UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE INFORMÁTICA.
GRUPO DE SEGURIDAD EN LAS TECNOLOGÍAS DE LA INFORMACIÓN

PROYECTO FIN DE GRADO

ANÁLISIS DINÁMICO DE MALWARE EN ENTORNOS CONTROLADOS

[Resumen] *El objetivo de este proyecto es obtener una herramienta que sea capaz de analizar el malware de forma dinámica y automatizada en función de las características que pueden derivarse de análisis estáticos. El malware será identificado mediante servicios antivirus on-line y posteriormente se analizarán sus características. En base a esta información se creará un entorno de análisis vulnerable y se ejecutará, con el propósito de analizar su comportamiento y generar un informe en un formato estándar de enumeración de malware, que permita obtener suficiente información para identificarlo mediante reglas sintácticas.*

TUTOR: Guillermo Suárez de Tangil
AUTOR: Sergio Ortega Fernández

INDICE DE CONTENIDO

INDICE DE ILUSTRACIONES

INDICE DE TABLAS

GLOSARIO

- **Malware:** Es todo aquel tipo de software cuyo objetivo es realizar acciones no autorizadas por el usuario y sin su consentimiento, siendo en la mayoría de casos dañino.
- **Sandbox:** Hace referencia al aislamiento de sistemas. En este caso es posible ejecutar y analizar el malware en entornos controlados que permitan extraer información de su ejecución sin poner en peligro el resto de sistemas.
- **Asset:** Cada uno de los recursos con valor dentro de un sistema que debe considerarse durante el análisis de seguridad.
- **Exploit:** Fragmento de código desarrollado con el propósito de aprovechar una vulnerabilidad de un sistema o aplicación.
- **0-Day:** Denominación de exploits que aún no han sido publicados y por tanto no existen medidas de defensa.
- **Open Source:** Software distribuido y desarrollado libremente.

1. INTRODUCCIÓN Y OBJETIVOS

En este capítulo se realizará una introducción que tendrá el propósito de explicar los cambios que sufre la sociedad en la actualidad, la implicación de las tecnologías de la información en la misma y el impacto que representa la seguridad TIC para los sistemas de uso diario.

1.1. INTRODUCCIÓN

Durante la última década estamos asistiendo a una revolución tecnológica que ha cambiado la mentalidad de la sociedad respecto a los ordenadores e internet. La llegada de los s*martphones* en primer lugar, y ahora la expansión de las denominadas *tablets*, han propiciado tanto el aumento de la demanda de servicios informáticos como la movilidad mediante las conexiones inalámbricas. Además de esto, el perfil de usuario ha cambiado, siendo mucho más heterogéneo que hace unos años, motivado principalmente por la facilidad de acceso a la tecnología y las telecomunicaciones.

La expansión de la tecnología y la mejora en las telecomunicaciones se ha traducido en un uso cada vez mayor de servicios a través de internet (gestión de activos financieros, gestión de servicios domóticos, etc). Estos servicios deben considerarse como críticos debido a que, en su mayoría, utilizan datos de carácter privado o incluyen datos económicos, lo que los convierte en principal objetivo de actividades criminales.

La seguridad informática no siempre evoluciona al mismo tiempo que los sistemas, y a día de hoy podemos encontrar varios casos diarios relacionados con problemas en la seguridad de grandes empresas [1], [2]. La mayoría de estos problemas son derivados de fallos de seguridad en aplicaciones de terceros, como java o flash [3], por lo que en muchos casos el usuario final está desprotegido a pesar de utilizar las medidas de seguridad adecuadas.

Al mismo tiempo, el objetivo de los criminales está cambiando y en 2013 el número de programas maliciosos detectados destinado a sistemas operativos para móviles, principalmente Android, ha incrementado notablemente su número [4]. Estos usuarios no parecen estar tan concienciados sobre la seguridad en dispositivos móviles como lo están los usuarios de PC. Según revela un estudio del año 2013 sobre el uso de antivirus en móviles [5], cerca del 60% de los encuestados no utiliza ninguna medida de protección en su dispositivo portable.

A pesar de que la mayoría de usuarios de PC están concienciados de la necesidad de utilizar un antivirus, los últimos malware detectados están cambiando las técnicas de infección, haciéndolos difícilmente detectables y peligrosos debido a la utilización de vulnerabilidades de terceros, como está siendo el caso de java. Este hecho pone de manifiesto que se necesita algo más que un antivirus para poder frenar los ataques más modernos. Es importante conocer todos los posibles vectores de ataque antes de que sean utilizados de forma masiva, y para ello se debe mejorar y profundizar en los métodos actuales para analizar malware que, en muchos casos, resultan ser lentos e ineficaces.

1.2. PROBLEMAS DETECTADOS

Por estos motivos, el análisis de malware es fundamental para conocer el comportamiento de los ataques, identificar qué efecto tienen y cómo un usuario puede protegerse ante ellos.

Aunque existen muchas empresas dedicadas a ofrecer protección contra el malware, entre ellas *Kaspersky*, *Symantec* o la Española *Panda*, a día de hoy no hay un criterio unificado para identificar las muestras analizadas. Entre las causas de este problema podemos destacar dos factores:

- El elevado número de empresas que tienen intereses comerciales en elaborar un producto de seguridad.

- La gran cantidad de muestras que son recibidas y deben clasificarse al día, siendo en muchos casos variaciones de versiones existentes y conocidas de malware.

Estos factores hacen complicado estandarizar la identificación de malware y dificultan enormemente la tarea de analizarlo, ya que en muchas ocasiones no se dispone de suficiente información pública para proceder a un análisis dinámico controlado.

La falta de información previa a la ejecución puede provocar que el malware que está siendo analizado en un entorno dinámico no llegue a mostrar en ningún momento su carácter malicioso. Esto puede deberse a varios factores, entre ellos que el sistema no

tiene la vulnerabilidad específica que requiere para ejecutarse o que el malware esté utilizando técnicas de detección de máquinas virtuales, anulando su comportamiento en caso de detectar que está siendo analizado.

Dada la gran cantidad de malware que debe procesarse a diario, es imposible realizar un análisis dinámico sobre comportamiento y pautas para posteriormente elaborar una firma o un sistema de identificación de la muestra. Por ello, en muchos casos los antivirus solo se basan en análisis estáticos del código para generar identificadores, de ahora en adelante denominados firmas.

La realización de un análisis estático tiene dos consecuencias negativas:

- Se debe emplear mucho tiempo en analizar el código, lo cual puede resultar una tarea muy compleja. Esto se debe principalmente a la técnicas de cifrado y ofuscación que se emplean durante el diseño del *malware*.

- Que se desconozca el comportamiento del malware analizado. Por norma general no se realizan análisis manuales en profundidad, si no que se utilizan sistemas automáticos heurísticos que buscan patrones comunes para elaborar una firma genérica en el menor tiempo posible.

Una vez expuestos los problemas identificados del actual sistema de análisis de malware, se debe buscar una solución eficiente que permita un análisis más exhaustivo de los ficheros sospechosos, permitiendo recuperar los resultados de este

análisis en formatos estándar y trabajar con estos resultados con el propósito de poder identificar de forma rápida y con una alta tasa de éxito amenazas similares en base a su comportamiento.

1.3. OBJETIVO DEL PROYECTO

El objetivo del proyecto es desarrollar un sistema que, dado un archivo, sea capaz de identificar su comportamiento, generar una identificación y clasificarlo como malware en caso de cumplir una serie de condiciones.

Este proceso se realizará utilizando servicios de análisis on-line y realizando búsquedas de información conocida sobre la muestra analizada en distintas bases de datos. Esta búsqueda se centrará en la identificación de las plataformas afectadas, el software afectado y el identificador de vulnerabilidad asociado.

Posteriormente, se podrán desplegar distintos módulos de análisis, sobre diferentes configuraciones obtenidas de este análisis previo, que incluirán software de análisis de comportamientos, *sandboxing* o analizadores de conexiones. Estas herramientas permitirán obtener información sobre el comportamiento del malware y generar una serie de informes.

Una vez que se disponga de estos informes, se procederá a la normalización de la información, utilizando alguno de los estándares de enumeración y comportamiento de malware que se están desarrollando en la actualidad.

Por último, cuando se disponga de toda la información procesada y debidamente organizada, se generará de forma automática una firma que permita identificar el comportamiento dinámico del malware. También se podrán aplicar métodos heurísticos que permitan clasificar una muestra de software mediante técnicas de *machine learning* en función de si son *malware* o *goodware*.

Todo este proceso deberá ser realizado de forma modular, lo cual permitirá ejecutar los distintos módulos de forma independiente según las necesidades del usuario. Además, esto facilitará la posibilidad de añadir nuevos módulos en cualquier momento, de forma que el proyecto sirva como base para futuras extensiones y mejoras.

La representación estandarizada de la información obtenida fomentará la utilización de estos estándares. La motivación subyacente es facilitar la identificación, el tratamiento y la detección de amenazas mediante bases de datos abiertas, estandarizadas y usadas por los investigadores de forma que se garantice tanto la interoperabilidad como la disponibilidad de esta información.
Esto debería disminuir el tiempo de respuesta frente a incidente y permitirá desarrollar sistemas de identificación colaborativos.

1.4. ALCANCE DEL PROYECTO

Este proyecto ha obtenido un sistema completo y funcional, que abarca desde la recolección de datos hasta la posterior ejecución de módulos que permitan obtener información detallada y estructurada sobre las actividades sospechosas, todo ello de forma desatendida una vez iniciado el proceso.

Como parte de la plataforma se ha diseñado un sistema de generación de firmas automático y una base de datos de uso público donde se puedan consultar las firmas generadas. Cualquier investigador con un informe en formato estándar, generado tras un análisis dinámico, podrá comprobar si coincide o no con los resultados de algún malware analizado previamente por el sistema.

Para facilitar la obtención de muestras para plataformas Android se ha desarrollado un módulo que permite descargar las últimas novedades de aplicaciones para un conocido mercado alternativo de aplicaciones. La integración de este módulo permitirá que la plataforma se utilice como base para monitorizar nuevos comportamientos maliciosos de forma automatizada.

Con el propósito de resultar más intuitivo para los usuarios y poder publicar el servicio en distintas redes, se ha desarrollado una aplicación web que gestiona la subida y recuperación de análisis, de forma que resulte más sencillo consultar los resultados obtenidos por la plataforma.

El alcance del proyecto se basa en cuatro puntos fundamentales:

1. Recuperación, análisis y tratamiento de la información. Durante esta fase previa se pretende recoger toda la información posible sobre el malware identificado durante el análisis estático. Para ello se deberá crear una base de datos con fuentes de información existentes en internet, analizar el tipo de información que contiene cada una de estas fuentes e identificar cuál es la forma más apropiada de obtener información en estas páginas.

2. Procesamiento y normalización de la información recuperada durante la fase de análisis. Para ello se agregará toda la información encontrada, prestando especial atención a identificadores de vulnerabilidad encontrados, que ofrecen gran cantidad de información de forma estandarizada sobre el tipo y la configuración de los sistemas afectados por el malware.

3. Disponer de alguna tecnología o lenguaje que permita definir una serie de reglas y aplicarlas a un informe dinámico mediante técnicas de análisis sintáctico, de tal forma que se puedan identificar de forma flexible ciertos comportamientos.

4. Identificación de comportamientos. Mediante el uso la anterior tecnología poder generar firmas de manera automática en función de los atributos, características y comportamientos detectados durante el análisis dinámico.

2. ESTADO DEL ARTE

En esta sección se analizará el estado actual en relación a varios de los problemas detectados en la sección anterior, con el propósito de conocer sus limitaciones y cubrir las necesidades del proyecto.

2.1.1 ENUMERACIÓN Y TAXONOMIZACIÓN DE MALWARE

En la actualidad no existe un estándar extendido y completamente implantado a la hora de catalogar e identificar un malware, ni se dispone de una base de datos fiable y completa donde poder encontrar información en referencia a los mismos. Cada compañía de antivirus tiene su propia nomenclatura para identificar el malware, la cual en muchos casos no coincide con el resto de compañías. Esta fuente de información será la utilizada durante este proyecto, aunque tiene ciertas carencias que serán analizadas posteriormente.

Durante esta fase del proyecto tampoco se encontró ninguna base de datos que provea, en función de un identificador común, información sobre los sistemas afectados. Por tanto, resulta costoso elaborar un estudio previo que nos permita conocer los entornos a los que afecta un malware concreto, paso fundamental para poder realizar un análisis dinámico sobre los ficheros sospechosos.

La principal fuente de información son las empresas dedicadas a la protección contra malware. Algunas de ellas disponen de bases on-line donde consultar los resultados de sus antivirus, pero se ha detectado que en muchos casos esta información no es

completa o es inexacta, no ofreciendo los identificadores de vulnerabilidad ni las versiones exactas que se ven afectadas.

A pesar de ello, se utilizarán estas bases de datos para obtener toda la información posible que pueda ser usada para configurar los entornos de análisis, centrando la búsqueda en tres aspectos fundamentales: plataformas, software y vulnerabilidades identificadas.

Actualmente no existe una estandarización en los criterios de clasificación, aunque desde el año 2007 existe un proyecto impulsado por MITRE y conocido como "Making security measurable" [6]. Este proyecto a su vez agrupa varios proyectos que convergen con el propósito de mejorar el actual sistema de evaluación de la seguridad, como puede verse en el siguiente diagrama:

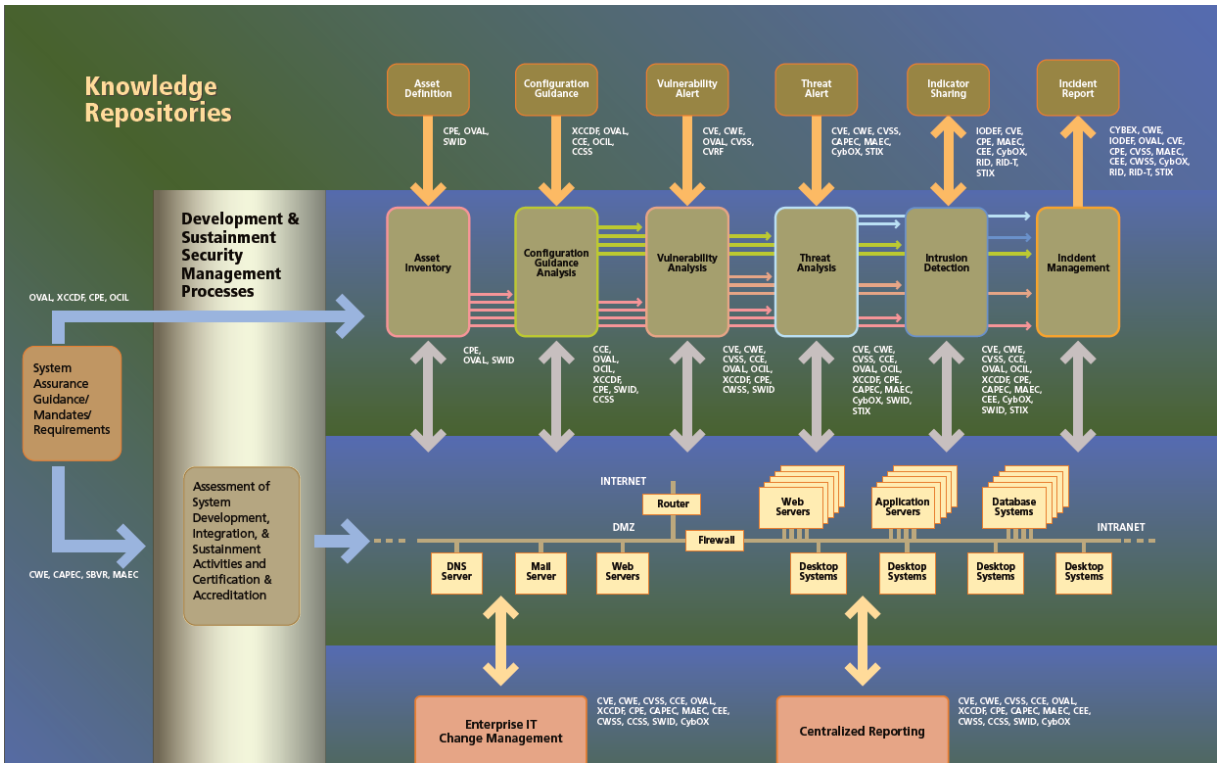

Ilustración 1. MITRE: Interoperabilidad de proyectos

Los más interesantes desde el punto de vista de este proyecto son:

- **CPE (Common platform enumeration)** [7]: Mediante un esquema estándar permite definir la configuración de los sistemas existentes en una red, de tal forma que puedan ser analizados posteriormente. En este caso es interesante encontrar los CPE relacionados con una vulnerabilidad para configurar las plataformas de evaluación donde realizar el análisis dinámico.

- **CVE (Common vulnerabilities and exposures)** [8]: Este identificador es asignado a las vulnerabilidades que son descubiertas, de tal forma que a través de páginas como la de NIST [9] se pueden encontrar las descripciones de cada vulnerabilidad y el CPE afectado.

- **CVSS (Common vulnerability score system)** [10]: Permite unificar el criterio de evaluación de forma que se pueda medir el riesgo de un CVE. Esto es importante para conocer el alcance y el posible riesgo objetivo del malware a evaluar.

- **MAEC (Malware attribute enumeration and characterization)** [11]: Este lenguaje permite describir las acciones que realiza el malware en un formato estándar y abierto, con el propósito de poder ser interpretado de forma estandarizada.

- **CAPEC (common attack pattern enumeration and classification)** [12]**:** Permite identificar patrones conocidos de ataques y asignarles un identificador, de forma que sea más sencillo clasificar el malware en función de sus identificadores.

Estos estándares usados de forma conjunta, por ejemplo en herramientas *SCAP*, podrán realizar análisis automáticos de los sistemas completos, diferenciales, evaluar los *assets*, conocer las vulnerabilidades y finalmente realizar un informe que incluya las amenazas, así como generar sistemas de actualización automáticos cuando se detecte una vulnerabilidad, mejorando la prevención. Es importante enfatizar la importancia de que iniciativas como esta cobren mayor prioridad en el día a día de los investigadores y empresas.

Al ser un lenguaje estándar toda esta información se podría enviar, distribuir y compartir de forma rápida. Esto aumentaría notablemente la cantidad de información disponible y facilitaría la difusión de información y análisis sobre nuevas amenazas.

Para poder gestionar toda la información referente a vulnerabilidades, análisis de malware y otra información de interés, a principios del año 2013 aparece la iniciativa MISP [13], una plataforma de intercambio de información que podría utilizarse para gestionar toda la información recogida a través de diversos análisis, aunque a fecha de este trabajo aún no permite su integración automática con formatos estandarizados.

2.2.2. ANÁLISIS DE FICHEROS

Con el propósito de analizar ficheros sospechosos, existen varios servicios como *Anubis* [14], *Comodo* [15] o *Eureka* [16]. Estas páginas permiten a un usuario subir el fichero y realizan un análisis de comportamiento que incluya las entradas en el registro que modifica, los ficheros que crea o mueve, las peticiones web, etc.

Este proceso se conoce como análisis dinámico de ficheros y la mayoría se basan en implementaciones similares a *Cuckoo Sandbox* [17], herramienta de software libre que proporciona los recursos necesarios para la gestión de los análisis dentro de entornos controlados. La principal ventaja de este proyecto es que al ser de código libre puede ser modificado para adaptarlo a las necesidades del sistema, por lo que poco a poco va teniendo una importante comunidad a sus espaldas, que trabaja día a día para mejorar las capacidades que ofrece. Otro factor decisivo a la hora de elegir *Cuckoo* es que permite emitir los informes en un formato estándar como MAEC, por lo que se adaptaría a las necesidades del proyecto.

Con el propósito de obtener de forma sencilla una evaluación de un software y conocer si se trata o no de malware, se puede recurrir a VirusTotal, el cual es un metabuscador de antivirus. El usuario selecciona el archivo sospechoso y este es examinado por cerca de cincuenta antivirus de diferentes compañías. Esto permite obtener identificaciones positivas, las cuales pueden ser usadas para extraer la información que se necesita.

2.2.3. IDENTIFICACIÓN DE MALWARE

Aunque la información generada durante la fase de análisis es esencial para comprender el funcionamiento del software sometido a estudio, por sí solo no permite clasificar el software ni evaluarlo, es necesario analizar los resultados obtenidos para identificar posibles patrones de ataque y determinar si tiene un carácter malicioso.

Con el fin de poder aplicar técnicas de *machine learning* sobre los informes obtenidos tras el análisis dinámico se han investigado diversas técnicas, aunque destacarían dos acercamientos distintos, [18] y [19]. Este último resulta especialmente relevante dada su sencillez, ya que en función de la asignación de valores a las distintas operaciones que realiza, se puede obtener un sistema objetivo que mida el riesgo de las operaciones y determine si se trata o no de malware. Esto permitiría añadir un nuevo módulo al proyecto que, independientemente de si se conoce si es malware o no, siempre que se pueda realizar el análisis dinámico y se tenga la información normalizada se podría procesar y clasificar automáticamente en función de si es una amenaza o no.

Por otra parte, resulta igual de interesante disponer de una serie de reglas sintácticas que permitan generar de forma flexible una firma del malware, independientemente de si estos datos se han obtenido de un análisis dinámico o estático. Para este caso existe un proyecto Español conocido como *Yara* [20] que cubriría esta necesidad. A pesar de la relativa juventud del proyecto cada vez se está extendiendo más y podría ser otra vía por la cual tener un estándar de identificación de malware. Al ser otro

proyecto de código libre podría modificarse en caso de necesidad, con el objetivo de extender alguna de sus funcionalidades y adecuarla al objetivo del proyecto.

Además de este lenguaje existe ClamAV, un antivirus de software libre basado en firmas. Existen multitud de *scripts* que tienen la capacidad de convertir una regla Yara a formato ClamAV y viceversa, por lo cual también es una herramienta interesante a utilizar durante el proyecto, dada la mayor eficiencia en lo relativo a tiempos y la facilidad de conversión de firmas.

En el momento de realizar este proyecto si bien existen todas estas herramientas, ninguna cubre el objetivo principal, crear un sistema que dada una serie de *inputs* sea capaz de analizarlos a distintos niveles, crear los entornos de pruebas, evaluarlos y finalmente obtener unos resultados en un lenguaje estándar de enumeración de malware que permita su posterior identificación y clasificación.

3. ANÁLISIS

Una vez que han sido identificadas las carencias, se ha fijado un objetivo y se conoce la base y el estado actual en el que se encuentra el problema, se debe proceder a la fase de análisis del sistema. Durante esta fase se ha recopilado información sobre el problema, los requisitos asociados al desarrollo de la solución y el entorno tecnológico de la propia solución.

3.1. DESCRIPCIÓN GENERAL

El proyecto tiene el propósito de crear de un sistema automático que, en función de un fichero, tenga la capacidad de obtener la suficiente información previa para ejecutarlo en un entorno de análisis dinámico y obtener toda la información posible mediante los informes que generen las herramientas de análisis.

Para ello el sistema deberá subdividirse en varios subsistemas con funciones independientes, pero que en conjunto aporten la información necesaria a todo el proceso y que finalmente genere una salida normalizada con información sobre el comportamiento del fichero analizado, de tal forma que pueda ser utilizada para generar identificadores o clasificar el software.

3.2. CASOS DE USO

El propósito de esta sección es representar aquellas interacciones que puede tener un usuario con la aplicación, así como establecer los distintos pasos que debe seguir la plataforma dependiendo de la acción solicitada.

Se considera como actor a cada una de las personas que pueden interactuar con la aplicación. Debido a que el propósito del proyecto es disponer de una plataforma abierta y colaborativa no existen roles definidos y por tanto se simplifica en un único actor.

CU-01: Generar firma	
Actores	Usuario
Descripción	Un usuario solicita generar una firma identificativa.
Precondiciones	El usuario debe tener un archivo en formato válido para realizar el proceso de análisis completo. Alternativamente, si dispone de un informe en formato *MAEC* o *XML*, para el caso de los *.apk*, podrá generar la firma sin realizar el proceso previo de búsqueda y análisis.
Post-condiciones	El módulo debe registrar la creación de la firma y almacenarla.
Escenario Principal	1. El usuario desea analizar un archivo ejecutable. 2. El usuario inicia el proceso de análisis mediante el módulo de análisis previo. 3. En base a los resultados se realiza un análisis dinámico en una *sandbox* que permita recuperar el informe. 4. Con este informe se genera la firma identificativa.
Escenario Alternativo	1. El usuario ya dispone de un informe en uno de los formatos aceptados. 2. Se utiliza este informe como input para el módulo de generación de firma y se genera la firma.

Tabla 1. Caso de uso CU-01

CU-02: Analizar fichero	
Actores	Usuario.
Descripción	Un usuario solicita obtener información sobre un fichero para Windows o Android.
Precondiciones	El usuario debe tener un archivo en formato válido para realizar el proceso de análisis.
Post-condiciones	El módulo debe registrar la información obtenida del análisis en un servicio on-line de antivirus.
Escenario Principal	1. El usuario desea analizar un archivo ejecutable. 2. El usuario inicia el proceso de análisis mediante el módulo de análisis previo.

Tabla 2. Caso de uso CU-02

CU-02: Obtener información sobre la plataforma afectada	
Actores	Usuario
Descripción	Un usuario desea obtener información sobre si un fichero para Windows es catalogado como malware y la configuración de sistema a la que afecta.
Precondiciones	El usuario debe tener un archivo en formato válido.
Post-condiciones	El módulo debe registrar la información obtenida al realizar la operación.
Escenario Principal	1. El usuario desea analizar un archivo ejecutable.

	2. El usuario inicia el proceso de análisis mediante el módulo de análisis previo. 3. Si existe información asociada al malware se muestra al usuario.

Tabla 3. Caso de uso CU-03

CU-04: Generar informe en formato hexadecimal	
Actores	Usuario
Descripción	Un usuario solicita generar un informe en formato hexadecimal.
Precondiciones	El usuario debe tener un archivo en formato válido para realizar el proceso de análisis completo. Alternativamente, si dispone de un informe en formato *MAEC*, podrá generar el fichero sin el proceso completo.
Post-condiciones	El módulo debe registrar la creación del fichero y almacenarlo.
Escenario Principal	1. El usuario desea analizar un archivo ejecutable. 2. El usuario inicia el proceso de análisis mediante el módulo de análisis previo. 3. En base a los resultados se realiza un análisis dinámico en una sandbox que permita recuperar el informe. 4. Con este informe se genera el fichero hexadecimal.

Tabla 4. Caso de uso CU-04

CU-05: Obtener *apks*	
Actores	Usuario
Descripción	Un usuario desea descargar las últimas aplicaciones añadidas en un mercado alternativo de aplicaciones para Android.
Precondiciones	-
Post-condiciones	Se deben almacenar todas las aplicaciones descargadas en fomato *.apk* y comprobar si el hash coincide para asegurar la integridad.
Escenario Principal	1. El usuario inicia el proceso de descarga mediante el módulo de búsqueda de aplicaciones. 2. Se descargan las aplicaciones y su hash, con el propósito de validar la integridad de la descarga. 3. Se almacenan en formato *.apk* si se valida su integridad.

Tabla 5. Caso de uso CU-05

3.3. ANÁLISIS DE REQUISITOS

Un requisito es toda aquella necesidad documentada sobre el contenido, forma o funcionalidad de un producto o servicio.

A continuación se mostrarán los requisitos de usuario y software obtenidos durante la etapa de análisis. La fuente de estos requisitos principalmente será el tutor, y han sido obtenidos a lo largo de diversas entrevistas durante el proceso de diseño de la plataforma.

3.3.1. REQUISITOS DE USUARIO

A continuación se detallan los parámetros utilizados en las tablas para describir los requisitos de este apartado:

- **Identificador:** El identificador determina de forma unívoca cada uno de los requisitos siguiendo la estructura XX-YY, siendo XX el tipo e YY el número único que lo identifica.

- **Descripción:** Breve descripción de la funcionalidad o restricción de dicho requisito.

- **Necesidad:** Indica si el requisito tiene que ser implementado de forma obligatoria u opcional. Los valores que puede tomar este campo son:
  - Esencial: El requisito debe ser implementado.
  - Opcional: El requisito no debe ser implementado de forma obligatoria pero aporta una funcionalidad importante al proyecto.

- **Prioridad:** Indica la importancia del requisito en el proceso de diseño e implementación. Los valores que puede tomar este campo son:
  - Alta: El requisito debe ser añadido al sistema en primer lugar.
  - Media: El requisito debe ser añadido al sistema tras haber acabado con los requisitos de prioridad alta.
  - Baja: El requisito debe ser añadido al sistema tras haber realizado la implementación de los anteriores.

- **Origen:** Origen del requisito, puede ser por parte del tutor o por el alumno.

- **Tipo:** La clase del requisito de usuario.

Identificador	RC-01		
Descripción	La plataforma debe tener la capacidad de realizar análisis on-line sobre fichero sospechosos.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Capacidad

Tabla 6. Requisito RC-01

Identificador	RC-02		
Descripción	La plataforma debe ser capaz de obtener información sobre los análisis on-line y almacenarla.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Capacidad

Tabla 7. Requisito RC-02

Identificador	RC-05		
Descripción	La plataforma debe ser capaz de buscar información relevante sobre los resultados del análisis realizado: palabras claves, CVE´s, CPE´s, etc…		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Capacidad

Tabla 8. Requisito RC-05

Identificador	RC-06		
Descripción	La plataforma debe integrar módulos de análisis con los que realizar análisis dinámicos de software.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Capacidad

Tabla 9. Requisito RC-06

Identificador	RC-07		
Descripción	La información obtenida por la plataforma deberá ser almacenada para su posterior consulta.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Media	**Tipo**	Capacidad

Tabla 10. Requisito RC-07

Identificador	RC-08		
Descripción	La plataforma debe ser capaz de generar una firma identificativa por cada elemento software analizado de forma dinámica.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Capacidad

Tabla 11. Requisito RC-08

Identificador	RC-09		
Descripción	La plataforma debe ser capaz de generar un informe en formato hexadecimal por cada elemento software analizado de forma dinámica.		
Necesidad	Opcional	**Origen**	Alumno
Prioridad	Baja	**Tipo**	Capacidad

Tabla 12. Requisito RC-09

Identificador	RC-10		
Descripción	La plataforma podrá analizar todo tipo de ficheros ejecutables, las extensiones más comunes (.pdf, .swf, etc) y aplicaciones para Android (.apk).		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Capacidad

Tabla 13. Requisito RC-10

Identificador	RC-11		
Descripción	La plataforma dispondrá de un módulo que permita descargar las últimas aplicaciones Android de mercados alternativos.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Alta	**Tipo**	Capacidad

Tabla 14. Requisito RC-10

Identificador	RC-12		
Descripción	La plataforma debe ser capaz de generar toda la información en formatos estándar.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Media	**Tipo**	Capacidad

Tabla 15. Requisito RC-12

Identificador	RC-13		
Descripción	La plataforma debe ser capaz de funcionar como un sistema automatizado e independiente con el propósito de poder utilizarse como monitorización continua.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Media	**Tipo**	Restricción

Tabla 16. Requisito RC-13

Identificador	RC-14		
Descripción	La plataforma debe ser capaz de gestionar y mostrar la información contenida en la base de datos mediante una interfaz web.		
Necesidad	Opcional	**Origen**	Alumno
Prioridad	Baja	**Tipo**	Capacidad

Tabla 17. Requisito RC-14

Identificador	RC-15		
Descripción	La plataforma debe ser capaz de intercambiar información con otras plataformas o gestores de información para incorporar o enviar nuevas identificaciones.		
Necesidad	Opcional	**Origen**	Alumno
Prioridad	Baja	**Tipo**	Capacidad

Tabla 18. Requisito RC-15

3.3.2. REQUISITOS SOFTWARE

Este tipo de requisitos se obtiene tras el análisis de los requisitos de usuario y atiende a la descripción de las necesidades detectadas por los requisitos de usuario.

Se clasifican en funcionales y no funcionales, dependiendo de si se refieren a algún proceso que debe realizar la aplicación o por el contrario atiende a características de diseño.

3.3.2.1. REQUISITOS FUNCIONALES

Identificador	RSF-01		
Descripción	La plataforma debe ser capaz de subir un fichero para su análisis en un servicio de antivirus on-line.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Funcional

Tabla 19. Requisito RSF-01

Identificador	RSF-02		
Descripción	La plataforma debe ser capaz de recuperar el resultado del análisis en un servicio antivirus on-line.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Funcional

Tabla 20. Requisito RSF-02

Identificador	RSF-03		
Descripción	La plataforma debe ser capaz de almacenar el resultado del análisis en un servicio on-line en una base de datos.		
Necesidad	Opcional	**Origen**	Alumno
Prioridad	Media	**Tipo**	Funcional

Tabla 21. Requisito RSF-03

Identificador	RSF-04		
Descripción	La plataforma debe ser capaz de enviar peticiones a distintas bases de datos con los resultados obtenidos durante el análisis en un servicio antivirus on-line.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Funcional

Tabla 22. Requisito RSF-04

Identificador	RSF-05

Descripción	La plataforma debe ser capaz de buscar en los resultados de cada petición por palabras clave: Software, Sistemas operativos y CVE.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Funcional

Tabla 23. Requisito RSF-05

Identificador	RSF-06		
Descripción	La plataforma debe ser capaz de agregar toda la información obtenida durante el proceso de búsqueda y normalizarla.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Funcional

Tabla 24. Requisito RSF-06

Identificador	RSF-07		
Descripción	La plataforma debe ser capaz de buscar el CPE asociado a un CVE encontrado.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Funcional

Tabla 25. Requisito RSF-07

Identificador	RSF-08		
Descripción	La plataforma debe generar los informes de análisis en *sandboxes* en un formato estándar de enumeración de comportamiento de malware.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Funcional

Tabla 26. Requisito RSF-08

Identificador	RSF-09		
Descripción	La regla identificativa generada deberá permitir como entrada tanto archivos ejecutables como informes dinámicos en formato estándar.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Funcional

Tabla 27. Requisito RSF-09

Identificador	RSF-10

Descripción	La plataforma debe generar una firma identificativa automáticamente dando un informe en formato MAEC.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Funcional

Tabla 28. Requisito RSF-10

Identificador	RSF-11		
Descripción	El módulo de análisis debe generar un archivo en formato CVE dado un informe en formato estándar donde se detallarán las funciones y el número de veces que son llamadas durante el análisis del software.		
Necesidad	Opcional	**Origen**	Alumno
Prioridad	Baja	**Tipo**	Funcional

Tabla 29. Requisito RSF-11

Identificador	RSF-12		
Descripción	El módulo de análisis debe generar un archivo CVE dado un informe en formato estándar donde se detallarán todos los valores asociados a cada una de las funciones usadas.		
Necesidad	Opcional	**Origen**	Alumno
Prioridad	Baja	**Tipo**	Funcional

Tabla 30. Requisito RSF-12

Identificador	RSF-13		
Descripción	El módulo de análisis debe generar un archivo hexadecimal que pueda ser usado para la clasificación automática mediante técnicas de *machine learning*.		
Necesidad	Opcional	**Origen**	Alumno
Prioridad	Baja	**Tipo**	Funcional

Tabla 31. Requisito RSF-13

Identificador	RSF-14		
Descripción	La plataforma debe poder generar informes de aplicaciones para Android (.apk).		
Necesidad	Opcional	**Origen**	Alumno
Prioridad	Alta	**Tipo**	Funcional

Tabla 32. Requisito RSF-14

Identificador	RSF-15

Descripción	El módulo de análisis debe poder convertir el informe de un *.apk* a un formato estándar de enumeración de malware.		
Necesidad	Opcional	**Origen**	Alumno
Prioridad	Baja	**Tipo**	Funcional

Tabla 33. Requisito RSF-15

Identificador	RSF-16		
Descripción	El módulo de generación de firma debe poder generar firmas válidas con un informe de *.apk* en formato no estándar.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Alta	**Tipo**	Funcional

Tabla 34. Requisito RSF-16

Identificador	RSF-17		
Descripción	La plataforma debe permitir automatizar la recogida de muestras de aplicaciones Android en mercados alternativos.		
Necesidad	Opcional	**Origen**	Tutor/Alumno
Prioridad	Baja	**Tipo**	Funcional

Tabla 35. Requisito RSF-17

Identificador	RSF-18		
Descripción	La plataforma debe tener la capacidad de lanzar aplicaciones y scripts en lenguajes como *Shell script*, *Perl*, *Python* o *Ruby*.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Alta	**Tipo**	Funcional

Tabla 36. Requisito RSF-18

3.3.2.2. REQUISITOS NO FUNCIONALES

Identificador	RSNF-01		
Descripción	La plataforma deberá ser modular para facilitar la ampliación de sus funcionalidades.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Alta	**Tipo**	Escalabilidad

Tabla 37. Requisito RSNF-01

Identificador	RSNF-02

Descripción	La plataforma deberá estar implementada en Ruby.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Alta	**Tipo**	Interoperabilidad

Tabla 38. Requisito RSNF-02

Identificador	RSNF-03		
Descripción	La plataforma utilizará *Nokogiri* para *parsear* los resultados XML, HTML y JSON obtenidos en cada módulo.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Alta	**Tipo**	Escalabilidad

Tabla 39. Requisito RSNF-03

Identificador	RSNF-04		
Descripción	La plataforma utilizará *REST Client* para gestionar las peticiones POST y GET a las páginas y bases de datos.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Alta	**Tipo**	Escalabilidad

Tabla 40. Requisito RSNF-04

Identificador	RSNF-05		
Descripción	Los módulos ofrecerán distintos modos de operación en función de la entrada del usuario.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Alta	**Tipo**	Interfaz

Tabla 41. Requisito RSNF-05

Identificador	RSNF-06		
Descripción	Los módulos mostrarán una ayuda que incluya todas las opciones que pueden utilizar los usuarios.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Alta	**Tipo**	Escalabilidad

Tabla 42. Requisito RSNF-06

Identificador	RSNF-07		
Descripción	Cada operación realizada debe tener control de errores para garantizar que puede ser integrada en un sistema automático.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Alta	**Tipo**	Seguridad

Tabla 43. Requisito RSNF-07

Identificador	RSNF-08		
Descripción	Los algoritmos utilizados deben ser eficientes para garantizar el buen aprovechamiento de los recursos del sistema.		
Necesidad	Opcional	**Origen**	Alumno
Prioridad	Media	**Tipo**	Rendimiento

Tabla 44. Requisito RSNF-08

Identificador	RSNF-09		
Descripción	La información se mostrará en formato estándar siempre que sea posible.		
Necesidad	Esencial	**Origen**	Alumno
Prioridad	Alta	**Tipo**	Información

Tabla 45. Requisito RSNF-09

Identificador	RSNF-10		
Descripción	Cada módulo mostrará información al usuario sobre el estado de la operación.		
Necesidad	Opcional	**Origen**	Alumno
Prioridad	Baja	**Tipo**	Información

Tabla 46. Requisito RSNF-10

Identificador	RSNF-11		
Descripción	Para la interfaz web se utilizará un servidor *ligero* en python.		
Necesidad	Opcional	**Origen**	Alumno
Prioridad	Baja	**Tipo**	Rendimiento

Tabla 47. Requisito RSNF-11

4. Identificador	RSNF-12		
Descripción	La aplicación debe ejecutarse en entornos UNIX.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Restricción

Tabla 48. Requisito RSNF-12

Identificador	RSNF-13		
Descripción	La aplicación debe ser capaz de poder combinarse a otros módulos programados en Shell script.		
Necesidad	Esencial	**Origen**	Tutor
Prioridad	Alta	**Tipo**	Restricción

Tabla 49. Requisito RSNF-13

4. DISEÑO

En este capítulo quedarán detallados todos los aspectos de diseño una vez obtenidos los requisitos de software. En primer lugar se hará una descripción general del sistema, que contará con un diagrama y una breve explicación de los subsistemas y posteriormente se detallará el diseño de cada uno de ellos.

4.1. DESCRIPCIÓN GENERAL DEL SISTEMA

Dado que uno de los requisitos es la modularidad, se han definido tres subsistemas con características independientes que faciliten la reutilización y permitan realizar cada una de las tareas definidas durante la fase de análisis:

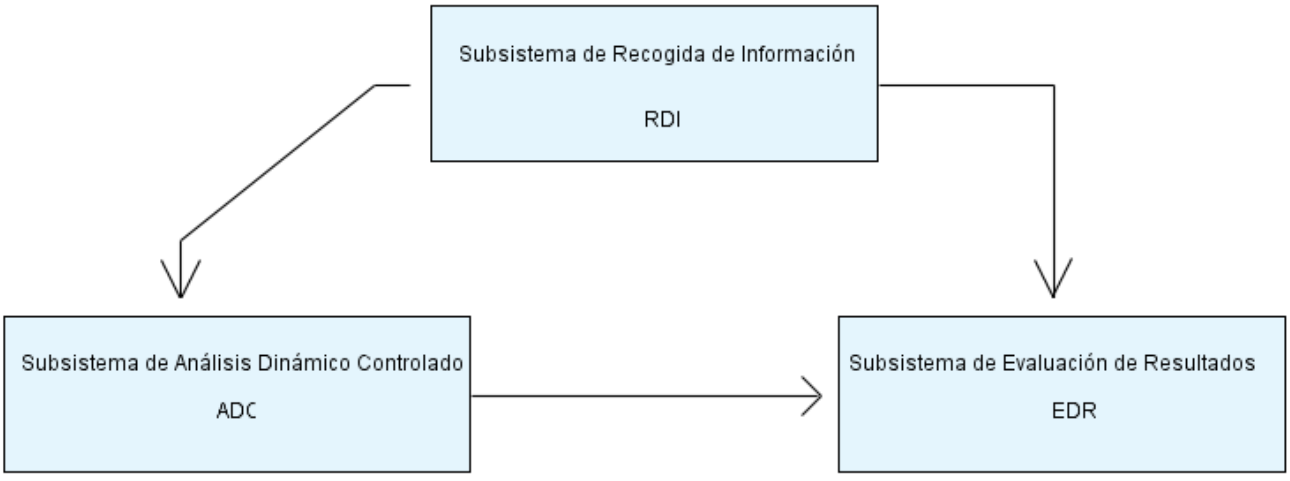

Ilustración 2. Diagrama de arquitectura

- **Subsistema de recogida de información (RDI):** Este subsistema contendrá los módulos relacionados con las tareas previas de análisis on-line, búsqueda de información, gestión de la plataforma y normalización de la información encontrada.

  El propósito de este subsistema es obtener la mayor cantidad de información e identificadores de malware sobre un fichero analizado. Además debe contener módulos para la automatización de tareas y la publicación de información.

- **Subsistema de análisis dinámico controlado (ADC):** Este subsistema contendrá los módulos de análisis dinámico que permitan realizar el análisis del fichero solicitado.

  Para la configuración de las máquinas del sistema de análisis se utilizará la salida obtenida de la ejecución del subsistema anterior. De esta forma, se pretende garantizar que los análisis se realicen en entornos vulnerables con el propósito de obtener toda la información posible derivada de las ejecuciones.

- **Subsistema de evaluación de resultados (EDR):** Este subsistema contendrá los módulos de evaluación de resultados.

  La entrada de los módulos de este subsistema serán los informes generados durante la ejecución del subsistema ADC. El propósito de este subsistema podrá variar en función de las necesidades, con módulos para la identificación de comportamientos mediante firmas o la clasificación automática con técnicas de inteligencia artificial.

A continuación se detallará el diseño de cada uno de estos subsistemas, definiendo las entradas y salidas que tendrán cada uno de los módulos dentro del subsistema, así como a la interacción entre los mismos.

4.2. SUBSISTEMA I: RECOGIDA DE INFORMACIÓN

4.2.1. DESCRIPCIÓN DETALLADA DEL SUBSISTEMA I: MÓDULO DE ANÁLISIS Y BÚSQUEDA DE INFORMACIÓN

El propósito de este subsistema es identificar toda la información conocida asociada a un software analizado. Para ello se utilizará cualquiera de los servicios on-line de análisis de ficheros sospechosos que existen en la actualidad, con el propósito de obtener resultados identificativos en aquellos casos donde el análisis resulte positivo.

Debido a la falta de una base de datos común y estandarizada basada en patrones de ataque, en muchos casos encontrar información sobre un software catalogado como malware es una tarea compleja y con escasos resultados. Además, dependiendo del tipo de malware, no tiene por qué hacer uso de vulnerabilidades, por lo que es imposible obtener información más allá del tipo de sistema operativo que se ve afectado.

Independientemente de este hecho, a la hora de elaborar este sistema se ha partido de la base de que cualquier información, por mínima que sea, puede aportar valor al posterior análisis. Por ello se ha puesto especial énfasis en el diseño funcional de este subsistema.

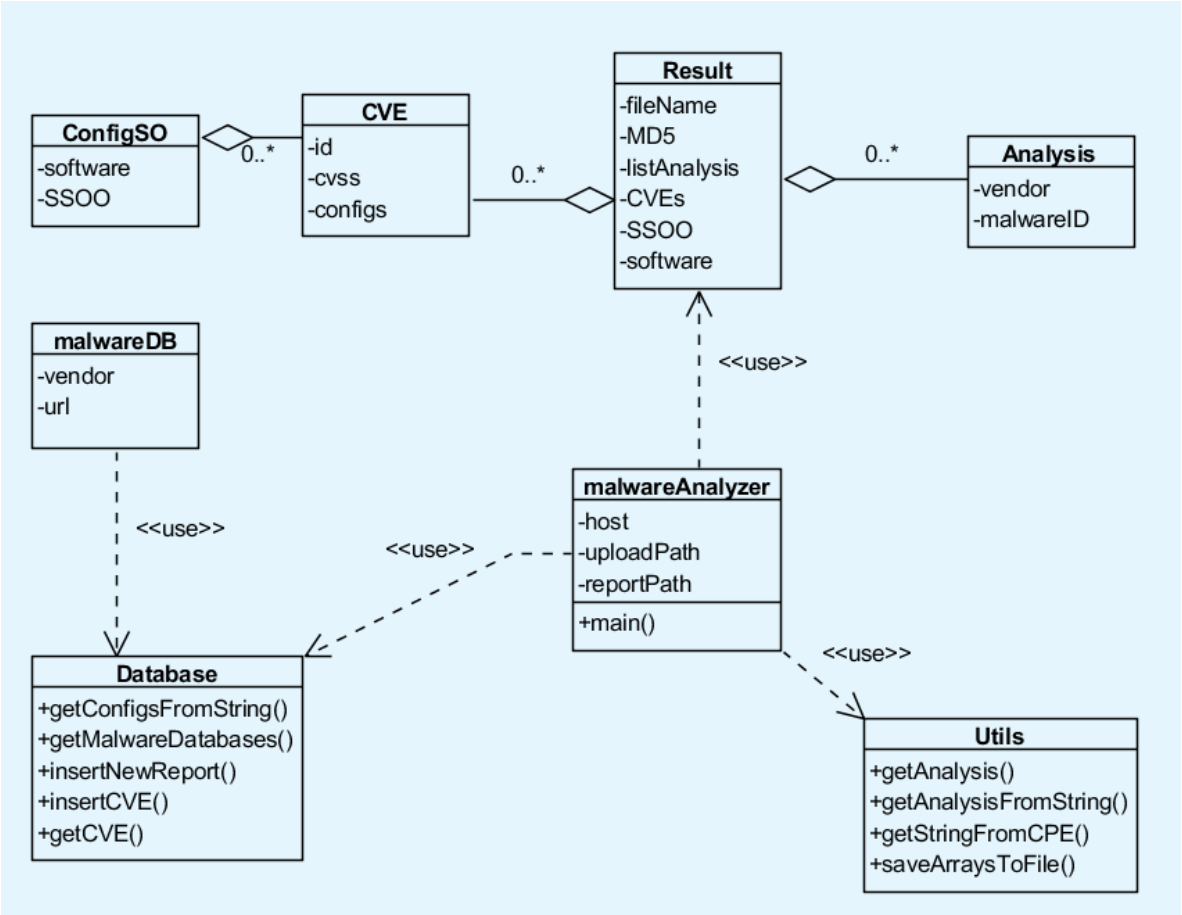

Ilustración 3. Diseño de clases: Subsistema I

Su diseño pretende garantizar al máximo la modularidad, con el objetivo de ampliar sus funcionalidades de forma sencilla. Deberán existir una serie de interpretadores sintácticos que, en función del informe generado por el servicio de análisis on-line, realicen consultas a ciertas bases de datos de malware. En las respuestas de estas peticiones se buscarán patrones concretos mediante expresiones regulares, con el objetivo de identificar información sensible.

A la hora de realizar esta búsqueda y por requerimientos críticos obtenidos durante la fase de análisis, se buscan principalmente tres tipos de datos: sistemas operativos, software y CVE´s afectados.

El mejor escenario es obtener un resultado donde se hayan encontrado uno o varios CVE´s. Esto se debe a que existe una base de datos del NIST donde se pueden realizar búsquedas para estos identificadores. De esta forma se puede obtener en formato CPE el software y/o plataformas afectadas. A pesar de esto, los estudios previos demuestran que estos identificadores no se encuentran en la mayoría de análisis, por lo que se requiere otro sistema de búsqueda de propósito más general.

Por este motivo se ha optado por añadir listas genéricas donde puedan incluirse nombres de sistemas operativos y software para que, en caso de no encontrar ningún CVE, se realicen búsquedas sobre esos términos. Con este tipo de solución es importante disponer de detallados listados de palabras claves con los que identificar la mayor cantidad de información posible, evitando generar falsos resultados durante las búsquedas.

Esta información será almacenada en una base de datos donde se guardarán tanto los resultados obtenidos tras realizar el análisis on-line como los CVE, sus CPE, las plataformas, el software encontrado y las bases de datos que se vayan añadiendo a la aplicación. El propósito de disponer de una base de datos tiene dos objetivos:

- Aligerar la carga de peticiones a páginas web. Si ya se ha analizado un fichero con el mismo *hash* se recupera el resultado anterior y no se genera una nueva búsqueda para evitar saturar el servicio.

- Si se dispone de una aplicación web, se puede consultar la base de datos y obtener información sobre los distintos análisis realizados de una forma mucho más intuitiva para un usuario.

Un primer diseño del subsistema trataba de obtener resultados de cada página de forma independiente, analizando sus respuestas y devolviendo toda la información posible. Este procedimiento demostró no ser eficaz y tuvo que ser descartado ya que, aún dentro de las mismas páginas, la información ofrecida entre un análisis y otro se diferenciaba mucho, por lo que la información que se obtenía no siempre era la correcta. Por ello finalmente se adoptará un diseño más general siguiendo estos pasos:

1. Identificar las páginas que contienen información sobre malware y que son utilizadas por el servicio de análisis on-line.

2. Si se obtienen resultados referentes a estas páginas, solicitar la información siempre y cuando se pueda acceder a ella mediante una petición web y no sea necesaria la intervención directa de un usuario.

3. Una vez obtenida la respuesta, analizar la página e identificar en que secciones se encuentra la información útil. A continuación se filtrará, con el propósito de

minimizar en la medida posible el *ruido* y los resultados incorrectos que pueda aportar el resto de la página.

4. Convertir la información de esas secciones a texto y, mediante un analizador genérico, buscar patrones según las listas definidas y personalizables para identificar tanto el software como las plataformas y los CVE que aparezcan mediante una expresión regular.

Finalizado este proceso, se deberá comprobar si ha sido posible identificar uno o más CVE´s. En caso afirmativo, se comprueba en la base de datos si existe información asociada a los mismos y se recupera. Si no existe, se envía una consulta a la página del NIST, se obtienen los resultados y se almacenan en la base de datos para futuras peticiones.

Debido a la falta de estandarización en los resultados resulta complicado obtener información concreta sobre gran parte de las amenazas, sobre todo si son variantes. Durante los análisis previos a la implementación se han identificado casos en los que ni en la propia base de datos de la compañía del antivirus podía encontrarse información sobre el malware identificado.

Por ello queda patente que el modelo de identificación de malware debe cambiar y añadir la identificación de patrones, de forma que no importe tanto la identificación o la versión, si no el disponer de un catálogo de comportamientos maliciosos que pueden ser detectados durante el análisis dinámico.

La principal ventaja de este modelo de diseño es que resulta muy sencillo añadir nuevos patrones de búsqueda, así como añadir más información a las listas utilizadas para buscar software y sistemas operativos en la base de datos.

4.2.2. DESCRIPCIÓN DETALLADA DEL SUBSISTEMA I: MODULO DE DESCARGA DE APLICACIONES MÓVILES

Con el propósito de facilitar la ejecución de las pruebas y disponer de una herramienta de monitorización de aplicaciones móviles publicadas en mercados alternativos, se ha realizado el diseño de un módulo que permita la descarga automatizada de las aplicaciones más recientes y populares.

El objetivo es que un investigador pueda fijar un periodo de ejecución, de forma que cada vez que se ejecute descargue las nuevas aplicaciones que estén disponibles. Si se configura con este propósito la plataforma, estas aplicaciones podrán ser enviadas a la plataforma y generar un informe que permita obtener una firma identificativa.
Gracias a este módulo la plataforma además podría ser utilizada como un sistema preventivo de alerta temprana de nuevos malware para móviles distribuidos mediante estos mercados alternativos.

4.2.3. DESCRIPCIÓN DETALLADA DEL SUBSISTEMA I: MODULO DE DESCARGA DE APLICACIONES MÓVILES

La plataforma ha sido diseñada para trabajar desde línea de comandos, pero debido a las limitaciones y la complejidad que esto añade a la hora de ser utilizada se ha

diseñado una interfaz web. El objetivo de esta interfaz es ofrecer al usuario de un sistema básico donde pueda seleccionar ficheros para su análisis y pueda visualizar los resultados de todos los análisis con un rápido acceso a los ficheros generados.

4.2.4. COMPONENTES DEL SUBSISTEMA I

Identificador		SUBI-01	**Nombre**	MalwareAnalyzer
Propósito		Obtener información sobre plataformas y software afectados por un fichero analizado y catalogado como malware en un servicio de análisis on-line.		
Descripción		Mediante línea de comandos un usuario tendrá la posibilidad de seleccionar un fichero que debe ser analizado. Esto iniciará un proceso en el cual el fichero será subido a un servicio de análisis on-line y se recuperará un informe. Si el fichero ha sido catalogado como malware, este componente buscará cuales de los resultados pueden ser utilizados para obtener información y realizará una serie de peticiones contra distintas bases de datos relacionadas con el identificador. La información recibida será tratada y normalizada, almacenándola en una base de datos. En caso de que se encuentre algún identificador CVE, este será buscado en la página del NIST con el propósito de encontrar el CPE asociado. Todas las búsquedas se realizarán en primer lugar en la base de datos de la plataforma con el propósito de evitar la sobrecarga de peticiones contra páginas externas.		
Datos	**Entrada**	Fichero (.exe, .apk, .pdf, etc…).		
	Salida	Cadenas de texto normalizadas con información sobre la plataforma, el software y los posibles CVE identificadores.		

Tabla 50. Componente SUBI-01

Identificador	SUBI-02	**Nombre**	Utils
Propósito	Refractorizar diversos métodos usados por los componentes del subsistema con el propósito de aumentar la legibilidad del código y facilitar la reutilización de métodos.		
Descripción	- GetAnalysis(): Convierte la salida del análisis on-line con los resultados positivos en una cadena de texto con formato "vendor@id;" para facilitar su almacenamiento en base de datos. - GetAnalysisFromString(): Convierte un string en el formato anterior almacenado en base de datos a una lista de objetos. - GetStringFromCPE(): Convierte un array de CPE´s en un string con formato "platform,software;" - SaveArrayToFile(): Dada una serie de resultados obtenidos de la búsqueda de información se almacenan dentro de un objeto de tipo Result con toda la información asociada.		

Tabla 51. Componente SUBI-02

Identificador	SUBI-03	**Nombre**	Database
Propósito	Refractorizar los métodos de acceso a la base de datos para facilitar la reutilización en otras partes del sistema.		
Descripción	- GetConfigsFromString(): Permite recuperar los CPE almacenados en base de datos y guardarlos en un objeto de tipo Config con el propósito de acceder de una forma más sencilla a los software y plataformas afectados.		

	- FindMD5(): Permite buscar un MD5 en base de datos y obtener el resultado de una búsqueda previa si existe. - GetMalwareDatabases(): Permite recuperar los nombres y direcciones web de todas las bases de datos que se utilizan para recuperar la información. - InsertNewFile(): Permite guardar en la base de datos el fichero analizado con una serie de información como el MD5, los strings en caso de poder ser obtenidos, el SSOO en caso de ser Android y generar la estructura de la tabla para su posterior visualización mediante la interfaz web. - InsertNewReportForFile(): Una vez que se ha obtenido el informe del análisis en el servicio on-line se actualiza la tabla usando el MD5 con los resultados positivos obtenidos. - InsertInfoForFile(): Una vez realizadas las búsquedas por las distintas bases de datos se actualiza la información del fichero usando el MD5 y se incluyen las cadenas con software, sistemas operativos y CVE´s encontrados durante el proceso. - InsertDownloadableFile(): Cada vez que se genera un fichero (firmas Yara, reportes de análisis, ficheros de estadísticas) se almacena su ruta para permitir un acceso directo a la descarga desde la interfaz web. - InsertCVE(): Permite almacenar en base de datos la información asociada a un CVE. - getCVE(): Permite recuperar de base de datos la información asociada a un CVE.

Tabla 52. Componente SUBI-03

Identificador	SUBI-04	**Nombre**	Classes
Propósito	Definir los distintos objetos que serán utilizados para almacenar la información obtenida en cada ejecución.		
Descripción	- Result: Almacena el nombre del fichero, el hash MD5, la lista de resultados positivos del análisis, los CVE´s, los sistemas operativos afectados, el software afectado y los strings que pueden obtenerse del fichero. - Analysis: Almacena el nombre de la compañía de antivirus y el identificador asignado al malware detectado. - VirusDB: Almacena el nombre de la compañía de antivirus y la *url* utilizada para enviar la petición de búsqueda con un identificador dado. - CVE: Almacena el identificador, el CVSS (indicador de impacto), y las configuraciones (CPE´s) afectadas. - ConfigSO: Almacena el software y el sistema operativo afectado. Cada CPE puede tener varios sistemas operativos y software afectados, por lo que se almacena en este objeto para su posterior tratamiento.		

Tabla 53. Componente SUBI-04

Identificador	SUBI-05	**Nombre**	Parsers
Propósito	Interpretar sintácticamente la información existente en cada una de las páginas web que tiene información sobre un malware concreto o un CVE.		
Descripción	Con un estudio previo se ha concluido que cada página debe ser accedida de una forma distinta, por lo tanto debe existir una clase		

		para gestionar las distintas peticiones por cada una de las páginas que vayan a ser consultadas. Con la implementación de un analizador sintáctico genérico basado en listas de palabras clave se mejora la eficiencia y la sencillez de implementación, ya que únicamente se debe enviar una petición con el identificador a la base de datos, recoger el resultado en HTML y ejecutar sobre el texto el analizador genérico para obtener tanto plataformas, como software y CVE´s.
Datos	**Entrada**	Identificador de malware y compañía.
	Salida	Arrays con datos sobre plataforma, software y CVE´s.

Tabla 54. Componente SUBI-05

Identificador		SUBI-06	**Nombre** aptoideCrawler
Propósito		Permitir obtener las últimas aplicaciones aparecidas en mercados de aplicaciones alternativos.	
Descripción		Tras realizar un análisis sobre el proceso de descarga de aplicaciones en un mercado alternativo, se implementará un módulo que de forma automática proceda a descargar las últimas aplicaciones publicadas y las más populares. Además se comprobará la integridad de la descarga, se evitará descargar aplicaciones que han sido previamente descargadas y se realizarán tiempos de espera aleatorios entre descargas para ofuscar el comportamiento de la herramienta con el propósito de evitar un posible bloqueo.	
Datos	**Entrada**	-	
	Salida	Número indeterminado de aplicaciones móviles en formato *.apk*.	

Tabla 55. Componente SUBI-06

Identificador		SUBI-07	**Nombre**	reportWatch
Propósito		Automatizar la generación de información a partir de los informes obtenidos por el módulo de análisis.		
Descripción		Se listarán periódicamente las carpetas donde se almacenan los informes obtenidos y, en caso de existir nuevos informes, se ejecutarán los módulos del subsistema de evaluación de resultados. En caso de utilizar módulo de análisis externo como *cuckoo sandbox* se podrá pasar la ruta por parámetro para buscar en la correspondiente carpeta.		
Datos	**Entrada**	Opcional: Ruta de instalación de un módulo externo.		
	Salida	Ejecución de un módulo del subsistema de evaluación.		

Tabla 56. Componente SUBI-07

Identificador	SUBI-08	**Nombre**	webInterface
Propósito	Ejecutar un servidor web que permita obtener resultados almacenados en la base de datos y ejecutar análisis de ficheros.		
Descripción	Al ejecutar el módulo se creará un servidor web que dispondrá de dos páginas con funcionalidad diferenciada: • La página principal permitirá iniciar el proceso de análisis de la plataforma seleccionando un fichero, configurar algunas opciones durante el análisis y activar a desactivar otros módulos como el reportWatcher o aptoideCrawler. • La página de resultados mostrará los resultados de análisis ya realizados donde consultar información como el nombre del fichero, el MD5, los identificados positivos, los CVE,		

		las plataformas, etc… Así como permitir descargar los ficheros asociados a cada uno de estos análisis. Este módulo deberá poder ejecutarse en cualquier momento sin tener que disponer de una infraestructura de servidores en la máquina con el propósito de facilitar su uso.
Datos	**Entrada**	-
	Salida	Un servidor web ejecutándose en la IP y puerto por defecto.

Tabla 57. Componente SUBI-08

Con la implementación de estos componentes se podrá dar por finalizado el subsistema de recolección de datos. Posteriormente se deberá realizar una serie de pruebas para garantizar tanto el correcto funcionamiento (control de errores, control de ficheros, posibles fallos derivados de implementación), como validar los resultados del diseño escogido.

4.3. SUBSISTEMA II: ANÁLISIS DEL MALWARE

4.3.1. DESCRIPCIÓN DETALLADA DEL SUBSISTEMA II: ANÁLISIS DINÁMICO DE APLICACIONES WINDOWS

El propósito del subsistema de recolección de información es poder ejecutar el malware en un entorno dinámico con capacidad de análisis. Dependiendo del tipo de malware es posible que necesite unas precondiciones para activarse, tales como que exista una versión concreta de *Adobe reader*, *Flash*, o *Java* por poner algunos ejemplos actuales.

Sin este software, el malware no podría ejecutar los *exploits* que utiliza y no tendría efecto, pasando desapercibido durante la ejecución y generando reglas falsas que no definirían el comportamiento real del malware.

Otro punto que sería interesante evaluar durante el subsistema de análisis es el de las protecciones con las que cuenta el malware. Cada vez es más frecuente que incluyan mecanismos para detectar el uso de máquinas virtuales o cierto software de análisis, incluso comportamientos del propio entorno. En caso de que el malware que está siendo ejecutado en un entorno de análisis detecte una serie de características comunes no se ejecutará, inhibiendo su comportamiento y evitando de nuevo que se realice un análisis.

Para evitar esto, se deben realizar análisis con mayor profundidad y ofrecer más información. En los análisis realizados durante el diseño, las diferentes fuentes de información no ofrecían detalles sobre este aspecto. Esto provoca que a priori sea imposible en base a los resultados del primer subsistema conocer y evadir las medidas de protección que pueda incluir el malware.

Por estos motivos, es importante disponer de una plataforma de análisis de malware que nos permita modificar sus características para adaptar su comportamiento y facilitar la evasión de las nuevas técnicas que vayan surgiendo. Existe una solución de software libre llamada *Cuckoo Sandbox* que además de permitirnos adaptar el código, permite generar informes en formato MAEC (ver **Anexo 1**) y tiene una comunidad muy amplia detrás, por lo que, a pesar de su relativa juventud como

proyecto, está permitiendo que crezca y mejore de forma muy rápida. Por ello se ha considerado que cumple con las necesidades de la plataforma y no se deberá diseñar un módulo de análisis para este tipo de aplicaciones.

4.3.2. DESCRIPCIÓN DETALLADA DEL SUBSISTEMA II: ANÁLISIS DINÁMICO DE APLICACIONES PARA ANDROID

Una de las carencias que presenta la anterior solución es que no permite el análisis de aplicaciones Android, y a pesar de que hay soluciones similares como *DroidBox*, estás no cuentan con una comunidad tan grande y su desarrollo no está siendo tan rápido.

Debido a los motivos mencionados en la introducción, uno de los requisitos fundamentales de la plataforma era poder analizar aplicaciones móviles.

Ya que los análisis realizados sobre el funcionamiento de *Droidbox* no ofrecieron los resultados esperados, se ha decidido utilizar un analizador online, *Anubis*, con su versión para móviles, *Andrubis*.

De esta forma, aunque se pierde la capacidad de personalización y la privacidad y confidencialidad que en ocasiones es necesaria, podemos disponer de un sistema que en menos de diez minutos emita un informe en formato *XML* con las principales operaciones que realice la aplicación al instalarse y ejecutarse en el emulador (ver **Anexo 2**).

Dado que uno de los objetivos del proyecto es estandarizar los formatos de información este informe generado deberá ser convertido a un formato tipo MAEC.

El propósito final de este subsistema es que, independientemente del módulo que se utilice, ya sea *Cuckoo*, *OSSIM* u otra plataforma como Anubis, al finalizar el proceso de análisis dinámico se disponga de uno o varios informes, representados en formato estándar, que sirvan para describir el comportamiento del software analizado y puedan utilizarse como entrada para el módulo de resultados.

4.3.3. COMPONENTES DEL SUBSISTEMA II

Dado que no se realizará ninguna modificación sobre el diseño de *Cuckoo sandbox*, en esta sección solo se detallará el componente diseñado para el análisis en la plataforma *Anubis*.

Identificador	SUBII-01	**Nombre**	AnubisUploader
Propósito	Enviar un archivo ejecutable de cualquier tipo, incluyendo *apk*, y obtener un informe en formato XML o HTML (dependiendo de lo que el usuario quiera) para su posterior análisis.		
Descripción	Este componente facilita que un usuario o un proceso envíen un archivo de tipo ejecutable a un servidor de análisis dinámico on-line. Para realizar el proceso es necesario que junto con el archivo se envíen una serie de datos y un *captcha*, ya que sin este el tiempo para recibir el informe es cercano a un año. Al lanzar el módulo, el usuario podrá elegir el formato de salida como HTML o XML y en cuanto el informe esté disponible se descargará.		

		- GetReportFromFile(): Automatiza el proceso de subida y recuperación del informe dados un fichero ejecutable y un formato. - SaveToFile(): Posibilita guardar el informe obtenido en el disco duro.
Datos	**Entrada**	Fichero ejecutable y String con formato de salida (XML o HTML)
	Salida	Informe en el formato solicitado

Tabla 58. SUBII-01

4.4. SUBSISTEMA III: EVALUACIÓN DE RESULTADOS

4.4.1. DESCRIPCIÓN DETALLADA DEL SUBSISTEMA III

El último paso de la plataforma confiere la capacidad de identificar las características y comportamientos del malware en base a los análisis dinámicos. Para ello, este subsistema deberá analizar de forma sintáctica los informes generados en el subsistema de análisis, que podrán estar en formato MAEC 1.1 o en el formato XML de Anubis, realizará dos acciones distintas:

- Generar una firma Yara, que servirá tanto para identificar el fichero ejecutable mediante las cadenas de texto que pueden extraerse como mediante un informe dinámico de su comportamiento. Para ello se identificarán patrones tales como el número de veces que se solicita una librería, el número de conexiones, las librerías que se cargan o los ficheros que se leen y escriben.

- Generar un fichero en formato hexadecimal que facilite la tarea de aplicar sistemas automáticos de *Machine Learning* y pueda clasificar el comportamiento en función del tipo de operaciones que realiza y sus valores.

4.4.1.1. IDENTIFICACIÓN DEL MALWARE MEDIANTE FIRMAS YARA

Desde el momento que el malware se detecta dentro de una red con activos de riesgo, el tiempo de reacción para identificarlo y controlar la amenaza puede determinar el impacto para el sistema al completo. En muchas ocasiones se dispondrá de medidas que pueda identificar esta amenaza, pero en muchas otras solo se dispondrá de una muestra del propio malware sospechoso obtenida en los análisis realizados tras detectar el problema.

El objetivo de este módulo es disponer de una herramienta que, una vez analizada la muestra de malware, pueda generar una firma (ver **Anexo 3**) que lo identifique de forma automática, sin necesidad de que un experto tenga que analizar la muestra de forma manual.

Para ello, este módulo utilizará un proyecto *open source* llamado *Yara* [20]. La implementación actual se centra en el análisis de firma tanto en ejecutables como en procesos, pero al estar basado en las librerías *PCRE* de *C* el análisis de todo tipo de ficheros de texto puede realizarse de forma sencilla y rápida, sin apenas modificar el proyecto original. La potencia de este proyecto reside en que en base a unos valores

y unos operadores sencillos (*and*, *or*, comparadores y enumeradores) se pueden generar una serie de reglas que permitan identificar de forma inmediata el malware analizado mediante el uso de la propia herramienta.

A pesar de que esto presenta una serie de ventajas notables en un entorno de producción donde el tiempo de reacción es primordial, se considera que la firma autogenerada no puede sustituir a la firma producida por un análisis manual en profundidad por no ser suficientemente específica.

Según los estudios previos realizados sobre *Yara* es que, por lo general, las reglas generadas por la comunidad y las empresas involucradas en el desarrollo son reglas muy concretas y genéricas con el propósito de identificar familias completas de malware. Esto solo se puede realizar con un extenso análisis, identificando que diferencia al malware en concreto y expresando esa característica mediante una regla.

El módulo del subsistema será capaz de analizar el informe generado en formato MAEC 1.1, obteniendo una serie de valores sobre llamadas al sistema y comunicaciones que pueden ser un indicador de la actividad del malware.

El número de reglas incluidas en la firma debe ser tan alto en comparación con otras reglas ya que, en caso de ser muy pequeño, es posible que se induzcan más falsos positivos. Por el contrario, si el número de reglas en la firma es muy elevado se generaría una regla muy específica, lo que provocaría que cualquier mínima modificación en el comportamiento evadiese la identificación.

Posteriormente se deberá analizar mediante una serie de pruebas la probabilidad de identificar correctamente el malware en base a estas reglas, poniendo especial atención en los falsos positivos que se pueden dar.

A pesar de esto, la mejora en la generación de reglas es una línea de investigación muy interesante ya que, aplicando técnicas de clasificación de malware, podrían llegar a identificarse las características interesantes del comportamiento y comunes a todos los *malware* y de ahí obtener un set más reducido de reglas que pudiera emplease de forma genérica. Esto solo sería posible con un estudio en profundidad de los comportamientos y reglas generadas, pero debido a la amplia base de pruebas que se ha especificado para la plataforma existen suficientes datos para continuar el desarrollo en esa línea.

Dado que el informe generado por Android se presenta en un formato no estandarizado, se ha diseñado otro módulo específico para analizar el informe y obtener los datos útiles que serán utilizados al generar la firma (ver **Anexo 4**).

4.4.1.2. GENERACIÓN DE UN FICHERO HEXADECIMAL

Para facilitar el uso de técnicas de *Machine Learning* se ha decidido convertir un fichero en formato MAEC 1.1 a hexadecimal (ver **Anexo 5)**. Para ello se ha asignado un número en hexadecimal a cada una de las operaciones catalogadas como API_Call. A continuación se coloca el código de respuesta recibido separado por barras verticales y finalmente se transforman todos los valores de la operación a valores

hexadecimales y se añaden, de tal forma que el nuevo formato queda definido con la siguiente estructura:

00 | 000000000 | 00FF012EA04BBBBA0001

Esta transformación se basa en el proceso que aplican con el proyecto MIST pero simplificado, de tal forma que sea más sencillo generar este archivo sin conocer todos los valores que se pueden obtener. Esto facilita su portabilidad entre plataformas y estándares.

La idea del diseño se basa en que si son conocidas cien muestras de *malware*, catalogadas como tal, con un identificador AA, una respuesta 0000001 y los mismos valores que un informe no clasificado, el sistema podrá identificarlo y decidirá de forma automática si debe ser o no catalogado como *malware*.
Esto requiere muchas pruebas y una gran base de datos ya clasificada, por lo que es posible que la integración con los sistemas expertos y su implementación formen parte de las líneas futuras de la investigación.

4.4.2. COMPONENTES DEL SUBSISTEMA III

Identificador	SUBIII-01	**Nombre**	AnubisAnalyzer
Propósito	Obtener la información clasificada y estructurada de un informe en formato XML emitido por el servicio de análisis Anubis.		

Descripción		Este componente permite extraer la información de un informe generado por Anubis. Para ello se identificarán las etiquetas que contienen información y se obtendrá la información asociada a cada una de ellas. Para simplificar esta tarea habrá 3 categorías: - StaticAnalysis: El contenido de cada una de estas etiquetas se almacenará en un hash, de modo que se pueda acceder a su contenido con el nombre de la etiqueta. - DynamicAnalysis: En este caso se diferenciarán las operaciones de lectura y escritura de ficheros de las operaciones de red. - Services: Por último se almacenará el nombre de los servicios llamados por la aplicación. Todos ellos tendrán almacenado el momento (tiempo) en el que ocurre el suceso, con el propósito de poder trazar un mapa de eventos. Por último se almacenará un valor indicativo que se obtiene en función del tipo de llamadas. Este módulo llamará al módulo de generación de firma y enviará el objeto donde se encuentra toda la información, comenzando el proceso de generación de firma.
Datos	**Entrada**	Informe en formato XML de Anubis.
	Salida	Objeto de tipo ResultAPK.

Tabla 59. SUBIII-01

Identificador	SUBIII-02	**Nombre**	MAECAnalyzer

<table>
<tr><td colspan="2">Propósito</td><td>Obtener la información clasificada y estructurada de un informe en formato MAEC 1.1 obtenido tras realizar un análisis dinámico.</td></tr>
<tr><td colspan="2">Descripción</td><td>Este componente permite extraer información de un informe en formato MAEC 1.1. Para ello se identificarán las distintas etiquetas y se obtendrá toda la información de valor asociada a cada una de ellas. Para simplificar la recogida de información se obtendrá la siguiente información:

- Objects: Información relativa a aquellos objetos que aparecen en el código.

- API_Call: La mayor parte de la información se extraerá de estas etiquetas. Dependiendo del tipo de llamada tendrá unos valores u otros, por lo que se prestará atención a dos atributos en concreto:

o El número de llamadas que se realiza a cada una de las APIS

o Los valores que se pasan a cada una de estas llamadas.

De esta forma se podrán evaluar los resultados tanto en función del número de operaciones como de los valores que se utilizan.

- Network: También se almacenará la dirección y el puerto al que se realizan las peticiones.

- Strings: Las cadenas de texto que aparecen en los ficheros ejecutables compilados pueden utilizarse para realizar análisis rápidos sobre los propios ficheros, por lo que también se deberá almacenar esta información.</td></tr>
<tr><td rowspan="2">Datos</td><td>Entrada</td><td>Informe en formato MAEC 1.1</td></tr>
<tr><td>Salida</td><td>Objeto de tipo Result</td></tr>
</table>

Tabla 60. SUBII-01

Identificador		SUBIII-03	**Nombre**	MISTConverter
Propósito		Generar un fichero hexadecimal a partir de un informe en MAEC 1.1 que facilite su clasificación por métodos de *Machine Learning*.		
Descripción		Este módulo convertirá la información obtenida en el apartado de llamadas API a código hexadecimal con un formato adaptado que facilite su posterior clasificación. Para ello se recorre cada uno de los elementos dentro de API_Call, se asigna el mismo valor para mismas llamadas, se almacena el código de resultado y se transforman todos los valores a formato hexadecimal, de forma que se obtiene un informe adaptado mucho más compacto y estructurado, más fácil de procesar con técnicas avanzadas de inteligencia artificial.		
Datos	**Entrada**	Informe en formato MAEC 1.1		
	Salida	Fichero en formato Hexadecimal con todas las llamadas a la API.		

Tabla 61. SUBII-01

Identificador	SUBIII-04	**Nombre**	YaraSignatureGenerator
Propósito	Generar una firma Yara en base a un informe proporcionado.		
Descripción	Este módulo generará una firma Yara en función del tipo de objeto recibido. Esta diferenciación viene motivada por que el proceso para generar la firma difiere ligeramente según si es de un APK o de un informe en MAEC. En el caso de un informe en MAEC se atiende principalmente a una serie de *strings* escogidas de forma aleatoria, al número de llamadas que se hace a las APIS y a los valores que algunas de esas llamadas contienen, atendiendo principalmente a la carga de librerías, lectura y escritura de ficheros, etc. Estos valores deben ser escogidos en		

		función de las estadísticas y tendencias detectadas durante las pruebas para mayor exactitud. Por el contrario para los APK se tendrá en cuenta datos como los permisos concedidos, los *broadcast* utilizados, los servicios invocados, las conexiones realizadas (y a donde) o los ficheros leídos y escritos. Una vez introducidas las cadenas a evaluar en condiciones se tendrá que evaluar si se cumplen o no y de qué forma. El objetivo es tener dos métodos distintos de identificación, mediante strings (para ficheros) o mediante los resultados del análisis (para análisis dinámicos). Para el análisis dinámico se tendrá en cuenta el número de veces que se ejecuta cada operación y que aparezca al menos uno de los valores encontrados, de forma que si se cumplen todos posiblemente sea el software analizado.
Datos	**Entrada**	Objeto de tipo Result con información estructurada.
	Salida	Fichero .yara con la firma asociada a cada informe.

Tabla 62. SUBII-01

5. IMPLEMENTACION

En esta sección se discutirán diversos aspectos referentes a la implementación final realizada de los subsistemas previamente diseñados, justificando las decisiones en base a las cuales se han desarrollado los distintos módulos.

Debido a que no existe un requisito de restricción impuesto sobre el lenguaje a utilizar se ha optado por Ruby [21]. Este lenguaje interpretado ofrece la suficiente flexibilidad como para permitir un diseño modular, eficiente e interoperable, tanto entre plataformas como con otros lenguajes. Esto facilitará su integración con todo tipo de módulos en cualquiera de los distintos subsistemas.

Otro punto a destacar es la gran cantidad de funcionalidades ya desarrolladas existentes, que se distribuyen en forma de gemas. Estas gemas facilitan tareas como la interpretación sintáctica de ficheros *JSON, HTML y XML (Nokogiri),* la gestión de peticiones post (*Rest client*) o incluso aplicar métodos de transformación OCR (*Tesseract*).

5.1. MalwareAnalyzer

La ejecución del módulo viene dada por tres parámetros:

- **f**: La ruta al fichero que se analizará, siendo obligatorio.
- **A**: Parámetro opcional y sin argumento que cuando está presente envía el archivo para su análisis en el servicio on-line Anubis.
- **C**: Parámetro opcional y sin argumento que cuando está presente comprueba si existe una variable de entorno con la ruta de *Cuckoo sandbox* configurada y procede a enviar los ficheros para su análisis dinámico.

Para ello se utiliza la librería *optargs*, que se encarga de validar y gestionar los parámetros introducidos por el usuario. Esta comprobación podría incluir algún tipo de expresión regular para garantizar los formatos aceptados, pero dada la gran variedad de formatos que acepta VirusTotal no se ha considerado necesario.

Una vez que se dispone de la dirección local del archivo, se comprueba si su valor MD5 ya está almacenado en la base de datos. En ese caso, en lugar de realizar la petición a VirusTotal, se recuperarán los valores asociados almacenados en la base de datos. En caso de no haber sido analizado previamente se envía una petición POST al servicio con los siguientes datos:

- **File**: La ruta del fichero a analizar.
- **ApiKey**: La clave de desarrollador para acceder a la API que se obtiene tras registrarse en VirusTotal.

Una vez que es enviada la petición, se solicita el resultado mediante una nueva petición POST, que incluye el *sha256* del fichero enviado para el análisis y la clave de acceso a la API.

En caso de que el informe no esté disponible, se detiene la ejecución del módulo durante treinta segundos y se vuelve a realizar la petición hasta obtener el reporte.

Cada una de las peticiones recibidas se encuentra en formato *JSON*, por lo que se utiliza la librería *JSON* para analizar las respuestas obtenidas.

Una vez que se obtiene el informe generado por VirusTotal, se recorren todos los resultados en busca de aquellos cuya columna '*Result'* tenga un valor distinto de *NIL*, es decir, aquellos resultados con un valor positivo durante el análisis. En ese caso se añadirá a un *array* de objetos tipo *Analysis* los datos de *vendor* (compañía) y *MalwareName* (identificador de malware usado por la compañía).

Con estos datos se crea un objeto de tipo *Result*, donde se almacena el nombre del fichero, el hash MD5 y la lista de identificadores. Si esta lista está vacía significará que no se ha encontrado ningún resultado positivo. A pesar de que el servicio no lo detecte como una amenaza, es posible que sea algún tipo de malware sin identificar, por lo cual este proceso no debería excluir la realización de un análisis en profundidad, ya sea estático o dinámico.

Dado que el propósito de este módulo es concluir si el fichero es una amenaza conocida y obtener información sobre ella, en caso de no estar identificado no se podrá encontrar información sobre ella y por tanto finalizará su ejecución.

Si la lista tiene resultados, se almacenará en la base de datos junto con el identificador y el MD5, de forma que pueda consultarse en un futuro o ser mostrada por la interfaz web. Por último, mostrará un mensaje al usuario que contiene los resultados positivos encontrados durante el análisis.

El siguiente paso es comprobar para cada resultado si tiene una base de datos pública y está incluida dentro de la plataforma. Para ello se recorre toda la lista de resultados positivos y si alguno de ellos pertenece a alguna de las bases de datos usadas por la plataforma, y de las cuales se dispone del módulo correspondiente para análisis (ver **Anexo 6**), se ejecuta ese módulo, devolviendo un *array* con el software, los sistemas operativos y los CVE´s identificados.

Los analizadores sintácticos funcionan de forma similar entre ellos. Dado que cada caso es distinto y no se puede generalizar, existe un módulo para cada una de las bases de datos que se encarga de enviar una petición con el identificador de malware y recuperar la respuesta. El análisis de la respuesta es realizado por medio de las funciones implementadas en la gema *Nokogiri*, principalmente haciendo uso de *xpath* para obtener la información que está siendo buscada. Posteriormente, la respuesta obtenida es enviada a un analizador genérico, el cual busca palabras concretas en listas de software y sistemas operativos. Estas listas son ficheros de texto independientes, con el propósito de que puedan ser ampliadas añadiendo palabras clave separadas por “;”.

Dado que el formato de los identificadores es estándar, además se buscarán todas aquellas coincidencias existentes con la expresión regular */CVE-\d+-\d+/* para obtener todos los posibles identificadores.

Una vez que termina la ejecución de este paso, el resultado es devuelto en forma de *arrays* que serán almacenados en la base de datos para su posterior consulta. En caso de que no se encuentren resultados válidos, se guardarán valores vacíos.
El propósito de esta segunda etapa del proceso es disponer de la información de sistemas operativos, software y CVE´s afectados según las búsquedas realizadas.

El último paso solo se realiza en caso de disponer de algún CVE identificado. Desde la página del NIST es posible buscar los identificadores de las vulnerabilidades y recuperar el CPE asociado, lo cual permite recoger la información sobre plataformas y software vulnerable de manera estandarizada. En caso de que se hayan identificado CVE´s se consulta en la base de datos si ya existe información asociada a ese identificador. En caso negativo, se realiza una petición a la página para obtener las distintas configuraciones afectadas y almacenarlas en la base de datos para futuras consultas.

A continuación se muestra un diagrama de actividad donde se reflejan los pasos seguidos por este módulo:

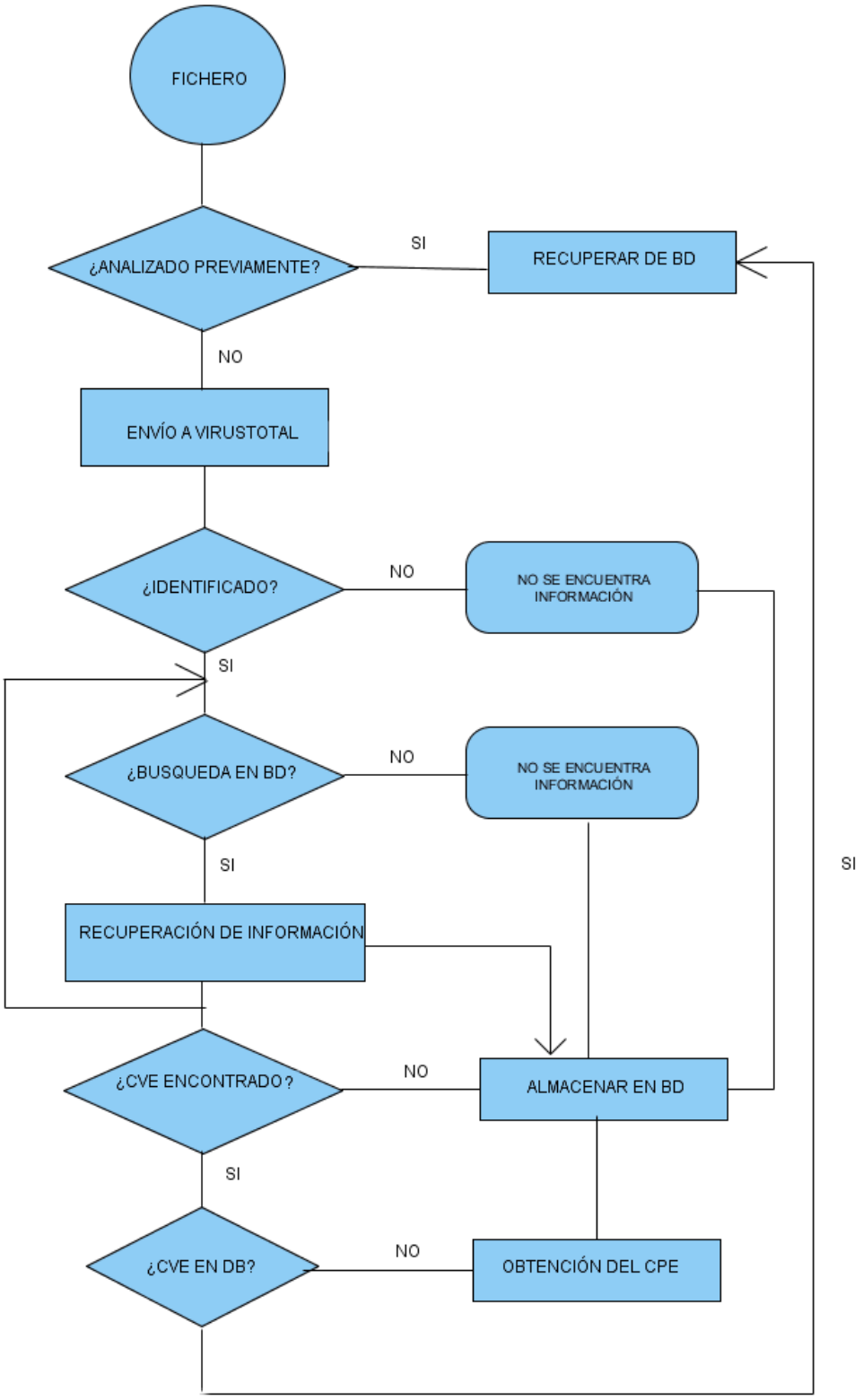

Ilustración 4. Diagrama de flujo de MalwareAnalyzer

5.2. AnubisUploader

La ejecución del módulo viene dada por dos parámetros:

- **f**: La ruta al fichero que se analizará, siendo obligatorio.
- **t**: Parámetro para indicar el formato en el que se obtendrá el informe (HTML o XML), siendo obligatorio.

Una vez introducidos ambos parámetros, se realizará una petición GET a la página con el propósito de simular el comportamiento normal de un usuario y poder obtener la imagen del *captcha*. Esta imagen es utilizada para comprobar que el análisis sea solicitado por una persona y no un proceso automatizado. En caso de no introducir el *captcha* se variará la prioridad del análisis y el tiempo de espera pasará a ser de semanas.

Una vez que se ha obtenido toda la información se tendrá que realizar una nueva petición *POST* donde se enviará la cookie obtenida en la anterior petición, el contenido del *captcha* y el fichero escogido para el análisis.

Para obtener el contenido del *captcha* se ha utilizado una librería llamada *Tesseract* que, de forma sencilla, es capaz de leer el contenido de una imagen y transcribir las letras, siempre y cuando no cuenten con ningún tipo de protección. El ratio de acierto es bastante elevado, pero aun así se deben gestionar los fallos para repetir la subida en caso de que se detecte un *captcha* no válido.

Esta prueba de concepto sirve para demostrar que los *captcha* básicos son una medida insuficiente para evitar los denominados *bots* y que se requiere de técnicas más

sofisticadas, como modificaciones en la imagen, para dificultar la interpretación automatizada de la imagen y añadir un extra de seguridad en este tipo de procesos.

Una vez enviada la petición se tendrá que analizar la respuesta. Si el texto de la página incluye "*succesful*", lo cual se comprueba mediante una expresión regular, la petición de análisis ha sido enviada correctamente, por lo que usando de nuevo Nokogiri se buscará la dirección asociada al informe mediante la cadena *"/action=result&task_id/"*. Esta dirección permitirá la descarga del informe una vez que se encuentre disponible.

El siguiente paso será realizar una nueva petición a esa dirección y, si el contenido de la respuesta incluye *"Time remaining",* significará que el informe no está listo por lo que se esperará 120 segundos y se realizará una nueva petición.
Cuando la respuesta no contenga esa cadena significará que se ha recibido el informe y se procederá a salvar en una ruta predefinida el contenido en el formato previamente seleccionado durante la ejecución.

A pesar de que el diseño de este módulo está destinado a la descarga de informes de aplicaciones móviles para Android no se ha restringido con el propósito de poder analizar cualquier tipo de aplicación por separado en Anubis, por lo que también se pueden enviar ficheros *.exe*, comprimidos o *pdf* y obtener su informe en formato XML o HTML.

A continuación se muestra un diagrama de actividad donde se reflejan los pasos seguidos por este módulo:

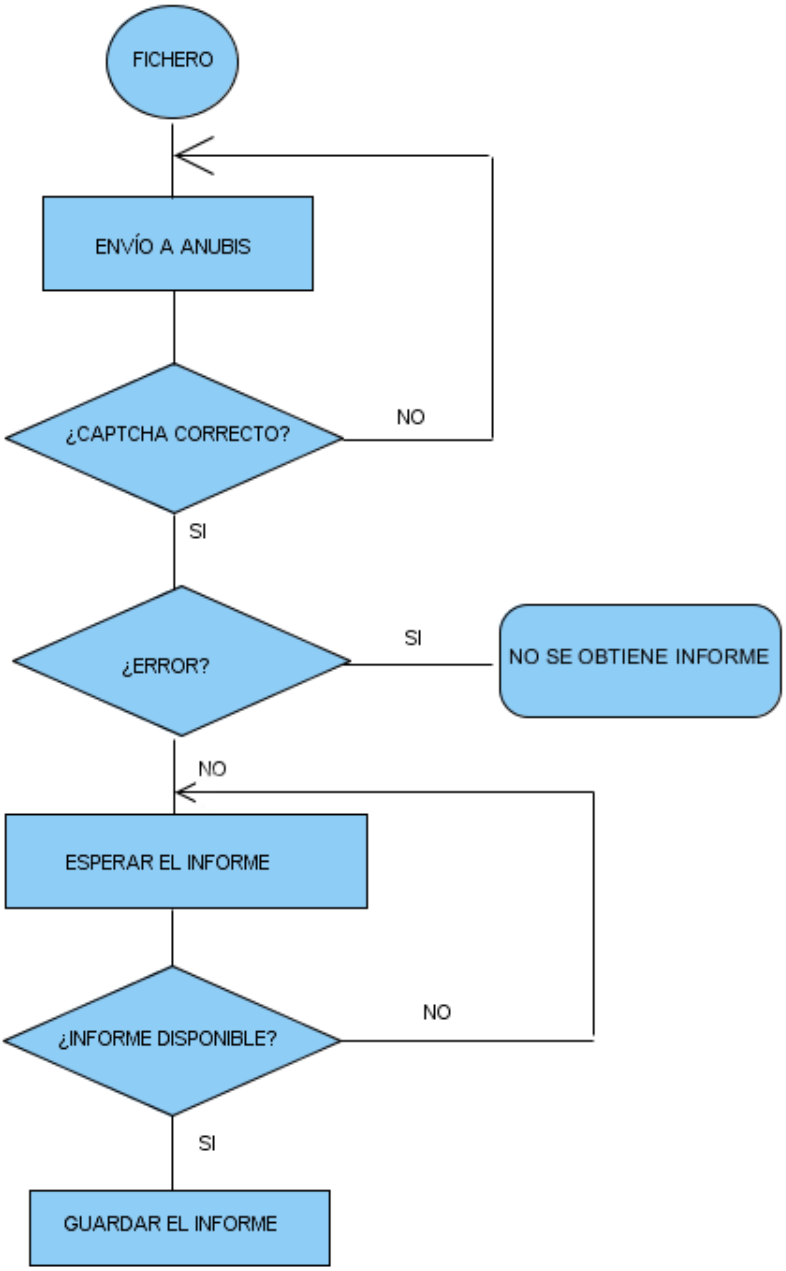

Ilustración 5. Diagrama de flujo de AnubisUploader

5.3. AnubisAnalyzer

Este módulo tiene como entrada un informe emitido por la plataforma de análisis dinámico Anubis en formato HTML o XML. Actualmente el módulo no genera ningún tipo de análisis sobre los ficheros HTML, por lo que quedaría pendiente para futuras ampliaciones del módulo. Su ejecución tiene un único parámetro obligatorio (-r) que debe recibir la ruta del informe en formato XML para su procesamiento.

En primer lugar se inicializa un objeto donde se almacenará información general sobre el malware tal como el identificador, la familia, una breve descripción, el nivel de amenaza o si ha sido descubierto fuera de un espacio controlado. Esta información se utilizará posteriormente para contextualizar la firma *yara* generada.

Para localizar una sección determinada dentro del fichero XML se utilizará *xpath* que, mediante el uso de expresiones regulares del tipo (//*), podrá localizar secciones identificadas por una etiqueta y recorrerlas, mostrando o almacenando la información relativa a estas y a los nodos hijo que pueda tener cada una de las secciones.

Para almacenar en este objeto toda la información referente al análisis estático se ha creado un hash que utilizará el nombre de la sección para almacenar el valor. Este *hash* se crea dinámicamente según se encuentran nuevos atributos, de forma que no sería necesario extender este apartado si se incluyen cambios en los atributos del análisis.

Este paso recogerá información de carácter estático que puede obtenerse analizando la aplicación, tal como los permisos configurados, los *broadcasters* utilizado o el paquete al que pertenece.

La segunda parte de este módulo se centrará en obtener información de las escrituras y lecturas. Para ello se utilizará de nuevo *xpath*, buscando en primer lugar las lecturas y posteriormente las escrituras. Una vez que se localiza un nodo que pertenezca a esta clase se almacena el tiempo, la ruta y la información. Esta información se almacenará en un objeto específico y se añadirá a un *array*, dependiendo de si es escritura o lectura.

El tercer paso será buscar las conexiones de red y almacenarlas. Para ello se utilizará *xpath* y se diferenciará si son conexión de escritura o lectura o inicios de conexión.

Por último se realizará el mismo proceso con los servicios y el valor de riesgo. Tras finalizar el proceso, la información obtenida se almacenará en un objeto y se ejecutará el módulo encargado de generar la firma, el cual recibe este objeto como parámetro de entrada.

A continuación se muestra un diagrama de actividad donde se reflejan los pasos seguidos por este módulo:

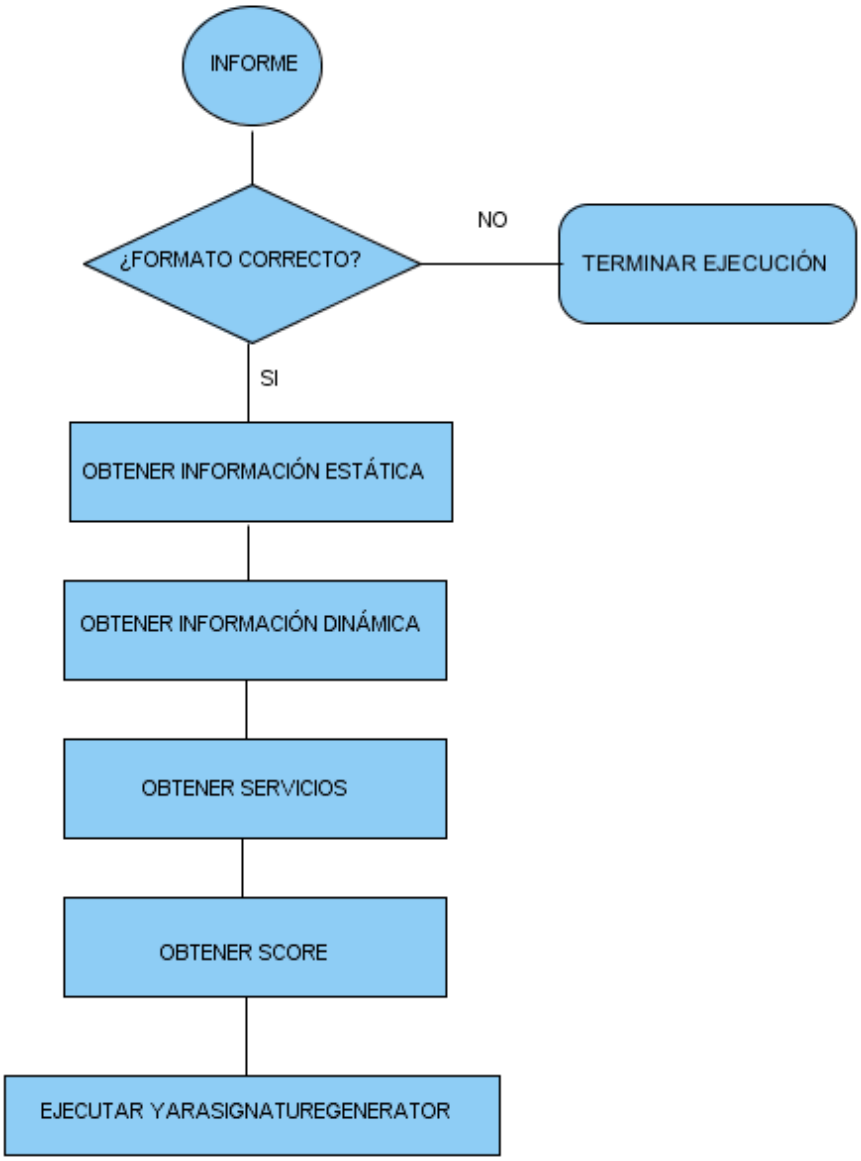

Ilustración 6. Diagrama de flujo de AnubisAnalyzer

5.4. MAECAnalyzer

La ejecución de este módulo viene dada por tres parámetros:

- **r:** La ruta del informe generado por un análisis dinámico en formato MAEC 1.1, de carácter obligatorio.
- **n**: Parámetro opcional que crea un fichero en formato .csv con todas las funciones API_CALL encontradas y el número de veces que son llamadas.
- **v**: Parámetro opcional que crea un fichero en formato .csv con todos los valores asociados a cada una de las funciones API_CALL encontradas.

Este módulo actúa de forma similar a AnubisAnalyzer pero aplicado a informes en formato MAEC. El analizador sintáctico obtendrá la información de dos fuentes dentro del informe:

- Los *objects*: Serán identificados mediante *xpath* con la expresión *"'/*//Object'"*. Dado que contienen información sobre los ficheros y librerías, se considera conveniente almacenar toda la información para su futuro uso.

  Con el propósito de poder recuperar información sobre el fichero analizado en la base de datos, se recupera del último elemento de esta categoría el MD5, ya que siempre corresponde con el *hash* del fichero analizado. Esto permite, por ejemplo, recuperar el campo *strings* que se almacena en base de datos durante la ejecución del subsistema de recogida de información.

- Las *Action_implementations*: Estos valores se recuperan usando *xpath* con la expresión *"//Actions/*/Action_Implementation/*"* y resultan fundamentales para la creación de la firma ya que contienen las API_CALL. Siempre que se localice una de estas llamadas se añadirá a un *hash*, que será usado para llevar un registro

de los tipos de llamadas que se realizan y su número. Además, se almacenarán en otro *hash* los valores que aparezcan en la llamada, para poder ser usados posteriormente.

Además de los anteriores, dentro de estos objetos también aparecen los *NetworkActionAttributes*, que se encargarán de almacenar toda la información referente a cada una de las conexiones que se realicen. Por ello, se guardarán tantos los puertos como las IP´s externas e internas que sean detectadas durante el análisis dinámico.

Una vez que se ha obtenido toda la información con el proceso posterior, se añadirá al objeto *result*, que tendrá atributos muy similares al caso de AnubisAnalizer. Dependiendo de si el usuario ha ejecutado el módulo con las opciones para generar los informes sobre el tipo y número de llamadas al sistema o los valores de estas llamadas, se generarán estos informes.

Para facilitar el análisis de los mismos se ha definido un array con cada una de las operaciones detectadas hasta la fecha para que, a la hora de listar en el fichero CSV, estos valores aparezcan ordenados y sea más sencillo y rápido trabajar con los valores de cara a generar estadísticas y clasificaciones. En el caso de que estos archivos se generen, se actualizará la base de datos con la rula absoluta a estos ficheros, con el propósito de facilitar su consulta desde la interfaz web.

Una vez que toda la información ha sido identificada se añadirá al objeto result y se ejecutará el módulo de generación de firma, recibiendo como parámetro la información obtenida tras la ejecución de este módulo.

A continuación se muestra un diagrama de actividad donde se reflejan los pasos seguidos por este módulo:

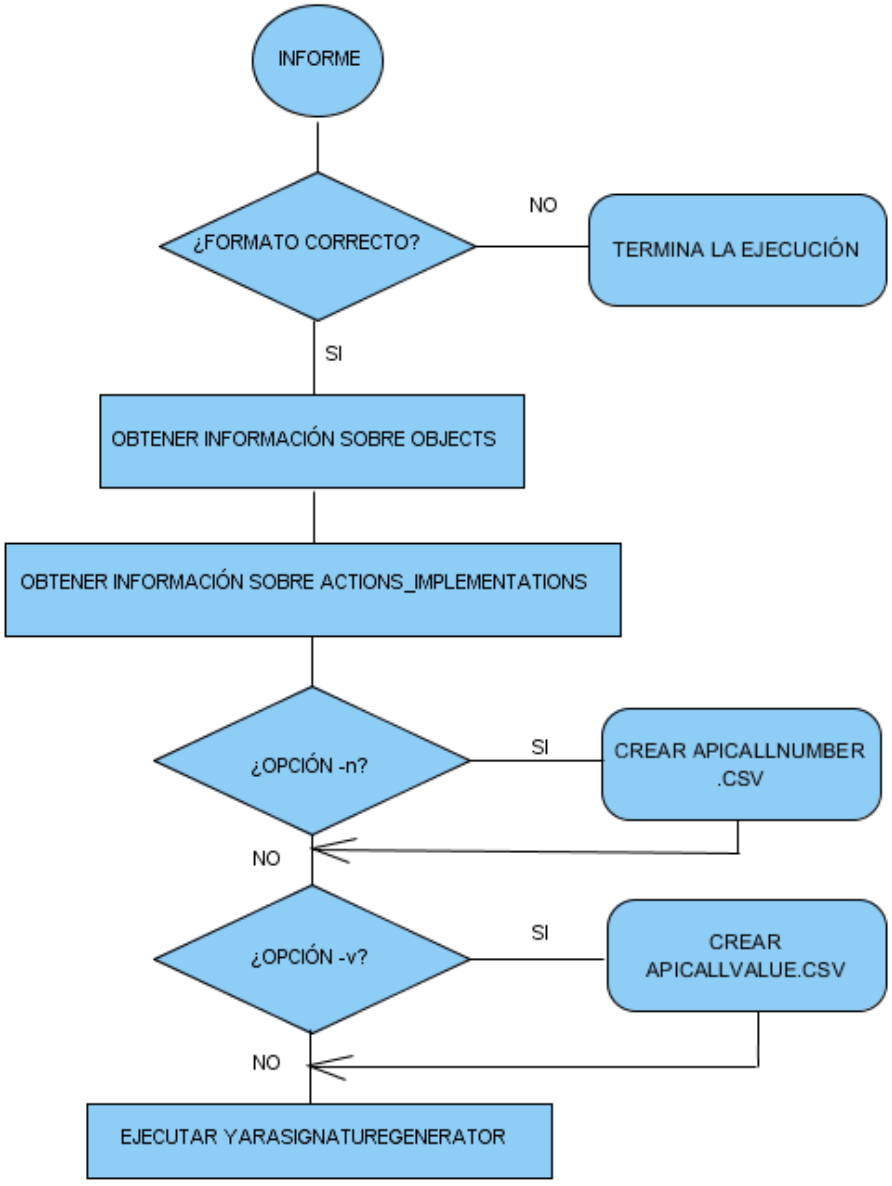

Ilustración 7. Diagrama de flujo de MAECAnalyzer

5.5. MISTConverter

Este módulo recibe como parámetro (-r) un fichero en formato MAEC 1.1 y convierte los valores de las llamadas a funciones del sistema en cadenas de texto en formato hexadecimal, con el propósito de facilitar su posterior análisis mediante técnicas automáticas.

Para ello se utilizará *Nokogiri* y *xpath*, identificando que nodos son del tipo "API_CALL" y transformando cuando sea necesario su valor a hexadecimal. Se han identificado tres casos distintos:

- El valor ya está en hexadecimal, en cuyo caso se elimina el 0x que aparece identificándolo.

- El valor es un número, en cuyo caso se transforma en su valor hexadecimal equivalente, usando la función *to_s(16)*.

- El valor es una cadena de texto, en cuyo caso se transforma en hexadecimal mediante la función *unpack('U'*actionChildren2.text.length).collect {|x| x.to_s 16}.join.*

Para asignar el mismo valor a cada tipo de llamada se ha definido un *array* con los valores encontrados hasta la fecha, de forma que su identificador sea la posición que ocupan.

Para mantener y detallar toda la información posible también se han incluido las conexiones de red, usado el identificador FF.

Tal y como se indicó en la fase de diseño, la cadena final tiene el formato:

11 |00000001| 24897234234236423654234

Un problema detectado de este método es que se desconoce el número exacto de llamadas al sistema. Con el propósito de completar el listado de estas llamadas, se ha incluido un *array* que irá añadiendo las llamadas que no hayan sido previamente identificadas, de forma que resulte sencilla su identificación y pueda añadirse posteriormente. Finalmente cada una de estas líneas será almacenada en un fichero que se guardará en la ruta especificada.

A continuación se muestra un diagrama de actividad donde se reflejan los pasos seguidos por este módulo:

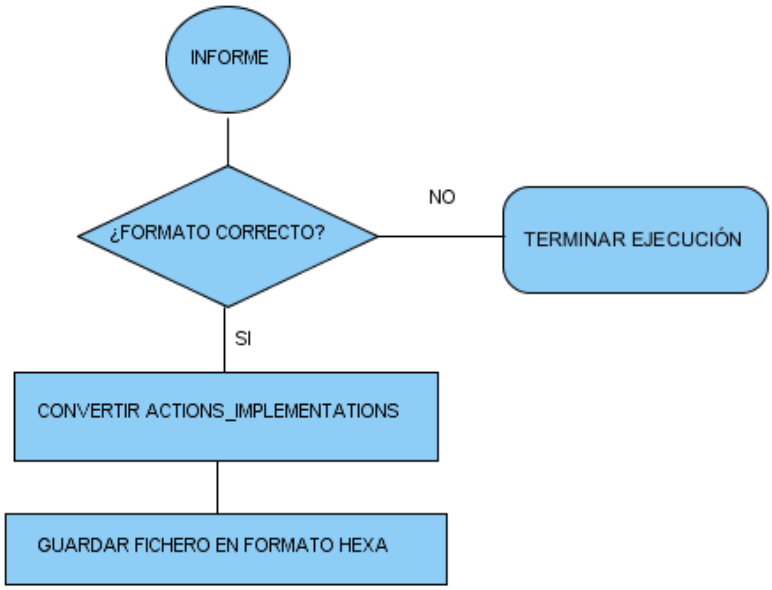

Ilustración 8. Diagrama de flujo de MISTConverter

5.6. YaraSignatureGenerator

Este módulo solo podrá ejecutarse desde los módulos MAECAnalyzer y AnubisAnalyzer, ya que la entrada debe ser un objeto de tipo *result*. En caso contrario la ejecución finalizará sin generar ningún fichero.

En función del atributo "*type*" se ejecutará el método asociado al informe MAEC o el asociado al informe de Anubis.
Ambos métodos comparten la primera parte del proceso, que sirve para describir la firma. Para ello se utilizan los atributos del objeto relacionados con la información, nombre, descripción, etc. que se obtienen en los módulos anteriores.

Para la firma MAEC se utilizan como valores los nombres de todas las llamadas a las librerías que han sido detectadas, con el propósito de poder identificar el número de veces que aparecen. Además, se utilizan una serie de valores que han sido estadísticamente determinados como "interesantes" tras un análisis manual previo dentro de las llamadas al sistema. Estos valores pueden ser nombres de librerías, nombres de registros o nombres de ficheros que aparecen durante los análisis y son un claro indicativo de la actividad.

Por último, también se incluye la información única referente a direcciones IP y puertos utilizados en las conexiones. Esto, aunque es importante para malware que tenga paneles de control o que realice peticiones, puede ser un impedimento si las direcciones cambian ya que la identificación quedaría inutilizada.

Como condiciones para estos valores recogidos se utiliza el número de conexiones (redondeado a la baja), el número de llamadas a las librerías que aparecen y por último alguno de los valores "interesantes" que han sido identificados, de tal forma que si hay un número de llamadas determinadas a una librería y además aparece un valor concreto se puede identificar como un patrón de comportamiento.

En aquellos casos que sea posible, para permitir la realización de los análisis estáticos sobre ficheros y memoria, se ha añadido a la firma generada los *strings* obtenidos durante la ejecución del subsistema de análisis.

Para la generación de las firmas para aplicaciones Android provenientes de análisis con Anubis, dentro de los valores se indicarán los permisos, las operaciones de lectura y de escritura, las conexiones (separadas por tipo) y finalmente los servicios.
Como condiciones se tendrá en cuenta el número de conexiones, así como las direcciones y puertos que se utilizan y que incluyan todos los valores dentro de permisos, lecturas y escrituras.

En ambos casos se almacenará el fichero resultante en formato *.yara* en la dirección especificada y se almacenará la ruta absoluta en la base de datos para su posterior uso en la interfaz web.

A continuación se muestra un diagrama de actividad donde se reflejan los pasos seguidos por este módulo:

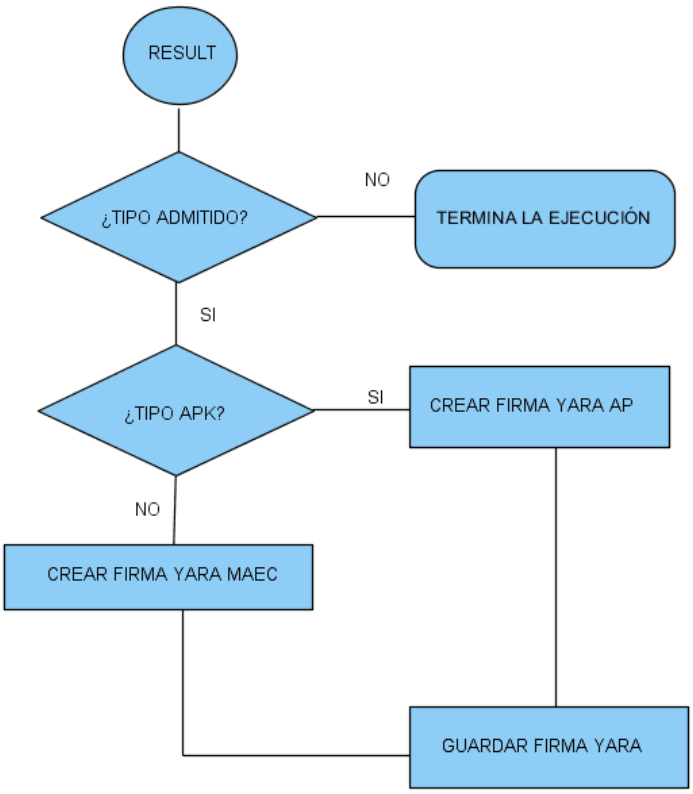

Ilustración 9. Diagrama de flujo de YaraSignatureGenerator

5.7. AptoideCrawler

Para la implementación de este módulo se requirió un análisis previo de la aplicación *aptoide*. Para ello se instaló en un emulador y se procedió a la captura del tráfico de red con *Wireshark* mientras se iniciaba la aplicación. Tras analizar las distintas peticiones realizadas al servidor, se localizó la petición y respuesta de los ficheros XML con la información sobre las aplicaciones que posteriormente eran utilizados para mostrar la información. Además, entre los datos de esta respuesta se incluía el *path*, y el hash md5, por lo que se tenía toda la información necesaria para replicar con un módulo estas peticiones.

Los pasos que sigue el módulo se detallan a continuación:

1. Se obtienen los ficheros XML marcados como *top* y *latest.*
2. Estos ficheros se analizan utilizando la herramienta *Nokogiri* para obtener un nombre de fichero, un hash y la dirección de descarga.
3. La información se almacena en una lista y se compara con los ficheros que ya existen en el directorio, de forma que no se descargue una aplicación que ya ha sido descargada previamente.
4. Una vez que se tienen todas las aplicaciones únicas se procede a realizar una petición por cada una de ellas, obtener el fichero y almacenarlo como .apk tras comprobar su hash MD5.
5. En caso de que el hash no coincida se informa al usuario y se descarta la aplicación para evitar ficheros corruptos.

Tras finalizar la descarga de una aplicación existe una espera aleatoria que tiene una duración comprendida entre dos y cinco minutos para enmascarar el comportamiento de la aplicación. Si esto se combina con el uso de una IP distinta en cada petición es posible que el sistema no sea identificado. Esto permitiría utilizar el módulo como un servicio para obtener las nuevas aplicaciones publicadas en el servidor para su posterior análisis.

A continuación se muestra un diagrama de actividad donde se reflejan los pasos seguidos por este módulo:

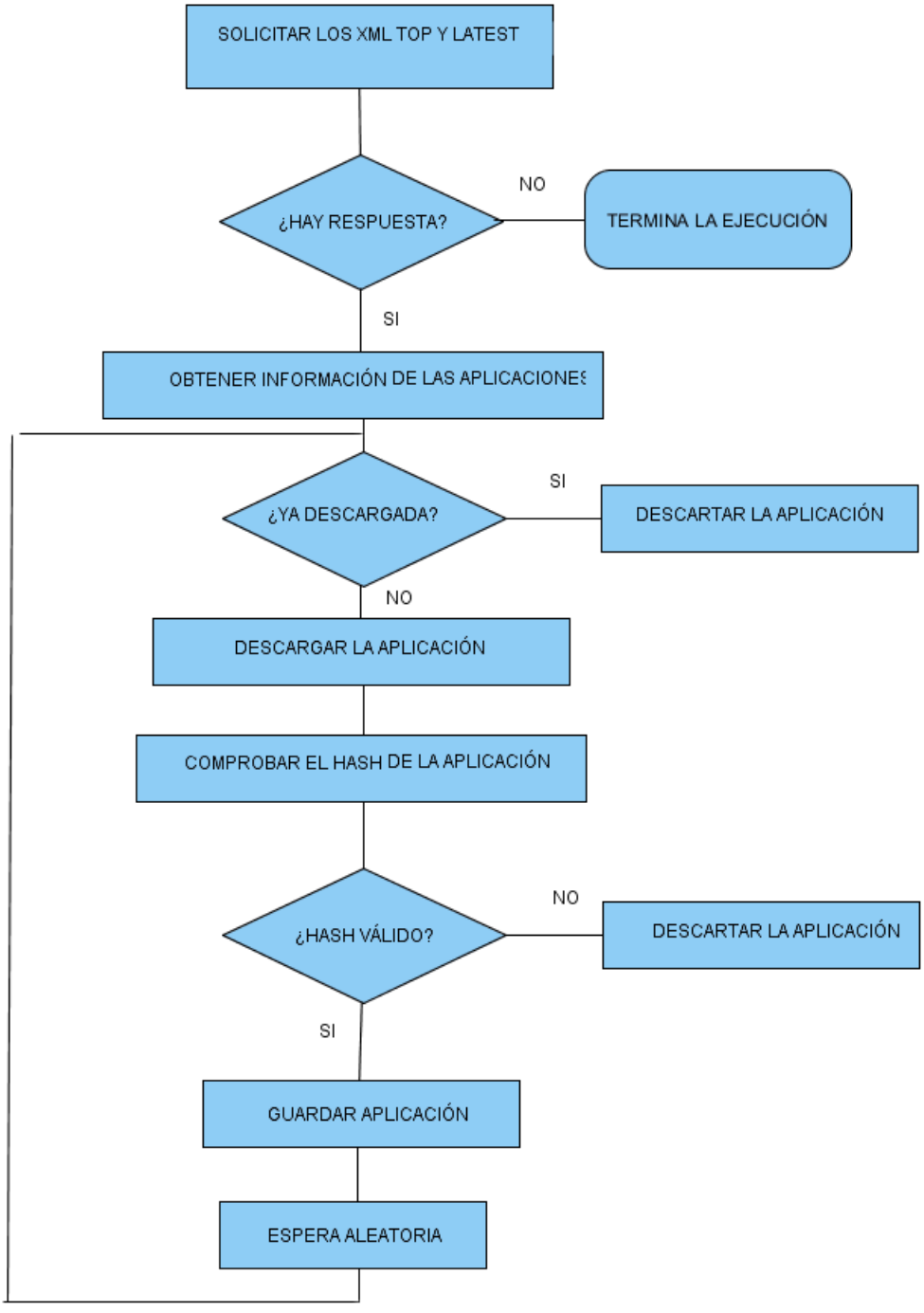

Ilustración 10. Diagrama de flujo de aptoideCrawler

5.8. APLICACIÓN WEB

Con el propósito de facilitar la consulta de los datos y realizar los análisis de forma sencilla, se ha implementado una aplicación web basada en Bottle [26]. Esto permitirá ejecutar un servidor web ligero que no requiere de infraestructura, por lo que podrá desplegarse en cualquier entorno que tenga configurado Python y bottle.

La aplicación web consta de dos páginas:

- Una página donde el usuario puede seleccionar un archivo del sistema y enviarlo para su análisis. Además existe un checkbox que en caso de ser marcado enviará el fichero para su análisis en Anubis.

- Una página de resultados donde se podrá consultar la información asociada a cada análisis realizado, incluyendo el nombre del fichero, el MD5, los positivos detectados, el software y los sistemas operativos vulnerables, los CVE y enlaces de descarga a los ficheros generados por el módulo de análisis de MAEC y la firma Yara.

Debido a las medidas de seguridad que implementan los navegadores más modernos se ha implementado un mecanismo que copia el archivo seleccionado al directorio actual, realiza las operaciones necesarias y posteriormente lo borra ya que, para reforzar la seguridad, el navegador no puede acceder directamente a la ruta del fichero seleccionado. La comunicación con la base de datos se realiza mediante un *plugin* del propio *framework.*

A continuación se muestra un diagrama de actividad donde se reflejan los pasos seguidos por este módulo:

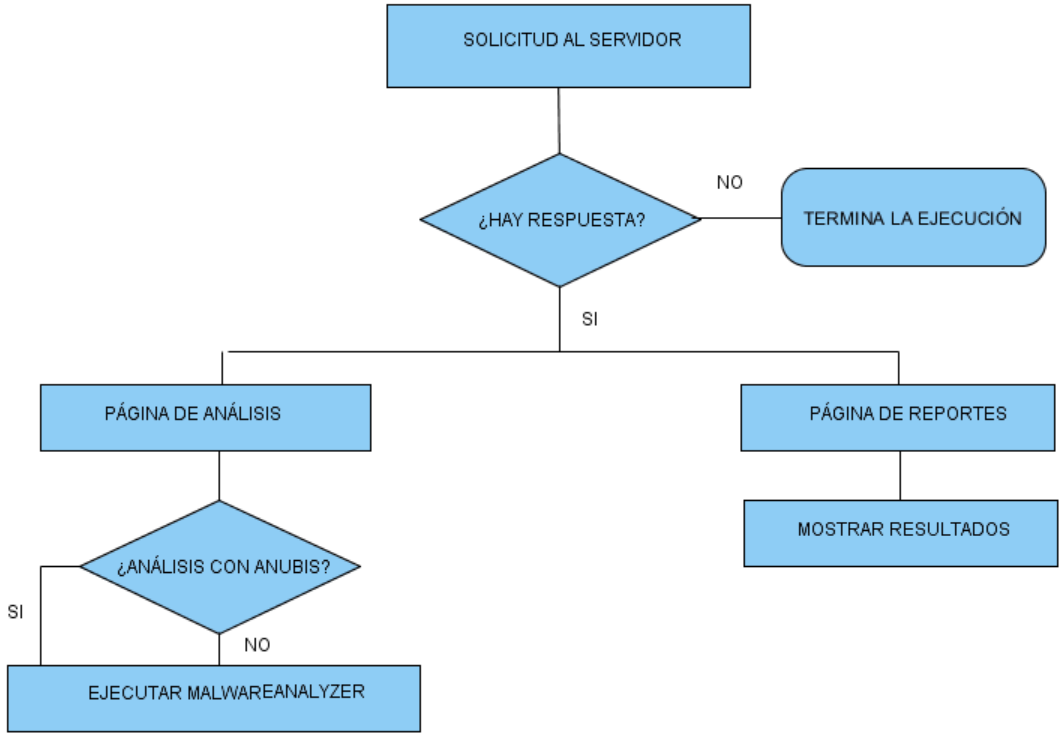

Ilustración 11.Diagrama de flujo de webInterface

A continuación se muestran dos capturas de pantalla para mostrar la interfaz web diseñada:

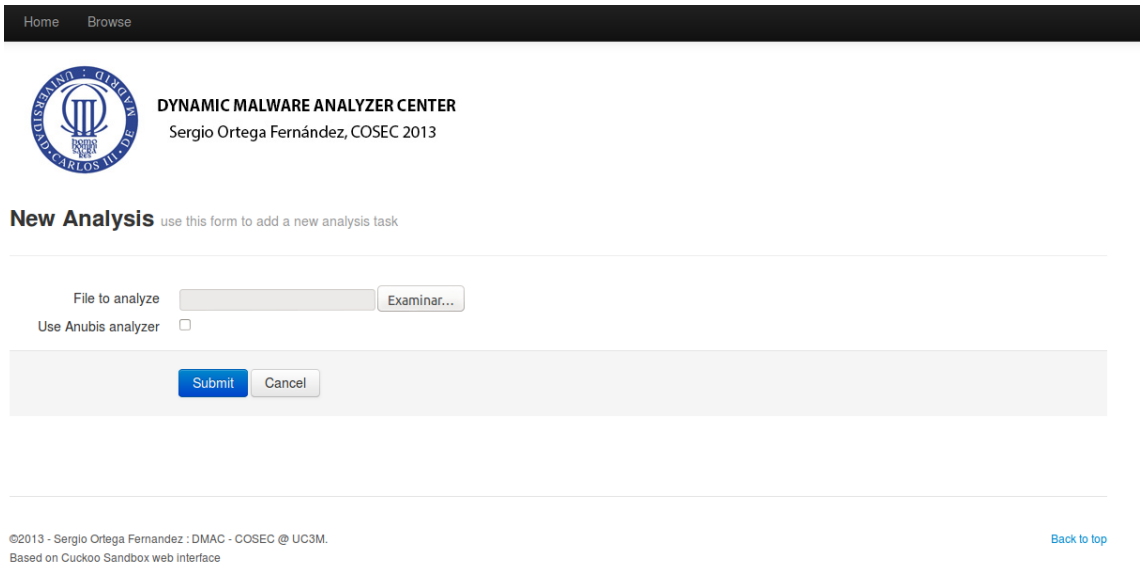

Ilustración 12. Página para gestionar el envío de análisis

CVE's	Software	SSOO	API calls	API values	Yara signature
None	None	Windows 2000;Windows 95;Windows 98;Windows Me;Windows NT;Windows Server 2003;Windows XP;Windows 7;	Download	Download	Download
None	None	Android	N/A	N/A	N/A
None	None	Android	N/A	N/A	N/A
None	None	None	N/A	N/A	N/A
CVE-2012-4969;	None	None	N/A	N/A	N/A
CVE-2012-4969;	Internet explorer; ;	Windows 2000;Windows 95;Windows 98;Windows Me;Windows NT;Windows XP;Windows 7;	N/A	N/A	N/A
None	internet explorer;Internet	Windows	N/A	N/A	Download

Ilustración 13. Página donde se muestran los resultados de los análisis previos

6. PRUEBAS

En esta sección se detallan las pruebas realizadas con el propósito de evaluar los resultados obtenidos tras la implementación del sistema. Dada la heterogénea naturaleza de los ficheros sobre los que se puede realizar el análisis, las pruebas se estructurán en dos bloques independientes; pruebas de aplicaciones para sistemas Android y pruebas para ficheros analizados sobre la plataforma Windows.

En el **Anexo 10** puede consultarse una tabla resumen sobre las pruebas realizadas que incluye la relación de requisitos funcionales que comprueba cada una de ellas.

6.1. DESCRIPCIÓN GENERAL DEL ENTORNO DE PRUEBAS

Para la realización de las pruebas se ha utilizado una plataforma **Windows 8** con **VMWare Workstation 9.0.1** donde se ha virtualizado un sistema operativo **Ubuntu 12.07** con todas las actualizaciones a fecha de Marzo de 2013.

Para las pruebas realizadas en entornos Windows, se ha optado por configurar una máquina virtual con **VirtualBox** y **Windows XP SP3** sin acceso a internet, de forma que la única comunicación posible sea a través de la interfaz virtual creada por la máquina virtual. La configuración del sistema se ha ido modificando manualmente en función de las vulnerabilidades detectadas durante la ejecución del subsistema de recogida de información.

A pesar de que la virtualización dentro de una máquina virtual no es la solución idónea a nivel de rendimiento, esta decisión ha sido tomada durante el diseño de las pruebas para garantizar la integridad del sistema que realiza las funciones de *host* para ambos sistemas virtualizados. En caso de que se analice un malware que explote una vulnerabilidad para *VirtualBox* que permita acceder al sistema anfitrión, aún se tendría que explotar una segunda vulnerabilidad totalmente distinta acceder al anfitrión físico y comprometer los datos e información existentes.

6.2. PRUEBAS DE APLICACIONES PARA SISTEMAS ANDROID

6.2.1. DESCRIPCIÓN DE LAS PRUEBAS

Para la realización de estas pruebas se utilizará el módulo de descarga automática de aplicaciones implementado durante el desarrollo de la plataforma. Este módulo permite disponer de una amplia base de aplicaciones móviles en formato *.apk* actualizada. Además, al ser obtenidas de una plataforma de distribución de software independiente a *google play* como es *Aptoide*, también se podrá realizar un estudio sobre la cantidad de aplicaciones de propósito malicioso que aparecen publicadas en canales no oficiales.

Esta prueba se realiza en el sistema sin haber realizado ninguna ejecución previa del módulo, por lo que la recolección de información y posterior descarga de aplicaciones tiene una duración aproximada de cinco horas. En ejecuciones posteriores, se descargarán solo las aplicaciones que no hayan sido obtenidas previamente, por lo

que el tiempo oscila en función de los cambios realizados en la publicación de aplicaciones por parte del servidor.

Estas aplicaciones se obtendrán de la lista de más populares y últimas añadidas. En las distintas pruebas realizadas sobre este módulo, los favorables resultados han permitido automatizar la ejecución, por lo que en un futuro podría utilizarse para analizar periódicamente las nuevas aplicaciones que aparecen sobre la plataforma de distribución, identificar el malware y generar una firma para su posterior distribución y uso.

Para esta prueba, el resultado de la ejecución es una carpeta con ciento una aplicaciones móviles, verificadas mediante su hash MD5, que conforman la base de pruebas (ver **Anexo 6**).

Tras obtener la base de aplicaciones, las pruebas consistirán en realizar una ejecución completa de la plataforma de análisis de forma completamente desatendida. Para ello se analizará la aplicación tanto en VirusTotal como en *Anubis* (siempre que el tamaño lo permita), obteniendo toda la información posible de los análisis previos y generando una firma que identifique el comportamiento del software, independientemente de si es catalogado como malware o no. Esto tendrá como propósito evaluar la precisión con la que el sistema es capaz de identificar el software en función de los informes generados por la plataforma de análisis.

Debido a las limitaciones del sistema, la inclusión de una plataforma personalizada de análisis basada en *Droidbox* queda fuera del alcance ya que, en la mayoría de ocasiones, los informes generados por *Anubis* han resultado más completos y detallados, ofreciendo una mayor información con una menor utilización de los recursos.

6.2.2. ANÁLISIS DE LOS RESULTADOS OBTENIDOS

A continuación se incluyen una serie de gráficas donde se muestran los resultados más destacados obtenidos tras la ejecución:

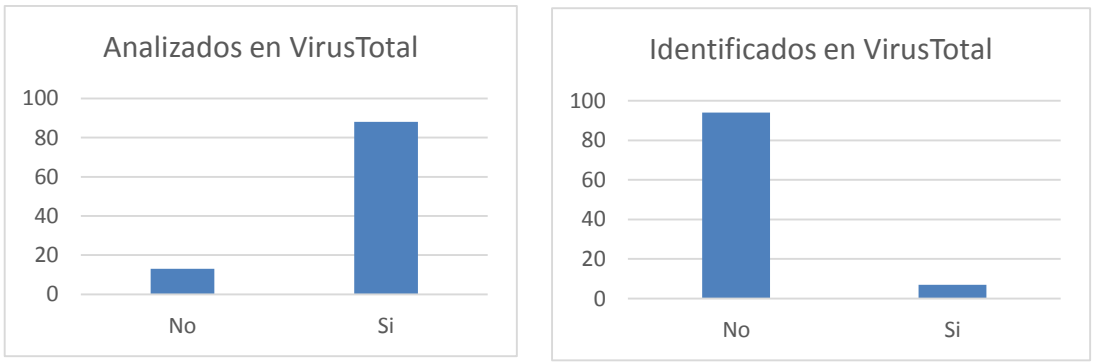

Ilustración 14. Aplicaciones analizadas e identificadas en VirusTotal

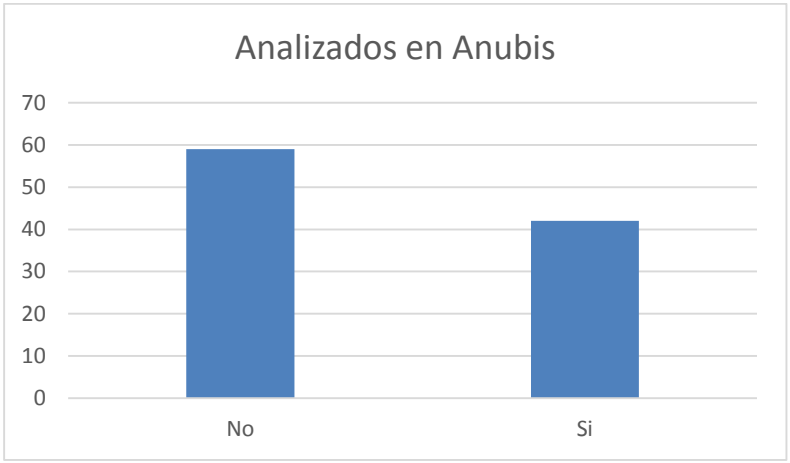

Ilustración 15. Aplicaciones analizadas en Anubis

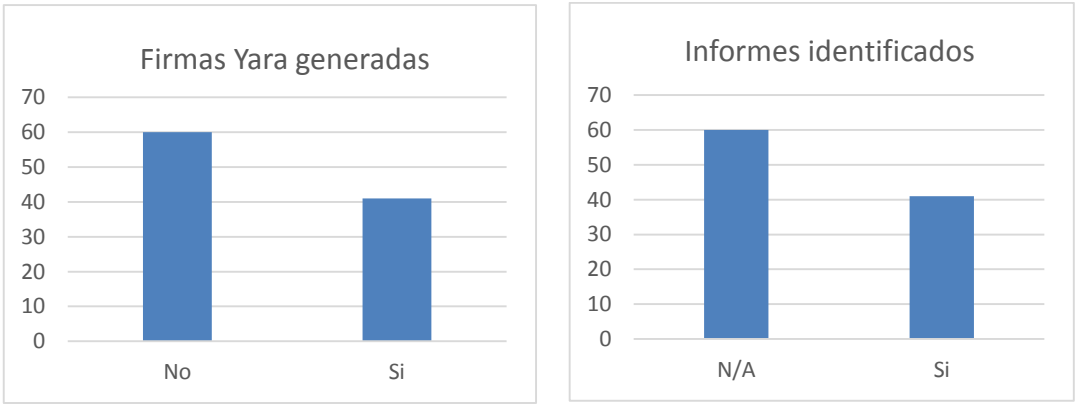

Ilustración 16. Firmas generadas e informes identificados

Como puede observarse en la ilustración número catorce, un **6.93%** de las aplicaciones analizadas estaban consideradas como malware. Por otra parte, según se observa en la ilustración número quince, solo se ha conseguido generar la firma de un **40,60%** de las aplicaciones. Este resultado se debe principalmente a dos factores:

- El tamaño máximo que permite Anubis por fichero, **8MB**.
- Problemas con el servicio derivados de la disponibilidad del servicio, principalmente por parte de Anubis. Esto provocó que, durante ciertos

momentos de la ejecución, no se pudiesen obtener los resultados del análisis y por tanto generar la firma, pero debido a que no estaban identificados como malware por VirusTotal no se consideró necesario repetir las pruebas por disponer de una base suficientemente amplia de aplicaciones legítimas.

Tras la primera fase de las pruebas, se han identificado **siete** aplicaciones consideradas como malware, de las cuales se tiene firma de **cinco** de ellas. Además, se han identificado y generado la firma de **treinta y cuatro aplicaciones** que no son consideradas malware. El siguiente paso en las pruebas es ejecutar las firmas generadas y comprobar en qué casos se dan identificaciones positivas, negativas, o falsas identificaciones.

Las pruebas de identificación se basarán en el porcentaje de identificaciones correctas de las firmas sobre los informes obtenidos. Para ello se utilizará la aplicación *Yara*, que dada una firma y un directorio, evaluará todos los archivos dentro del mismo y mostrará para cuales ha tenido un resultado positivo.
Debido a que todas las pruebas se realizarán con las firmas generadas en el paso previo, el resultado esperado es que únicamente exista una identificación positiva por firma, ya que solo debería identificar un informe, perteneciente a la misma aplicación.

En el caso de las firmas reportadas como malware obtenemos una coincidencia del **100%** sin falsos positivos, es decir, que la firma identifica inequívocamente al informe que lo ha generado. Para el caso de las firmas calificadas como no maliciosas

los resultados son los mismos, el **100%** de las firmas identifican los informes generados, sin realizar falsos positivos.

Un análisis manual de las firmas generadas permite identificar de forma sencilla los permisos de la aplicación y las escrituras y lecturas, tanto a nivel de aplicación como de red. Esto agiliza el análisis del comportamiento de la aplicación, ya que se dispone de toda la información relevante estructurada de una forma clara y concisa.

6.2.3. DISCUSIÓN

A pesar de que la precisión de las firmas generadas es muy alta según confirman las pruebas, se debe experimentar con distintos módulos de análisis. Otros analizadores dinámicos como *Droidbox*, u otros servicios on-line pueden ofrecer informes distintos donde la presentación de la información difiera y las firmas no puedan identificar el comportamiento.

Con el propósito de solucionar esto se pueden desarrollar dos alternativas distintas:

- Profundizar en el análisis y la simplificación de la información obtenida en los informes, identificar las operaciones más genéricas y realizar firmas más cortas y genéricas, sin perder el factor de identificación actual.

- Diseñar un módulo que tenga la capacidad de representar toda la información extraída de los servicios de análisis a un formato estándar como MAEC.

Otro factor a valorar y que ha sido demostrado tras realizar las pruebas, es la elevada tasa de fallos derivados de la limitación y la falta de disponibilidad del servicio. Esto en muchas ocasiones supondrá un problema y una limitación de la plataforma. Por ello se debería utilizar una plataforma tipo *Droidbox* que permita no tener restricciones a la hora de realizar el análisis. Además esto también permitirá introducir mecanismos de interacción con la aplicación cuando se realicen las pruebas, con el propósito de evitar que las aplicaciones apliquen técnicas de ocultación al detectar que están siendo analizadas, como ocurre en muchos *malware* actuales.

Una vez realizadas las pruebas de la plataforma para análisis de aplicaciones móviles para Android se puede afirmar que, identificados los problemas y carencias, presenta una base funcional sólida para la identificación de malware una vez que se dispone de un informe generado por un análisis dinámico.

Además, la posibilidad de utilizar el módulo de descargas para la monitorización continua de los canales de distribución alternativos convierte la plataforma en una herramienta para el estudio continuo de malware que facilita el descubrimiento y la clasificación de nuevos comportamientos maliciosos.

6.3. PRUEBAS DE ANÁLISIS DE FICHEROS PARA PLATAFORMAS WINDOWS

Debido a las limitaciones actuales, todas las pruebas realizadas en este apartado harán uso de una plataforma *Windows XP SP3* genérica y *Cuckoo Sandbox* como analizador dinámico para las aplicaciones. Esto, además de simplificar las ejecuciones, permitirá evaluar la fiabilidad de la *sandbox* y estimar si sería posible basar la plataforma en la utilización de este sistema.

Las pruebas se han diseñado en función de tres bloques distintos con el propósito de evaluar diferentes aspectos de los resultados. A continuación se hace una breve descripción sobre cada uno de estos bloques:

- **Pruebas genéricas de identificación sobre malware aleatorio:** Para ello se obtendrá una colección de aplicaciones consideradas maliciosas y se ejecutará la plataforma sobre cada una de ellas. Posteriormente se evaluarán los resultados en función del número de firmas generadas, la cantidad de identificaciones positivas y los falsos positivos.

- **Pruebas específicas de identificación sobre familias de malware:** Con el propósito de conocer si las firmas generadas pueden servir para identificar comportamientos similares se obtendrán muestras de una serie de familias de malware, se ejecutará la plataforma sobre cada una de ellas y posteriormente se evaluará el número de positivos que la firma obtiene para cada familia.

- **Pruebas específicas sobre identificación de CVE´s en malware conocido:** Con el propósito de conocer la capacidad de la plataforma para obtener información sobre el malware, se realizarán una serie de ejecuciones sobre muestras de malware conocidas. En base a los resultados se estimarán la cantidad de información que ha sido capaz de obtener sobre muestras ya identificadas.

Común a estas pruebas, se evaluarán los resultados de la fase de obtención de información en función de los sistemas, aplicaciones y vulnerabilidades que sean encontrados durante esta fase.

6.3.1. PRUEBAS GENÉRICAS DE IDENTIFICACIÓN SOBRE MALWARE ALEATORIO

6.3.1.1. DESCRIPCIÓN DE LAS PRUEBAS

Para estas pruebas se ha realizado una ejecución con 103 aplicaciones catalogadas como malware (ver **Anexo 7**). Con el propósito de obtener una visión general sobre la plataforma, se ha realizado una selección previa de malware que contase con tipos y familias distintos, obtenida de un catálogo con más de 70GB de software identificado como malicioso. Esto permitirá conocer en qué casos resulta más sencillo encontrar e identificar información sobre las aplicaciones analizadas.

Para la automatización del proceso únicamente se ha utilizado un script en *bash* que ejecute la plataforma para cada uno de los ficheros listados en una carpeta. El resto del proceso es totalmente desasistido hasta generar la firma y realizar las

comprobaciones, por lo que se ha conseguido el objetivo de disponer de una plataforma de análisis automático.

6.3.1.2. ANÁLISIS DE LOS RESULTADOS OBTENIDOS

Tras la ejecución se han generado un total de 98 informes de 103 ficheros, lo que supone un **95,14%** de los informes. La realización de estas pruebas ha posibilitado encontrar y aislar algunos errores durante el proceso de generación de firmas que ha obligado a la repetición de las pruebas. Los resultados de identificación obtenidos se muestran en las siguientes gráficas:

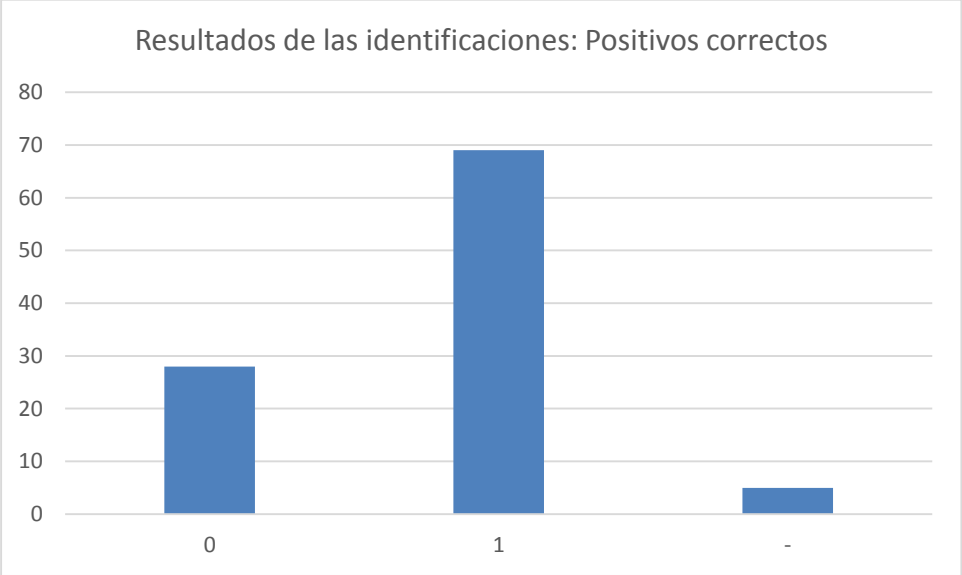

Ilustración 17. Resultados correctos para pruebas genéricas

En base a los datos de la ilustración 17, el **34,95%** de las firmas obtenidas identifican a un único informe. En un **4,85%** de los casos, no se ha podido generar ninguna firma por errores durante el proceso de análisis dinámico en *Cuckoo Sandbox*, por causas ajenas a la plataforma.

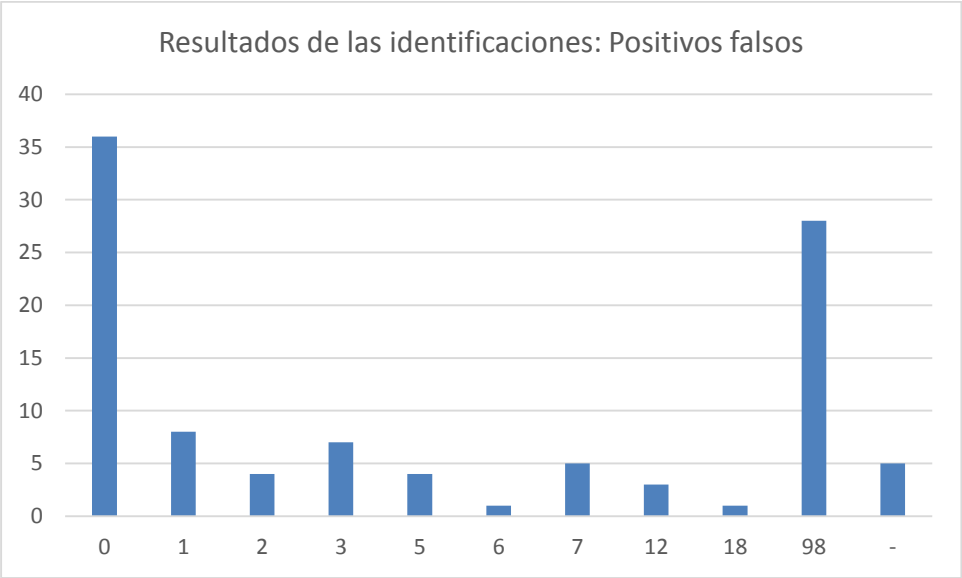

Ilustración 18, Identificaciones falsas en pruebas genéricas

Según la ilustración 18, un **32,03%** de las firmas identifican más de un informe generado. En **28** de los casos, esto es debido a que el análisis dinámico no ha podido obtener información sobre el comportamiento, por tanto se genera una firma vacía que identifica cualquier informe. A pesar de que este tipo de firmas podrían ser descartadas durante su creación, se ha optado por mantenerlas con el propósito de poder identificar de forma clara que análisis dinámicos han fallado.

6.3.1.3. DISCUSIÓN

Aunque se han considerado como positivos falsos, en algunos casos se han identificado comportamientos similares entre distintos tipos de malware. Esto, aun siendo clasificado como un falso positivo, puede considerarse como la identificación correcta de un comportamiento malicioso, por lo que no necesariamente debe invalidar los resultados de esta prueba.

Una vez que se ha identificado esta característica, se debe valorar si la plataforma debe generar firmas más específicas que solo identifiquen el malware con el que se han generado o se pueden permitir firmas de carácter más generales, enfocadas a la identificación de patrones.

6.3.2. PRUEBAS ESPECÍFICAS DE IDENTIFICACIÓN SOBRE FAMILIAS DE MALWARE

6.3.2.1. DESCRIPCIÓN DE LAS PRUEBAS

El propósito de estas pruebas es determinar en qué casos la firma es capaz de identificar más de un informe generado durante el análisis de varias muestras de la misma familia. Por lo general, las familias de malware comparten características y patrones de comportamiento, por lo que estas pruebas pueden ser un indicativo de lo específica o genérica que es la regla en función de los positivos que se obtengan tras su identificación.

Para la realización de estas pruebas se han escogido tres familias que serán detalladas durante el análisis de los resultados.

6.3.2.2. ANÁLISIS DE LOS RESULTADOS OBTENIDOS

6.3.2.2.1. HackTool.Win32.Agent.*

Para las pruebas se han utilizado 72 versiones distintas de esta familia (ver **Anexo 9**). A continuación se muestran las gráficas de dispersión con los resultados obtenidos tras los análisis:

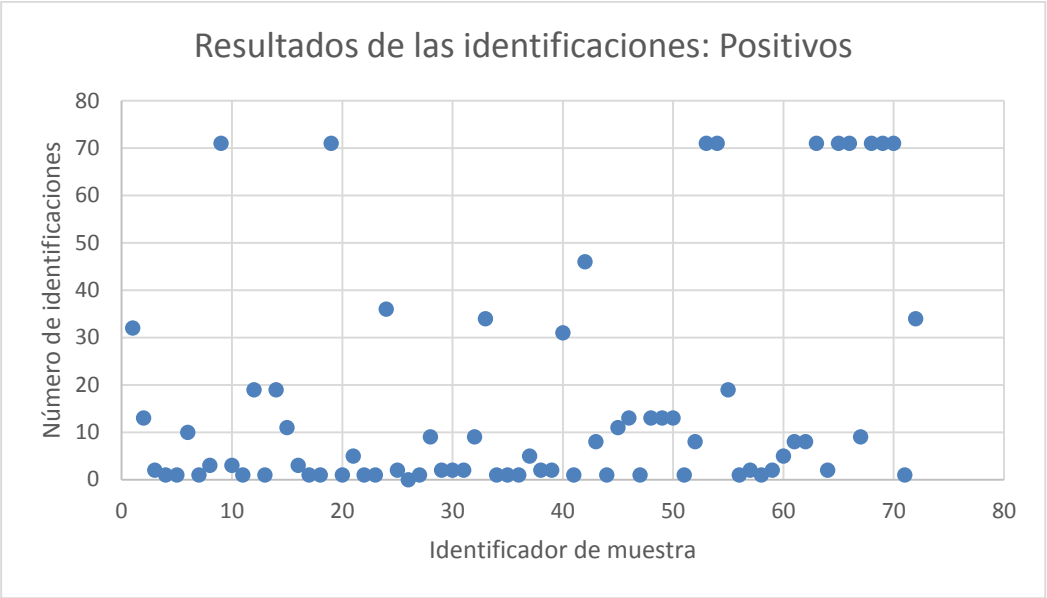

Ilustración 19. Identificaciones positivas en familias: Agent

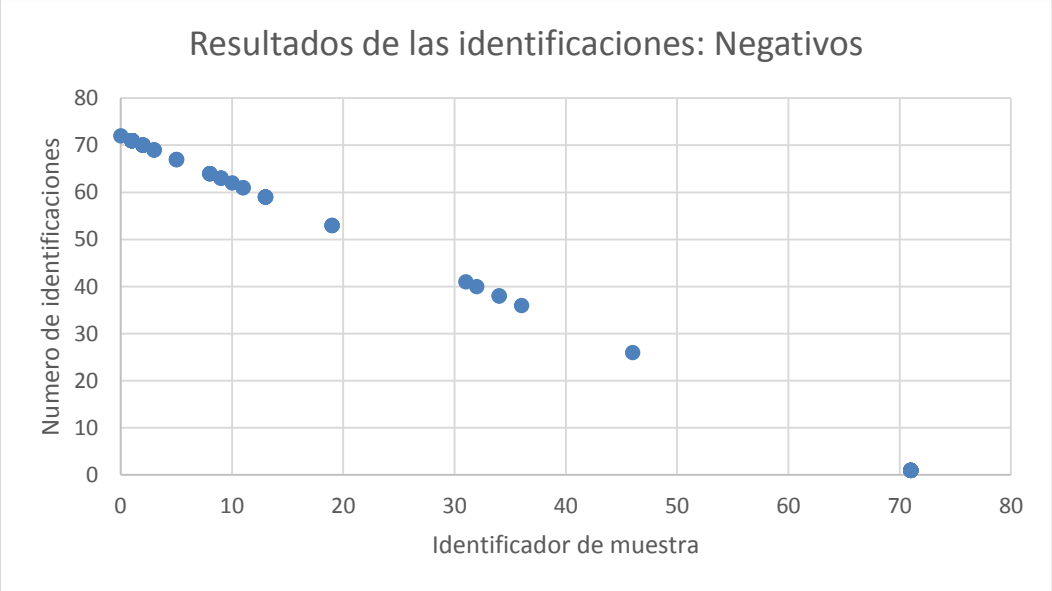

Ilustración 20. Identificaciones negativas en familias: Agent

Según los resultados mostrados en las gráficas anteriores, sobre el total de ficheros analizados se han generado un **98,61%** de informes. Un **29,16%** de las firmas no se han creado correctamente, mientras que un **1,38%** no identifican ningún informe. Del resto de firmas, un **29,16%** solo identifica un informe, mientras que un **55,5%**

identifican más de un informe. La media de identificaciones por firma es de **16,86** informes identificados por cada firma.

Tras analizar los resultados obtenidos, se puede afirmar que, a pesar de considerarse de la misma familia por sus características, en muchos casos los comportamientos entre versiones son distintos y las reglas generadas son demasiado específicas a la hora de identificar librerías y llamadas utilizadas.

6.3.2.2.2. Backdoor.Win32.Bancodor.*

Para las pruebas se han utilizado 23 versiones distintas de esta familia (ver **Anexo 9**). A continuación se muestran las gráficas de dispersión con los resultados obtenidos tras los análisis:

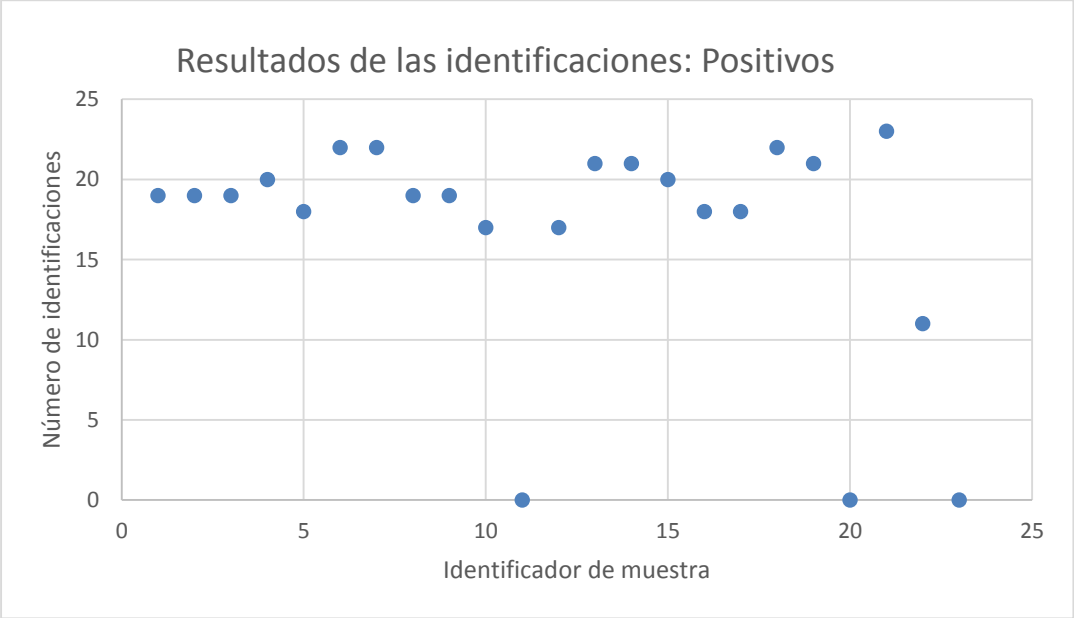

Ilustración 21. Identificaciones positivas en familias: Bancodor

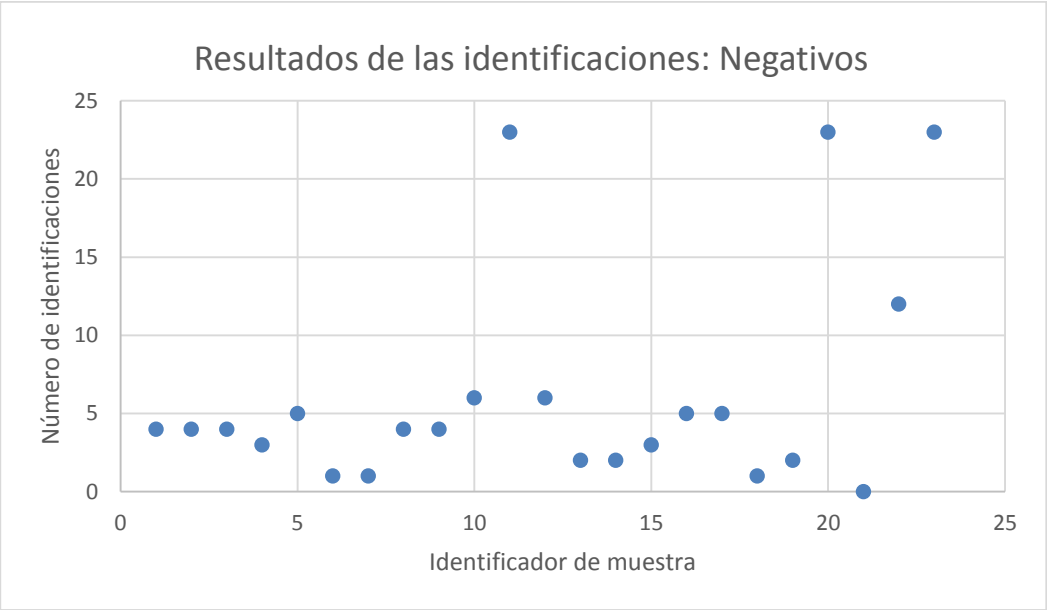

Ilustración 22. Identificaciones negativas en familias: Bancodor

Sobre el total de ficheros analizados se han generado el **100%** de informes. Un **13,04%** de las firmas no se han creado correctamente, mientras que un **4,34%** no identifican ningún informe. Del resto de firmas, un **13,04%** solo identifica un informe, mientras que un **69,56%** identifican más de un informe. En general, la media de identificaciones por firma es de **5,45** informes.

En este caso el número de identificaciones múltiples es mayor que en el anterior ya que no existe tanta variedad y comparten más características de comportamientos.

6.3.2.2.3. Backdoor.Win32.BackEnd.*

Para las pruebas se han utilizado 4 versiones distintas de esta familia (ver **Anexo 9**). A continuación se muestran las gráficas de dispersión con los resultados obtenidos tras los análisis:

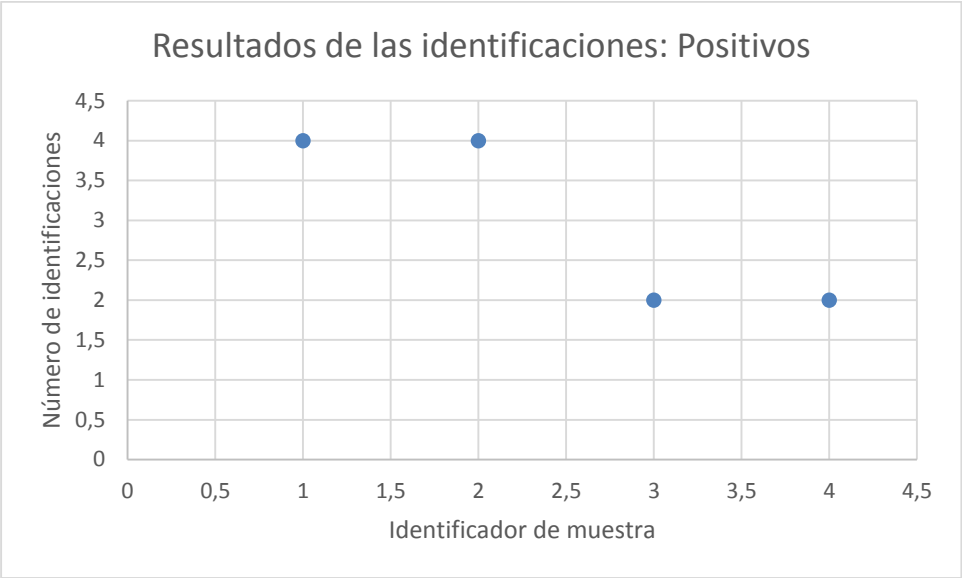

Ilustración 23. Identificaciones positivas en familias: BackEnd

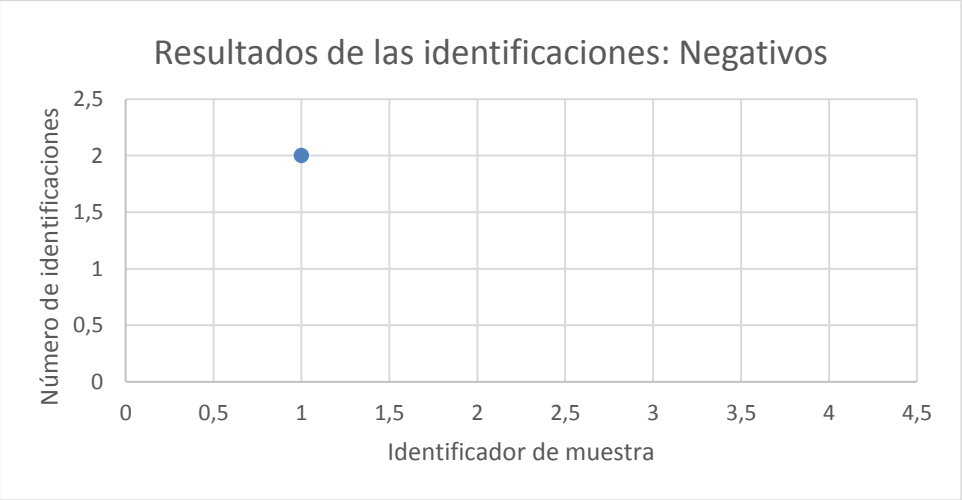

Ilustración 24. Identificaciones negativas en familias: BackEnd

En este caso se comprueba como las versiones más simples del *malware* son capaces de identificar a las más nuevas, pero no al revés. Esto es debido a que las nuevas usan patrones distintos no utilizados por las viejas y no al revés, por lo que con una firma de una versión anterior si se pueden localizar patrones de comportamiento en las nuevas versiones.

6.3.2.3. DISCUSIÓN

Una vez concluido este bloque se puede observar como la tendencia de identificación es similar en todas las pruebas. Hay un mayor número de firmas que identifican más de un malware, lo que significa que incluso entre distintas versiones se observan patrones de comportamiento similar. Esto, si bien puede ser un problema a la hora de identificar una amenaza concreta, puede ser usado en posteriores investigaciones para abstraer la identificación de amenazas a comportamientos y poder aplicar técnicas más avanzadas de detección conjunta con las firmas generadas por la plataforma.

6.3.3. PRUEBAS ESPECÍFICAS SOBRE IDENTIFICACIÓN DE CVE`S EN MALWARE IDENTIFICADO

6.3.3.1. DESCRIPCIÓN DE LAS PRUEBAS

Para la realización de estas pruebas se ha seleccionado una muestra de varios malware identificados y conocidos. El propósito de estas pruebas es evaluar en qué medida es posible identificar los CVE asociados y obtener la configuración del sistema asociada con el propósito de poder generar una plataforma vulnerable que sirva como entorno de análisis.

Además de realizar el proceso de análisis completo, se incluirá una prueba en una plataforma vulnerable y otra en una no vulnerable, con el propósito de analizar los diferentes resultados del comportamiento del malware en cuestión.

Dado que la ejecución de estas pruebas se realizará de forma manual, los resultados obtenidos a través de las distintas ejecuciones serán explicadas, con el propósito de profundizar en la ejecución de la plataforma.

6.3.3.2. ANÁLISIS DE LOS RESULTADOS OBTENIDOS

6.3.3.2.1. 2011-12_CVE-2011-2462

Este fichero obtenido de Contagio [22] dispone de una serie de muestras en formato .pdf. Estas muestras hacen uso de un *exploit* que afecta a la vulnerabilidad CVE-2011-2462 [23]. Para la prueba se escoge aleatoriamente una muestra y se inicia la ejecución de la plataforma.

Tras su análisis en VirusTotal se obtiene el siguiente resultado:

```
Known Aliasses: [Exploit.PDF-U3D.Gen Exploit.PDF-U3D.Gen PDF.Suspicious.Gen
Exploit.Pdf.Pidief.rhedu JS/Pdfka.DT.gen Trojan.Pidief TROJ_PIDIEF.RC1 JS
Exploit.pdf-28742 Exploit.Win32.CVE-2011-2462.b Exploit.PDF-U3D.Gen JS.S.EX-
CVE-2011-2462.711584 Troj/PDFEx-FJ UnclassifiedMalware Exploit.PDF-U3D.Gen
Exploit.PDF.CVE-2011-2462.b (v) EXP/2011-2462.A TROJ_PIDIEF.RC1 Exploit.PDF-
U3D.Gen (B) Trojan/win32.agent Exploit Exploit.PDF-U3D.Gen CVE-2011-
2462!Camelot PDF/Cve-2011-2462 Trojan.Pidief PDF/Exploit.CVE-2011-2462.B.Gen
Exploit.Win32.CVE-2011-2462 W32/CVE_2011_2462.B!exploit Generic5_c.WXT ]
```

La inclusión del CVE en el identificador de malware ofrece una forma sencilla de obtenerlo, por lo que a partir del momento de realizar esta prueba, se añadió una

función para identificar vulnerabilidades en formato CVE en el listado de resultados ofrecido por VirusTotal.

A la vista de los resultados, es un *malware* identificado por muchas compañías, lo cual facilita la búsqueda de información en varias bases de datos que están presentes en la plataforma. Estas son: *"Trend Call", "Sophos", "F-Secure", "ESET" y "McAfee"*.

En este caso, la búsqueda genérica por términos ha encontrado coincidencias con "PDF" y "reader", por lo que a partir de este momento se conoce que es una vulnerabilidad que afecta a software que permite leer formatos *.pdf*. Además, aparecen enumeradas plataformas Windows, desde la versión 95 a la más reciente 7.

Por último en el caso de McAfee se han identificado las siguientes vulnerabilidades:

```
["CVE-2007-5020", "CVE-2007-5659", "CVE-2007-5663", "CVE-2007-5666",
 "CVE-2008-0655", "CVE-2008-0667", "CVE-2008-0726", "CVE-2008-2042",
 "CVE-2008-2992", "CVE-2009-0658", "CVE-2009-0927", "CVE-2009-4324",
 "CVE-2010-0188", "CVE-2010-1297", "CVE-2013-0640", "CVE-2013-0641",
                          "CVE-2011-2462"]
```

A pesar de que el resultado es impreciso y no identifica el CVE detectado, si analizamos cada uno de estos identificadores se puede observar que todos hacen referencia a configuraciones con *adobe reader*, desde versiones desactualizadas como en el caso de los CVE de 2007 hasta las versiones más nuevas en identificadores más recientes. Por tanto en este caso ya se dispone de suficiente información para analizar la muestra de malware en un entorno dinámico.

Este análisis se realiza dos veces. En la primera ocasión no se dispone de ninguna versión de adobe instalada, por lo que, en este caso C*uckoo sandbox*, identifica que no existe un software compatible con la extensión y finaliza el análisis.
Los resultados en este caso son muy negativos. Debido a que no se dispone de ningún informe, únicamente aparecen los *strings* almacenados en el análisis previo y los *tags* de conexiones. Por ello, el análisis con Yara revela un 100% de coincidencias y falsos positivos entre los informes que se están usando como base de pruebas.

Una vez queda comprobada la importancia de disponer de una plataforma vulnerable, se procede a la instalación de una de las versiones *adobe reader* detectada durante el análisis previo para evaluar los resultados.

Esta nueva ejecución se realiza de forma normal y se obtienen todas las llamadas al sistema que realiza, así como algunas librerías y peticiones de red. Estos datos proporcionan suficiente información para generar una firma identificativa que no obtiene falsos positivos.
Además se procede a analizar el fichero, en este caso *.pdf*, para comprobar si es posible identificarlo mediantes las *strings* obtenidas de forma aleatoria.

En este caso, las pruebas realizadas sobre las distintas muestras obtenidas del malware analizado demuestran que únicamente se identifica el malware del cual ha sido generada la firma. Esto se debe a que, a pesar de que comparten vulnerabilidad, son familias distintas de malware, con comportamientos distintos, como se puede deducir tras analizar cada una de las muestras.

Durante las pruebas se han tenido ciertos problemas al almacenar las *strings* obtenidas en la base de datos, ya que al ser ficheros ofuscados muestran caracteres especiales como ";", lo cual provoca errores durante la inserción o posteriormente al interpretar las reglas. Estos errores deberán depurarse a medida que vayan siendo identificados.

Finalizada esta primera prueba, queda demostrado que la plataforma tiene la capacidad de identificar las vulnerabilidades asociadas e identifica los software y sistemas operativos afectados y generar correctamente una firma que puede ser utilizada para identificar versiones concretas de malware, utilizando tanto el fichero como el informe procedente del análisis dinámico.

6.3.3.2.2. IE-CVE-2012-4969

En este caso el fichero también dispone de una serie de muestras en distintos formatos (exe, swf…) que utilizan la vulnerabilidad identificada como CVE-2012-4969 [24].

La primera prueba se ha realizado con el ejecutable cifrado, técnica ampliamente utilizada para evadir los antivirus y así evitar que sean detectados como maliciosos. Los resultados obtenidos de VirusTotal son los siguientes:

```
Known Aliasses: [Trojan.Gen Agent.AGBWK TROJ_GEN.RCBH1IO
Win32:Agent-AQCE [Trj] Win.Trojan.Agent-838 Trojan.Win32.Agent.tuzg
Trojan.Win32.A.Agent.16896.AF Mal/Generic-S UnclassifiedMalware
Trojan.MulDrop3.64537 TR/Agent.AQCE Trojan.Win32.Agent (A)
Trojan/Win32.Agent Win32:Agent-AQCE Trojan.Gen probably a variant
of Win32/Poison.NKX Trojan.Win32.Agent W32/Agent.TUZG!tr
Trj/Agent.JHT ]
```

En la mayoría de casos se identifica un malware genérico. Esto provoca que en la búsqueda de información posterior solo se consiga recuperar una enumeración de plataformas Windows genérica, sin software afectados ni CVE`s.

Al no tener información suficiente sobre la plataforma vulnerable, la ejecución en *cuckoo sandbox* falla ya que no dispone del recurso al que intenta acceder. En este caso se debería intentar configurar una plataforma genérica de pruebas donde tener versiones conocidas como vulnerables de software considerado como altamente explotable (Oracle java, adobe reader, internet explorer, etc.) con el propósito de obtener alguna información durante el análisis dinámico en caso de ser posible.

Para estas pruebas, y ya que se dispone de una muestra distinta del virus en formato ejecutable, se va a realizar un nuevo análisis con otra versión para comprobar si los resultados ofrecen mayor información. Las identificaciones obtenidas en este caso son las siguientes:

```
Known Aliasses: [TR/Agent.fkj (ES) TROJ_GEN.R06H1IK Win32:Agent-AQCI
[Trj]    TrojWare.Win32.UMal.~A    TR/Agent.fkj    Win32:Agent-AQCI
Trojan.Win32.Agent Trj/Agent.JHT ]
```

Que tampoco logran identificar una configuración válida.

Por último, las pruebas se realizan con la muestra en formato flash (swf) que tiene los siguientes resultados:

```
Known Aliasses: [Trojan-Exploit/W32.SWFlash.13631.UD SWF.Exploit
Exploit-CVE2010-2884 Trojan Trojan Exploit.Swf.Agent.xpohq
Trojan.Swifi Exploit.UL SWF_DROPPR.II SWF:Dropper [Heur]
Win32.Trojan Exploit.SWF-28 Exploit.SWF.Agent.gr Script.SWF.Cxx
SWF.S.Agent.13631 Troj/SWFDL-I UnclassifiedMalware
Exploit:W32/SWFdloader.R Exploit.SWF.225 Trojan.SWF.Generic (v)
EXP/SWF.DI SWF_DROPPR.II Exploit-CVE2010-2884 Script.SWF.Cxx (B)
Exploit.CVE-2012-4969.c Exploit/SWF.Agent Exploit:SWF/ShellCode.G
Script.SWF.Cxx SWF/Exploit Trojan.Swifi SWF/Exploit.Agent.EL
Exploit.SWF.Agent SWF/Agent.73A7!exploit Exploit_c.VQP
Exploit/CVE-2012-4969 ]
```

En este caso se identifica correctamente el CVE. Buscando información sobre este CVE se obtiene el CPE asociado, por lo que ya se dispone de una configuración válida para realizar el análisis dinámico.

Tras obtener el análisis dinámico se genera la firma y se realizan las pruebas de identificación, siendo positivas tanto para el informe como para el ejecutable.

Como conclusión, en este caso se puede observar como el análisis previo para obtener información no siempre será posible, por lo que en ocasiones se tendrán que utilizar máquinas con configuraciones aleatorias o proceder a un análisis manual en profundidad.

6.3.3.2.3. Flu Project

Flu [25] es un troyano diseñado por un grupo de investigadores españoles con el propósito de concienciar sobre los peligros de internet. Además su última versión está siendo utilizada por las fuerzas y cuerpos de seguridad de algunos países latinoamericanos en la lucha contra la ciberdelincuencia.

En base a sus características y funcionamiento, está considerado como Dropper ya que, una vez ejecutado, realiza una serie de conexiones para descargar los módulos que necesita para desarrollar su actividad. Por tanto en este caso no se tendrá un identificador ni ningún dato característico ya que no utiliza ninguna vulnerabilidad concreta en el sistema.

Tras obtener el informe de análisis de VirusTotal se tienen los siguientes resultados:

```
Known Aliasses: [Trojan.Generic.8423257 Trojan-
Dropper/W32.Dapato.30557 RDN/Generic Dropper!gd Trojan.Agent
Trojan Trojan Trojan/Spy.Agent.bv Trojan.Gen Troj_Generic.IZHJS
TROJ_GEN.R47COCL MSIL:Keylog-A [Trj] Trojan.Kuluoz-1078 Trojan-
Ransom.Win32.Blocker.axpl Trojan.Generic.8423257
UnclassifiedMalware Trojan.Generic.8423257
Trojan.DownLoader6.2376 Trojan.Win32.Generic!BT TR/Dropper.Gen
TROJ_GEN.R47COCL RDN/Generic Dropper!gd Trojan.Generic.8423257
(B) TrojanDropper.Dapato.idq Win32.Troj.Dapato.(kcloud)
Trojan.Generic.8423257 W32/Trojan.ZRWM-0816 Dropper/Win32.Dapato
TrojanDropper.Dapato Trojan.Gen MSIL/Spy.Agent.BV Trojan-Dropper
MSIL/Agent.BV!tr.spy Dropper.Generic5.COOZ Generic Malware ]
```

Tras realizar las consiguientes búsquedas, la única información que se ha obtenido es la referente a la plataforma, ya que ataca a todas las posibles plataformas de Windows. En este punto lo único que se puede hacer es realizar el análisis dinámico para observar el comportamiento de la muestra y, en base a los resultados, decidir si el malware ha realizado su actividad o la plataforma no era vulnerable, y por tanto no se ha encontrado suficiente información tras realizar las búsquedas.

Una vez realizado el análisis se puede observar como la muestra realiza distintas conexiones y carga una serie de librerías, creando nuevos ficheros y entradas de registro, por lo que se considera que la muestra ha sido capaz de ejecutarse y la plataforma es vulnerable. Con ello podemos afirmar que disponemos de una firma que tiene la capacidad de identificar tanto el informe dinámico generado como el ejecutable analizando los *strings* que existen.

Como conclusión para esta prueba podría afirmarse que si bien el sistema ha funcionado e identificado la muestra, el problema de no encontrar suficiente información tiene una solución difícil, ya que en muchos casos no se sabrá si esto es que la muestra afecta de forma genérica o realmente es una falta de información al respecto del malware en cuestión. La única solución es disponer de entornos genéricos que permitan ejecutar ciertas pruebas y en base a los resultados catalogar el malware dentro de una serie de familias con características concretas, que permitan obtener unos criterios más consistentes a la hora de decidir si la información encontrada es suficiente.

6.3.3.3. DISCUSIÓN

Una vez concluido el bloque de pruebas se puede determinar que la plataforma cumple con los requisitos sobre obtención de información y generación de una firma que permita identificar comportamientos.

Los resultados presentados se encuentran dentro de los límites esperados ya la automatización de resultados en base a un análisis resulta una tarea complicada que tiene muchos factores.

Como punto negativo se debería destacar la alta tasa de fallos producida por *Cuckoo sandbox*. Tras estudiar estos problemas se ha determinado que en su mayoría son producidos por bloqueos durante el análisis o procesos que no pueden ser analizados, lo cual pone de manifiesto la importancia de disponer de una plataforma de análisis dinámico eficaz.

6.4. CONCLUSIONES

Tras la realización de las pruebas, **223** muestras de malware diferentes han sido evaluadas por la plataforma. De estas, solo en un **4.1%** de los casos se ha podido identificar un CVE. En un **25%** de los casos se ha logrado identificar la plataforma vulnerable y en un **16,6%** software vulnerable.

Estos datos se ajustan a lo esperado ya que, tal y como se explicó durante el análisis del problema, la cantidad de información pública sobre las características del *malware* es insuficiente. Aun así, en los casos en los que se ha logrado identificar una plataforma vulnerable, se ha podido demostrar que este hecho marca la diferencia entre poder extraer e identificar patrones de las muestras o no poder hacerlo.

Otro factor identificado durante el análisis previo es que la mayoría de *malware* no utiliza vulnerabilidades en las aplicaciones o en el sistema, si no que requiere de la ejecución directa del usuario. Dado que no se requieren configuraciones específicas de plataformas, más allá del sistema operativo, esto justifica la escasa información referente a CVE´s y software.

En cuando a la cantidad de firmas generadas, la plataforma depende en gran medida de la calidad de las soluciones de análisis dinámico. Sin unos buenos resultados obtenidos durante los análisis, la generación de firmas es imprecisa, por lo que no logra identificar el comportamiento del *malware* estudiado. De esta forma se demuestra que cada uno de los módulos, aunque independiente, es crucial para el siguiente, lo cual provoca que la plataforma sea compleja de mantener y desarrollar.

Finalmente, aunque las firmas han generado más falsos positivos de los esperados, analizando con detalle los informes se ha llegado a la conclusión de que, a pesar de ser familias distintas, en muchos casos las muestras compartían patrones comunes, lo que provoca que los patrones más básicos puedan identificar a otros más avanzados.

La realización de estas pruebas ha contribuido de forma activa al desarrollo de la plataforma ya que ha permitido la identificación de algunos problemas y fallos derivados de su utilización que no habían sido tenidos en cuenta con anterioridad.

7. GESTIÓN DEL PROYECTO

7.1. COSTES

En esta sección se detallarán tanto los costes estimados como los costes finales una vez realizado el proyecto.

7.1.1. COSTES ESTIMADOS

La estimación de costes es crucial para el desarrollo de un proyecto ya que, sin ella, en muchos casos no se dispondría de la financiación necesaria para llevar a cabo el proyecto ni podría ofrecerse a potenciales inversores y compradores.

Para la estimación de estos costes, se debe tener en cuenta los gastos de personal, de hardware, de software y los gastos indirectos derivados. A continuación se ofrece un desglose de todos ellos seguido del coste total que representaría el proyecto.

Apellidos, Nombre	Categoría	Horas	Coste	Coste Total
Ortega Fernández, Sergio	Analista, Programador, Jefe de Proyecto	360	55.5€	19.980€
			55.5€	**19.980€**

Tabla 63. Costes de personal

Descripción	Coste (Euros)	%Uso dedicado proyecto	Dedicación (meses)	Coste Imputable
Plataforma de virtualización (Intel Q9550, 12GB RAM)	850€	20%	4	170€
				170€

Tabla 64. Costes de Hardware

Descripción	Coste (Euros)	%Uso dedicado proyecto	Dedicación (meses)	Coste Imputable
VMWare Workstation 9.2	193,5€	80%	4	154,8€
Microsoft Office 2012	99€	40%	4	39,6€
Microsoft Project 2012	775€	100%	4	775€
Ubuntu 12.07	0€	100%	4	0€
Windows 8 Professional	90€	10%	4	9€
				978,4€

Tabla 65. Costes de Software

Descripción	Coste (Euros)	%Uso dedicado proyecto	Dedicación (meses)	Coste Imputable
Luz	67€	10%	4	6,7€
Internet	51,9€	10%	4	5,9€
				12,6€

Tabla 66. Costes indirectos

Descripción	Costes (Euros)
Costes de personal	19.980€
Costes de Software	978,4€
Costes de Hardware	170€
Costes indirectos	12,6€
Subtotal	**21.141€**
IVA (21%)	4.439,61€
Margen de beneficios (15%)	3.171,15€
Margen de riesgo (10%)	2.114,1€
Total	**30.865,86**

Tabla 67. Costes totales estimados

7.1.1. COSTES REALES

En esta sección se detallan los costes reales que ha tenido el desarrollo una vez completado.

Apellidos, Nombre	Categoría	Horas	Coste	Coste Total
Ortega Fernández, Sergio	Analista, Programador, Jefe de Proyecto	480	47,5€	22.800€
			47, 5€	**22.800€**

Tabla 68. Costes de personal

Descripción	Coste (Euros)	%Uso dedicado proyecto	Dedicación (meses)	Coste Imputable
Plataforma de virtualización (Intel Q9550, 12GB RAM)	850€	30%	5	255€
				255€

Tabla 69. Costes de Hardware

Descripción	Coste (Euros)	%Uso dedicado proyecto	Dedicación (meses)	Coste Imputable
VMWare Workstation 9.2	193,5€	80%	5	154,8€
Microsoft Office 2012	99€	40%	5	39,6€
Microsoft Project 2012	775€	100%	5	775€
Ubuntu 12.07	0€	100%	5	0€
Windows 8 Professional	90€	10%	5	9€
				978,4€

Tabla 70. Costes de Software

Descripción	Coste (Euros)	%Uso dedicado proyecto	Dedicación (meses)	Coste Imputable
Luz	67€	40%	5	26,8€
Internet	51,9€	10%	5	5,9€
				32,7€

Tabla 71. Costes indirectos

Descripción	Costes (Euros)
Costes de personal	22.800€
Costes de Software	978,4€
Costes de Hardware	255€
Costes indirectos	32,7€
Subtotal	**24.066,1€**
IVA (21%)	5.053,8 €
Margen de beneficios (15%)	3.609,9€
Margen de riesgo (10%)	2.406,61€
Total	**35.136,41€**

Tabla 72. Costes totales estimados

Una vez analizados los costes, se debe analizar la desviación y, en caso de que exista, analizar los motivos por los cuales se ha tenido este error. Este proceso permitirá retroalimentar la gestión del proyecto y mejorar la eficiencia del proceso en futuros proyectos.

En este caso los análisis demuestran que si bien las estimaciones han sido bastante acertadas, se ha producido un aumento del tiempo empleado y la duración del proyecto, lo que ha repercutido en una bajada del precio por horas para ajustar el presupuesto, teniendo en cuenta la actual condición socio-económica de España.

7.2. PLANIFICACIÓN: DIAGRAMAS DE GANTT

7.2.1. PLANIFICACIÓN ESTIMADA

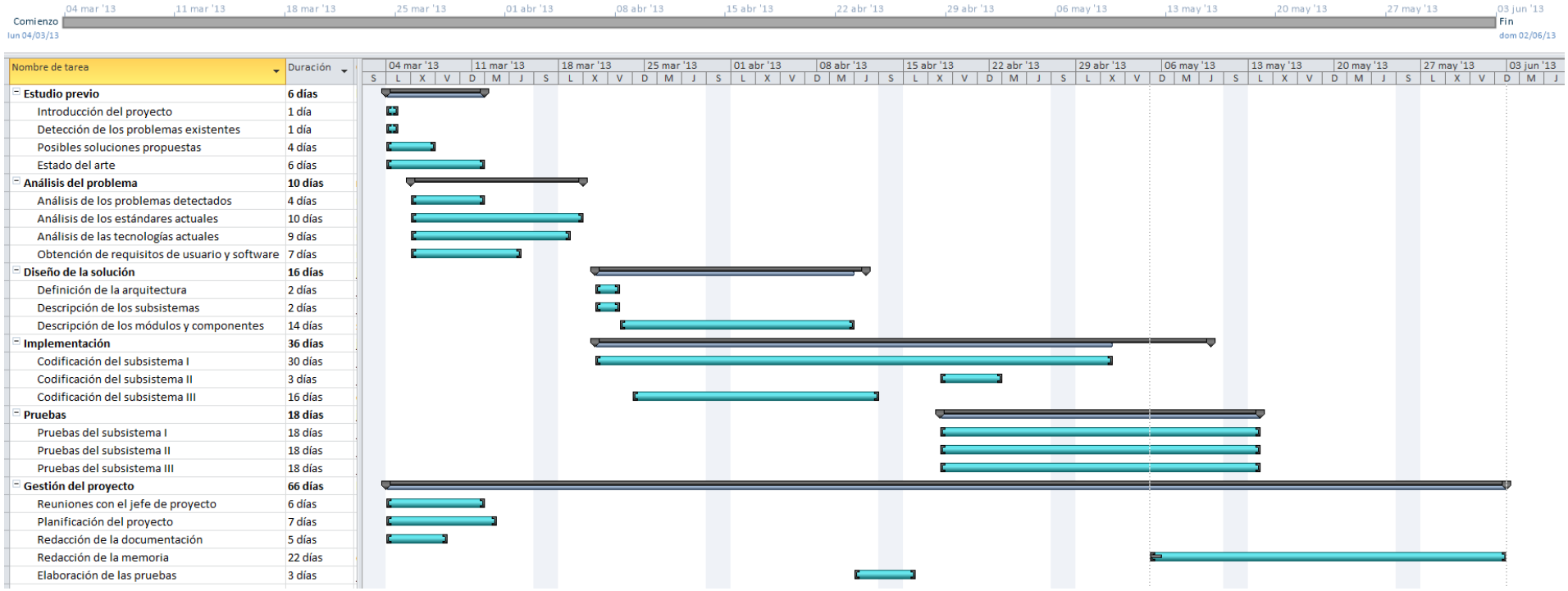

Ilustración 25. Planificación estimada

7.2.2. PLANIFICACIÓN REAL

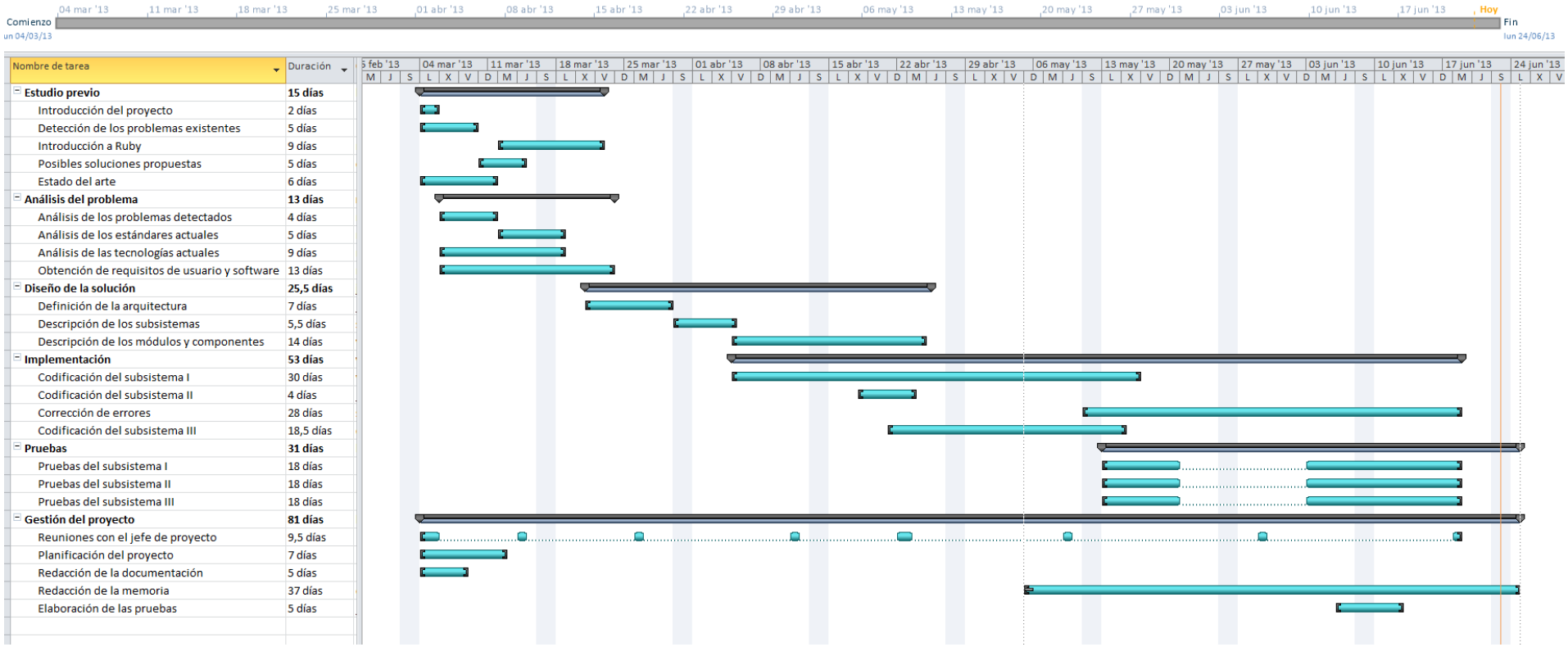

Ilustración 26. Planificación real

8. CONCLUSIONES Y LÍNEAS FUTURAS

En esta sección se expondrán las conclusiones obtenidas sobre el desarrollo de la plataforma así como las líneas futuras de trabajo sugeridas tras el estudio de la misma.

8.1. CONCLUSIONES

La plataforma de análisis dinámico de malware en entornos controlados surge tras analizar las necesidades que existen sobre la detección de aplicaciones maliciosas.

En la actualidad el análisis estático no puede ser considerado la solución óptima ya que existen muchos factores que dificultan su realización, por ello el desarrollo de esta plataforma propone como solución los siguientes puntos:

1. Obtener toda la información pública sobre un *malware* identificado que esté disponible.
2. Utilizar esta información para la creación automatizada de entornos de análisis donde ejecutar las muestras.
3. Hacer uso de herramientas de análisis dinámico que permitan, mediante la ejecución de las muestras, obtener información sobre su comportamiento.
4. Agregar y estandarizar toda la información, con el propósito de evitar la actual situación donde cada herramienta representa la información de una forma distinta.
5. Utilizar toda la información recogida durante el proceso para realizar firmas de comportamiento y sistemas de clasificación automática.
6. Compartir toda la información obtenida mediante bases de datos públicas de referencia y herramientas que permitan el acceso fácil a la plataforma.

En base a los resultados de las pruebas se puede afirmar que los objetivos propuestos para este proyecto han sido cumplidos en su mayoría. La plataforma es capaz de realizar el proceso de análisis de forma desatendida y, si bien los resultados en la identificación no son todo lo exactos que se esperaba, esto podría permitir la reorientación de los objetivos y diseñar una plataforma de identificación y clasificación de comportamientos maliciosos que pudiera ser integrada dentro de otros sistemas de correlación de eventos.

A modo de comentario personal sobre el proyecto puedo afirmar que estoy satisfecho con el resultado final obtenido y todo el proceso de aprendizaje y trabajo que conlleva. La realización de este proyecto me ha permitido conocer de primera mano las diversas técnicas y herramientas que son utilizadas en el análisis de *malware* y profundizar en diversas materias como el análisis de comportamientos o la gestión y documentación.

Además buscaba la oportunidad de utilizar tecnologías y recursos no vistos durante la carrera, con el propósito no solo de afianzar los conocimientos obtenidos durante estos años, sino también de poder ampliar los conocimientos y habilidades de cara a una futura inserción laboral. Esto se ha conseguido mediante la utilización de lenguajes como *Ruby* o *Python*, *git* para la gestión del proyecto, *Cuckoo Sandbox* para el análisis dinámico, distintos estándares de información que cada vez se aplican más en el mundo de la seguridad informática, etc…

Espero que esta plataforma sirva de base para futuros desarrollos y mejoras, con líneas de investigación similares a las propuestas a lo largo de este trabajo.

8.2. LÍNEAS FUTURAS

A continuación se describen de forma breve una serie de propuestas y líneas de trabajo para continuar con el desarrollo de la plataforma, de forma que permita mejorar los resultados presentados en esta memoria.

8.2.1. AGREGACIÓN DE LA INFORMACIÓN DE DISTINTAS FUENTES EN FORMATOS ESTANDAR

El objetivo de esta línea es implementar un mayor número de módulos de análisis y desarrollar un sistema que permita agregar toda la información recogida de cada uno de ellos, seleccionarla y posteriormente representarla en un formato estándar con el objetivo de permitir al máximo la interoperabilidad entre sistemas.

Esto permitiría a la plataforma evaluar informes en distintos formatos y obtener una salida independientemente de la fuente, lo que supondría una mejora sustancial sobre lo propuesto en la actualidad.

8.2.2. MEJORAR LA IDENTIFICACIÓN DE COMPORTAMIENTOS

La mejora de este proceso permitiría a la plataforma abstraerse de la identificación de *malware* y comenzar a identificar patrones de comportamiento. Esto sería posible aplicando técnicas de inteligencia artificial y correlación, de forma que la generación de firmas se realizase de forma más general, incidiendo en la capacidad de identificación que posee.

Esta línea de trabajo repercutiría positivamente en los resultados de identificación obtenidos durante las pruebas.

8.2.3. CLASIFICACIÓN AUTOMATIZADA CON TÉCNICAS DE MACHINE LEARNING.

Una vez que se han desarrollado las herramientas para permitir la clasificación automatizada es necesario llevar a cabo la integración con sistemas expertos que permitan la clasificación de la muestra en función de su comportamiento.

Este proceso no se ha podido abordar durante esta etapa del desarrollo por el alto número de muestras que deben procesarse con el propósito de *entrenar* el sistema para ajustar los resultados.

Esta línea de trabajo puede ser la más interesante a corto plazo ya que, con ella, la plataforma no solo podría identificar el comportamiento si no que podría ser integrada con *crawlers* y ser destinada a la identificación y clasificación *in the wild* de *malware.*

9. REFERENCIAS

[1] - LivingSocial Hacked: Cyber Attack Affects More Than 50 Million Customers [http://abcnews.go.com/Technology/livingsocial-hacked-50-million-customers-data-compromised/story?id=19057439, Marzo 2013]

[2] - Espía WhatsApp [http://www.securitybydefault.com/2012/11/espia-whatsapp.html, Marzo 2013]

[3] – First Quaerter Zero-Day vulnerabillities [http://www.symantec.com/connect/blogs/2013-first-quarter-zero-day-vulnerabilities, Junio 2013]

[4] – Sophos, Security Thread Report 2013 [http://www.sophos.com/en-us/security-news-trends/reports/security-threat-report/android-malware.aspx, Junio 2013]

[5] - 15ª encuesta AIMC a usuarios de Internet [http://download.aimc.es/aimc/4uT43Wk/macro2012.pdf, Abril 2013]

[6] - Making Security Measurable [http://measurablesecurity.mitre.org/, Marzo 2013]

[7] - MITRE : Common Platform Enumeration [http://cpe.mitre.org/, Marzo 2013]

[8] - MITRE : Common Vulnerabilities and Exposures [http://cve.mitre.org/, Marzo 2013]

[9] - National Vulnerability Database [http://nvd.nist.gov/, Marzo 2013]

[10] - Common Vulnerability Scoring System

[http://www.first.org/cvss, Abril 2013]

[11] - Malware Attribute Enumeration and Characterization

[http://maec.mitre.org/, Abril 2013]

[12] – Common Attack Pattern Enumeration and Classification

[http://capec.mitre.org/, Marzo 2013]

[13] – MISP Malware Information Sharing

[http://christophe.vandeplas.com/2013/03/misp-malware-information-sharing.html, Abril 2013]

[14] – Anubis Malware Analyzer

[http://anubis.iseclab.org/, Marzo 2013]

[15] – Comodo Analyzer

[http://camas.comodo.com/cgi-bin/submit, Marzo 2013]

[16] – Eureka Analyzer

[http://eureka.cyber-ta.org/, Marzo 2103]

[17] - Cuckoo Sandbox

[http://www.cuckoosandbox.org/, Marzo 2013]

[18] – Toward extracting malware features for classification using static and dynamic analysis

[http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=6418637&contentType=Conference+Publications, Abril 2013]

[19] – Automatic Analysis of Malware Behavior using Machine Learning

[http://www.sunbeltsoftware.com/documents/sunbelt-cwsandbox-malheur.pdf, Junio 2013]

[20] Yara Project

[https://code.google.com/p/yara-project/, Mayo 2013]

[21] Ruby

[http://www.ruby-lang.org/es/, Marzo 2013]

[22] Contagio Dump

[http://contagiodump.blogspot.com.es/, Mayo 2013]

[23] CVE-2011-2462

[http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-2462, Mayo 2013]

[24] CVE-2012-4969

[http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-4969, Mayo 2013]

[25] Flu Project

[http://www.flu-project.com/, Marzo 2013]

[26] Bottle

[http://bottlepy.org/docs/dev/, Junio 2013]

10. ANEXOS

10.1. ANEXO 1: INFORME MAEC GENERADO POR CUCKOO

```
<?xml version='1.0' ?>
<!--
Cuckoo Sandbox MAEC 1.1 malware analysis report
http://www.cuckoosandbox.org
-->
<MAEC_Bundle
xsi:schemaLocation='http://maec.mitre.org/XMLSchema/maec-core-1
file:MAEC_v1.1.xsd'
id="maec:d9cc204b591641e960f75f71ee5ad8ee:bnd:1"
schema_version="1.100000">
    <Analyses>
        <Analysis start_datetime="2013-06-17T02:01:37"
lastupdate_datetime="2013-06-17T02:01:59"
id="maec:d9cc204b591641e960f75f71ee5ad8ee:ana:1"
complete_datetime="2013-06-17T02:01:59" analysis_method="Dynamic">
            <Subject>
                <Object_Reference type="Object"
object_id="maec:d9cc204b591641e960f75f71ee5ad8ee:obj:1"/>
            </Subject>
            <Tools_Used>
                <Tool
id="maec:d9cc204b591641e960f75f71ee5ad8ee:tol:1">
                    <Name>Cuckoo Sandbox</Name>
                    <Version>0.5</Version>

<Organization>http://www.cuckoosandbox.org</Organization>
                </Tool>
            </Tools_Used>
        </Analysis>
    </Analyses>
    <Behaviors/>
    <Actions>
        <Action successful="true" timestamp="2013-06-17
00:40:49,678" ordinal_position="1"
id="maec:211230d9749744596c1ee69be1e37876:act:1">
            <Action_Initiator type="Process">
                <Initiator_Object type="Object"
object_id="maec:211230d9749744596c1ee69be1e37876:obj:1"/>
            </Action_Initiator>
            <Action_Implementation type="API_Call"
id="maec:211230d9749744596c1ee69be1e37876:imp:1">
                <API_Call
apifunction_name="NtAllocateVirtualMemory"
id="maec:211230d9749744596c1ee69be1e37876:api:1">
                    <ReturnValue>0x00000000</ReturnValue>
                    <APICall_Parameter ordinal_position="1">
                        <Name>ProcessHandle</Name>
                        <Value>0xffffffff</Value>
                    </APICall_Parameter>
                    <APICall_Parameter ordinal_position="2">
```

```
                        <Name>BaseAddress</Name>
                        <Value>0x00b00000</Value>
                    </APICall_Parameter>
                    <APICall_Parameter ordinal_position="3">
                        <Name>RegionSize</Name>
                        <Value>0x00010000</Value>
                    </APICall_Parameter>
                    <APICall_Parameter ordinal_position="4">
                        <Name>Protection</Name>
                        <Value>0x00000004</Value>
                    </APICall_Parameter>
                </API_Call>
            </Action_Implementation>
        </Action>
        <Action successful="true" timestamp="2013-06-17
00:40:49,688" ordinal_position="2"
id="maec:211230d9749744596c1ee69be1e37876:act:2">
            <Action_Initiator type="Process">
                <Initiator_Object type="Object"
object_id="maec:211230d9749744596c1ee69be1e37876:obj:1"/>
            </Action_Initiator>
            <Action_Implementation type="API_Call"
id="maec:211230d9749744596c1ee69be1e37876:imp:2">
                <API_Call
apifunction_name="NtAllocateVirtualMemory"
id="maec:211230d9749744596c1ee69be1e37876:api:2">
                    <ReturnValue>0x00000000</ReturnValue>
                    <APICall_Parameter ordinal_position="1">
                        <Name>ProcessHandle</Name>
                        <Value>0xffffffff</Value>
                    </APICall_Parameter>
                    <APICall_Parameter ordinal_position="2">
                        <Name>BaseAddress</Name>
                        <Value>0x00b00000</Value>
                    </APICall_Parameter>
                    <APICall_Parameter ordinal_position="3">
                        <Name>RegionSize</Name>
                        <Value>0x00001000</Value>
                    </APICall_Parameter>
                    <APICall_Parameter ordinal_position="4">
                        <Name>Protection</Name>
                        <Value>0x00000004</Value>
                    </APICall_Parameter>
                </API_Call>
            </Action_Implementation>
        </Action>
   </actions>
    <Pools>
        <Object_Pool>
            <Object object_name="Trojan-Spy.Win32.CashBack.a"
type="File" id="maec:d9cc204b591641e960f75f71ee5ad8ee:obj:1">
                <Object_Size units="Bytes">28672</Object_Size>
                <File_System_Object_Attributes>
                    <Hashes>
                        <Hash type="MD5">

<Hash_Value>d9cc204b591641e960f75f71ee5ad8ee</Hash_Value>
                        </Hash>
```

```
                        <Hash type="SHA1">

<Hash_Value>956d5627a7dde48f858be204dc0de798bc059e40</Hash_Value>
                        </Hash>
                    </Hashes>
                    <File_Type type="PE32 executable (GUI) Intel
80386, for MS Windows"/>
                    <File_Type_Attributes>
                        <PE_Binary_Attributes dll_count="9">
                            <Version_Block>

<Product_Name>cashback</Product_Name>

<Company_Name>(\xc8fc)\xd2f0\xc564\xc5d8</Company_Name>
                                <Product_Version_Text>1, 0, 0,
1</Product_Version_Text>
                                <File_Version_Text>1, 1, 4,
3</File_Version_Text>

<Original_File_Name>csbkupd.exe</Original_File_Name>
                            </Version_Block>
                            <Imports>
                                <Import>

<File_Name>SHLWAPI.dll</File_Name>
                                    <Imported_Functions>
                                        <Imported_Function>

<Function_Name>StrFormatByteSizeA</Function_Name>

<Virtual_Address>0x40540c</Virtual_Address>
                                        </Imported_Function>
                                    </Imported_Functions>
                                </Import>
                            </Imports>
                            <Sections>
                                <Section>

<Section_Name>.text</Section_Name>
                                    <Entropy>5.981973</Entropy>

<Virtual_Address>0x1000</Virtual_Address>

<Virtual_Size>12900</Virtual_Size>
                                </Section>
                            </Sections>
                        </PE_Binary_Attributes>
                    </File_Type_Attributes>
                </File_System_Object_Attributes>
            </Object>
        </Object_Pool>
    </Pools>
</MAEC_Bundle>
```

10.2. ANEXO 2: INFORME XML GENERADO POR ANDRUBIS

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/REC-html40/loose.dtd">
<?xml version="1.0" encoding="ISO-8859-1"?><html>
<body>
<analysis api-level="3" file-size="196522"
md5="9f90699f056768fe4d94907bc801db42" name="Superuser.apk" report-
version="0.5" sha1="27f2052daff6704763c112297a6aa1cd353c1e01">
<report_version>
   <major>0</major>
   <minor>5</minor>
</report_version>
<static-analysis>
   <permissions>
   <permission>android.permission.WRITE_EXTERNAL_STORAGE</permission>
         <permission>android.permission.INTERNET</permission>
   <permission>android.permission.ACCESS_NETWORK_STATE</permission>
         <permission>com.noshufou.android.su.RESPOND</permission>
   </permissions>
<enforced-permissions>
<permission>com.noshufou.android.su.RESPOND</permission>
</enforced-permissions>
<broadcast-receivers>
<broadcast-receiver
action="com.noshufou.android.su.REQUEST">SuRequestReceiver</broadcast-
receiver>
<broadcast-receiver
action="com.noshufou.android.su.NOTIFICATION">SuNotificationReceiver</br
oadcast-receiver>
<broadcast-receiver
action="android.intent.action.PACKAGE_REMOVED">UninstallReceiver</broadc
ast-receiver>
<broadcast-receiver
action="android.intent.action.PACKAGE_ADDED">InstallReceiver</broadcast-
receiver>
</broadcast-receivers>
<activities>
<activity action="android.intent.action.MAIN"
exported="true">Su</activity>
<activity exported="false">AppListActivity</activity><activity
exported="false">LogActivity</activity>
<activity action="android.intent.action.MAIN"
exported="true">SuRequest</activity>
<activity action="android.intent.action.VIEW"
exported="true">SuPreferences</activity>
</activities>
<general-apk-info>
<application-name>com.noshufou.android.su</application-name>
<valid_manifest>True</valid_manifest>
<valid_zipfile>False</valid_zipfile>
<valid_androguard_zipfile>True</valid_androguard_zipfile>
<uses-native-code>False</uses-native-code>
<uses-dynamic-code>False</uses-dynamic-code><uses-
reflection>False</uses-reflection>
<uses-crypto>False</uses-crypto>
</general-apk-info>
```

```
   <certificate>
         <owner>CN=Adam Shanks, OU=Android, O=SoupCoconut, L=FWB,
ST=Florida, C=US</owner>
         <issuer>CN=Adam Shanks, OU=Android, O=SoupCoconut, L=FWB,
ST=Florida, C=US</issuer>
         <serial-number>4c614057</serial-number>
         <validity from="Tue Aug 10 12:04:39 GMT 2010" until="Sat Dec 26
12:04:39 GMT 2037"></validity>
         <fingerprints
md5="D2:42:30:AA:BE:81:62:30:FE:B4:0E:F1:CF:11:B0:C0"
sha1="5F:11:3F:C2:C2:0A:7C:9B:D9:28:19:22:6A:32:A1:90:4B:75:EF:8B"></fin
gerprints>
   </certificate>
   <manifest-meta-data>
   </manifest-meta-data>
   <embedded-files>
   </embedded-files>
   <java-packages>
         <package name="com.noshufou.android.su"></package>
   </java-packages>
</static-analysis>

<dynamic-analysis>
   <file-operations>
         <file-read seconds="33.0699968338">
   <path>/data/data/com.android.music/shared_prefs/Music.xml|</path>
               <data>&lt;?xml version='1.0' encoding='utf-8'
standalone='yes' ?&gt;
                     &lt;map&gt;
                     &lt;string name="queue"&gt;&lt;/string&gt;
                     &lt;int name="curpos" value="-1" /&gt;
                     &lt;int name="cardid" value="-1" /&gt;
                     &lt;int name="shufflemode" value="0" /&gt;
                     &lt;int name="repeatmode" value="0" /&gt;
                     &lt;/map&gt;
               </data>
         </file-read>
         <file-write seconds="21.0663559437">
   <path>/data/data/com.noshufou.android.su/shared_prefs/com.noshufou.an
droid.su_preferences.xml|</path>
               <data>&lt;?xml version='1.0' encoding='utf-8'
standalone='yes' ?&gt;
                     &lt;map&gt;
                     &lt;int name="last_run" value="25" /&gt;
                     &lt;/map&gt;
               </data>
         </file-write>
   </file-operations>
   <network-operations>
         <network-opened seconds="33.0697669983">
               <host>dl.dropbox.com</host>
               <port>80</port>
         </network-opened>
         <network-write seconds="33.0701239109">
               <host>dl.dropbox.com</host>
               <port>80</port>
               <data>GET /u/6408470/Superuser/manifest.json HTTP/1.1
```

```
                    User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.4;
generic Build/GRJ22)
                    Host: dl.dropbox.com
                    Connection: Keep-Alive
                    Accept-Encoding: gzip
                </data>
          </network-write>
   </network-operations>
   <started-services>
          <service
seconds="12.0701189041">com.android.vending.util.WorkService</service>
   </started-services>
   <network_traffic_analysis>
          <dns_queries>
                <dns_query dest_ip="10.0.2.3" dest_port="53"
name="dl.dropbox.com" protocol="udp" result="54.225.218.224"
src_ip="10.0.2.15" src_port="60620" successful="1" type="DNS_TYPE_A">
</dynamic-analysis>
<rating model_version="2013-01-08" nr_of_features="22/27"
score="0.0117904"></rating>
</analysis>
</body>
</html>
```

10.3. ANEXO 3: FIRMA YARA GENERADA

```
rule yTrojanWin32Monderbfob
{
      meta:
        description = "N/A"

      strings:
        $connectionsTag = "<Network_Action_Attributes>"
        $localPortTag = "<Internal_Port>"
        $externalPortTag = "<External_Port>"
        $localIP0 = "10.0.2.15"
        $externalIP0 = "10.0.2.255"
        $API_Call0 = "LdrGetProcedureAddress"
        $API_Call1 = "NtAllocateVirtualMemory"
        $API_Call2 = "LdrLoadDll"
        $API_Call3 = "VirtualProtectEx"
        $API_Call4 = "LdrGetDllHandle"
        $API_Call5 = "DeviceIoControl"
        $API_Call6 = "VirtualFreeEx"
        $API_Call7 = "RegCreateKeyExW"
        $API_Call8 = "RegSetValueExW"
        $API_Call9 = "RegCloseKey"
        $API_Call10 = "RegSetValueExA"
        $API_Call11 = "NtCreateMutant"
        $API_Call12 = "RegOpenKeyExW"
        $API_Call13 = "RegQueryInfoKeyW"
        $API_Call14 = "RegEnumValueW"
        $API_Call15 = "RegQueryValueExW"
        $API_Call16 = "NtClose"
        $str_a = "|!$-"
        $str_b = "3  js"
        $str_c = "g!js1"
        $str_d = "j!js8"
        $str_e = "9~cl"
        $str_f = "{&$w"
        $str_g = "d3~d"
        $str_h = "1d%{"
        $str_j = "g!js"
        $str_k = "   jy3"
        $str_l = "|/+&"
        $LdrGe2 = "USER32.dll"
        $LdrGe4 = "WININET.dll"
        $LdrGe6 = "wininet.dll"
        $LdrGe8 = "kernel32.dll"
        $LdrGe9 = "shlwapi.dll"
        $LdrGe11 = "rsaenh.dll"
        $LdrGe15 = "mscoree.dll"
        $LdrLo1 = "msvcrt.dll"
        $LdrLo4 = "WININET.dll"
        $LdrLo9 = "USER32.dll"
        $LdrLo12 = "urlmon.dll"
        $LdrLo15 = "Secur32.dll"

      condition:
        (#connectionsTag >= 1 and #localPortTag >= 1 and
#externalPortTag >= 1
```

```
and 1 of ($localIP*,$externalIP*)
        and #API_Call0 >= 51
        and #API_Call1 >= 45
        and #API_Call2 >= 5 and 1 of ($LdrLo*)
        and #API_Call3 >= 3
        and #API_Call4 >= 5 and 1 of ($LdrGe*)
        and #API_Call9 >= 1067
        and #API_Call10 >= 4
        and #API_Call11 >= 197
        and #API_Call12 >= 66
        and #API_Call13 >= 1
        and #API_Call14 >= 2
        and #API_Call15 >= 20
        and #API_Call16 >= 498
        and #API_Call17 >= 40
        and #API_Call18 >= 4
        and #API_Call19 >= 1861 and 1 of ($RegQu*)
        and #API_Call20 >= 17
        and #API_Call21 >= 199 and 1 of ($NtOpe*)
        and #API_Call22 >= 17
        and #API_Call23 >= 40
        and #API_Call24 >= 45 and 1 of ($NtCre*)
        and #API_Call25 >= 204
        and #API_Call26 >= 1
        and #API_Call27 >= 2
        and #API_Call28 >= 21
        and #API_Call29 >= 12
        and #API_Call30 >= 3
        and #API_Call31 >= 5
        and #API_Call32 >= 1
        and #API_Call33 >= 194
        and #API_Call34 >= 1
        and #API_Call35 >= 38
        and #API_Call44 >= 1006
        and #API_Call45 >= 3
        and #API_Call46 >= 3
        ) or all of ($str_*)
}
```

10.4. ANEXO 4: FIRMA YARA PARA APLICACIONES MÓVILES

```
rule ycomappspace631575921c7f5132c4ece7f8ea52dd9b5a6572ed
{
      meta:
        description = "N/A"

      strings:
        $permission0 = "android.permission.INTERNET"
        $permission1 = "android.permission.ACCESS_NETWORK_STATE"
        $readOp0 = "/dev/urandom|"
        $readOp1 = "/data/data/com.appspace/files/_themes.xml|"
        $readOp2 = "/data/data/com.appspace/files/_themes.xml|"
        $readOp3 =
"/data/data/com.android.mms/shared_prefs/_has_set_default_values.xm
l|"
        $readOp4 =
"/data/data/com.android.mms/shared_prefs/com.android.mms_preference
s.xml|"
        $readOp5 =
"/data/data/com.appspace/shared_prefs/com.appspace.xml|"
        $readOp6 =
"/data/data/com.appspace/app_Parse/applicationId|"
        $writeOp0 =
"/data/data/com.appspace/shared_prefs/com.appspace.xml|"
        $writeOp1 =
"/data/data/com.appspace/shared_prefs/com.appspace.xml|"
        $writeOp2 =
"/data/data/com.appspace/shared_prefs/com.appspace.xml|"
        $writeOp3 =
"/data/data/com.appspace/shared_prefs/com.appspace.xml|"
        $writeOp4 =
"/data/data/com.appspace/shared_prefs/com.appspace.xml|"
        $writeOp5 =
"/data/data/com.appspace/shared_prefs/com.appspace.xml|"
        $writeOp6 =
"/data/data/com.appspace/app_Parse/applicationId|"
        $writeOp7 =
"/data/data/com.appspace/shared_prefs/com.appspace.xml|"
        $writeOp8 =
"/data/data/com.appspace/files/fnt2131099650.otf|"
        $writeOp9 =
"/data/data/com.appspace/files/fnt2131099650.otf|"
        $writeOp10 =
"/data/data/com.appspace/files/fnt2131099650.otf|"
        $writeOp11 =
"/data/data/com.appspace/files/fnt2131099650.otf|"
        $writeOp12 =
"/data/data/com.appspace/files/fnt2131099650.otf|"
        $writeOp13 =
"/data/data/com.appspace/files/fnt2131099648.otf|"
        $writeOp14 =
"/data/data/com.appspace/files/fnt2131099648.otf|"
        $writeOp15 =
"/data/data/com.appspace/files/fnt2131099648.otf|"
        $writeOp16 =
"/data/data/com.appspace/files/fnt2131099648.otf|"
```

```
        $writeOp17 =
"/data/data/com.appspace/files/fnt2131099648.otf|"
        $writeOp18 =
"/data/data/com.appspace/files/fnt2131099649.otf|"
        $writeOp19 =
"/data/data/com.appspace/files/fnt2131099649.otf|"
        $writeOp20 =
"/data/data/com.appspace/files/fnt2131099649.otf|"
        $writeOp21 =
"/data/data/com.appspace/files/fnt2131099649.otf|"
        $writeOp22 =
"/data/data/com.appspace/files/fnt2131099649.otf|"
        $writeOp23 =
"/data/data/com.appspace/files/fnt2131099649.otf|"
        $writeOp24 =
"/data/data/com.appspace/files/fnt2131099649.otf|"
        $writeOp25 =
"/data/data/com.appspace/files/fnt2131099649.otf|"
        $writeOp26 =
"/data/data/com.appspace/files/fnt2131099649.otf|"
        $writeOp27 =
"/data/data/com.appspace/files/fnt2131099649.otf|"
        $writeOp28 =
"/data/data/com.appspace/shared_prefs/com.appspace.xml|"
        $writeOp29 =
"/data/data/com.appspace/shared_prefs/com.appspace.xml|"
        $writeOp30 = "/data/data/com.appspace/files/_themes.xml|"
        $writeOp31 = "/data/data/com.appspace/files/_themes.xml|"
        $writeOp32 =
"/data/data/com.appspace/shared_prefs/com.appspace.xml|"
        $writeOp33 =
"/data/data/com.appspace/app_Parse/applicationId|"
        $connectionOpenTag = "network-opened"
        $openHost0 = "www.appspaceforandroid.com"
        $port34 = "80"
        $connectionWriteTag = "network-write"
        $readHost0 = "www.appspaceforandroid.com"
        $port35 = "80"
        $connectionReadTag = "network-read"
        $writeHost0 = "www.appspaceforandroid.com"
        $port36 = "80"

      condition:
        #connectionOpenTag >= 1 and 1 of ($openHost*,$port*) and
        #connectionReadTag >= 1 and 1 of ($readHost*,$port*) and
        #connectionWriteTag >= 1 and 1 of ($writeHost*,$port*) and
   all of ($permission*)and
        all of ($readOp*)and
        all of ($writeOp*)
}
```

10.5. ANEXO 5: INFORME EN FORMATO MIST

```
13|00000000|6b65726e656c33322e646c6c7c800000
14|00000000|7c800000536574446c6c4469726563746f7279570
14|00000000|7c8000005365745365617263685061746 84d6f64650
14|00000000|7c80000053657450726f6365737344455050 6f6c6963790
42|00000002|80000001536f6674776172655c436f6465476561725c4c6f63616c6
57300000000
42|00000002|80000002536f6674776172655c436f6465476561725c4c6f63616c6
57300000000
42|00000002|80000001536f6674776172655c426f726c616e645c4c6f63616c657
300000000
42|00000002|80000001536f6674776172655c426f726c616e645c44656c7068695
c4c6f63616c657300000000
19|00000000|0000004c
13|c0000135|000000000
13|c0000135|000000000
18|00000000|ffffffff009500000014000000000004
13|00000000|6b65726e656c33322e646c6c7c800000
14|00000000|7c800000476574446973 6b4672656553706163654578570
13|00000000|6b65726e656c33322e646c6c7c800000
14|c0000139|7c800000576f77363444697361626c65576f7736344673526564697
2656374696f6e0
13|00000000|6b65726e656c33322e646c6c7c800000
14|c0000139|7c800000576f773634526576657274576f7736344673526564697 26
56374696f6e0
15|00000000|12fea47368656c6c33322e646c6c7e6a0000
18|00000000|ffffffff00a900000001000000000004
18|00000000|ffffffff00a900000000100000000004
13|c0000135|6e65746d73672e646c6c00000000
1a|00000000|0000004c80100080433a5c57494e444f57535c73797374656d33325
c6e65746d73672e646c6c15
1c|00000000|00000054000f00050000004c
19|00000000|0000004c
19|00000000|00000054
50|00000001|ffffffff00a90000000000008000
51|00000001|ffffffff00400000000100000000040
51|00000001|ffffffff00400000000100000000002
51|00000001|ffffffff00401000000f00000000040
51|00000001|ffffffff00401000000f00000000020
51|00000001|ffffffff00419000000530000000040
51|00000001|ffffffff00419000000530000000002
1a|00000000|0000005480100080011
28|00000000|000000540012fee0
2d|00000000|000000540012fef0
2b|00000000|0000005400416668
2b|00000000|000000540012ff14
2b|00000000|000000540012ff0f
28|00000000|000000540012feb4
2d|00000000|000000540012feb4
28|00000000|000000540012fe8c
2b|00000000|000000540012fe64
2b|00000000|0000005400a3166c
18|00000000|ffffffff00ad0000002040000000004
2b|00000000|0000005400a3166c
50|00000001|ffffffff00ad90000000000008000
28|00000000|000000540012fee0
```

```
2d|00000000|000000540012fee0
2d|00000000|000000540012fed4
13|00000000|6b65726e656c33322e646c6c7c800000
14|00000000|7c8000047657455736572446566617 56c7455494c616e677561676
50
18|00000000|ffffffff00ad00000014000000000004
1|00000001|0
2d|00000000|000000540012fef0
1a|00000000|0000004c40100080050
18|00000000|ffffffff7fe400000017000000000004
2b|00000000|000000540012fefc
2b|00000000|000000540012fef7
28|00000000|000000540012fe9c
2d|00000000|000000540012fe9c
28|00000000|000000540012fe74
2b|00000000|000000540012fe88
2b|00000000|0000005400b8c37c
18|00000000|ffffffff00c100000020400000000004
19|00000000|0000004c
19|00000000|00000054
15|00000000|12f878433a5c57494e444f57535c73797374656d33325c557854686
56d652e646c6c5b150000
12|00000000|
15|00000000|12f7c475787468656d652e646c6c5b150000
15|00000000|12f5f4433a5c57494e444f57535c73797374656d33325c557854686
56d652e646c6c5b150000
15|00000000|12f3c4433a5c57494e444f57535c73797374656d33325c557854686
56d652e646c6c5b150000
15|00000000|12f3c0433a5c57494e444f57535c73797374656d33325c557854686
56d652e646c6c5b150000
15|00000000|12f87c433a5c57494e444f57535c73797374656d33325c4d5343544
62e646c6c746b0000
1b|40000000|0000007000
26|00000000|00000074000f001f0
4b|06e10145|2746c07c3746b000021c
4b|00e80111|7746c04cd746b000021c
19|00000000|00000128
35|00000000|0000011c
19|00000000|00000120
35|00000000|0000016c
19|00000000|00000170
35|00000000|00000164
19|00000000|00000168
35|00000000|000000fa
19|00000000|00000068
19|00000000|0000006c
19|00000000|00000064
19|c0000008|00000000
50|00000001|ffffffff00f10000000000008000
50|00000001|ffffffff00a50000000000008000
8|00000000|0
FF|8a8a18777f4b0f
FF|8a8a18777f4b0f
FF|8a8a18777f4b0f
FF|8a8a18777f4b0f
FF|898918777f4b0f
FF|898918777f4b0f
FF|898918777f4b0fFF
```

10.6. ANEXO 6: BASES DE DATOS UTILIZADAS

F-Secure - http://www.f-secure.com/v-descs/

ESET-NOD32 - http://www.virusradar.com/en/Win32_Cimag.CN/description

AhnLab-V3 - http://global.ahnlab.com/en/site/threat/viruscenter/

Emsisoft - http://www.emsisoft.com/en/malware/Adware.Win32.00008-remove.aspx

viRobot - http://www.globalhauri.com/security/virus_view.html?intSeq=1401&strPart=3&key=&cpage=1

Hacker - http://www.hacksoft.com.pe/dmalwares/desamenaza.php?valor=Bambra.gen

Symantec - http://www.symantec.com/es/es/security_response/writeup.jsp?docid=2006-082416-2803-99&tabid=2

Sophos - http://www.sophos.com/es-es/threat-center/threat-analyses/viruses-and-spyware/Troj~Zbot-EAO/detailed-analysis.aspx

Panda - http://www.pandasecurity.com/spain/homeusers/security-info/223189/RedCrossAntivirus

10.7. ANEXO 7: APLICACIONES MÓVILES ANALIZADAS

Nombre	Malware identificado
albumteck-penthouse-nude-23-9942-b8c7898fec7d1b3ac5e1aee02a4e05b7.apk	-
cento-n2-lib-7-3077974-983c53fe37b2c90786ce146b5d8c115a.apk	-
cm-aptoide-pt-403-3008098-e7984e3bee93c1891e98dcc01be82886.apk	-
com-a0soft-gphone-acc-free-211-3158032-d944085eb2b41619f566102b3c56fbb2.apk	-
com-aastocks-dzh-19-3159057-5d4b9182cb7c658e464fd3ec188803fb.apk	TROJ_GEN.FCBHZJ1
com-adamschmelzle-origami-rose-430-3158669-06c51f7a2ea24c71dd8ad60cb2ffb4bb.apk	Adware/AirPush.KL
com-adobe-flashplayer-111115012-729262-85e6139f5f0e79e45215b0ceff4337e7.apk	-
com-android-chrome-1410058-3136784-f4377b85ce33b5fe981c124d71fe5201.apk	-
com-android-vending-8016010-2073977-14bebec455594bd71c47b6f57164ca91.apk	-
com-appspace-6-3157592-1c7f5132c4ece7f8ea52dd9b5a6572ed.apk	TROJ_GEN.FCKJ5
com-bytesequencing-spades-ads-173-3158685-a5f36772d0c7f00cf4ba469f39d718c8.apk	TROJ_GEN.FSDDB5
com-catstudio-sogmw-29-3159056-3d09320319a5a8842bae03e58b4afd27.apk	-
com-control4-myhome-9-3157724-9c741a5e52dec3e068c84a50bd6ded59.apk	-
com-creativemobile-dragracingbe-11033-3158060-2b9c4b5f0d4d9b4814f9c524a0b066e8.apk	Trojan/AndroidOS.eee
com-denper-addonsdetector-29-3158138-c22254d8863a3663e5692773e63da8fb.apk	-
com-dexetra-friday-45-3159054-26607b6a2749b581b50f8484dafae167.apk	-
com-dreamstudio-epicdefense-53-3158690-040c62696216f97ad59c28e066572200.apk	Android/Adware.WapsG
com-ea-game-simpsons4-row-6-2963661-0ff977509b038bf7681eb439b48c64fb.apk	-
com-ebuddy-android-xms-43-3157713-384c8132b08cd1941d2af5303c208298.apk	-
com-estrongs-android-pop-181-612627-ab326c722f5a5b9e3962e2017afaf7ca.apk	-
com-evernote-15040-3158068-dd31135d4f57f17d73a933eb84c600fe.apk	-
com-ezjoynetwork-jewelsmaze-30-3158697-b1235c0f818a6564019a1bb1517ea885.apk	-
com-ezjoynetwork-marbleblast2-18-3158077-5c4f3288576e145b2c675c1ccdfb028f.apk	-
com-facebook-katana-149649-2774570-63c9ee238b83c11a254b975eff5e84df.apk	-
com-facebook-katana-177847-3081009-1739ebc87f5c9921cea8e3961fcb5ece.apk	-
com-frimastudio-nunattack-17-3116363-c25d2f6e7e706f87ca370063a475c169.apk	-
com-game-bubbleshooter-15-3159043-5e469b3e3c507819bfadd1bcfa8fbdf7.apk	-
com-gamelion-ck-9-3025629-9e030f2f793b3af124402a7da00836cb.apk	-
com-gameloft-android-anmp-gloftasphalt5free-asphalt5-332-210630-2abfc18db31640db6e189c31fbe90a43.apk	-
com-gameloft-android-tbfv-gloftn2hp-ml-103-451631-abb3f60ae4c72bba77b15df79878400e.apk	-
com-ggnes-supermario2-23-30220-1662673410c634e95244d0077735c581.apk	AndroidOS/GinMaster.L
com-glu-flcn-new-103-3025632-d27e44eb7c8b2fd68b9df241d8f747f0.apk	-
com-gogii-textplus-57543260-3077967-1f97d543778d6df4a4eb3859a6761281.apk	-
com-google-android-apps-maps-614030402-2604784-588a5dc557066669afd7fa6b0faa9437.apk	-
com-google-android-gms-3027110-2777102-2d3daf756852d813cd78366342b33d4d.apk	-
com-google-android-marvin-talkback-89-2720023-b8725c756e99919919e956522bcf9056.apk	-
com-google-android-marvin-talkback-95-3135571-2fe49d1e6ece808f8727d64399f81346.apk	-

com-google-android-youtube-4411-3046340-4a48b01d2cdcff37c80a75ded8ea0c81.apk	-
com-halfbrick-fruitninjafree-1703-375379-729b0c28aa24ec0d032bb7ac26a188b6.apk	-
com-halfbrick-fruitninjahd-1504-41277-a01e53fa91e8f5d6a6f50102a3692dbc.apk	-
com-halfbrick-jetpackjoyride-1360-1296937-d1120e16a9a6441b3da3856b58eb2e4a.apk	-
com-hk-xsutdio-app10-1-24048-813acb8eadbcfaaedad471e19eb8616a.apk	-
com-imangi-templerun2-3-2435191-deb9339e12fbcf3c40e2ffb7a8f8f768.apk	-
com-jnj-mocospace-android-83-3158139-151b2635e3a53fd79cc588a83a33afff.apk	-
com-kiloo-frisbeeforever-11-3116551-300678effc3a19c1017d7594db9c5d4d.apk	-
com-kiloo-subwaysurf-17-2226622-46eff9baa02a6e209eaedb4575b954f3.apk	-
com-kiloo-subwaysurf-26-3130652-ac685b9623e9398ad2b6cf5b0630a60a.apk	-
com-konami-pes2011-1000001-35497-8ef828d66ab408c3b060f936e1ec4d05.apk	-
com-kugou-android-5447-3157585-bbb0ed0bb37f60b884021a9e261fb912.apk	-
com-leokay-windroid-62-3158036-2a4f8265997891d74295d28685f3dd87.apk	-
com-lookout-81100-2775469-ca4c5d67c075ece5284e8c4d09669a65.apk	-
com-macropinch-pearl-29-3157597-cb29cf3c154c05549fbe8ef17704422c.apk	-
com-makru-minecraftbook-45-3158143-e03c0e91e939a127750944d066485ea1.apk	-
com-mdream-sex-2147483647-8876-0b7991b2fbaff154d9fe2b63535674d1.apk	-
com-mendhak-gpslogger-30-3157816-245219aef31f230f1be38b9bcd724bdd.apk	-
com-moderati-zippo2-10-3159228-f5c6f2b0366501fa6e6fbca306f0b87c.apk	-
com-mufumbo-android-recipe-search-129-3157619-91e2a7a92f7ec2dfca191d2a82f0b4ff.apk	-
com-myyearbook-m-57-3159229-fb2ad5f843e402ece6814a0dc63bac34.apk	-
com-nekki-vector-1000009-3116365-70b7a5a5fce7b4b73359fe25ac3935e0.apk	-
com-popcap-pvz-1-615950-5368dfd01b4ab0adb786525718f6761b.apk	-
com-quadrimind-ipaciencia-5-3157635-91cd5c9ccfc8719289778e2a98c97d71.apk	-
com-rovio-angrybirds-3100-2719922-336fcacbcf17bfe41e5ec5160de84235.apk	-
com-runtastic-android-45-2461602-34f7766f0d38ca712b80916cb208edf1.apk	-
com-skype-raider-50469393-2654246-6497a3d737935c5eb379835efa9068f0.apk	-
com-soa-sega-soniccdlite-5-2833966-a0cf750e654d9cd4d8f09f9c3e44f36a.apk	-
com-songbirdnest-mediaplayer-20130503-3159226-83354b4c75ec725496d886f5f01951a2.apk	-
com-soundcloud-android-61-3159045-446b55f9116ad4d7a2bc7346fd0a702d.apk	-
com-stac-empire-main-98-3157632-a9d3a9c763178971959b0bc955a7341e.apk	-
com-swype-android-inputmethod-39062-810282-baebf6ba0c15ca6052511e01ed5d0d50.apk	-
com-tecsta-com-annakarenina-4-3032654-d73156c6c21eb3d101eab4aca703c756.apk	-
com-theolivetree-ftpserver-13-3159225-26342579248fac8f9a95e60a8e878535.apk	-
com-think2-movie-300-9908-62891a33a1928f08e78e4e1224e6caca.apk	-
com-trapster-android-4000001-3159313-1daeccd17e59ceb46e4edfca548efdf4.apk	-
com-umobisoft-igp-camera-35-3159314-dcb5a48817188a0929ef29d50a1c8e0a.apk	-
com-verge-android-421-3157588-1392ebd5d804bb3915737af90199ac39.apk	-
com-viber-voip-28-2757513-ad04065ad0eb4db65d53a50271418aaa.apk	-
com-whatsapp-41547-2455029-60adb0e6448cc78d01f0c18369aa2e90.apk	-
com-whatsapp-44171-2777736-af8a4c8821f4c86679c9a95f6a88d89e.apk	-
com-whatsapp-45196-3082862-eba07d85381778e250c57233eb8e0ec3.apk	-

com-x3m-tx3-6-2383009-a994b8a25ef02574916ab0b5bed4765a.apk	-
com-youku-paike-33-3157613-fca349af2b5f93e1c34e82b307a38031.apk	-
com-yt-diablo-62-3159309-b027a0afa16729ce0c5ed4d97908bf51.apk	-
com-yt-diablosc-62-3159305-26cce5194c62053af1c8660840b4bc78.apk	-
com-zeptolab-timetravel-free-google-734104-3077959-8cfc83f30f07e978c90e3fe9a5f2ea09.apk	-
devian-tubemate-home-212-790462-c20b22aa5bf0d98abe87a29cb3248daa.apk	-
fieldbird-yourself-85-3157596-ca232e5ba496119f704e2e5c18176f74.apk	-
kik-android-37-3078560-13fcc5d1fb053796e57963115d27c904.apk	-
kr-sira-measure-34-3159050-88f982691d140d69bd3d47a013dba960.apk	-
kr-sira-metal-15-3159061-de0602a735c821f1d50a9dcc01ebdc46.apk	-
logos-quiz-companies-game-79-3159274-c93d3e7ffb14d3c2abac53f0372f64db.apk	-
mobi-infolife-uninstaller-52-3159281-9527d9eff7896f9c7b06b91ecbd4cffc.apk	-
mypack-pack-28-3157604-0cee967ce913e760a020792b07a0cf54.apk	-
net-alouw-alouwcheckin-62-3157614-682dc958dfdaddb00356efb4526e58cd.apk	-
org-blackmart-market-51-13136-2032927f878471e43deb5d8962ff47e7.apk	-
org-frosty-fromegle-52-3159273-d983f088e4d8f56110838467064e98c4.apk	-
org-wopnersoft-unitconverter-56-3159237-a3d9c9a66022cb153a465ac968b7a0d7.apk	-
org-wordpress-android-70-3157985-405d7efbcd2302b5ddc5a36a7f700278.apk	-
ru-funapps-games-frutcoctail-44-3159272-2f201ff6af6bd206242f73e9b96a2950.apk	-
trackthisout-compass-com-56-3159090-8bee64052f29be32c5266910e1a6fc50.apk	-
trackthisout-try-com-56-3159084-05ad42328f7371e71d7630d552627221.apk	-
zok-android-shapes-3760012-3159231-5a563b11fc33e38da59daf1989ead3d0.apk	-

10.8. ANEXO 8: MALWARE SELECCIONADO PARA PRUEBAS GENÉRICAS

Nombre	Positivos verdaderos	Positivos falsos
Backdoor.Win32.BackStreets	1	0
Backdoor.Win32.BO.ButtTrumpet	1	12
Backdoor.Win32.BO.SpeakEasy	1	12
Exploit.Win32.Pidief.uk	1	0
Exploit.Win32.RPCLsa.01.x	1	6
Exploit.Win32.Sapi	1	3
Exploit.Win32.SQLhuc.b	1	5
Exploit.Win32.Ssl.02	0	98
Exploit.Win32.Storm.11	1	0
Exploit.Win32.WinRar.b	1	1
Exploit.Win32.YouHack.a	1	2
Exploit.Win32.Zang	1	3
Flooder.Java.NewsAgent.109	0	98
Flooder.Win32.ArpPack.a	1	1
HackTool.PHP.Zbot.a	0	98
HackTool.Win32.Binder.v	1	0
HackTool.Win32.Skype.a	1	0
Hoax.Win32.Renos.me	1	0
Net-Worm.Win32.Koobface.by	1	0
Net-Worm.Win32.Sasser.c	1	0
P2P-Worm.Win32.Spear.j	1	0
P2P-Worm.Win32.SpyBot.k	1	0
Packed.Win32.Klone.bg	1	0
Packed.Win32.Mondera.b	1	1
Packed.Win32.NSAnti.ar	1	0
Rootkit.Linux.Matrics.a	0	98
Rootkit.Win32.Agent.aee	0	98
Trojan-Banker.Win32.Banbra.dlz	1	1
Trojan.BAT.Nightstar.581	0	98
Trojan.BAT.NoDelDir.a	0	98
Trojan.BAT.Ops	0	98
Trojan.BAT.Passer.a	0	98
Trojan.BAT.PatchSystemini	0	98
Trojan.BAT.Qhost.be	0	98
Trojan.BAT.Ratty.AntiAVP.d	0	98
Trojan.BAT.StartPage.ap	1	0
Trojan.Boot.AntiMD.b	1	0

Trojan.Boot.ArjGuru	0	98
Trojan.Boot.BlackHack.2641	1	3
Trojan-Downloader.BAT.Isql.a	1	0
Trojan-Downloader.DOS.RunAuto	1	0
Trojan-Downloader.HTML.Agent.ga	0	98
Trojan-Downloader.HTML.IFrame.jg	0	98
Trojan-Downloader.JS.Agent.buv	0	98
Trojan-Downloader.VBS.Agent.ug	0	98
Trojan-Downloader.VBS.Obfuscated.c	0	98
Trojan-Downloader.Win32.Banload.aazc	1	0
Trojan-Downloader.Win32.Delf.bpl	1	0
Trojan-Downloader.Win32.FraudLoad.vezo	1	5
Trojan-Downloader.Win32.Konix.d	1	3
Trojan-Downloader.Win32.Small.zzm	1	1
Trojan-Downloader.Win32.Suurch.aad	1	2
Trojan-GameThief.Win32.Magania.mrz	1	0
Trojan-GameThief.Win32.Magania.mtm	1	0
Trojan-GameThief.Win32.OnLineGames.lps	1	0
Trojan-GameThief.Win32.OnLineGames.slaw	1	0
Trojan-PSW.Win32.LdPinch.adc	1	0
Trojan-PSW.Win32.Lmir.cdx	1	5
Trojan-PSW.Win32.Lolita	1	5
Trojan-PSW.Win32.Lomaster	1	0
Trojan-PSW.Win32.M2.18	1	3
Trojan-PSW.Win32.Madcap	0	98
Trojan-PSW.Win32.Madzumba	1	3
Trojan-PSW.Win32.Magania.krn	1	1
Trojan-PSW.Win32.OnLineGames.admu	1	0
Trojan-PSW.Win32.QQPass.cbz	1	7
Trojan-Spy.Win32.BewLoader	1	0
Trojan-Spy.Win32.Briss.a	1	7
Trojan-Spy.Win32.Cardspy.q	1	3
Trojan-Spy.Win32.CashBack.a	0	98
Trojan-Spy.Win32.Coiboa.m	1	0
Trojan-Spy.Win32.ControlRandom	1	12
Trojan-Spy.Win32.Delf.aeh	1	0
Trojan-Spy.Win32.Flux.qw	1	0
Trojan.Win32.Agent.qsl	1	0
Trojan.Win32.BHO.cbd	1	0
Trojan.Win32.Buzus.hqo	1	2
Trojan.Win32.Delf.exe	1	7
Trojan.Win32.Dialer.dxj	1	7

Trojan.Win32.ExitWin.Predator	1	18
Trojan.Win32.Favadd.aj	1	0
Trojan.Win32.FraudPack.jz	1	2
Trojan.Win32.Genome.chm	1	7
Trojan.Win32.Inject.ahte	1	0
Trojan.Win32.Monderb.fob	1	0
Trojan.Win32.Scapur.g	1	0
Trojan.Win32.SecondThought.cc	1	1
Virus.Boot.Chance.a	0	98
Virus.DOS.Abraxas.Cleton.1508	0	98
Virus.DOS.GoldSoft.1263	0	98
Virus.DOS.Gollum.758	-	-
Virus.DOS.Gotcha.906	0	98
Virus.DOS.Grog.1200	-	-
Virus.MSWord.Bobo.g	0	98
Virus.VBS.Manuela	0	98
Virus.Win32.Expiro.a	-	-
Virus.Win32.FlyStudio.a	0	98
Virus.Win32.Folcom.b	1	0
Worm.Win32.AutoRun.dbx	-	-
Worm.Win32.PornRun.m	0	98
Worm.Win32.Randex.c	1	1
Worm.Win32.Viking.ja	-	-

10.9. ANEXO 9: MALWARE SELECCIONADO PARA PRUEBAS DE FAMILIAS

Nombre	Positivos familia	Negativos familia
HackTool.Win32.Agent.ad	32	40
HackTool.Win32.Agent.ag	13	59
HackTool.Win32.Agent.ai	2	70
HackTool.Win32.Agent.aj	1	71
HackTool.Win32.Agent.ak	1	71
HackTool.Win32.Agent.an	10	62
HackTool.Win32.Agent.ao	1	71
HackTool.Win32.Agent.ar	3	69
HackTool.Win32.Agent.av	71	1
HackTool.Win32.Agent.ax	3	69
HackTool.Win32.Agent.ba	1	71
HackTool.Win32.Agent.bb	19	53
HackTool.Win32.Agent.bc	1	71
HackTool.Win32.Agent.be	19	53
HackTool.Win32.Agent.bn	11	61
HackTool.Win32.Agent.bp	3	69
HackTool.Win32.Agent.br	1	71
HackTool.Win32.Agent.bu	1	71
HackTool.Win32.Agent.by	71	1
HackTool.Win32.Agent.ch	1	71
HackTool.Win32.Agent.cs	5	67
HackTool.Win32.Agent.cu	1	71
HackTool.Win32.Agent.cx	1	71
HackTool.Win32.Agent.cy	36	36
HackTool.Win32.Agent.de	2	70
HackTool.Win32.Agent.dh	0	72
HackTool.Win32.Agent.dj	1	71
HackTool.Win32.Agent.dk	9	63
HackTool.Win32.Agent.dm	2	70
HackTool.Win32.Agent.ed	2	70
HackTool.Win32.Agent.eu	2	70
HackTool.Win32.Agent.e	9	63
HackTool.Win32.Agent.fq	34	38
HackTool.Win32.Agent.fr	1	71
HackTool.Win32.Agent.fs	1	71
HackTool.Win32.Agent.ft	1	71
HackTool.Win32.Agent.fu	5	67

HackTool.Win32.Agent.fx	2	70
HackTool.Win32.Agent.f	2	70
HackTool.Win32.Agent.fz	31	41
HackTool.Win32.Agent.ga	1	71
HackTool.Win32.Agent.gd	46	26
HackTool.Win32.Agent.gg	8	64
HackTool.Win32.Agent.gi	1	71
HackTool.Win32.Agent.gj	11	61
HackTool.Win32.Agent.gl	13	59
HackTool.Win32.Agent.gm	1	71
HackTool.Win32.Agent.gn	13	59
HackTool.Win32.Agent.go	13	59
HackTool.Win32.Agent.gs	13	59
HackTool.Win32.Agent.ha	1	71
HackTool.Win32.Agent.he	8	64
HackTool.Win32.Agent.hf	71	1
HackTool.Win32.Agent.hi	71	1
HackTool.Win32.Agent.hm	19	53
HackTool.Win32.Agent.hq	1	71
HackTool.Win32.Agent.ia	2	70
HackTool.Win32.Agent.ib	1	71
HackTool.Win32.Agent.ic	2	70
HackTool.Win32.Agent.ik	5	67
HackTool.Win32.Agent.it	8	64
HackTool.Win32.Agent.li	8	64
HackTool.Win32.Agent.lk	71	1
HackTool.Win32.Agent.lr	2	70
HackTool.Win32.Agent.lv	71	1
HackTool.Win32.Agent.lw	71	1
HackTool.Win32.Agent.l	9	63
HackTool.Win32.Agent.lz	71	1
HackTool.Win32.Agent.ma	71	1
HackTool.Win32.Agent.mk	71	1
HackTool.Win32.Agent.ot	1	71
HackTool.Win32.Agent.z	34	38

Nombre	Positivos familia	Negativos familia
Backdoor.Win32.Bancodor.ae	4	19
Backdoor.Win32.Bancodor.ai	4	19
Backdoor.Win32.Bancodor.an	4	19
Backdoor.Win32.Bancodor.ao	3	20
Backdoor.Win32.Bancodor.aq	5	18

Backdoor.Win32.Bancodor.ar	1	22
Backdoor.Win32.Bancodor.as	1	22
Backdoor.Win32.Bancodor.at	4	19
Backdoor.Win32.Bancodor.au	4	19
Backdoor.Win32.Bancodor.ax	6	17
Backdoor.Win32.Bancodor.a	23	0
Backdoor.Win32.Bancodor.ba	6	17
Backdoor.Win32.Bancodor.bd	2	21
Backdoor.Win32.Bancodor.be	2	21
Backdoor.Win32.Bancodor.bf	3	20
Backdoor.Win32.Bancodor.bg	5	18
Backdoor.Win32.Bancodor.bh	5	18
Backdoor.Win32.Bancodor.bi	1	22
Backdoor.Win32.Bancodor.b	2	21
Backdoor.Win32.Bancodor.c	23	0
Backdoor.Win32.Bancodor.i	0	23
Backdoor.Win32.Bancodor.j	12	11
Backdoor.Win32.Bancodor.k	23	0

Nombre	Positivos familia	Negativos familia
Backdoor.Win32.BackEnd.a	4	0
Backdoor.Win32.BackEnd.b	4	0
Backdoor.Win32.BackEnd.c	2	2
Backdoor.Win32.BackEnd.d	2	2

10.10. ANEXO 10: TABLA DE DESCRIPCIÓN DE PRUEBAS

Nombre de la prueba	Breve descripción	Número de muestras	RSF-01	RSF-02	RSF-03	RSF-04	RSF-06	RSF-07	RSF-08	RSF-09	RSF-10	RSF-11	RSF-12	RSF-13	RSF-14	RSF-15	RSF-16	RSF-17	RSF-18
Pruebas de aplicaciones para sistemas Android	Evaluar los resultados de la plataforma para aplicaciones móviles.	101	X	X	X										X	X	X	X	
Pruebas genéricas de identificación sobre malware aleatorio	Evaluar los resultados obtenidos por la plataforma para una colección no escogida de malware.	103	X	X	X	X	X		X	X	X	X							
Pruebas específicas de identificación para familias de malware	Evaluar los resultados obtenidos por la plataforma para una colección de malware de la familia Agent.	72	X	X	X	X	X		X	X	X	X							
	Evaluar los resultados obtenidos por la plataforma para una colección de malware de la familia Bancodor.	23	X	X	X	X	X		X	X	X	X							
	Evaluar los resultados obtenidos por la plataforma para una colección de malware de la familia Backend.	4	X	X	X	X	X		X	X	X	X							
Pruebas específicas de identificación de CVE´s en malware identificado	Evaluar los resultados obtenidos por la plataforma para una serie de muestras identificadas.	4	X	X	X	X	X	X	X	X	X	X							
Verificación manual	Evaluar ciertas funcionalidades no recogidas por las pruebas realizadas.												X	X					X