Universidad Internacional de La Rioja (UNIR)

Escuela de Ingeniería

Grado en Ingeniería Informática

Herramienta online para el análisis estático de malware

Ubicación del código fuente:
https://github.com/u29165053/wasa

Trabajo Fin de Grado

presentado por: Fernández Valero, Diego

Director/a: Fernández Lanza, Santiago

Ciudad: Madrid
Fecha:
08/07/2016

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En el mundo del análisis de malware, existen dos enfoques para llevar a cabo dicho análisis, el dinámico y el estático. El primero se centra en el comportamiento del software malicioso durante su ejecución, mientras que en análisis estático se enfoca a la información que se puede obtener del fichero sin llegar a ejecutarlo.

Se ha desarrollado una aplicación web que permite realizar el análisis estático de un fichero, generalmente malware, y que muestra los resultados vía web, o mediante un informe descargable.

Palabras clave: malware, análisis estático, binario, apk, exe

ABSTRACT

In the world of malware analysis, there are two approaches to analyze: the dynamic analysis and static analysis. The first focuses on the behavior of malware during its execution, while static analysis focuses on the information that you can get the file without actually executing it.

It has developed a web application that enables static analysis of a file, usually malware, and displays the results via the web, or through a downloadable report.

Keywords: malware, static analysis, binary, exe, apk

1. INTRODUCCIÓN

En la actualidad, el malware es una amenaza cada vez más a tener en cuenta. Supone un peligro tanto para grandes organizaciones como para usuarios particulares.

Pasaron ya los tiempos en los que los ataques informáticos con malware (comúnmente conocidos como “virus”) buscaban llamar la atención, o causar algún tipo de daño en el sistema del usuario.

Hoy en día, el malware ha evolucionado tantísimo que puede llegar incluso a acabar con los sistemas de información de una compañía. Existe un tipo de amenazas, llamadas APT, que se enfocan exclusivamente en un objetivo, y pueden permanecer “dormidas” durante periodos largos de tiempo, hasta que consiguen su objetivo.

Estamos, por tanto, ante una situación en la que estas técnicas se utilizan para objetivos de espionaje industrial, o incluso gubernamental.

Debido al avanzado nivel técnico de estos malware, es necesario que un grupo de analistas cualificados estudien y analicen el comportamiento del programa, para determinar qué es lo que hace, cómo lo hace, y si es posible, para quién lo hace.

1.1 Objetivo y Alcance

Se ha desarrollado una herramienta llamada WASA (*Windows and Android Static Analyzer)*. Esta herramienta permite facilitar la tarea del analista que se encuentre ante la necesidad de analizar un código malicioso de forma estática.

Es posible que el nivel de detalle que pueda obtener la herramienta, no esté a la altura de la valoración que un analista de malware con experiencia pudiera derivar tras su estudio. Sin embargo, al tratarse de una herramienta online, permite tener una aproximación del código malicioso, simplemente subiendo la muestra a la web.

Esto permite que no sea necesario disponer de un costoso laboratorio de malware, que asegure la confidencialidad, integridad y disponibilidad del sistema.

No obstante, la aplicación sólo realiza análisis estático, por lo que sigue siendo necesario contar con un laboratorio de malware si se desea realizar un análisis dinámico del ejecutable.

1.2 Glosario

A continuación se incluye un glosario con los términos utilizados a través de este documento, y una breve descripción sobre lo que significan.

Tabla 1 - Glosario de términos

Término	Proviene de	Descripción
WASA	*Windows and Android Static Analyzer*	Se trata de la herramienta desarrollada en este trabajo.
Malware	*Malicious software*	Es un tipo de software cuya intención es causar daño, o infiltrarse en una máquina, sin consentimiento de su propietario.
APT	*Advanced Persistent Threat*	Es la ejecución de cierto malware sigiloso y sofisticado, que consigue infiltrarse durante un largo periodo de tiempo en una organización específica sin ser descubierto.
Sandbox		Es un entorno controlado en el que no se pueden producir daños, pues se aísla los datos a los que se tiene acceso. Además, se suele poder revertir a un estado original.
Backend		Parte de código de una aplicación web que se ejecuta en el servidor, sin que el usuario lo vea.
Frontend		Parte de una aplicación de web que es visible por un usuario.
DLL	*Dynamic-link library*	Son archivos del sistema operativo que contienen código ejecutable y funciones exportadas, para que puedan ser utilizadas por otros programas.
Debugging		En malware, se dice debuggear al ejecutar el programa malicioso en un depurador, que permite ver todo lo que sucede paso a paso.
Antidebugging		Técnicas utilizadas por el malware para evitar desvelar su verdadera funcionalidad cuando se está debuggeando.
PE	*Portable Executable*	Es un tipo de formato de archivos ejecutables usados en versiones de 32 y 64 bits de Microsoft Windows.

API	*Application Programming Interface*	Conjunto de procesos y funciones que ofrece unas funcionalidades para ser utilizadas por otro programa como una capa de abstracción.
IoC	*Indicator of Commitment*	Descripción de un patrón identificable en un incidente de malware. Puede ser una dirección IP o un dominio a la que se conecta, una serie de bytes que definen su firma, etc.
Modding		En términos de malware, el modding es el proceso de modificación de un malware para hacerlo indetectable.
AJAX	*Asynchronous JavaScript And XML*	Es una técnica para mantener una comunicación asíncrona con el servidor en segundo plano, de forma que no sea necesario recargar la página.
Patrón		En este trabajo se refiere a patrón de diseño, en el campo de la ingeniería de software. Un patrón de diseño es una solución a un problema dado, que debe ser eficaz y reutilizable.
MVC	*Modelo Vista Controlador*	Es un patrón de diseño que divide una aplicación web en tres capas diferentes. Una se encarga de las vistas, otra de la lógica de negocio, y otra de la comunicación entre ambas.
SGBD	*Sistema Gestor de Base de Datos*	Es el software que soporta la gestión de diferentes bases de datos. Una base de datos es accesible a través de un SGBD. Por ejemplo: MySQL, SQLServer, etc.
DAO	*Data Access Object*	Es una clase que contiene métodos para interactuar con el SGBD y proporciona el acceso a los datos de la aplicación.
SQLi	*SQL Injection*	Ataque que consiste en inyectar un payload SQL en una consulta no saneada. Con esto se consigue ejecutar código SQL arbitrario.
Hash		Es una función matemática que transforma un conjunto de bytes en una cadena de un determinado número de caracteres. Los mismos bytes siempre se obtendrá la misma cadena, y el proceso es irreversible.

Salt		En criptografía, se trata de un conjunto de bits aleatorios que se utilizan junto al valor original en la función de hash. Esto proporciona un nivel extra de seguridad.
RC4	*Rivest Cipher 4*	Es un algoritmo de cifrado simétrico de flujo
Serialización		Es el proceso de transofrmar un objeto en un medio de almacenamiento, en un conjunto de bytes, con el fin de poder transmitirlo a través de una conexión de red.

2. PROCEDIMIENTO DE ANÁLISIS DE MALWARE

A la hora de realizar un análisis de malware, hay gente que opina que con un análisis estático es suficiente, mientras que otros opinan que con un análisis dinámico es también suficiente. No obstante, la mayoría de analistas coinciden en que es necesario tanto un análisis dinámico, como un análisis estático.

Pero, para poder realizar el análisis del malware, se deben tener claro los conceptos:

- **Análisis dinámico de malware:** Se trata de un tipo de análisis que estudia el comportamiento del malware en ejecución. Normalmente corre en un entorno de sandboxing, como una máquina virtual aislada del resto de la red.
- **Análisis estático de malware:** Es una técnica que analiza el malware sin ejecutarlo. Para ello estudia su código ensamblador, sus cabeceras y las dependencias, entre otras cosas.

Más allá de estos dos modos de análisis, es necesario definir una metodología de análisis de malware, que indique los pasos a seguir en cada una de las fases. Debido a que no se ha encontrado una metodología estándar para el análisis de malware, se ha optado por desarrollar una propia.

La metodología propuesta consta de cuatro fases:

- **Recopilar información:** Esta fase cubre las actividades destinadas a obtener información del fichero, como su hash y las dependencias.
- **Análisis dinámico:** En esta fase se ejecuta el malware en un entorno controlado (sandbox) preparado para monitorizar los cambios producidos en el sistema.
- **Análisis estático:** Se analiza el código ensamblador del malware, así como cabeceras, dependencias, cadenas de texto, etc.
- **Elaborar el informe:** Esta es la última fase, en la que se plasman los resultados del análisis en un informe

Figura 1: Fases de la metodología de análisis de malware

En cada una de estas fases se realizan diferentes actividades, que se detallan a continuación:

- **Fase 1. Recopilar información**
  - ***Realizar hash al archivo**:* Con esto se identifica de manera única el archivo, aunque cambie de nombre. Se deberían realizar varios hashes, como por ejemplo MD5, SHA1 y SHA256.
  - ***Analizar cabeceras**:* Se analizan las cabeceras del fichero PE en busca de información que pueda ser de utilidad para el futuro análisis, como por ejemplo direcciones de memoria.
  - ***Listar librerías (DLL) utilizadas**:* Esto permite conocer a simple vista a que APIs del Sistema Operativo está llamando, lo que da una aproximación de la funcionalidad del fichero.
  - ***Listar las funciones externas que utiliza**:* Dentro de la librería, se listan las funciones que utiliza, con el fin de afinar aún más para conocer el código del sistema que se ejecuta con el malware.
  - ***Listar las cadenas de texto**:* Esto puede servir para identificar alguna cadena que sea relevante para el análisis, como una contraseña, o un mensaje determinado. Si se utiliza el comando “strings” se va a obtener lo que podría ser una representación ASCII de una cadena, por lo que aparecerán muchos falsos positivos.
- **Fase 2. Análisis dinámico**
  - ***Preparación del entorno de sandbox:*** A través de este entorno se podrá ejecutar malware para conocer su comportamiento, pero no se exponen otros recursos de la red, ni la máquina anfitrión. Además, se puede revertir a su estado original, para limpiar la infección.
  - ***Establecer la línea base de configuración del sistema:*** Define el estado original de la máquina en un estado en el que esté libre de toda infección, y se guarda dicho estado en un snapshot.
  - ***Preparación de Sniffer de Red:*** Se pone a escuchar un sniffer de red para capturar todo el tráfico de la máquina de laboratorio.
  - ***Preparación de un servicio Fake DNS:*** Con un servicio de falsificación de DNS se puede hacer que todas las peticiones DNS devuelvan una IP determinada (normalmente la IP local), y se registran dichas peticiones, lo que permite tener un listado de sitios a los que se conecta.
  - ***Guardar el estado del registro y el sistema de archivos:*** Se guarda el estado del registro y el sistema de ficheros, haciendo un hash a cada fichero, lo que permita comparar modificaciones tras ejecutar el malware
  - ***Ejecución del malware:*** Se ejecuta el malware para analizar su comportamiento.
  - ***Captura del tráfico de red con el malware en ejecución:*** Mientras el malware se ha ejecutado se captura el tráfico de red, y se almacena en un fichero para analizarlo posteriormente.
  - ***Comparación del estado del sistema con la línea base:*** Se vuelve a capturar el estado del registro y el sistema de ficheros, para compararlo con el estado anterior, y poder descubrir que ficheros y/o claves de registro se han modificado.

- ***Identificación de las conexiones realizadas:*** Se identifican las conexiones realizadas por el malware, ya sea por petición DNS o por acceso directo a una IP.
- ***Capturar el tráfico desde la máquina virtual a la máquina host:*** Antes de ejecutar el malware se ha puesto a escuchar el sniffer para capturar el tráfico desde la máquina virtual a la máquina anfitrión. Esto permitirá saber si el malware está intentando propagarse por la red anfitrión.

- **Fase 3. Análisis estático**
  - ***Identificar posibles ofuscaciones o cifrados del fichero***: Si el malware se esfuerza por cifrar un payload o un dato, es que es importante para él. Por este motivo, al identificar los algoritmos de cifrado que utiliza, y los datos que cifra, se tendrá una visión más detallada de las intenciones del fichero malicioso.
  - ***Determinar la existencia de técnicas antidebugging:*** Se debe tener en cuenta que un malware puede utilizar técnicas que comprueben si están siendo ejecutados en un depurador, o una máquina virtual, y de ser así modifica su comportamiento. Esto complica la tarea del análisis dinámico, por eso es importante identificar posibles técnicas antidepuración, y si se encuentra alguna, volver a ejecutar un análisis dinámico, teniendo esto en cuenta.
  - ***Determinar si el malware tiene el mismo funcionamiento en una maquina física que en una VM:*** Este punto deriva del anterior. Si el comportamiento del malware es diferente en una máquina física que en una máquina virtual, se debe documentar las diferencias entre ambos comportamientos.
  - ***Determinar si se comunica con algún recurso a través de Internet:*** Se buscan dominios, URL, o direcciones IP con las que el malware se intenta comunicar.
  - ***Determinar los ficheros de los que hace uso:*** Se buscan rutas y ficheros en el código del fichero para determinar que ficheros del sistema de archivos, o que claves de registro utiliza.
  - ***Buscar algoritmos de cifrado, y en su caso, las claves:*** Si se determina que el malware utiliza algún tipo de cifrado es necesario descubrir qué algoritmo utiliza, e intentar obtener la clave de cifrado. En malware poco sofisticado, la clave estará “hardcodeada”. Sin embargo, en malware más sofisticado la clave de cifrado puede enviarse desde un servidor, y a su vez cifrada con criptografía asimétrica.
  - ***Búsqueda de cadenas de texto:*** Puede ser interesante buscar las cadenas de texto que figuran en el binario, pues podrían indicar algún tipo de contraseñas o mensajes.
  - ***Analizar firma digital, si procede:*** Los binarios pueden ir firmados digitalmente. Algunos malware también están firmados con la clave de alguna empresa ficticia, o real, si se ha conseguido la clave privada a través de alguna infección.
- **Fase 4. Elaborar el informe**
  - ***Incluir información de contexto del ejecutable:*** En todo informe sobre el análisis de un malware, o fichero binario, debe aparecer el nombre, los diferentes hashes, el formato, el tamaño del fichero, etc.

- ***Incluir fechas de compilación y creación:*** Además, es interesante indicar las fechas de compilación y creación del fichero, si pueden obtenerse.
- ***Incluir los detalles de la firma digital del malware:*** Si el malware está firmado digitalmente, se deben incluir también los detalles de dicha firma.
- ***Incluir las librerías externas que utiliza:*** Se muestran también las librerías del sistema operativo que utiliza el fichero.
- ***Incluir las funciones externas a las que llama:*** Dentro de cada una de las librerías que utilice, usará unas determinadas funciones, que también deben reflejarse en el informe.
- ***Listar las rutinas de cifrado***: Se indican las rutinas de cifrado con el máximo nivel de detalle, incluido algoritmos y claves si es posible.
- ***Incluir las comunicaciones que realiza:*** Las comunicaciones que realiza el malware son esenciales, por eso deben ser una parte fundamental del informe.
- ***Incluir el listado de ficheros que utiliza:*** Los ficheros que utiliza también son de suma importancia en un análisis de un binario, especialmente de malware.

3. ESTADO DEL ARTE

En este capítulo se pretende analizar otras soluciones del mercado que cumplan con alguna función similar a la de la herramienta que se ha desarrollado.

Cabe mencionar que las opciones del mercado actual están bastante más avanzadas que la herramienta desarrollada, debido a la escasa disponibilidad de tiempo.

En el mundo de las aplicaciones web para análisis de malware, existen varias que son las más conocidas:

- Malwr
- Koodous

A continuación se detallan cada una de ellas:

3.1 Malwr

Malwr es una herramienta online, que utiliza Cuckoo Sandbox en el backend para analizar malware. Es una herramienta bastante completa, especialmente en el análisis dinámico, y que contiene también una parte de análisis estático.

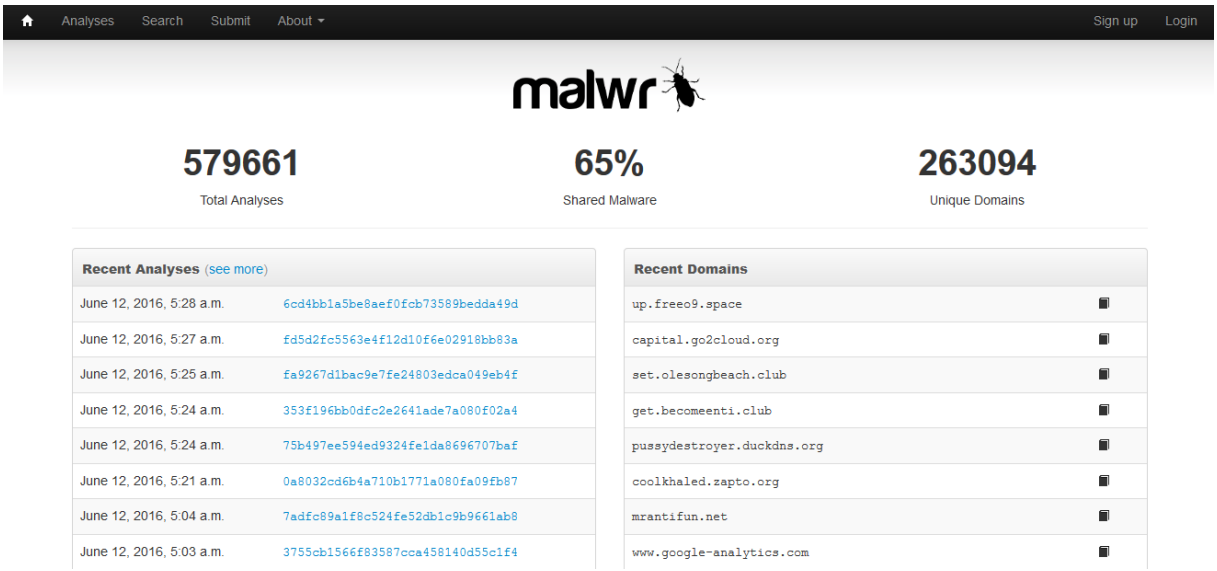

Figura 2: Malwr.com

Esta herramienta tiene un funcionamiento similar que la aplicación desarrollada, aunque más completa, ya que no sólo se centra en el análisis estático, sino que también realiza análisis dinámico del fichero binario.

Tras realizar el análisis de un fichero se redirige a la página de resultados en la que se puede ver diferente tipo de información, como información de contexto (Figura 3), firmas, screenshots y hosts con los que se comunica (Figura 4); y resultados del análisis estático del fichero binario (Figura 5).

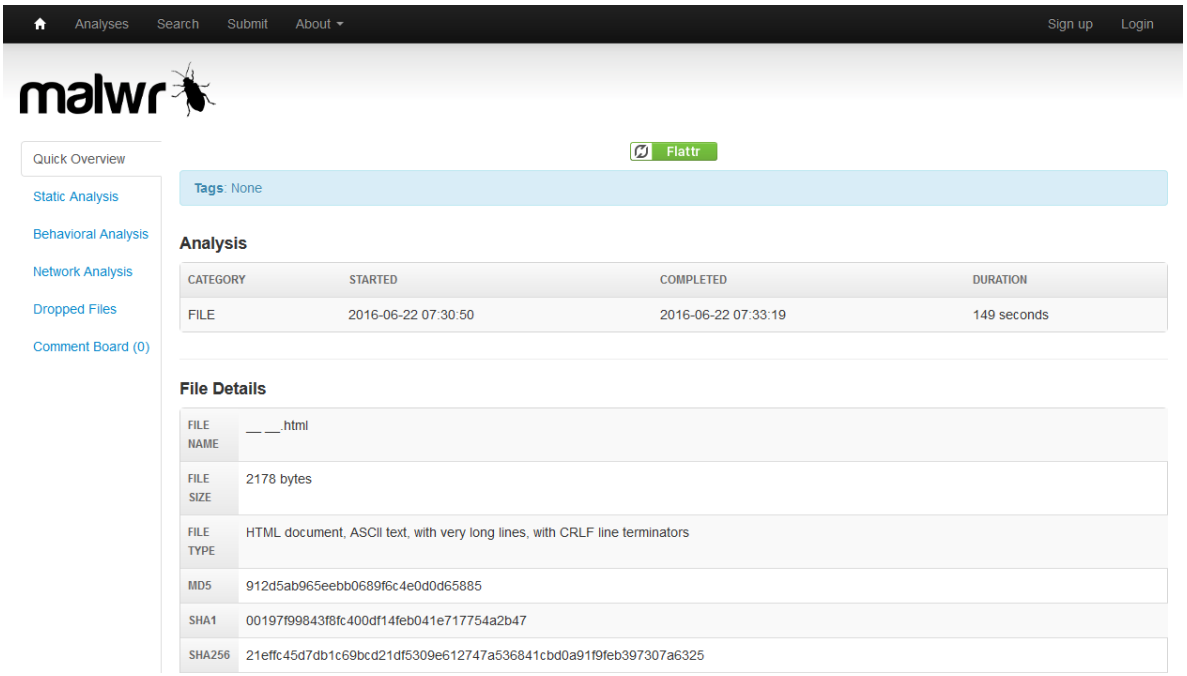

Figura 3: Información de contexto

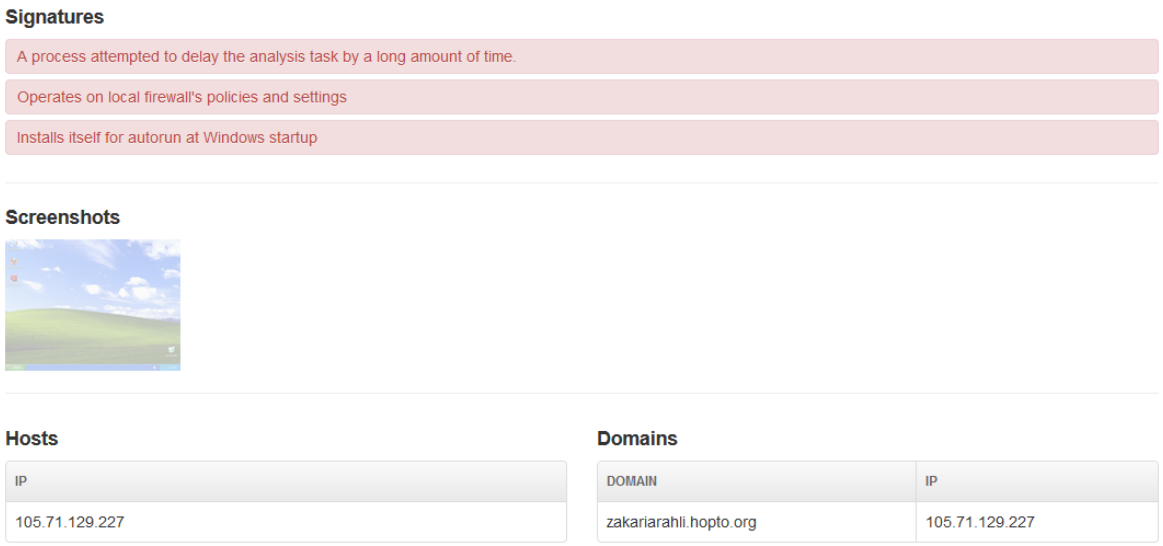

Figura 4: Screenshots y comunicaciones

Sin embargo, en la parte estática, hay una funcionalidad de *WASA* que no incluye Malwr, y es la de **desensamblado de código**.

Gracias a esta funcionalidad, se puede obtener el código ensamblador del fichero binario, lo que permitiría un análisis más exhaustivo partiendo de éste.

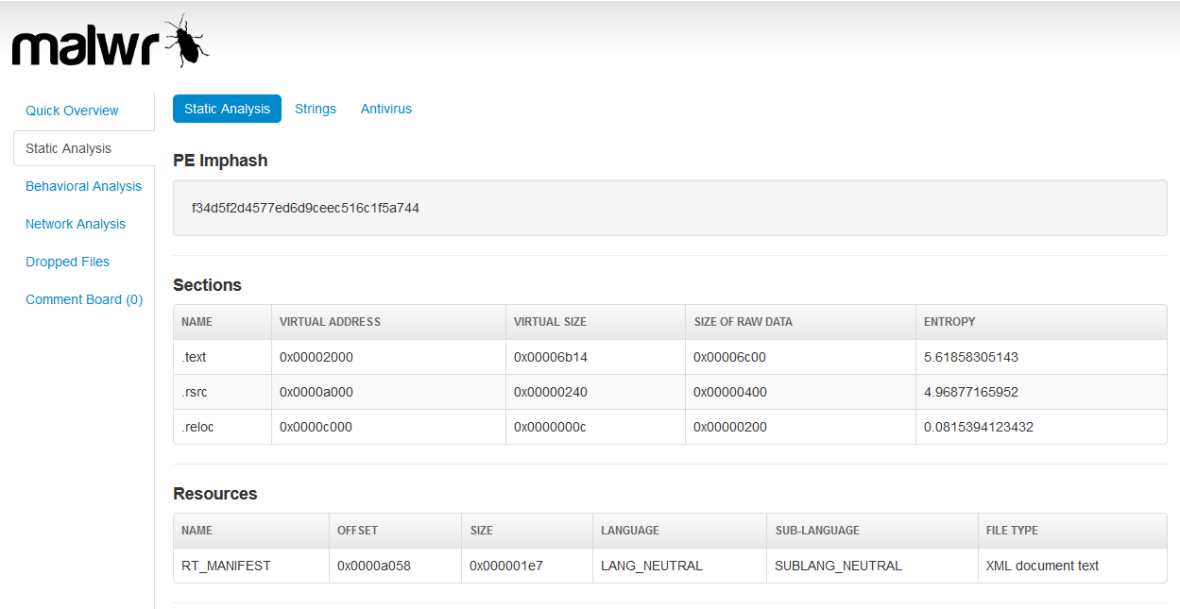

Figura 5: Análisis estático de Malwr

Como conclusión, esta aplicación es el principal competidor de WASA, la herramienta desarrollada, y que además incluye más funcionalidades. No obstante, desde el principio se supo que intentar igualar a esta herramienta en cuanto a funcionalidades y popularidad era un objetivo demasiado ambicioso para un Trabajo de Fin de Grado.

3.2 Koodous

En este caso se trata de una aplicación web para el análisis colaborativo de malware en plataformas Android. Los usuarios pueden subir ficheros apk, que la aplicación analizará. Además, existe una comunidad de usuarios que realiza análisis manuales y emiten una valoración (positiva o negativa) de una aplicación en concreto, para ayudar a determinar si se trata de una aplicación maliciosa o no.

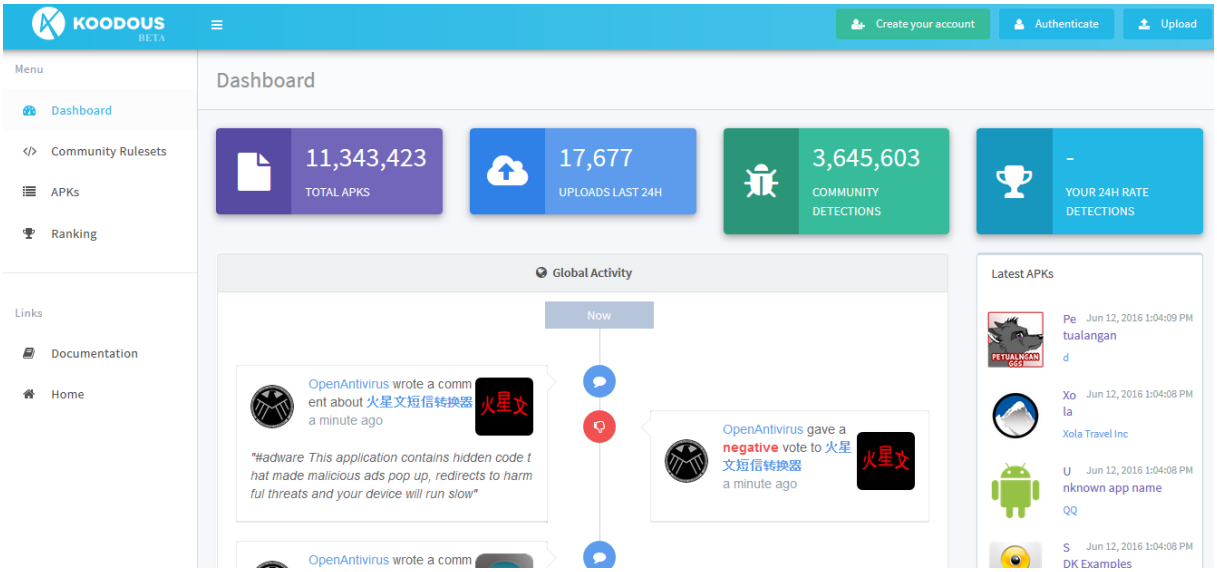

Figura 6: Koodous

Consultando la página del informe del análisis se puede obtener diferente información. Se parte de la información del fichero (Figura 7), pero también se tienen menús con la información de activities, receivers, permisos, etc (Figura 8); Strings, Hosts y URL con los que se comunica.

Además, también se incluye un apartado en el que se puede copiar todos los datos del análisis en formato JSON, lo que puede ser de utilizar para importar datos a algún otro sistema u aplicación.

Como conclusión, se trata de una herramienta muy interesante para los usuarios de Android, que permite comparar las valoraciones que le han dado otros usuarios a una aplicación para determinar si se trata de una aplicación maliciosa.

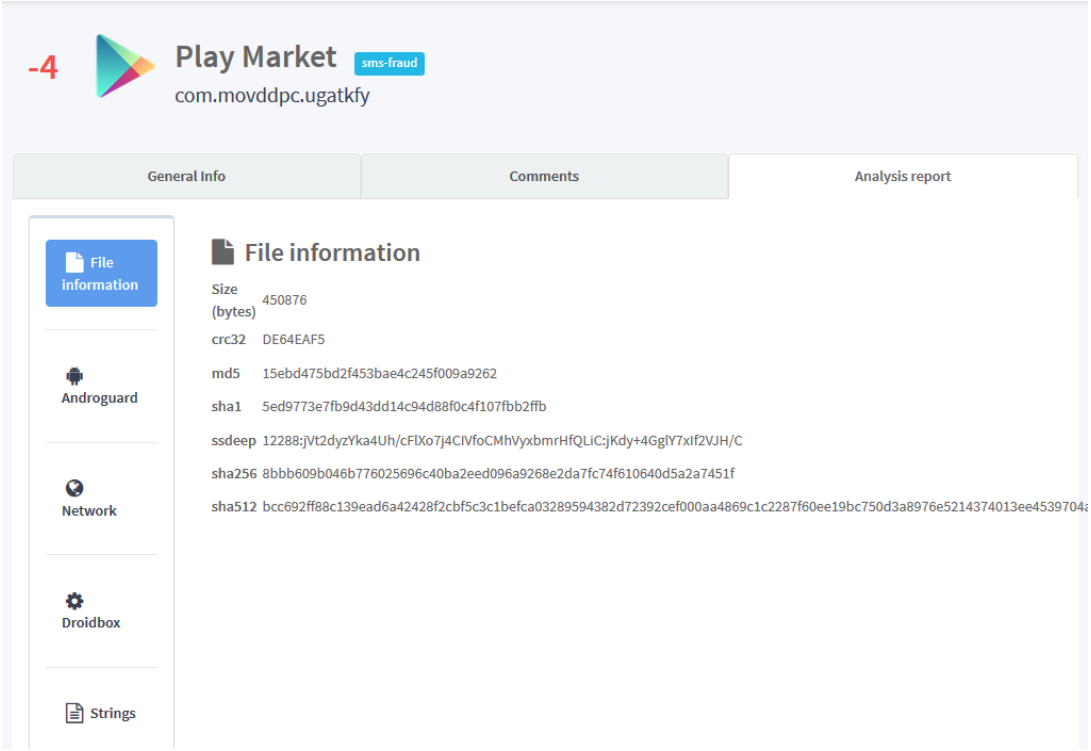

Figura 7: Información del fichero

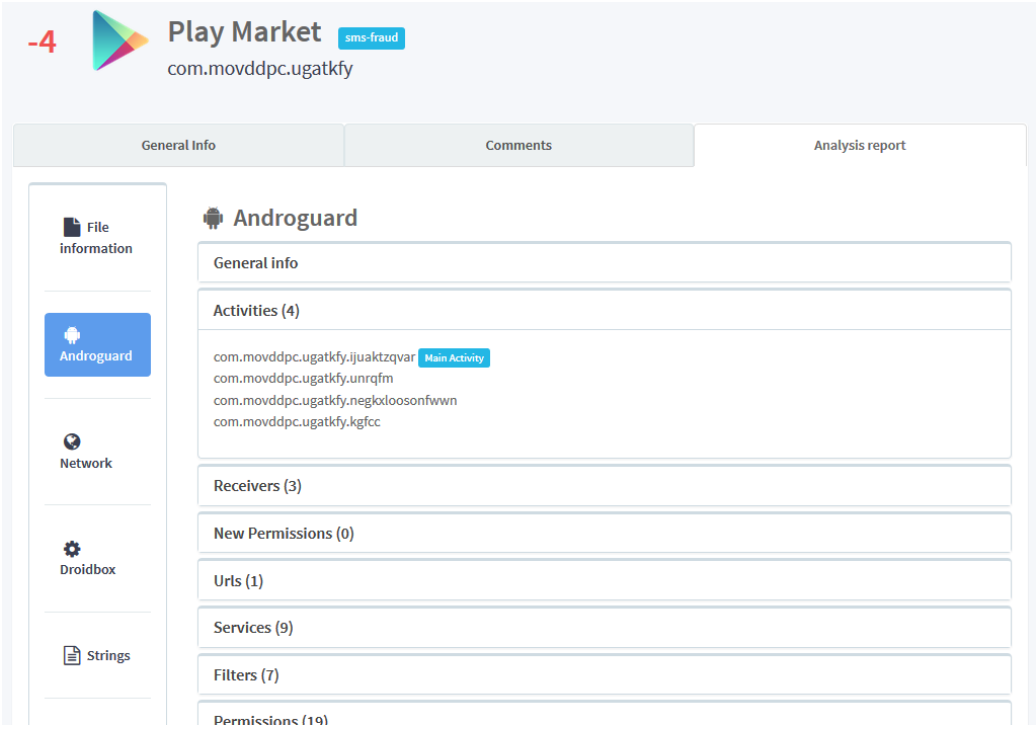

Figura 8: Androguard

4. IDENTIFICACIÓN DE REQUISITOS

La herramienta a desarrollar debe tener los siguientes requisitos:

- Requisitos funcionales:
  - Debe ser una aplicación web
  - Se debe poder analizar un fichero binario en formato PE (.exe)
  - Se debe poder analizar una aplicación de android (.apk)
  - Debe tener un formulario de registro
  - Debe tener dos modos: público y privado
  - Debe poder generar informes de los análisis
  - Se deben guardar las muestras del malware analizado
- Requisitos no funcionales
  - Las cargas de las páginas deben ser asíncronas a través de AJAX
  - Las muestras deben estar comprimidas y con contraseña
  - Se deben realizar varios hashes diferentes de los ficheros

4.1 Caso de uso

Para modelar los requisitos funcionales, se utilizan los **casos de uso**, que son descripciones de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso. Los casos de uso para esta aplicación se detallan a continuación.

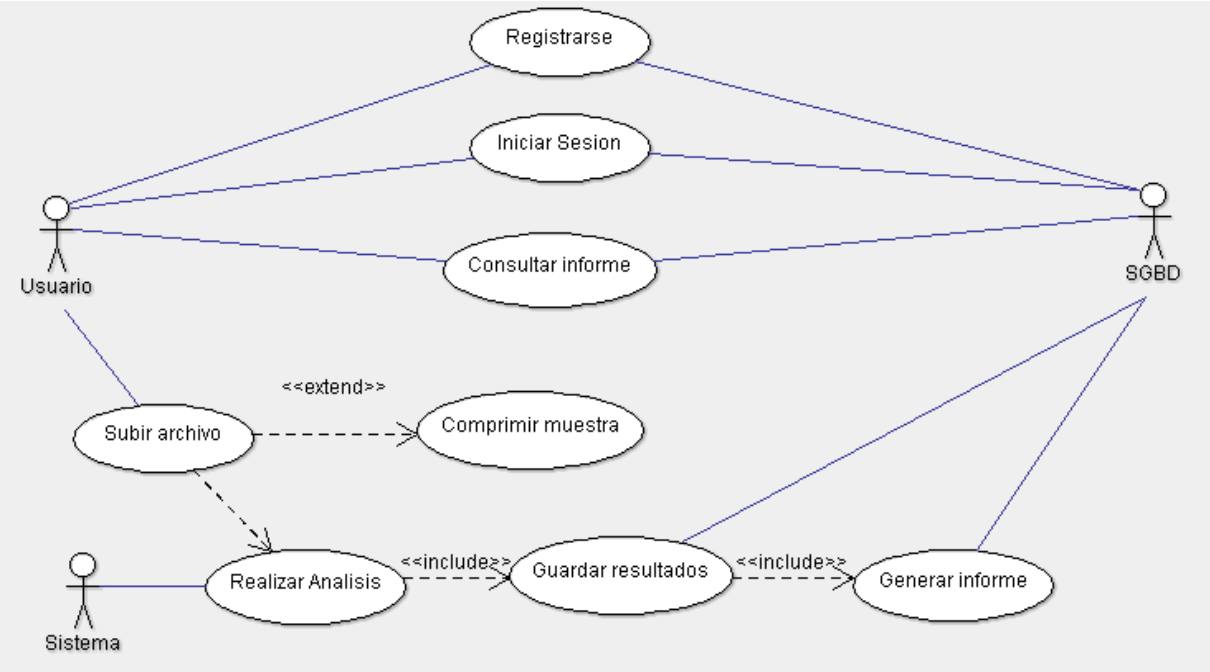

Figura 9: Casos de uso

- **Registrarse:** La aplicación debe tener un formulario de registro que permita a los usuarios registrarse. Las ventajas de registrarse con una cuenta es que permite realizar análisis en privado. Los análisis que haya realizado un usuario registrado no están disponibles al público.
- **Iniciar Sesión:** Obviamente, si un usuario puede registrarse, debe poder iniciar sesión en el sistema.
- **Consultar informe:** Cuando se ha realizado el análisis, se puede consultar los resultados online. En la página de visualización de resultados se ofrece la opción de descargar los resultados en un informe.
- **Subir archivo:** Para poder realizar el análisis de un fichero, la aplicación debe proporcionar un medio para subir los ficheros al servidor.
- **Comprimir muestra:** De cara a poder descargar la muestra para futuros análisis manuales, cada muestra se comprime con una contraseña aleatoria, y se deja disponible para el público, o para el usuario que ha realizado en análisis.
- **Realiza análisis:** Esta es la funcionalidad principal de la herramienta. Obtiene el fichero y realiza un análisis estático, dependiendo del tipo de fichero.
- **Guardar resultados:** Los resultados se almacenan en la base de datos para que se puedan consultar en cualquier momento.
- **Generar informe:** Se puede generar un informe con los datos del análisis, disponible para su descarga.

5. ARQUITECTURA Y DISEÑO DE LA HERRAMIENTA

En este capítulo se pretende abordar el tema del diseño de la herramienta, especialmente su arquitectura.

Al tratarse de una aplicación web, se compone de un frontend y un backend. El frontend se ha realizado con HTML y CSS, apoyado en el framework Bootstrap de Twitter, mientras que el backend se ha desarrollado con el lenguaje PHP.

El backend de la aplicación se ha programado teniendo en cuenta el paradigma POO (Programación Orientada a Objetos), por lo que se han desarrollado diversas clases que se encargan de funcionalidades muy concretas.

Para interactuar con la base de datos se utiliza un DAO (*Data Access Object)*, que es la clase que contiene todos los métodos y atributos para realizar las operaciones necesarias con la capa de persistencia, que en este caso es un SGBD MySQL.

Además, se ha estructurado la aplicación siguiendo un patrón MVC (Modelo Vista Controlador), lo que permite tener una capa de presentación para las vistas, una capa de controlador que recoge los datos desde la vista y los entrega al modelo, que es la capa encargada de realizar las funciones de negocio.

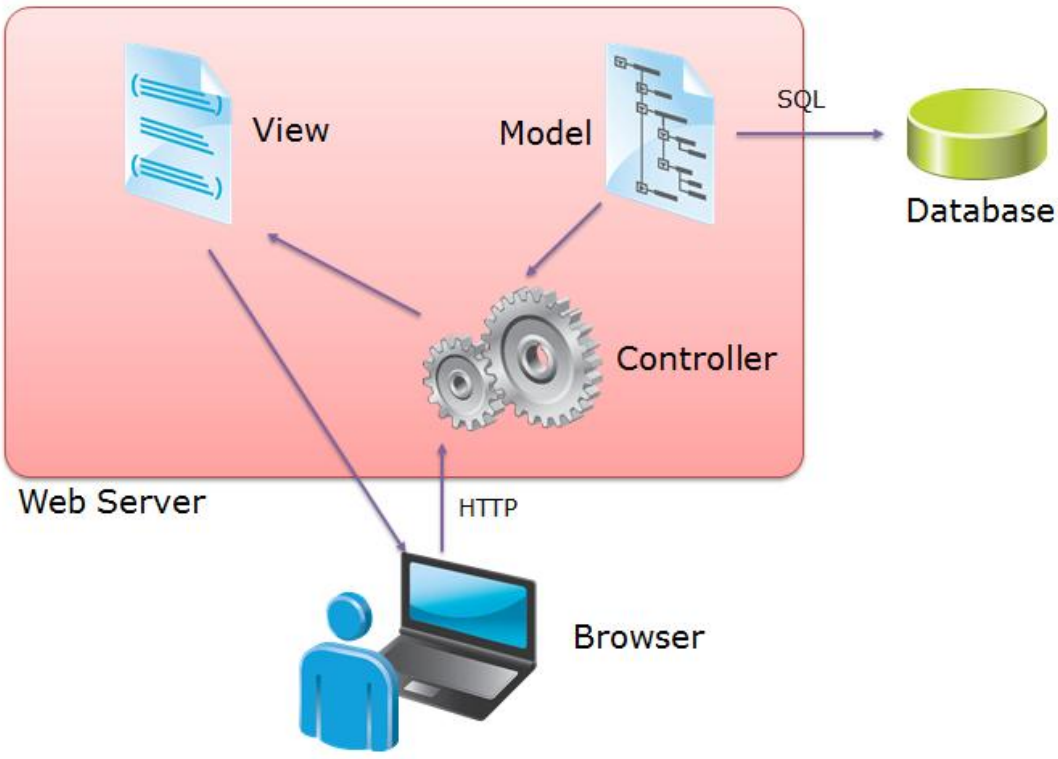

Figura 10: Modelo Vista Controlador

Incluso se podría considerar una capa adicional, la capa de persistencia, que es la que se encarga de comunicarse con el SGBD, que como ya se ha comentado anteriormente, se trata del DAO desarrollado. Este comportamiento queda reflejado en la Figura 10.

6. ESQUEMA DE LA BASE DE DATOS

El esquema de la base de datos es muy sencillo, y se detalla en la siguiente figura. Consta de tres tablas, que gestionan diferentes aspectos de la aplicación.

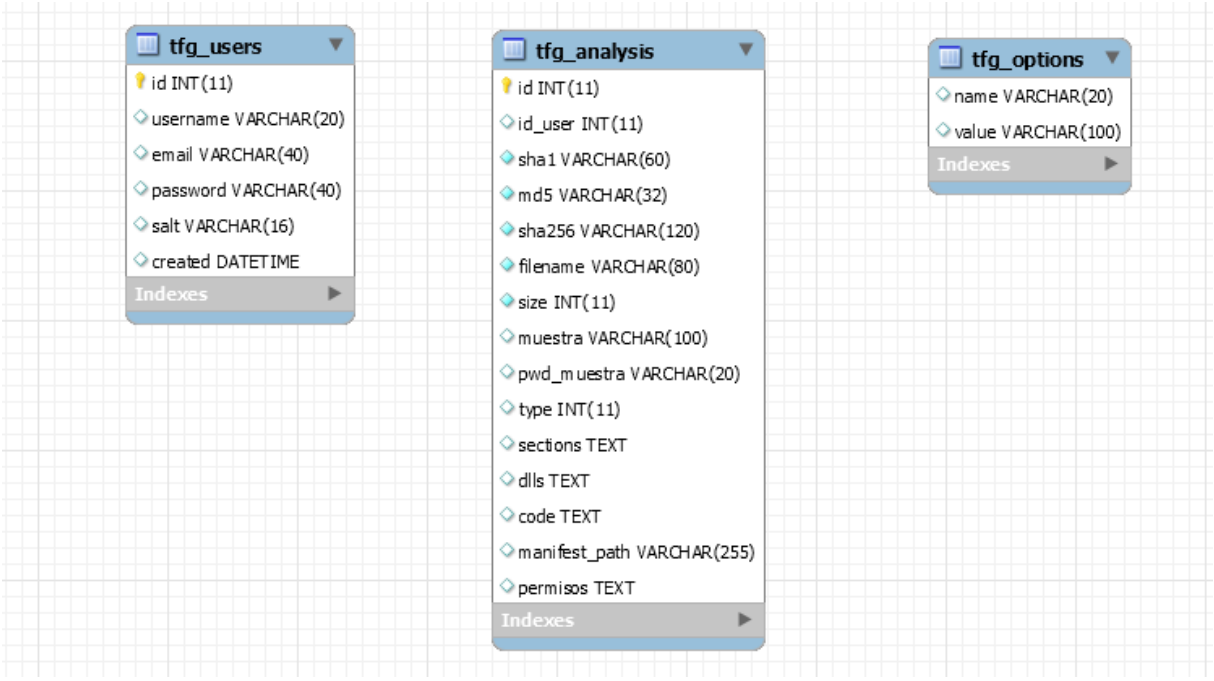

Figura 11: Esquema de base de datos

Se utiliza un prefijo en las tablas, en este caso "tfg_", por motivos de seguridad, para evitar un descubrimiento de tablas en un posible ataque de SQLi (inyecciones de SQL).

Como se puede observar, la base de datos consta de tres tablas:

- **tfg_users:** Almacena datos de los usuarios, como el email, nombre de usuario y contraseña. La contraseña se almacena "hasheada" y con "salt".
- **tfg_analysis**: Es la tabla que almacena los resultados de los análisis.
- **tfg_options**: Aquí se almacenan algunos datos que utiliza la aplicación, como la ruta absoluta, o claves de cifrado RC4 para la serialización de datos.

7. ESTUDIO DE DIFERENTES FORMATOS DE INFORME

La herramienta desarrollada almacena los resultados de los diferentes análisis de ficheros en una base de datos MySQL. Estos resultados se pueden consultar a través de la aplicación.

Sin embargo, es muy posible que se quiera disponer de una versión descargable e imprimible de los resultados de dicho análisis. Debido a esto se ha desarrollado un modulo de generación de informes automáticos, a partir de los datos almacenados en la base de datos.

A la hora de desarrollar este módulo se tuvieron en cuenta varios formatos de informe, entre ellos:

- Formato .doc(x), de Mifrosoft Word
- Formato .html
- Formato .xls, de Microsoft Excel
- Formato PDF

Para el caso del formato de MIcrosoft Word, hubiese sido una opción válida y viable, pues en realidad un archivo .docx no es más que un conjunto de fichero XML comprimidos con una estructura determinada. Sin embargo, se descartó esta opción debido a que Microsoft Word es un software propietario, y el formato no es del todo compatible con LibreOffice u OpenOffice, lo que podría producir fallos visuales accidentales.

Otra opción perfectamente válida es un archivo html con los datos del análisis, pero como ya se pueden consultar a través de la aplicación, se pensó que en caso de necesitar un informe, se puede guardar el documento de la página como un fichero html.

Se barajó la posibilidad de generar un informe en Excel, pero se descartó rápidamente debido a que se consideró que no aportaba demasiado valor al usuario final, y lector del informe.

Al final se optó por lo más obvio, un informe en PDF, ya que es un formato estándar, que no depende de software propietario, está muy extendido y no hay riesgo de cambios accidentales en el formato visual del informe.

Para realizar esta funcionalidad se utiliza un módulo de PHP que permite convertir lo que se mostraría en el buffer, en un fichero PDF. Esta librería se llama **dompdf**.

8. DISEÑO TÉCNICO DE LA HERRAMIENTA (UML)

En este capítulo se va a abordar el diseño técnico de la herramienta, mostrando diferentes diagramas UML y explicando el detalle de cada uno de ellos.

El diagrama que indica las funcionalidades de la aplicación es el diagrama de casos de usos, del que no se va a entrar en detalle, pues ya se ha explicado en el capítulo 4.

El diagrama que muestra la interacción entre componentes de la aplicación es el diagrama de clases, que se procede a explicar a continuación.

8.1 Diagrama de clases

Un diagrama de clases representa las relaciones que tienen las clases de la aplicación. El diagrama de clases de la herramienta desarrollada se detalla a continuación:

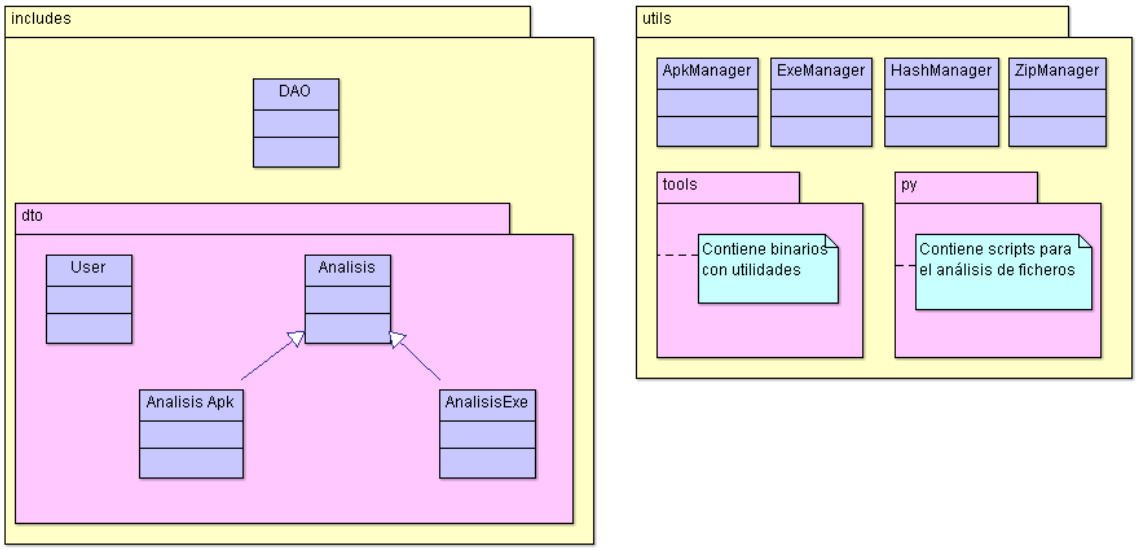

Figura 12: Diagrama de clases

Tal y como se puede observar en la Figura 12, la aplicación dispone de dos paquetes que contienen las clases. Estos paquetes son *includes* y *utils.*

En el paquete *includes* se encuentran clases que representan las entidades de los componentes de la aplicación. En la raíz de este directorio se encuentra la clase DAO, que es la encargada de la comunicación con el SGBD MySQL. También se incluye un directorio llamado *dto* que, como su nombre indica, se incluyen los DTO (*Data Transfer Object*) de las entidades, que son unas clases que tienen los atributos que modelan una tabla, de forma

que para guardar o recuperar datos de esta tabla, se recupera o inserta un objeto instancia de estas clases.

En el paquete *utils* se encuentran unas clases que funcionan como una capa de controlador, que interactúan con las herramientas y scripts utilizados para analizar los ficheros binarios. En definitiva, se tratan de clases que abstraen la comunicación con los scripts Python.

Se puede observar que, por simplicidad, no se han incluido atributos ni métodos de las clases. Esto es así para evitar que el diagrama fuese excesivamente grande. Sin embargo, a continuación se muestra el detalle de las clases una por una.

8.1.1 Clase DAO

Figura 13: Clase DAO

Esta es la clase encargada de gestionar la comunicación, y las operaciones, con la base de datos. Por ello, tiene cuatro atributos (host, user, password, database) que se utilizar para establecer la conexión. El resto de atributos se utilizan para referenciar a las tablas.

Contiene diferentes métodos que se explican a continuación:

- *escape(query):* Recibe una consulta SQL y la "escapa", evitando posibles inyecciones de código SQL.
- *getOption(name):* Recupera una opción de la tabla opciones en base a su nombre.
- *setOption(name, value):* Establece una opción en la tabla de opciones.
- *getSaltFromUser(username):* Obtiene el *salt* de un usuario. Utilizado para la autenticación.
- *getUser(username, password):* Obtiene un usuario que tenga el nombre de usuario y la contraseña indicada. Utilizado para la autenticación.
- *existsUsername(username):* Comprueba si un nombre de usuario ya existe.
- *existsEmail(email):* Comprueba si un mail ya se encuentra registrado en el sistema.
- *saveUser(user):* Inserta un usuario en la base de datos
- *existsAnalisis(sha256):* Comprueba si un análisis existe en base al hash SHA256 del fichero analizado.
- *saveAnalisis(análisis):* Almacena los resultados de un análisis en la base de datos.
- *getPublicAnalysis():* Obtiene una lista de todos los análisis públicos.
- *getPrivateAnalysis(id):* Obtiene un análisis privado en base al id de usuario.
- *getAnalysis(sha256):* Obtiene un análisis en base al hash SHA256 del fichero.
- *close():* Cierra la instancia de base de datos.

8.1.2 Clase User

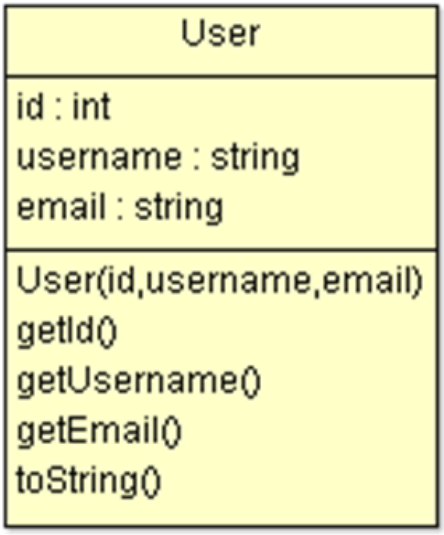

Figura 14: Clase User

Esta clase representa un DTO, por lo que tiene unos atributos que modelan al usuario, y unos métodos *getter* para obtener estos atributos. Debido a que tiene poco atributos, se ha prescindido de los métodos *setter*, y se han incluido en el constructor de la clase:

- *User(id, username, email):* Constructor de la clase que crea una instancia de la clase y establece sus atributos con los valores indicados
- *getId():* Obtiene el identificador de usuario
- *getUsername():* Obtiene el nombre de usuario
- *getEmail():* Obtiene el email del usuario
- *toString():* Devuelve una representación en forma de String del objeto.

8.1.3 Clase Analisis

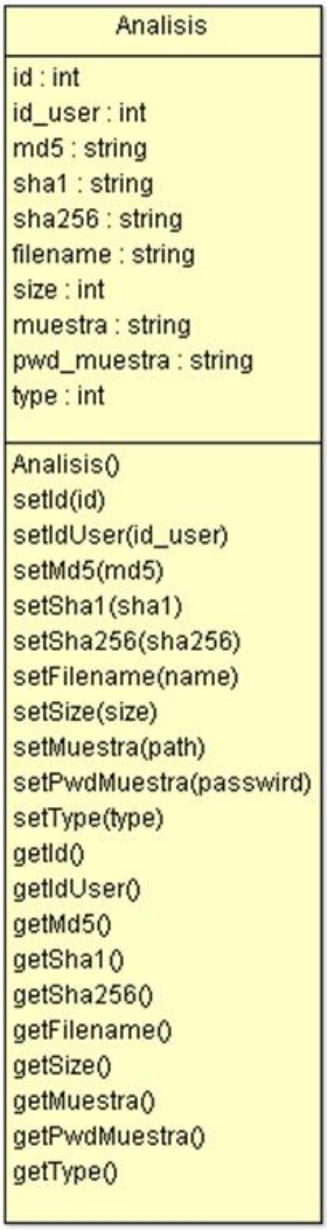

Figura 15: Clase Análisis

Esta es la clase base para los análisis, y es la clase padre para AnálisisExe y AnálisisApk, que se detallarán más adelante. También se trata de un DTO, por lo que tiene atributos y métodos *getter* y *setter*.

- *Analisis():* Constructor vacío de la clase
- *setId(id):* Establece el valor de su atributo id
- *setIdUser(id_user):* Establece el valor de su atributo id_user
- *setMd5(md5):* Establece el valor de su atributo md5
- *setSha1(sha1):* Establece el valor de su atributo sha1

- *setSha256(sha256):* Establece el valor de su atributo sha256
- *setFilename(name):* Establece el valor de su atributo filename
- *setSize(size):* Establece el valor de su atributo size
- *setMuestra(path):* Establece el valor de su atributo muestra
- *setPwdMuestra(password):* Establece el valor de su atributo pwd_muestra
- *setType(type):* Establece el valor de su atributo type.
- *getId():* Obtiene el valor de su atributo id
- *getIdUser():* Obtiene el valor de su atributo id_user
- *getMd5():* Obtiene el valor de su atributo md5
- *getSha1():* Obtiene el valor de su atributo sha1
- *getSha256():* Obtiene el valor de su atributo sha256
- *getFilename():* Obtiene el valor de su atributo filename
- *getSize():* Obtiene el valor de su atributo size
- *getMuestra():* Obtiene el valor de su atributo muestra
- *getPwdMuestra():* Obtiene el valor de su atributo pwd_muestra
- *getType():* Obtiene el valor de su atributo type.

8.1.4 Clase AnalisisExe

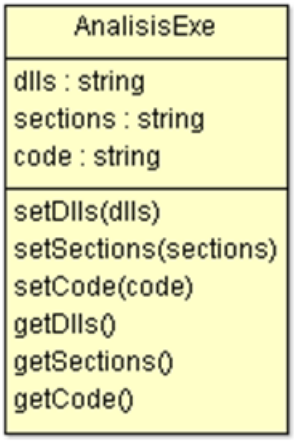

Figura 16: Clase AnalisisApk

Esta clase hereda de la clase Análisis, por lo que tendrá acceso a todos sus métodos no declarados como private. También se trata de un DTO, por lo que tiene atributos y métodos *getter* y *setter.*

- *setDlls(dlls):* Establece el valor de su atributo dlls
- *setSections(sections):* Establece el valor de su atributo sections
- *setCode(code):* Establece el valor de su atributo code
- *getDlls():* Obtiene el valor de su atributo dlls
- *getSections():* Obtiene el valor de su atributo sections
- *getCode():* Obtiene el valor de su atributo code

8.1.5 Clase AnalisisApk

Figura 17: Clase AnalisisApk

Esta clase hereda de la clase Análisis, por lo que tendrá acceso a todos sus métodos no declarados como private. También se trata de un DTO, por lo que tiene atributos y métodos *getter* y *setter.*

- *setManifest(path):* Establece el valor de su atributo manifest
- *setPermisos(permisos):* Establece el valor de su atributo permisos
- *getManifest():* Obtiene el valor de su atributo manifest
- *getPermisos():*Obtiene el valor de su atributo permisos

8.1.6 Clase ApkManager

Figura 18: Clase ApkManager

Esta clase actúa como capa de abstracción de los script y utilidades utilizadas para analizar un fichero apk.

- *ApkManager(file):* Constructor de la clase. Recibe la ruta de un fichero apk.
- *getAndroidManifest(output):* Decompila el fichero apk en la ruta indicada y devuelve el path al fichero AndroidManifest descifrado.
- *getPermisos(fileManifest):* Recibe la ruta del fichero AndroidManifest, lo lee y lo parsea para obtener una lista de los permisos solicitados por la aplicación.

8.1.7 Clase ExeManeger

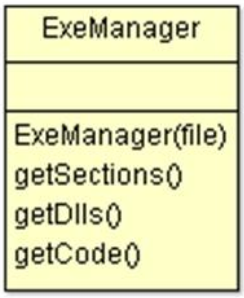

Figura 19: Clase ExeManager

Esta clase actúa como capa de abstracción de los script y utilidades utilizadas para analizar un fichero exe, en formato PE.

- *ExeManager(file):* Constructor de la clase. Recibe la ruta del fichero exe.
- *getSections():* Analiza el fichero y obtiene las diferentes secciones que contiene, así como su dirección y tamaño.
- *getDlls():* Obtiene una lista de todas las DLL que utiliza, asi como sus funciones
- *getCode():* Desensambla el fichero y obtiene su código ensamblador ASM

8.1.8 Clase HashManager

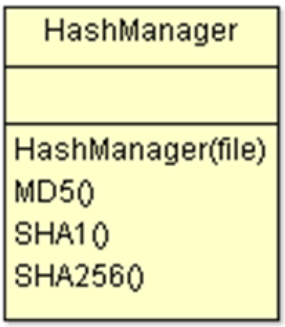

Figura 20: Clase HashManager

Esta clase actúa como capa de abstracción para las operaciones de hash que se realizan sobre un fichero.

- *HashManager(file):* Constructor de la clase. Recibe la ruta a un fichero.
- *MD5():* Calcula el hash MD5 del fichero
- *SHA1():* Calcula el hash SHA1 del fichero
- *SHA256():* Calcula el hash SHA256 del fichero

8.1.9 Clase ZipManager

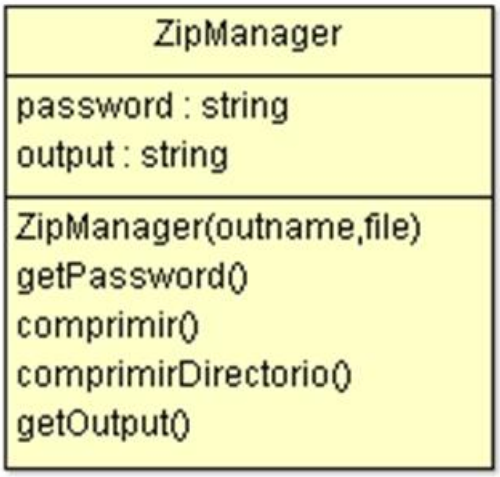

Figura 21: Clase ZipManager

Esta clase actúa como capa de abstracción para las operaciones de compresión que se realizan sobre los ficheros o directorios. Tiene dos atributos, *password* y *output.* El primero se utiliza para la compresión de muestras de malware, y se genera de forma aleatoria. El segundo es la ruta de destino del fichero comprimido.

- *ZipManager(outname, file):* Constructor de la clase, recibe el nombre del zip resultante y el fichero (o directorio) a comprimir. También generará el password para este fichero.
- *getPasssword():* Obtiene el password del fichero
- *comprimir():* Comprime el fichero indicado en el constructor, con el password generado. Utilizado para las muestras de malware.
- *comprimirDirectorio():* Comprime un directorio sin contraseña. Utilizado para la compresión del código smali de una aplicación Android.
- *getOutput():* Obtiene la ruta del fichero comprimido resultante.

8.2 Diagramas de Secuencia

Los diagramas de secuencia muestran una interacción entre clases, en el que se detallan llamadas a métodos y paso de mensajes entre estos componentes.

En este caso se han realizado dos diagramas de secuencia, que muestran el comportamiento de la aplicación para los dos tipos de análisis soportados, tanto de un fichero exe, como de un fichero apk.

8.2.1 Análisis de un fichero EXE

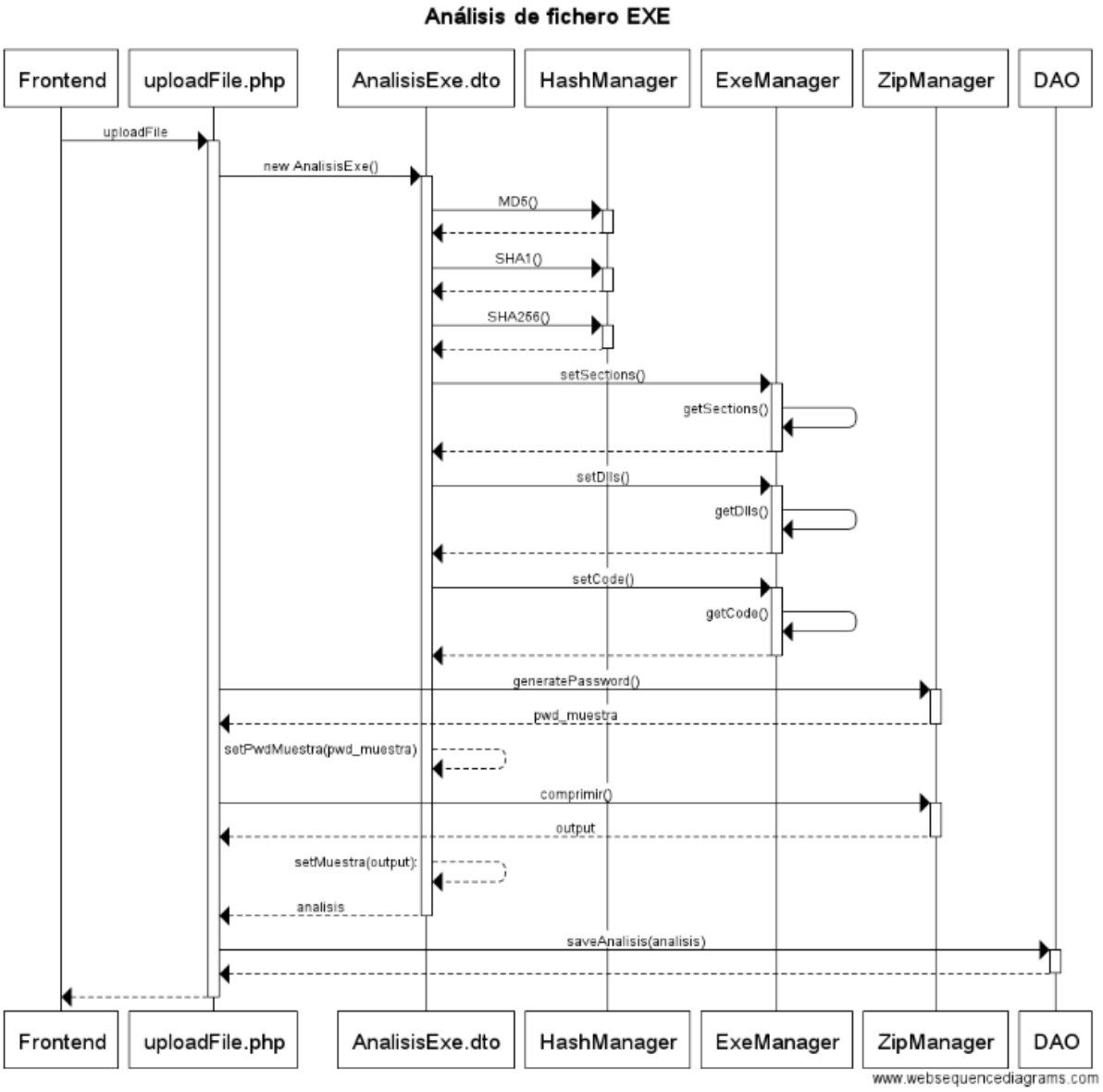

Figura 22: Diagrama de secuencia (Analisis EXE)

8.2.2 Análisis de un fichero APK

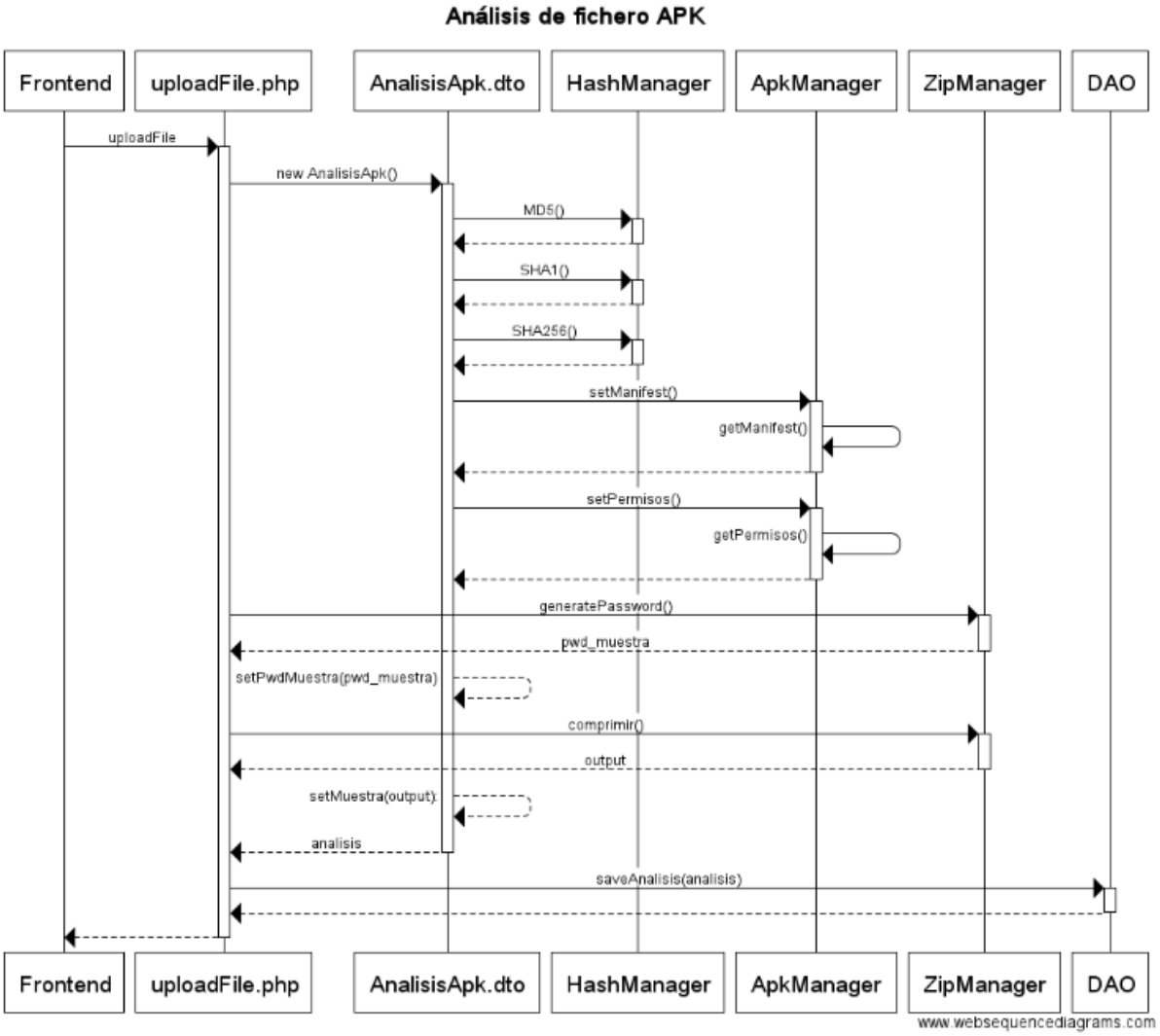

Figura 23: Diagrama de secuencia (Analisis Apk)

8.3 Diagrama de flujo

Los diagramas de flujo muestran el flujo de ejecución de un programa, teniendo en cuenta las posibles alternativas que existan en un punto dado. Un diagrama de flujo debe tener un inicio y un final.

A continuación se muestran diagramas de flujo de algunos de los procesos más significativos de la aplicación.

8.3.1 Analizar un fichero

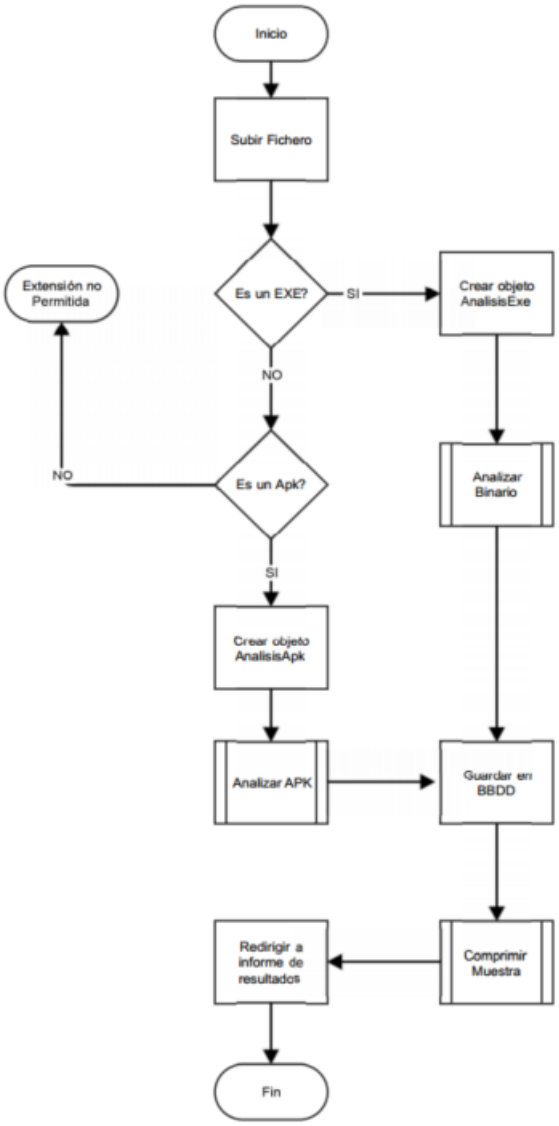

Figura 24: Diagrama de flujo (Analizar un fichero)

En el diagrama de flujo anterior se puede observar varios componentes. Los rectángulos ovalados representan los puntos de inicio y de fin. Los rombos representan una decisión condicional. Los rectángulos son procesos triviales, y los rectángulos con una línea en los laterales, representan subprocesos más complejos, que a su vez también tienen asociado un diagrama de flujo.

Por ello, a continuación se detallarán los diagramas de los subprocesos indicados.

8.3.2 Analizar un fichero binario

Figura 25: Diagrama de Flujo (Analizar un fichero binario)

8.3.3 Analizar un fichero APK

Figura 26: Diagrama de Flujo (Análisis de un fichero APK)

8.3.4 Comprimir Muestra

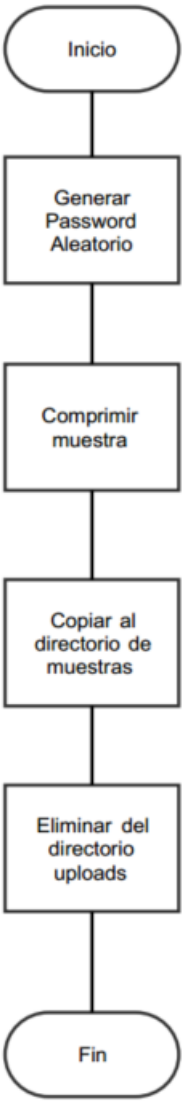

Figura 27: Diagrama de Flujo (Comprimir Muestra)

9. ANÁLISIS DE FICHEROS

Durante este capítulo se va a detallar cómo se obtiene la diferente información de los ficheros que se analizar, tanto a nivel teórico, como a nivel práctico.

9.1 Análisis de ficheros exe

A lo largo de este documento, cuando se hace referencia a un fichero ejecutable (.exe), en realidad se está haciendo referencia a un tipo de fichero que tiene el formato **Portable Executable (PE)**, que es el que tienen los programas en un sistema operativo Windows con una arquitectura de 32 bits.

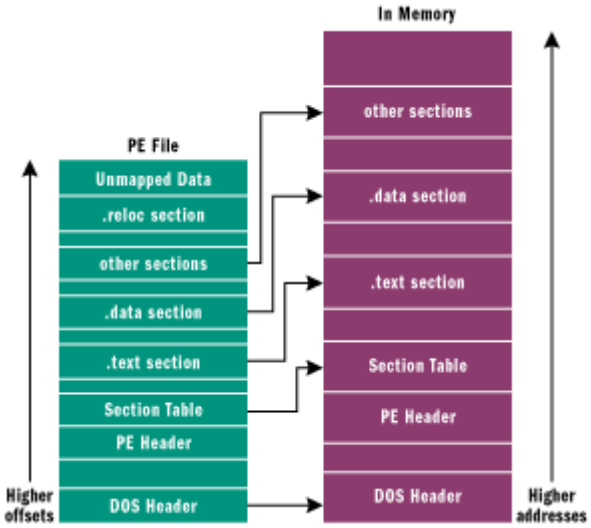

Figura 28: Formato PE

Los archivos de este tipo tienen una cabecera y un cuerpo. Las cabeceras contienen información que es utilizada por el *loader* de Windows para poder ejecutar los programas. Dentro de la sección de cabeceras, hay varios tipos, como se detalla en la Tabla 2.

Tabla 2 - Estructura de un fichero PE

Elemento	Descripción
DOS Header	Cabecera de archivos DOS
DOS Stub	Mensaje en sistemas DOS
NT Header	Inicio del ejecutable
File Header	Información base
Optional Header	Información adicional
Data Directory	*Image Data Directory*
Section Header	Información de las secciones del ejecutable
Cuerpo	Contenido de cada seccion
Final del fichero	Fin del fichero

Alguna de las principales cabeceras se detallan a continuación:

- **DOS HEADER:** Es una cabecera obsoleta que solo tiene dos campos útiles:
  - **e_magic:** Contiene las siglas 'MZ' que todo archivo ejecutable debe tener.
  - **l_fanew:** Puntero a NT_HEADER, que empiezo por 'PE\x0\x0'
- **DOS STUB:** Es una aplicación válida que muestra por consola el mensaje "This program cannot be run in DOS mode".
- **NT HEADER:** A partir de 'PE\x0\x0' se encuentra la estructura de IMAGE_FILE_HEADER.

9.1.1 Cabecera IMAGE_FILE_HEADER

Esta cabecera contiene información general sobre el fichero. Sigue la estructura que se detalla en la Tabla 3. En esta sección se detallan sus campos:

- **Machine:** Tipo de arquitectura de la máquina donde se puede ejecutar el programa.
- **NumberOfSections:** Numero de secciones que tiene el ejecutable.
- **TimeDateStamp**: Timestamp del fichero. Indica la fecha de creación.
- **PointerToSymbolTable**: En archivos ejecutables su valor es 0.
- **NumberOfSymbols**: En archivos ejecutables su valor es 0.
- **SizeOfOptionalHeader**: Tamaño de la cabecera IMAGE_OPTIONAL_HEADER
- **Characteristics**: Es el valor de la suma de diferentes flags, que indican características del fichero. Los valores de las características se detallan en la Tabla 4.

Tabla 3 - Estructura de IMAGE_FILE_HEADER

Campo	Tamaño
Machine	2 bytes
NumberOfSections	2 bytes
TimeDateStamp	4 bytes
PointerToSymbolTable	4 bytes
NumberOfSymbols	4 bytes
SizeOfOptionalHeader	4 bytes
Characteristics	4 bytes

Tabla 4 - Caracteristicas de un fichero PE

Constante	Valor
IMAGE_FILE_RELOCS_STRIPPED	0x0001
IMAGE_FILE_EXECUTABLE_IMAGE	0x0002
IMAGE_FILE_LINE_NUMS_STRIPPED	0x0004
IMAGE_FILE_LOCAL_SYMS_STRIPPED	0x0008
IMAGE_FILE_AGGRESSIVE_WS_TRIM	0x0010
IMAGE_FILE_LARGE_ADDRESS_AWARE	0x0020
Unknown	0x0040
IMAGE_FILE_BYTES_REVERSED_LO	0x0080
IMAGE_FILE_32BIT_MACHINE	0x0100
IMAGE_FILE_DEBUG_STRIPPED	0x0200
IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP	0x0400
IMAGE_FILE_NET_RUN_FROM_SWAP	0x0800
IMAGE_FILE_SYSTEM	0x1000
IMAGE_FILE_DLL	0x2000

9.1.2 Cabecera IMAGE_OPTIONAL_HEADER

La estructura de esta cabecera se detalla en la Tabla 5, y sus campos se detallan a continuación:

- **Magic:** Determina si el ejecutable es para sistemas x86 o x64
- **MajorLinkerVersion:** Versión más alta del enlazador
- **MinorLinkerVersion**: Versión más baja del enlazador
- **SizeOfCode**: Tamaño de la sección del código
- **SizeOfInitializedData**: Tamaño de los datos inicializados
- **SizeOfUninitializedData**: Tamaño de los datos sin inicializar
- **AddressOfEntryPoint**: Dirección Virtual Relativa hacia la primera instrucción
- **BaseOfCode:** Dirección Virtual Relativa hacia la primera instrucción de la sección de código
- **BaseOfData:** Dirección Virtual Relativa hacia la primera instrucción de la sección de datos
- **ImageBase:** Dirección de preferencia donde se cargará el ejecutable.
- **SectionAlignment:** Alineamiento en bytes de las secciones del ejecutable en memoria
- **FileAlignment:** Alineamiento para los datos físicos de las secciones.
- **MajorOperatingSystemVersion**: Versión principal del sistema operativo requerido.
- **MinorOperatingSystemVersion**: Versión mínima del sistema operativo requerido.
- **MajorImageVersion**: Versión principal del ejecutable
- **MinorImageVersion**: Versión mínima del ejecutable
- **MajorSubsystemVersion**: Versión principal del subsistema
- **MinorSubsystemVersion**: Versión mínima del subsistema
- **Win32VersionValue**: Reservado para el sistema
- **SizeOfImage**: Tamaño a reservar en la memoria para cargar el ejecutable.
- **SizeOfHeaders**: Tamaño de todas las cabeceras juntas
- **CheckSum**: Suma de verificación del fichero.
- **Subsystem**: Indica el subsistema requerido para ejecutar el programa
- **DllCharacteristics**: Suma de los flags de caracteristicas de una dll
- **SizeOfStackReserve**: Numero de bytes a reservar para la pila
- **SizeOfStackCommit**: Parte de la memoria reservada para la pila que se verá comprometida
- **SizeOfHeapReserve**: Tamaño a reservar para el heap
- **SizeOfHeapCommit**: Parte de la memoria reservada para el heap que se verá comprometido

- **LoaderFlags**: Obsoleto
- **NumberOfRvaAndSizes**: Cantidad de entradas del DATA_DIRECTORY

Tabla 5 - Estructura de IMAGE_OPTIONAL_HEADER

Campo	Tamaño
Magic	2 bytes
MajorLinkerVersion	1 byte
MinorLinkerVersion	1 byte
SizeOfCode	4 bytes
SizeOfInitializedData	4 bytes
SizeOfUninitializedData	4 bytes
AddressOfEntryPoint	4 bytes
BaseOfCode	4 bytes
BaseOfData	4 bytes
ImageBase	4 bytes
SectionAlignment	4 bytes
FileAlignment	4 bytes
MajorOperatingSystemVersion	2 bytes
MinorOperatingSystemVersion	2 bytes
MajorImageVersion	2 bytes
MinorImageVersion	2 bytes
MajorSubsystemVersion	2 bytes
MinorSubsystemVersion	2 bytes
Win32VersionValue	4 bytes
SizeOfImage	4 bytes
SizeOfHeaders	4 bytes
CheckSum	4 bytes
Subsystem	2 bytes
DllCharacteristics	2 bytes
SizeOfStackReserve	4 bytes
SizeOfStackCommit	4 bytes
SizeOfHeapReserve	4 bytes
SizeOfHeapCommit	4 bytes
LoaderFlags	4 bytes

NumberOfRvaAndSizes	4 bytes

9.1.3 Cabecera IMAGE_DATA_DIRECTORY

Es una matriz de una cantidad determinada de entradas, determinada por el valor del campo NumberOfRvaAndSizes de la cabecera IMAGE_OPTIONAL_HEADER, en la que cada entrada se estructura según lo que se puede ver en la Tabla 6.

Tabla 6 - Estructura de IMAGE_DATA_DIRECTORY

Campo	Tamaño
VirtualAddress	4 bytes
Size	4 byte

Un ejemplo de esta cabecera se puede ver en la Figura 29.

Directory Name	RVA	Size
Export Table	00000000	00000000
Import Table	00003794	000000C8
Resource Table	00005000	000D9848
Exception Table	00000000	00000000
Certificate Table	000E0000	00001BD8
Relocation Table	000DF000	00000234
Debug Data	00003180	0000001C
Architecture	00000000	00000000
GlobalPtr	00000000	00000000
TlsTable	00000000	00000000
LoadConfig	00003670	00000040
Bound Import	00000000	00000000
IAT	00003000	00000164
Delay Import	00000000	00000000
CLR Runtime Header	00000000	00000000
Reserverd	00000000	00000000

Figura 29: Ejemplo de IMAGE_DATA_DIRECTORY

9.1.4 Cabecera IMAGE_SECTION_HEADER

Esta cabecera contiene información acerca de las secciones del fichero. Su estructura se detalla en la Tabla 7.

Tabla 7 - Estructura de IMAGE_SECTION_HEADER

Campo	Tamaño
Name	8 bytes
VirtualSize	4 byte
VirtualAddress	4 byte
SizeOfRawData	4 byte
PointerToRawData	4 byte
PointerToRelocations	4 byte
PointerToLineNumbers	4 byte
NumberOfRelocations	2 byte
NumberOfLineNumbers	2 byte
Characteristics	4 byte

El detalle de los campos se explica a continuación:

- Name: Es un campo de 8 bytes con el nombre de la sección. Si dicho nombre ocupa menos de 8 bytes, se incluyen bytes nulos.
- VirtualSize: Tamaño que ocupará la sección una vez cargada en memoria.
- VirtualAddress: Dirección Virtual donde se cargaran los datos de la sección.
- SizeOfRawData: Tamaño de la información inicializada en disco.
- PointerToRawData: Desplazamiento del archivo donde se encuentra el primer byte de la sección en disco.
- PointerToRelocations: En ejecutables, este valor es cero.
- PointerToLineNumbers: En ejecutables, este valor es cero.
- NumberOfRelocations: En ejecutables, este valor es cero.
- NumberOfLineNumbers: En ejecutables, este valor es cero.
- Characteristics: Su valor es la suma de diferentes flags que representan características que se detallan en la Tabla 8.

Tabla 8 - Características de IMAGE_SECTION_HEADER

CARACTERISTICA	VALOR
IMAGE_SCN_CNT_CODE	0x00000020
IMAGE_SCN_CNT_INITIALIZED_DATA	0x0000040
IMAGE_SCN_CNT_UNINITIALIZED_DATA	0x00000080
IMAGE_SCN_MEM_SHARED	0x10000000
IMAGE_SCN_MEM_EXECUTE	0x20000000
IMAGE_SCN_MEM_READ	0x40000000
IMAGE_SCN_MEM_WRITE	0x80000000
IMAGE_SCN_MEM_NOT_PAGED	0x08000000
IMAGE_SCN_GPREL	0x00008000
IMAGE_SCN_MEM_DISCARDABLE	0x02000000
IMAGE_SCN_MEM_NOT_CACHED	0x04000000

9.1.5 Script de análisis

Teniendo en cuenta la información anterior, se ha desarrollado un script para el análisis de ficheros binarios, en formato PE.

Este script se encuentra en la carpeta utils/py/exeFile.py y utiliza las librerías **pefile** y **pydasm.**

El código del script se muestra a continuación:

```
#!/usr/bin/env python
#-*- encoding: utf-8 -*-
#
# Script que realiza el análisis de un fichero ejecutable en formato PE
#
# Author: Diego Fernández Valero

import sys, pefile, pydasm, json, re
```

```
def getCode(pe):
      """Obtiene la representación en código ensamblador de las instrucciones del fichero binario. Recibe un objeto PE, y devuelve un string con el código ASM"""
      code = ""
      ep = pe.OPTIONAL_HEADER.AddressOfEntryPoint
      ep_ava = ep + pe.OPTIONAL_HEADER.ImageBase
      data = pe.get_memory_mapped_image()[ep:]
      offset = 0
      l = long(len(data))
      while offset < l:
            i = pydasm.get_instruction(data[offset:], pydasm.MODE_32)
            if i is None:
                  break
            code += pydasm.get_instruction_string(i, pydasm.FORMAT_INTEL, ep_ava+offset) + "\n"
            offset += int(i.length)
      return code


def getSections(pe):
      """Obtiene un listado de las secciones de un archivo ejecutable, así como su dirección virtual, el tamaño de la sección, etc.  Recibe un objeto PE, y devuelve un diccionario en el que las claves son el nombre de la
      sección, y el valor una tupla con los datos recuperados."""
      sections = {}
      pattern = re.compile('\.?\w+')
      for section in pe.sections:
            m = pattern.match(section.Name)
            if m is None:
                  sectionName = section.Name
            else:
                  sectionName = m.group()
            sections[sectionName] = [hex(section.VirtualAddress), section.Misc_VirtualSize, section.SizeOfRawData]
      return sections
```

```
def getDlls(pe):
      """Obtiene un listado de las DLL que utiliza un fichero, y las funciones de cada DLL. Recibe un objeto PE, y devuelve un diccionario, en el que las claves son la dll y sus valores son las funciones y sus direcciones de memoria"""
      imports = pe.DIRECTORY_ENTRY_IMPORT
      dlls = {}

      for _import in imports:
            regs = []
            for reg in _import.imports:
                  regs.append( (hex(reg.address), reg.name) )
            dlls[_import.dll] = regs
      return dlls


if __name__ == "__main__":

      if len(sys.argv) <= 1:
            sys.exit(-1)
      elif len(sys.argv) == 2:
            sys.exit(-2)

      fichero = sys.argv[1]
      option = sys.argv[2]

      pe = pefile.PE(fichero)

      if option == "sections":
            print json.dumps(getSections(pe))
      elif option == "dll":
            print json.dumps(getDlls(pe))
      elif option == "code":
            print getCode(pe)
```

9.2 Análisis de ficheros APK

Un fichero APK no es más que un fichero comprimirdo con otra extensión. Tiene una estructura definida, y suele tener los siguientes archivos:

- AndroidManifest.xml
- classes.dex
- resources.arsc
- res/
- META-INF/
- lib/

AndroidManifest es un archivo codificado en XML Binario que contiene información acerca de la aplicación, como el nombre, versión, permisos, clases principales, etc. Para decodificar este fichero se utiliza la herramienta **apktool** que permite descomprimir el apk, y genera un directorio con el contenido de la aplicación que incluye, entre otras cosas:

- AndroidManfest.xml decodificado
- Directorio smali, con el código smali de la aplicación

Estos dos artefactos son los que utiliza WASA para inspeccionar de forma estática un fichero apk.

10. API DE LA APLICACIÓN

De cara a conocer la funcionalidad de las clases de la aplicación y sus métodos, se ha desarrollado una documentación que es accesible en formato HTML a través de los archivos contenidos en la carpeta *doc* del proyecto de la aplicación.

Las páginas de la aplicación se han desarrollado a partir de comentarios especiales en la aplicación, utilizando la utilidad **phpDocumentator**, y con el look and feel *responsive-twig*.

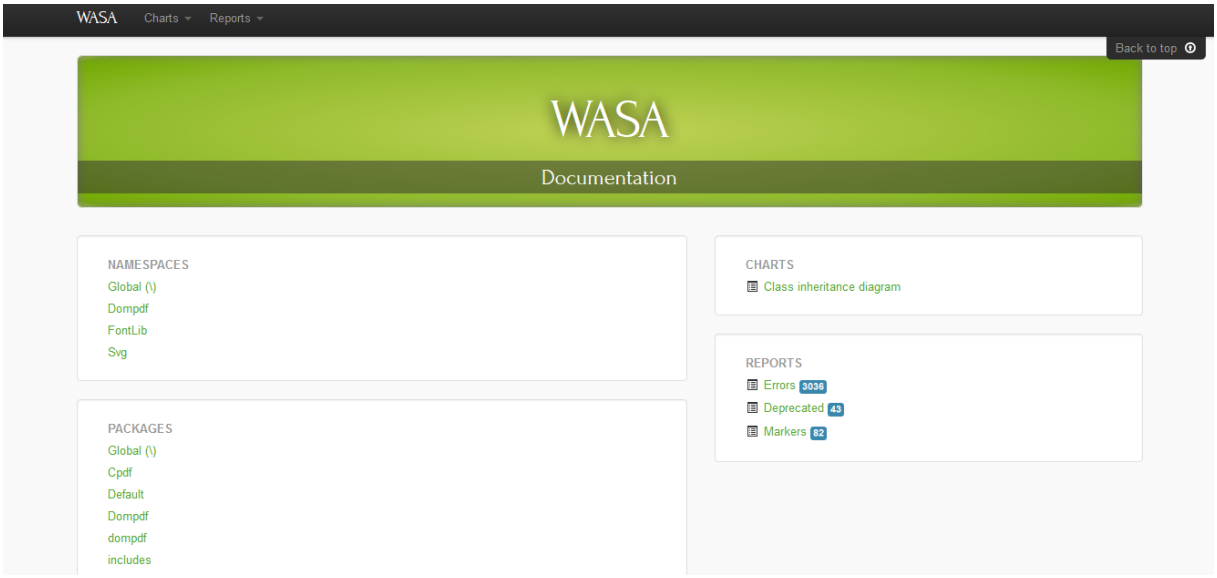

Figura 30: Página principal de documentación

En la documentación se hace un repaso a cada uno de los *namespaces* y *paquetes* de la aplicación. Hay que tener en cuenta que algnos paquetes se corresponden con librerias de terceros, que se han incluido en el código de la aplicación desarrollada. De hecho, los paquetes de estas librerías son:

- Cpdf
- Dompdf
- php-font-lib
- php-svg-lib
- svg

Los paquetes correspondientes con código propio son los siguientes:

- default
- includes

- utils

En el paquete *default* se encuentra toda la documentación de fichero que no se corresponden al paquete *utils* ni *includes*; o las funciones y constantes declaradas fuera de una clase.

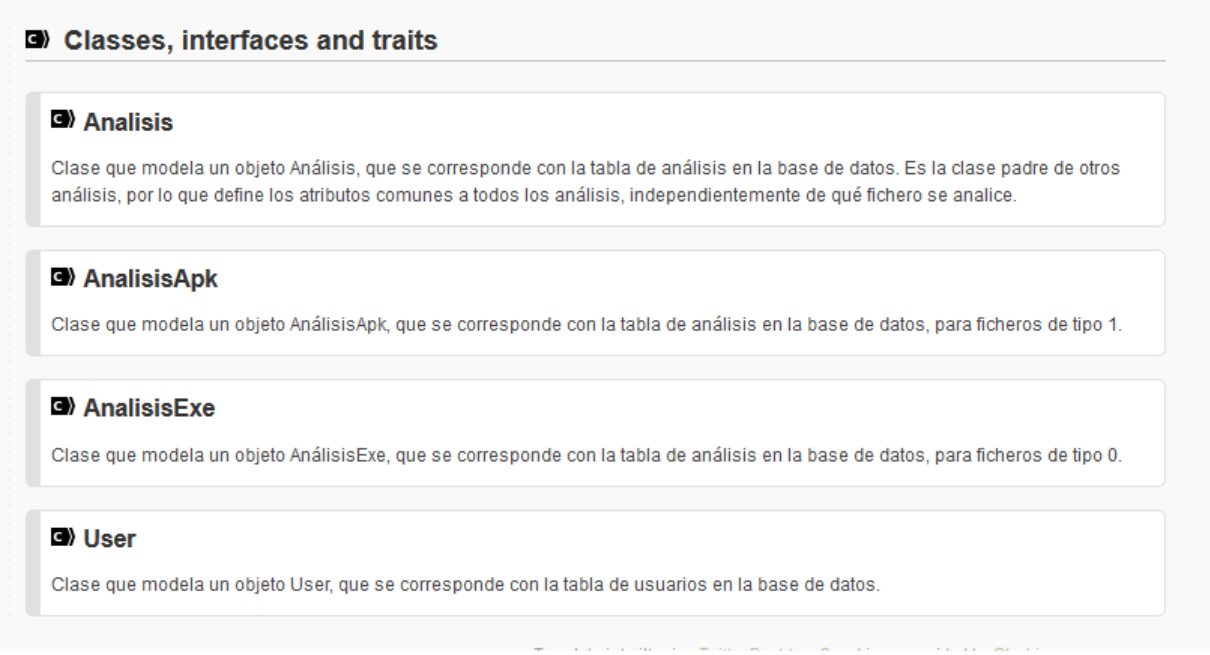

Figura 31: Documentación del paquete includes

Haciendo clic en el nombre de un paquete, se puede acceder a la documentación de las clases del mismo y, obviamente, pulsando en la clase, se accede a la documentación de la clase y sus métodos, como se puede ver en la Figura 32.

Así pues, esta serie de ficheros HTML facilitan la navegación interactiva a través de la documentación de las clases y métodos que componen la aplicación desarrollado. La aplicación de documentación generada es intuitiva y fácil de usar.

Figura 32: Documentación de la clase Analisis

11. GUÍA DE INSTALACIÓN

En este capítulo se pretende detallar los pasos necesarios para llevar a cabo el despliegue de la herramienta en un entorno local, y así poder probar su funcionamiento. Aunque el proceso de despliegue en un entorno de producción es similar, pero se deberían tener más precauciones, como una política de contraseñas más robustas, utilización de usuarios no privilegiados, etc.

Lo primero que se necesita es un servidor web y un servidor de base de datos. Para no complicarnos con la instalación, se va a optar por un software que incluya ambos integrados, como XAMPP, para plataformas Windows, o LAMPP para plataformas Linux. En este capítulo se va a detallar el proceso bajo un sistema operativo Windows.

Una vez se ha descargado e instalado el software, se accede al panel de control del software, para asegurarse de que el servidor web y base de datos estén ejecutándose, y sea como proceso o como servicio.

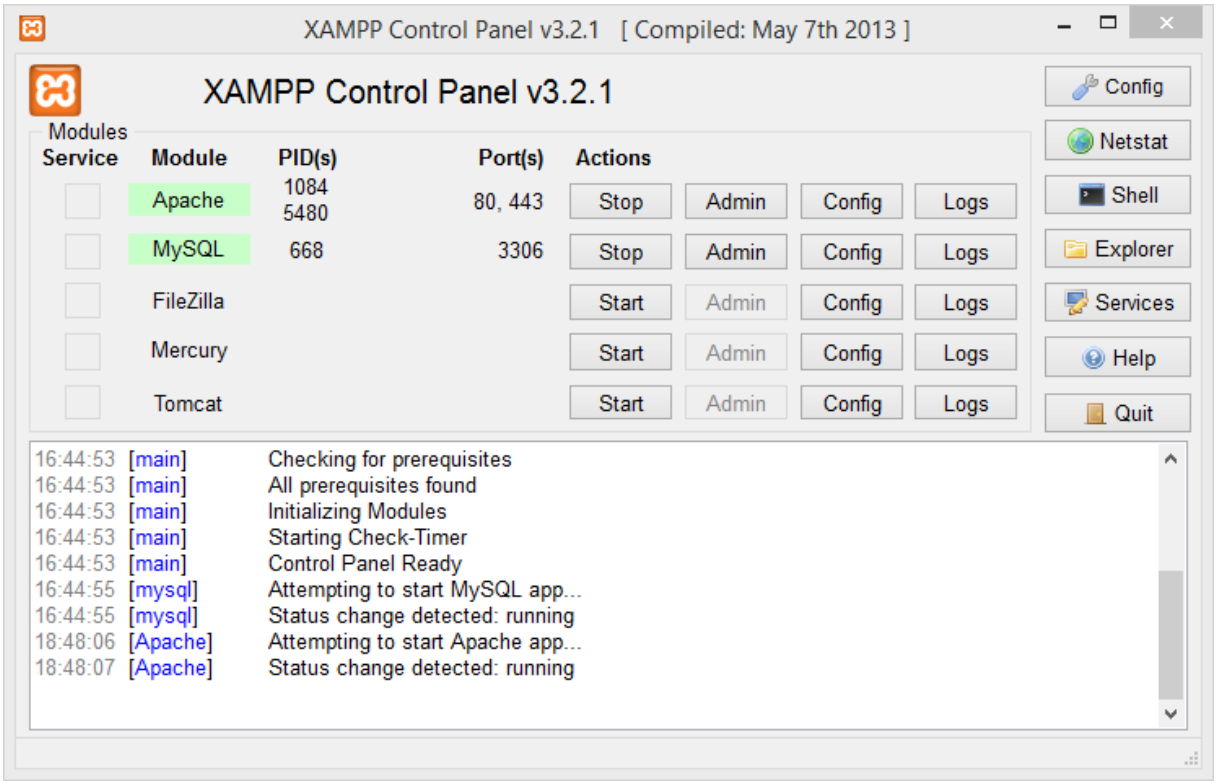

Figura 33: Panel de control de XAMPP

Una vez se tienen los servidores ejecutándose, se procede a crear la base de datos, y los datos iniciales. Dicha estructura y datos se encuentran en el archivo SQL que se encuentra en el repositorio, sólo hace falta ejecutar el archivo en una consola MySQL.

Para hacer esto, podemos optar por el típico cliente web *phpmyadmin*, pero yo prefiero optar por un software de escritorio, en este caso HeidiSQL. Una vez descargado e instalado, se abre una conexión al servidor local, con las credenciales indicadas durante la instalación, que por defecto son **root** como usuario, y sin contraseña.

Una vez abierta la conexión, se hace clic en Archivo > Cargar archivo SQL, y se selecciona el fichero sql descargado. Posteriormente se ejecuta el fichero, y ya se tendrá lista la base de datos.

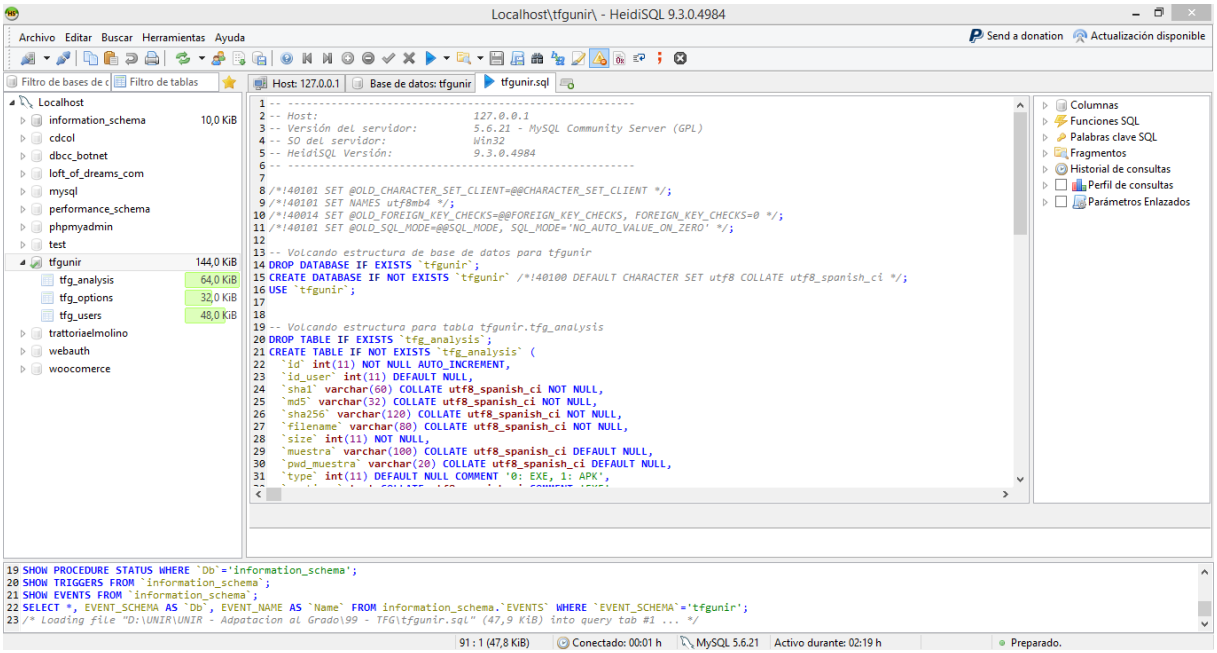

Figura 34: Ejecución del script SQL

Ahora solo queda cargar los archivos de la aplicación web en el servidor web. Por defecto, la ruta en la que XAMPP almacena las aplicaciones es C:\xampp\htdocs, así que se crea una carpeta en ese directorio, con el nombre de **tfg**, por ejemplo.

Po último, hay que modificar las credenciales de acceso a la base de datos en la aplicación. Para ello se edita el fichero **includes/database.php**, y se modifican los valores de los atributos de la clase DAO, tal y como se muestra en la siguiente imagen.

```php
class DAO{
    private static $host = 'localhost';
    private static $user = 'root';
    private static $password = '';
    private static $database = 'tfgunir';
    private static $prefix = 'tfg_';
    private static $tables = array(
            "users" => DB_PREFIX."users",
            "options" => DB_PREFIX."options",
            "analisis" => DB_PREFIX."analysis",

        );
```

Figura 35: Configuración de Base de Datos en la aplicación

Una vez realizados estos cambios, la aplicación ya está lista para funcionar. Simplemente hay que abrir un navegador web e introducir la url http://localhost/tfg

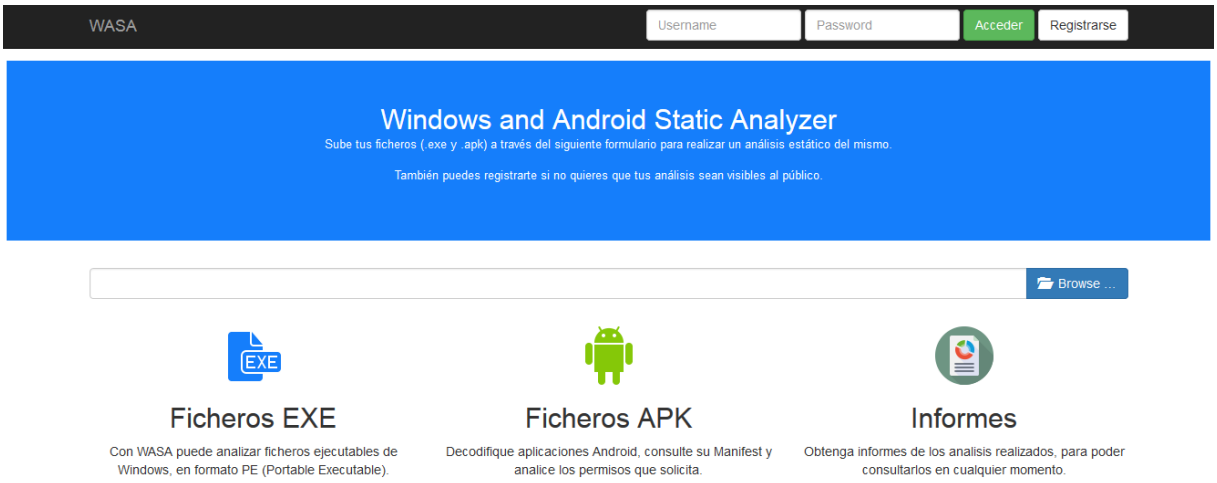

Figura 36: Windows and Android Static Analyzer

12. GUÍA DE USUARIO

En este capítulo se recogen las indicaciones para utilizar correctamente la aplicación. Nada más entrar en la aplicación vemos el formulario para subir un fichero. En este punto podemos realizar tres acciones: subir un fichero, iniciar sesión y registrarse.

12.1 Registrarse en la aplicación.

Para llevar a cabo el registro en la aplicación se debe pulsar el botón "Registrarse" de la barra superior. Esto cargará un formulario de registro en la página principal y, tras rellenarlo, se creará una cuenta de usuario.

12.2 Inicio de sesión

Para iniciar sesión tan sólo hay que introducir el nombre de usuario y la contraseña en los campos destinados para ello, también en la barra superior.

12.3 Subir un fichero

Para subir un fichero a la aplicación se hace clic en el botón "Browse" del formulario principal, y se selecciona el fichero que se desea subir, que debe tener extensión exe o apk.

Si se produce algún error, se mostrará en una alerta en rojo en la parte superior del formulario. Si va todo bien, aparecerá un enlace a la muestra comprimida, y al análisis, además de indicar la contraseña del archivo comprimido.

Una vez se sube un fichero, el análisis comienza automáticamente, en una tarea **síncrona**, por lo que hay que esperar a que el análisis termine, aunque al tratarse de un análisis estático, esto lleva muy poco tiempo.

Cuando el análisis ha terminado, y si ha sido un análisis público, se añadirá a la lista de análisis públicos de la página principal.

12.4 Consultar resultados

Los resultados de los análisis se pueden consultar haciendo clic en el enlace que aparece al terminar de subir y analizar el fichero, o en los enlaces de la lista de análisis anteriores, públicos o privados.

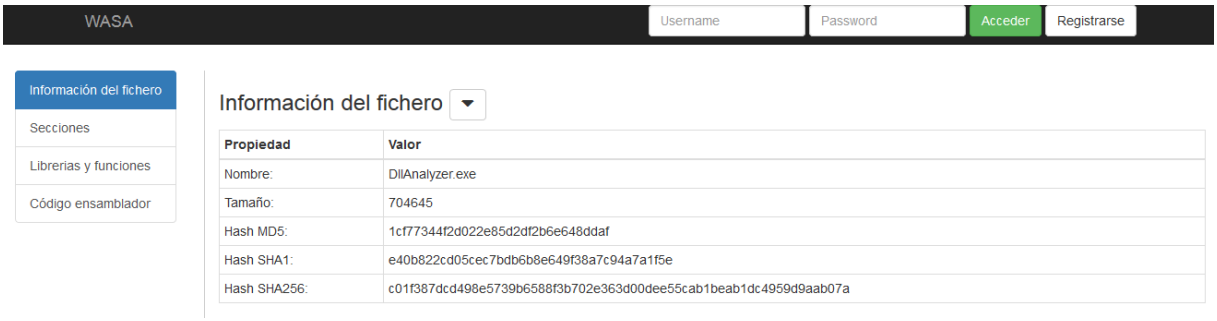

Figura 37: Análisis de un fichero EXE

En esta página, la información mostrada cambiará dependiendo del tipo de fichero que se haya analizado. Así pues, si es un fichero exe, se mostrará información del fichero, sus secciones de código, las librerías y sus funciones utilizadas, así como el código ensamblador del binario.

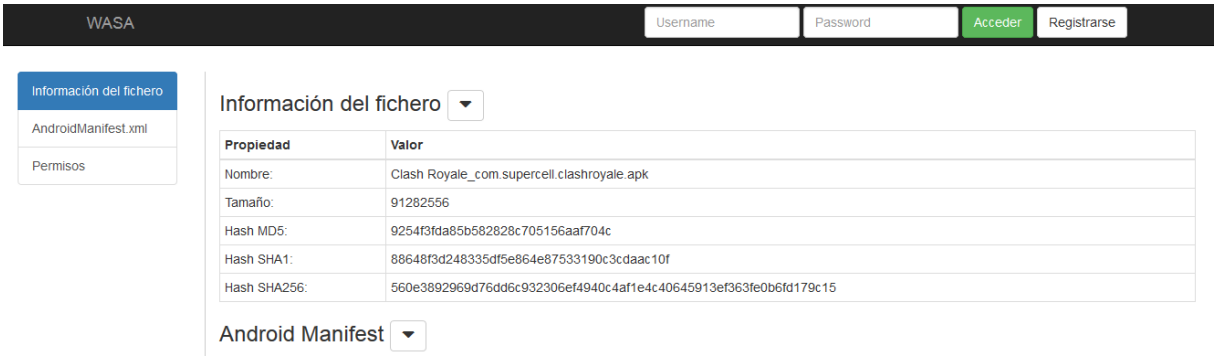

Figura 38: Análisis de un fichero APK

Sin embargo, si el fichero es una aplicación android, con extensión apk, la información mostrada será la relacionada con el fichero, el volcado de AndroidManifest decodificado, y una tabla con los permisos solicitados por la aplicación.

12.5 Descargar muestra

Se puede descargar la muestra del fichero una vez subido, a través del enlace que aparece en el mensaje de confirmación. También se puede descargar a través de la página de resultados del análisis.

12.6 Descargar informe

En la página de resultados de análisis se puede descargar el informe en formato PDF con los datos del análisis.

13. CONCLUSIONES

Tal y como se ha visto en el capítulo 2, en un procedimiento de análisis de malware se contempla tanto el análisis dinámico como el estático. El que más información aporta suele ser el análisis dinámico, pero el análisis estático ofrece otro tipo de información, que permite al analista hacerse una idea del comportamiento del malware, sin tener que ejecutarlo.

La herramienta que se ha desarrollado durante este Trabajo de Fin de Grado ofrece al analista de malware la posibilidad de automatizar la obtención de diferente información necesaria en un análisis estático.

Un análisis estático completo requiere de más información, y más tiempo de análisis. No obstante, la información que se obtiene con WASA es bastante útil e interesante de cara a afrontar un procedimiento a análisis de malware.

Es importante tener en cuenta que en un proceso de análisis de malware, el análisis estático es sólo una parte del proceso global. Por tanto, esta aplicación no es una herramienta de análisis de malware completa, sino que sirve como apoyo al analista.

En conclusión, esta herramienta puede ser el primer punto de entrada de un análisis de malware pero, tal y como se comenta en el capítulo 14 (más adelante), se debería añadir un soporte para análisis dinámico, y cruzar toda la información obtenida para poder generar un informe completo del malware analizado.

14. TRABAJO FUTURO

Debido al escaso tiempo que supone un semestre del curso, no ha sido posible incluir otro tipo de funcionalidades en la aplicación.

Si el proyecto sigue adelante, se deberían incluir al menos las siguientes funcionalidades:

- Soporte para el análisis dinámico en un entorno de SandBox
- Conexión a aplicaciones de terceros para consultar posibles análisis ya realizados
- Derivación de los IoC (indicadores de compromiso) del malware
- Desarrollo de una API que proporcione a terceros acceso a la base de datos de IoC
- Análisis del fichero con diferentes motores de antivirus
- "Modding" del malware.

A continuación se explican cada una de las funcionalidades que, lamentablemente, han quedado fuera del desarrollo del proyecto, en esta iteración.

14.1 Soporte para el análisis dinámico

Esta funcionalidad es de gran importancia para el análisis de malware, ya que permite conocer en un corto periodo de tiempo información como URL's o IP's a las que se conecta, ficheros a los que accede o modifica, claves de registro utilizadas, etc.

Toda esta información es muy útil para tener un primer vistazo de la funcionalidad del malware.

14.2 Conexión a aplicaciones de terceros

Puede ser útil conectarse con herramientas similares de terceros, con el fin de consultar la existencia de algún análisis ya realizado para ese fichero. De ser así, se descargaría dicho análisis, y se mostraría, indicando la fuente.

Esto permite tener una rápida comparación de los resultados de un análisis en diferentes plataformas.

14.3 Derivación de los IoC

Al tratarse de un malware, es interesante obtener todos los indicadores de compromiso de un fichero malicioso, con el fin de poder alimentar otras bases de datos que permitan reaccionar ante la aparición de esos IoC (indicadores de compromiso).

Esto permite que las organizaciones aprendan de incidentes anteriores, y malware de terceros, para adoptar una estrategia proactiva de prevención de malware.

14.4 Desarrollo de una API

Es buena idea poner a disposición de terceros una API que permita consultar los IoC derivados del análisis de los ficheros analizados por la aplicación.

Además, puede tratarse un API RESTful, que encapsule las funcionalidades de negocio. Con esto se consigue que se puedan desarrollar otras aplicaciones que utilicen las funcionalidades de ésta, de forma rápida y sencilla.

14.5 Análisis con diferentes motores

Durante el análisis dinámico, es interesante que se analice el malware con diferentes motores e antivirus, lo que permite tener una comparativa de las detecciones del fichero malicioso por parte de las casas de antivirus, además de su catalogación en la familia de malwares.

14.6 Modding de malware

Una vez se tiene un fichero que ha sido detectado como malware por algún antivirus, puede ser interesante moddear el fichero, para conocer cuál es la firma que ha detectado el antivirus.

Este proceso se puede hacer dividiendo el fichero en varios subficheros de un determinado tamaño, y pasando cada uno de ellos por el motor de antivirus. Se vuelve a repetir este proceso hasta que los subficheros generados son tan pequeños, que se puede asegurar que la firma del malware empieza en un determinado byte.

15. BIBLIOGRAFÍA

[1]. Astudillo, K. (2013): *Cómo hacer análisis dinámico de malware*. [HTML] Disponible en: http://blog.elixircorp.biz/como-hacer-un-analisis-de-malware-parte-3-end/

[2]. Astudillo, K. (2013): *Cómo hacer un análisis estático de malware*. [HTML] Disponible en: http://blog.elixircorp.biz/como-hacer-un-analisis-de-malware-parte-2/

[3]. Carrera, E. (2016): *Pefile* [HTML] Disponible en: https://github.com/erocarrera/pefile

[4]. Cheron, A. (2015): *Pydasm* [HTML] Disponible en: https://github.com/axcheron/pydasm

[5]. Mendoza García Leo, J. R. y Altamirano Guzmán G.N. (2015): *Sandbox: Análisis dinámico de* malware [PDF] Disponible en: http://www.seguridad.unam.mx/img/8_sandbox.pdf

[6]. Pietrek, M. (2010): *An In-Depth Look into the Win32 PE file format P1* [HTML] Disponible en: http://blog.sina.com.cn/s/blog_4e8be51f0100gp6n.html

[7]. The Swash (2011): *Formato Portable Executable Bajo Windows* [PDF] Disponible en: http://ns2.elhacker.net/timofonica/manus/Formato_de_ficheros%20ejecutables%20_Formato%20PE_.pdf

[8]. Unknown (2016). *Estilo APA* [HTML] Disponible en: https://es.wikipedia.org/w/index.php?title=Estilo_APA&oldid=91735433

[9]. Unknown (2009): *Análisis de Malware, Introducción y Contenido* [HTML] Disponible en: http://www.dragonjar.org/laboratorios-analisis-de-malware-introduccion-y-contenido.xhtml

[10]. Unknown (2016): *APK (formato)* [HTML] Disponible en: https://es.wikipedia.org/wiki/APK_(formato)