flujo de ejecución de la aplicación: BackEnd

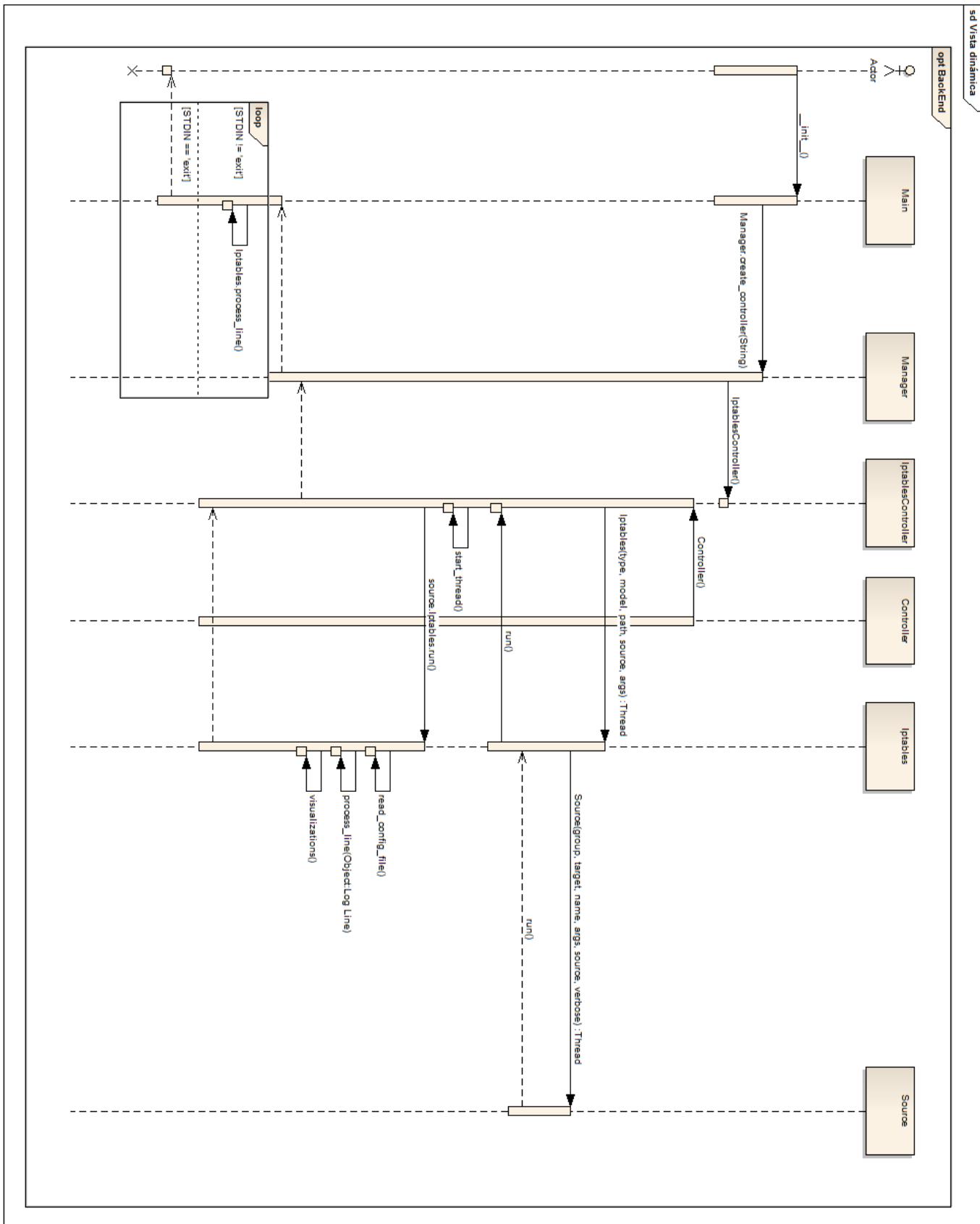


Figura 4.14: Diagrama de Secuencia para la parte BackEnd

4.3.2. Flujo de ejecución de la aplicación: FrontEnd

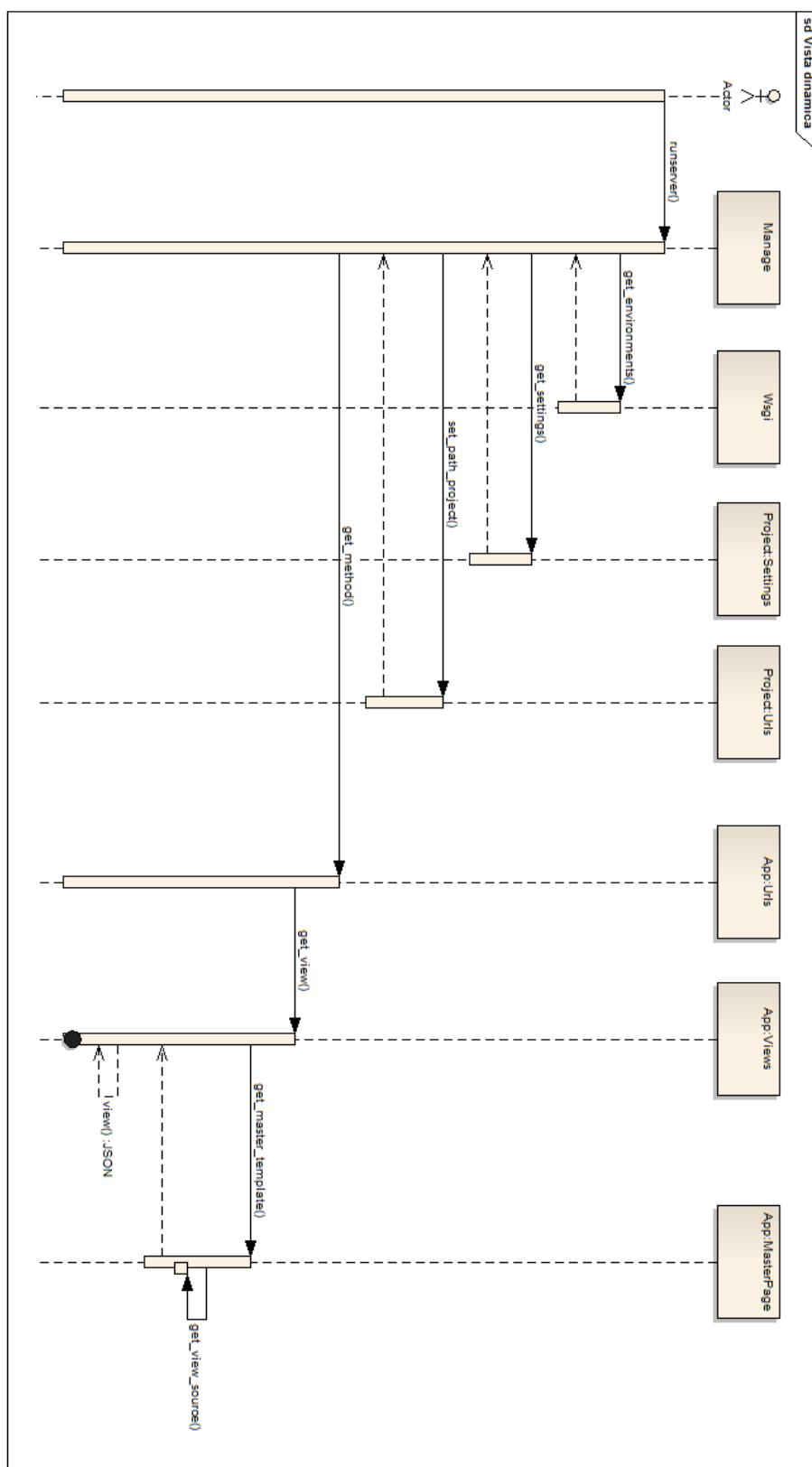


Figura 4.15: Diagrama de Secuencia para la parte FrontEnd

4.4. Diseño de la vista

En esta sección se definen los flujos de interacción para un usuario y la vista final de aplicación en el navegador web.

4.4.1. Flujo de interacción de la vista principal

Interacción del usuario con los items de la vista principal.

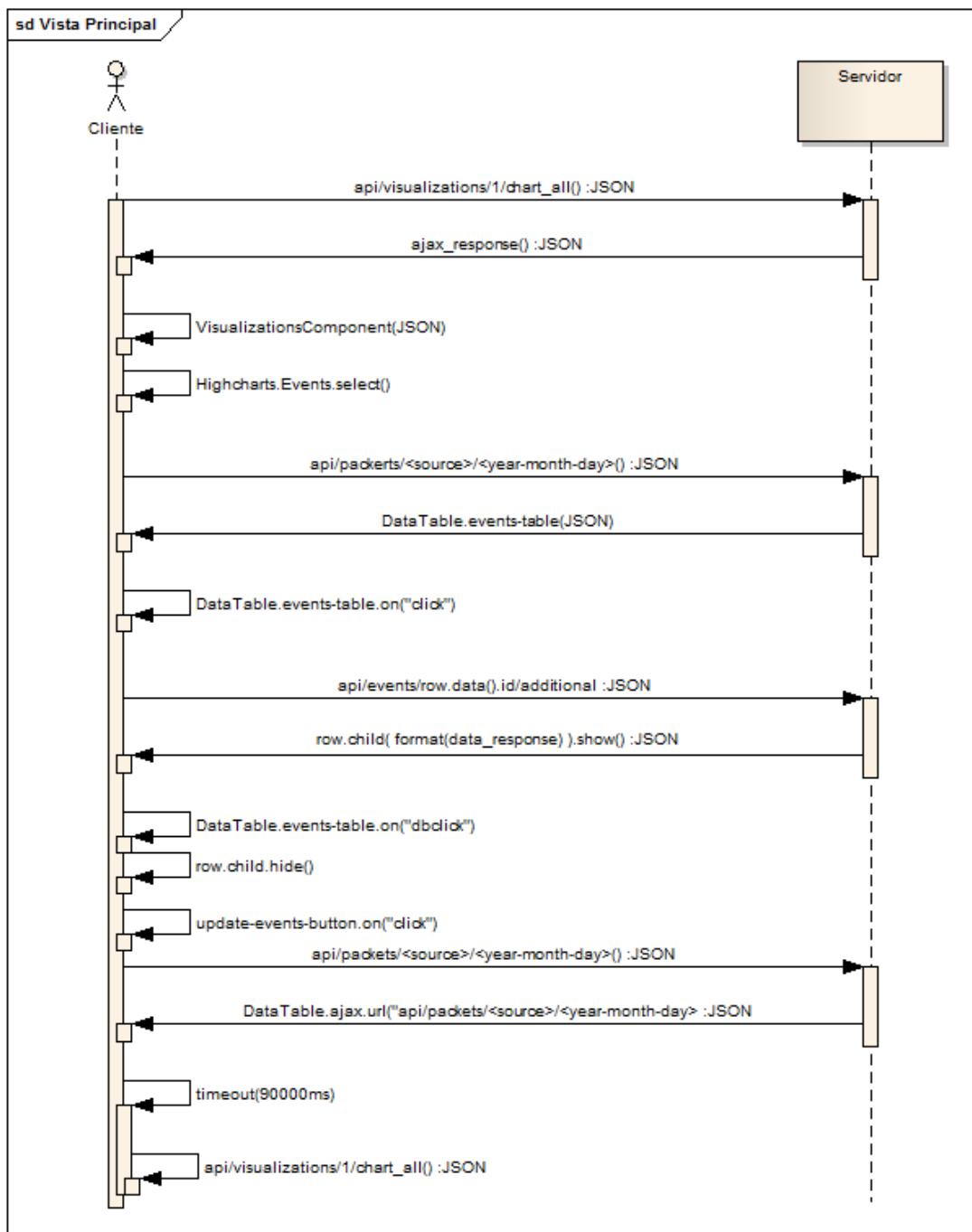


Figura 4.16: Diagrama de Secuencia para la vista principal

4.4.2. Flujo de interacción de la vista de eventos en tiempo real

Interacción del usuario con el ítem de eventos en tiempo real de la vista principal.

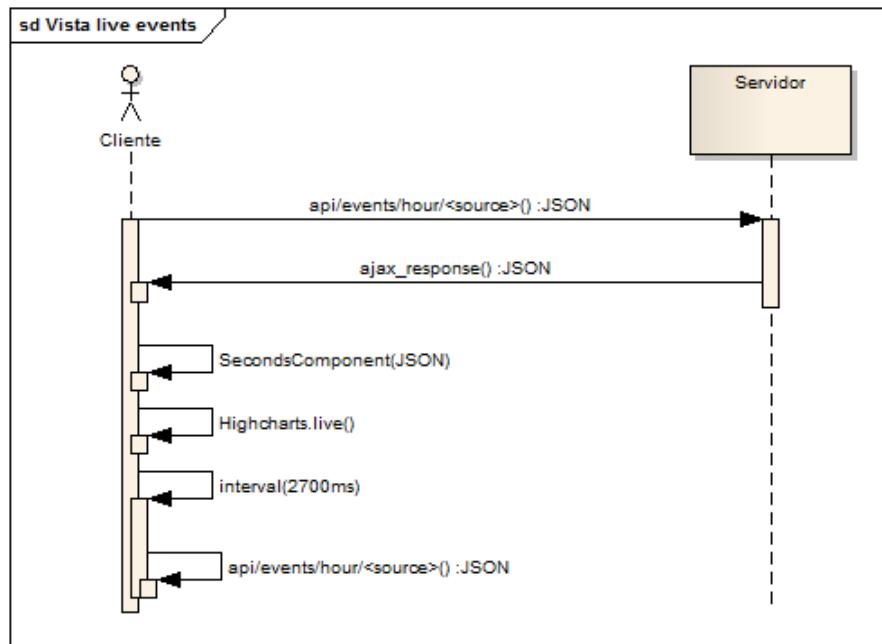


Figura 4.17: Diagrama de Secuencia para la vista de eventos en tiempo real

4.4.3. Flujo de interacción de la vista de estadísticas de los paquetes

Interacción del usuario con el ítem de generación de estadísticas asociadas a eventos de la vista principal.

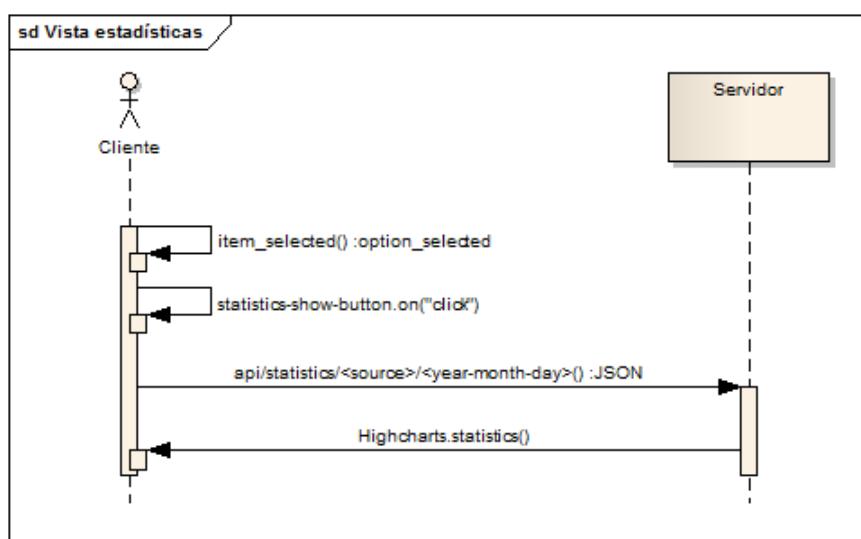


Figura 4.18: Diagrama de Secuencia para la vista de estadísticas de los paquetes

Estos han sido los diagramas de secuencia o interacción que podrá tener la parte de la web con el usuario. Ahora se mostrarán algunas capturas de la parte visual (web) que finalmente se ha obtenido:

Vista Principal

Resultado de la vista final que el usuario podrá acceder y manejar.

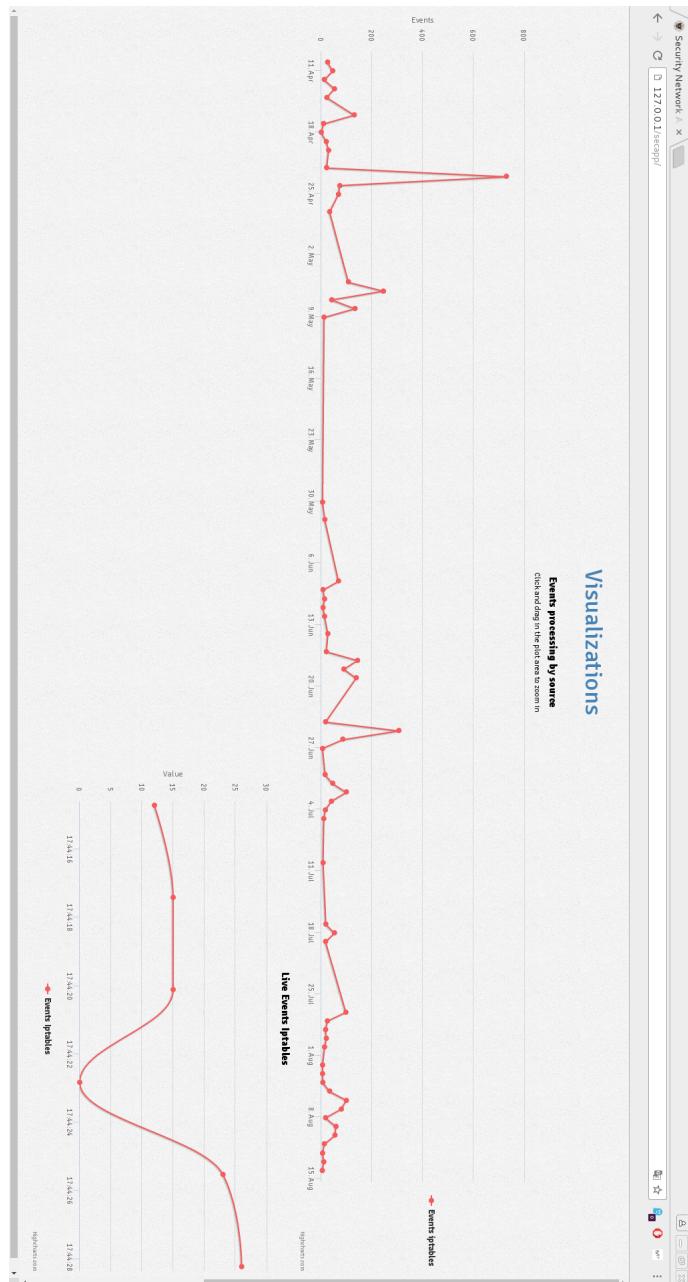


Figura 4.19: Vista principal de la Web

Vista con información de los eventos

Vista en donde se podrá obtener información asociada al evento pulsado en la gráfica principal.

4.4. DISEÑO DE LA VISTA

CAPÍTULO 4. DISEÑO

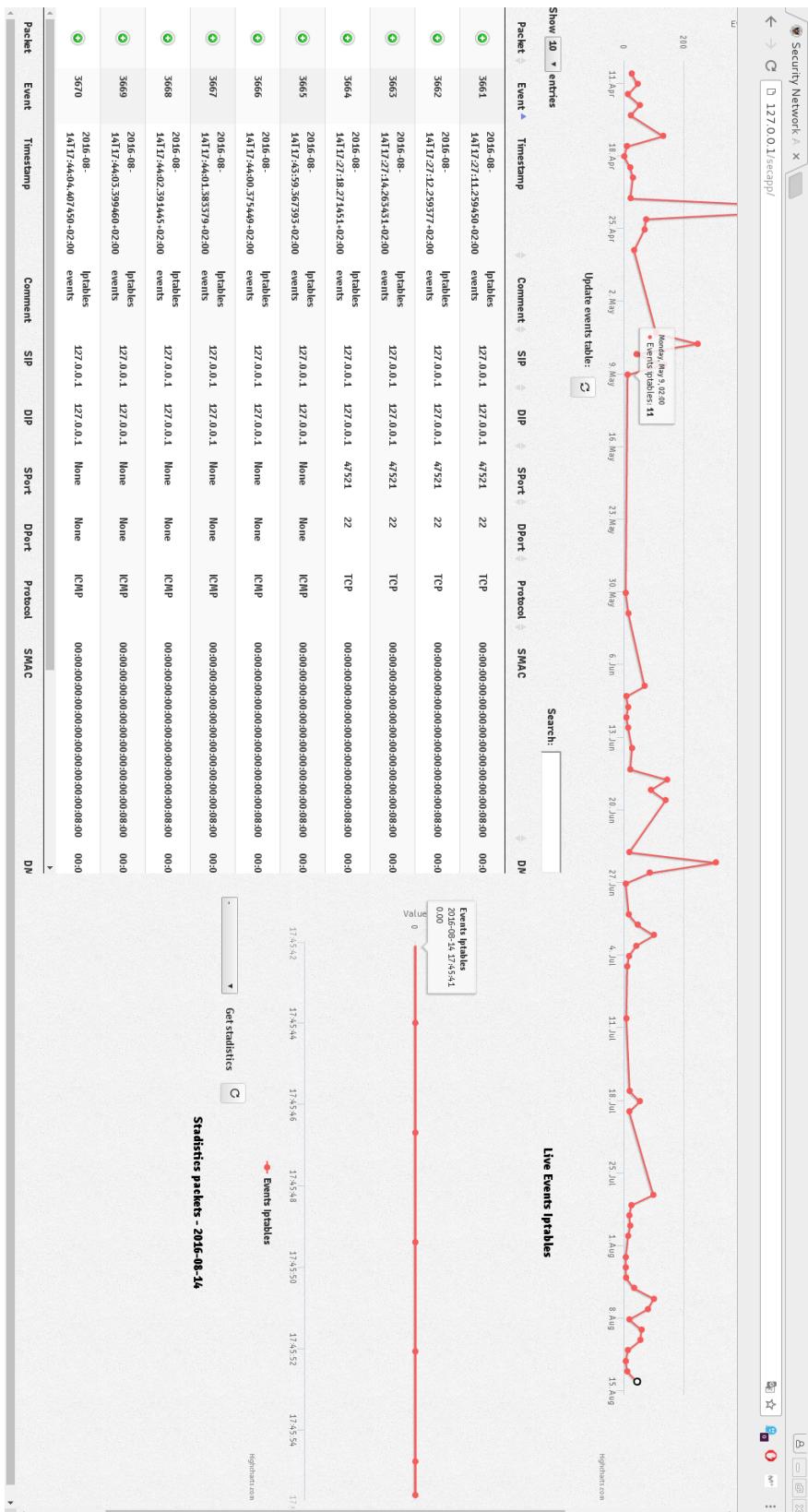


Figura 4.20: Vista principal con la información de eventos del día seleccionado

Vista con la información adicional por evento

Item de la tabla de paquetes en donde haciendo click (despliega) y doble click (pliega) la información adicional del paquete capturado en el procesamiento.

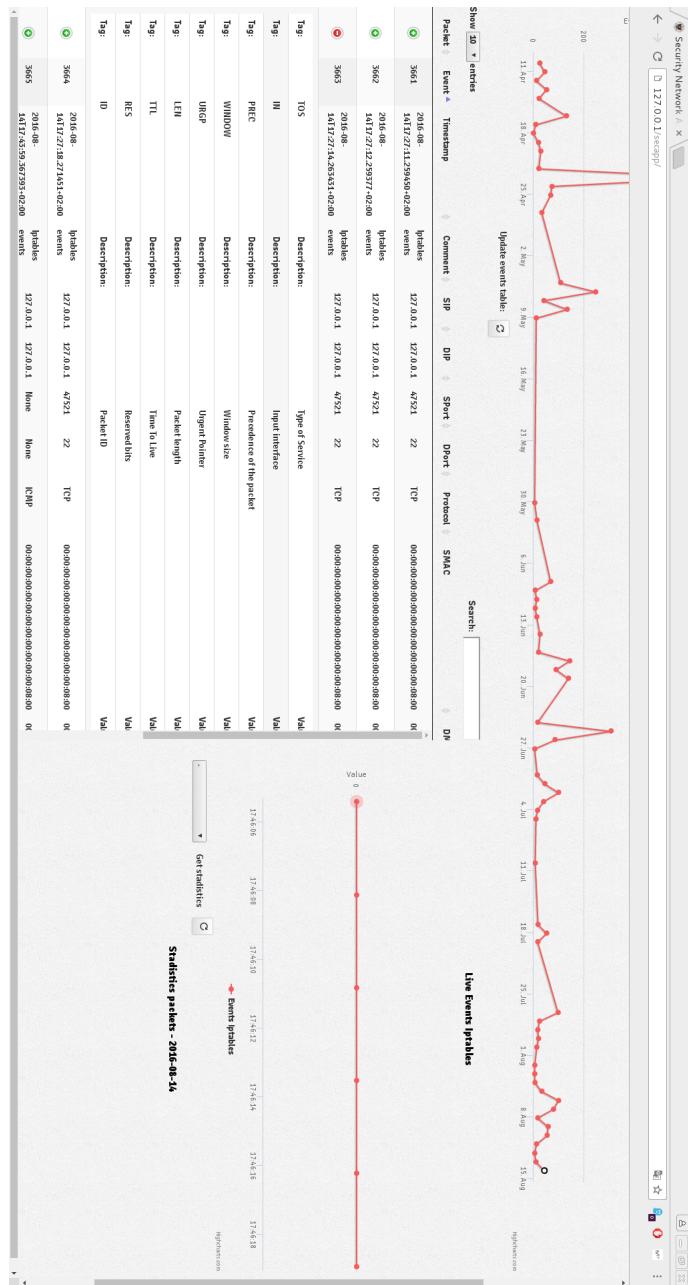


Figura 4.21: Vista principal con la información adicional por cada evento seleccionado

Vista con las estadísticas del día seleccionado

Gráfico en formato tarta (pie chart) con la información que el usuario ha podido elegir del ítem desplegable y que posteriormente ha generado al pulsar el botón.

4.4. DISEÑO DE LA VISTA

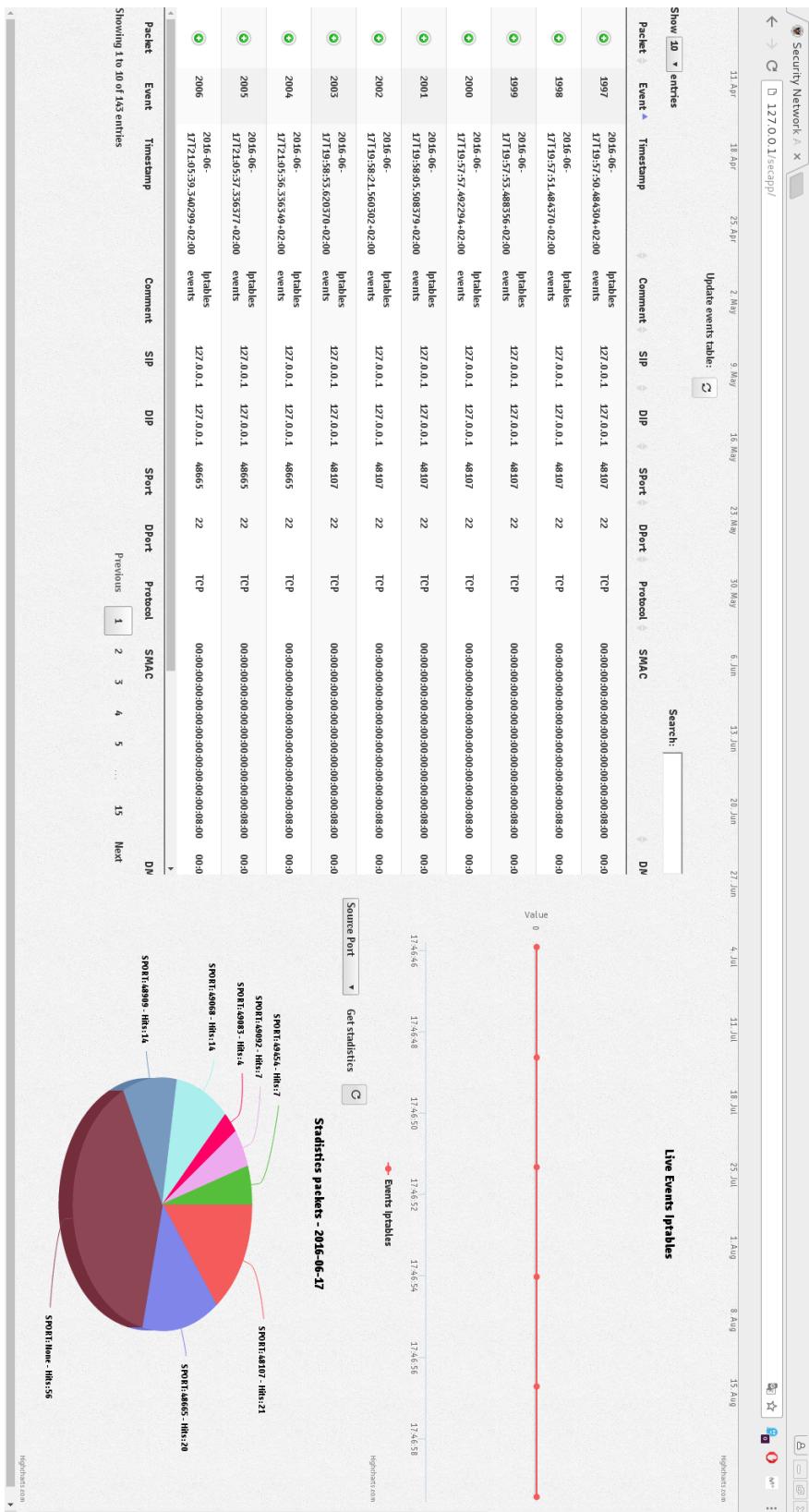


Figura 4.22: Vista principal con las estadísticas de los eventos del día seleccionado

Capítulo 5

Implementación

En este capítulo se definirán las configuraciones, diagramas e implementaciones que se han llevado a cabo en el proyecto. Primero, especificaremos los diagramas de clases para el BackEnd y FrontEnd (Flujo de ejecución Django) de la aplicación.

5.1. Flujo de ejecución BackEnd

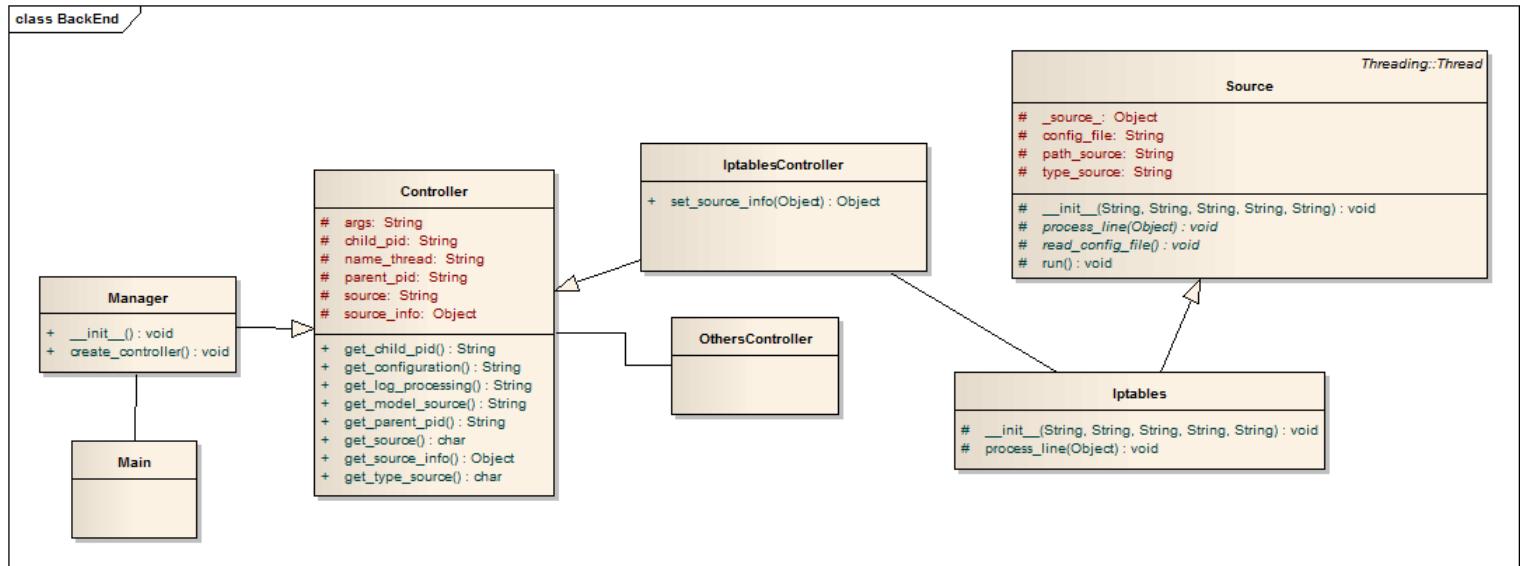


Figura 5.1: Diagrama de clases BackEnd

En el diagrama podemos ver de izquierda a derecha la herencia que hay entre las clases que se encargan de recoger la información de la sonda.

5.2. Flujo de ejecución FrontEnd

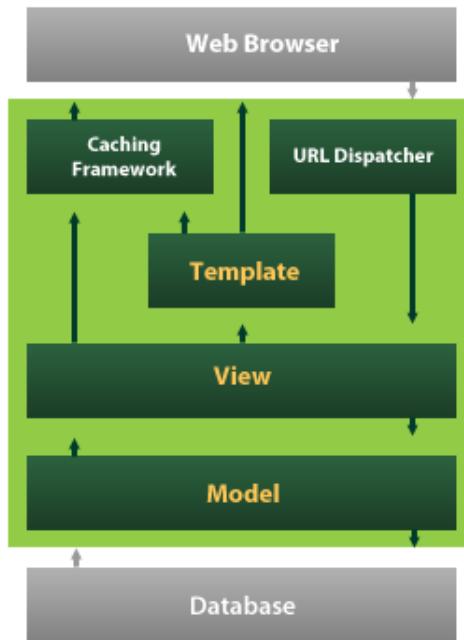


Figura 5.2: Flujo de ejecución del FrontEnd - [15]

Para este caso se ha decidido mostrar el flujo de ejecución que sigue una petición desde que llega al servidor de Django hasta que se sirve una vista asociada al navegador.

5.3. Configuración de la aplicación

En la fase de implementación del proyecto entran en juego la configuración previa de la máquina en la cual se van a desplegar los componentes para su utilización. Hay ciertos factores a tener en cuenta.

Dado que se ha desarrollado una primera versión funcional de la aplicación, en el futuro se podrían incluir ciertas herramientas software de automatización de despliegues y configuraciones como podrían ser:

5.3.1. Chef

Web: <https://www.chef.io/>



Chef es una plataforma de automatización de gran alcance que nos transforma el control de infraestructuras en código. Da igual el sistema operativo o plataforma sobre el que se tenga que operar, Chef automatiza configuraciones de infraestructuras, desplegados y gestionado a través de la red, sin importar el tamaño de la solución software a controlar.

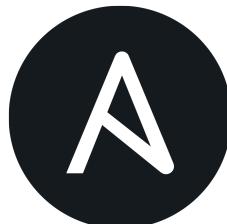
A continuación un diagrama explicando la arquitectura de un sistema bajo Chef:



Figura 5.3: Arquitectura de un sistema bajo Chef

5.3.2. Ansible

Web: <https://www.ansible.com/>



Ansible es una herramienta de código abierto que se utiliza para distribuir aplicaciones en nodos o servidores remotos de una manera automatizada. Nos proporciona un framework común a todas las aplicaciones, permitiendo así configurar cada una de ellas de una manera más fácil y

eficaz. Ansible se basa en “playbooks” (al contrario de Chef que se basa en Cookbooks) donde se pueden definir una amplia variedad de sistemas para el despliegue de una aplicación.

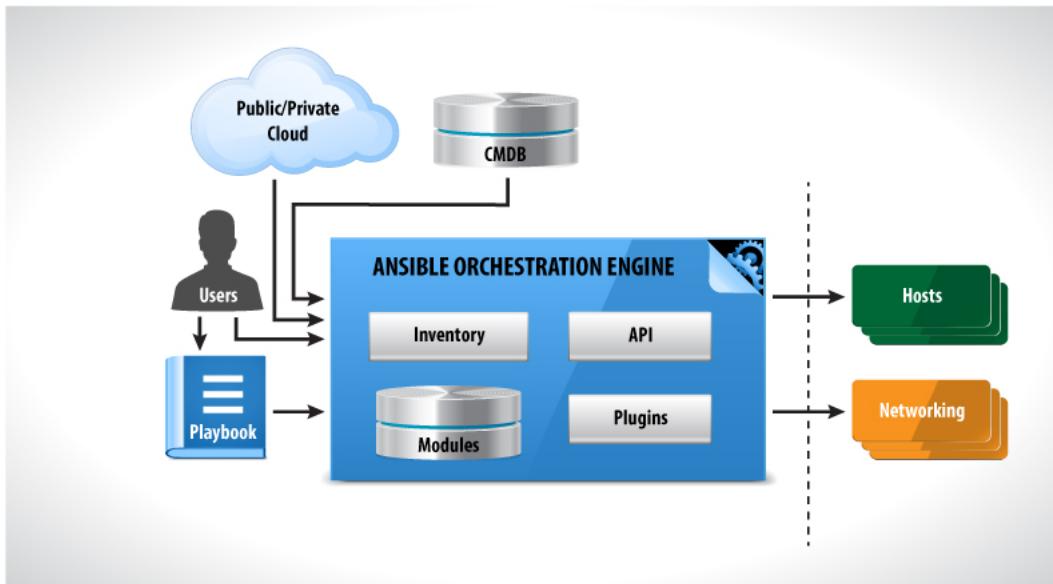
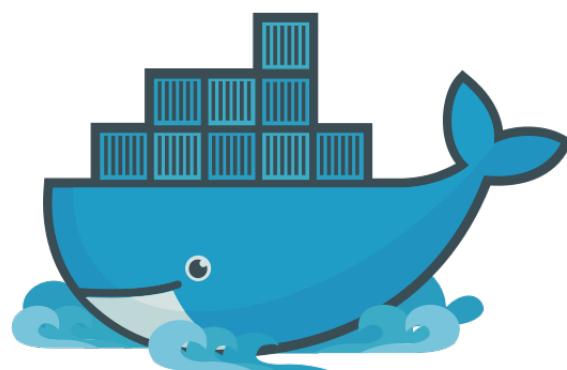


Figura 5.4: Diagrama de la arquitectura Ansible

5.3.3. Docker

Web: <https://www.docker.com/>



Docker es un sistema de desarrollo de sistemas completos, utilizando el concepto de contenedor (LXC) de los sistemas UNIX. Un contenedor de Docker, envuelve un fragmento de software en un sistema de archivos completo (imagen) que contiene todo lo necesario para funcionar: código, instancias de arranque, herramientas del sistema, bibliotecas del sistema... cualquier cosa que tendría un sistema operativo normal pero en una versión más optimizada y ligera (estos sistemas normalmente se obtienen de un Docker Repository y en su totalidad son sistemas open source). Esto garantiza que el software se ejecutará siempre de la misma forma, independientemente del medio en el que se despliegue.

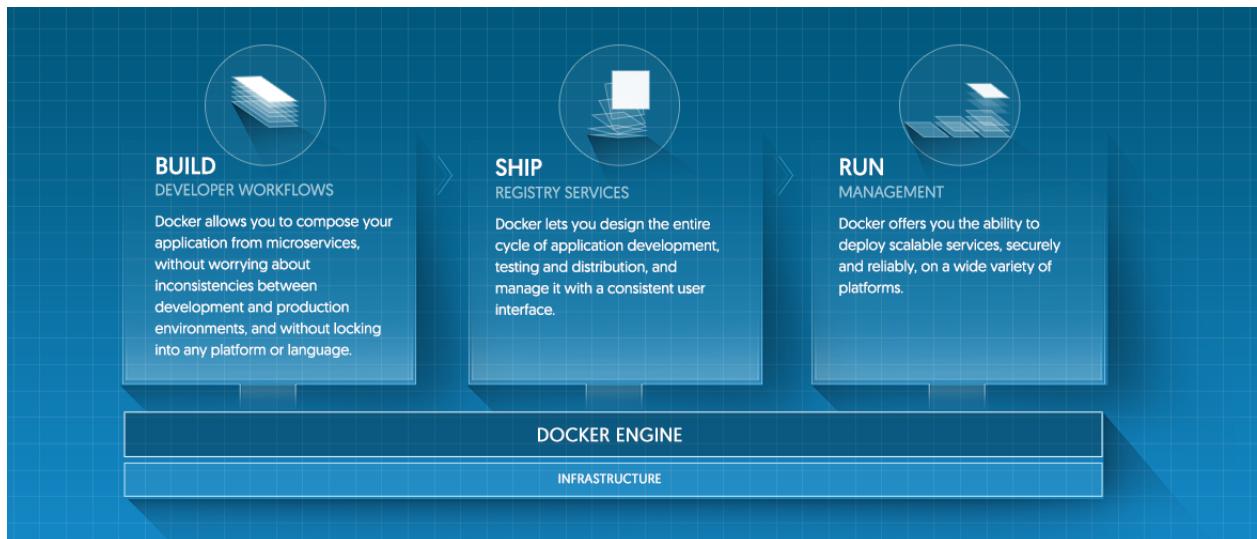


Figura 5.5: Diagrama de la arquitectura general de Docker

5.4. Configuración local para procesamiento

Los primeros pasos para la obtención de logs o eventos generados por la fuente de seguridad, iptables, serán los de configurar el sistema interno de correlación de logs rsyslog junto con el sistema de rotación de logs logrotate. Para el caso de rsyslog tenemos que definir un filtro para que cualquier evento que genere el sistema con un determinado mensaje definido en las reglas de iptables, sea capturado y almacenado en un determinado directorio. También hemos dotado a los logs del sistema de timestamp con mayor precisión para poder diferenciar eventos con mayor afinamiento y cambiado la tupla de permisos a la hora de crear un archivo con **FileCreateMode**.

```

1  #
2  # Use traditional timestamp format.
3  # To enable high precision timestamps, comment out the following
4  # line.
5  #
6  # $ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat
7  #
8  # Set the default permissions for all log files.
9  #
10 $FileOwner root
11 $FileGroup adm
12 $FileCreateMode 0644
13 $DirCreateMode 0755
14 $Umask 0022
15
16 # IPTABLES
17
18 :msg,contains,"IPTMSG= " -/var/log/iptables.log
19 :msg,regex,"^[\ *[\ 0-9]*\.[0-9]*\] IPTMSG= " -/var/log/iptables.log
20 :msg,contains,"IPTMSG= " ~

```

Figura 5.6: Configuración de iptables para Rsyslog

Hemos de configurar Logrotate (más información sobre los campos visitar sección [2.4.11](#)):

```

1 /var/log/iptables.log
2 {
3     rotate 7
4     daily
5     missingok
6     notifempty
7     delaycompress
8     compress
9     postrotate
10    invoke-rc.d rsyslog restart > /dev/null
11    endscript
12 }
```

Figura 5.7: Configuración de iptables para Logrotate

Tenemos que definir una regla específica para el demonio de rsyslog en la cual se filtre también por los campos que queramos de iptables:

```

1 # into separate file and stop their further processing
2 if ($syslogfacility-text == 'kern') and \\
3 ($msg contains 'IPTMSG=' and $msg contains 'IN=') \\
4 then      -/var/log/iptables.log
5 & ~
```

Figura 5.8: Configuración de iptables.conf para Rsyslog.d

Estos tres pasos o configuraciones nos permiten redireccionar un log de iptables al directorio `/var/log/` y en concreto para el archivo `iptables.log`. Esta configuración tendrá efecto una vez hayamos reiniciado el servicio rsyslog o en el próximo inicio de sesión sobre la máquina.

5.4.1. Recogida y almacenamiento de logs

Para el caso que nos ocupa, iptables tiene una forma de definir reglas internas para el filtrado de paquetes según el tipo de comunicación que se establezca contra la máquina o desde la máquina hacia el exterior.

```

1 iptables -A INPUT -p tcp -m tcp --dport 22 -j LOG --log-prefix
  "IPTMSG=Connection SSH"
```

Figura 5.9: Ejemplo de regla iptables

En la siguiente sección explicaremos en profundidad cada una de las opciones de la regla, pero a simple vista podemos observar que el mensaje asociado a la regla coincide con la palabra clave del filtro empleado en rsyslog: `IPTMSG=`. Así pues, una vez dicho evento se genere el sistema y syslog lo procese como un mensaje de un servicio determinado, en este caso iptables, syslog filtrará dicho mensaje según su configuración para almacenarlo posteriormente en `/var/log/iptables.log`.

5.4.2. Iptables

El servicio de firewall del kernel de GNU Linux, iptables, nos proporciona una interfaz de reglas y tablas en donde podemos definir patrones o reglas que actúen sobre el tráfico que llega o sale desde nuestra máquina. Las reglas que se han definido por defecto son las siguientes (si queremos más filtros hay que implicar al protocolo y su puerto asociado con un mensaje):

```

1 # Generated by iptables-save v1.4.21 on Mon Jan 25 20:37:18 2016
2 *filter
3 :INPUT ACCEPT [0:0]
4 :FORWARD ACCEPT [0:0]
5 :OUTPUT ACCEPT [0:0]
6 -A INPUT -d 127.0.0.1/32 -p icmp -m icmp --icmp-type 8 -m state
   --state NEW,RELATED,ESTABLISHED -j LOG --log-prefix
   "IPTMSG=Connection ICMP"
7 -A INPUT -d 127.0.0.1/32 -p icmp -m icmp --icmp-type 8 -m state
   --state NEW,RELATED,ESTABLISHED -j DROP
8 -A INPUT -p tcp -m tcp --dport 22 -j LOG --log-prefix
   "IPTMSG=Connection SSH"
9 -A INPUT -p tcp -m tcp --dport 22 -j DROP
10 COMMIT

```

Figura 5.10: Configuración reglas iptables

A continuación una breve explicación de cada opción o comando:

- -A: Añadir una nueva regla a una cadena de la tabla.
- -d: Especificación para la ip destino, en este caso localhost con una máscara de subred determinada.
- -dport: Especificación para el puerto destino al que se realizará la posible conexión o envío de paquetes.
- -p: Especificación del protocolo del paquete.
- -m: Especificación de matching, en este caso icmp o tcp dentro de la descripción del paquete.
- -icmp-type: Extensión del tipo de ping que se va a procesar desde la regla.
- -state: Tipo de paquete según conexión:
 - NEW: Paquete que crea una nueva conexión.
 - RELATED: Paquete que está relacionado a una conexión existente, pero que no es parte de ella, como un error ICMP o, un paquete que establece una conexión de datos FTP.
 - ESTABLISHED: Paquete que pertenece a una conexión existente (que tuvo paquetes de respuesta).
 - INVALID (no usado): Paquete que no pudo ser identificado por alguna razón: quedar sin memoria o errores ICMP que no corresponden a ninguna conexión conocida. Normalmente estos paquetes deben ser descartados.

- -j: Acción de salto cuando encuentre dicha regla de paquetes. Se especifican dos acciones:
 - ◊ LOG –log-prefix “message” : Cuando se encuentre dicha regla se recolecta como log de la misma adjuntando un mensaje para diferenciarla del resto de reglas.
 - ◊ DROP : Cuando se encuentre dicha regla se descarta el paquete dentro del propio sistema. Al ir seguido LOG de un DROP el paquete se muestra en el registro de syslog para posteriormente ser eliminado del registro de almacenamiento de paquetes (Para esto usamos rsyslog que se encarga de almacenar dicho mensaje en un archivo de log).

Así pues, una vez tengamos un evento o paquete o log de iptables en nuestro sistema generados mediante ssh 127.0.0.1 o ping 127.0.0.1 obtendremos lo siguiente:

```

1 [dom jul  3 17:04:03 2016] IPTMSG=Connection SSH IN=lo OUT=
  MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00 SRC=127.0.0.1
  DST=127.0.0.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=39454 DF
  PROTO=TCP SPT=47706 DPT=22 WINDOW=43690 RES=0x00 SYN URGP=0
2 [dom jul  3 17:04:05 2016] IPTMSG=Connection SSH IN=lo OUT=
  MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00 SRC=127.0.0.1
  DST=127.0.0.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=39455 DF
  PROTO=TCP SPT=47706 DPT=22 WINDOW=43690 RES=0x00 SYN URGP=0

```

Figura 5.11: Evento de ssh localhost en el sistema

Lo anterior correspondía a la salida del comando **\$ dmesg -T** que muestra todos los mensajes del sistema que se han pasado al syslog. La opción -T se usa para especificar el timestamp de cada mensaje con una mayor precisión.

```

1 2016-07-03T17:03:35.664324+02:00 debian kernel: [23337.363387]
  IPTMSG=Connection SSH IN=lo OUT=
  MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00 SRC=127.0.0.1
  DST=127.0.0.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=39454 DF
  PROTO=TCP SPT=47706 DPT=22 WINDOW=43690 RES=0x00 SYN URGP=0
2 2016-07-03T17:03:37.668326+02:00 debian kernel: [23339.369692]
  IPTMSG=Connection SSH IN=lo OUT=
  MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00 SRC=127.0.0.1
  DST=127.0.0.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=39455 DF
  PROTO=TCP SPT=47706 DPT=22 WINDOW=43690 RES=0x00 SYN URGP=0

```

Figura 5.12: Log capturado y almacenado por rsyslog en /var/log/iptables.log

Lo anterior corresponde con la manipulación por parte de rsyslog del mensaje obtenido en syslog. Como podemos observar se añade un campo de timestamp de mayor precisión según la configuración que hemos establecido en rsyslog.conf para poder diferenciar con mayor exactitud eventos entre diferentes franjas de tiempo.

```

1 ++++++
2 -----
3
4 Procesando linea --> 2016-07-03T17:12:53.632264+02:00 debian kernel:
[23896.003739] IPTMSG=Connection ICMP IN=lo OUT=
MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00 SRC=127.0.0.1
DST=127.0.0.1 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=64157 DF
PROTO=ICMP TYPE=8 CODE=0 ID=14177 SEQ=7
5 -----
6 -----
7 ++++++
8 ---> Insertado registro: {'TAG': 'Connection ICMP', 'ID_Source_PORT': None,
  'Protocol': u'ICMP', 'RAW_Info': '2016-07-03T17:12:53.632264+02:00 debian kernel 23896.003739
  IPTMSG=Connection ICMP IN=lo OUT
  MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00 SRC=127.0.0.1
  DST=127.0.0.1 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=64157 DF
  PROTO=ICMP TYPE=8 CODE=0 ID=14177 SEQ=7', 'ID_Source_MAC': <Macs: 00:00:00:00:00:00:00:00:00:00:00:08:00>, 'ID_Source_IP': <Ips: 127.0.0.1>, 'ID_Dest_IP': <Ips: 127.0.0.1>, 'ID_Dest_PORT': None,
  'ID_Dest_MAC': <Macs: 00:00:00:00:00:00:00:00:00:00:00:08:00>}
9 -----
10 ++++++
11 ---> Fin de procesado de linea

```

Figura 5.13: Procesamiento del log capturado y almacenado en la bd interna de la aplicación

5.4.3. Transformación de log en información útil: Parser

Una vez hemos descrito los pasos a seguir para obtener un log para un evento iptables, llega la hora de procesar y hacer útil todos esos datos que tenemos. Para esta finalidad tenemos que usar expresiones regulares para generar un parser con el que ser capaz de traducir todos esos datos en información útil para nuestra aplicación.

```

1 obj = Pygtail("/var/log/iptables.log")
2
3 while True:
4     try:
5         for line in obj:
6             if len(line) > 1:
7                 self.process_line(line)
8     except Exception as ex:
9         print "Pygtail processing -> ", ex

```

Figura 5.14: Instancia de la clase Pygtail y lectura de las líneas del log

Hacemos uso del módulo o paquete [Pygtail](#) que nos permite leer archivos de log cuyos registros internos aún no han sido leídos. Es una especie de `$ tail -f` a un archivo en concreto, pero usa un concepto de offset e inode para saber la última actualización y posición del archivo antes

y después de ser abierto para así saber que parte de la última lectura se quedo en ejecución. Éste es otro punto importante dado que para hacer uso de esta funcionalidad debemos crear un archivo con el nombre del log, véase `/var/log/iptables.log` cuya extensión final sea offset (`/var/log/iptables.log.offset`) y sus permisos los siguientes:

```
1 -rw-r--r-- 1 root adm 10391 jul  3 17:12 /var/log/iptables.log
2 -rw-r--rw- 1 root root 14 jul  3 17:13 /var/log/iptables.log.offset
```

Figura 5.15: Permisos de los archivos `iptables.log` e `iptables.log.offset`

Una vez tengamos el archivo abierto y con sus líneas procesadas por Pygtail, es el momento de usar regex sobre los objetos string de cada línea de log. Para ello hacemos uso del método `split` para dividir por palabras, es decir, posiciones separadas en una lista a todas las coincidencias con una palabra que pudiera tener toda la cadena.

```
1 # Se trocea por palabras el log leido
2 line = re.split("\W? ", line)
```

Figura 5.16: Uso del método `split` sobre la entrada de lineas de log

Una vez separado por palabras toda la línea de log, pasamos a diferenciar entre las etiquetas que iptables pone a cada campo con su valor, es decir, una tupla `key=>value`.

```
1 tag_str = ((re.compile('^(.*=)').search(str(line))).group(0)
2 tag_split = tag_str.split(',') )
```

Figura 5.17: Obtenemos la tupla `key=>value` para cada etiqueta del log

Ya tenemos todas las etiquetas de los campos del log y ahora nos toca extraer su valor asociado para asignarlo al ORM de la base de datos, es decir, almacenar la información en la BD.

```

1 db_column = [ 'ID_Source_IP' , 'ID_Dest_IP' ,
2               'ID_Source_PORT' , 'ID_Dest_PORT' ,
3               'Protocol' , 'ID_Source_MAC' ,
4               'ID_Dest_MAC' ]
5
6 # El nombre de las tags, segun el orden de la
7 # columnas en db_column, las extraigo del fichero
8 # de configuracion a traves del registro info_config_file
9
10 labels = [self.info_config_file["Source_Ip"] ,
11            self.info_config_file["Dest_Ip"] ,
12            self.info_config_file["Source_Port"] ,
13            self.info_config_file["Dest_Port"] ,
14            self.info_config_file["Protocol"]]
15
16 # Almacenamos las etiquetas o campos del log de iptables
17 for it in tag_split:
18     if len(it.split('=')) == 2:
19         self.tag_log.append((it.split('='))[0].strip('\' '))
20
21 # Buscamos la correlacion entre los campos definidos en la
22 # configuracion con los extraidos del log de iptables
23
24 for label in labels:
25     if (re.compile(label)).search(tag_str):
26         if self.tag_log.index(label) > 0:
27             db_column_name = db_column[0]
28             register[db_column.pop(0)] = self.regexp(db_column_name ,
29                                           label, str(line))
30             self.tag_log.remove(label)
31     else:
32         register[db_column.pop(0)] = None

```

Figura 5.18: Vamos asignando cada etiqueta y su valor a su asociado del ORM

Los siguientes pasos de comprobación de integridad de valores y demás se relegan a la visualización del interesado en el método process_line del archivo código fuente iptables.py

5.4.4. Workflow

Ahora vamos a detallar los pasos que seguirá la aplicación a la hora de la visualización de información en la web mediante el framework Django. Pero primero cómo se procesan y almacenan los datos de los logs:

- Primera instancia a ejecutar main.py: Se definen las llamadas al sistema Manager que se encargará a su vez de llamar al controlador que se asocia con la fuente encargada (iptables).
- Segunda instancia a ejecutar manager.py: Gestión de los controladores que hacen uso de la herramienta. Cada controlador tendrá asociada su fuente.

- Tercera instancia a ejecutar controller.py: Esta clase contiene unas características que herederán los controladores de cada fuente y a su vez lanza la ejecución de los hilos para cada una de las fuentes. Aquí se define o exporta la variable de entorno de Django para utilizar el ORM del mismo.
- Cuarta instancia a ejecutar source.py: Aunque se vaya directamente al siguiente punto, primero se pasa el control a esta clase de la cuál hereda iptables y se crean los fragmentos de código para la ejecución en iptables.
- Quinta instancia a ejecutar iptables.py: Ejecución y configuración de todo el entramado de iptables, de la insercción de datos en la BD y de la extracción de características de los logs.

Ahora corresponde el turno al flujo de ejecución para la visualización web:

- Primera instancia a ejecutar manage.py: Aquí se cargan las configuraciones por defecto del proyecto Django para la aplicación definida.
- Segunda instancia a ejecutar wsgi.py: Se usa la biblioteca wsgi para poder lanzar la aplicación por su nombre dentro del namespace definido por Django.
- Tercera instancia a ejecutar settings.py: Se cargan todas las configuraciones establecidas por el usuario para el proyecto y que a su vez usarán como configuración base para las aplicaciones derivadas del mismo.
- Cuarta instancia a ejecutar urls.py (archivo que pertenece al proyecto Django): Aquí se encamina la visualización entre la aplicación o la parte de administración del proyecto.
- Quinta instancia a ejecutar urls.py (archivo que pertenece a la aplicación): Aquí es dónde se define el router de la aplicación web y los métodos que se lanzarán una vez se hagan las peticiones sobre el servidor web Django.
- Sexta instancia a ejecutar views.py: Se ejecuta el método asociado a la renderización de la vista, que por defecto, sino hay ninguna ruta será index. Dependiendo del método a ejecutar sólo mostrará contenido estático o dinámico (JSON) obtenido de la base de datos.
- Séptima instancia a ejecutar aplicacion/index.html (Paso previo de MasterPage.html): Se carga el trozo de esta plantilla o template (que en nuestro caso es una vista) que contiene el esqueleto principal de la parte web para todas las vistas/plantillas que heredan de esta general.
- Octava instancia a ejecutar aplicacion/index.html (Carga de la plantilla index.html): Se carga el trozo de esta plantilla dentro del bloque definido en la plantilla MasterPage que anteriormente se proceso.

5.4.5. Visualización de eventos

Aquí tenemos una vista general de todas las interacciones posibles con la web de los eventos almacenados en el sistema.

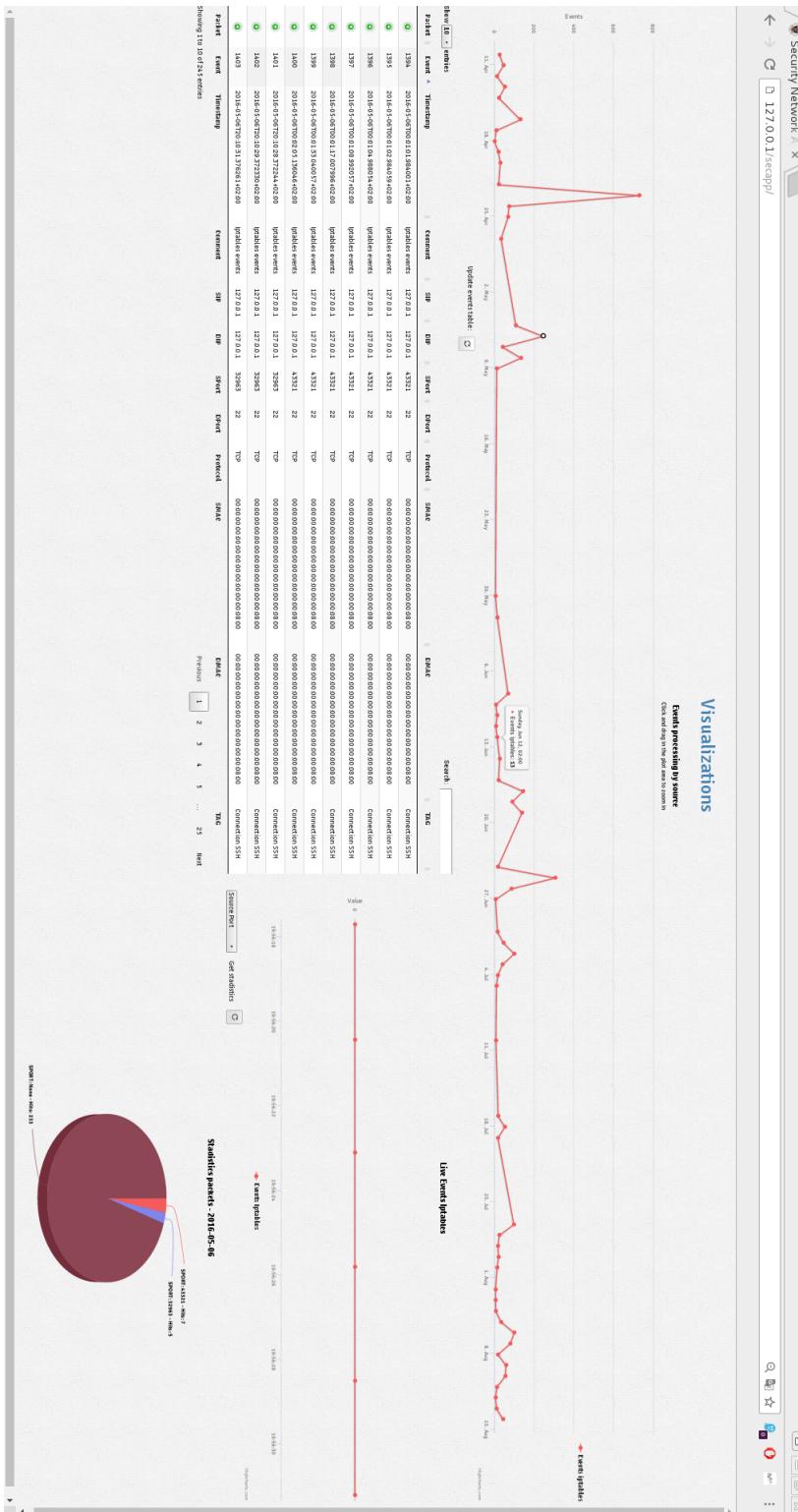


Figura 5.19: Vista general

Capítulo 6

Evaluación

A continuación se dejan algunos ejemplos de las pruebas funcionales. Estos son test unitarios que se han realizado sobre la parte más importante del proyecto que es la base de datos y la integridad de dichos datos a la hora de poder operar con ellos desde la api/interfaz web.

Para una mayor profundidad en su estudio, estos se encuentran en el archivo “tests.py” dentro de la jerarquía del código fuente y también en el Apéndice 2: Tests [10]. Pasemos a analizar cada uno de los test realizados sobre el modelo PacketEventsInformation de la base de datos.

6.1. Test: PacketEventsInformation

Clase del modelo relacional ORM - PacketEventsInformation. Es la clase que almacena todas las referencias al resto de clases internas de la arquitectura del sistema de base de datos.

```
1 class PacketEventsInformationTestCase(TestCase):
2
3     # Atributos internos de la clase
4
5     timestamp = timezone.now()
6     timestamp_insertion = timezone.now()
```

6.1.1. Método: setUp

Con éste método se crean las instancias temporales en la base de datos **test** a la hora de la ejecución de la misma para comprobar su integridad. Para este caso tenemos que crear instancias para todas las referencias que contiene un objeto de la clase PacketEventsInformation.

```
1 def setUp(self):
2     ip_source = Ips.objects.create(Ip="127.0.0.2",
3                                     Hostname="localhost", Tag="localhost")
4     ip_dest = Ips.objects.create(Ip="127.0.0.3", Hostname="localhost",
5                                  Tag="localhost")
6     port_source = Ports.objects.create(Tag="ftp")
7     port_dest = Ports.objects.create(Tag="ssh")
8     mac_source = Macs.objects.create(
9         MAC="00:00:00:00:00:00:00:00:00:00:00:08:00",
```

```

8         TAG="Mac local1"
9     )
10    mac_dest = Macs.objects.create(
11        MAC="00:00:00:00:00:00:00:00:00:00:00:08:00",
12        TAG="Mac local2"
13    )
14    log_sources = LogSources.objects.create(
15        Description="Firewall of gnu/linux kernel",
16        Type="Iptables",
17        Model="iptables v1.4.21",
18        Active=1,
19        Software_Class="Firewall",
20        Path="iptables",
21    )
22    event = Events.objects.create(
23        Timestamp=self.timestamp,
24        Timestamp_Insertion=self.timestamp_insertion,
25        ID_Source=log_sources,
26        Comment="Iptables Events",
27    )
28    PacketEventsInformation.objects.create(
29        ID_IP_Source=ip_source,
30        ID_IP_Dest=ip_dest,
31        ID_Source_Port=port_source,
32        ID_Dest_Port=port_dest,
33        Protocol="ICMP",
34        ID_Source_MAC=mac_source,
35        ID_Dest_MAC=mac_dest,
36        RAW_Info="LOG RAW INFO",
37        TAG="Connection ICMP",
38        id=event
39    )

```

- ip-source: Instancia Clase Ips.
- ip-dest: Instancia Clase Ips.
- port-source: Instancia Clase Ports.
- port-dest: Instancia Clase Ports.
- mac-source: Instancia Clase Macs.
- mac-dest: Instancia Clase Macs.
- log-sources: Instancia Clase LogSources.
- event: Instancia Clase Events.
- PacketEventsInformation: Instancia a su propia clase.

Los métodos siguientes sirven para la consecución del test a la clase.

6.1.2. Método: test_id_ip_source

Descripción: Comprobación de que el objeto de ip origen (referencia al modelo relacional) coincide con su asociado.

Pasos:

- Se consulta sobre la instancia Ips creada anteriormente, en el método setUp, si existe algún campo (**Ip**) cuyo valor coincida con el valor de: “127.0.0.2”.
- Se consulta sobre la instancia PacketEventsInformation creada anteriormente, si existe una coincidencia para el campo identificador **ID_IP_Source** con una referencia de una instancia similar, en este caso: **ip_source**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de Ips como la consulta de la referencia interna asociada dentro de PacketEventsInformation.

```

1 def test_packeteventsinformation_id_ip_source(self):
2     """
3         Comprobacion de que el objeto de ip origen (referencia al modelo
4             relacional) coincide con su asociado
5     Returns:
6         """
7     ip_source = Ips.objects.get(Ip="127.0.0.2")
8     pei = PacketEventsInformation.objects.get(ID_IP_Source=ip_source)
9     self.assertEqual(pei.get_id_ip_source(), ip_source)

```

6.1.3. Método: test_id_ip_dest

Descripción: Comprobación de que el objeto de ip destino (referencia al modelo relacional) coincide con su asociado.

Pasos:

- Se consulta sobre la instancia Ips creada anteriormente, en el método setUp, si existe algún campo (**Ip**) cuyo valor coincida con el valor de: “127.0.0.3”.
- Se consulta sobre la instancia PacketEventsInformation creada anteriormente, si existe una coincidencia para el campo identificador **ID_IP_Dest** con una referencia de una instancia similar, en este caso: **ip_dest**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de Ips como la consulta de la referencia interna asociada dentro de PacketEventsInformation.

```

1 def test_packeteventsinformation_id_ip_dest(self):
2     """
3         Comprobacion de que el objeto de ip destino (referencia al modelo
4             relacional) coincide con su asociado
5     Returns:
6         """
7     ip_dest = Ips.objects.get(Ip="127.0.0.3")

```

```

8     pei = PacketEventsInformation.objects.get(ID_IP_Dest=ip_dest)
9     self.assertEqual(pei.get_id_ip_dest(), ip_dest)

```

6.1.4. Método: test_id_source_port

Descripción: Comprobación de que el objeto de puerto origen (referencia al modelo relacional) coincide con su asociado.

Pasos:

- Se consulta sobre la instancia Ports creada anteriormente, en el método setUp, si existe algún campo (**Tag**) cuyo valor coincida con el valor de: “ftp”.
- Se consulta sobre la instancia PacketEventsInformation creada anteriormente, si existe una coincidencia para el campo identificador **ID_Source_Port** con una referencia de una instancia similar, en este caso: **port_source**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de Ports como la consulta de la referencia interna asociada dentro de PacketEventsInformation.

```

1 def test_packeteventsinformation_id_source_port(self):
2     """
3         Comprobacion de que el objeto de puerto origen (referencia al
4             modelo relacional) coincide con su asociado
5     Returns:
6     """
7     port_source = Ports.objects.get(Tag="ftp")
8     pei =
9         PacketEventsInformation.objects.get(ID_Source_Port=port_source)
self.assertEqual(pei.get_id_source_port(), port_source)

```

6.1.5. Método: test_id_dest_port

Descripción: Comprobación de que el objeto de puerto destino (referencia al modelo relacional) coincide con su asociado.

Pasos:

- Se consulta sobre la instancia Ports creada anteriormente, en el método setUp, si existe algún campo (**Tag**) cuyo valor coincida con el valor de: “ssh”.
- Se consulta sobre la instancia PacketEventsInformation creada anteriormente, si existe una coincidencia para el campo identificador **ID_Dest_Port** con una referencia de una instancia similar, en este caso: **port_dest**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de Ports como la consulta de la referencia interna asociada dentro de PacketEventsInformation.

```

1 def test_packeteventsinformation_id_dest_port(self):
2     """
3         Comprobacion de que el objeto de puerto destino (referencia al
4             modelo relacional) coincide con su asociado
5         Returns:
6         """
7     port_dest = Ports.objects.get(Tag="ssh")
8     pei = PacketEventsInformation.objects.get(ID_Dest_Port=port_dest)
9     self.assertEqual(pei.get_id_dest_port(), port_dest)

```

6.1.6. Método: test_protocol

Descripción: Comprobación de que el protocolo del paquete coincide con su asociado.

Pasos:

- Se consulta sobre la instancia PacketEventsInformation creada anteriormente, en el método setUp, si existe algún campo (**Protocol**) cuyo valor coincida con el valor de: “ICMP”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información comparando que la devolución del método interno de la clase PacketEventsInformation es igual al argumento pasado al método: “ICMP”.

```

1 def test_packeteventsinformation_protocol(self):
2     """
3         Comprobacion de que el protocolo del paquete coincide con su
4             asociado
5         Returns:
6         """
7     pei = PacketEventsInformation.objects.get(Protocol="ICMP")
8     self.assertEqual(pei.get_protocol(), "ICMP")

```

6.1.7. Método: test_id_source_mac

Descripción: Comprobación de que el objeto de mac origen (referencia al modelo relacional) coincide con su asociado.

Pasos:

- Se consulta sobre la instancia Macs creada anteriormente, en el método setUp, si existe algún campo (**TAG**) cuyo valor coincida con el valor de: “Mac local1”.
- Se consulta sobre la instancia PacketEventsInformation creada anteriormente, si existe una coincidencia para el campo identificador **ID_Source_MAC** con una referencia de una instancia similar, en este caso: **mac_source**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de Macs como la consulta de la referencia interna asociada dentro de PacketEventsInformation.

```

1 def test_packeteventsinformation_id_source_mac(self):
2     """
3         Comprobacion de que el objeto de mac origen (referencia al modelo
4             relacional) coincide con su asociado
5         Returns:
6             """
7     mac_source = Macs.objects.get(TAG="Mac local1")
8     pei = PacketEventsInformation.objects.get(ID_Source_MAC=mac_source)
9     self.assertEqual(pei.get_id_source_mac(), mac_source)

```

6.1.8. Método: test_id_dest_mac

Descripción: Comprobación de que el objeto de mac destino (referencia al modelo relacional) coincide con su asociado.

Pasos:

- Se consulta sobre la instancia Macs creada anteriormente, en el método setUp, si existe algún campo (**TAG**) cuyo valor coincida con el valor de: “Mac local2”.
- Se consulta sobre la instancia PacketEventsInformation creada anteriormente, si existe una coincidencia para el campo identificador **ID_Dest_MAC** con una referencia de una instancia similar, en este caso: **mac_dest**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de Macs como la consulta de la referencia interna asociada dentro de PacketEventsInformation.

```

1 def test_packeteventsinformation_id_dest_mac(self):
2     """
3         Comprobacion de que el objeto de mac destino (referencia al modelo
4             relacional) coincide con su asociado
5         Returns:
6             """
7     mac_dest = Macs.objects.get(TAG="Mac local2")
8     pei = PacketEventsInformation.objects.get(ID_Dest_MAC=mac_dest)
9     self.assertEqual(pei.get_id_dest_mac(), mac_dest)

```

6.1.9. Método: test_raw_info

Descripción: Comprobación de que el log en formato RAW coincide con su asociado.

Pasos:

- Se consulta sobre la instancia PacketEventsInformation creada anteriormente, en el método setUp, si existe algún campo (**RAW_Info**) cuyo valor coincida con el valor de: “LOG RAW INFO”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información comparando que la devolución del método interno de la clase PacketEventsInformation es igual al argumento pasado al método “LOG RAW INFO”.

```

1 def test_packeteventsinformation_raw_info(self):
2     """
3         Comprobacion de que el log en formato RAW coincide con su asociado
4     Returns:
5
6     """
7     pei = PacketEventsInformation.objects.get(RAW_Info="LOG RAW INFO")
8     self.assertEqual(pei.get_raw_info(), "LOG RAW INFO")

```

6.1.10. Método: test_tag

Descripción: Comprobación de que la etiqueta especificada para el paquete coincide con su asociado.

Pasos:

- Se consulta sobre la instancia PacketEventsInformation creada anteriormente, en el método setUp, si existe algún campo (**Tag**) cuyo valor coincide con el valor de: “Connection ICMP”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información comparando que la devolución del método interno de la clase PacketEventsInformation es igual al argumento pasado al método “Connection ICMP”.

```

1 def test_packeteventsinformation_tag(self):
2     """
3         Comprobacion de que la etiqueta especificada para el paquete
4             coincide con su asociado
5     Returns:
6
7     """
8     pei = PacketEventsInformation.objects.get(TAG="Connection ICMP")
9     self.assertEqual(pei.get_tag(), "Connection ICMP")

```

6.1.11. Método: test_id

Descripción: Comprobación de que el objeto de ip origen (referencia al modelo relacional) coincide con su asociado.

Pasos:

- Se consulta sobre la instancia Events creada anteriormente, en el método setUp, si existe algún campo (**Timestamp**) cuyo valor coincida con el valor de: Timestamp del sistema en la hora actual que se almacenó como atributo interno de la clase Test.
- Se pasa como parámetro la referencia de la instancia anterior a la instancia temporal de la clase PacketEventsInformation (la destinataria del test).
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de Events como la devolución de la referencia interna asociada dentro de PacketEventsInformation.

```
1 def test_packeteventsinformation_id(self):
2     """
3         Comprobacion de que el objeto de ip origen (referencia al modelo
4             relacional) coincide con su asociado
5         Returns:
6         """
7     event = Events.objects.get(Timestamp=self.timestamp)
8     pei = PacketEventsInformation.objects.get(id=event)
9     self.assertEqual(pei.get_id(), event)
```

Capítulo 7

Planificación temporal

La planificación general del proyecto siguiendo un modelo SCRUM, para su aprendizaje, se encuentra en un panel de tareas en Taiga

<https://tree.taiga.io/project/mgautier-proyecto-fin-de-carrera/backlog>.

Taiga [2.4.18], es un gestor de proyectos que permite aplicar una metodología de desarrollo ágil SCRUM o Kanban para la realización del mismo.

PROYECTO FIN DE CARRERA DESARROLLO FINAL APLICACIÓN 02 NOV. 2015-31 AGO. 2016

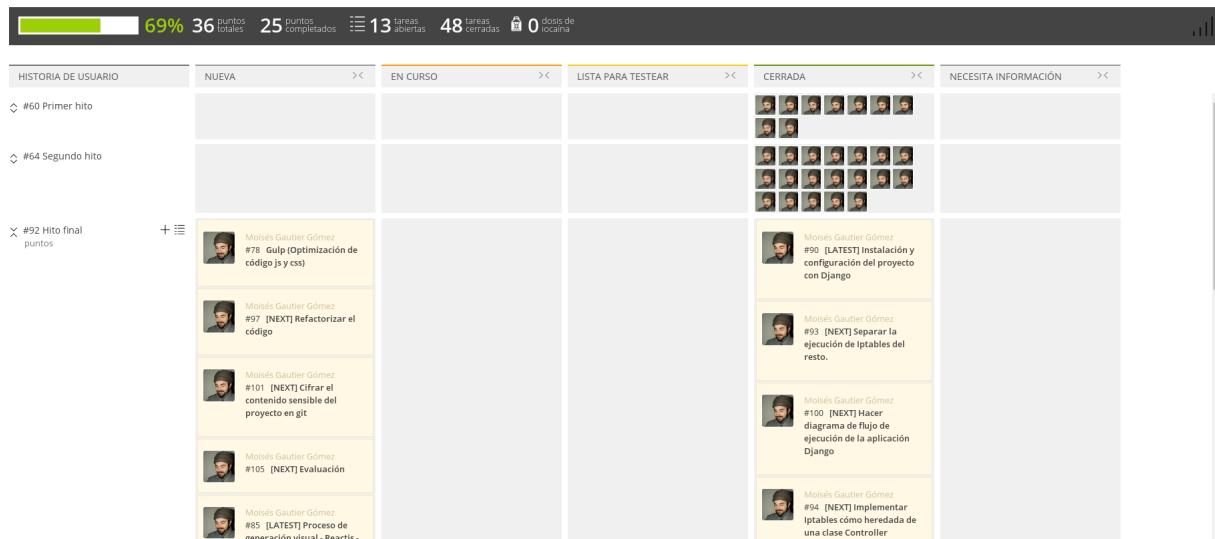


Figura 7.1: Panel de actividades - Taiga

Los puntos más importantes del proyecto se han dividido en hitos, así como entregas que se definieron en cada reunión que se realizaba con el director del proyecto. También se ha definido una planificación temporal del desarrollo del proyecto mediante un diagrama de gantt con la duración de las tareas definidas en el panel de actividades de Taiga.

7.1. Planificación temporal de tareas

A continuación se definirán los diferentes hitos que componen el diagrama de Gantt así como las fechas estimadas para los cuales se realizaron los hitos:

7.1.1. Hito 1: 1 Mayo a 1 Julio 2015

En esta primera etapa de desarrollo del proyecto final de carrera, se realizaron los estudios de las tecnologías a implementar, la arquitectura del sistema, las tecnologías de BD, visualización, etc.

7.1.2. Hito 2: 1 Julio a 1 Septiembre 2015

Implementación de los primeros componentes del sistema, como la BD. Definición de las tecnologías finales a implementar después del estudio del estado del arte realizado en el hito anterior. Gestión del código fuente bajo control de versiones Git y estudio de las tecnologías de recogida y correlación de logs con rsyslog. También se diseñan las reglas de filtrado de paquetes para la fuente de seguridad: Iptables.

7.1.3. Hito 3: 1 Septiembre a 1 Noviembre 2015

Se definen las clases del sistema que harán uso de la fuente de seguridad (Iptables, Source, Controller, etc). Se redefine el modelo de la BD.

7.1.4. Hito 4: 1 Noviembre 2015 a 1 Enero 2016

Primera aproximación funcional de la recolección (Pygtail) y correlación de eventos de Iptables en el sistema almacenados en la BD. Una vez la aplicación funciona mediante el uso del terminal se realiza estudio de la mejor forma de adaptar dicha aplicación a formato web con peticiones HTTP para visualizar los eventos generados.

7.1.5. Hito 5: 1 Enero a 1 Marzo 2016

Se decide implementar la solución web con el framework de desarrollo Django para el lenguaje Python. Se realizan los primeros estudios de bibliotecas gráficas que permitan la visualización de eventos en la interfaz web: D3js, C3js, Reactjs, etc.

7.1.6. Hito 6: 1 Marzo a 1 Mayo 2016

Proceso de desarrollo web y generación de visualización de los eventos del sistema en la interfaz web usando la tecnología de C3js y Reactjs. Se comienza a definir la estructura principal de la memoria para disponer de la mayor puntos posibles de cara a una posible defensa en Julio.

7.1.7. Hito 7: 1 Mayo a 1 Julio 2016

Se prosigue con el desarrollo de la memoria y se decide optar por una nueva biblioteca gráfica (Highcharts) para la visualización e interacción de los eventos generados y almacenados en el sistema. Se obtiene una versión definitiva de la aplicación web a falta de la resolución de conflictos en las peticiones ajax de los eventos de la interfaz.

7.1.8. Hito 8: 1 Julio a 19 Septiembre 2016

Refinamiento del modelo de BD para la realización de test de integridad sobre los mismos. Se refactoriza y comenta todos los métodos que tiene la aplicación para facilitar su comprensión al desarrollador futuro. Se finaliza la memoria para la revisión por parte del director del proyecto y su posterior impresión. Se prepara la presentación para la defensa ante tribunal por parte del alumno.

7.2. Diagrama de Gantt

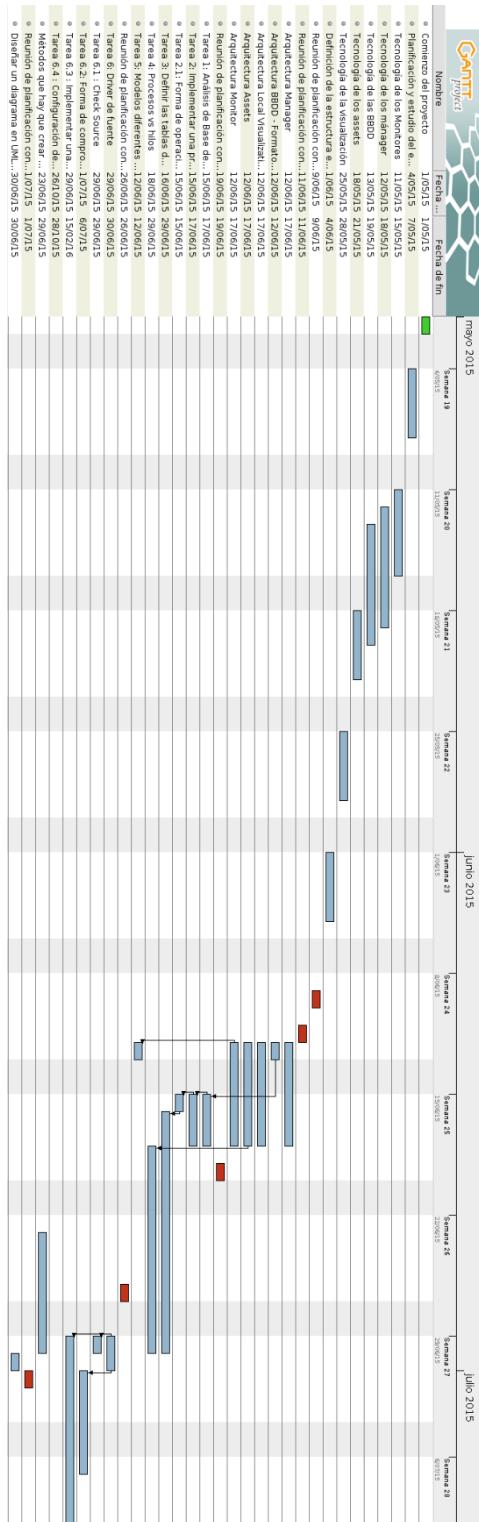


Figura 7.2: Hito 1 - 7.1.1

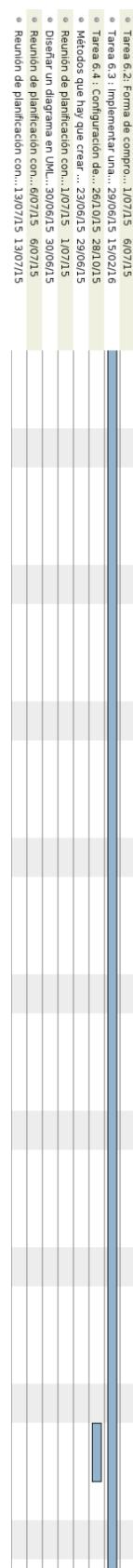


Figura 7.3: Hito 2 - 7.1.2

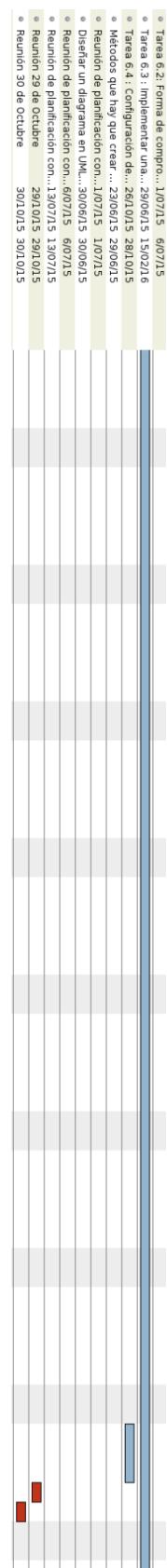


Figura 7.4: Hito 3 - 7.1.3

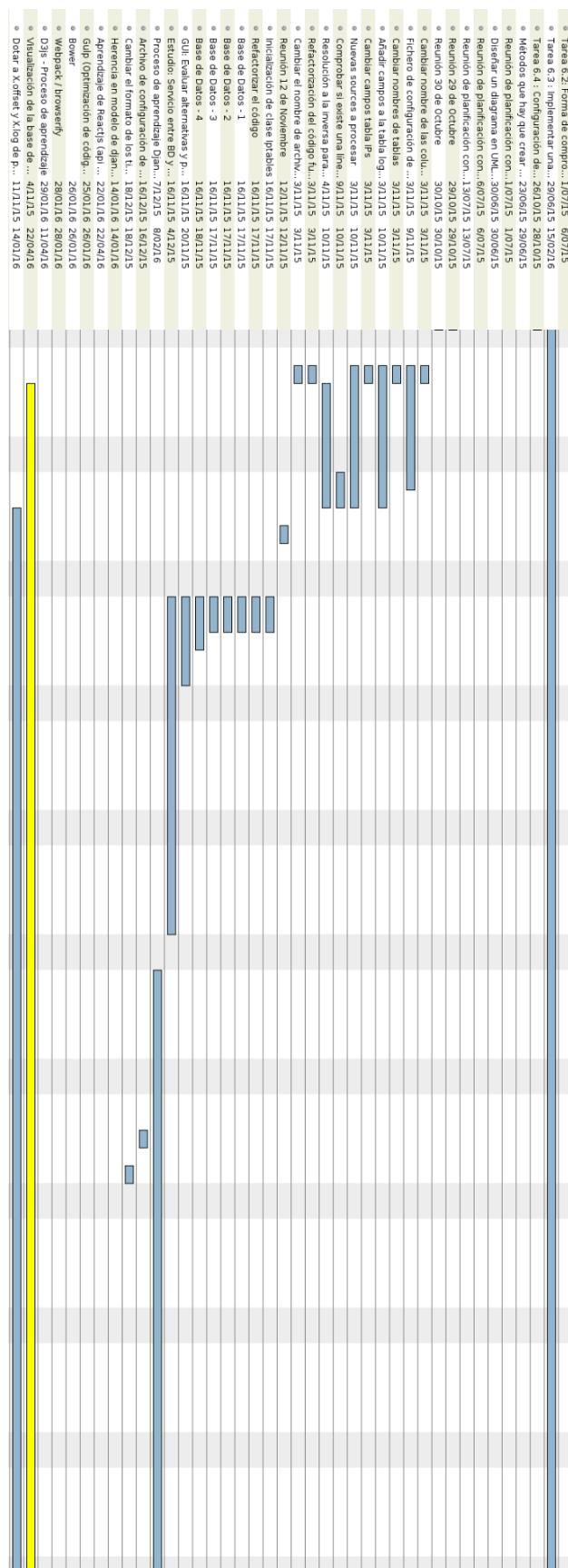


Figura 7.5: Hito 4 - 7.1.4

7.2. DIAGRAMA DE GANTT

CAPÍTULO 7. PLANIFICACIÓN TEMPORAL

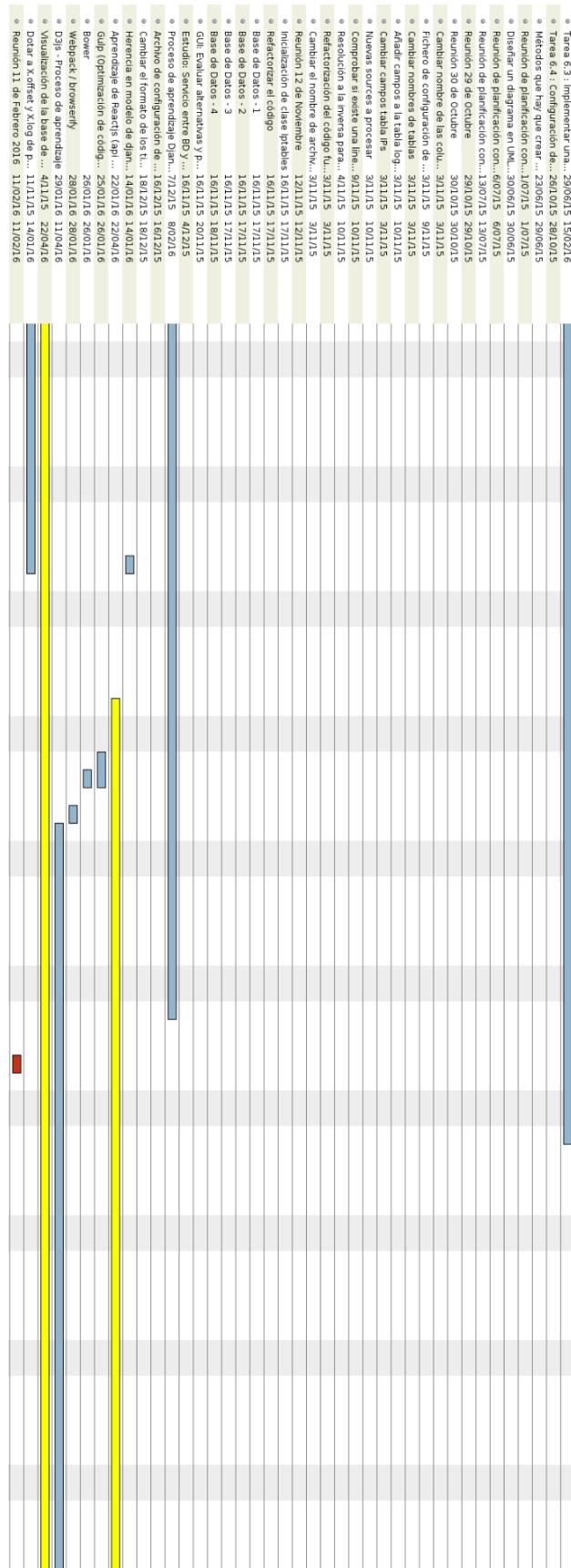


Figura 7.6: Hito 5 - 7.1.5

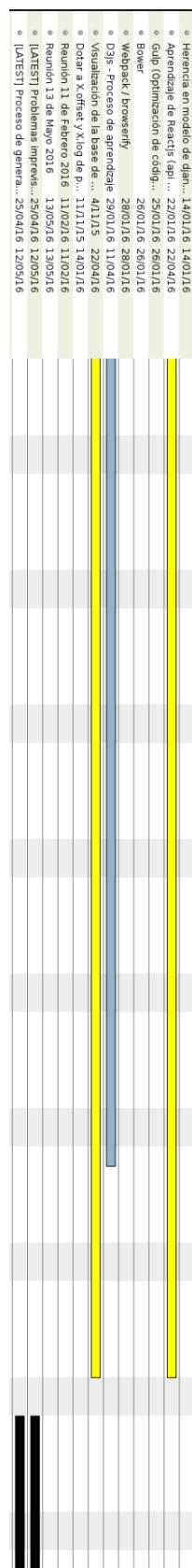


Figura 7.7: Hito 6 - 7.1.6

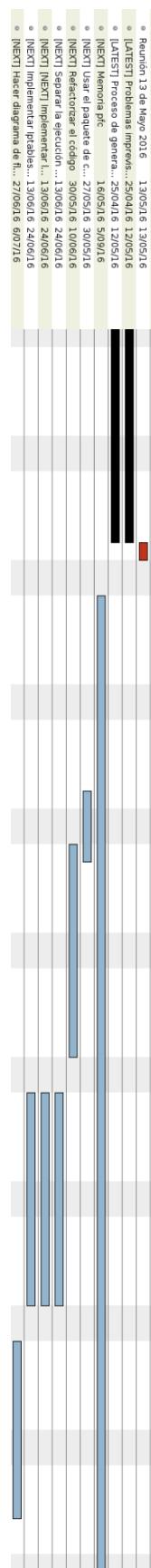


Figura 7.8: Hito 7 - 7.1.7

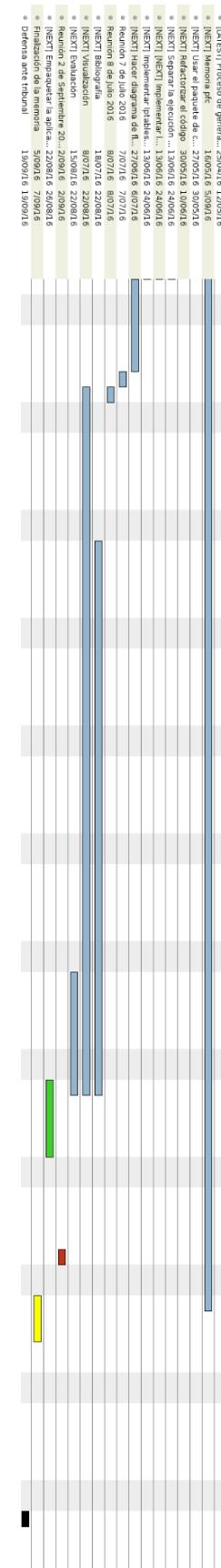


Figura 7.9: Hito 8 - 7.1.8

Capítulo 8

Conclusiones

Durante la realización de éste proyecto, se han conseguido los siguientes resultados:

- Comprender el funcionamiento de los servicios del sistema: rsyslog, syslog, logrotate, nginx.
- Comprender el funcionamiento del firewall del kernel, Iptables.
- Desarrollar una solución software usando el lenguaje de programación Python.
- Desarrollar una solución software, para sistemas web, usando el framework Django.
- Manipulación de bases de datos relacionales usando un modelo orientado a objetos (ORM) proporcionado por el framework de desarrollo.
- Diseñar el prototipo base para los sensores de recopilación de información de seguridad.
- Diseñar una interfaz web donde el usuario pueda visualizar la información que el sensor ha analizado.
- Despliegue de la aplicación en un servidor web en la nube para realizar demostraciones de la herramienta (Digital Ocean).
- Documentar todo el proceso de realización del proyecto.

CAPÍTULO 8. CONCLUSIONES

A partir de los resultados obtenidos durante la realización de éste proyecto se han extraído las siguientes conclusiones:

- La aplicación *Sensor para recopilación y visualización de información de seguridad en nodos de una red* es una herramienta que permite el análisis de información de dispositivos de seguridad para una máquina en una red.
- El código fuente implementado, responde a los objetivos que debe cumplir el desarrollo del proyecto.
- Con el diseño de la interfaz web se ha conseguido que un usuario sin conocimientos sobre el sistema, sea capaz de entender lo que se está registrando en él y que tipo de información de seguridad puede disponer a la hora de realizar un estudio de la misma.

Capítulo 9

Apéndice 1: Instalación

9.1. Instalación de Django y VirtualEnv

Primero tenemos que instalarnos un entorno virtual de desarrollo para que nuestra aplicación no modifique nuestros paths internos de Python, o si bien queremos que todo lo que nos installemos sea de uso general no tendríamos que hacer este paso. Para ello vamos a la página oficial del proyecto [virtualenv](#) y seguimos los puntos de instalación que nos indican.

Una vez configurado e instalado el paquete VirtualEnv en nuestra máquina, pasamos a utilizarlo dentro de nuestro proyecto base. Para ello simplemente, una vez clonado el mismo, vamos a la ruta “trunk/version-1-0/webapp/” y ejecutamos lo siguiente:

```
1 $ virtualenv .
2 $ . bin/activate
```

Figura 9.1: Configuración de nuestro entorno virtual de Python

Una vez dispuesto nuestro entorno virtual vamos a instalar las dependencias necesarias para el funcionamiento de la aplicación. [Para este paso es necesario tener instalado el paquete [pip](#), que es el gestor de paquetes del lenguaje Python]

```
1 $ pip install -U pip
2 $ pip install -r requirements.txt
```

Figura 9.2: Instalación de las dependencias del proyecto

Ahora es el momento de configurar nuestro proyecto Django para su ejecución:

```

1 $ cd secproject
2 $ ./manage.py makemigrations
3 $ ./manage.py migrate
4 $ ./manage.py createsuperuser #Esto nos crea el superuser de
    administracion

```

Figura 9.3: Configuración de la base de datos y creación del super usuario

Si queremos especificar un usuario distinto a uno general para diferenciar entre entornos de desarrollo y producción, tendremos que acceder a la dirección “<http://127.0.0.1:8000/admin>”, ingresar con el super usuario y definir nuevos usuarios para nuestra aplicación. Para nuestro propósito se ha definido un usuario llamado “pfc” para la parte de desarrollo de la aplicación. Una vez finalizada, se podrá establecer para un usuario normal o para otro específico.

9.2. Rsyslog

Ya tenemos configurado nuestro entorno de desarrollo de Django, ahora tenemos que configurar los servicios internos de la máquina para que corren la información generada por iptables en nuestro caso. Así pues, vamos a configurar rsyslog para que redirija los eventos de iptables a “/var/log/iptables.log”.

Incluimos las siguientes líneas en el archivo “/etc/rsyslog.conf”:

```

1 # IPTABLES
2
3 :msg,contains,"IPTMSG= " -/var/log/iptables.log
4 :msg,regex,"^ [ 0-9].[0-9]*] IPTMSG= " -/var/log/iptables.log
5 :msg,contains,"IPTMSG= " ~

```

Figura 9.4: Configuración para filtrado de eventos de Iptables

Damos la tupla de permisos 0644 a la creación de archivos:

```

1 $ FileCreateMode 0644

```

Figura 9.5: Permisos a la creación de archivos

Para obtener timestamps más precisos tenemos que comentar la siguiente línea dentro del archivo de configuración de rsyslog:

```

1 #$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat

```

Figura 9.6: Timestamp más preciso

9.3. LogRotate

Ahora tenemos que configurar el servicio LogRotate para que una vez finalizado el día, los archivos de logs antiguos, los comprima y los almacene en “/var/log/”. Para ello vamos a la ruta “/etc/logrotate.d/” y creamos el archivo “iptables” con el siguiente contenido:

```

1   /var/log/iptables.log
2   {
3     rotate 7
4     daily
5     missingok
6     notifempty
7     delaycompress
8     compress
9     postrotate
10    invoke-rc.d rsyslog restart > /dev/null
11    endscript
12 }
```

Figura 9.7: Configuración de LogRotate

9.4. /var/log/iptables.log

Tenemos que crear un archivo llamado “iptables.log” en “/var/log” con las siguientes características:

```

1 -rw-r--r-- 1 root adm 15604061 oct 26 20:28 /var/log/iptables.log
2 $ chmod 644
3 $ chown root:adm
```

Figura 9.8: Creación del archivo base de iptables.log

9.5. Offset paquete PygTail

El paquete PygTail usa para la lectura de logs dentro de nuestro sistema un archivo “.offset” del que consultará información relacionada del archivo del cual queremos obtener el texto correspondiente. Como por defecto, éste no puede ejecutarse con privilegios de super usuario dentro de la ruta “/var/log” o hacemos que nuestra aplicación corra directamente sobre super usuario (opción desaconsejada) o creamos el siguiente archivo para cada log que queramos procesar mediante PygTail.

```

1 -rw-r--rw- 1 root root      13 may  9 20:24
2   /var/log/iptables.log.offset
3 $ chmod 646
4 $ chown root:root
```

Figura 9.9: Creación del archivo offset de PygTail

9.6. Rsyslog.d

Tenemos que decirle al demonio de Rsyslog que todo lo que contenga el mensaje “IPTMSG” (que será nuestro mensaje prefijo para cada paquete obtenido mediante Iptables) lo mande a “/var/log/iptables.log”. Para ello vamos a “/etc/rsyslog.d/iptables.conf” e introducimos lo siguiente:

```

1 # into separate file and stop their further processing
2 if ($syslogfacility-text == 'kern') and \\
3   ($msg contains 'IPTMSG=' and $msg contains 'IN=') \\
4 then   -/var/log/iptables.log
5   & ~

```

Figura 9.10: Creación del archivo de configuración iptables para el demonio Rsyslog

9.7. Iptables

Los pasos anteriores es para la recolección de eventos generados por Iptables dentro de nuestra máquina. Obviamente habrá que definir unas reglas de filtrado en Iptables cuyo campo de mensaje contenga la siguiente clave/prefijo “IPTMSG= ” (incluir un espacio al final del mensaje). Aquí un ejemplo de las reglas que se han usado para la generación de eventos Iptables:

```

1 # Generated by iptables-save v1.4.21 on Mon Jan 25 20:37:18 2016
2 *filter
3 :INPUT ACCEPT [0:0]
4 :FORWARD ACCEPT [0:0]
5 :OUTPUT ACCEPT [0:0]
6 -A INPUT -d 127.0.0.1/32 -p icmp -m icmp --icmp-type 8 -m state
    --state NEW,RELATED,ESTABLISHED -j LOG --log-prefix
    "IPTMSG=Connection ICMP "
7 -A INPUT -d 127.0.0.1/32 -p icmp -m icmp --icmp-type 8 -m state
    --state NEW,RELATED,ESTABLISHED -j DROP
8 -A INPUT -p tcp -m tcp --dport 22 -j LOG --log-prefix
    "IPTMSG=Connection SSH "
9 -A INPUT -p tcp -m tcp --dport 22 -j DROP
10 COMMIT

```

Figura 9.11: Reglas de Iptables usadas para el proyecto

9.8. Web Server

Ahora ya sólo nos queda configurar nuestro servidor web, que en este caso será Nginx. **Importante:** Previamente no debe haberse instalado una versión de servidor web Apache, sino habrá que desinstalar todo y aún así dará muchos problemas. Por lo que es altamente recomendable que la instalación este limpia del paquete apache en cualquiera de sus versiones.

9.8.1. Nginx

Instalación de Nginx:

```
1 $ sudo apt-get install nginx
```

Figura 9.12: Instalación del paquete Nginx

Configuración de Nginx:

- Vamos a la carpeta “/etc/nginx/sites-available/” y creamos nuestro archivo de configuración “myproject.conf” con el siguiente contenido:

```
1      server {
2
3          root /var/www/html;
4
5          # Tipos de archivos index de nuestro sistema
6
7          index index.html index.htm index.nginx-debian.html;
8
9          # Nombre del servidor en local
10
11         server_name localhost;
12
13         location /static/ {
14             alias <ruta-descarga-proyecto>/securityproject/trunk/ \ \
15                 version-1-0/webapp/secproject/secapp/static/;
16             expires 30d;
17         }
18
19         location / {
20             proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
21             proxy_set_header Host $http_host;
22             proxy_redirect off;
23             proxy_pass http://127.0.0.1:8000;
24             proxy_pass_header Server;
25             proxy_set_header X-Real-IP $remote_addr;
26             proxy_connect_timeout 10;
27             proxy_read_timeout 10;
28
29
30     }
31 }
```

Figura 9.13: Configuración del servidor web en Nginx

- Una vez hemos escrito el archivo de configuración hacemos un enlace simbólico del mismo a otra carpeta de nginx, en este caso a “sites-enabled/”

```
1 $ sudo ln -s /etc/nginx/sites-available/myproject.conf  
      /etc/nginx/sites-enabled/
```

Figura 9.14: Enlace simbólico a nuestra configuración previa

- Para comprobar que los archivos de configuración no tienen errores, ejecutamos el siguiente comando y si es éxito, reiniciamos el servicio:

```
1 $ sudo nginx -t  
2 $ sudo service nginx restart
```

Figura 9.15: Comprobación de sintaxis de Nginx

- Ahora nos vamos al proyecto Django y lanzamos la instancia:

```
1 $ ./manage.py runserver
```

Figura 9.16: Ejecución del servidor de Django

Si la configuración se ha realizado correctamente, los contenidos estáticos de la web se verán en el navegador y no tendremos que entrar por el puerto 8000 sino por la dirección de loopback directamente: <http://127.0.0.1/secapp> ó <http://127.0.0.1/admin>

Si hubiése dudas sobre la instalación dirigirse a los contenidos de instalación actualizados en la web principal dónde se encuentra alojado el proyecto en Github:

<https://github.com/MGautier/security-sensor>

Capítulo 10

Apéndice 2: Tests

En éste apéndice se tratarán en profundidad el resto de test realizados a las clases del modelo relacional de la base de datos del sistema.

10.1. Test: Ips

Clase del modelo relacional ORM - Ips. Es la clase que almacena todas las ips que se han ido obteniendo resultado del procesamiento de log de la fuente de seguridad.

Con éste método se crean las instancias temporales en la base de datos **test** a la hora de la ejecución de la misma para comprobar su integridad. Para este caso creamos dos instancias de la clase Ips, que se usarán para poder obtener la información en los test siguientes.

```
1 class IpsTestCase(TestCase):
2     def setUp(self):
3         Ips.objects.create(Ip="127.0.0.2", Hostname="localhost",
4                             Tag="localhost")
5         Ips.objects.create(Ip="127.0.0.3", Hostname="localhost",
6                             Tag="localhost")
```

10.1.1. Método: test_ips

Descripción: Comprobación de que las ips asignadas coinciden con su asociado.

Pasos:

- Se consultan sobre las dos instancias generadas anteriormente, en el método setUp, si existen algún campo (**Ip**) cuyo valor coincida con el valor de: “127.0.0.[2-3]”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de Ips como la consulta de la referencia interna asociada dentro de Ips.

```
1 def test_ips(self):
2     """
3     Comprobacion de que las ips asignadas coinciden
4     Returns:
```

```

5      """
6
7      loopback_1 = Ips.objects.get(Ip="127.0.0.2")
8      loopback_2 = Ips.objects.get(Ip="127.0.0.3")
9      self.assertEqual(loopback_1.get_ip(), "127.0.0.2")
10     self.assertEqual(loopback_2.get_ip(), "127.0.0.3")

```

10.1.2. Método: test_ips_hostname

Descripción: Comprobación de que coinciden las ips asignadas con su hostname.

Pasos:

- Se consultan sobre las dos instancias generadas anteriormente, en el método setUp, si existen algún campo (Ip) cuyo valor coincide con el valor de: “127.0.0.[2-3]”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de Ips como la consulta de la referencia interna asociada dentro de Ips.

```

1 def test_ips_hostname(self):
2     """
3     Comprobacion de que coinciden las ips con su hostname
4     Returns:
5
6     """
7     loopback_1 = Ips.objects.get(Ip="127.0.0.2")
8     loopback_2 = Ips.objects.get(Ip="127.0.0.3")
9     self.assertEqual(loopback_1.get_hostname(), "localhost")
10    self.assertEqual(loopback_2.get_hostname(), "localhost")

```

10.1.3. Método: test_ips_tag

Descripción: Comprobación de que los tags asociados a las ips coinciden.

Pasos:

- Se consultan sobre las dos instancias generadas anteriormente, en el método setUp, si existen algún campo (Ip) cuyo valor coincide con el valor de: “127.0.0.[2-3]”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de Ips como la consulta de la referencia interna asociada dentro de Ips.

```

1 def test_ips_tag(self):
2     """
3     Comprobacion de que los tags asociados a las ips coinciden
4     Returns:
5
6     """
7     loopback_1 = Ips.objects.get(Ip="127.0.0.2")
8     loopback_2 = Ips.objects.get(Ip="127.0.0.3")
9     self.assertEqual(loopback_1.get_tag(), "localhost")
10    self.assertEqual(loopback_2.get_tag(), "localhost")

```

10.2. Test: Ports

Clase del modelo relacional ORM - Ports. Es la clase que almacena todos los puertos que se han ido obteniendo resultado del procesamiento de log de la fuente de seguridad.

Con éste método se crean las instancias temporales en la base de datos **test** a la hora de la ejecución de la misma para comprobar su integridad. Para este caso creamos una instancia de la clase Ports, que se usará para poder obtener la información en los test siguientes.

```

1 class PortsTestCase(TestCase):
2
3     def setUp(self):
4         Ports.objects.create(Tag="ftp")

```

10.2.1. Método: test_ports_tag

Descripción: Comprobación de que el tag asociado al puerto coincide.

Pasos:

- Se consulta sobre la instancia generada anteriormente, en el método setUp, si existen algún campo (**Tag**) cuyo valor coincida con el valor de: "ftp".
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de Ports como la consulta de la referencia interna asociada dentro de Ports.

```

1 def test_ports_tag(self):
2     """
3     Comprobacion de que el tag asociado al puerto coincide
4     Returns:
5     """
6     port = Ports.objects.get(Tag="ftp")
7     self.assertEqual(port.get_tag(), "ftp")

```

10.3. Test: Tcp

Clase del modelo relacional ORM - Tcp (hereda de Ports). Es la clase que almacena todos los puertos asociados a conexión TCP que se han ido obteniendo resultado del procesamiento de log de la fuente de seguridad.

Con éste método se crean las instancias temporales en la base de datos **test** a la hora de la ejecución de la misma para comprobar su integridad. Para este caso creamos una instancia de la clase Tcp y Ports, que se usarán para poder obtener la información en los test siguientes.

```

1 class TcpTestCase(TestCase):
2
3     def setUp(self):

```

```

4     port = Ports.objects.create(Tag="ssh")
5     Tcp.objects.create(id=port, Service="ssh",
6                         Description="Conexion ssh")

```

10.3.1. Método: test_tcp_service

Descripción: Comprobación de que el servicio asociado al protocolo coincide.

Pasos:

- Se consulta sobre la instancia de Ports, generada anteriormente en el método setUp, si existen algún campo (**Tag**) cuyo valor coincide con el valor de: “ssh”.
- Se consulta sobre la instancia Tcp creada anteriormente, si existe una coincidencia para el campo identificador **id** con una referencia de una instancia similar, en este caso: **port**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **tcp** coincide con el valor: “ssh”.

```

1 def test_tcp_service(self):
2     """
3         Comprobacion de que el servicio asociado al protocolo coincide
4         Returns:
5     """
6     port = Ports.objects.get(Tag="ssh")
7     tcp = Tcp.objects.get(id=port)
8     self.assertEqual(tcp.get_service(), "ssh")

```

10.3.2. Método: test_tcp_description

Descripción: Comprobación de que la descripción asociada al protocolo coincide.

Pasos:

- Se consulta sobre la instancia de Ports, generada anteriormente en el método setUp, si existen algún campo (**Tag**) cuyo valor coincide con el valor de: “ssh”.
- Se consulta sobre la instancia Tcp creada anteriormente, si existe una coincidencia para el campo identificador **id** con una referencia de una instancia similar, en este caso: **port**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **tcp** coincide con el valor: “Conexion ssh”.

```

1 def test_tcp_description(self):
2     """
3         Comprobacion de que la descripcion asociada al protocolo coincide
4         Returns:
5     """
6     port = Ports.objects.get(Tag="ssh")
7     tcp = Tcp.objects.get(id=port)
8     self.assertEqual(tcp.get_description(), "Conexion ssh")

```

10.3.3. Método: test_tcp_id

Descripción: Comprobación de que el puerto (objeto heredado) coincide con el asociado al protocolo.

Pasos:

- Se consulta sobre la instancia de Ports, generada anteriormente en el método setUp, si existen algún campo (**Tag**) cuyo valor coincida con el valor de: “ssh”.
- Se consulta sobre la instancia Tcp creada anteriormente, si existe una coincidencia para el campo identificador **id** con una referencia de una instancia similar, en este caso: **port**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la referencia de la instancia de **tcp** (identificador de la clase padre **Ports**) coincide con la referencia de la instancia temporal **port**.

```

1 def test_tcp_id(self):
2     """
3         Comprobacion de que el puerto (objeto heredado) coincide con el
4             asociado al Protocolos
5     Returns:
6     """
7     port = Ports.objects.get(Tag="ssh")
8     tcp = Tcp.objects.get(id=port)
9     self.assertEqual(tcp.get_id(), port)

```

10.4. Test: Udp

Clase del modelo relacional ORM - Udp (hereda de Ports). Es la clase que almacena todos los puertos asociados a conexión UDP que se han ido obteniendo resultado del procesamiento de log de la fuente de seguridad.

Con éste método se crean las instancias temporales en la base de datos **test** a la hora de la ejecución de la misma para comprobar su integridad. Para este caso creamos unas instancias de las clase Udp y Ports, que se usarán para poder obtener la información en los test siguientes.

```

1 class UdpTestCase(TestCase):
2
3     def setUp(self):
4         port = Ports.objects.create(Tag="ssh")
5         Udp.objects.create(id=port, Service="ssh",
6                           Description="Conexion ssh")

```

10.4.1. Método: test_udp_service

Descripción: Comprobación de que el servicio asociado al protocolo coincide.

Pasos:

- Se consulta sobre la instancia de Ports, generada anteriormente en el método setUp, si existen algún campo (**Tag**) cuyo valor coincida con el valor de: “ssh”.

- Se consulta sobre la instancia Udp creada anteriormente, si existe una coincidencia para el campo identificador **id** con una referencia de una instancia similar, en este caso: **port**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **udp** coincide con el valor: “ssh”.

```

1 def test_udp_service(self):
2     """
3         Comprobacion de que el servicio asociado al protocolo coincide
4         Returns:
5
6     """
7     port = Ports.objects.get(Tag="ssh")
8     udp = Udp.objects.get(id=port)
9     self.assertEqual(udp.get_service(), "ssh")

```

10.4.2. Método: `test_udp_description`

Descripción: Comprobación de que la descripción asociada al protocolo coincide.

Pasos:

- Se consulta sobre la instancia de Ports, generada anteriormente en el método setUp, si existen algún campo (**Tag**) cuyo valor coincide con el valor de: “ssh”.
- Se consulta sobre la instancia Udp creada anteriormente, si existe una coincidencia para el campo identificador **id** con una referencia de una instancia similar, en este caso: **port**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **udp** coincide con el valor: “Conexion ssh”.

```

1 def test_udp_description(self):
2     """
3         Comprobacion de que la descripcion asociada al protocolo coincide
4         Returns:
5
6     """
7     port = Ports.objects.get(Tag="ssh")
8     udp = Udp.objects.get(id=port)
9     self.assertEqual(udp.get_description(), "Conexion ssh")

```

10.4.3. Método: `test_udp_id`

Descripción: Comprobación de que el puerto (objeto heredado) coincide con el asociado al protocolo.

Pasos:

- Se consulta sobre la instancia de Ports, generada anteriormente en el método setUp, si existen algún campo (**Tag**) cuyo valor coincide con el valor de: “ssh”.
- Se consulta sobre la instancia Udp creada anteriormente, si existe una coincidencia para el campo identificador **id** con una referencia de una instancia similar, en este caso: **port**.

- Usando el método interno de la clase TestCase, `assertEqual`, se comprueba que la referencia de la instancia de `udp` (identificador de la clase padre `Ports`) coincide con la referencia de la instancia temporal `port`.

```

1 def test_udp_id(self):
2     """
3         Comprobacion de que el puerto (objeto heredado) coincide con el
4             asociado al Protocolos
5         Returns:
6             """
7     port = Ports.objects.get(Tag="ssh")
8     udp = Udp.objects.get(id=port)
9     self.assertEqual(udp.get_id(), port)

```

10.5. Test: Tags

Clase del modelo relacional ORM - Tags. Es la clase que almacena todas las etiquetas que pueden contener los paquetes capturados y obtenidos de los logs de la fuente de seguridad a monitorizar.

Con éste método se crean las instancias temporales en la base de datos `test` a la hora de la ejecución de la misma para comprobar su integridad. Para este caso creamos una instancia de la clase Tags, que se usará para poder obtener la información en los test siguientes.

```

1 class TagsTestCase(TestCase):
2
3     def setUp(self):
4         Tags.objects.create(Description="Urgent Pointer", Tag="URGP")

```

10.5.1. Método: test_tags_description

Descripción: Comprobación de que la descripción de la etiqueta coincide con su asociada.
Pasos:

- Se consulta sobre la instancia de Tags, generada anteriormente en el método `setUp`, si existen algún campo (**Description**) cuyo valor coincide con el valor de: “Urgent Pointer”.
- Usando el método interno de la clase TestCase, `assertEqual`, se comprueba que la información almacenada en la instancia temporal `tags` coincide con el valor: “Urgent Pointer”.

```

1 def test_tags_description(self):
2     """
3         Comprobacion de que la descripcion de la etiqueta coincide con su
4             asociada
5         Returns:
6             """
7     tags = Tags.objects.get(Description="Urgent Pointer")
8     self.assertEqual(tags.get_description(), "Urgent Pointer")

```

10.5.2. Método: test_tags_tag

Descripción: Comprobación de que la etiqueta (keyword) coincide con su asociada.

Pasos:

- Se consulta sobre la instancia de Tags, generada anteriormente en el método setUp, si existen algún campo (**Tag**) cuyo valor coincida con el valor de: “URGP”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **tags** coincide con el valor: “URGP”.

```

1 def test_tags_tag(self):
2     """
3         Comprobacion de que la etiqueta (keyword) coincide con su asociada
4         Returns:
5     """
6     tags = Tags.objects.get(Tag="URGP")
7     self.assertEqual(tags.get_tag(), "URGP")
8 
```

10.6. Test: Macs

Clase del modelo relacional ORM - Macs. Es la clase que almacena todos los valores asociados a la direcciones MAC de los logs procesados y obtenidos de la fuente de seguridad a monitorizar.

Con éste método se crean las instancias temporales en la base de datos **test** a la hora de la ejecución de la misma para comprobar su integridad. Para este caso creamos una instancia de la clase Macs, que se usará para poder obtener la información en los test siguientes.

```

1 class MacsTestCase(TestCase):
2
3     def setUp(self):
4         Macs.objects.create(
5             MAC="00:00:00:00:00:00:00:00:00:00:08:00",
6             TAG="Mac local") 
```

10.6.1. Método: test_macs_mac

Descripción: Comprobación de que la dirección mac coincide con su asociada.

Pasos:

- Se consulta sobre la instancia de Macs, generada anteriormente en el método setUp, si existen algún campo (**MAC**) cuyo valor coincida con el valor de: “0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 8 : 0 0”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **macs** coincide con el valor: “0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 8 : 0 0”.

```

1 def test_macs_mac(self):
2     """
3         Comprobacion de que la direccion mac coincide con su asociada
4     Returns:
5
6     """
7     macs =
8         Macs.objects.get(MAC="00:00:00:00:00:00:00:00:00:00:00:08:00")
9     self.assertEqual(macs.get_mac(),
10                     "00:00:00:00:00:00:00:00:00:00:00:08:00")

```

10.6.2. Método: test_macs_tag

Descripción: Comprobación de que la etiqueta que identifica a la dirección mac coincide con su asociada.

Pasos:

- Se consulta sobre la instancia de Macs, generada anteriormente en el método setUp, si existen algún campo (TAG) cuyo valor coincide con el valor de: “Mac local”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **macs** coincide con el valor: “Mac local”.

```

1 def test_macs_tag(self):
2     """
3         Comprobacion de que la etiqueta que identifica a la direccion mac
4             coincide con su asociada
5     Returns:
6
7     """
8     macs = Macs.objects.get(TAG="Mac local")
9     self.assertEqual(macs.get_tag(), "Mac local")

```

10.7. Test: LogSources

Clase del modelo relacional ORM - LogSources. Es la clase que almacena toda la información de la fuente de seguridad a monitorizar. Estos son la descripción de los campos que se registran.

- Description: Descripción de la fuente de seguridad.
- Type: Tipo de la fuente de seguridad.
- Model: Modelo o versión de la fuente de seguridad.
- Active: Si se encuentra activa dentro del sistema o no (1 ó 0).
- Software_Class: Clase a la que pertenece la fuente (Firewall, IDS, etc).
- Path: Comando que se usará para la ejecución de la fuente dentro del sistema.

Con éste método se crean las instancias temporales en la base de datos **test** a la hora de la ejecución de la misma para comprobar su integridad. Para este caso creamos una instancia de la clase LogSources, que se usará para poder obtener la información en los test siguientes.

```

1 class LogSourcesTestCase(TestCase):
2
3     def setUp(self):
4         LogSources.objects.create(
5             Description="Firewall of gnu/linux kernel",
6             Type="Iptables",
7             Model="iptables v1.4.21",
8             Active=1,
9             Software_Class="Firewall",
10            Path="iptables",
11        )

```

10.7.1. Método: test_logsources_description

Descripción: Comprobación de que la descripción de la fuente de seguridad coincide con su asociada.

Pasos:

- Se consulta sobre la instancia de LogSources, generada anteriormente en el método `setUp`, si existen algún campo (**Description**) cuyo valor coincida con el valor de: “Firewall of gnu/linux kernel”.
- Usando el método interno de la clase TestCase, `assertEqual`, se comprueba que la información almacenada en la instancia temporal **log_source** coincide con el valor: “Firewall of gnu/linux kernel”.

```

1 def test_logsources_description(self):
2     """
3         Comprobacion de que la descripcion de la fuente de seguridad
4             coincide con su asociada
5     Returns:
6     """
7     log_source = LogSources.objects.get(Description="Firewall of
8             gnu/linux kernel")
9     self.assertEqual(log_source.get_description(), "Firewall of
10            gnu/linux kernel")

```

10.7.2. Método: test_logsources_type

Descripción: Comprobación de que el tipo de la fuente de seguridad coincide con su asociado.

Pasos:

- Se consulta sobre la instancia de LogSources, generada anteriormente en el método `setUp`, si existen algún campo (**Type**) cuyo valor coincida con el valor de: “Iptables”.
- Usando el método interno de la clase TestCase, `assertEqual`, se comprueba que la información almacenada en la instancia temporal **log_source** coincide con el valor: “Iptables”.

```

1 def test_logsources_type(self):
2     """
3         Comprobacion de que el tipo de la fuente de seguridad coincide con
4             su asociado
5         Returns:
6             """
7     log_source = LogSources.objects.get(Type="Iptables")
8     self.assertEqual(log_source.get_type(), "Iptables")

```

10.7.3. Método: test_logsources_model

Descripción: Comprobación de que el modelo de la fuente de seguridad coincide con su asociado.

Pasos:

- Se consulta sobre la instancia de LogSources, generada anteriormente en el método setUp, si existen algún campo (**Model**) cuyo valor coincide con el valor de: “iptables v1.4.21”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **log_source** coincide con el valor: “iptables v1.4.21”.

```

1 def test_logsources_model(self):
2     """
3         Comprobacion de que el modelo de la fuente de seguridad coincide
4             con su asociado
5         Returns:
6             """
7     log_source = LogSources.objects.get(Model="iptables v1.4.21")
8     self.assertEqual(log_source.get_model(), "iptables v1.4.21")

```

10.7.4. Método: test_logsources_active

Descripción: Comprobación de que la fuente de seguridad se encuentra activa una vez instanciada.

Pasos:

- Se consulta sobre la instancia de LogSources, generada anteriormente en el método setUp, si existen algún campo (**Active**) cuyo valor coincide con el valor de: 1.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **log_source** coincide con el valor: 1.

```

1 def test_logsources_active(self):
2     """
3         Comprobacion de que la fuente de seguridad se encuentra activa una
4             vez instanciada
5         Returns:
6             """
7
8

```

```

5
6     """
7     log_source = LogSources.objects.get(Active=1)
8     self.assertEqual(log_source.get_active(), 1)

```

10.7.5. Método: test_logsources_software_class

Descripción: Comprobación de que la clase de software de la fuente de seguridad coincide con su asociada.

Pasos:

- Se consulta sobre la instancia de LogSources, generada anteriormente en el método setUp, si existen algún campo (**Software_Class**) cuyo valor coincide con el valor de: “Firewall”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **log_source** coincide con el valor: “Firewall”.

```

1 def test_logsources_software_class(self):
2     """
3         Comprobacion de que la clase de software de la fuente de seguridad
4             coincide con su asociada
5             Returns:
6
7     log_source = LogSources.objects.get(Software_Class="Firewall")
8     self.assertEqual(log_source.get_software_class(), "Firewall")

```

10.7.6. Método: test_logsources_path

Descripción: Comprobación de que el comando o path de ejecución de la fuente de seguridad coincide con su asociado.

Pasos:

- Se consulta sobre la instancia de LogSources, generada anteriormente en el método setUp, si existen algún campo (**Path**) cuyo valor coincide con el valor de: “iptables”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **log_source** coincide con el valor: “iptables”.

```

1 def test_logsources_path(self):
2     """
3         Comprobacion de que el comando o path de ejecucion de la fuente
4             coincide con su asociado
5             Returns:
6
7     log_source = LogSources.objects.get(Path="iptables")
8     self.assertEqual(log_source.get_path(), "iptables")

```

10.8. Test: Historic

Clase del modelo relacional ORM - Historic. Es la clase que almacena toda la información histórica de eventos en un determinado espacio de tiempo de la fuente de seguridad a monitorizar.

Con éste método se crean las instancias temporales en la base de datos **test** a la hora de la ejecución de la misma para comprobar su integridad. Para este caso creamos unas instancias de las clases LogSources e Historic, que se usarán para poder obtener la información en los test siguientes.

```

1 class TestCase(HistoricTestCase):
2
3     # Atributos internos de la clase
4
5     timestamp = timezone.now()
6
7     def setUp(self):
8         log_sources = LogSources.objects.create(
9             Description="Firewall of gnu/linux kernel",
10            Type="Iptables",
11            Model="iptables v1.4.21",
12            Active=1,
13            Software_Class="Firewall",
14            Path="iptables",
15        )
16        Historic.objects.create(
17            ID_Source=log_sources,
18            Timestamp=self.timestamp,
19            Events=1,
20        )

```

10.8.1. Método: `test_historic_source`

Descripción: Comprobación de que la fuente de seguridad a la que pertenece es igual a la asociada.

Pasos:

- Se consulta sobre la instancia de LogSources, generada anteriormente en el método `setUp`, si existen algún campo (**Type**) cuyo valor coincida con el valor de: “Iptables”.
- Se consulta sobre la instancia Historic creada anteriormente, si existe una coincidencia para el campo identificador **ID_Source** con una referencia de una instancia similar, en este caso: **log_sources**.
- Usando el método interno de la clase TestCase, `assertEqual`, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de LogSources como la consulta de la referencia interna asociada dentro de Historic.

```

1 def test_historic_source(self):
2     """
3         Comprobacion de que la fuente de seguridad a la que pertenece es
4             igual a la asociada

```

```

4     Returns:
5
6     """
7     log_sources = LogSources.objects.get(Type="Iptables")
8     historic = Historic.objects.get(ID_Source=log_sources)
9     self.assertEqual(historic.get_source(), log_sources)

```

10.8.2. Método: test_historic_timestamp

Descripción: Comprobación de que el timestamp del histórico coincide con su asociado.

Pasos:

- Se consulta sobre la instancia de Historic, generada anteriormente en el método setUp, si existen algún campo (**Timestamp**) cuyo valor coincide con el valor de la variable **self.timestamp** perteneciente a los atributos internos de la clase de test.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **Historic** coincide con el valor de la variable interna de la clase: **self.timestamp**.

```

1 def test_historic_timestamp(self):
2     """
3         Comprobacion de que el timestamp del historico coincide con su
4             asociado
5     Returns:
6
7     """
8     historic = Historic.objects.get(Timestamp=self.timestamp)
9     self.assertEqual(historic.get_timestamp(), self.timestamp)

```

10.8.3. Método: test_historic_events

Descripción: Comprobación de que el número de eventos del histórico coincide con su asociado.

Pasos:

- Se consulta sobre la instancia de Historic, generada anteriormente en el método setUp, si existen algún campo (**Events**) cuyo valor coincide con el valor: 1.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **Historic** coincide con el valor: 1.

```

1 def test_historic_events(self):
2     """
3         Comprobacion de que el numero de eventos del historico coincide
4             con su asociado
5     Returns:
6
7     """
8     historic = Historic.objects.get(Events=1)
9     self.assertEqual(historic.get_events(), 1)

```

10.9. Test: Events

Clase del modelo relacional ORM - Events. Es la clase que almacena toda la información de eventos recogidos por la fuente de seguridad que se han procesado y obtenidos de los logs.

Con éste método se crean las instancias temporales en la base de datos **test** a la hora de la ejecución de la misma para comprobar su integridad. Para este caso creamos unas instancias de las clases LogSources y Events, que se usarán para poder obtener la información en los test siguientes.

```

1 class EventsTestCase(TestCase):
2
3     # Atributos internos de la clase
4
5     timestamp = timezone.now()
6     timestamp_insertion = timezone.now()
7
8     def setUp(self):
9         log_sources = LogSources.objects.create(
10             Description="Firewall of gnu/linux kernel",
11             Type="Iptables",
12             Model="iptables v1.4.21",
13             Active=1,
14             Software_Class="Firewall",
15             Path="iptables",
16         )
17         Events.objects.create(
18             Timestamp=self.timestamp,
19             Timestamp_Insertion=self.timestamp_insertion,
20             ID_Source=log_sources,
21             Comment="Iptables Events",
22         )

```

10.9.1. Método: `test_events_timestamp`

Descripción: Comprobación de que el timestamp del evento corresponde al del asociado.

Pasos:

- Se consulta sobre la instancia de Events, generada anteriormente en el método setUp, si existen algún campo (**Timestamp**) cuyo valor coincide con el valor de la variable **self.timestamp** perteneciente a los atributos internos de la clase de test.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **Events** coincide con el valor de la variable interna de la clase: **self.timestamp**.

```

1 def test_events_timestamp(self):
2     """
3         Comprobacion de que el timestamp del evento corresponde al del
4             asociado
5         Returns:

```

```

6     """
7     events = Events.objects.get(Timestamp=self.timestamp)
8     self.assertEqual(events.get_timestamp(), self.timestamp)

```

10.9.2. Método: test_events_timestamp_insertion

Descripción: Comprobación de que el timestamp de inserción del evento corresponde con el asociado.

Pasos:

- Se consulta sobre la instancia de Events, generada anteriormente en el método setUp, si existen algún campo (**Timestamp**) cuyo valor coincide con el valor de la variable **self.timestamp_insertion** perteneciente a los atributos internos de la clase de test.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **Events** coincide con el valor de la variable interna de la clase: **self.timestamp_insertion**.

```

1 def test_events_timestamp_insertion(self):
2     """
3         Comprobacion de que el timestamp de insercion del evento
4             corresponde con el asociado
5         Returns:
6
7     events =
8         Events.objects.get(Timestamp_Insertion=self.timestamp_insertion)
9         self.assertEqual(events.get_timestamp_insertion(),
10                         self.timestamp_insertion)

```

10.9.3. Método: test_events_source

Descripción: Comprobación de que la fuente de seguridad, a la que pertenece, es igual a la asociada.

Pasos:

- Se consulta sobre la instancia de LogSources, generada anteriormente en el método setUp, si existen algún campo (**Type**) cuyo valor coincide con el valor de: “Iptables”.
- Se consulta sobre la instancia Events creada anteriormente, si existe una coincidencia para el campo identificador **ID_Source** con una referencia de una instancia similar, en este caso: **log_sources**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de LogSources como la consulta de la referencia interna asociada dentro de Events.

```

1 def test_events_source(self):
2     """
3         Comprobacion de que la fuente de seguridad, a la que pertenece, es
4             igual a la asociada

```

```

4     Returns:
5
6     """
7     log_sources = LogSources.objects.get(Type="Iptables")
8     events = Events.objects.get(ID_Source=log_sources)
9     self.assertEqual(events.get_source(), log_sources)

```

10.9.4. Método: test_events_comment

Descripción: Comprobación de que el comentario asociado al evento pertenece al asociado.

Pasos:

- Se consulta sobre la instancia de Events, generada anteriormente en el método setUp, si existen algún campo (**Comment**) cuyo valor coincide con el valor de: “Iptables Events”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **events** coincide con el valor: “Iptables Events”.

```

1 def test_events_comment(self):
2     """
3         Comprobacion de que el comentario asociado al evento pertenece al
4             asociado
5     Returns:
6
7     """
8     events = Events.objects.get(Comment="Iptables Events")
9     self.assertEqual(events.get_comment(), "Iptables Events")

```

10.10. Test: Visualizations

Clase del modelo relacional ORM - Visualizations. Es la clase que almacena toda la información de las visualizaciones de la parte web que pertenecen a los eventos recogidos por la fuente de seguridad que se han procesado y obtenidos de los logs.

Con éste método se crean las instancias temporales en la base de datos **test** a la hora de la ejecución de la misma para comprobar su integridad. Para este caso creamos unas instancias de las clases LogSources y Visualizations, que se usarán para poder obtener la información en los test siguientes.

```

1 class VisualizationsTestCase(TestCase):
2
3     def setUp(self):
4         log_sources = LogSources.objects.create(
5             Description="Firewall of gnu/linux kernel",
6             Type="Iptables",
7             Model="iptables v1.4.21",
8             Active=1,
9             Software_Class="Firewall",
10            Path="iptables",
11        )

```

```

12     Visualizations.objects.create(
13         Week_Month=1,
14         Week_Day=2,
15         Name_Day="Wednesday",
16         Date=date(2016, 8, 10),
17         Hour=18,
18         ID_Source=log_sources,
19         Process_Events=5
20     )

```

10.10.1. Método: `test_visualizations_week_month`

Descripción: Comprobación de que la semana del mes pertenece a la asociada.

Pasos:

- Se consulta sobre la instancia de `Visualizations`, generada anteriormente en el método `setUp`, si existen algún campo (`Week_Month`) cuyo valor coincida con el valor de: 1.
- Usando el método interno de la clase `TestCase`, `assertEqual`, se comprueba que la información almacenada en la instancia temporal `visualizations` coincide con el valor: 1.

```

1 def test_visualizations_week_month(self):
2     """
3     Comprobacion de que la semana del mes pertenece a la asociada
4     Returns:
5     """
6     visualizations = Visualizations.objects.get(Week_Month=1)
7     self.assertEqual(visualizations.get_week_month(), 1)

```

10.10.2. Método: `test_visualizations_week_day`

Descripción: Comprobación de que el día de la semana pertenece a la asociada.

Pasos:

- Se consulta sobre la instancia de `Visualizations`, generada anteriormente en el método `setUp`, si existen algún campo (`Week_Day`) cuyo valor coincida con el valor de: 2.
- Usando el método interno de la clase `TestCase`, `assertEqual`, se comprueba que la información almacenada en la instancia temporal `visualizations` coincide con el valor: 2.

```

1 def test_visualizations_week_day(self):
2     """
3     Comprobacion de que el dia de la semana pertenece a la asociada
4     Returns:
5     """
6     visualizations = Visualizations.objects.get(Week_Day=2)
7     self.assertEqual(visualizations.get_week_day(), 2)

```

10.10.3. Método: test_visualizations_name_day

Descripción: Comprobación de que el nombre del día procesado coincide con su asociado.

Pasos:

- Se consulta sobre la instancia de Visualizations, generada anteriormente en el método setUp, si existen algún campo (**Name_Day**) cuyo valor coincida con el valor de: “Wednesday”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **visualizations** coincide con el valor: “Wednesday”.

```

1 def test_visualizations_name_day(self):
2     """
3         Comprobacion de que el nombre del dia procesado coincide con su
4             asociado
5         Returns:
6     """
7     visualizations = Visualizations.objects.get(Name_Day="Wednesday")
8     self.assertEqual(visualizations.get_name_day(), "Wednesday")

```

10.10.4. Método: test_visualizations_date

Descripción: Comprobación de que la fecha registrada en el sistema coincide con la asociada.

Pasos:

- Se consulta sobre la instancia de Visualizations, generada anteriormente en el método setUp, si existen algún campo (**Date**) cuyo valor coincida con el valor de una instancia de la clase Date (python): date(2016, 8, 10).
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **visualizations** coincide con el valor de una instancia de la clase Date (python): date(2016, 8, 10).

```

1 def test_visualizations_date(self):
2     """
3         Comprobacion de que la fecha registrada en el sistema coincide con
4             la asociada
5         Returns:
6     """
7     visualizations = Visualizations.objects.get(Date=date(2016, 8, 10))
8     self.assertEqual(visualizations.get_date(), date(2016, 8, 10))

```

10.10.5. Método: test_visualizations_hour

Descripción: Comprobación de que la hora registrada en el sistema para la fecha procesada, coincide con la asociada.

Pasos:

- Se consulta sobre la instancia de Visualizations, generada anteriormente en el método setUp, si existen algún campo (**Hour**) cuyo valor coincide con el valor de: 18.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **visualizations** coincide con el valor: 18.

```

1 def test_visualizations_hour(self):
2     """
3         Comprobacion de que la hora registrada en el sistema para la fecha
4             procesada, coincide con la asociada
5         Returns:
6     """
7     visualizations = Visualizations.objects.get(Hour=18)
8     self.assertEqual(visualizations.get_hour(), 18)

```

10.10.6. Método: **test_visualizations_source**

Descripción: Comprobación de que la fuente de seguridad a la que pertenece, es igual a la asociada.

Pasos:

- Se consulta sobre la instancia de LogSources, generada anteriormente en el método setUp, si existen algún campo (**Type**) cuyo valor coincide con el valor de: "Iptables".
- Se consulta sobre la instancia Visualizations creada anteriormente, si existe una coincidencia para el campo identificador **ID_Source** con una referencia de una instancia similar, en este caso: **log_sources**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de LogSources como la consulta de la referencia interna asociada dentro de Visualizations.

```

1 def test_visualizations_source(self):
2     """
3         Comprobacion de que la fuente de seguridad a la que pertenece, es
4             igual a la asociada
5         Returns:
6     """
7     log_sources = LogSources.objects.get(Type="Iptables")
8     visualizations = Visualizations.objects.get(ID_Source=log_sources)
9     self.assertEqual(visualizations.get_source(), log_sources)

```

10.10.7. Método: **test_visualizations_process_events**

Descripción: Comprobación de que el número de eventos registrados para la fecha coincide con el asociado.

Pasos:

- Se consulta sobre la instancia de Visualizations, generada anteriormente en el método setUp, si existen algún campo (**Process_Events**) cuyo valor coincide con el valor de: 5.

- Usando el método interno de la clase TestCase, `assertEqual`, se comprueba que la información almacenada en la instancia temporal **visualizations** coincide con el valor: 5.

```
1 def test_visualizations_process_events(self):
2     """
3         Comprobacion de que el numero de eventos registrados para la fecha
4             coincide con el asociado
5     Returns:
6     """
7     visualizations = Visualizations.objects.get(Process_Events=5)
8     self.assertEqual(visualizations.get_process_events(), 5)
```

10.11. Test: Packet Additional Information

Clase del modelo relacional ORM - Packet Additional Information. Es la clase que almacena toda la información adicional de los paquetes registrados en el sistema para una fuente de seguridad.

Con éste método se crean las instancias temporales en la base de datos **test** a la hora de la ejecución de la misma para comprobar su integridad. Para este caso creamos instancias de todas las clases que componen el modelo ORM (a excepción de Historic, Tcp y Udp), que se usarán para poder obtener la información en los test siguientes.

```
1 class PacketAdditionalInfoTestCase(TestCase):
2
3     # Atributos internos de la clase
4     timestamp = timezone.now()
5     timestamp_insertion = timezone.now()
6
7     def setUp(self):
8         ip_source = Ips.objects.create(Ip="127.0.0.2",
9             Hostname="localhost", Tag="localhost")
10        ip_dest = Ips.objects.create(Ip="127.0.0.3",
11            Hostname="localhost", Tag="localhost")
12        port_source = Ports.objects.create(Tag="ftp")
13        port_dest = Ports.objects.create(Tag="ssh")
14        mac_source = Macs.objects.create(
15            MAC="00:00:00:00:00:00:00:00:00:00:00:08:00",
16            TAG="Mac local1")
17        mac_dest = Macs.objects.create(
18            MAC="00:00:00:00:00:00:00:00:00:00:00:08:00",
19            TAG="Mac local2")
20        log_sources = LogSources.objects.create(
21            Description="Firewall of gnu/linux kernel",
22            Type="Iptables",
23            Model="iptables v1.4.21",
24            Active=1,
25            Software_Class="Firewall",
26            Path="iptables",
27        )
28        event = Events.objects.create(
29            Timestamp=self.timestamp,
```

```

28         Timestamp_Insertion=self.timestamp_insertion ,
29         ID_Source=log_sources ,
30         Comment="Iptables Events",
31     )
32     packet = PacketEventsInformation.objects.create(
33         ID_IP_Source=ip_source ,
34         ID_IP_Dest=ip_dest ,
35         ID_Source_Port=port_source ,
36         ID_Dest_Port=port_dest ,
37         Protocol="ICMP" ,
38         ID_Source_MAC=mac_source ,
39         ID_Dest_MAC=mac_dest ,
40         RAW_Info="LOG RAW INFO" ,
41         TAG="Connection ICMP" ,
42         id=event
43     )
44     tag = Tags.objects.create(Description="Packet ID", Tag="ID")
45     PacketAdditionalInfo.objects.create(
46         ID_Tag=tag ,
47         ID_Packet_Events=packet ,
48         Value="32731 DF"
49     )

```

10.11.1. Método: test_packet_additional_id_tag

Descripción: Comprobación de que el objeto tag (referencia al modelo relacional) coincide con su asociado.

Pasos:

- Se consulta sobre la instancia de Tags, generada anteriormente en el método setUp, si existen algún campo (**Tag**) cuyo valor coincide con el valor de: "ID".
- Se consulta sobre la instancia Packet AdditionalInfo creada anteriormente, si existe una coincidencia para el campo identificador **ID_Tag** con una referencia de una instancia similar, en este caso: **tag**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de Tags como la consulta de la referencia interna asociada dentro de PacketAdditionalInfo.

```

1 def test_packetadditionalinfo_id_tag(self):
2     """
3         Comprobacion de que el objeto tag (referencia al modelo
4             relacional) coincide con su asociado
5         Returns:
6         """
7     tag = Tags.objects.get(Tag="ID")
8     pai = PacketAdditionalInfo.objects.get(ID_Tag=tag)
9     self.assertEqual(pai.get_id_tag(), tag)

```

10.11.2. Método: test_packet_additional_id_packet_events

Descripción: Comprobación de que el packet events information al que pertenece (referencia al modelo relacional) coincide con su asociado.

Pasos:

- Se consulta sobre la instancia de Events, generada anteriormente en el método setUp, si existen algún campo (**Timestamp**) cuyo valor coincida con el valor de la variable **self.timestamp** perteneciente a los atributos internos de la clase de test.
- Se consulta sobre la instancia de PacketEventsInformation creada anteriormente, si existe una coincidencia para el campo identificador **id** con una referencia de una instancia similar, en este caso: **event**.
- Se consulta sobre la instancia PacketAdditionalInfo creada anteriormente, si existe una coincidencia para el campo identificador **ID_Packet_Events** con una referencia de una instancia similar, en este caso: **packet**.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba la integridad de la información que deben contener tanto la consulta a la instancia temporal de PacketEventsInformation como la consulta de la referencia interna asociada dentro de PacketAdditionalInfo.

```

1 def test_packetadditionalinfo_id_packet_events(self):
2     """
3         Comprobacion de que el packet events information al que pertenece
4             (referencia al modelo relacional) coincide con su asociado
5         Returns:
6     """
7     event = Events.objects.get(Timestamp=self.timestamp)
8     packet = PacketEventsInformation.objects.get(id=event)
9     pai = PacketAdditionalInfo.objects.get(ID_Packet_Events=packet)
10    self.assertEqual(pai.get_id_packet_events(), packet)

```

10.11.3. Método: test_packet_additional_value

Descripción: Comprobación de que el valor del paquete, correspondiente a la etiqueta, coincide con su asociado.

Pasos:

- Se consulta sobre la instancia de PacketAdditionalInfo, generada anteriormente en el método setUp, si existen algún campo (**Value**) cuyo valor coincida con el valor de: “32731 DF”.
- Usando el método interno de la clase TestCase, *assertEqual*, se comprueba que la información almacenada en la instancia temporal **pai** coincide con el valor: 5.

```

1 def test_packetadditionalinfo_value(self):
2     """
3         Comprobacion de que el valor del paquete, correspondiente a la
4             etiqueta, coincide con su asociado

```

```
4     Returns:  
5  
6     """  
7     pai = PacketAdditionalInfo.objects.get(Value="32731 DF")  
8     self.assertEqual(pai.get_value(), "32731 DF")
```

Bibliografía

- [1] Nosql vs sql: Sql is the new nonosql. Disponible en la web: <http://www.nosql-vs-sql.com/>.
- [2] App acelerator. Working with a sqlite database. Disponible en la web: <https://wiki.appcelerator.org/display/guides2/Working+with+a+SQLite+Database#WorkingwithaSQLiteDatabase-ClosingthedatabaseandResultSet>.
- [3] Shadypixel Blog. Log iptables messages to a separate file with rsyslog. Disponible en la web: <https://blog.shadypixel.com/log-iptables-messages-to-a-separate-file-with-rsyslog/>.
- [4] Takipi Blog. 5 reasons you should stop hosting your elk stack locally. Disponible en la web: <http://blog.takipi.com/hosted-elasticsearch-the-future-of-your-elk-stack/>.
- [5] Takipi Blog. Log management tools face-off: Splunk vs. logstash vs. sumo logic. Disponible en la web: <http://blog.takipi.com/log-management-tools-face-off-splunk-vs-logstash-vs-sumo-logic/>.
- [6] Xorl blog. Descripción del formato log para iptables. Disponible en la web: <https://xorl.wordpress.com/2009/02/03/logging-with-iptables-and-syslogd/>.
- [7] Python Central. Python's sqlalchemy vs other orms. Disponible en la web: <http://pythoncentral.io/sqlalchemy-vs-orms/>.
- [8] Cisco. Cisco application networking manager 5.2. Disponible en la web: http://www.cisco.com/c/en/us/products/collateral/application-networking-services/application-networking-manager/data_sheet_c78-706167.pdf.
- [9] Cyberciti. Force iptables to log messages to a different log file. Disponible en la web: <http://www.cyberciti.biz/tips/force-iptables-to-log-messages-to-a-different-log-file.html>.
- [10] Maria DB. Maria db. Disponible en la web: <https://mariadb.com/>.
- [11] debianHackers. Threads, procesos y cómo estos afectan la seguridad de un sistema. Disponible en la web: <https://debianhackers.net/threads-procesos-y-como-estos-afectan-la-seguridad-de-un-sistema-explicado-de-forma-que>.
- [12] django. Logging. Disponible en la web: <https://docs.djangoproject.com/en/1.9/topics/logging/>.
- [13] Django docs. Django db managers. Disponible en la web: <https://docs.djangoproject.com/en/dev/topics/db/managers/#manager-names>.

- [14] Django docs. Testing reusable applications. Disponible en la web: <https://docs.djangoproject.com/es/1.10/topics/testing/advanced/#testing-reusable-applications>.
- [15] Django docs. Writing tests. Disponible en la web: <https://docs.djangoproject.com/es/1.9/topics/testing/overview/>.
- [16] Elastic. Github repository: Elastic. Disponible en la web: <https://github.com/elastic>.
- [17] Firebird. Firebird. Disponible en la web: <http://www.firebirdsql.org/>.
- [18] Web Forefront. Set up logging for a django project. Disponible en la web: <https://www.webforefront.com/django/setupdjangologging.html>.
- [19] Genbeta:dev. Multiprocesamiento en python: Threads a fondo, enumeración, herencia y temporizadores. Disponible en la web: <http://www.genbetadev.com/python/multiprocesamiento-en-python-threads-a-fondo-enumeracion-herencia-y-temporizadores>.
- [20] Genbeta:dev. Nuestra primera aplicación con google app engine (python). Disponible en la web: <http://www.genbetadev.com/programacion-en-la-nube/nuestra-primer-aplicacion-con-google-app-engine-python>.
- [21] Highcharts. Highcharts web. Disponible en la web: <http://www.highcharts.com/>.
- [22] Cambridge Intelligence. The keylines toolkit. Disponible en la web: <http://cambridge-intelligence.com/keylines/>.
- [23] ipswitch. Log and event management. Disponible en la web: <https://www.ipswitch.com/solutions/log-and-event-management>.
- [24] Jooq. Jooq. Disponible en la web: <http://www.jooq.org/>.
- [25] Michal Karzynski. Setting up django with nginx, gunicorn, virtualenv, supervisor and postgresql. Disponible en la web: <http://michal.karzynski.pl/blog/2013/06/09/django-nginx-gunicorn-virtualenv-supervisor/>.
- [26] Groovy lang. Groovy: Differences with java. Disponible en la web: <http://www.groovy-lang.org/differences.html>.
- [27] lansweeper. Network management. Disponible en la web: <http://www.lansweeper.com/>.
- [28] LogRhythm. Siem. Disponible en la web: <https://logrhythm.com/es/products/siem/>.
- [29] Macaron. Macaron: Python o/r mapper. Disponible en la web: <http://nibrin.github.io/macaron/>.
- [30] Markwingerd. Setting up django with ubuntu 12.10, sqlite, gunicorn, and nginx on a digital ocean droplet. Disponible en la web: <https://markwingerd.wordpress.com/2013/09/01/setting-up-django-with-ubuntu-12-10-sqlite-gunicorn-and-nginx-on-a-digital-ocean-droplet>.
- [31] McAfee. Información de seguridad y administración de eventos. Disponible en la web: <http://www.mcafee.com/es/products/siem/index.aspx>.

- [32] MGautier. Iptables log timestamp value is from 1970. Disponible en la web: <http://stackoverflow.com/questions/32914701/iptables-log-timestamp-value-is-from-1970/32914903#32914903>.
- [33] MGautier. Proyecto fin de carrera - backlog. Resto de referencias en: <https://tree.taiga.io/project/mgautier-proyecto-fin-de-carrera/backlog>.
- [34] MGautier. Read a file with root permissions from python script. Disponible en la web: <http://stackoverflow.com/questions/34533505/read-a-file-with-root-permissions-from-python-script>.
- [35] MongoDB. Mongodb. Disponible en la web: <https://www.mongodb.com/>.
- [36] MyTardis. Django workflow. Disponible en la web: http://mytardis.readthedocs.io/en/3.5/_images/DjangoArchitecture-JeffCroft.png.
- [37] Erick Navarro. Despliegue django con nginx, supervisor y gunicorn. Disponible en la web: <https://codeandoando.com/despliegue-django-con-nginx-supervisor-y-gunicorn/>.
- [38] Neo4j. Graph visualization for neo4j. Disponible en la web: <https://neo4j.com/developer/guide-data-visualization/>.
- [39] Netfilter. Uso de iptables. Disponible en la web: <http://www.netfilter.org/documentation/HOWTO/es/packet-filtering-HOWTO-7.html>.
- [40] Digital Ocean. A comparison of nosql database management systems and models. Disponible en la web: <https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models>.
- [41] Digital Ocean. How to install and manage supervisor on ubuntu and debian vps. Disponible en la web: <https://www.digitalocean.com/community/tutorials/how-to-install-and-manage-supervisor-on-ubuntu-and-debian-vps>.
- [42] Digital Ocean. How to set up django with postgres, nginx, and gunicorn on ubuntu 14.04. Disponible en la web: <https://www.digitalocean.com/community/tutorials/how-to-set-up-django-with-postgres-nginx-and-gunicorn-on-ubuntu-14-04>.
- [43] Digital Ocean. Sqlite vs mysql vs postgresql: A comparison of relational database management systems. Disponible en la web: <https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems>.
- [44] Digital Ocean. Understanding sql and nosql databases and different database models. Disponible en la web: <https://www.digitalocean.com/community/tutorials/understanding-sql-and-nosql-databases-and-different-database-models>.
- [45] Stack overflow. django.request logger not propagated to root? Disponible en la web: <http://stackoverflow.com/questions/20282521/django-request-logger-not-propagated-to-root/22336174#22336174>.
- [46] Stack overflow. Simple log to file example for django 1.3+. Disponible en la web: <http://stackoverflow.com/questions/5739830/simple-log-to-file-example-for-django-1-3>.

- [47] Philogb. Javascript infovis toolkit. Disponible en la web: <http://philogb.github.io/jit/>.
- [48] PostgreSQL. Postgresql. Disponible en la web: <https://www.postgresql.org/>.
- [49] Python. Pep 249 – python database api specification v2.0. Disponible en la web: <https://www.python.org/dev/peps/pep-0249/>.
- [50] Python. Python - data model. Disponible en la web: <https://docs.python.org/2/reference/datamodel.html>.
- [51] Python. sqlite3 — db-api 2.0 interface for sqlite databases. Disponible en la web: <https://docs.python.org/2/library/sqlite3.html>.
- [52] Python. Web frameworks for python. Disponible en la web: <https://wiki.python.org/moin/WebFrameworks>.
- [53] Mail Python. [python-es] threads con operaciones i/o en python. Disponible en la web: <https://mail.python.org/pipermail/python-es/2010-June/027561.html>.
- [54] Qbox. Elk stack. Disponible en la web: <https://qbox.io/blog/tag/elk-stack>.
- [55] Qbox. Parsing logs using logstash. Disponible en la web: <https://qbox.io/blog/parsing-logs-using-logstash>.
- [56] redis. redis. Disponible en la web: <http://redis.io/>.
- [57] Rsyslog. omfile: File output module. Disponible en la web: <http://www.rsyslog.com/doc/v8-stable/configuration/modules/omfile.html>.
- [58] Rthalley. Dnsso toolkit for python. Disponible en la web: <https://github.com/rthalley/dnspython>.
- [59] s21sec. Lookwise. Disponible en la web: <https://www.s21sec.com/es/productos/lookwise>.
- [60] s21sec. Lookwise: ficha técnica. Disponible en la web: <https://www.s21sec.com/es/docs-a-resources/category/14-bitacora?download=176%3Aficha-bitacora>.
- [61] Tom Sawyer Software. Tom sawer software. Disponible en la web: <https://www.tomsawyer.com/>.
- [62] SpiceWorks. Spiceworks. Disponible en la web: <http://www.spiceworks.com/free-network-monitoring-management-software/>.
- [63] SQLAlchemy. Sqlalchemy. Disponible en la web: http://docs.sqlalchemy.org/en/rel_1_0/orm/tutorial.html.
- [64] SQLite. Command line shell for sqlite. Disponible en la web: <https://www.sqlite.org/cli.html>.
- [65] SQLite. SQLite encryption extension. Disponible en la web: <http://www.sqlite.org/doc/trunk/www/readme.wiki>.
- [66] Sqlite. Sqlite foreign key support. Disponible en la web: https://www.sqlite.org/foreignkeys.html#fk_schemacommands).

- [67] Unix StackExchange. How can i configure syslog.conf file, to log iptables messages in a separate file? Disponible en la web: <http://unix.stackexchange.com/questions/96484/how-can-i-configure-syslog-conf-file-to-log-iptables-messages-in-a-separate-file>.
- [68] Supervisord. How to configure python script to run as a daemon. Disponible en la web: <http://serverfault.com/questions/501071/how-to-configure-python-script-to-run-as-a-daemon>.
- [69] Supervisord. Supervisor: A process control system. Disponible en la web: <http://supervisord.org/>.
- [70] tom's IT PRO. A guide to security information and event management. Disponible en la web: http://www.tomsitpro.com/articles/siem-solutions-guide_2-864.html.
- [71] Libros web. Python - polimorfismo. Disponible en la web: https://librosweb.es/libro/algoritmos_python/capitulo_15/polimorfismo.html.
- [72] Tornado Web. Tornado web server. Disponible en la web: <http://www.tornadoweb.org/en/stable/>.
- [73] Wikipedia. List of tcp and udp port numbers. Disponible en la web: https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers.
- [74] Zetcode. Inserting, updating, and deleting data in sqlite. Disponible en la web: <http://zetcode.com/db/sqlite/datamanipulation/>.

GNU Documentation Free License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a

Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text. The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this

CAPÍTULO 10. GNU DOCUMENTATION FREE LICENSE

License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

CAPÍTULO 10. GNU DOCUMENTATION FREE LICENSE

- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

CAPÍTULO 10. GNU DOCUMENTATION FREE LICENSE

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in

an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

See <http://www.gnu.org/copyleft/>.

CAPÍTULO 10. GNU DOCUMENTATION FREE LICENSE

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with … Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

CAPÍTULO 10. GNU DOCUMENTATION FREE LICENSE

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.