TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Análisis estático y dinámico de malware en Android

Análisis de malware y contramedidas en la plataforma Android

Autor
José Francisco Guerrero Collantes

Director
José Antonio Gómez Hernández

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, junio de 2017

Análisis estático y dinámico de malware en Android: Análisis de malware y contramedidas en la plataforma Android

José Francisco Guerrero Collantes

Palabras clave: malware, aplicación, análisis estático, análisis dinámico, ingeniería inversa, listas negras.

Resumen

El presente trabajo tiene como objetivo determinar si una aplicación Android posee malware implícito en su código. Esta caracterización se realiza tanto de información obtenida de forma estática como de forma dinámica. La información relevante se obtiene analizando los ficheros que forman la aplicación como puede ser el AndroidManifest o las clases que definen la aplicación y monitorizando la ejecución de la aplicación en una máquina virtual. Con la información obtenida se realiza una valoración para determinar en un intervalo de confianza la probabilidad de que la aplicación analizada posea malware. Como se pretende que además un usuario común pueda analizar cualquier aplicación, se le ofrece una interfaz web donde poder enviar dicha aplicación para ser analizada y consultar los datos de una forma fácil y sencilla.

Static and dynamic malware analysis on Android: Analysis of malware and countermeasures on the Android platform

José Francisco Guerrero Collantes

Keywords: malware, application, static analysis, dynamic analysis, reverse engineering, blacklists.

Abstract

The present work aims to determine if an Android application has malware implicit in its code. This characterization is carried out as much of information obtained statically as of a dynamic form. This relevant information is obtained analyzing the files that make up the application such as the AndroidManifest or classes that define the application and monitoring the execution of the application in a virtual machine. With the obtained information an evaluation is made to determine in a confidence interval the probability that the analyzed application has malware. As it is intended that in addition a common user can analyze any application, you are offered a web interface where you can send the application to be analyzed and query the data in an easy and simple.

Yo, **José Francisco Guerrero Collantes**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 14276388N, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: José Francisco Guerrero Collantes

Granada a 20 de junio de 2017 .

D. **José Antonio Gomez Hernández**, Profesor del Área de Sistemas Operativos del Departamento de Lenguajes y Sistema Informáticos de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Análisis estático y dinámico de malware en Android: Análisis de malware y contramedidas en la plataforma Android***, ha sido realizado bajo su supervisión por **José Francisco Guerrero Collantes**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 20 de junio de 2017 .

El director:

José Antonio Gomez Hernández

Índice

Índice de figuras

Índice de tablas

1. Objetivos

El objetivo de este trabajo es ofrecer una plataforma para determinar si una aplicación Android contiene malware antes de su instalación. La información que se le pretende mostrar al usuario se obtiene tanto de forma estática (los permisos y lo contenido en las clases Java) como de forma dinámica (mediante la monitorización de las peticiones a servidores externos y de las llamadas al sistema de la aplicación).

En lo referente a la información dinámica, se quiere transmitir el listado de permisos que requiere cada aplicación que puedan comprometer la privacidad del usuario o puedan facilitar el uso de vulnerabilidades por parte de terceros. Además, se le mostrará al usuario las líneas de las clases Java que forman la aplicación que puedan suponer algún perjuicio al usuario.

No podemos dejar de lado la información que se obtiene de la aplicación en tiempo de ejecución ya que de ella podemos obtener comportamientos no comunes que no podemos detectar analizando de una forma estática en una aplicación. Esta información pueden ser las llamadas al sistema que pueden determinar comportamientos anómalos o que sobrepasen los privilegios de una aplicación normal y el tráfico generado por la aplicación que nos sirve para determinar si se intenta conectar con servidores maliciosos.

Con la información anterior se pretende realizar una serie de valoraciones y mostrarlas en un intervalo de confianza que den al usuario una valoración sobre el comportamiento malicioso de la aplicación que están instalando y las consecuencias que puede acarrear.

Como esto se quiere ayudar a usuarios con pocos conocimientos tecnológicos a detectar software malicioso que puede comprometer toda su información personal al no darle suficiente importancia al funcionamiento interno y a los requisitos que precisan aplicaciones que provienen de orígenes desconocidos o de otras apps.

2. Antecedentes

2.1. Malware

El malware se define como un tipo de sofware dañino cuya finalidad es acceder al dispositivo para obtener información, dañarla, etc. sin conocimiento del propietario. Existen diversos tipos pero todos tienen la intencionalidad de enriquecer o aportar información valiosa al desarrollador.

El crecimiento del malware en dispositivos móviles cree de una forma exponencial y según fuentes de Kaspersky[25] se han detectado más de 8.500.000 de instalaciones de paquetes maliciosos. Además en este último año ha destacado el crecimiento del software malicioso que usa permisos de superusuario, el aumento del rasomware afectando a los ficheros del dispositivo y la introducción de código maligno en aplicaciones de gestión de banca.

Según la función que realiza el malware y del daño que produce al dispositivo podemos diferenciar distintos tipos.

2.1.1. Tipos de malware

- *Troyanos*. Pueden eliminar y enviar información predeterminada de forma automática aunque lo más destacable es la posibilidad que ofrece el creador de utilizar nuestro dispositivo de forma remota.
- *Rasomware*. Cifra o prohíbe el acceso a determinados archivos pidiendo al usuario un rescate para poder acceder a la información con total normalidad.
- *Spyware*. Recopila determinada información de un dispositivo y la transmite al desarrollador sin el conocimiento del propietario del dispositivo.
- *Adware*. Recopilan datos del usuario que puedan determinar que tipo de publicidad mostrar y mostrársela al usuario para conseguir beneficio económico.
- *Aplicaciones falsas*. Aplicaciones que simulan a otras aplicaciones con la intencionalidad de lucrarse con publicidad o con las mismas funciones que la original pero obligando a pagar para desbloquear algunas opciones. Otras aplicaciones de este tipo son exactamente iguales a la original pero ocultando adware.
- *Troyanos SMS*. Envían SMS o realizan a llamadas a servicios premium sin el consentimiento del usuario para obtener beneficios económicos. Son uno de los tipos más comunes de malware y de los más fáciles de detectar analizando sus permisos.

2.2. Descripción general del sistema operativo Android

Android es un sistema operativo diseñado para dispositivos móviles como smartphones, tablets o smartwatches.Actualmente es el sistema operativo más usado en el mundo.

Dicho sistema operativo fue desarrollado por Android Inc., empresa que pasó a ser parte de Google tras ser adquirida en 2005. Pese a pertenecer a Google, Android posee la mayoría de su código bajo la licencia Apache, licencia totalmente libre y de código abierto.

El sistema operativo Android está basado en el kernel de Linux con pequeñas modificaciones. El lenguaje de su desarrollo es C para su número, C++ para algunas bibliotecas de terceros y Java para la interfaz de usuario.

La ejecución de las aplicaciones se realiza en una máquina virtual llamada Dalvik[7] hasta la versión 4.4. Dicha máquina ejecuta códigos Java compilados en bytecode. El código es interpretado en tiempo de ejecución. Google desarrolló esta máquina con la finalidad de funcionar con pocos recursos y producir un bajo gasto de batería. En las últimas versiones se ha dejado de lado el desarrollo de Dalvik y se ha sustituido por ART[8] ya que ART compila los procesos y los guarda en la caché en el momento de la instalación de la aplicación y no en tiempo de ejecución.

En la Figura 2.1 aparece la estructura del sistema operativo Android donde se aprecia con total claridad las distintas partes del sistema. Apreciamos la máquina Dalvik en naranja, las bibliotecas en verde, el kernel en rojo y los frameworks y aplicaciones en azul.

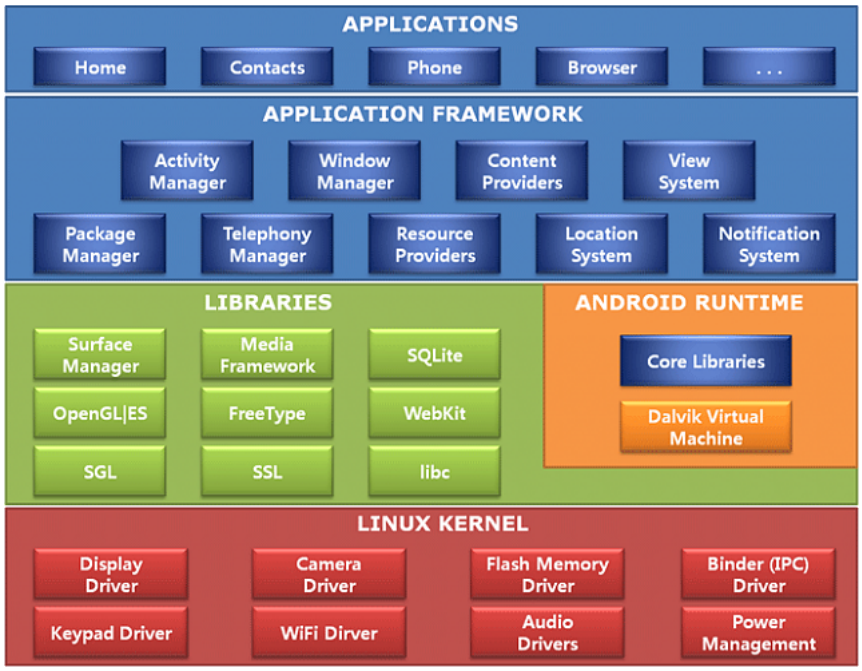


Figura 2.1: Estructura del SO Android

2.3. Aplicaciones Android

Como todo sistema operativo, Android permite la ejecución de aplicaciones únicamente compatibles con este sistema operativo. Desarrolladas en Java, poseen una estructura y una serie de características que son únicamente existentes en Android.

2.3.1. Estructura de las aplicaciones Android

Las aplicaciones que instalamos en nuestro dispositivo Android vienen en formato APK (Android Package Files). Los ficheros APK son ficheros ZIP firmados los cuales en su interior contienen la aplicación en bytecode junto, a los recursos externos, bibliotecas de terceros y el fichero AndroidManifest. Se limitan los privilegios de las aplicaciones mediante un sistema de permisos[4] y se aíslan unas aplicaciones de otras con la finalidad de conseguir la mayor seguridad posible.

En la Figura 2.2 podemos apreciar la estructura[6] de una aplicación Android.

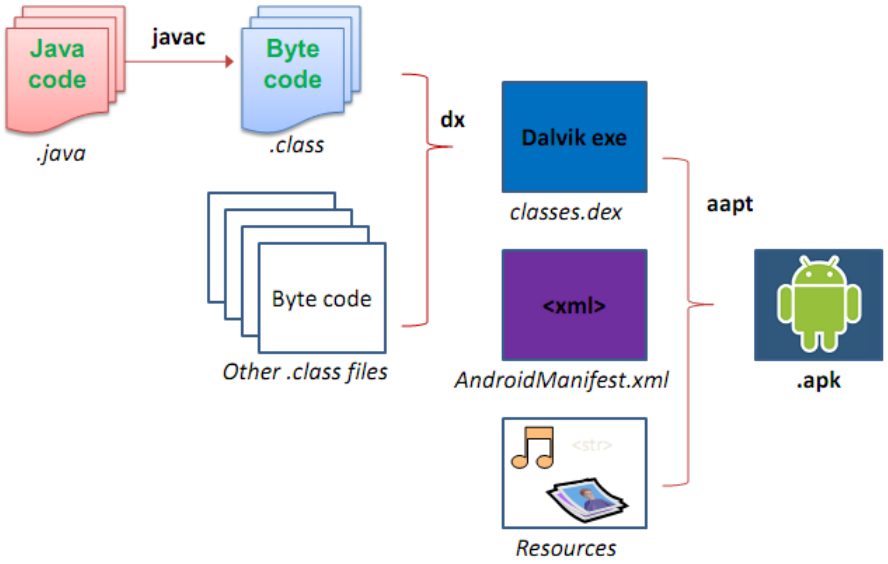


Figura 2.2: Estructura de una aplicación Android

2.3.2. Fichero AndroidManifest

El fichero AndroidManifest.xml[5] es un fichero que debe estar siempre presente en el directorio raíz de la aplicación para así poder realizar su ejecución. Proporciona al sistema operativo información esencial sobre la aplicación como:

- Nombre del paquete Java de la aplicación. Es único y sirve como identificador de la aplicación.
- Versión mínima de la API de Android requerida para la ejecución de la aplicación.
- Listado de bibliotecas con las que está vinculada la aplicación.
- Permisos. Existen una serie de recursos del sistema Android a los cuales no puede acceder una aplicación con naturalidad. Para realizar unas operaciones específicas aparecieron los permisos. Permiten controlar los recursos que usan las aplicaciones lo que proporciona un mayor control y seguridad en el sistema de nuestro dispositivo.

  Una aplicación por defecto no tiene permisos que puedan acarrear connotaciones negativas a otras aplicaciones o al sistema. Los permisos que requiere una aplicación deben indicarse en el fichero AndroidManifest de la aplicación.
- Acciones. Las acciones nos permiten determinar que ejecutar en determinado momento. Generalmente están dentro de las actividades y definen en un string que actividad de la aplicación, del sistema o de otra aplicación externa ejecutar.
- Actividades. Podríamos definirlas como las tareas que puede realizar la aplicación. Cada actividad puede contener una o varias acciones internas u externas para obtener y ofrecer toda la información que la aplicación necesita.

3. Metodología

Como el objetivo del trabajo es analizar posibles acciones malintencionadas la aplicación a analizar, primero necesitamos extraer información sobre las acciones que ésta puede realizar.

En este capítulo abordaremos la forma en la que se extraerá la información de las aplicaciones y la obtención de una conclusión sobre la peligrosidad de la aplicación mediante el análisis de la información estática y dinámica obtenida. Adicionalmente se propondrá un boceto de como el usuario podrá analizar una aplicación y de la forma en la que se mostrarán los resultados.

3.1. Arquitectura del sistema

La arquitectura del sistema consta de 4 componentes principales como se muestra en la Figura 3.1.

- *Preprocesamiento.* Se descomprime el APK de cada aplicación para obtener el fichero AndroidManifest.xml. De él se extraen los permisos y actividades. También de decompila la aplicación para obtener las clases Java y buscar líneas de código que cumplan determinados patrones. Aquí también monitorizaremos el tráfico de la maquina virtual Android y sus llamadas al sistema para obtener información dinámica de la aplicación.

- *Análisis de la aplicación.* Con la información estática y dinámica extraída del paso anterior se realizará un análisis de la información para obtener una valoración que determine si la aplicación analizada contiene malware.

- *Interfaz de usuario.* Muestra el usuario de una forma sencilla e intuitiva toda la información referente al análisis de la aplicación.

- *Bases de datos.* Se disponen de dos bases de datos, una destinada para almacenar ejemplos de aplicaciones que conocemos su naturaleza y así establecer los criterios para valorar aplicaciones desconocidas y otra donde se almacena toda la información que se extrae de las aplicaciones que analizamos.

Hubiese sido posible la fusión del preprocesamiento con el análisis pero esto nos dificultaría la implementación de nuevos tipos de análisis[18][36] al estar la valoración de información a la par que la extracción.

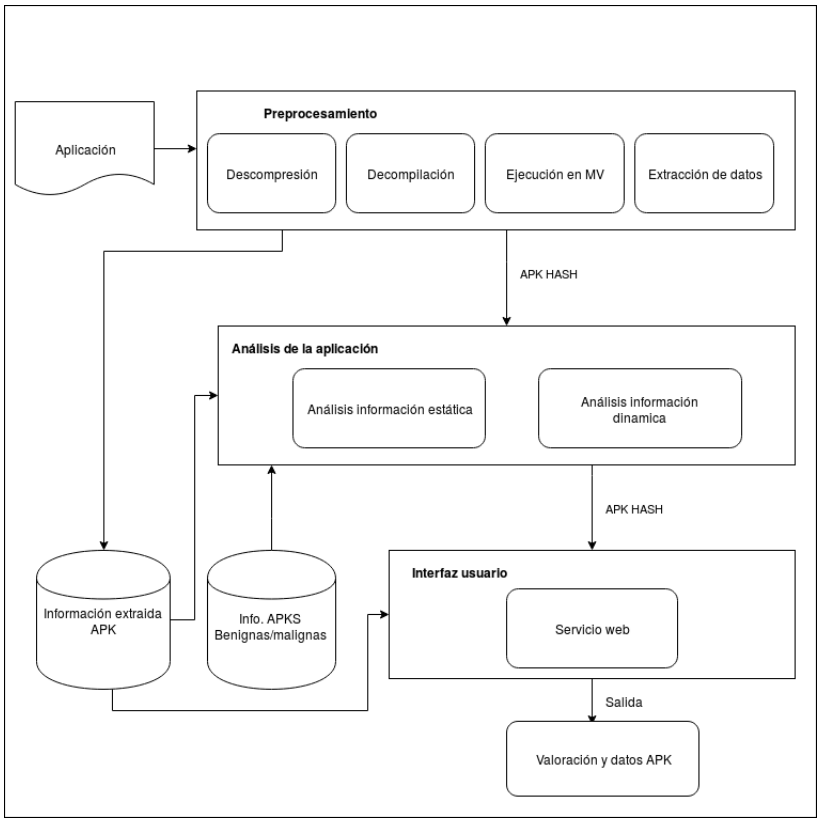


Figura 3.1: Arquitectura del sistema

3.2. Fase de extracción de información

De una aplicación Android se puede obtener multitud de información pero gran parte de esta puede que no nos sea de nuestra utilidad. Junto a qué extraer, debemos plantearnos la forma de obtener la información.

Como se ha mencionado anteriormente todas las aplicaciones Android están desarrolladas en Java. Si accedemos al código donde se implementa la aplicación podremos examinarlo para intentar detectar acciones no deseadas. Para obtener tanto el código como el fichero AndroidManifest.xml que incluye toda la información esencial de la aplicación, deberemos usar ingeniería inversa[19] ya que partimos de un fichero compilado en formato .apk. Estos pasos son los que definirían el análisis estático[34].

Mediante el análisis estático por desgracia no podemos obtener toda la información relevante para intentar detectar si una aplicación esconde algún tipo de malware. Necesitaremos ejecutar la aplicación en una máquina virtual para así poder obtener los datos relevantes que no podemos extraer de sólo analizar el código. Son por ejemplo las IPs a las que se conecta la aplicación o las llamadas al sistema. Esto forma parte del análisis dinámico[23].

3.2.1. Análisis estático

Lo primero que haremos es obtener información del fichero contenido en la aplicación AndroidManifest.xml. De aquí podremos obtener los permisos, las acciones y las actividades de la aplicación.

En los referente a los permisos los guardaremos de dos formas, una donde se almacena-

rán todos los permisos independientemente de si comprometen información personal del usuario y otra donde solo se almacenarán los que pongan en riesgo los datos personales. Los primeros pese a no comprometer información personal, pueden ser muy comunes en aplicaciones maliciosas otorgando a la aplicación diversas capacidades como instalar terceras aplicaciones o montar y desmontar sistemas de archivos. Por tanto, los almacenaremos para realizar una valoración sobre la peligrosidad de la aplicación basándose en los permisos. En Tabla 3.1 se pueden apreciar los permisos que Android ha determinado[4] que comprometen datos personales del usuario.

Grupo de permisos	Permisos
CALENDAR	• READ_CALENDAR • WRITE_CALENDAR
CAMERA	• CAMERA
CONTACTS	• READ_CONTACTS • WRITE_CONTACTS • GET_ACCOUNTS
LOCATION	• ACCESS_FINE_LOCATION • ACCESS_COARSE_LOCATION
MICROPHONE	• RECORD_AUDIO
PHONE	• READ_PHONE_STATE • CALL_PHONE • READ_CALL_LOG • WRITE_CALL_LOG • ADD_VOICEMAIL • USE_SIP • PROCESS_OUTGOING_CALLS
SENSORS	• BODY_SENSORS
SMS	• SEND_SMS • RECEIVE_SMS • READ_SMS • RECEIVE_WAP_PUSH • RECEIVE_MMS
STORAGE	• READ_EXTERNAL_STORAGE • WRITE_EXTERNAL_STORAGE

Tabla 3.1: Permisos peligrosos en Android

Las acciones y las actividades nos serán importantes en el análisis dinámico ya que con ello podremos simular el uso de la aplicación en una máquina virtual.

El siguiente paso es analizar las clases Java de la aplicación. Mediante ingeniería inversa podemos obtener las clases. En el código deberemos intentar localizar las posibles líneas que escondan la ejecución de algún tipo de malware. Esto en muchas ocasiones puede que nos devuelva información incorrecta ya que no siempre será algo que perjudique nuestro dispositivo. De este modo debemos ser precisos en la búsqueda si queremos obtener información útil que sirva al usuario final tomar la decisión de usar o no la aplicación y no de transmitirle la sensación equivocada como puede ser un falso positivo o negativo[13].

Los datos que podemos extraer de forma automática con facilidad son los siguientes:

- *Instalaciones de terceros.* La aplicación puede instalar terceras aplicaciones o frameworks que permitan explotar alguna vulnerabilidad, instalarlos cualquier tipo de malware o simplemente instalarnos alguna aplicación con la que hayan pactado tener algún beneficio económico (afiliados, referidos...).

- *Inyección de código en Dalvik.* Comprueba si en el código hay algo relacionado con la inyección de código en la máquina Dalvik con la intencionalidad de escalar privilegios aprovechando alguna vulnerabilidad y ejecutar código malicioso.
- *Uso de root.* Se controla el uso del directorio root para determinar si la aplicación requiere permisos de superusuario lo cual es nada recomendable.
- *IPs declaradas en código.* La aplicación puede que tenga declarada una o varias IPs en el código para así poder conectarse a posteriori y transmitir información del usuario u obtener terceras aplicaciones.
- *Envío de SMS.* Detectar el envío de SMS por parte de la aplicación. Esto no determinada que la aplicación contenga malware ya que simplemente puede ser una verificación del número de teléfono pero si es cierto que en muchas ocasiones se envían mensajes de texto en segundo plano a servicios de tarificación premium sin el conocimiento del usuario.

3.2.2. Análisis dinámico

Mediante la simulación del uso de la aplicación en una máquina virtual obtendremos toda la información que no hemos podido conseguir analizando el fichero .apk.

La primera información relevante que se puede obtener con facilidad es el tráfico de red que realiza la aplicación. Monitorizando el tráfico de la máquina virtual obtenemos la lista de IPs donde intenta conectarse.

Otra información que podemos obtener en el análisis dinámico es la relacionada con las acciones que realiza la aplicación. Podemos obtener las llamadas al sistema y las acciones que realiza la aplicación. Así, con estos datos podemos detectar un comportamiento anómalo y comunicárselo al usuario. Un claro ejemplo que podríamos detectar en un log es el envío de SMS premium de forma automática por parte de una aplicación.

3.3. Fase de análisis de la información extraída

3.3.1. Análisis estático basado en permisos

¿Cómo podemos determinar si una aplicación contiene malware o no mediante sus permisos? Generalmente las aplicaciones con malware siguen patrones similares donde poseen requieren los mismos permisos[31]. Por tanto la mejor opción será recopilar los permisos más usados que soliciten aplicaciones maliciosas y comparar las coincidencias con los que solicita la aplicación a examinar.

Esto puede llevarnos a obtener información errónea ya que muchos permisos también serán usados por aplicaciones que no provoquen ningún prejuicio al usuario por lo que para evitar esto dentro de lo posible obtendremos dos listados, uno para aplicaciones maliciosas y otro para aplicaciones comunes. De esta forma en los permisos que existan mayor diferencia de apariciones entre aplicaciones malignas y no malignas serán los que serán más significativos para determinar si la aplicación es malware o no.

Con los permisos más significativos se hará un sistema de puntuaciones donde cada permiso tendrá un valor en función de la diferencia de apariciones que haya tenido entre los dos tipos de aplicaciones. Las aplicaciones comunes tendrán un valor mínimo para que perjudiquen el análisis lo menos posible. El resultado se puede expresar con una puntuación acotada

en un intervalo, por ejemplo 0 y 1 siendo 0 sin malware y 1 con la total certeza de que la aplicación posee malware.

3.3.2. Análisis de información dinámica

El análisis de información dinámica consta de dos partes: la comprobación de las IPs en listas negras y la obtención de una valoración mediante las llamadas al sistema.
Por una primera parte las IPs obtenidas mediante la ejecución de la aplicación en una máquina virtual las comprobamos por las diversas *blacklists*[26] existentes para así determinar si se está intentando conectar a sitios maliciosos. Las IPs que estén en las listas negras se pueden localizar su país de origen para darle al usuario una información más fidedigna. Esta información se almacenará en la base de datos para que el usuario sea consciente de la peligrosidad de las peticiones que realiza su aplicación pudiendo estar comprometiendo su información personal.

Posiblemente la información obtenida no nos sea suficiente para tomar una conclusión sobre la confianza en una aplicación pero, junto a los datos del análisis estático y partiendo del uso principal de la aplicación, el usuario puede determinar si existen acciones anómalas que no tienen relación alguna con la función principal que viene a satisfacer la aplicación. Sería muy extraño por ejemplo que una aplicación que hace que nuestro smartphone actúe como una linterna requiera el envío de SMS y se conecte a IPs maliciosas.

La segunda parte del análisis es una valoración de las llamadas al sistema. Es difícil determinar mediante las llamadas al sistema si una aplicación corre malware en su interior ya que, la mayor parte de las llamadas son siempre las mismas en cualquier tipo de aplicación. Apreciadamente existen un pequeño grupo de llamadas que por norma general no aparecen en una aplicación común pero sí aparecen con cierta frecuencia en aplicaciones maliciosas. Varios estudios[22][27] nos dicen que basádonos en las apariciones de estas llamadas, podemos detectar malware dándole una valoración numérica a dichas llamadas o haciendo estudios estadísticos. De esta forma, le indicaremos una valoración numérica a estás extrañas llamadas y devolveremos una calificación numérica que determine si posee malware o no según sus llamadas al sistema.

3.4. Entorno de usuario

Para que el usuario pueda realizar el análisis de la aplicación, está debe tener una interfaz sencilla con la que pueda interactuar con facilidad con el usuario. Como el usuario puede tener la aplicación en cualquier tipo de dispositivo la mejor manera es que sea una interfaz web y no de escritorio, facilitando así el uso de otros sistemas operativos que no tengan relación con Android. El usuario podrá enviar el fichero desde un formulario web sin la necesidad de instalar nada, solo tendrá que acceder a una url.

Para la consulta de los resultados será similar a la subida de ficheros. Se dispondrá de un buscador para localizar la aplicación analizada y un listado con todas ellas por si el usuario desea ver los resultados de otras aplicaciones no enviadas por él. Todo esto desde un sitio web.

4. Desarrollo

4.1. Esquema del despliegue

En la Figura 4.1 podemos ver el esquema del despliegue donde las dos máquinas que se usan se conectan por el protocolo SSH y se realiza una replicación de las bases de datos mediante maestro-esclavo.

Podríamos haber dejado la capa de interfaz de usuario en la misma máquina que el preprocesamiento y el análisis pero esto nos repercutiría negativamente en la disponibilidad en la red de los datos. El uso de un VPS nos garantiza un uptime cercano al 99,99 %.

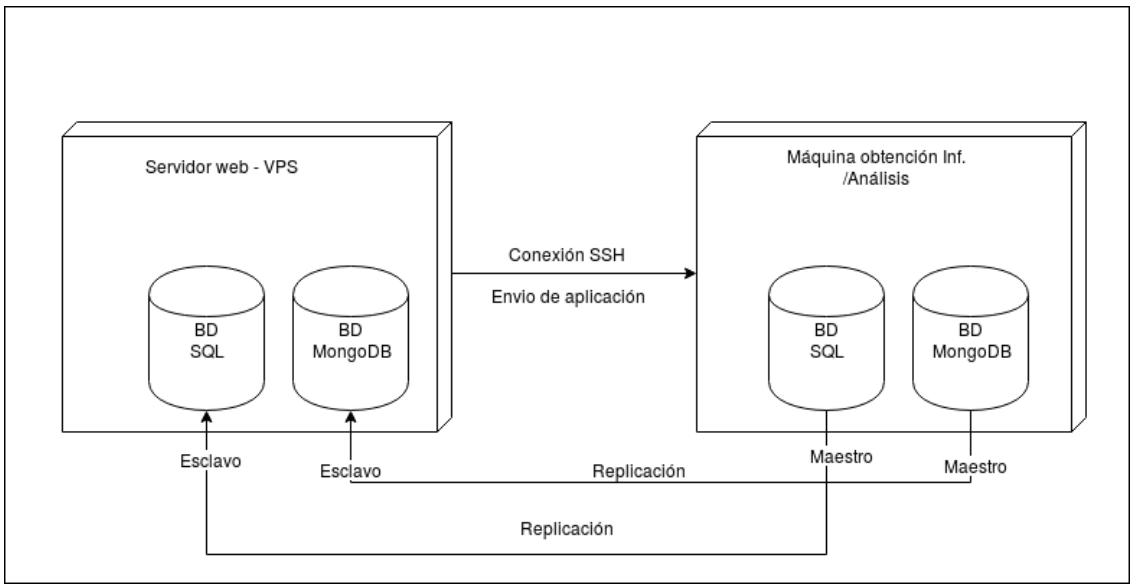


Figura 4.1: Esquema del despliegue

4.2. Lenguajes de desarrollo

Para el desarrollo de este trabajo se ha utilizado en en su mayoría el lenguaje de programación Python en su versión 3.4. Python ofrece como ventajas su sencillez a la hora de su aprendizaje, la poca cantidad de líneas necesarias para resolver un problema, su variedad de módulos que pueden sernos de interés a la hora de la realización de este trabajo y la posibilidad de su uso para el desarrollo web.

Además se ha usado HTML y CSS para el diseño de la página web que sirve como interfaz de usuario y órdenes shell para el lanzamiento de los códigos en Python.

4.3. Servicios previos

Para la realización del trabajo necesitamos una serie de servicios previos desplegados antes de comenzar a desarrollar el código que provea al usuario de los diferentes análisis de las aplicaciones.

Partiremos por la máquina virtual Android. Se necesita el SDK de Android contenido en Android Studio[3]. Este incluye herramientas esenciales para la comunicación con la máquina virtual.

ADB[2] (Android Debug Bridge) es una herramienta de línea de órdenes que nos permite comunicarnos con la máquina virtual o un dispositivo Android físico. Nos permitirá instalar y lanzar actividades de las aplicaciones a la hora de realizar el análisis dinámico.

La siguiente aplicación a instalar será la máquina virtual Genymotion[21]. Se podría usar la máquina virtual ofrecida por Android Studio pero dicha máquina no es capaz de ejecutar aplicaciones desarrolladas para dispositivos ARM. Genymotion mediante un parche[10], simula las instrucciones de ARM y las puede ejecutar en su mayoría. De esta forma se consiguen analizar la mayoría de aplicaciones desarrolladas para generaciones anteriores de smartphones ya que las actuales son compatibles para x86 en casi su totalidad.

El principal inconveniente de usar máquinas virtuales es la necesidad de usar un ordenador o servidor que las aloje con interfaz gráfica. Las máquinas virtuales Android que disponen de todas sus funcionalidades no permiten la ejecución headless lo que nos dificulta su ejecución en un servicio en la nube o en un servidor dedicado o VPS en una empresa de hosting. La mejor solución sería usar el servicio que ofrece AWS(Amazon Web Services)[1] el cual permite ejecutar instancias de Genymotion con su lanzamiento y destrucción mediante línea de comandos. Tal vez el lado negativo de AWS es su coste, que ronda los 100 dólares mensuales. Por este hecho descartamos esta opción.

Creamos varias máquinas con las distintas versiones de Android, o al menos las más conocidas, para así aumentar la compatibilidad con las distintas aplicaciones. A cada máquina le aplicamos el parche para conseguir la compatibilidad con aplicaciones ARM.

Los servicios web los separaremos de la máquina donde se ejecutan las aplicaciones para así ofrecer una mayor disponibilidad al usuario a la hora de consultar la aplicación. Para estos servicios se requiere un servidor dedicado o un VPS. En nuestro caso se va a usar un VPS de la empresa Arubacloud[12] que solo tiene un coste de un euro al mes y cumple con nuestras necesidades.

VPS Arubacloud	
Sistema Operativo	Linux
CPU	1 Core Intel® Xeon® E5-2650L v4
Memoria RAM	2 GB
Almacenamiento	20 GB SSD
Transferencia	2 TB

Tabla 4.1: Características VPS de Arubacloud

En el VPS instalaremos el servidor web Apache junto a Django[16] y virtualenv[39] ya que el desarrollo web será en Python. La instalación de Django es my sencilla y simplemente es introducir pip install django. La configuración de Apache para Django[17] la podemos ver en el anexo A.

De dicho código tenemos las siguientes variables a destacar que son esenciales para la correcta configuración del servidor web.

- En `WSGIDaemonProcess` se indica la ruta donde está ubicado Python en el entorno virtual (virtualenv).
- `WSGIProcessGroup` Es el grupo donde estará el demonio que lanza el framework de Django.

- A `WSGIScriptAlias` se le indica la ruta de la interfaz de Django para conectarlo con el servidor web.
- `Directory`. En estos de definen las rutas de los ficheros estáticos y multimedia.

Como se ha visto en la configuración se ha usado un nombre de dominio y no localhost. Esto simplemente es recomendable para ofrecer al usuario una mayor facilidad a la hora de analizar las aplicaciones pudiendo acceder mediante una url sencilla como puede ser un nombre de dominio y no una IP. Se ha registrado el dominio apkcheck.com para esta ocasión.

Un último detalle opcional para el servidor web sería añadir un certificado SSL[14]. Este nos garantiza que los datos circulan por la red de una manera íntegra y segura mediante un cifrado de claves. De esta manera no podrá obtener información una tercera persona mediante un *sniffer*.

Existen diversos tipos con mayor o menor confianza y variedad de precio. Para lo que vamos a realizar en este trabajo lo vamos a generar nosotros mismos por lo que nos saldrá totalmente gratis. Usaremos Let's Encrypt[20] el cual nos ofrece un script shell[32] que nos generará el certificado y nos lo instalará en el servidor web de forma automática.

Como disponemos de dos computadores, uno para realizar los análisis con interfaz gŕafica y otro para actuar como servicio web, debemos buscar una manera para almacenar la información de los resultados del primer ordenador al VPS que mostrará los datos al usuario. Para ello usaremos MySQL como base de datos relacional y MongoDB[28] para almacenar los datos difícil de almacenar en tablas. Estos servicios actuarán como maestros en la máquina que analiza las aplicaciones y como esclavos en la máquina que ofrece el servicio web. De esta forma toda información obtenida en un análisis se replicará de forma automática y siempre tendremos disponibilidad de los datos si accedemos al entorno web.

En primer lugar instalaremos MySQL en ambas máquinas. Este paso es tremendamente sencillo y simplemente es introducir el comando sudo apt-get install mysql-server mysql-client. En la instalación se preguntará el usuario y contraseña deseado para el servidor sql y ya estará totalmente instalado.

Como queremos una configuración maestro-esclavo[35] lo primero que debemos hacer es modificar el fichero de configuración de MySQL en la máquina que actuará como maestro. La configuración se encuentra en la ruta /etc/mysql/my.cnf. Introduciremos los siguientes datos:

```
bind-address = 0.0.0.0
server-id = 1
log-bin = mysql-bin
binlog-ignore-db = "mysql"
```

Lo importante de esta configuración es la `bind-address` que acepta cualquier conexión interna o externa, el `server-id` que es el identificador del servidor y el `log-bin` que es el nombre del log binario.

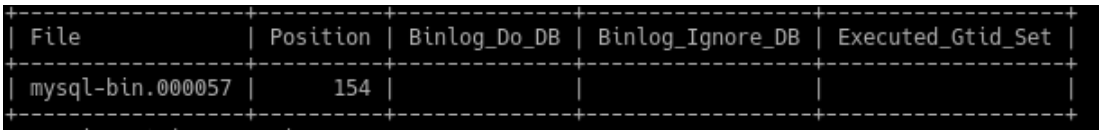
Reiniciamos el servicio y creamos un nuevo usuario y le asignamos permisos de réplica. Esto lo realizaríamos de la siguiente forma:

```
CREATE USER replica IDENTIFIED BY 'password';
```

```
GRANT REPLICATION SLAVE ON *.* TO 'replica'@'%' IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
```

El usuario y la contraseña pueden ser los deseados aunque se recomienda una combinación compleja para evitar el éxito de cualquier tipo de ataque por diccionario o fuerza bruta.

Solo nos falta introducir "SHOW MASTER STATUS"para obtener la información del fichero sql y su posición para posteriormente indicársela a la máquina esclava.



File	Position	Binlog_Do_DB	Binlog_Ignore_DB	Executed_Gtid_Set
mysql-bin.000057	154			

Figura 4.2: Fichero MySQL y posición

Ahora accedemos a la máquina esclava y modificamos el fichero /etc/mysql/my.cnf. En este caso solo añadiremos al fichero `server-id=2`. Reiniciamos el servicio y accedemos a MySQL. En la línea de comandos introduciremos lo siguiente:

```
CHANGE MASTER TO
MASTER_HOST='host',
MASTER_USER='replica',
MASTER_PASSWORD='password',
MASTER_PORT=3306,
MASTER_LOG_FILE='mysql-bin.000057',
MASTER_LOG_POS=154,
MASTER_CONNECT_RETRY=10;
```

El `host` deberá ser la IP de la máquina maestra o dominio si disponemos de él, el usuario y la contraseña creados para la réplica, el puerto donde opera MySQL y el nombre del fichero y la posición que hemos obtenido anteriormente. Le indicamos que comience a funcionar con start slave y todo comenzará a replicarse de forma automática.

Una configuración[29] similar se deberá hacer con MongoDB. Instalamos MongoDB en sendas máquinas mediante:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80
 --recv 0C49F3730359A14518585931BC711F9BA15703C6

echo "deb [ arch=amd64,arm64 ]
http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4 multiverse"
 | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list

sudo apt-get update
sudo apt-get install -y mongodb-org
```

En la máquina maestra lanzamos el servicio de MongoDB indicándole que es maestro mediante:

```
mongod --master --fork --logpath /var/log/mongod.log
```

Realizado esto accedemos a la máquina del servicio web, es decir, la esclava y lanzamos el servicio de MongoDB como esclavo indicándole la IP y el puerto de la máquina maestra.

```
mongod --fork --slave --source <IPMaquinaMaestra><:<PuertoMaquinaMaestra>>
 --logpath /var/log/mongod.log
```

Con esto ya tenemos funcionando correctamente MongoDB y los datos serán replicados de forma automática.

4.4. Obtención de información

Para la ejecución de los dos análisis que se van a realizar se usará un script shell con la finalidad de cohesionar ambos y automatizar todo el proceso. La forma en la que se llamará el script y el envío de las aplicaciones se detallará junto a la interfaz web, en este momento solo nos centraremos en la realización de los análisis y la generación del script al cual le pasaremos el fichero apk como argumento.

Para almacenar el estado del análisis y los datos principales de la aplicación lo primero que haremos es generar una tabla para alojar dicha información. Esto también nos servirá para evitar repetir análisis de la misma aplicación.

```
CREATE TABLE aplicaciones(
HASH varchar(32) primary key not null,
nombre_paquete varchar(60),
version_android varchar(20),
estado varchar(15) not null,
fecha DATETIME DEFAULT CURRENT_TIMESTAMP);
```

El HASH se obtendrá mediante el algoritmo MD5[37] lo que nos garantiza que el fichero analizado tendrá un identificador único y que no volverá a analizarse más veces con el fin de evitar carga innecesaria en las máquinas.

El nombre_paquete será el `package name` contenido en el fichero AndroidManifest de la aplicación. Este nos permitirá lanzar la aplicación en el análisis dinámico y que el usuario pueda buscar por nombre de paquete en la interfaz web.

Version_android aloja la versión de Android para la que está desarrollada la aplicación. Con ello podremos saber en que máquina virtual es más idónea ejecutar la aplicación.

El estado nos indica el progreso de los análisis. Los posibles estados son:

- Creado. Se produce al introducirse los datos básicos de la apk en la base de datos.
- A_Manifest. Indica que se ha analizado el fichero AndroidManifest.
- Estatico. Se ha analizado el código java de la aplicación.
- Get_IPs. Se han obtenido las IPs a las que se comunica la aplicación.
- Bad_IPs Se han comprobado que IPs están en las listas negras.
- Finalizado. Se han analizado las llamadas al sistema.

Lo primero a desarrollar en el script es creación de los datos de la aplicación en la base de datos y la obtención del HASH. El siguiente código desarrollado en Python obtiene el código MD5 de la aplicación y añade los datos de importancia en MySQL.

```
hash_md5 = hashlib.md5()

with open(sys.argv[1], "rb") as f:
for chunk in iter(lambda: f.read(4096), b""):
hash_md5.update(chunk)

c = conn.cursor()
c_hash=hash_md5.hexdigest()
c.execute('SELECT * FROM aplicaciones WHERE HASH=%s', (c_hash,))
data=c.fetchall()
if len(data)==0:
c.execute("INSERT INTO aplicaciones (HASH, estado) VALUES
 (%s, %s,'Creado')",(c_hash))
conn.commit()
conn.close()

print c_hash
```

De esta forma obtenemos el MD5 del fichero, comprobamos que aún no existe la información de la aplicación a analizar e introducimos los datos de interés.

Realizados los pasos anteriores ya si podemos comenzar a desarrollar los análisis.

4.4.1. Obtención de información estática

Como explicamos en la metodología, lo primero que se hará en el análisis estático es examinar el fichero Androidmanifest.xml para extraer la información necesaria. Para acceder al fichero xml necesitamos usar ingeniería inversa para así conseguir descodificar la aplicación y obtener los recursos de una forma casi similar a la original. Para dicha tarea usaremos la herramienta Apktool[11].

Su funcionamiento es muy sencillo, simplemente le pasamos a la herramienta la aplicación como parámetro y obtendrá los ficheros de esta bajo un directorio con el mismo nombre de la aplicación. Por si existiese el directorio podemos usar el argumento -f y así sobreescribirlo.

```
apktool d fichero.apk -f
```

Obtendríamos un listado de ficheros como este donde el fichero destacable es el AndroidManifest.

```
AndroidManifest.xml  apktool.yml  assets  lib  original  res  smali
smali_classes2  unknown
```

Parseamos el fichero para obtener el nombre del paquete, los permisos, las acciones y las actividades que ya explicamos en la metodología. Esto lo llevamos a cabo mediante el código del anexo B al cual le pasamos el HASH de la aplicación y la ruta del fichero AndroidManifest como argumento. No todo el código se ha indicado en el anexo ya que se repite las inserciones de cada tipo de dato en la base de datos, con mostrar los distintos tipos de inserciones es suficiente.

Parseamos el xml con la biblioteca xml.dom.minidom. Mediante las funciones getElementsByTagName y getAttribute obtenemos la información relevante y la añadimos en tablas en MongoDB asociadas al HASH de la aplicación. En lo referente a los permisos solo añadimos los peligrosos los cuales los tenemos en una lista para así, comparar si son nocivos

o no. Hecho esto, cambiamos el estado de la aplicación a A_Manifest.

Cabe destacar del código cómo se obtienen los distintos tipos de datos del fichero AndroidManifest. Esto lo conseguimos mediante:

```
package_name = dom.getElementsByTagName('manifest')[0].
getAttribute('package')
target_sdk = dom.getElementsByTagName('manifest')[0].
getAttribute('platformBuildVersionCode')
nodes = dom.getElementsByTagName('uses-permission')
nodes = dom.getElementsByTagName('activity')
nodes = dom.getElementsByTagName('action')
```

Aplicándose a Nombre del paquete, versión SDK, permisos, actividades y acciones, respectivamente.

A continuación, necesitamos analizar las clases java que forman la aplicación. Obtener estas clases es más complicado y también deberemos hacerlo mediante ingeniería inversa. Para lograrlo nos ayudaremos de las herramientas dex2jar y jd-core.

Dex2jar[30] nos permite convertir los fichero .dex en .jar. Los ficheros .dex son ficheros Dalvik ejecutables contenidos en el fichero .apk. Ejecutamos la herramienta y obtenemos el fichero java compilado .jar.

```
./d2j-dex2jar.sh -f fichero.apk -o fichero.jar
```

Ahora que tenemos el fichero java ejecutable debemos obtener a partir de él las clases java. Con jd-core lo podremos llevar a cabo de la siguiente forma:

```
java -jar jd-core.jar fichero.jar directorio-java
```

Ya tenemos las clases java que forman la aplicación Android. En ellas debemos buscar las líneas de código que puedan ser indicios de la existencia de malware en la aplicación.

La primera cosa a buscar serán los intentos de instalaciones de terceras aplicaciones. La búsqueda en el directorio de las clases java la haremos mediante la orden grep. Los resultados los guardaremos en en ficheros de texto para posteriormente introducirlos en MongoDB. La búsqueda sería de la siguiente forma:

```
grep -i ".*\.apk[^ a-z]" -r $dir-java/ | while read line; do   echo "$line";
 done > $dir-apks.txt

grep -i "application/vnd.android" -r $dir-java/ | while read line; do echo
"$line"; done > $dir-apks_vnd.txt
```

Realizaremos lo mismo con las inyecciones de código malicioso en la máquina Dalvik.

```
grep -i "DexClassLoader InjectCall" -r $dir-java/ | while read line; do echo
"$line"; done > $dir-inject.txt
```

El uso del directorio root es en muchas ocasiones una puerta trasera para introducir malware al dispositivo del usuario por lo que también buscaremos esto.

```
grep -i "/bin/su" -r $dir-java/ | while read line; do   echo  "$line";
done > $dir-root.txt
```

Generalmente si se declaran IPs en el código es para enviar o recibir información comprometida del usuario. Una aplicación común puede conectar a un servidor mediante su URL pero en ningún caso es una situación normal acceder a una IP concreta preestablecida en el código.

```
grep -i "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?
[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|
[01]?[0-9][0-9]?)" -r $dir-java/ | while read line; do echo  "$line"; done >
 $dir-ips.txt
```

En envío de SMS es una de las acciones más comunes por parte de las aplicaciones maliciosas ya que es una forma rápida de obtener dinero mediante los SMS premium. Por tanto, comprobamos si existe en el código una llamada al envío de SMS de nuestro dispositivo.

```
grep -i "sendTextMessage" -r $dir-java/ | while read line; do echo  "$line";
 done > $dir-sms.txt
```

Buscamos el uso de payloads que exploten alguna vulnerabilidad de nuestro sistema para ejecutar código malicioso.

```
grep -i "payload" -r $dir-java/ | while read line; do  echo  "$line"; done >
$dir-payload.txt
```

La información obtenida se almacena en ficheros de texto. Para almacenarla en MongoDB ejecutamos el siguiente código donde se le pasa como argumentos el HASH de la aplicación y la ruta de cada fichero. La única diferencia para cada fichero es la tabla donde se insertará modificando así el término datos de db.datos.update por apk_install, static_inject, static_root, static_ips o static_sms.

```
with open(sys.argv[2]) as f:

for line in f:
class_name= line.split(':', 1)[0]
mensaje = line.split(" ",1)[1].lstrip()
db.datos.update({"HASH": HASH}, {'$push':{ "log" : {'Clase' : class_name,
 'Mensaje' : mensaje }}}, True)

f.close()
```

4.4.2. Obtención de información dinámica

En el análisis dinámico se pretende detectar posibles indicios de malware en tiempos de ejecución. Para ello mediante ADB se instala la aplicación en la máquina virtual y se ejecutan las principales actividades. De esta manera podemos monitorizar el tráfico entrante y saliente para comprobar si se intenta conectar a IPs maliciosas así como revisar los logs

para asegurarnos que no envia SMS a servicios premium.

En el análisis dinámico guardamos la versión de Android para la que está desarrollada la aplicación. La recuperaremos para así determinar que versión usar en una máquina virtual.

```
version_android=$(python obtener_version.py $hashapk)
```

```
import sys
import MySQLdb

conn = MySQLdb.connect()
c = conn.cursor()

hash_apk=sys.argv[1]


c.execute("SELECT version_android FROM aplicaciones WHERE
HASH=%s",(hash_apk,))
tmp=c.fetchone()[0]
if tmp is '':
       print 17
else:
       print tmp
```

Si la aplicación no indica que versión usar usaremos por defecto Android 4.2. Con esta información determinamos el nombre que tiene definido la máquina virtual en genymotion para así lanzarla mediante un simple comando.

```
if [ $version_android -le 10 ]; then
maquina_android=Android2.3.7
elif [ $version_android -le 17 ]; then
maquina_android=Android4.2
elif [ $version_android -le 20 ]; then
maquina_android=Android4.4
elif [ $version_android -le 22 ]; then
maquina_android=Android5.1
elif [ $version_android -ge 23 ]; then
maquina_android=Android6
fi
```

Antes de lanzar la máquina empezamos a monitorizar el tráfico el tráfico mediante un comando que nos ofrece Virtualbox[38]. Le indicamos la máquina que necesitamos monitorizar.

```
VBoxManage modifyvm $maquina_android --nictrace2 on --nictracefile2 $dir.cap &
```

Lanzamos la máquina virtual con la versión Android seleccionada para así proceder a la instalación de la aplicación.

```
player --vm-name $maquina_android &
```

Antes de intentar conectar con la máquina virtual esperamos a que se inicie correctamente. Mediante la orden sleep podemos esperar un tiempo determinado hasta que la máquina

esté totalmente funcional. Esperando 40 segundos es suficiente.

Como no sabemos la IP y el puerto de la máquina virtual tendremos que obtenerlo con la orden adb devices y guardarlo en una variable. Con esto simplemente será llamar a adb connect. Esto lo realizaremos de la siguiente forma:

```
ip_android=$(adb devices | grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b")
adb connect $ip_android
```

Instalamos la aplicación mediante adb y esperamos unos seguidos a que se haya realizado correctamente.

```
adb install -r $dir.apk
sleep 3
```

Ejecutamos la actividad principal del apk, es decir, lanzamos la aplicación.

```
adb shell monkey -p $nombre_paquete -c android.intent.category.LAUNCHER 1
```

Para obtener las llamadas al sistema lanzaremos strace[9] en la máquina virtual mediante adb. Las salvaremos en un fichero para procesar la información posteriormente.

```
adb shell 'set `ps | grep' $nombre_paquete'` && strace -p $2' >
$dir-syscall.txt &
```

Obtenemos de MongoDB las actividades de la aplicación que habíamos almacenado en el análisis estático. Así podremos ejecutarlas y capturar todas las conexiones y llamadas al sistema que realiza la aplicación en su totalidad.

```
actividades_apk=$(python launch.py $hashapk 1 | tr -d "''" | tr -d '[],')
```

Vamos lanzando cada actividad dándole unos segundos a cada actividad para que pueda ejecutarse correctamente.

Concluida la ejecución de actividades detenemos strace mediante el siguiente comando.

```
adb shell killall strace
```

Desinstalamos la aplicación para que la máquina este limpia en el próximo análisis y no se vean los resultados alterados.

```
adb uninstall $nombre_paquete
```

Detenemos la máquina virtual. Al no tener la versión de pago de genymotion no podremos usar linea de comandos y deberemos hacerlo mediante virtualbox y matando el proceso.

```
vboxmanage controlvm $maquina_android poweroff || true
ps x | grep "player --vm-name $maquina_android" | awk '{print $1}' |
xargs kill
```

Ya con la máquina apagada paramos la captura de paquetes para procesarlos a continuación.

```
VBoxManage modifyvm $maquina_android --nictrace2 off --nictracefile2
 $dir.cap
```

Parseamos el fichero .cap donde está toda la información del tráfico de la máquina virtual y extraemos las IPs con las que se ha conectado la máquina Android. Llamamos al código de la siguiente forma.

```
python parsercap.py $dir.cap $hashapk
```

La implementación que extrae las IPs y las guarda en MongoDB la tenemos en el apartado C. Lo más reseñable de la implementación es como obtenemos cada IP de fichero cap, usamos una expresión regular para evitar obtener IPs locales y de no coincidir la IP con una local la añadimos en la base de datos.

```
f = open(filename)
pcap = dpkt.pcap.Reader(f)
IPs = Set([])

rex=re.compile("/(^127\.)|(^192\.168\.)|(^10\.)|(^0\.)
|(^172\.1[6-9]\.)|(^172\.2[0-9]\.)|(^172\.3[0-1]\.)/")
for ts, buf in pcap:
       eth=dpkt.ethernet.Ethernet(buf)
       if eth.type!=dpkt.ethernet.ETH_TYPE_IP:
              continue

       ip=eth.data

       destino = socket.inet_ntoa(ip.dst)
       src=dst_ip_addr_str=socket.inet_ntoa(ip.src)

       if rex.match(destino) == None:
              IPs.add(destino)

       if rex.match(src) == None:
              IPs.add(src)
```

4.5. Análisis

4.5.1. Análisis estático de permisos

Para poder llevar a cabo el análisis estático de permisos lo primero que debemos hacer el crear una base de datos donde estén almacenados los permisos de multitud de aplicaciones, tanto aplicaciones maliciosas como comunes.

En nuestro caso disponemos de un listado de 158 aplicaciones que contienen malware a analizar obtenidas de la wiki de un alumno de la universidad de Indiana. A su vez se han descargado 89 aplicaciones de las más conocidas en Google Play carentes de cualquier tipo de código malicioso.

Crearemos dos tablas para almacenar los permisos de las aplicaciones diferenciando las que poseen malware de las comunes. De esta forma nos será más simple el análisis.

```
CREATE TABLE malignas(
HASH varchar(32) not null,
Permiso varchar(32) not null
);

CREATE TABLE benignas(
HASH varchar(32) not null,
```

```
Permiso varchar(32) not null
);
```

Situamos las aplicaciones en un directorio único para cada tipo de aplicación para que así nos sea más fácil automatizar la tarea de extracción de permisos. Para recorrer todas las aplicaciones nos ayudaremos del siguiente script.

```
#!/bin/bash
# -*- ENCODING: UTF-8 -*-
#!/bin/bash

FILES=/home/francisco/buenas/*
FILES2=/home/francisco/malas/*
for f in $FILES
do
echo "Obteniendo hash MD5"
hashapk=$(python obtenerhashdata.py $f)

dir=$(echo ${f%.*})
echo "Intentando descomprimir el fichero apk" $f
apktool d $f -f -o $dir

echo "Analizando fichero AndroidManifest.xml"
python addtodataset.py $dir/AndroidManifest.xml $hashapk 1

rm -R $dir
done

for f in $FILES2
do
echo "Obteniendo hash MD5"
hashapk=$(python obtenerhashdata.py $f)

dir=$(echo ${f%.*})
echo "Intentando descomprimir el fichero apk" $f
apktool d $f -f -o $dir

echo "Analizando fichero AndroidManifest.xml"
python addtodataset.py $dir/AndroidManifest.xml $hashapk 2

rm -R $dir
done
```

FILES indica la ruta de las aplicaciones que carecen de malware y FILES2 las que lo poseen. Obtenemos el HASH de cada aplicación mediante el fichero obtenerhashdata.py definido de la siguiente forma.

```
import hashlib
import sys

hash_md5 = hashlib.md5()
with open(sys.argv[1], "rb") as f:
for chunk in iter(lambda: f.read(4096), b""):
hash_md5.update(chunk)
```

```
c_hash=hash_md5.hexdigest()
print c_hash
```

Extraemos el fichero apk como en la obtención de información estática con apktool y llamamos al fichero addtodataset.py para parsear el fichero AndroidManifest. Se le debe pasar como argumentos el fichero AndroidManifest, el HASH de la aplicación y un valor numérico entre 1 y 2 que determina si la aplicación que le estamos pasando es común o malware. El código es el siguiente.

```
import sys
from xml.dom.minidom import parseString
import MySQLdb

conn = MySQLdb.connect()

informacion=''
with open(sys.argv[1],'r') as f:
        informacion = f.read()

hash_apk=sys.argv[2]
tipo==sys.argv[3]
dom = parseString(informacion)
c = conn.cursor()

if tipo==1:
        c.execute('SELECT * FROM benignas WHERE HASH=%s', (hash_apk,))
        data=c.fetchall()
        if len(data)==0:
                nodes = dom.getElementsByTagName('uses-permission')

                for node in nodes:
                        item= node.getAttribute('android:name')
                        item=item.replace('android.permission.','')
                        item=item.replace('
                        com.android.launcher.permission.','')
                        item=item.replace('
                        com.android.browser.permission.','')
                        item=item.replace('com.android.vending.','')
                        item=item.replace('
                        com.google.android.c2dm.permission.','')
                        item=item.replace('
                        com.google.android.providers.gsf.permission.','')

                        c.execute('SELECT * FROM malignas WHERE HASH=%s and
                        Permiso=%s',(hash_apk, item))
                        data=c.fetchall()
                        if len(data)==0:
                                print item
                                c.execute("INSERT INTO benignas (HASH, Permiso)
                                VALUES (%s, %s)",(hash_apk, item))
                                conn.commit()
else:
        c.execute('SELECT * FROM malignas WHERE HASH=%s', (hash_apk,))
        data=c.fetchall()
        if len(data)==0:
                nodes = dom.getElementsByTagName('uses-permission')

                for node in nodes:
```

```
                        item= node.getAttribute('android:name')
                        item=item.replace('android.permission.','')
                        item=item.replace('
                        com.android.launcher.permission.','')
                        item=item.replace('
                        com.android.browser.permission.','')
                        item=item.replace('com.android.vending.','')
                        item=item.replace('
                        com.google.android.c2dm.permission.','')
                        item=item.replace('
                        com.google.android.providers.gsf.permission.'
                        ,'')

                        c.execute('SELECT * FROM malignas WHERE HASH=%s and
                        Permiso=%s',
                        (hash_apk, item))
                        data=c.fetchall()
                        if len(data)==0:
                               print item
                               c.execute("INSERT INTO malignas (HASH, Permiso)
                               VALUES (%s, %s)",(hash_apk, item))
                               conn.commit()
conn.close()
```

El proceso es largo debido a la gran cantidad de aplicaciones de las que se extraen sus permisos. Cuando hayamos finalizado podremos acceder a MySQL y obtener toda la información realizada con los permisos de las aplicaciones.

Lo primero que haremos es obtener los 15 permisos más usados por aplicaciones malignas y por aplicaciones comunes. Esto lo obtendremos mediante las siguientes consultas SQL.

```
select Permiso, count(*)/(select count(distinct HASH) from malignas)
from malignas where Permiso in (select distinct Permiso from malignas)
group by Permiso  order by count(*) desc limit 15;
```

```
select Permiso, count(*)/(select count(distinct HASH) from benignas)
from benignas where Permiso in (select distinct Permiso from benignas)
group by Permiso  order by count(*) desc limit 15;
```

Las respectivas salidas de las consultas anteriores las podemos ver en las siguientes tablas.

Permiso	count(*)/(select count(distinct HASH)from malignas)
INTERNET	0.9557
READ_PHONE_STATE	0.8608
SEND_SMS	0.7152
WRITE_EXTERNAL_STORAGE	0.6329
READ_SMS	0.6076
ACCESS_NETWORK_STATE	0.5759
READ_CONTACTS	0.5759
RECEIVE_SMS	0.5380
CALL_PHONE	0.4557
SET_WALLPAPER	0.4557
WRITE_SMS	0.4114
VIBRATE	0.4114
WRITE_APN_SETTINGS	0.3228
ACCESS_COARSE_LOCATION	0.3165
READ_HISTORY_BOOKMARKS	0.3101

Tabla 4.2: Permisos más usados en aplicaciones malignas

Permiso	count(*)/(select count(distinct HASH)from benignas)
INTERNET	0.9663
ACCESS_NETWORK_STATE	0.8876
WRITE_EXTERNAL_STORAGE	0.7978
WAKE_LOCK	0.5730
ACCESS_WIFI_STATE	0.4944
VIBRATE	0.4607
RECEIVE	0.3933
READ_EXTERNAL_STORAGE	0.3708
ACCESS_FINE_LOCATION	0.3596
BILLING	0.3596
CAMERA	0.3483
READ_PHONE_STATE	0.3483
ACCESS_COARSE_LOCATION	0.3258
GET_ACCOUNTS	0.3034
RECEIVE_BOOT_COMPLETED	0.2584

Tabla 4.3: Permisos más usados en aplicaciones sin malware

A partir de esta información, calcularemos la diferencia entre los permisos más usados en las aplicaciones malignas menos la proporción de esos permisos en las aplicaciones que no poseen malware. Posiblemente muchos de estos permisos no aparezcan los 15 más usados en las aplicaciones sin malware pero obtendremos su resultado indicándo el permiso en la siguiente consulta.

```
select Permiso, count(*)/(select count(distinct HASH) from benignas) from
```

```
benignas where Permiso in ('NOMBRE_PAQUETE');
```

Haremos la diferencia de cada permiso y obtendremos los resultados de la siguiente tabla.

Permiso	Diferencia	Puntuación
INTERNET	-0.0106	No malware info
READ_PHONE_STATE	0.5125	Información media - 6
SEND_SMS	0.6590	Información muy alta - 8
WRITE_EXTERNAL_STORAGE	-0.1649	No malware info
READ_SMS	0.5627	Información alta - 7
ACCESS_NETWORK_STATE	-0.3117	No malware info
READ_CONTACTS	0.4074	Información media - 5
RECEIVE_SMS	0.4706	Información media - 6
CALL_PHONE	0.3883	Información media - 5
SET_WALLPAPER	0.3883	Información media - 5
WRITE_SMS	0.3889	Información media - 5
VIBRATE	-0.0493	No malware info
WRITE_APN_SETTINGS	0.3228	Baja información - 4
ACCESS_COARSE_LOCATION	-0.0093	No malware info
READ_HISTORY_BOOKMARKS	0.2652	Baja información - 4

Tabla 4.4: Puntuación de los permisos maliciosos según diferencia

Las puntuaciones de cada permiso aproximando la diferencia a la décima más cercana, multiplicándola por 10 y sumándole 1. Solo usaremos los permisos que tengan una puntuación de 3 o más ya que son los que poseen una diferencia significativa entre la cantidad existente en aplicaciones con malware y sin ellos.

$Puntuacion_{Permiso} = 1 + 10 * [Diferencia_{Permiso}]$

Existen otros permisos que también son determinantes a la hora de indicar que una aplicación es malware aunque no aparezcan en la tabla. Multitud de estudios determinan el uso malicioso de estos permisos[33]. Se ha realizado la siguiente puntuación en función de la presencia de estos permisos en aplicaciones con malware.

Permiso	Puntuación
INSTALL_PACKAGES	Información alta - 6
MOUNT_UNMOUNT_FILESYSTEMS	Información media - 4
READ_LOGS	Información media - 4
RECEIVE_WAP_PUSH	Información media - 5

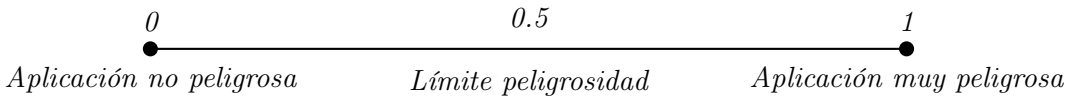
Tabla 4.5: Puntuación de otros permisos que implican malware

El resultado de la valoración de los permisos de una aplicación estarán en el intervalo [0,1], donde 0 es la ausencia de información que pueda hacernos pensar que la aplicación posee

malware y 1 la certeza de que contiene código malicioso. El cálculo lo realizaremos en la siguiente fórmula donde N es el número de permisos peligrosos, $P_{Maligno}$ es la puntuación de los permisos peligrosos y C es el número de permisos carentes de peligro.

$$Resultado = \frac{N * P_{Maligno}}{N * P_{Maligno} + C}$$

En el siguiente intervalo se puede entender a la perfección que cuanto mayor sea el resultado del análisis de los permisos existirá una mayor posibilidad de que contenga malware. Cuando el valor es 0.5 la aplicación está en un estado intermedio donde es difícil determinar si es peligrosa o no.



Lo primero que haremos para implementar el cálculo de esta valoración es crear una tabla donde guardaremos los resultados para así poder ofrecérselos al usuario.

```
CREATE TABLE puntuaciones(HASH varchar(32) primary key not null,
Puntuacion float not null);
```

La implementación en Python del cálculo y almacenamiento en MySQL del resultado la tenemos en el fichero calculaValoracionMalwarePermisos.py. En él código se obtienen todos los permisos de la aplicación, se comprueban si están en el listado de peligros y, de ser así, se usa su puntuación para realizar el cálculo de la valoración. Le debemos pasar como argumento el HASH de la aplicación.

```
import sys
from xml.dom.minidom import parseString
from pymongo import MongoClient
import MySQLdb

conn = MySQLdb.connect()
c = conn.cursor()
client = MongoClient()

db = client.aplicaciones

tofind=sys.argv[1]
numerador=0.0
denominador=0.0

roots=db.permisos_totales.find({"HASH": tofind})

for perm in roots:
	for i in perm['permisos']:
		c.execute('SELECT Puntuacion FROM valoraciones WHERE Permiso=%s',
		 (i,))
		data=c.fetchall()
		if len(data)!=0:
			for d in data:
				valor=d[0]
				numerador=numerador+float(valor)
				denominador=denominador+float(valor)
		else:
			denominador=denominador+1
```

```
if denominador !=0.0:
        result=float(numerador/denominador)
        c.execute("INSERT INTO puntuaciones (HASH, Puntuacion) VALUES (%s, %s)",
        (tofind, result))
        conn.commit()

else :
        c.execute("INSERT INTO puntuaciones (HASH, Puntuacion) VALUES (%s, %s)",
        (tofind, 0))
        conn.commit()

conn.close()
```

4.5.2. Análisis de información dinámica

En primer lugar hablaremos de la comprobación de las IPs en las blacklists. Posteriormente se detallará la forma de obtener una valoración numérica para determinar si la aplicación contiene malware mediante sus llamadas al sistema.

Con las IPs obtenidas de la aplicación ejecutándose bajo máquina virtual y que hemos almacenado en MongoDB nos cercionaremos de si las IPs obtenidas están en alguna blacklist o no. En el caso de que alguna IP esté en una blacklist comprobaremos también el país de origen. Dicho análisis no nos determina una puntuación numéria como puede ser el basado en permisos pero si nos puede dar una visión clara de hacia qué tipo de servidores intenta la aplicación conectarse. El código completo está en el Anexo D.

Del código mencionado tenemos la comprobación de IPs de la siguiente forma donde se realiza la petición DNS de la siguiente forma.

```
for ip in valor['IPs']:
for bl in bls:
try:
my_resolver = dns.resolver.Resolver()

query = '.'.join(reversed(str(ip).split(".")))
+ "." + bl
answers = my_resolver.query(query, "A")

mensaje_respuesta = my_resolver.query(query, "TXT")

if 'Dynamic' not in
str(mensaje_respuesta.response):

db.IPsMaliciosas.update({"HASH": hash_apk},
{'$push':{
"IPs" : {'IP' : ip , 'blacklist' : bl }}}, True)

except dns.resolver.NXDOMAIN:
pass
```

Para la obtención del país de la IP simplemente utilizamos una API[24] que pasándole una IP nos devuelve un JSON con el país del que proviene la IP.

```
url="http://ip-api.com/json/%s"%ip
response = urllib.urlopen(url)
data = json.loads(response.read())
```

```
pais= data['country']
```

Concluido el tema de las IPs el siguiente punto que trataremos es el de las llamadas al sistema.

Las llamadas al sistema las tenemos almacenadas en un fichero de texto como se explicó en la sección de obtención de información dinámica. Lo primero que tenemos que hacer con ellas es parsearlas y añadirlas a una base de datos. Para ello crearemos la siguiente tabla donde asociaremos a cada HASH cada tipo de llamada y el número de repeticiones de cada llamada.

```
CREATE TABLE syscalls(
HASH varchar(32) not null,
llamada varchar(20),
numero int default 1
);
```

Creamos el código que parsea el fichero con las llamadas. Le debemos pasar la ruta del fichero y el HASH de la aplicación. Recorrerá de forma secuencial todo el fichero contando el número de apariciones de cada llamada e introduciéndolas en la base de datos.

```
import re
import sys
import time
import MySQLdb

conn = MySQLdb.connect()
c = conn.cursor()


regex = re.compile("^[^\(,]*")
hash_apk=sys.argv[2]
with open(sys.argv[1],'r') as f:
       next(f)
       for line in f:
              result = regex.findall(line)
              if len(result[0].split()) <= 1:
                     c.execute('SELECT * FROM syscalls WHERE HASH=%s and llamada
                         =%s', (hash_apk,result[0]))
                     data=c.fetchall()
                     if len(data)==0:
                                   c.execute("INSERT INTO syscalls (HASH,
                                       llamada, numero) VALUES (%s, %s,1)",(
                                       hash_apk, result[0]))
                     else:
                            c.execute('UPDATE syscalls set numero=numero+1 WHERE
                                HASH=%s and llamada=%s', (hash_apk,result[0]))

conn.commit()
conn.close()
```

Para obtener una valoración basaremos en los siguiente permisos los cuales a priori, aparecen mayoritariamente en aplicaciones con malware y no en aplicaciones comunes.

- bind. Asocia un socket con un puerto.

- brk. Modifica el tamaño de la pila.
- fchown32. Cambia el propietario de un fichero o directorio.
- fdatasync. Vuelta los búferes de datos de un fichero a disco.
- fsync. Vuelta los búferes de datos de un fichero a disco.
- mkdir. Crea un directorio.
- msgget. Acceso a una cola de mensajes.
- rmdir. Elimina un directorio.
- select. Espera a que un número de descriptores de fichero cambien de estado.
- semget. Obtiene el identificador de un conjunto de semáforos
- semop. Realiza operaciones con un semáforo seleccionado.
- setsockopt. Cambia opciones del socket.
- statfs64. Devuelve información de un sistema de archivos montado.
- umask. Establece los permisos por defecto para los nuevos archivos y directorios creados por el proceso actual.

Le asignaremos una misma puntuación a todos en una tabla. La tabla sería de la siguiente forma.

```
CREATE TABLE valoraciones_sys(
llamada varchar(32) primary key not null,
Puntuacion int not null
);
```

La salida de la tabla asignándole una puntuación de 110 a cada llamada sería la de la Tabla 4.6.
El cálculo de la valoración lo haremos de una forma muy sencilla. Se multiplicará el número de llamadas peligrosas por su puntuación y se dividirá entre el número total de llamadas obtenidas.

$$Resultado = \frac{N_{LLMaligna} * P_{LLMaligna}}{N_{Total}}$$

Si el resultado supera 0,5 tiene grandes indicios de ser malware la aplicación. En este caso no disponemos de resultado máximo ya que se puede dar el caso de que cientos de llamadas al sistema de la aplicación estén categorizadas como peligrosas. Normalmente, una aplicación sin malware obtiene una puntuación inferior a 0.2.

Generalmente las aplicaciones con malware tienen poco desarrollo lo que hace que mediante la ejecución de sus actividades y la captura de llamadas al sistema, obtengamos una menor cantidad de llamadas que en una aplicación común. Esto ocurre debido a que la principal función de la aplicación es la de obtener un beneficio y no la de ofrecer un servicio al usuario. Por tanto, una aparición de una de estas llamadas peligrosas tendrá mucho más peso en el cálculo que en una aplicación con más funcionalidades.

Para la realización del cálculo de la valoración creamos una tabla donde almacenar los resultados obtenidos del cálculo y poder mostrar la información desde la interfaz web.

llamada	Puntuacion
bind	110
brk	110
fchown32	110
fdatasync	110
fsync	110
mkdir	110
msgget	110
rmdir	110
select	110
semget	110
semop	110
setsockopt	110
statfs64	110
umask	110

Tabla 4.6: Puntuación de llamadas al sistema que pueden implicar malware

```
CREATE TABLE puntuaciones_sys(
HASH varchar(32) primary key not null,
Puntuacion float not null
);
```

El código para calcular el resultado y añadirlo a la base de datos sería el siguiente. Solo debemos pasarle el HASH de la aplicación y si la aplicación dispone de llamadas al sistema almacenadas en la base de datos se realizará el cálculo.

```
import sys
import MySQLdb

conn = MySQLdb.connect()
c1 = conn.cursor()
hash_apk=sys.argv[1]
numerador=0.0
denominador=0.0

c.execute('SELECT * FROM syscalls WHERE HASH=%s', (hash_apk,))
data=c.fetchall()
if len(data)!=0:
    for p in data:
        c1.execute('SELECT * from valoraciones_sys where llamada=%s', (p[1],))
        data2=c1.fetchone()
        if data2 is not None:
            numerador=numerador+(p[2]*data2[1])

    c1.execute('SELECT sum(numero) FROM syscalls WHERE HASH=%s', (hash_apk,))
    data2=c1.fetchone()
    denominador= data2[0]
    result=float(numerador/denominador)
    c.execute("INSERT INTO puntuaciones_sys (HASH, Puntuacion) VALUES (%s, %s)",(
        hash_apk, result))
    conn.commit()
```

```
    conn.close()
```

Concluido esto los resultados podrán ser visibles desde la interfaz web para que el usuario valore los distinta información sobre la aplicación.

4.6. Interfaz web

Para un usuario con escasos conocimientos informáticos realizar cualquier gestión mediante su ordenador se le hace muy complicado, ya imposible si es usando la línea de órdenes. La idea es que cualquiera sea capaz de envía una aplicación a analizar y que pueda examinar los resultados sin demasiados conocimientos.

La mejor forma para que un usuario común pueda analizar una aplicación sin complicaciones es mediante un interfaz web. Una web donde mediante un formulario pueda seleccionar el fichero apk a analizar y donde pueda consultar los resultados.

Como se mencionó en el apartado de "Servicios previos"se usará Django sobre Apache para el servicio web. Django nos permite desarrollar la web en Python como si de una aplicación de escritorio se tratase y mostrar la información a través de plantillas HTML. Su funcionamiento y desarrollo es muy sencillo por lo que solo hablaremos de las partes más significativas que requieran un mayor conocimiento como la información a mostrar o el formulario para subir aplicaciones.

El diseño en HTML de la web es muy sencillo. Django templates[15] nos permite extender bloques a partir de una plantilla base. Usaremos una plantilla con una barra lateral a la izquierda y con una cabecera donde podamos ubicar un buscador para localizar las aplicaciones por HASH o nombre de paquete. En el anexo E se puede consultar el código HTML de la plantilla.

En el comiendo del código apreciamos como se cargan los ficheros estáticos donde se almacenan las imágenes y los CSS. Esto lo hacemos con {% load staticfiles %}. Cuando llamamos a la ruta del CSS también debemos indicar que es un fichero ubicado en static.

Lo siguiente que tenemos es la barra superior de búsqueda la cual simplemente es un formulario que pasa como parámetro el string de búsqueda y nos redirige a la página donde aparecen los resultados.

```
<div class="row">
       <form action="/buscar/" method="GET">
              <input class="flipkart-navbar-input col-xs-11" type="" placeholder
                  ="Busca HASH o package.name" name="busqueda">
              <button type="submit" class="flipkart-navbar-button col-xs-1">
              <svg width="15px" height="15px">
              </svg>
              </button>
       </form>
 </div>
```

Disponemos una barra lateral que nos sirve como menú para acceder a la ruta de subida de aplicaciones, al listado o al formulario de contacto.

Tel vez la parte más significativa es la parte donde se declara el bloque body "{% block body %}{% endblock %}". Aquí es donde irá el contenido principal y que lo mostraremos

mediante otros HTML extendiendo el bloque body de la plantilla base.

En la siguiente imagen podemos apreciar el aspecto visual de la plantilla base descrita. La parte central en blanco es el body que extenderemos con la información relevante de la web.

Figura 4.3: Plantilla HTML Base

Pasamos ahora a como mostramos los datos de una aplicación en la página web. Realmente es algo bastante sencillo donde simplemente debemos volcar los datos de MySQL y MongoDB en el HTML. Como en Django se desarrolla en Python la implementación será similar a una aplicación de escritorio.

La información que se envía a las plantillas HTML se realiza mediante vistas. En el fichero views.py están todas las vistas que definen el funcionamiento de la web. En el anexo F podemos ver la vista donde enviamos al HTML toda la información de la aplicación analizada.

Lo primero que hacemos es comprobar si el parámetro recibido es el HASH de una aplicación o el nombre del paquete. Si es el nombre del paquete devolvemos el listado de HASH que poseen ese paquete para que el usuario pueda acceder al deseado.

Si el HASH buscado existe, extraemos toda la información que disponemos en MySQL y MongoDB y las almacenamos en variables que enviaremos al HTML para que el usuario pueda ver toda esta información desde su navegador. En todas las consultas comprobamos si hay elementos y guardamos este contador en una variable para, si el valor de la variable es None no crear la tabla en el HTML y mostrar estructuras innecesarias al usuario.

Llamamos al HTML pasándole todos las variables con la información como podemos apreciar en este extracto del código anterior.

```
return render(request,'informacion.html', {'resultado':resultsList,
'permisos':res, 'num_apk_install':num_apk_install,'apk_install':apk_install,
'num_inject':num_inject,'injects':injects,
'num_root':num_root,'roots':roots, 'num_ips':num_ips, 's_ips': s_ips,
'num_sms':num_sms, 'smss':smss, 'num_dips':num_dips, 'paises':paises,
'maliciosas':maliciosas, 'eshash': eshash,
'num_pay': num_pay, 'payload': payload,'tiene_valoracion_permisos'
```

```
:tiene_valoracion_permisos,'puntuacion_permisos':puntuacion_permisos})
```

Ahora veremos el fichero informacion.html que es el encargado de mostrar estas variables al usuario de una forma estética y no como simples cadenas de datos. Dicho fichero es una extensión de la plantilla base. Podemos consultarlo en el anexo G.

El código del anexo es más extenso pero es innecesario mostrar como se recorre cada elemento ya que siempre es de la misma forma. Comprobamos si la variable de cantidad es superior a cero. Si es así se pinta la tabla y se recorre su respectiva variable que contiene los permisos, IPs o trozos de códigos peligrosos según corresponda. Se realiza de la misma forma en todos los casos y todas las tablas son desplegables gracias a bootstrap. Así en ningún caso tendremos tablas vacías que dañen la estética de la web.

En la misma página también mostramos dos gráficos con información de la aplicación si esta lo requiere. Los gráficos son creados con la biblioteca Highcharts y usados para mostrar el número de permisos que comprometan información personal y la localización de las IPs maliciosas. Su implementación es muy simple, solo deberemos llamar a los ficheros javascript de sus bibliotecas y añadir en el código la función prediseñada con los parámetros que deseamos mostrar en las gráficas. En nuestro caso las dos gráficas que mostraremos serían de la siguiente forma.

```
<script>
$(document).ready(function() {
  Highcharts.chart('contpermisos', {
    chart: {
      plotBackgroundColor: null,
      plotBorderWidth: null,
      plotShadow: false,
      type: 'pie'},
    title: {
      text: 'Numero de permisos peligrosos requeridos'},
    tooltip: {
      pointFormat: '{series.name}: <b>{point.percentage:.1f}%</b>'},
    plotOptions: {
      pie: {
        allowPointSelect: true,
        cursor: 'pointer',
        dataLabels: {
          enabled: false},
        showInLegend: true}},
    series: [{
      name: 'Brands',
      colorByPoint: true,
      data: [{
        name: 'Permisos peligrosos no requeridos',
        y: (24-{{permisos.0.permisos|length}})*100/24,
      }, {
        name: 'Permisos peligrosos requeridos',
        y: {{permisos.0.permisos|length}}*100/24,
        sliced: true,
        selected: true,
        color: 'red'
      }]
    }]
  });
});
```

```
</script>

<script>
$(document).ready(function() {
Highcharts.chart('contmalips', {
    chart: {
        type: 'pie',
        options3d: {
            enabled: true,
            alpha: 45}
    },
    title: {
        text: 'Numero de IPs maliciosas accedidas por Pais'},
    plotOptions: {
        pie: {
            innerSize: 100,
            depth: 45
        }},
    series: [{
        name: 'Numero de IPs',
        data: [
          {% load app_tags %}
          {% for dip in paises %}
            ['{{dip|mongo_id}}',{{dip.sum}}],
            {% endfor %}
        ]
    }]
});
});
</script>
```

al primer gráfico le pasamos el número de permisos que usa la aplicación y al segundo el nombre de cada país y el número de IPs por cada país.

El resultado final de la página con toda la información sería como en la Figura 4.4.

En la imagen apreciamos los datos principales, de la aplicación, sus permisos, los gráficos de permisos e IPs y las partes del código que pueden ser dañinas. En función de si la aplicación dispone de otras cosas peligrosas como llamadas al sistema, inyección de código en Dalvik o la puntuación basa en permisos por ejemplo, nos aparecerá más información. Solo se mostrará la información que se disponga en la base de datos pero en ningún caso aparecerán tablas vacías.

El último punto a describir de la interfaz web es el formulario de subida de las aplicaciones y su sincronización con la máquina que realiza la extracción de información y ejecuta las máquinas virtuales.

Lo primero que se necesita es la creación de un formulario en la web para la subida de ficheros. Dicho formulario[?] en Django requiere de la creación de una clase en el fichero forms.py donde se le indican de qué tipo son cada variable.

```
from django import forms
from .validators import validate_file_extension

class ProfileForm(forms.Form):
       picture = forms.FileField(validators=[validate_file_extension])
```

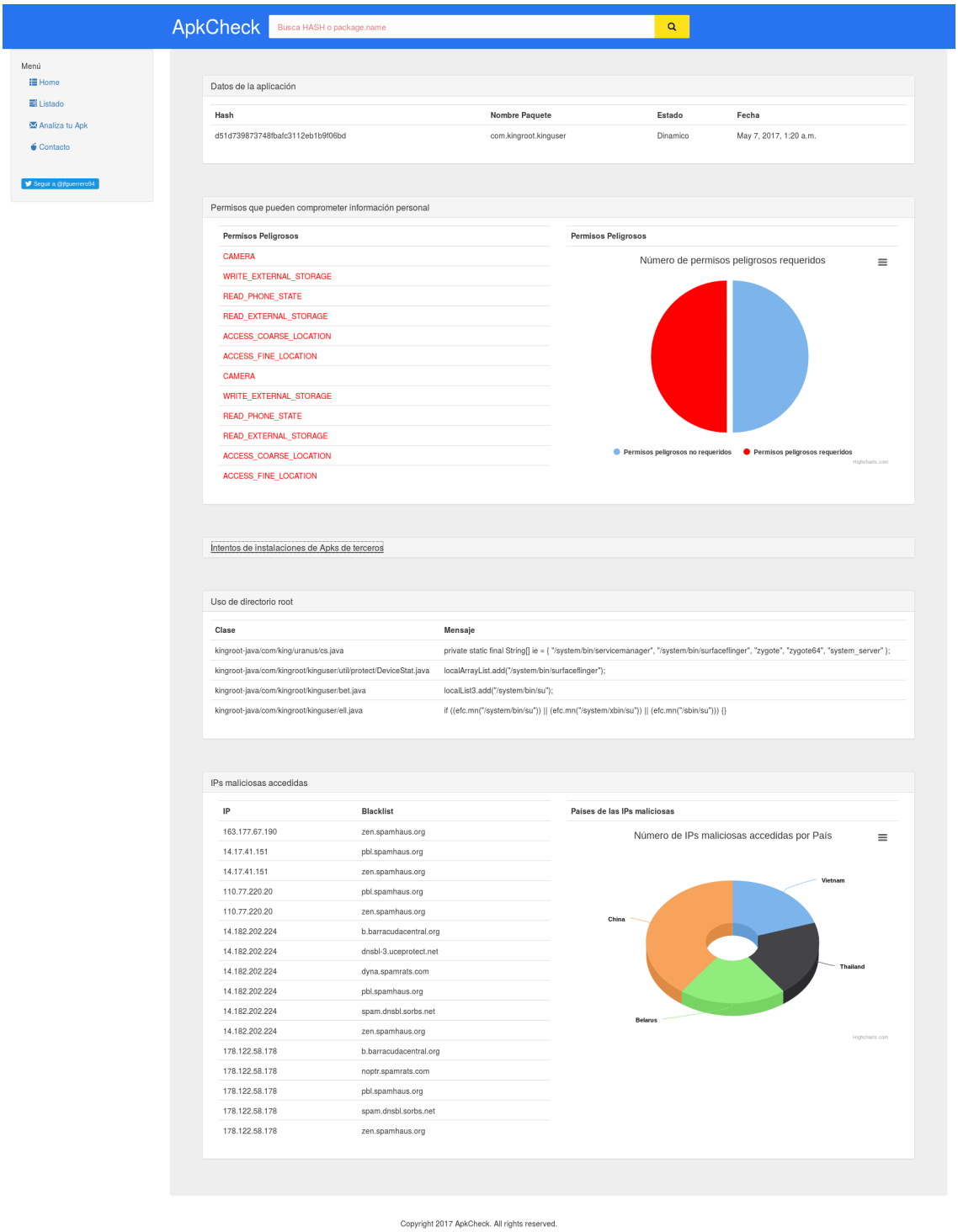


Figura 4.4: Ejemplo de información de aplicación en web

Le indicamos que el parámetro del formulario es de tipo FileField el cual nos permite guardar cualquier tipo de fichero de datos. Como solo queremos que se puedan subir realizaremos una comprobación mediante validate_file_extension. En dicha valoración nos cercioramos del tipo de fichero que se está subiendo y de no ser correcto indicamos un mensaje de error.

```
def validate_file_extension(value):
        import os
        from django.core.exceptions import ValidationError
        ext = os.path.splitext(value.name)[1] # [0] returns path+filename
        valid_extensions = ['.apk']
        if not ext.lower() in valid_extensions:
                raise ValidationError(u'Unsupported file extension.')
```

Ahora implementaremos la vista que almacene los ficheros subidos mediante el formulario. No tendría sentido que una aplicación que ya ha sido analizada la analicemos de nuevo. Para evitar esto obtenemos el HASH de la aplicación subida y comprobamos que no existe en la base de datos de aplicaciones analizadas. En el caso de existir eliminamos el fichero para evitar tener ficheros innecesarios almacenados en nuestro VPS. Si no se ha analizado la aplicación, movemos el fichero de ruta para usar inotifywait, servicio del que hablaremos posteriormente.

```
def SaveProfile(request):
        saved = False
        existe = False
        nano=''
        if request.method == "POST":
                MyProfileForm = ProfileForm(request.POST, request.FILES)

        if MyProfileForm.is_valid():
                profile = Profile()
                profile.picture = MyProfileForm.cleaned_data["picture"]
                profile.save()
                nano=profile.filename
                hash_apk=hashlib.md5(open(os.path.join(settings.MEDIA_ROOT, nano),
                     'rb').read()).hexdigest()
                db = MySQLdb.connect()
                cursor=db.cursor()
                saved = True
                cursor.execute("select * from aplicaciones where HASH=%s",[
                    hash_apk])
                data=cursor.fetchall()

                if len(data)!=0:
                        os.remove(os.path.join(settings.MEDIA_ROOT,nano))
                        existe= True
                else:
                        nano=os.path.basename(nano)
                        os.rename(settings.MEDIA_ROOT+'/pictures/'+nano,settings.
                            MEDIA_ROOT+'/files/'+nano)
        else:
                MyProfileForm = Profileform()

        return render(request, 'saved.html', locals())
```

La presentación el formulario en la web es muy simple. Extenderemos la plantilla base introduciéndole sencillamente el botón para seleccionar el fichero y el botón de enviar el formulario. En un principio se realizaron las pruebas para imágenes y cuando se comprobó

que funcionaba a la perfección se adaptó para ficheros apk.

```
{% extends "base.html" %}
{% block body %}
<html>
   <body>
        <center>
      <form name = "form" class="formup" enctype = "multipart/form-data"
         action = "{% url 'saved' %}" method = "POST" >{% csrf_token %}
         <div style = "max-width:470px;">
            <center>
               <input type = "file" style = "margin-left:20%;"
                  placeholder = "file" name = "file" />
            </center>
         </div>
         <div style = "max-width:470px;">
            <center>
               <button style = "border:0px;background-color:#4285F4; margin-top
                   :8%;
                  height:35px; width:80%; margin-left:19%;" type = "submit" value
                      = "Login" >
                  <strong>Analizar Apk</strong>
               </button>
            </center>
         </div>
      </form>
      </center>
   </body>
</html>
{% endblock %}
```

Visualmente el formulario de envío de aplicaciones quedaría de la siguiente forma. Es lo más simple posible y compatible con dipositivos móviles.

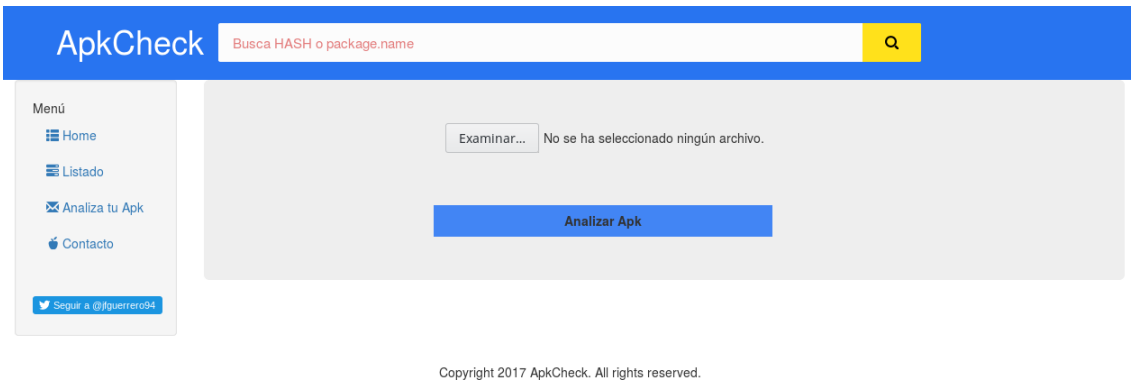


Figura 4.5: Formulario de envío de aplicaciones

Hemos mencionado hace un par de párrafos inotifywait. Éste nos será esencial para la transferencia de ficheros desde el VPS a el servidor que evalúa las aplicaciones. Inotifywait nos permite saber cuando ocurre alguna novedad en un directorio determinado. Aprovechando esto realizaremos un script que cuando se suba un fichero y se mueva al directorio donde se almacenan las aplicaciones de forma automática se enviará desde el VPS al servidor donde se analizan las aplicaciones. Para saber cuando ejecutar el análisis se hará algo

similar mediante un script que detecte nuevos ficheros y lance la ejecución cuando sea así.

En el VPS para enviar las aplicaciones usaremos el siguiente script que envía los ficheros mediante scp.

```
#!/bin/bash

inotifywait -m -r --timefmt '%d/%m/%y %H:%M' --format '%T %w %f' -e move /var/
    www/web_tfg/media/files | while read date time dir file; do
        FILECHANGE=${dir}${file}
        FL=${file}
        echo "$FILECHANGE"
        scp $FILECHANGE francisco@sqlmaster.dynu.net:/home/francisco/Escritorio/
            apk
        rm $FILECHANGE
done
```

Para automatizar esta tarea lo mejor es la creación de un demonio que en todo momento envíe las aplicaciones nuevas a la otra máquina sin tener que estar lanzando el script.

Como queremos que cuando la máquina reciba una aplicación automáticamente se ejecute la obtención de información y el análisis crearemos también este script en la máquina donde emularemos Android.

```
#!/bin/bash
inotifywait -m -r --timefmt '%d/%m/%y %H:%M' --format '%T %w %f' -e create /
    home/francisco/Escritorio/apk | while read date time dir file; do
        FILECHANGE=${dir}${file}
        FL=${file}
        echo "$FL"
        echo "$FILECHANGE"
        sleep 5
        mv $FILECHANGE /home/francisco/Escritorio/TFG
        cd /home/francisco/Escritorio/TFG
        ./procesarapk.sh $FL
done
```

De esta forma en todo momento la máquina estará pendiente de si se recibe una nueva aplicación a analizar.

5. Experimentos

5.1. Pruebas del análisis estático mediante permisos

Vamos a comprobar que nuestro análisis estático basado en permisos es realmente un análisis fiable. Para ello vamos a analizar 60 aplicaciones malignas y 60 aplicaciones que no poseen malware y con los resultados calcular el porcentaje de falsos positivos y falsos negativos.

Aplicaciones	Cantidad	Positivos	Negativos	P.F.P.	P.F.N.
Comunes	60	7	53	11,66 %	-
Malignas	60	58	2	-	3,33 %

Tabla 5.1: Pruebas análisis basado en permisos

Analizando los resultados obtenidos se puede decir que el análisis ha sido un éxito. La valoración de cada aplicación está entre el intervalo 0 y 1, siendo el límite que define una aplicación sin malware a una con ello en 0,5.

En el análisis de aplicaciones que carecen de malware solo se han producido 7 falsos positivos y con una valoración que no llegaba a 0,6 ninguna de las aplicaciones que nos ha dado positivo. Con solo un 11,66 % de porcentaje de falso positivo se puede determinar que las aplicaciones sin malware han obtenido unos resultados bastante fidedignos a la realidad.

En el caso de las aplicaciones que contienen malware el resultado ha sido aún mejor. Solo 2 de las 60 aplicaciones analizadas con malware han dado un resultado que implicaría que no poseen malware. Esto es solo un porcentaje de falso negativo del 3,33 %. Además la mayor parte de las aplicaciones poseen una valoración de 0,8 lo que nos da un valoración con una gran rotundidad indicándonos que la aplicación posee en su interior código maligno.

5.2. Pruebas del análisis dinámico basado en llamadas al sistema

Como en el análisis anterior, vamos a hacer un leve estudio para hacernos una idea del porcentaje de éxito y error que nos proporciona el análisis dinámico basado en llamadas al sistema o syscalls. Usaremos en este caso sólo 35 aplicaciones de cada tipo ya que muchas de las que poseen malware presentan problemas a la hora de obtener sus llamadas al sistema. Esto se debe a alteraciones de sus desarrolladores para evitar instalaciones por ADB y que se pueda monitorizar su funcionamiento. De todas formas tan sólo se producen errores debido a esto en un 15 % de los casos.

Analizando las llamadas al sistema el resultado es el siguiente.

Aplicaciones	Cantidad	Positivos	Negativos	P.F.P.	P.F.N.
Comunes	35	0	35	0 %	-
Malignas	35	29	6	-	17,17 %

Tabla 5.2: Pruebas análisis basado en llamadas al sistema

En las aplicaciones que carecen de malware no se ha producido ningún positivo y la amplia mayoría ha quedado bastante lejos de 0,5, resultado que determina el corte que nos hace

sospechar de la aplicación. Las puntuaciones tan bajas de este tipo de aplicaciones en gran parte se deben a la ausencia o pocas apariciones de las llamadas al sistema que hemos listado en la sección de análisis dinámico y del gran número de llamadas al sistema que obtenemos mediante strace. Estas aplicaciones tienen un mayor número de actividades por norma general lo que hace que el número total de llamadas aumente y haga disminuir el resultado.

En el caso de las aplicaciones con malware de 35 aplicaciones analizadas se han obtenido 29 positivos y 6 negativos. Esto nos da un 17,17 % de falsos negativos. Esta aplicaciones disponen de menos actividades lo que nos hace obtener menos llamadas al sistema ya que tienden a ser aplicaciones que solo buscan colar el código malicioso al usuario por lo que carecen de una gran elaboración. De los 6 falsos negativos 2 de ellos se han quedado con una valoración de 0,4 lo que nos hace pensar que pese a ser la aplicación más discreta, tiene leve indicios de contener malware también.

6. Conclusiones

Concluido el desarrollo de este proyecto y tras haber detallado la propuesta realizada ante el peligro que supone a día de hoy el malware en sistemas Android es el momento de valorar los resultados obtenidos.

Se ha conseguido cerca de un 90 % de éxito analizando aplicaciones que poseen malware en su código. Esto nos indica que se han cumplido los objetivos marcados antes de la realización del proyecto. Es cierto en en muchos casos es difícil de determinar la naturaleza de la aplicación pero la idea es proporcionar toda la información posible al usuario para que, aunque se dude de si es peligrosa o no, tenga un conocimiento de los peligros que puede acarrear su uso.

El desarrollo del trabajo ha merecido la pena debido al aprendizaje adquirido sobre el tema y sobre los servicios externos que se necesitan para su realización. Como contrapartida, es cierto que el proyecto necesita perfeccionamiento en algunos aspectos para optimizar su funcionamiento y prevenir posibles errores así como implementar mejoras que hagan que la herramienta desarrollada puede tener fines comerciales.

7. Ideas futuras

Existen diversas mejoras que se podrían aplicar en futuras versiones como puede ser el procesamiento de datos en la nube. De esta forma conseguiríamos máxima disponibilidad y recursos pese a requerir un mayor coste de mantenimiento.

Otro aspecto a mejorar es el tema relacionado con las llamadas al sistema. Tal vez el uso de patrones de comportamiento los pueda proporcionar un mayor porcentaje de éxito a la hora de determinar si una aplicación es malware.

Finalmente como última idea sería el desarrollo de una aplicación que, aunque requiera permisos root, pueda obtener los datos de la aplicación desde el dispositivo del usuario y transmitírnoslos a una plataforma que los procesase y remitiese los resultados. Esto nos evitaría consumir tantos recursos y además ofrecer al usuario una mayor sencillez y unos resultados más fidedignos que los que se pueden ofrecer simulando el uso de una aplicación de forma automática.

Referencias

[1] Amazon. Genymotion on-Demand : Android 6.0 (marshmallow). https://aws.amazon.com/marketplace/pp/B01MREUQPU/, 2017. [Online; consultado 2-Abril-2017].

[2] Android. Android Debug Bridge. https://developer.android.com/studio/command-line/adb.html, 2014. [Online; consultado 23-Marzo-2017].

[3] Android. Funciones de Android Studio. https://developer.android.com/studio/features.html, 2014. [Online; consultado 19-Marzo-2017].

[4] Android. Permisos del sistema. https://developer.android.com/guide/topics/security/permissions.html?hl=es, 2015. [Online; consultado 30-Abril-2017].

[5] Android. Manifiesto de la app. https://developer.android.com/guide/topics/manifest/manifest-intro.html?hl=es, 2016. [Online; consultado 30-Abril-2017].

[6] Android. APK Structure. https://developer.android.com/studio/projects/index.html, 2017. [Online; consultado 30-Abril-2017].

[7] Android. ART and Dalvik. https://source.android.com/devices/tech/dalvik/, 2017. [Online; consultado 25-Abril-2017].

[8] Android. Debugging ART Garbage Collection. https://source.android.com/devices/tech/dalvik/gc-debug, 2017. [Online; consultado 25-Abril-2017].

[9] Android. Using Strace. https://source.android.com/devices/tech/debug/strace, 2017. [Online; consultado 27-Abril-2017].

[10] AndroidWorld.it. Genymotion ARM Translation. https://www.androidfilehost.com/?fid=23252070760974384, 2013. [Online; consultado 2-Abril-2017].

[11] Apktool. A tool for reverse engineering 3rd party, closed, binary Android apps. https://ibotpeaches.github.io/Apktool/, 2010. [Online; consultado 19-Abril-2017].

[12] Arubacloud. Cloud VPS - Hosting VPS . https://www.arubacloud.es/vps/tipos-virtual-private-server.aspx, 2011. [Online; consultado 7-Abril-2017].

[13] Sam Bowne. Decompiling and Trojaning an Android App with Smali Code. https://samsclass.info/128/proj/smali-trojan.htm, 2015. [Online; consultado 5-Mayo-2017].

[14] Certsuperior. Que es un certificado SSL. https://www.certsuperior.com/QueesunCertificadoSSL.aspx, 2011. [Online; consultado 12-Abril-2017].

[15] Djangoproject. Templates. https://docs.djangoproject.com/en/1.11/ref/templates/, 2010. [Online; consultado 15-Mayo-2017].

[16] Djnagoproject. Django overview. https://www.djangoproject.com/start/overview/, 2003. [Online; consultado 8-Abril-2017].

[17] Djnagoproject. How to use Django with Apache and mod_wsgi. https://docs.djangoproject.com/en/1.11/howto/deployment/wsgi/modwsgi/, 2008. [Online; consultado 9-Abril-2017].

[18] Ms.Prajakta D.Sawle and Prof. A. B. Gadicha. Analysis of malware detection techniques in android. *International Journal of Computer Science and Mobile Computing*, 3:176–182, March 2014.

[19] Vicente Aguilera Díaz. Android reverse engineering: understanding third party applications. http://vicenteaguileradiaz.com/pdf/OWASP_EU_Tour_2013-Android_Reverse_Engineering_Understanding_Third_Party_Applications.pdf, 2013. [Online; consultado 2-Mayo-2017].

[20] Let's Encrypt. Getting Started - Let's Encrypt - Free SSL/TLS Certificates. https://letsencrypt.org/getting-started/, 2015. [Online; consultado 12-Abril-2017].

[21] Genymotion. Features - Genymotion Android Emulator. https://www.genymotion.com/desktop/#features, 2016. [Online; consultado 1-Abril-2017].

[22] You Joung Ham, Daeyeol Moon, Hyung-Woo Lee, Jae Deok Lim, and Jeong Nyeo Kim. Android mobile application system call event pattern analysis for determination of malicious attack. *International Journal of Security and Its Applications*, 8:231–246, January 2014.

[23] Wenjun Hu. Dynamic Analysis of Android Applications. https://pacsec.jp/psj14/PSJ2014_Wenjun_Hey-%20We%20Catch%20You%20-%20Dynamic%20Analysis%20of%20Android%20Applications.pdf, 2014. [Online; consultado 1-Mayo-2017].

[24] IP-API. IP Geolocation API. http://ip-api.com/docs/, 2015. [Online; consultado 22-Mayo-2017].

[25] Kaspersky. Mobile Malware Evolution 2016. https://securelist.com/files/2017/02/Mobile_report_2016.pdf, 2017. [Online; consultado 21-Marzo-2017].

[26] Mailchimp. Cómo Funcionan las Listas Negras (Blacklists). https://samsclass.info/128/proj/smali-trojan.htm, 2016. [Online; consultado 7-Mayo-2017].

[27] Sapna Malik and Kiran Khatter. System Call Analysis of Android Malware Families. https://www.researchgate.net/publication/314086429_System_Call_Analysis_of_Android_Malware_Families, 2016. [Online; consultado 30-Mayo-2017].

[28] MongoDB. MongoDB: Reinventando la gestión de datos. https://www.mongodb.com/es, 2010. [Online; consultado 14-Abril-2017].

[29] MongoDB. Master Slave Replication. https://docs.mongodb.com/manual/core/master-slave/, 2014. [Online; consultado 16-Abril-2017].

[30] Bob Pan. Tools to work with android .dex and java .class files . https://github.com/pxb1988/dex2jar, 2013. [Online; consultado 20-Abril-2017].

[31] Nguyen Viet Duc Pham Thanh Giang and Pham Minh Vi. Permission analysis for android malware detection. In *The proceedings of the 7th vast - aist workshop "Research colaboration: Review and perspective*, pages 207–215, Hanoi, Vietnam, November 2015.

[32] Ex Ratione. A Simple Setup and Installation Script for Let's Encrypt SSL Certificates. https://www.exratione.com/2016/06/a-simple-setup-and-installation-script-for-lets-encrypt-ssl-certificates/, 2016. [Online; consultado 12-Abril-2017].

[33] Ryo Sato, Daiki Chiba, and Shigeki Got. Detecting android malware by analyzing manifest files. *Proceedings of the Asia-Pacific Advanced Network*, 36:23–31, November 2013.

[34] Suzanna Schmeelk. Static Analysis Techniques used in Android application Security Analysis. http://www.cs.columbia.edu/~aho/cs6998/Lectures/14-12-01_Schmeelk_AndroidSecurity.pptx, 2014. [Online; consultado 1-Mayo-2017].

[35] Etel Sverdlov. How To Set Up Master Slave Replication in MySQL. https://www.digitalocean.com/community/tutorials/how-to-set-up-master-slave-replication-in-mysql, 2012. [Online; consultado 15-Abril-2017].

[36] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Lorenzo Cavallaro. The evolution of android malware and android analysis techniques. *ACM Computing Surveys*, 4:1–41, January 2017.

[37] TechTarget. What is MD5? http://searchsecurity.techtarget.com/definition/MD5, 2017. [Online; consultado 19-Abril-2017].

[38] VirtualBox. Network Tips. https://www.virtualbox.org/wiki/Network_tips, 2014. [Online; consultado 25-Abril-2017].

[39] Virtualenv. Virtualenv documentation. https://virtualenv.pypa.io/en/stable/, 2007. [Online; consultado 8-Abril-2017].

Anexos

Anexos

A. Configuración Apache y Django

```
<VirtualHost apkcheck.com:80>

ServerName www.apkcheck.com
ServerAlias apkcheck.com *.apkcheck.com
WSGIDaemonProcess web_tfg python-path=/var/www/web_tfg:/var/www/web_tfg
/env_tfg/lib/python2.7/site-packages
WSGIProcessGroup web_tfg
WSGIScriptAlias / /var/www/web_tfg/web_tfg/wsgi.py

Alias /media/ /var/www/web_tfg/media/
Alias /static/ /var/www/web_tfg/static/

<Directory /var/www/web_tfg/static>
Require all granted
</Directory>

<Directory /var/www/web_tfg/media>
Order deny,allow
Allow from all
Require all granted
</Directory>
ServerAdmin webmaster@localhost
DocumentRoot /var/www/html

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

RewriteEngine on
RewriteCond %{SERVER_NAME} =*.apkcheck.com [OR]
RewriteCond %{SERVER_NAME} =www.apkcheck.com [OR]
RewriteCond %{SERVER_NAME} =apkcheck.com
RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
</VirtualHost>
```

B. Parser AndroidManifest

```
import sys
from xml.dom.minidom import parseString
import MySQLdb
from pymongo import MongoClient
client = MongoClient()

conn = MySQLdb.connect()
db = client.aplicaciones

lista_permisos = ['READ_CALENDAR', 'WRITE_CALENDAR', 'CAMERA',
'READ_CONTACTS', 'WRITE_CONTACTS', 'GET_ACCOUNTS' , 'ACCESS_FINE_LOCATION',
'ACCESS_COARSE_LOCATION' ,'RECORD_AUDIO', 'READ_PHONE_STATE', 'CALL_PHONE',
```

```
'READ_CALL_LOG', 'WRITE_CALL_LOG', 'ADD_VOICEMAIL','USE_SIP',
'PROCESS_OUTGOING_CALLS', 'BODY_SENSORS','SEND_SMS' ,'RECEIVE_SMS' ,
'READ_SMS', 'RECEIVE_WAP_PUSH' , 'RECEIVE_MMS' , 'READ_EXTERNAL_STORAGE',
'WRITE_EXTERNAL_STORAGE'];
informacion =''

with open(sys.argv[1],'r') as f:
informacion = f.read()
dom = parseString(informacion)

package_name = dom.getElementsByTagName('manifest')[0].getAttribute('package')
target_sdk = dom.getElementsByTagName('manifest')[0].getAttribute(
'platformBuildVersionCode')
hash_apk=sys.argv[2]
c = conn.cursor()
c.execute("SELECT * FROM aplicaciones WHERE HASH=%s and estado='Creado'",
(hash_apk,))
data=c.fetchall()
if len(data)!=0:

        if db.permisos.find({'HASH' : hash_apk}).count() == 0:
                result = db.permisos.insert_one(
                        {
                        "HASH": hash_apk,
                        "permisos": []
                        }
                )

        nodes = dom.getElementsByTagName('uses-permission')

        for node in nodes:
                item= node.getAttribute('android:name')
                item=item.replace('android.permission.','')

        if item in lista_permisos:
                db.permisos.update({"HASH": hash_apk}, {'$push':{ "permisos" :
                    item}},
                True)

        if db.actividades.find({'HASH' : hash_apk}).count() == 0:
                result = db.actividades.insert_one (
                        {
                        "HASH": hash_apk,
                        "actividades": []
                        }
                )

        nodes = dom.getElementsByTagName('activity')
        for node in nodes:
                item= node.getAttribute('android:name')
                db.actividades.update({"HASH": hash_apk}, {'$push':{ "actividades"
                     : item}},
                True)

        if db.acciones.find({'HASH' : hash_apk}).count() == 0:
                result = db.acciones.insert_one(
                        {
                        "HASH": hash_apk,
```

```
                "acciones": []
                }
        )

    nodes = dom.getElementsByTagName('action')
        for node in nodes:
                item= node.getAttribute('android:name')
                if(item.startswith("android")) and db.acciones.find({'HASH'
                    : hash_apk,
                "acciones" : item}).count() == 0:
                    db.acciones.update({"HASH": hash_apk}, {'$push':{ "
                        acciones" : item}}, True)

    c.execute("UPDATE aplicaciones SET nombre_paquete=%s, version_android=%s,
    estado='A_Manifest' WHERE HASH = %s and estado = 'Creado'",(package_name,
    target_sdk, hash_apk))
    conn.commit()
    conn.close()
```

C. Obtención IPs de fichero .cap

```
import dpkt
from dpkt.ip import IP
from dpkt.ethernet import Ethernet
from dpkt.arp import ARP
import socket
import re
import sys
from sets import Set
import MySQLdb
from pymongo import MongoClient

client = MongoClient()
conn = MySQLdb.connect()
c = conn.cursor()
db = client.aplicaciones
hash_apk=sys.argv[2]

c.execute("SELECT * FROM aplicaciones WHERE HASH=%s and
estado='Estatico'", (hash_apk,))
data=c.fetchall()

if len(data)!=0:

    filename=sys.argv[1]

    f = open(filename)
    pcap = dpkt.pcap.Reader(f)
    IPs = Set([])

    rex=re.compile("/(^127\.)|(^192\.168\.)|(^10\.)|(^0\.)
    |(^172\.1[6-9]\.)|(^172\.2[0-9]\.)|(^172\.3[0-1]\.)/")
    cont=1;
    for ts, buf in pcap:
        if cont > 1:
            eth=dpkt.ethernet.Ethernet(buf)
            if eth.type!=dpkt.ethernet.ETH_TYPE_IP:
                continue
```

```
                        ip=eth.data

                        destino = socket.inet_ntoa(ip.dst)
                        src=dst_ip_addr_str=socket.inet_ntoa(ip.src)

                        if rex.match(destino) == None:
                                IPs.add(destino)

                        if rex.match(src) == None:
                                IPs.add(src)

                cont=cont+1
        f.close()

        if db.IPs.find({'HASH' : hash_apk}).count() == 0:
        result = db.IPs.insert_one(
                {
                "HASH": hash_apk,
                "IPs": []
                }
        )

        for value in IPs:
                db.IPs.update({"HASH": hash_apk}, {'$push':
                { "IPs" : value}}, True)

        c.execute("UPDATE aplicaciones SET estado='Get_IPs' WHERE
        HASH = %s and estado = 'Estatico'",(hash_apk,))
        conn.commit()
        conn.close()
```

D. Comprobación de IPs malignas

```
from pymongo import MongoClient
import sys
import dns.resolver
import urllib, json
import MySQLdb

conn = MySQLdb.connect()
c = conn.cursor()
client = MongoClient()
db = client.aplicaciones

hash_apk=sys.argv[1]

c.execute("SELECT * FROM aplicaciones WHERE HASH=%s and
estado='Get_IPs'", (hash_apk,))

data=c.fetchall()

if len(data)!=0:

        bls = ["b.barracudacentral.org", "bl.spamcannibal.org",
        "bl.spamcop.net","blacklist.woody.ch", "cbl.abuseat.org",
         "cdl.anti-spam.org.cn",
        "combined.abuse.ch", "combined.rbl.msrbl.net"]
```

```
busqueda=db.IPs.find({'HASH' : hash_apk})

if db.IPsMaliciosas.find({'HASH' : hash_apk}).count() == 0:
       result = db.IPsMaliciosas.insert_one(
       {"HASH": hash_apk,"IPs": []})


if db.IPsPaises.find({'HASH' : hash_apk}).count() == 0:
       result = db.IPsPaises.insert_one({"HASH": hash_apk,
       "IPs": []})

for valor in busqueda:
       for ip in valor['IPs']:
              for bl in bls:
              try:
              my_resolver = dns.resolver.Resolver()

              query = '.'.join(reversed(str(ip).split(".")))
               + "." + bl

              answers = my_resolver.query(query, "A")

              mensaje_respuesta = my_resolver.query(query, "TXT")

              if 'Dynamic' not in
               str(mensaje_respuesta.response):

              db.IPsMaliciosas.update({"HASH": hash_apk},
               {'$push':{
              "IPs" : {'IP' : ip , 'blacklist' : bl }}}, True)

              url="http://ip-api.com/json/%s"%ip
              response = urllib.urlopen(url)
              data = json.loads(response.read())

              pais= data['country']

              db.IPsPaises.update(
              {'HASH': hash_apk},
       {'$addToSet': { 'IPs': {'IP' : ip , 'Pais' : pais } } }
              );

              except dns.resolver.NXDOMAIN:

              pass
              except dns.resolver.Timeout:
              print "timeout, sigo"

              except dns.resolver.NoNameservers:
              print "No Internet"

c.execute("UPDATE aplicaciones SET estado='Bad_IPs' WHERE
HASH = %s and estado = 'Get_IPs'",(hash_apk,))
conn.commit()


conn.close()
```

E. Plantilla HTML base Django

```
{% load staticfiles %}
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org
    /TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
<!-- Bootstrap -->

<!-- CSS de Bootstrap -->
 <link href="{% static 'css/bootstrap.min.css' %}"
    rel="stylesheet">
<link href="{% static 'css/propio.css' %}"
      rel="stylesheet">
<link href="{% static 'css/barra.css' %}" rel="stylesheet">
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<link rel="shortcut icon" type="image/png" href="{% static '/favicon.ico' %}"/>

<title>ApkCheck</title>


</head>

<body>

        <script type='text/javascript'>
        function openNav() {
    document.getElementById("mySidenav").style.width = "70%";
    // document.getElementById("flipkart-navbar").style.width = "50%";
    document.body.style.backgroundColor = "rgba(0,0,0,0.4)";
}

function closeNav() {
    document.getElementById("mySidenav").style.width = "0";
    document.body.style.backgroundColor = "rgba(0,0,0,0)";
}
        </script>

        <div id="flipkart-navbar">
            <div class="container">
                <div class="row row1">
                    <ul class="largenav pull-right">

                        </li>
                    </ul>
                </div>
                <div class="row row2">
            <div class="col-sm-2">
                <h2 style="margin:0px;"><span class="smallnav menu" onclick="
                    openNav()">ApkCheck</span></h2>
                <h1 style="margin:0px;"><span class="largenav">ApkCheck</span></h1
                    >
            </div>
            <div class="flipkart-navbar-search smallsearch col-sm-8 col-xs-11">
            <div class="row">
                                        <form action="/buscar/" method="GET">
```

```
      <input class="flipkart-navbar-input col-xs-11" type="" placeholder="
          Busca HASH o package.name" name="busqueda">
      <button type="submit" class="flipkart-navbar-button col-xs-1">
          <svg width="15px" height="15px">
          </svg>
      </button>
                                      </form>
    </div>
    </div>
           <div class="cart largenav col-sm-2">

           </div>
       </div>
    </div>
</div>
<div id="mySidenav" class="sidenav">
    <div class="container" style="background-color: #2874f0; padding-top:
        10px;">
        <span class="sidenav-heading">Home</span>
        <a href="javascript:void(0)" class="closebtn" onclick="closeNav()"
            >x</a>
    </div>

</div>

<div class="container-fluid">
               <div class="row row-offcanvas row-offcanvas-left">
                        <div class="row-offcanvas row-offcanvas-right">

        <div class="col-xs-6 col-sm-2 sidebar-offcanvas" id="sidebarLeft"
            role="navigation">

               <div class="well sidebar-nav">
                      <ul class="nav">
                                      <li>Menu</li>


                                      <li><a href={% url 'index' %}><span
                                          class="glyphicon glyphicon-th-list
                                          "></span> Home</a></li>

                                      <li><a href={% url 'listado' %}><span
                                           class="glyphicon glyphicon-tasks"
                                          ></span> Listado</a></li>
                                      <li><a href={% url 'upload' %}><span
                                          class="glyphicon glyphicon-
                                          envelope"></span> Analiza tu Apk</
                                          a></li>
                                      <li><a href={% url 'contacto' %}><
                                          span class="glyphicon glyphicon-
                                          apple"></span> Contacto</a></li>
                                      </br>
                      </ul>
               </div>
        </div>

        <div class="col-xs-12 col-sm-10">
        <div class="jumbotron container-fluid">
```

```
                        {% block body %}{% endblock %}
                    </div>
                </div>



         </div>
         </div>
        </div>
        <div id="footer">
                <center> <span>Copyright 2017 ApkCheck. All rights reserved.</span
                    ></center>
        </div>
    <script src="{% static 'js/bootstrap.min.js' %}"></script>
                <script src="{% static 'js/ts.js' %}"></script>
</body>
</html>
```

F. Vista Django obtención de información

```
@csrf_exempt
def buscar(request):
        eshash=1
        tofind=request.GET['busqueda']
        dba = MySQLdb.connect()
        cursor = dba.cursor()
        cursor.execute("select * from aplicaciones where nombre_paquete=%s",[
            tofind])
        resultado=cursor.fetchall()
        if cursor.rowcount!=0:
                eshash=0
                x = cursor.description
                hashlist = []
                for r in resultado:
                    i = 0
                    d = {}
                    while i < len(x):
                        d[x[i][0]] = r[i]
                        i = i+1
                    hashlist.append(d)
                return render(request,'informacion.html', {'eshash':eshash, '
                    hashlist':hashlist})
        else:
                cursor.execute("select * from aplicaciones where HASH=%s",[tofind
                    ])
                resultado=cursor.fetchall()
                if cursor.rowcount!=0:
                        x = cursor.description
                        resultsList = []
                        for r in resultado:
                            i = 0
                            d = {}
                            while i < len(x):
                                d[x[i][0]] = r[i]
                                i = i+1
                            resultsList.append(d)
                        cursor.execute("select * from aplicaciones_consultadas
```

```
    where HASH=%s",[tofind])
if cursor.rowcount!=0:
       ts = time.time()
       timestamp = datetime.datetime.fromtimestamp(ts).
           strftime('%Y-%m-%d %H:%M:%S')
       cursor.execute("update aplicaciones_consultadas set
           fecha=%s where HASH=%s",(timestamp,[tofind]))
else:
       cursor.execute("insert into aplicaciones_consultadas
            (HASH) VALUES (%s)",[tofind])
dba.commit()
client = MongoClient('localhost', 27017)
conn=pymongo.MongoClient()
db = conn.aplicaciones
res =db.permisos.find({"HASH": tofind})

salida=db.static_apk_install.aggregate([{'$match':{'HASH':
    tofind}},{'$project': {'HASH': 1,'cantidad': { '$size':
     "$log" }}}])
num_apk_install=None
for doc in salida:
    num_apk_install=doc['cantidad']
apk_install=db.static_apk_install.find({"HASH": tofind})

salida=db.static_inject.aggregate([{'$match':{'HASH':
    tofind}},{'$project': {'HASH': 1,'cantidad': { '$size':
     "$log" }}}])
num_inject=None
for doc in salida:
    num_inject=doc['cantidad']
injects=db.static_inject.find({"HASH": tofind})

salida=db.static_root.aggregate([{'$match':{'HASH': tofind
    }},{'$project': {'HASH': 1,'cantidad': { '$size': "$log
    " }}}])
num_root=None
for doc in salida:
    num_root=doc['cantidad']
roots=db.static_root.find({"HASH": tofind})

salida=db.static_ips.aggregate([{'$match':{'HASH': tofind
    }},{'$project': {'HASH': 1,'cantidad': { '$size': "$log
    " }}}])
num_ips=None
for doc in salida:
    num_ips=doc['cantidad']
s_ips=db.static_ips.find({"HASH": tofind})

salida=db.static_sms.aggregate([{'$match':{'HASH': tofind
    }},{'$project': {'HASH': 1,'cantidad': { '$size': "$log
    " }}}])
num_sms=None
for doc in salida:
    num_sms=doc['cantidad']
smss=db.static_sms.find({"HASH": tofind})

salida=db.IPsPaises.aggregate([ {'$match':{'HASH':tofind
    }},{ '$unwind' : "$IPs" }, { '$group' : {'_id' : "$_id"
```

```
                    , 'IPs' : {'$sum':1} } } ]);
                num_dips=None
                for doc in salida:
                    num_dips=doc['IPs']
                paises=db.IPsPaises.aggregate([{'$match':{'HASH':tofind}},{
                    '$unwind': '$IPs'},{'$group': {'_id': '$IPs.Pais', 'sum
                    ': {'$sum': 1}}}]);
                maliciosas=db.IPsMaliciosas.find({"HASH": tofind})

                salida=db.payload.aggregate([{'$match':{'HASH': tofind}},{'
                    $project': {'HASH': 1,'cantidad': { '$size': "$log"
                    }}}])
                num_pay=None
                for doc in salida:
                    num_pay=doc['cantidad']
                payload=db.payload.find({"HASH": tofind})

                tiene_valoracion_permisos=False
                puntuacion_permisos=None
                cursor.execute("select Puntuacion from puntuaciones where
                    HASH=%s",[tofind])
                if cursor.rowcount!=0:
                        tiene_valoracion_permisos=True
                        puntuacion_permisos = cursor.fetchone()
                cursor.close()
                dba.close()
                return render(request,'informacion.html', {'resultado':
                    resultsList, 'permisos':res, 'num_apk_install':
                    num_apk_install,'apk_install':apk_install,'num_inject':
                    num_inject,'injects':injects,
                'num_root':num_root,'roots':roots, 'num_ips':num_ips, '
                    s_ips': s_ips, 'num_sms':num_sms, 'smss':smss, '
                    num_dips':num_dips, 'paises':paises, 'maliciosas':
                    maliciosas, 'eshash': eshash,
                 'num_pay': num_pay, 'payload': payload, '
                     tiene_valoracion_permisos':tiene_valoracion_permisos, '
                     puntuacion_permisos':puntuacion_permisos})
            else:
                    return HttpResponseRedirect('/')
```

G. Página HTML de información de la aplicación

```
{% extends "base.html" %}
{% block body %}
{% ifequal eshash 0 %}
  <div class="table-responsive">
    <table class="table">
      <tr>
      <th>Hash</th>
      <th>Nombre Paquete</th>
      <th>Estado</th>
      <th>Fecha</th>
    </tr>

    {% for hl in hashlist %}
    <tr>
      <td><a href="{% url 'index' %}buscar/?busqueda={{ hl.HASH }}">{{ hl.HASH }}
```

```
          </a></td>
        <td>{{ hl.nombre_paquete }}</td>
        <td>{{ hl.estado }}</td>
        <td>{{ hl.fecha }}</td>
      </tr>
      {% endfor %}
      </table>
    </div>

{% else %}
    <script src="https://code.highcharts.com/highcharts.js"></script>
    <script src="https://code.highcharts.com/modules/exporting.js"></script>
    <script src="https://code.highcharts.com/highcharts-3d.js"></script>

    <div class="panel-group">
        <div class="panel panel-default">
          <div class="panel-heading">
            <h4 class="panel-title">
              <a data-toggle="collapse" href="#collapse1">Datos de la aplicacion</a>
            </h4>
          </div>
          <div id="collapse1" class="panel-collapse collapse in">
            <div class="panel-body">

              <div class="table-responsive">
                <table class="table">
                  <tr>
                  <th>Hash</th>
                  <th>Nombre Paquete</th>
                  <th>Estado</th>
                  <th>Fecha</th>
                </tr>

                {% for x in resultado %}
                <tr>
                  <td>{{ x.HASH }}</td>
                  <td>{{ x.nombre_paquete }}</td>
                  <td>{{ x.estado }}</td>
                  <td>{{ x.fecha }}</td>
              </tr>
                {% endfor %}
                </table>

              </div>
                 </div>

          </div>
        </div>
      </div>



    <br><br>

    {% if tiene_valoracion_permisos %}

    <div class="panel-group">
        <div class="panel panel-default">
```

```
        <div class="panel-heading">
          <h4 class="panel-title">
            <a data-toggle="collapse" href="#collapse3">Peligrosidad segun
                permisos</a>
          </h4>
        </div>
        <div id="collapse3" class="panel-collapse collapse in">
          <div class="panel-body">

            <div class="table-responsive">
             <div class="progress">
                {% if puntuacion_permisos.0 < 0.2 %}
                 <span style="width: 20%;"><span>{{ puntuacion_permisos.0 }}</span
                     ></span>
                {% elif puntuacion_permisos.0 < 0.4 %}
                 <span class="green" style="width: 40%;"><span>{{
                     puntuacion_permisos.0 }}</span></span>
                {% elif puntuacion_permisos.0 < 0.6 %}
                 <span class="orange" style="width: 60%;"><span>{{
                     puntuacion_permisos.0 }}</span></span>
                {% elif puntuacion_permisos.0 < 0.8 %}
                 <span class="red" style="width: 870%;"><span>{{
                     puntuacion_permisos.0 }}</span></span>
                {% elif puntuacion_permisos.0 < 1 %}
                 <span class="blue" style="width: 100%;"><span>{{
                     puntuacion_permisos.0 }}</span></span>
                {% endif %}

              </div>

            </div>
          </div>

        </div>
      </div>
    </div>

  <br><br>
  {% endif %}
{% endifequal %}
{% endblock %}
```