Universidad Zaragoza

Trabajo Fin de Grado

Análisis del entorno Android: aplicaciones y el sistema de permisos

Android Environment Analysis: Applications and Permissions System

UNIVERSITÀ DEGLI STUDI DI TORINO

Departamento de Informática e Ingeniería de Sistemas

Grado en Ingeniería Informática

Autor: Sergio Lázaro Magdalena

Director: Valerio Costamagna
Ponente: Javier Fabra Caro

UNIZAR

Febrero de 2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. SERGIO LÁZARO MAGDALENA,

con nº de DNI 73013178T en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) GRADO, (Título del Trabajo) ANÁLISIS DEL ENTORNO ANDROID: APLICACIONES Y EL SISTEMA DE PERMISOS

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 2 DE FEBRERO DE 2017

Fdo: SERGIO LÁZARO MAGDALENA

RESUMEN

Los teléfonos inteligentes o smartphones están extendidos por todo el mundo en sus diversas plataformas siendo Android la más extendida en el mercado. Estos dispositivos emplean aplicaciones o apps para incrementar el abanico de posibilidades que ofrecen al usuario.

El gran número de smartphones que existen en la actualidad empleando Android y la libertad a la hora de publicar nuevas aplicaciones por desarrolladores desconocidos hacen de esta plataforma un blanco fácil para la difusión de aplicaciones maliciosas, también conocidas como *malware*. A diferencia de Google, Apple realiza una revisión de las aplicaciones que se publican en su App Store y obliga a que toda aplicación esté certificada por Apple, añadiendo así una capa adicional de seguridad controlada directamente por ellos.

Para limitar la acción de dichas aplicaciones, estas grandes marcas implementan una serie de medidas de seguridad basadas en un sistema de permisos. Para que una aplicación pueda acceder a los diferentes componentes que ofrece un smartphone y desplegar toda su funcionalidad, el usuario debe confirmar previamente los permisos requeridos por la aplicación. En el caso de Android, el número de aplicaciones disponibles en la Play Store supera los dos millones, número que aumenta cada año considerablemente debido a la facilidad con la que un desarrollador puede publicar su propio trabajo.

Este proyecto se centra en el análisis del comportamiento de las aplicaciones y de un análisis del sistema de permisos de Android. Para ello, se ha realizado un analizador estático de aplicaciones para obtener estadísticas junto a una investigación sobre el forzado de los permisos por parte de las aplicaciones obteniendo un mapeo de permisos, clases y métodos.

Ambas partes han producido resultados útiles a partir de los cuales se ha podido generar una prueba de concepto empleando los permisos más importantes junto con información del mapeo. De esta forma, se ha podido mostrar la versatilidad que aportaría un buen mapeo en el desarrollo de una herramienta de instrumentación dinámica para el análisis de aplicaciones.

Índice general

Índice de figuras

Capítulo 1

Introducción

En el presente capítulo se muestra una breve introducción al proyecto. Dividido en motivación, objetivo y organización, se pretende dar una pequeña explicación de lo que va a tratar el proyecto así como su organización y la estructura del documento.

1.1. Motivación

Los *smartphones* o teléfonos inteligentes han conseguido una gran difusión en la sociedad debido a la necesidad de integrar tareas en un solo dispositivo. Desde la primera aparición de un teléfono inteligente, el Ericsson GS88 "Penelope" [Xat14] en el año 1983, estos dispositivos inteligentes no han dejado de evolucionar impulsados por las necesidades del mercado y la competencia entre las grandes empresas del sector. En la actualidad se han vendido cerca de un 1.500 millones de estos dispositivos a usuarios finales.

El crecimiento del número de ventas de estos dispositivos a lo largo de los años y la posibilidad de integrar toda la información personal en un solo dispositivo ha provocado que los *smartphones* se conviertan en un foco de interés para *hackers* con intenciones maliciosas. Un *smartphone* cuenta con la misma capacidad que un ordenador personal por lo que se incrementa así el tamaño de los posibles vectores de ataque a través de los cuales se puede acceder a información personal. Para proteger esa información personal, tanto Android como iOS cuentan con un sistema de permisos para limitar el acceso a distintas características del dispositivo aunque, en muchos casos, este sistema no es suficiente.

1.2. Objetivo

Con la política de Google en la que cualquier desarrollador puede publicar sus aplicaciones y, a diferencia de Apple, que no se aplique ningún tipo de filtro a las mismas, provoca que sea un mercado muy accesible para introducir *malware*. Por ello, la comunidad investigadora trata de publicar herramientas útiles para el análisis de aplicaciones.

La primera barrera a la que se enfrenta una aplicación es el sistema de permisos de Android. A pesar de ser útil, no es suficiente para dotar de seguridad el uso de una aplicación. Por ello, el proyecto gira entorno a los permisos de las aplicaciones y al proceso de verificación realizado cada vez que se accede a una funcionalidad que lo requiere.

Partiendo de proyectos realizados por otros investigadores, en este documento se va a tratar de resolver las limitaciones que sus proyectos imponen, profundizar en el sistema de permisos y, finalmente, establecer una unión de los resultados obtenidos durante el desarrollo del proyecto con una herramienta para la modificación del flujo de ejecución de una aplicación.

1.3. Organización

La documentación del proyecto sigue una línea progresiva de menos técnico a más técnico con la intención de que se pueda realizar una lectura completa sin necesidad de buscar información adicional para su completa comprensión.

El capítulo 2 define los conocimientos básicos a los que se va a hacer referencia a lo largo de todo el proyecto. Está compuesto por una explicación de los distintos tipos de análisis de aplicaciones en la sección 2.1, una mención a diferentes portales web que ofrecen análisis de aplicaciones en la sección 2.2, una definición de Android Open Source Project (AOSP) en la sección 2.3. Está formado por distintos elementos considerados de interés para comprender el proyecto, como la organización de las aplicaciones, el Compatibility Test Suite (CTS) y el Binder. Para finalizar, se comentan las principales herramientas desarrolladas por terceros y que han sido empleadas para obtener información del contenido de las aplicaciones analizadas.

El capítulo 3 tratará de mostrar la situación actual del entorno de investigación en Android, un entorno que evoluciona rápidamente debido a la gran cantidad de investigadores y a las facilidades que ofrece el propio sistema operativo al ser *open source*. Finalmente, se hablará de un proyecto de gran prestigio, Pscout, que ofrece un mapeo de permisos con otros aspectos interesantes como serían los métodos de una clase determinada. Dicho proyecto se tratará de mejorar a través de una investigación realizada.

El capítulo 4 hace referencia a los aspectos más relevantes del análisis de aplicaciones estático, comentando sus ventajas, sus inconvenientes y la ampliación de posibilidades que ofrece el análisis dinámico durante un proceso de análisis. Le sigue el capítulo 5 que contiene la explicación del proceso de investigación realizado sobre el sistema de permisos de Android: funcionamiento, modificaciones del AOSP y objetivos a conseguir.

En el capítulo 6 se analizarán los resultados obtenidos tras analizar una muestra de aplicaciones y modificar el AOSP para obtener un nuevo mapeo de permisos. Además, se explicará como se pueden integrar los resultados con una herramienta desarrollada por el director del proyecto.

Por último, en el capítulo 7 se comentarán las conclusiones obtenidas así como los objetivos cumplidos y las líneas de trabajo futuro que ofrece el proyecto.

A continuación, en la figura 1.1, se muestra una imagen resumen de los componentes

que forman el proyecto junto con el capítulo en el que se comentan. Se pretende dar con esta figura una visión general que se irá completando con más información a lo largo de este documento.

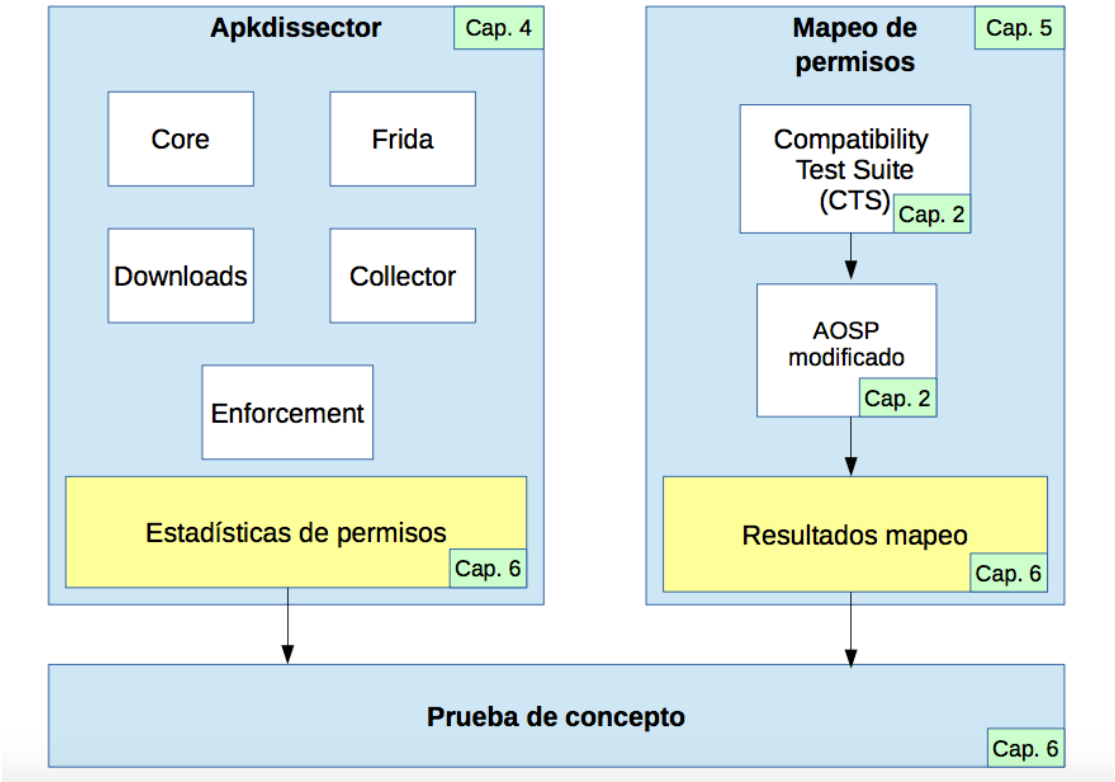

Figura 1.1: Visión general de los componentes del proyecto

Finalmente, se presentan una serie de Anexos a los que se hace referencia en las distintas partes del proyecto. Para explicar de forma más concisa su contenido o aclarar la explicación dada, algunos incluyen información más detallada, código o ejemplos de uso. Además, para facilitar la lectura del proyecto se ofrece un diccionario de términos en el Anexo B. Se recomienda la lectura del documento teniendo a mano el diccionario de términos para consultar en caso de duda.

Capítulo 2

Contexto

En este capítulo se definen conceptos básicos para la comprensión del proyecto. Se pretende dar una introducción a los distintos aspectos que se van a abordar en capítulos posteriores.

2.1. Análisis de malware

El *malware*, conocido comúnmente como virus, está presente desde los primeros ordenadores. Las tecnologías de la información facilitan mucho el día a día en la sociedad pero siempre existe la posibilidad de tener fugas de información, pérdida de datos o pérdida de control de un sistema. Por ello, el análisis del *malware* se vuelve necesario.

Generalmente, el *malware* es todo código desarrollado con un propósito malicioso pero, actualmente, con el avance de las tecnologías de la información y su accesibilidad, el propio concepto de *malware* ha evolucionado. De esta forma, se entiende como *malware* todo tipo de virus, gusano (*worms*), herramientas de intrusión, software espía (*spyware*) o herramientas de acceso con privilegios(*rootkits*).

En la sociedad actual la información es muy valiosa lo que ha provocado que el *malware* evolucione alcanzando un punto comercial. Esto ha provocado que *hackers* con intenciones malvadas ofrezcan sus servicios con el fin de obtener grandes sumas de dinero por vender reportes de información o vulnerabilidades que aún no han sido descubiertas. A pesar de ello, no todo es malo. Empresas como Instagram, en caso de recibir información sobre un fallo de seguridad, llegan a pagar una suma de dinero como agradecimiento [CBC16].

El análisis de *malware* es un proceso complejo en el que confluyen diversos intereses dentro de una empresa o entidad. En un primer lugar y como punto más interesante, se analizaría un *malware* para conocer el impacto que ha podido tener y descubrir los indicadores de compromiso. Además, podría ayudar a localizar vulnerabilidades desconocidas, conocer las capacidades técnicas del individuo o individuos que han logrado la intrusión e incluso dar con ellos.

Previamente se han comentado los intereses de analizar *malware* a nivel técnico pero no solo hay que dar relevancia al nivel técnico sino que a una empresa puede valorar más la repercusión económica y social que puede tener una intrusión en su sistema de información. Sería de interés conocer el porqué del ataque a dicha entidad, como llegó el intruso a ellos, cual sería el propósito, qué se podría haber llevado y cuanto tiempo habría permanecido allí.

Una intrusión a un sistema de información y su posterior brecha de datos, si el atacante llega a conseguirlo, podría acarrear problemas al responsable de esa información si los datos no están protegidos correctamente, en el caso de España, según la Ley Orgánica de Protección de Datos (LOPD).

2.1.1. Creación de un entorno seguro

Según el tipo de análisis que se vaya a llevar a cabo es necesaria la creación de un entorno seguro en el que no se propague por el resto del sistema factible mediante el uso de máquinas virtuales. Muchos *malware* intentan extenderse si les es posible por lo que es necesario emplear las posibilidades que ofrecen dichas máquinas virtuales para limitar el uso de la red. En ciertos casos, el *malware* podría ser capaz de explotar una vulnerabilidad del sistema operativo empleado para el análisis pudiendo llegar a escapar del entorno aislado configurado.

Ciertos *malware*s son capaces de detectar que se encuentran en un sistema aislado y, en caso de detección, el *malware* podría no ejecutarse para evitar así el análisis dinámico.

2.1.2. Tipos de análisis

El análisis de *malware* [KK07] se divide fundamentalmente en dos tipos: estático y dinámico. La mejor opción suele ser combinar ambos tipos para aprovechar las ventajas que ofrece cada uno de ellos.

Análisis estático

El análisis estático trata de analizar el *malware* sin ejecutarlo, es decir, se realiza una "autopsia" del código, implicando, en muchos casos, la necesidad de realizar ingeniería inversa a dicho código con el fin de comprender su funcionamiento. Al no llegar a ejecutar el código es más seguro y no es obligatorio crear un entorno seguro.

Análisis dinámico

El análisis dinámico trata de analizar el comportamiento de un *malware* mientras se ejecuta en un entorno controlado. De esta forma, se podría analizar su comportamiento en tiempo de ejecución e incluso se podría obtener información en claro a través de la salida generada por la muestra si el desarrollador no ha eliminado los mensajes de aclaración aunque esto es poco probable.

2.2. Webs de análisis de seguridad de aplicaciones

Existe un gran número de proyectos que ofrecen un portal web para que los usuarios analicen una aplicación o unaUniform Resource Locator (URL). Cada uno de ellos tiene unas características diferentes pero en su mayoría relegan el análisis a terceras partes como, por ejemplo, a antivirus. Con ello, se pueden escanear aplicaciones sin necesidad de instalar los antivirus. En este proyecto se han considerado relevantes Virustotal, Andrototal y Koodous.

2.2.1. VirusTotal

Virustotal [Sis04] es una aplicación web que ofrece la posibilidad de enviar un fichero y, partiendo de los datos que ofrecen distintos antivirus, responde al usuario con información sobre dicho fichero, comunicando si es *malware* detectado. Emplea análisis de antivirus conocidos como Avast, Symantec, Kaspersky o McAfee, entre otros.

Ofrece una Application Programming Interface (API) mediante una clave de pago o, en caso de investigación, se puede pedir una clave tras enviar una solicitud con información del proyecto que se pretende realizar y con firma del profesor titular al cargo. La clave de Virustotal obtenida es de la Università degli Studi di Torino pero no se ha llegado a emplear.

2.2.2. Andrototal

Andrototal [MVZ13] es un proyecto similar a Virustotal pero especializado en aplicaciones Android. Realiza un análisis de la aplicación devolviendo información sobre dicha aplicación como los permisos empleados, los paquetes que contiene o las *activities* de la aplicación.

También ofrece una clave privada para acceder a la API que, como en el caso de Virustotal, se ha empleado la obtenida por la Università degli Studi di Torino. A diferencia de Virustotal esta clave si que se ha empleado ya que con la API de Andrototal se pueden descargas aplicaciones para su posterior análisis. Por ello, la clave se empleó en unos scripts desarrollados para agilizar la descarga.

2.2.3. Koodous

Koodous [RLVS] es un proyecto mucho más ambicioso y se podría decir que menos fiable que los anteriores ya que relega el análisis a los propios usuarios. Koodous funcionaría con los análisis realizados por terceras personas de los que no se sabe si tienen los conocimientos suficientes para localizar un *malware* en una aplicación. Los usuarios pueden comentar publicaciones ajenas dando su puntuación de si es cierto o no que contiene *malware* pero no se asegura nada por lo que se recomienda no emplear Koodous para análisis de aplicaciones relevantes que, por ejemplo, se podrían desplegar en una empresa.

A diferencia de los anteriores, ofrece claves públicas con posibilidad de descargar o analizar cincuenta aplicaciones diarias mientras que, con claves privadas, dicho número aumenta.

Se ha empleado en la medida de lo posible con las claves públicas y sus correspondientes limitaciones.

2.3. Android Open Source Project

Android, al ser un proyecto de *open source* desarrollado por Google, ofrece su código para modificarlo. Se ofrecen unos repositorios en los que se encuentra el código de todas las versiones estables de Android para portar tanto a dispositivos reales como a accesorios que cumplen unos ciertos requisitos. Además, ofrece herramientas como el CTS que se explicará en la sección 2.3.3 así como herramientas para mejorar la seguridad del sistema desarrollado por terceras partes.

Con el paso de los años han ido apareciendo nuevas versiones que parten de las oficiales de Google pero que no son desarrolladas ni mantenidas por ellos. Ejemplos de ello podrían ser CyanogenMod [Cya] o CopperheadOS [Cop] que llegan a afirmar que son imagenes más seguras y más eficientes que la imagen oficial de Google. En los últimos meses Cyanogen anunció que iba a dejar este mercado.

2.3.1. Estructura de una aplicación Android

Al agrupar los ficheros que componen una aplicación se forman los archivos Android Application Package (APK) [Wika], la extensión empleada por Google. Es similar al formato *ZIP* de compresión. En la figura 2.1 se puede observar los componentes que la forman de forma más visual. Para comprender bien cada uno de estos componentes se va a realizar una breve explicación de cada uno de ellos:

- *AndroidManifest.xml:* Fichero principal de la aplicación Android con información como las *Activities* de una aplicación, los permisos declarados, *Services* empleados, *Content Providers*, nombre y versión, entre otra información importante.
- *assets:* Recursos como imágenes o ficheros que pueden ser obtenidos a través del *AssetManager*.
- *classes.dex:* Contiene el código Java compilado en Dalvik bytecode para la Dalvik Virtual Machine, código comprensible para Android.
- *res:* Directorio que contiene recursos no compilados en *resources.arsc*.
- *META-INF:* Directorio que contiene información para mantener la integridad de la aplicación y la seguridad del sistema.
- *resources.arsc:* Fichero que contiene rescursos precompilados como ficheros eXtensible Markup Language (XML) binarios.

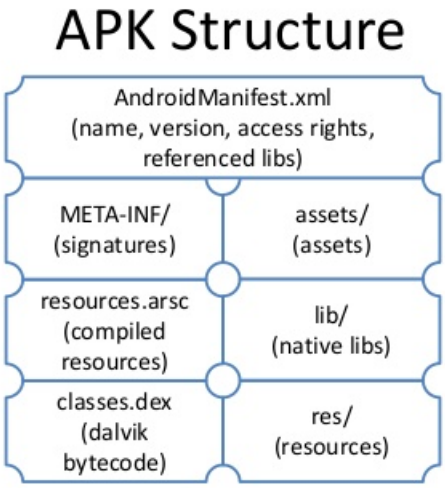

Figura 2.1: Estructura de un fichero APK

2.3.2. Ofuscación en Android

La ofuscación [Wikb] trata de hacer lo menos legible posible el código de un determinado programa o aplicación. En el caso de Android, al ser tan fácil aplicar ingeniería inversa, el uso de ofuscadores es prácticamente obligatorio. Para ello, Google ofrece su propio ofuscador, Proguard [Good], que optimiza el código y lo ofusca.

Por otro lado, terceras partes han elaborado ofuscadores que dificultan más la comprensión, como puede ser DexGuard [Gua].

La ofuscación impondrá limitaciones durante el proyecto ya que algunas herramientas ya cuentan con ella. A pesar de dificultar el trabajo, existen herramienta en contra de los ofuscadores que tratan de encontrar relaciones y volver a un código legible.

2.3.3. Compatibility Test Suite

Como se ha citado en la sección anterior, el CTS [Goob] se encuentra incluido en el AOSP. Se encarga de comprobar el correcto funcionamiento de una imagen modificada u oficial de Android instalada en un dispositivo real. Para ello, realiza pruebas de toda la funcionalidad incluida dentro de Android a través de la instalación de aplicaciones de prueba.

Para realizar un buen test de compatibilidad, Google pide al usuario que vaya a realizar dicho test que trate de no interferir durante el proceso ya que emplea completamente el framework, como puede ser la cámara, la pantalla, el bluetooth o el wifi. Tras finalizar el test, se facilita un reporte con la información generada durante la ejecución. En el capítulo 5 se explica el motivo por el cual se ha considerado de interés.

2.3.4. Binder

El Binder [Gooh] de Android es el encargado de realizar las comunicaciones internas dentro del sistema operativo de Google. Con él, se accede a las funcionalidades disponibles por el dispositivo como pueden ser la cámara o el acceso al estado de la red wifi ya que a través del Binder, un proceso de Android puede comunicarse con otro proceso.

Android emplea el Binder como un método Inter-Process Communication (IPC), es decir, como elemento intermediario para que sea posible la comunicación entre procesos, implementado con un protocolo sencillo de paso de mensajes. Al estar involucrado el kernel de Linux, Android aprovecha todas sus facetas y, para intermediar entre el framework y el kernel, se integró el Binder.

En este proyecto es de mucho interés ya que impuso la mayoría de las limitaciones encontradas. En este caso, el Binder toma parte en el proceso de verificación de los permisos otorgados a una aplicación por lo que tendrá una serie de consecuencias que se discutirán en el apartado 5.

2.4. Herramientas empleadas

En la presente sección se comentan dos de los proyectos más útiles y más empleados a la hora de analizar aplicaciones Android: Androguard y Apktool.

2.4.1. Androguard

Androguard [DG] es una de las herramientas principales a tener en cuenta en un proceso de análisis estático de aplicaciones Android. Esta compuesto por un conjunto de scripts que son independientes del sistema operativo empleado ya que están codificados en Python.

Fundamentalmente, Androguard permite desensamblar y decompilar aplicaciones Android. Está formado por un conjunto de ficheros en los que se ofrecen funcionalidades diferenciadas. El fichero principal es *androlyzer.py* que ofrece una consola de comandos con la que se pueden ir realizando comprobaciones a la aplicación analizada y obtener así los permisos empleados, las *activities*, las clases, los métodos o los strings. Al ser *open source*, es posible integrar el código en una herramienta propia como se verá más adelante en el capítulo 4. Por otro lado, otro aspecto importante es que emplea caches de datos una vez que se ha abierto una aplicación, es decir, si se abre una aplicación, las siguientes veces que se analice esa aplicación el tiempo se reducirá.

Para analizar el AndroidManifest que forma parte de la aplicación Androguard ofrece el fichero *androaxml.py* que al pasarle una aplicación como parámetro escribirá el AndroidManifest en el fichero que se le especifique. Contiene además la herramienta *androsim.py* que compara dos ficheros APK para encontrar similitudes o la herramienta *androdd.py* que exporta todos los métodos que contiene una aplicación en un directorio de salida pasado como parámetro y puede incluso reconstruir las clases JAVA de la aplicación con algunas limitaciones.

En el apéndice C se muestra la funcionalidad básica de Androguard con una aplicación de muestra obtenida de las webs citadas en la sección 2.2.

2.4.2. Apktool

Apktool [Wi] es otra de las herramientas fundamentales para el análisis estático. Cuenta con dos opciones fundamentales de trabajo: una para decodificar y otra para codificar. La opción de decodificación sobre un fichero APK permite obtener los ficheros que lo integran así como los ficheros XML que forman la aplicación (entre ellos el Manifest) y los ficheros Dalvik Executable (DEX).

Una vez decodificada la aplicación, se podría modificar el código smali que forma los ficheros DEX y, por consiguiente, alterar una aplicación de una tercera parte. Tras la modificación, apktool es capaz de volver a codificar dicha aplicación y ensamblarla en un fichero APK. De esta forma, se generará una nueva aplicación que al ser distinta de la inicial se tendrá que volver a firmar en el caso de que se quiera publicar en Google Play.

2.4.3. Herramientas de hook

En esta sección se van a comentar dos framework de análisis dinámico de aplicaciones mediante la modificación de las invocaciones a métodos (*hookeo*): ARTDroid y Frida. Ambos proyectos tienen un objetivo común pero se diferencian en el modo de aplicarlo ya que ARTDroid emplea la inyección de librerías mientras que Frida parte de una arquitectura cliente/servidor.

ARTDroid

ARTDroid [CZ] es la herramienta de análisis dinámico desarrollada por el director de este proyecto durante el Google Summer of Code (GSOC). Teniendo en cuenta que el proyecto partía con la intención de desarrollar una herramienta automática de análisis que integrase ARTDroid, se consideró importante comentarla.

ARTDroid es un framework que realiza *hooks* de llamadas virtuales en Java a través de la alteración de su tabla virtual, encargada de almacenar las direcciones de los métodos existentes. En la figura 2.2 se muestra su funcionamiento con un ejemplo de alteración de la función *getDeviceId()*.

Para alterar la tabla virtual es necesario escribir el método Java y sobrescribir el método objetivo empleando ARTDroid. Para ello, se emplea la inyección de librerías lo que implica que el dispositivo cuente con privilegios de Root. Una vez que se tiene la librería inyectada para realizar el *hook* se debe de insertar en la memoria virtual de la aplicación.

Nada más iniciarse el proceso, ARTDroid sobrescribe las tablas *vtable_* y *virtual_methods_* mediante la escritura de la dirección del *patchcode*. A pesar de modificar ambas tablas, incluye dentro de su estructura de datos los contenidos existentes en ambas. Una vez interceptado un método, todas las llamadas a ese método van a ser

interceptadas, ejecutándose el código existente en la librería inyectada. Por otro lado, ARTDroid ofrece también la posibilidad de llamar al método original.

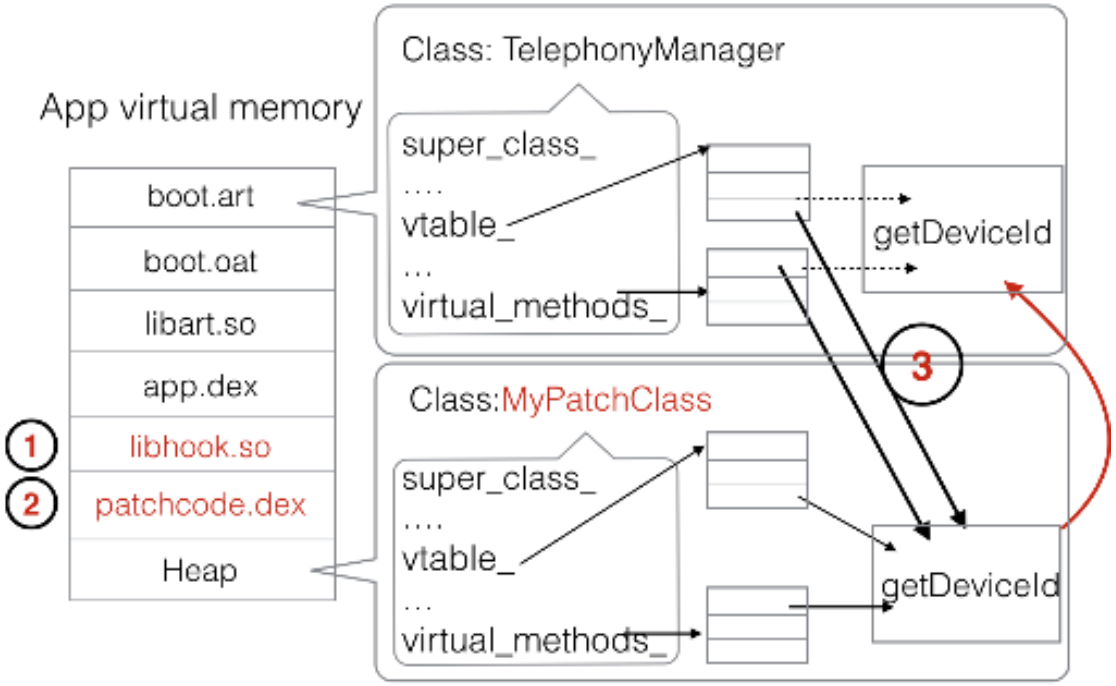

Figura 2.2: Interacción de ARTDroid

Para establecer que llamadas se tienen que interceptar, ARTDroid emplea un fichero de configuración que contiene datos como el nombre de la clase, el nombre del método, parámetros, valor de retorno y la clase a la que se deben de dirigir las invocaciones.

```
{"config" :
    {"debug" : 1,
    "dex" : [{"path":"/data/local/tmp/dex/target.dex"}],
    "hooks" : [{
        "class-name":"android/telephony/TelephonyManager",
        "method-name":" getDeviceId ",
        "method-sig":"()Ljava/lang/String;",
        "hook-cls-name":"org/sid/example/HookCls"
    }]
}}
```

En el ejemplo de la figura anterior se realiza una prueba con el método *getDeviceId()* pero, en nuestro caso, se deberá adaptar a la información obtenida durante el proceso de mapeo y, de esta forma, vincular los resultados obtenidos con la herramienta de *hook*. El motivo por el que se realizó la investigación del mapeo de permisos fue por este fichero

de configuración ya que se necesita el nombre de la clase, el nombre del método y los parámetros. Con la intención de elaborar una herramienta que se pudiese integrar con este framework y, a pesar de que Pscout ofrecía toda esta información, seguía siendo necesario obtener un mapeo correcto clase-método-permiso.

Finalmente no fue posible emplear este framework ya que cuando fue necesario estaba en proceso de adaptación. Afortunadamente se encontró otra herramienta más novedosa y que satisface igualmente las necesidades del proyecto.

Frida

Frida [Rav] es un proyecto open source reciente que ofrece un *toolkit* para la instrumentación de código dinámico. Su núcleo consiste en la inyección de código JavaScript o una librería propia en aplicaciones nativas Windows, Mac, Linux, iOS, Android o QNX.

Esta herramienta emplea Python y JavaScript en su núcleo lo que permite realizar un desarrollo rápido mediante la API que se ofrece. Por otro lado, se ofrece la posibilidad de emplear otros lenguajes como C, Node.js, Swift o .NET, entre otros.

Su funcionamiento está basado en una arquitectura cliente/servidor y necesita de un dispositivo rooteado. De esta forma es necesario realizar la instalación del cliente a través del comando *pip* de Python. Por otro lado se debe obtener un servidor de la plataforma y la arquitectura que se quiere emplear. Cuando el servidor este disponible en el dispositivo Android ejecutándose se podrán iniciar las pruebas. En la figura 2.3 se muestra la organización de la arquitectura de Frida y como se produce la comunicación de los elementos que integran la herramienta.

Frida ofrece diferentes componentes en su API de JavaScript de forma que se puede enviar mensajes por pantalla, emplear sockets, threads, sobrescribir métodos y funciones en Java o, lo que es más interesante, alterar el comportamiento de métodos nativos de Android, como podría ser la alteración del método que activa el Bluetooth, el método que permite obtener el IMEI del dispositivo o el método que emplea una determinada URL. Siendo esto posible, Frida se puede emplear perfectamente para aumentar la seguridad de una aplicación o para comprobar que la seguridad existente es óptima pero también puede servir para evitar una conexión HTTPS, modificar la URL de una conexión a una maliciosa o enviar un SMS a un servicio de subscripción premium. Para ello, en la sección 6.4.1 se van a mostrar los resultados obtenidos de las múltiples pruebas que se han realizo y que han concluido satisfactoriamente.

Además de su funcionalidad, Frida gestiona un sistema de excepciones con el objetivo de reportar la información del error sin que finalice la ejecución lo que es perfecto para una herramienta de análisis automático.

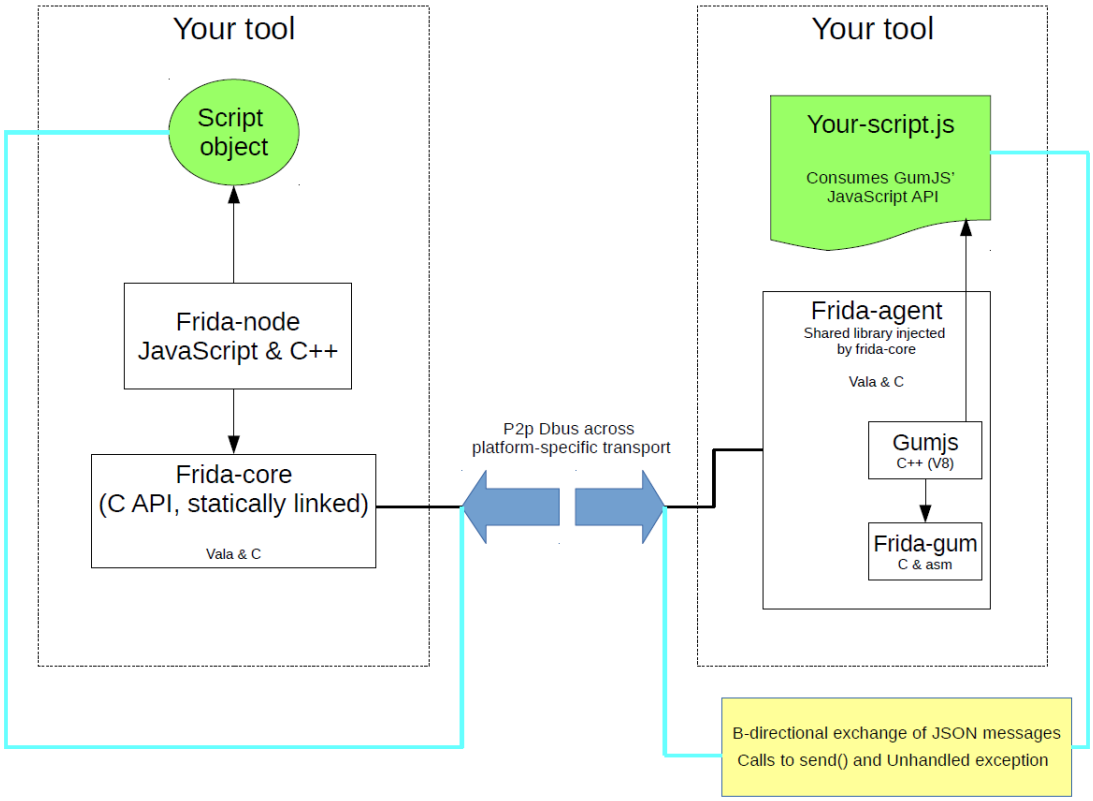

Figura 2.3: Interacción de Frida

Capítulo 3

Estado del arte

En este capítulo se van a comentar trabajos de investigación que han sido de gran interés para el desarrollo de este proyecto. Cada proyecto se va a comentar siguiendo una misma estructura: cómo funciona, qué objetivos pretenden conseguir y como se pueden emplear sus resultados en el proyecto desarrollado.

3.1. Inline Reference Monitor

Un Inline Reference Monitor (IRM) [Erl] es una metodología empleada para aumentar la seguridad de las aplicaciones. Para ello, se integran unas políticas de seguridad, también conocidas como *policies*, en una aplicación tras haber modificado su código original. Una vez modificada, en el caso de realizar una operación vinculada a las *policies*, la ejecución será igual que la original pero aplicando las políticas establecidas.

Para ilustrar de forma más detallada el funcionamiento de un IRM y aproximarlo a una situación en un entorno Android se ha considerado de interés el proyecto Appguard y la herramienta APIMonitor. En un primer lugar, en la sección sección 3.1.1, se va a explicar su funcionamiento, los componentes que lo forman y los objetivos que quiere conseguir. Más adelante, en el Anexo D, se analizan las implicaciones que tienen estas herramientas en el bajo nivel de Android.

3.1.1. AppGuard

El proyecto AppGuard - Enforcing User Requirements on Android Apps [BGH+] trata de incrementar la seguridad de las aplicaciones que instalan los usuarios mediante el empleo de *policies* para limitar la amplitud de acción que un permiso confirmado ofrece a una aplicación Android.

En versiones previas a Android Marshmallow (API level 23), los permisos se confirmaban en tiempo de instalación lo que impedía seleccionar que permisos se le otorgaban a una aplicación, es decir, era necesario aceptar todos los permisos para proceder con la instalación. A partir de Android Marshmallow los permisos pasaron a confirmarse en

tiempo de ejecución, permitiendo elegir qué permisos aceptamos y a que funcionalidad de la aplicación queremos acceder. Tras esta actualización, el usuario es quien controla la aplicación y no al revés.

Debido a la aceptación de permisos en tiempo de instalación surgieron proyectos como AppGuard para ampliar esa seguridad. El objetivo de estos proyectos consiste en modificar la aplicación descargada e introducirle las políticas de seguridad oportunas para un determinado permiso. Para ello, introducen un monitor que comprueba en tiempo de ejecución la llamada a las librerías del sistema de Android y verifica si esa acción está relacionada con alguna de las *policies*.

A continuación se van a describir los componentes que forman el proyecto y, en la figura 3.1, se mostrará de forma visual la interacción entre ellos.

Policies

AppGuard ofrece una serie de políticas de seguridad predeterminadas. Dichas políticas restringen la acción de permisos críticos de Android como pueden ser el permiso Internet, el permiso para acceder a los contactos o el permiso para enviar Short Message Service (SMS), muy codiciados por las aplicaciones con *malware*.

Rewriter

El Rewriter es el encargado de modificar la aplicación. Para ello, se descomprime el fichero APK y se obtienen los ficheros DEX, que posteriormente son desensamblados. Una vez obtenidos los ficheros desensamblados se procede a introducir las políticas de seguridad en la aplicación. Finalmente, se vuelve a ensamblar el código y se forma el fichero APK del que se partió en un principio.

Este proceso hace que la aplicación sea distinta que la original por lo que es obligatorio requerir al usuario la instalación de la aplicación asegurada. Además, toda aplicación de Android está firmada por el desarrollador por lo que, tras modificarla, es necesario volver a firmarla con una clave nueva. En el caso de AppGuard, las aplicaciones se firman con la misma clave ya que Google permite la comunicación entre aplicaciones que están firmadas con la misma clave, es decir, publicadas por el mismo desarrollador.

Manager

Ofrece la posibilidad de activar/desactivar políticas de seguridad introducidas en una aplicación.

Monitor

El monitor es el encargado de forzar las políticas de seguridad establecidas en el proceso de reescritura. Para ello, analiza el estado de la ejecución y decide cómo actuar según las políticas de seguridad elegidas mediante el Manager.

Con la implementación explicada se podría evitar que se compartiesen los contactos al confirmar el permiso de contacto, evitar el envio de SMS a números de teléfono

sospechosos (por ejemplo, servicios de subscripción premium) o evitar peticiones a URLs que no emplean Hypertext Transfer Protocol Secure (HTTPS).

El proyecto es muy interesante desde el punto de vista de incrementar la seguridad de una aplicación aunque tiene una serie de limitaciones que se analizan en el Anexo D.1.

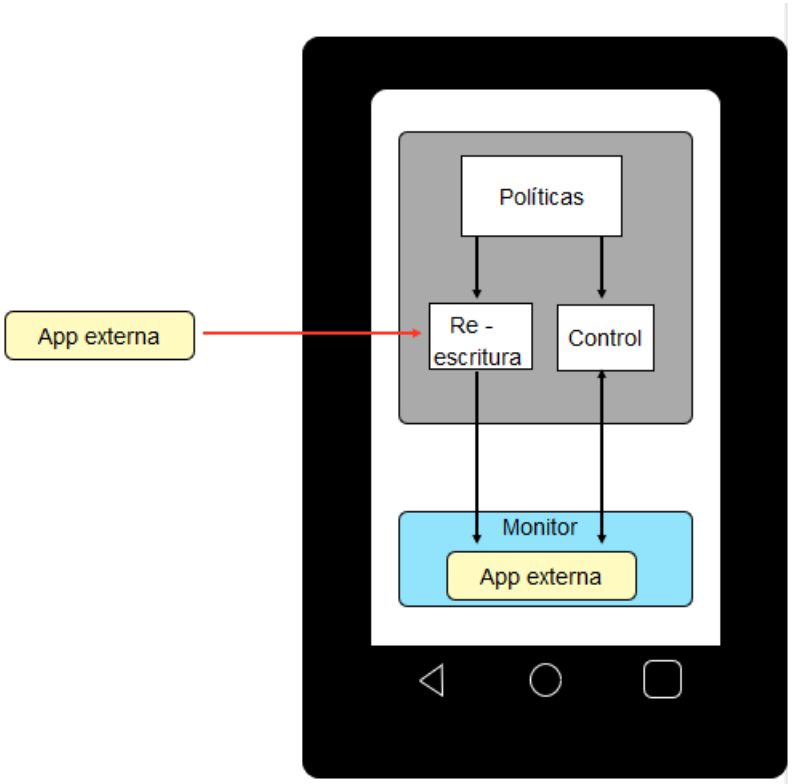

Figura 3.1: Estructura de AppGuard

3.2. Call Flow Graph

Un Call Flow Graph (CFG) [Gol] es una metodología empleada, sobre todo, en el análisis de software. Los CFG tiene la misma organización que los grafos comunes. En ellos, los nodos son las instrucciones del programa y los vértices muestran el flujo entre las instrucciones.

Tras el análisis de las implicaciones de AppGuard a bajo nivel, comentado en la sección D.1, una posible solución a las limitaciones que imponen algunos IRM serían los CFG. A continuación, como se ha realizado anteriormente con los IRM, se proporciona un proyecto de interés dentro de los CFG donde se explica su funcionamiento en el caso de aplicaciones Android y su lenguaje de bajo nivel (Smali).

3.2.1. Slicing Droids

El proyecto *Slicing Droids: Program Slicing for Smali Code* [HUHS] desarrollado por investigadores alemanes podría servir como solución al problema de *Java Reflection* que se explica en el Anexo D.1 y, para enlazar las limitaciones de un tipo de análisis con las ventajas de otro, se ha elegido el presente proyecto.

El proyecto ofrece la herramienta *SAAF*, un analizador estático para aplicaciones Android que emplea técnicas como data-flows y visualización de CFG. Además, ofrecen la posibilidad de emplear una Graphical User Interface (GUI) o, para facilitar tareas de automatización, ofrece también una línea de comandos.

En un primer lugar, el analizador lo que necesita es descomprimir el fichero APK. Para ello, *Slicing Droids* emplea la herramienta *Apktool*, comentada en la sección 2.4.2. De esta forma se obtiene los ficheros de interés de la aplicación, entre ellos el código Smali. *SAAF* emplea el código Smali para analizarlo y crear una aproximación apropiada de sus contenidos. Con ello, la herramienta empieza con el *program slicing* con el fin de localizar parámetros estáticos empleados en distintos métodos.

Debido a que realiza un análisis desde abajo hacia arriba, el *program slicing* sería el flujo de instrucciones que van siguiendo unos determinados registros (en el caso del *byte-code* de Android emplea registros virtuales). En un primer lugar, es necesario establecer una serie de criterios o *sclicing criterion* que incluiría el nombre del método, el nombre completo de la clase, los parámetros del método, el tipo de resultado que devuelve, en el caso de que devuelva, y, por último, el índice del parámetro que se debe seguir. De esta forma, se pretende identificar los *opcodes* que modifican o emplean el registro analizado hasta alcanzar una constante, es decir, una asignación a un registro.

Existen dos puntos en los que el análisis acabaría: si se alcanza un valor constante asignado a un registro o si una referencia a un objeto se almacena en el registro analizado. En el primer caso, se alcanza la constante buscada y, en el segundo, al tener una referencia a un objeto, se procede a analizar ese objeto que permitirá obtener todas las constantes incluidas en él.

Un ejemplo simple de su funcionalidad sería intentar localizar un número de teléfono en una aplicación que contiene el permiso SEND_SMS o una invocación a una función *toString()* en una aplicación que trata de obtener el International Mobile Station Equipment Identity (IMEI) del dispositivo mediante la clase *TelephonyManager*. En el mejor de los casos, se daría con dicho comportamiento.

Al automatizar tareas en un proyecto siempre pueden aparecer falsos positivos. Para ello, lo que implantan en este proyecto es un valor o *fuzziness*. Cuanto menor sea el valor del *fuzziness* mayor será la certeza y, cuanto mayor sea, mayor será la posibilidad de encontrarnos falsos positivos.

Anteriormente se ha mencionado que emplean búsquedas de la parte inferior del código a la superior pero, en realidad, es una combinación de búsquedas alternadas de abajo a arriba y de arriba a abajo para lograr alcanzar el objetivo. Esos aspectos de alternar el procedimiento serían necesarias en el caso de *arrays*, invocaciones a métodos o retorno de valores pero, por motivos de espacio limitado en el documento del proyecto, no lo explican.

La herramienta *SAAF* no se ha empleado en este proyecto pero se ha considerado interesante comentarlo ya que sería una solución a las limitaciones que tiene AppGuard y es una de las posibles líneas de trabajo futuro para la herramienta desarrollada en este proyecto.

3.3. Android permission mapping

Con el sistema de permisos de Android, se ofrece a los usuarios una protección imponiendo unas limitaciones a los desarrolladores de las aplicaciones. Para los investigadores de seguridad, ese sistema de permisos ha sido una de las grandes incógnitas a la hora de analizar aplicaciones sobre esta plataforma ya que Google no ofrece ninguna facilidad para conocer qué funciones están relacionadas con un determinado permiso.

Por ello, es normal que una de las líneas de investigación más optadas sea ese sistema de permisos. Varios proyectos importantes han surgido a lo largo de la vida del sistema operativo de Google y han tenido muy buen recibimiento dentro de la comunidad investigadora. Los dos más importantes son Stowaway, del que no es posible añadir la referencia ya que el propio autor ha considerado obsoleto su proyecto y redirecciona la búsqueda al siguiente proyecto importante: Pscout [AZHL].

A continuación, se comentan las partes interesante en las que se centra el proyecto de la Universidad de Toronto: el *background*, su diseño, su implementación y, finalmente, los resultados obtenidos.

3.3.1. Pscout

Pscout trata de encontrar una respuesta a la eterna pregunta para los investigadores de seguridad de Android: el mapeo de permisos con métodos y clases dentro de las librerías del sistema.

Para alcanzar el mapeo, desarrollaron un analizador estático que escaneaba todo el framework de Android haciendo uso de otro proyecto muy conocido, Stowaway, que fue de los primeros proyecto en analizar el sistema de permisos de Android y, aunque contaba con algunas limitaciones, la aproximación fue muy bien acogida para la versión de Android 2.2.

Background

Teniendo en cuenta que el framework de Android es complejo y permite la interacción entre distintos componentes, Pscout consideró importante analizar el comportamiento de las llamadas IPC, muy comunes en Android y donde los permisos están involucrados. Fundamentalmente, se distinguen dos tipos de mecanismos IPC en Android:

- *Intents:* Los Intents son mensajes unidireccionales con un String que indica la realización de una acción y con el que se puede hacer broadcast a todas las aplicaciones o enviar únicamente a una. Para restringir el uso de los Intents, se emplean permisos.

- *Binder:* Mecanismo que implementa Remote Procedure Call (RPC), comentado en el apartado 2.3.4. Las interfaces remotas se definen en un fichero Android Interface Definition Language (AIDL) y pueden ser invocados como si de un método local se tratase.

También hay que tener en cuenta los almacenes de datos del sistema, o Content Providers, empleados para dotar con datos persistentes a componentes de aplicaciones. Su acceso es a través de Uniform Resource Identifier (URI)s que empiezan con *content://*.

Android ofrece un sistema de permisos complejo formado por dos tipos: permisos para componentes del sistema, los llamados "system permissions" y los permisos regulares, empleados por aplicaciones y en los que se centra Pscout.

Diseño e implementación

El proyecto integra el desarrollo de un analizador estático y de un CFG para identificar todas las comprobaciones de permisos para el framework de Android. Una vez identificados, se genera un CFG donde se incluyen tanto las llamadas IPC como las RPC. Finalmente, se analiza desde el final del grafo hasta el inicio para ver las llamadas a la API de un cierto permiso.

Lo primero de todo es encontrar donde se fuerzan los permisos para cada uno de los elementos. Cada uno de ellos está relacionado con un método de una clase dentro del framework de Android. Así pues, el método *checkPermission* recibe como parámetro un String con el permiso que se va a comprobar y, si realmente se emplea en *checkPermission*, se sigue el flujo de uso de ese String. En caso contrario, si no aparece en la función, se considerará un Intent.

Para enviar y recibir Intents son necesarios permisos. Existen dos posibilidades: especificar el permiso que requiere un Intent en el Manifest o declarar el permiso en las llamadas a las funciones *sendBroadcast* y *registerReceiver*. Finalmente, Pscout obtiene información de ambas formas, analizando el Manifest y siguiendo el flujo de llamadas de las funciones citadas anteriormente, obteniendo así los permisos relacionados con los Intents.

Para conocer el caso de los Content Providers se emplea un objeto URI pasado a la clase *ContentResolver*. De forma similar a los Intents, también declaran los permisos en el Manifest por lo que Pscout analiza el Manifest para obtener los permisos relacionados a una determinada URI, es decir, se mapean los permisos requeridos con un Content Provider.

Una vez comprobados los componentes principales, faltarían de añadir los relacionados con las llamadas RPC e IPC. Al estar definidas en un fichero AIDL, analizando dicho fichero se obtendría la información de las interfaces remotas. Así pues, se procedería de forma similar a las anteriores para obtener el mapeo de la llamada al método con el permiso requerido. Existe un problema ya que estas llamadas añaden incertidumbre debido al empleo de invocadores genéricos. En Android, todo el uso de RPC depende del Binder lo que supone un problema que se analizará en el apartado siguiente.

Finalmente, tras el análisis, se va construyendo el grafo añadiendo por partes según el elemento analizado.

Resultados

Tras la investigación se publicaron ficheros Comma-Separated Values (CSV) para un gran número de versiones de Android. Estos resultados son de gran interés pero, a pesar de ser un gran proyecto, también tiene sus limitaciones. En este caso, y por los resultados obtenidos en el capítulo 5, se considera de interés el objeto *BluetoothAdapter*, empleado para todas las acciones relacionadas con el Bluetooth desde una aplicación Android. Para ello, se buscó un método de dicho objeto, como podría ser *isEnabled()*, que requiere el permiso de Bluetooth. Al buscar dicho método en los resultados se encuentra el mapeo con el permiso DUMP.

En el documento escrito por los desarrolladores de Pscout se menciona que no son capaces de comprobar los permisos que se fuerzan en el kernel y por ello aparece el problema del mapeo con el permiso DUMP que no sería el empleado por los desarrolladores en su declaración dentro del fichero *AndroidManifest.xml*.

Capítulo 4

Análisis estático de aplicaciones

En este capítulo se va a analizar de forma estática algunas aplicaciones Android para tratar de mostrar los pros y contras de este tipo de análisis. Como se ha hecho referencia en el capítulo 2, el análisis estático permite comprobar el funcionamiento de aplicaciones sin necesidad de ejecutarlas y, por tanto, no es necesario crear un entorno seguro.

Para llevar a cabo un análisis estático son necesarias una serie de herramientas citadas en la sección 2.4. Principalmente se va a utilizar Androguard ya que, en el caso de este proyecto, la utilidad de Apktool no se aprovecha en completo debido a que no se añade código a las aplicaciones de terceros.

En el análisis estático, para obtener información relevante se tiene que observar el código que contiene la aplicación, a veces en claro y otras veces en ensamblador. En el caso de Android, el código compilado es un punto medio entre ensamblador y código en claro ya que la notación que emplea, Smali, es fácil de comprender. Dicha notación emplea una serie de registros virtuales que emularían el sistema de registros de ensamblador.

En el anexo D se realiza un análisis del comportamiento de AppGuard, mencionado en la sección 3.1.1. En él se muestra como se comporta el proyecto y la forma con la que sería posible evitar la protección que añade en las invocaciones a los métodos de las clases que ofrece Android. Este anexo es de gran interés para conocer las debilidades del análisis estático a pesar de que sea el análisis más rápido y automático.

A continuación, se va a explicar la herramienta de análisis estático desarrollada en esta parte del proyecto. Se va a explicar la motivación de la herramienta, los componentes que la integran, para qué se quiere emplear cada uno de esos componentes y los resultados que permite obtener.

4.1. Apkdissector

En esta sección se va a presentar una herramienta para analizar un gran número de aplicaciones para almacenar información relevante en una base de datos. Con el fin de hacer más comprensible la herramienta, se va a partir del gráfico 4.1 formado por los componentes principales que se irán explicando a lo largo de esta sección. Es importante

resaltar que algunos de los componentes se han ido añadiendo durante el transcurso del proyecto según las necesidades que establecía cada una de las fases seguidas.

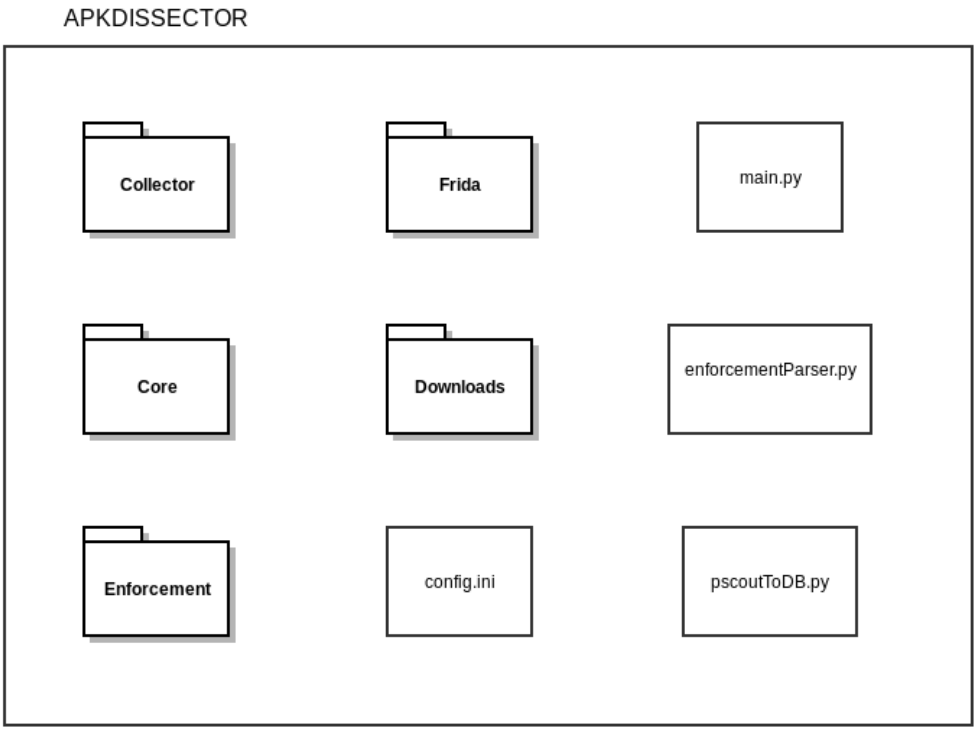

Figura 4.1: Componentes principales de Apkdissector

4.1.1. Collector

La función principal de este componente es recolectar información. Está compuesto por *scripts* Python para la recopilación de información a través de Androguard como *Activities*, *Services* o *Content Providers* y para la realización de consultas a las distintas bases de datos que se gestionan en el proyecto.

4.1.2. Core

Núcleo de la herramienta. En *core* se pueden localizar *scripts* encargados de tratar el fichero de configuración, las excepciones, el log, las estadísticas que se realizan, el gestor de *threads* y el objeto empleado como *thread*.

Con el fin de reducir el tiempo de análisis de una muestra de aplicaciones se decidió aplicar un sistema de *threads* especificados por el usuario en el fichero de configuración ofrecido y que emplea una cola First In First Out (FIFO). De esta forma, el tamaño de la cola dependerá del fichero de configuración y, por tanto, también el número de *threads*

lanzados. Una vez que se llena la cola, se hace una espera hasta que se vacía para volver a llenarla y lanzar otra vez la herramienta.

En cuanto a la labor que realizan los *threads* es bastante simple debido al limitado tiempo que se disponía para el proyecto. Cada thread analiza su aplicación y lee el fichero Manifest con el fin de conocer los permisos que emplea. En cambio, en un futuro, la herramienta se podría ampliar con más funcionalidad ya que, en el caso del Manifest, la herramienta se centra en los permisos y desprecia información como las *Activities*, los *Services* o los *Content Provider* debido a la naturaleza del proyecto. Además, hay gran cantidad de información dentro del código de la aplicación que únicamente se emplea en al sección 4.1.4.

4.1.3. Enforcement

Este componente es de interés únicamente para el proceso de análisis realizado en el capítulo 5 ya que se parte de un fichero JavaScript Object Notation (JSON) generado en esta etapa. A pesar de ello se ha unificado con la herramienta de análisis estático para tener todo integrado.

El componente incluye un *script* encargado de corregir el fichero JSON generado, leer dicho JSON una vez corregido y almacenar su información en una nueva base de datos para hacerla más accesible. Además, incluye también la posibilidad de relacionar la información obtenida con la información disponible en el proyecto PScout lo que será de utilidad en la sección 6.3 del capítulo de resultados.

4.1.4. Frida

En la parte final del proyecto se emplea la herramienta Frida, comentada previamente en la sección 2.4.3. A través de Androguard, se ha codificado una nueva utilidad capaz de generar el código necesario para interceptar las invocaciones realizadas por las clases Java de una aplicación Android a través de la API de Frida.

Para ello, es necesario analizar la aplicación. Mediante la herramienta Androguard incluida en Python se analizan las clases de la aplicación en busca de sus métodos. Conociendo una clase y sus métodos es posible elaborar un fichero que se empleará en un futuro para modificar el comportamiento de dichos métodos. Por otro lado, es necesario integrar el fichero generado anteriormente en un *script* Python ejecutable que se encargará de localizar el dispositivo en el que está Frida y con el que se debe comunicar para su funcionamiento. En la figura 4.2 se muestra el proceso para elaborar ambos ficheros.

Esta utilidad es interesante para aplicaciones que no están ofuscadas ya que la propia limitación nos viene impuesta por Androguard. Aunque la aplicación esté ofuscada no se producen errores pero si que se verá reflejado en los ficheros generados apareciendo vacíos.

A pesar de que aquí se menciona únicamente su utilidad y su funcionamiento no se explica para qué sirve. En la sección 2.4.3 se comenta en mucho más detalle el funcionamiento de Frida.

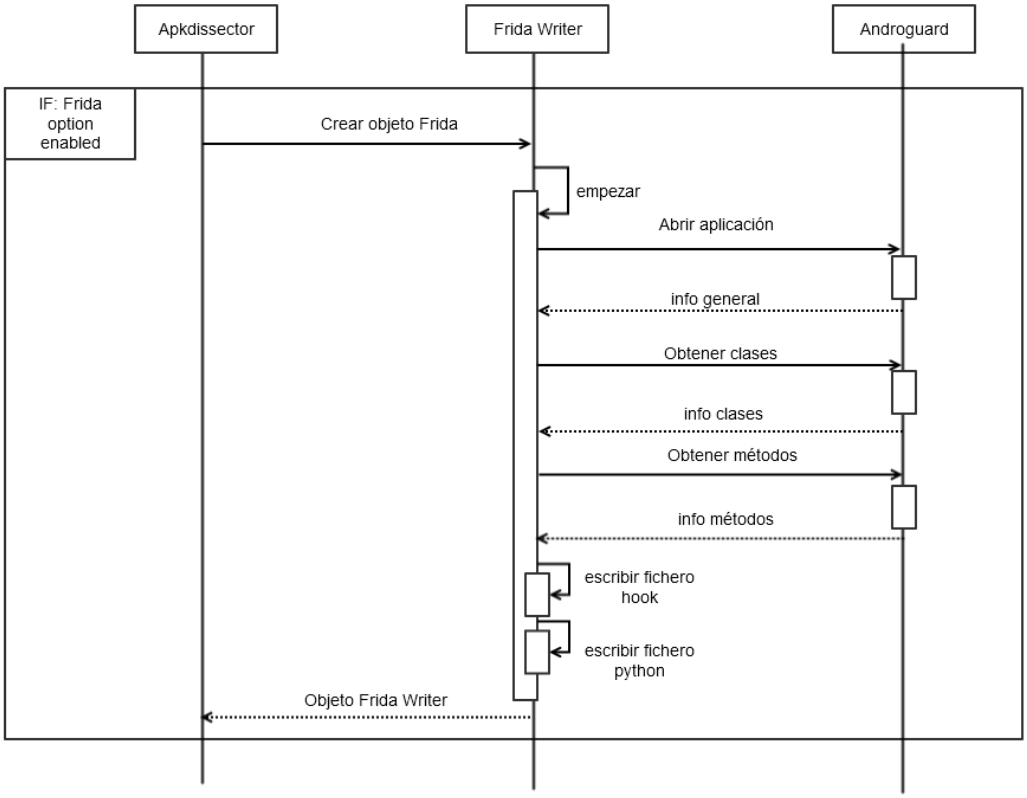

Figura 4.2: Diagrama de secuencia para la instrumentación dinámica de Frida

4.1.5. Downloads

Todos los componentes anteriores no serían útiles sin antes haber descargado una muestra de aplicaciones para realizar las pruebas por lo que se han incluido scripts independientes para la descarga de las mismas.

En la sección 2.2 se habla de las webs en las que se analizan aplicaciones pero que también permiten descargar aplicaciones si se tiene una clave para la API. Algunas APIs imponen un límite diario en el número de descargas.

Para mostrar estadísticas es necesario tener una muestra grande de aplicaciones por lo que se puso de objetivo descargar 10.000 aplicaciones de las tres webs comentadas previamente en la sección 2.2, Virustotal, Koodous y Andrototal, principalmente de las dos últimas. Finalmente, el número de aplicaciones obtenido ha sido menor que el inicial aunque sirve para dar una aproximación de los permisos más populares. Establecido el objetivo, se comenzó a desarrollar código en Python empleando las APIs que ofrecen las webs.

Para tener un mayor control de las aplicaciones descargadas, se gestionaron mediante una tabla en una base de datos SQLite (fácilmente accesible con Python). Al descar-

gar una aplicación, se realizará una inserción en la tabla *hashes* con el nombre de la aplicación, estableciendo que esa aplicación no está descargada junto con el nombre de la web donde se ha obtenido. Más tarde, además de emplearse para seguir descargando aplicaciones, se empleará en el cálculo de las estadísticas en la fase de análisis.

Observando la figura 4.3, se identifica la sucesión de acciones del código desarrollado en los *scripts* de descarga de aplicaciones. La estructura de los scripts, independientemente de la API a la que acceda, es siempre la misma: primero se obtienen N *hashes* sin descargar de la base de datos para la plataforma que se desee emplear, se itera sobre la lista de aplicaciones y se descargan una a una. Si se ha descargado correctamente se modificaría el elemento descargado en la base de datos estableciéndolo como descargado y, en caso de no descargarse, se mostrará un error y se pasará al siguiente de la lista. Se puede apreciar que después de cada aplicación se para la ejecución 10 segundos, tiempo establecido para no saturar al servidor con las peticiones generadas.

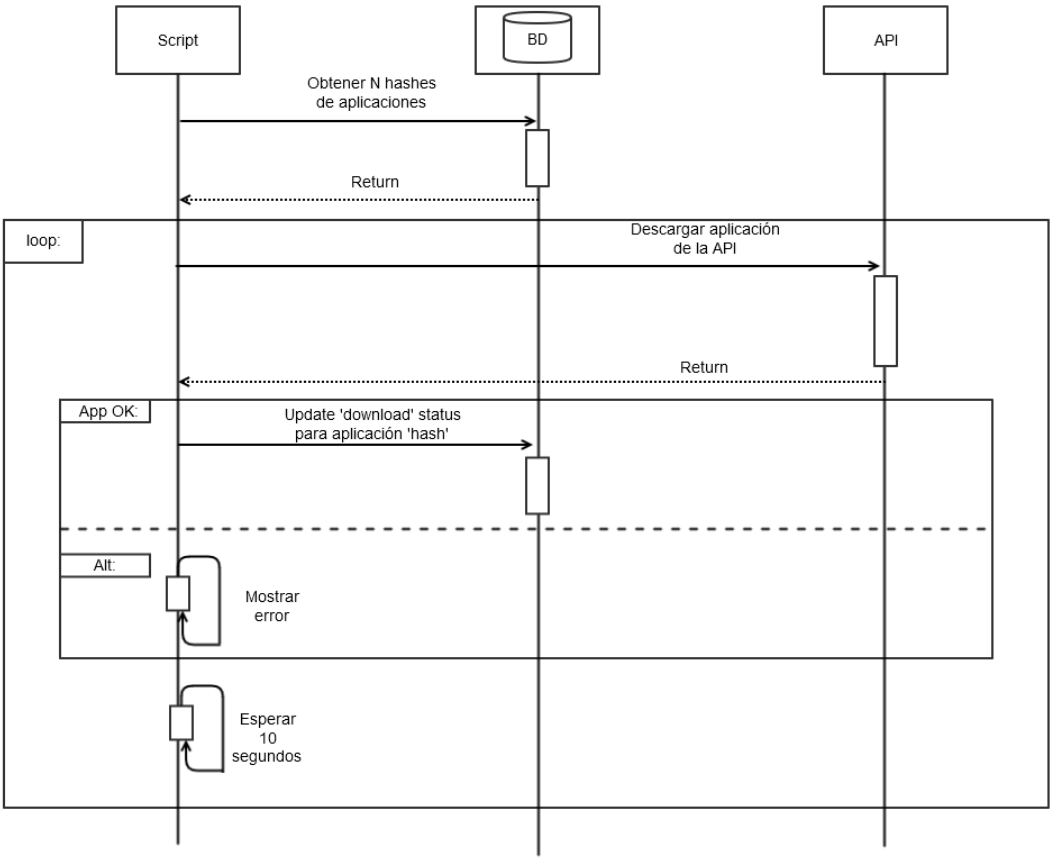

Figura 4.3: Diagrama de secuencia de los scripts de descarga

4.1.6. Otros

Una vez descritos los componentes principales es necesario integrarlos. Para ello, se ofrecen dos *scripts* principales para ejecutar la herramienta y, por otro lado, uno más secundario. Junto a ellos se ofrece además un fichero de configuración.

- **config.ini:** En un primer momento, la herramienta obtenía las aplicaciones de un directorio e iteraba de una en una obteniendo los permisos declarados en el Manifest. Teniendo en cuenta que se tenían que analizar un gran número de aplicaciones se consideró necesario establecer un fichero de configuración para indicar los directorios de trabajo y el número de threads a emplear, entre otras características. El fichero de configuración es totalmente adaptable a la máquina en la que se va a ejecutar y la finalidad principal fue reducir el número de parámetros de la herramienta. Tiene la siguiente estructura:

  ```
  [Configuration]
  OutputDirPath: /tmp/testing/results/
  PScoutVersion: 5.1.1
  ErrorLogPath: /tmp/testing/logs/
  DBPath: /tmp/testing/dbs/
  Threads: 5
  ```

  Como se puede apreciar, hay que establecer un directorio de trabajo (OutputDirPath), la versión de Pscout a emplear, un directorio para los ficheros de error que se produzcan al analizar una aplicación (ErrorLogPath), la ruta a un directorio con la base de datos (DBPath) o una ruta a una base de datos y, finalmente, el número de threads.

- **main.py:** El *script* principal de la herramienta. Recibe un único parámetro: un directorio con aplicaciones o una única aplicación. En el caso de tratarse de un fichero se realizará el análisis normal de la aplicación lanzando un *thread* y, por otro lado, si se trata de un directorio, se lanzarán tantos *threads* como se indique en el fichero de configuración.

  Además de gestionar el flujo de la herramienta, muestra también el tiempo total requerido para analizar tanto una aplicación como un directorio de aplicaciones. Junto al tiempo empleado se muestran las estadísticas de los permisos encontrados que se exportan también en un fichero JSON.

- **enforcementParser.py:** Segundo *script* más importante. No está relacionado directamente con la herramienta pero es tan importante como el descrito anteriormente. Es empleado para leer, corregir, poblar y realizar consultas a la base de datos generada durante el proceso de mapeo de permisos del capítulo 5.

- **pscoutToDB.py:** Este *script* se desarrolló en un principio para poder realizar consultas a los datos de PScout 3.3.1 de forma más sencilla. A grandes rasgos lo

único que hace es leer un fichero CSV y realizar distintas *queries* para poblar la base de datos con la información leída. Sirvió principalmente para familiarizarse con el lenguaje de programación y para generar una base de datos que se iría ampliando a lo largo del proyecto.

4.1.7. Optimizaciones

Durante el proceso de análisis de una aplicación, en un primer momento, se escribían los permisos en un fichero JSON almacenado en el directorio de trabajo declarado en el fichero de configuración. Al añadirle la información existente de un cierto permiso en la base de datos de Pscout, el fichero quedaba con una información muy interesante. Teniendo en cuenta el espacio que ocupa un fichero en memoria, si se analizan un gran número de aplicaciones el tamaño crecía enormemente con datos redundantes. Considerado el problema del espacio, se añadió la opción de emplear otra tabla dentro de la base de datos para reducirlo.

La opción de almacenar la información en una tabla fue simple ya que se contaba con una tabla con hashes de aplicaciones y otra tabla con la información de los permisos de Pscout. Por tanto, solo fue necesario introducir una tabla intermedia entre las dos citadas anteriormente permitiendo tener almacenados los permisos que tiene una aplicación analizada.

De esta forma, la base de datos completa tiene un diagrama entidad-relación igual al que se muestra en la figura 4.4:

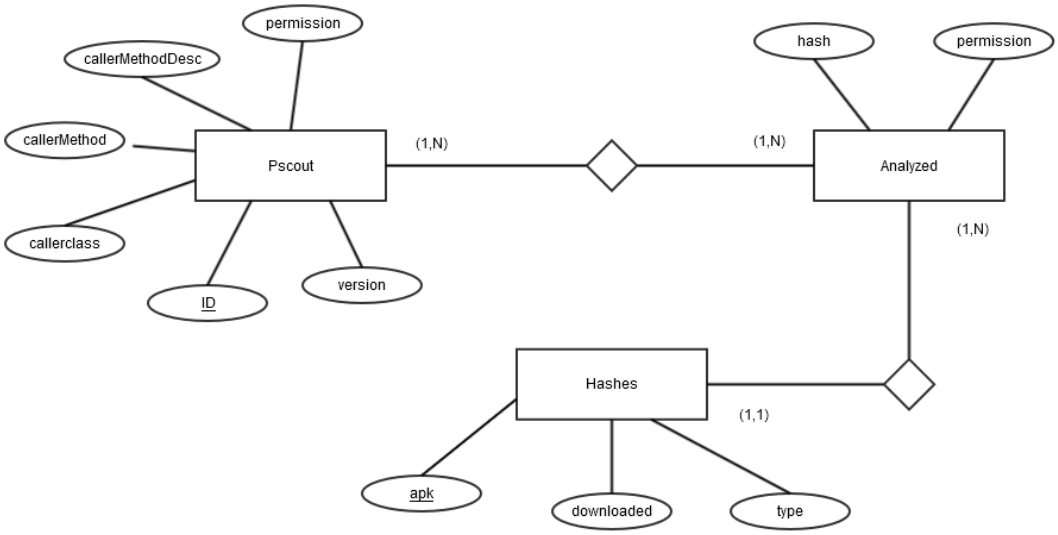

Figura 4.4: Diagrama Entidad-Relación de la base de datos

Comparando ambas aproximaciones, el empleo de ficheros JSON o la base de datos, se puede comprobar que realmente la base de datos es mucho más eficiente en tiempo

y espacio. En espacio porque ocupa 20MB aproximadamente mientras que, empleando ficheros JSON y analizando tan solo 800 aplicaciones, el espacio aumenta hasta los 38GB. Con forme se fue avanzando en el proyecto más aplicaciones se iban analizando y, por ello, el espacio en disco que se necesitaba. También es más eficiente en tiempo, ya que en vez de repetir escrituras de líneas idénticas, cada thread crea una conexión para evitar problemas de concurrencia, inserta tantas filas como permisos tenga una aplicación y se cierra la conexión.

En el Anexo E se muestran una serie de consultas realizadas a la base de datos del analizador para mostrar la información que almacena.

Capítulo 5

Mapeo de permisos

Los permisos son la primera defensa que protege la información personal de un *smartphone* de una aplicación de un desarrollador desconocido. Por ello, resulta de interés conocer qué clases y métodos requieren un permiso para así poder establecer qué información podría quedar accesible.

Con la limitación que impone el permiso DUMP en el mapeo de permisos del proyecto PScout, sección 3.3.1, se ha realizado una nueva aproximación para obtener un mapeo de permisos. A pesar de que los resultados obtenidos se van a tratar en el capítulo 6, se han obtenido resultados útiles para el permiso BLUETOOTH y BLUETOOTH_ADMIN. Estos resultados se van a emplear para la realización de una Prueba de Concepto (POC) en la sección 6.4.1 y así lo que se podría conseguir con un mapeo de permisos completo y las herramientas adecuadas.

5.1. Forzado de permisos

Teniendo en cuenta que se quieren obtener los permisos y los métodos vinculados, se debe conocer en un primer lugar dónde se fuerzan esos permisos, es decir, el lugar en el que Android verifica que una determinada aplicación posee un cierto permiso.

Para ello, una de las formas más factibles de comprobar el flujo que siguen las invocaciones dentro del código de Android es acudir a AndroidXRef [Chi]. AndroidXRef permite seleccionar la versión de Android y comprobar el código que lo forma. Además, permite enlazar invocaciones a otros elementos, direccionando al fichero del mismo. Debido al interés de conocer en qué punto se verifican los permisos, es una herramienta muy útil.

Tomando como ejemplo la clase *BluetoothAdapter.java*, encargada de dar funcionalidad al Bluetooth en Android, se va a analizar su funcionamiento para demostrar la comprobación del permiso *android.permission.Bluetooth* dentro de la aplicación.

En un primer lugar, necesitamos obtener el *adapter* del Bluetooth. Para ello, en la siguiente sección de código se muestra cómo se obtendría.

```
BluetoothAdapter BA = BluetoothAdapter.getDefaultAdapter();
```

En la sección de código inferior se muestra en qué consiste la llamada anterior. El método *static* va a comprobar si ya existe un adaptador y, en caso de que no exista, se realiza una petición al Binder (línea 4), encargado de obtener mediante el *ServiceManager* una referencia al servicio requerido, conocido como *stub* (línea 7). Los *stub*s en Android son referencias a servicio obtenidos a través del Binder y que están declarados mediante un fichero AIDL en el framework de Android. El uso del Binder es una barrera ya que cuando se emplea, en este caso para obtener un stub de *IBluetoothManager*, no se puede conocer la interacción que sigue en ese proceso. Por ello, no se puede continuar por aquí por falta de información.

```
1  public static synchronized BluetoothAdapter getDefaultAdapter()
2  {
3      if (sAdapter == null) {
4          IBinder b = ServiceManager.
5              getService(BLUETOOTH_MANAGER_SERVICE);
6          if (b != null) {
7              IBluetoothManager managerService =
8                  IBluetoothManager.Stub.asInterface(b);
9              sAdapter = new BluetoothAdapter(managerService);
10         }
11         [..]
12 }
```

Tras analizar aplicaciones, se observa que toda aplicación necesita un *Context*, encargado de acceder a recursos específicos y clases además de iniciar *Activities*, *Broadcasting* o recibir *Intents*. Analizando el código en AndroidXRef de la clase *ContentImpl.java* [Gooa] y la clase *Context.java* [Gooc] se puede encontrar los métodos *enforcePermission*, *enforceSelfPermission* o *enforceCallingOrSelfPermission* que reciben un parámetro que contiene el permiso en un String. Dichos métodos se emplean para lanzar una *SecurityException* en caso de que el permiso necesario no esté declarado en el Manifest. Teniendo una referencia donde conocer el nombre del permiso y, sabiendo que Java cuenta con la posibilidad de acceder a la pila de ejecución, se puede vincular un determinado permiso a un *stack*.

El *stack* en Java se obtendría mediante la invocación al método *getStackTrace()* del Thread que está en ejecución. De esta forma, se genera una lista de *StackTraceElement* que permite obtener el nombre de la clase, el nombre del fichero y el nombre del método. Así, a pesar de que se produzca cierto "ruido" en el *stack*, aparecerá el nombre de la función vinculada al permiso con el que se está lanzando la excepción.

5.2. Modificación de Android Open Source Project

Una vez pensado el sistema con el que obtener el mapeo se tiene que introducir la modificación de la clase *ContextImpl.java* en el framework, empaquetar el framework modificado e introducirlo en un dispositivo.

Inicialmente, el flujo de ejecución de la verificación de un permiso dentro de la clase *ContextImpl.java* sería como aparece en la figura 5.1. Si se cumple con el permiso requerido, la ejecución continuaría. En caso contrario, se cerraría la aplicación con un mensaje de error.

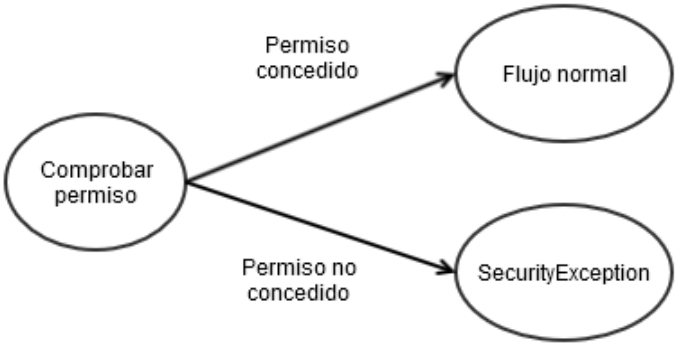

Figura 5.1: Flujo de la clase ContextImpl.java

La forma más sencilla de reproducir la información del analizador dinámico que se ha planteado es la escritura de un fichero por cada método que lanza la *SecurityException* (*enforcePermission*, *enforceSelfPermission* o *enforceCallingOrSelfPermission*). Para ello, lo que se propuso en un primer momento fue emplear notación JSON. Así, cuando se acabase el test, se podría acceder al dispositivo mediante Android Debug Bridge (ADB), una herramienta incluida en el Software Development Kit (SDK) de Android y que permite acceder a una *shell* del dispositivo.

Con la modificación, se añadió un paso intermedio que escribiría el *stack* en el fichero JSON con la información como se observa en la figura 5.2. Una consideración importante es que los métodos elegidos iban a acceder a un recurso compartido y, al añadirlos a una de las clases más importantes de las aplicaciones Android, se debían tener en cuenta los problemas de concurrencia. De esta forma, los métodos añadidos en ese paso intermedio debían ser *synchronized*.

A pesar de tratarse de una escritura en un fichero, SELinux[SEL] no lo iba a permitir debido al uso de *sandboxes* y a que se estaba escribiendo en una ubicación fuera del *sandbox* de la aplicación. Esto producía errores del CTS en tiempo de ejecución. Para hacer posible la escritura, SELinux debe estar en modo *permissive*. Así, se tuvo que dotar al AOSP modificado con un kernel que tuviese SELinux en este modo.

Al crear los ficheros se comprobó que sin ejecutar nada ya contenían información por lo que se accedió al dispositivo para extraerlos y analizarlos. Se concluyó que la estructura del fichero JSON añadida en la modificación realizada cumplía su labor y estaba preparada para funcionar con el CTS.

En cuanto a la estructura de los ficheros JSON, debido a la necesidad de ejecutar el CTS, esos ficheros JSON no se iban a cerrar hasta que acabase el test, es decir, la estructura no iba a estar bien formada y, una vez extraídos los ficheros, se tuvo que

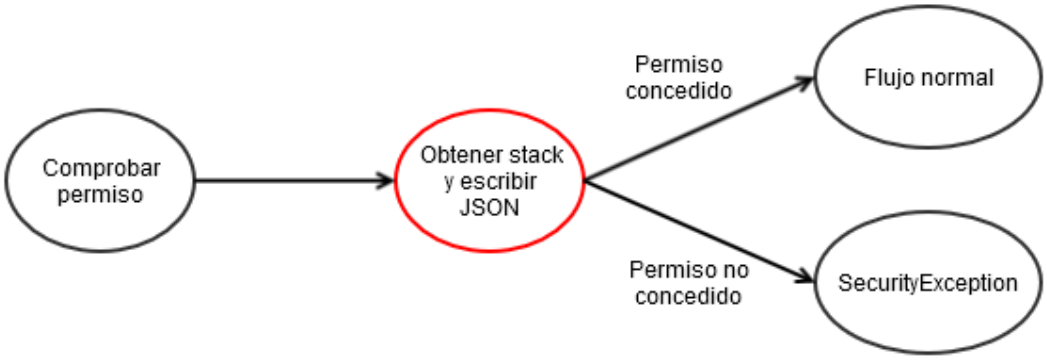

Figura 5.2: Flujo de la clase ContextImpl.java modificada

añadir una nueva utilidad al módulo *Enforcement* de la herramienta Apkdissector 4.1 para borrar la última coma e introducir las llaves necesarias para corregir dicho fichero.

Una vez confirmado que realmente la aproximación elegida para el mapeo de permisos estaba operativa y que no producía errores, se realizó el CTS.

5.3. Ejecución del Compatibility Test Suite

Debido a que la modificación descrita previamente depende de la ejecución de la API de Android, es necesario emplear algo que haga pruebas sobre todo el sistema. En la sección 2.3.3 se habló de la herramienta que ofrece Google a los fabricantes de dispositivos para que prueben si su modificación del AOSP es compatible. Por ello, en este intento de conseguir una mejora en los resultados de Pscout, se va a aprovechar dicha herramienta.

La modificación realizada se encuentra en una de las clases más importantes de una aplicación Android y, por tanto, los resultados que se puedan obtener deberían ser favorables ya que el CTS debe realizar una prueba de todo el sistema lo que implica que también tiene que comprobar que las excepciones funcionan correctamente y, en caso de probar esas excepciones, se obtendrá el fichero JSON que se genera.

Para ejecutar el CTS es necesario disponer de un dispositivo, en este caso se realizó con un Nexus 6 ofrecido por la Università degli Studi di Torino. En un primer lugar es necesario configurar tanto el entorno de trabajo, es decir, la *workstation*, y por otro lado, el dispositivo en el que se va a ejecutar el test [Goof]. En segundo lugar, se debe ejecutar el test una vez cumplidos todos los requisitos. Es muy importante que durante la ejecución del test no se emplee el dispositivo ya que se interferirá en la ejecución pudiendo provocar que los resultados no sean correctos o se produzcan errores y, por consiguiente, el test no termine.

Para aprovechar mejor las horas de trabajo, el CTS se dejaba en ejecución por la noche y se comprobaban los resultados al día siguiente. Realizando una estimación, ese tiempo podría estar entre las cuatro y las ocho horas aproximadamente.

Capítulo 6

Resultados obtenidos

En este capítulo se van a tratar los resultados obtenidos a lo largo de las distintas fases que forman el proyecto. En un primer lugar se analizarán los resultados obtenidos por la herramienta Apkdissector 4.1. Después, se comentarán los resultados obtenidos tras la modificación del AOSP, es decir, el nuevo mapeo de permisos obtenido con la comprobación del *stack*. Con ello, se incluirá una explicación del proceso de validación de los resultados y se comentará la utilidad de los mismos.

6.1. Estadísticas de permisos

Haciendo uso de Apkdissector, se analizó una muestra de 2064 aplicaciones descargadas de las webs comentadas en la sección 2.2. Para la obtención de las estadísticas se desarrolló otro *script* en Python. Ese script permite calcular las estadísticas a través de un directorio con aplicaciones analizadas, es decir, cada aplicación cuenta con un directorio con ficheros JSON con el mapeo de Pscout o, la opción más ligera en espacio, la consulta a la tabla *analyzed* de la base de datos. Además, se genera un fichero JSON con el contenido de las estadísticas obtenidas en caso de necesitar recuperarlas en un futuro.

Las estadísticas obtenidas se observan en la figura 6.1. Antes de comprobar los permisos se podían imaginar los resultados ya que el permiso de INTERNET debería ser el más empleado. Otra aproximación antes de realizar las estadísticas era el permiso de SEND_SMS ya que, al bajarse las aplicaciones de webs de análisis y pensando que podrían ser malware, este permiso está vinculado con aplicaciones de subscripción a servicios *premium*. Finalmente, se observa que ese permiso no está incluido en los diez más encontrados por lo que hay muchas posibilidades de que la muestra examinada no esté formada íntegramente por malware.

En la gráfica se muestran únicamente los diez permisos más comunes de la muestra analizada. En el anexo F se muestra parte del fichero JSON generado organizado en una tabla.

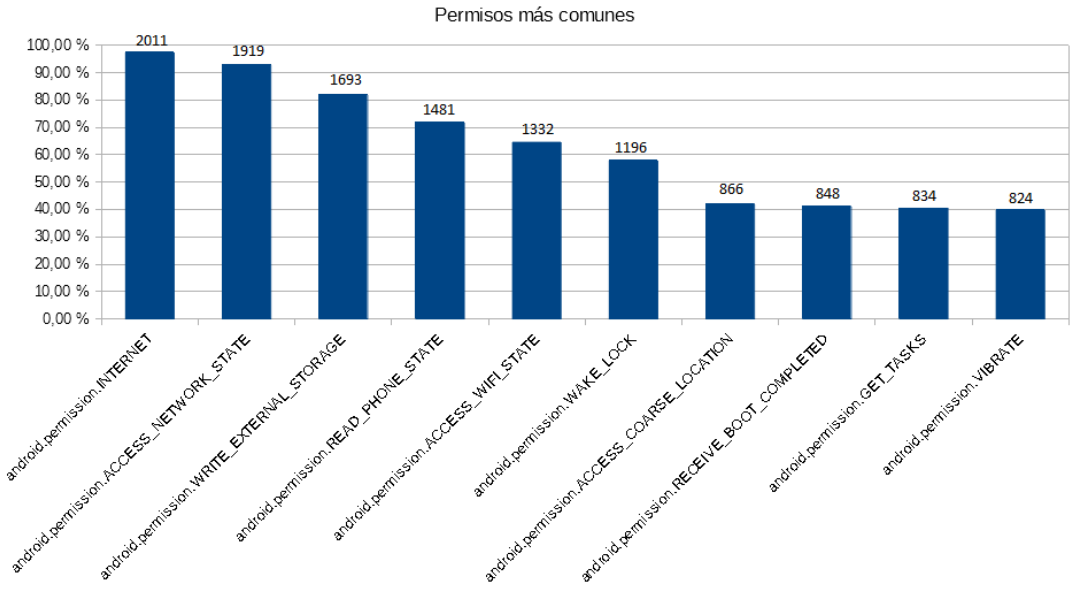

Figura 6.1: Estadísticas de permisos para una muestra de 2064 aplicaciones

6.2. Resultados del mapeo de permisos

Al finalizar el CTS y retirados los ficheros del dispositivo, se realizaron unas primeras comprobaciones a mano a través de la consola para comprobar qué permisos se habían obtenido y si realmente contenían métodos interesantes que en Pscout aparecen mapeados con el permiso DUMP. Estas primeras comprobaciones tenían como objetivo verificar si realmente se había optado por el buen camino durante el proyecto.

Para comprobar los resultados de una forma más eficiente y preparar esos resultados para el proceso de validación se elaboró otra utilidad para Apkdissector, el módulo *Enforcement*, que permitiría crear una base de datos a través de esos ficheros JSON retirados del dispositivo, agilizando el proceso de validación. Es una base de datos muy simple que se prefirió separar de la diseñada anteriormente y cuyo modelo entidad-relación se muestra en la figura 6.2. Así, los resultados obtenidos pueden ser evaluados mediante la confrontación con la base de datos del proyecto Pscout.

Debido a que los ficheros JSON pueden contener información redundante provocado por la escritura continua sobre ellos por parte de distintos componentes se establecieron unas medidas para limitar la cantidad de "ruido" que contendría la base de datos. Así pues, se eligieron los componentes de interés de un determinado *stack* mapeado con su permiso, concatenar toda la información y generar un *hash*. De esta forma, si el *hash* ya estaba en la base de datos se evitaba la redundancia por las propiedades de las funciones *hash*. Las colisiones no se han considerado ya que el tipo de *hash* empleado no está establecido como problemático en este aspecto.

Una vez obtenidos los datos y teniendo en cuenta el objetivo inicial, es decir, la unión

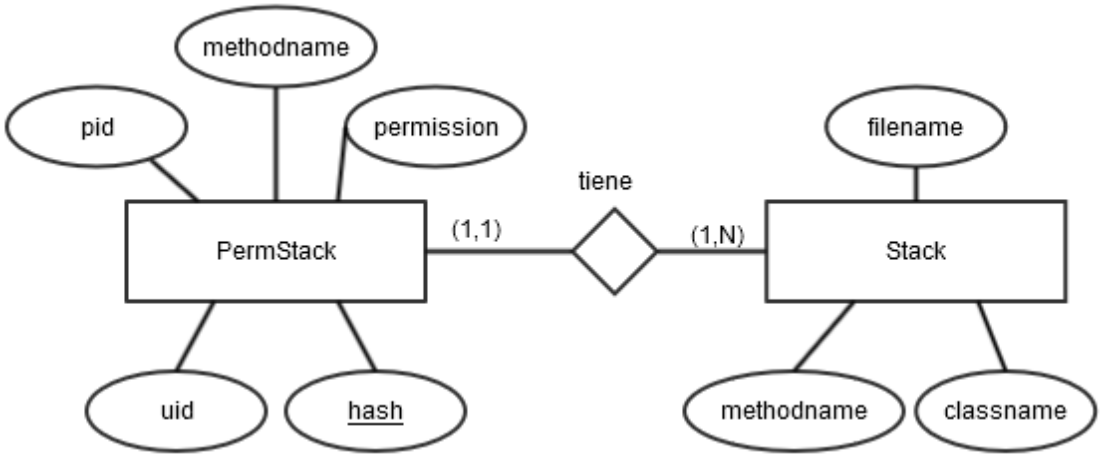

Figura 6.2: Modelo entidad-relación obtenido con el analizador dinámico

de dos herramientas de análisis con unos datos necesarios para su correcta integración, ya se puede proceder a validar los resultados.

6.3. Validación de resultados

Tras las comprobaciones manuales realizadas una vez descargados los ficheros del dispositivo, los resultados no parecían ser buenos ya que existía información relevante pero, por otro lado, los permisos obtenidos son ligeramente inferiores a los que contiene Pscout. Analizando bien la metodología de Pscout y la empleada en este proyecto era normal que se tuviesen menos permisos. Esto se debe a que PScout tenía en cuenta también los permisos relacionados con *Intents* y *ContentProviders*. Así, los permisos existentes en la base de datos producida tras analizar los ficheros eran 63 mientras que en el proyecto de Pscout aparecen 97.

La comparación de ambos proyectos, además de identificar si se han obtenido mejores resultados, sirve también para verificar la validez de los nuevos resultados. Por ello, tras confrontar los resultados con el *script* descrito anteriormente, no se obtenían buenos resultados para los permisos INTERNET o READ_PHONE_STATE que eran los permisos empleados en las pruebas. Finalmente, se encontraron unos buenos resultados para los permisos BLUETOOTH y BLUETOOTH_ADMIN.

El módulo *Enforcement* ofrece una funcionalidad diversa adecuándose a la etapa en la que se encontraba el proyecto y para permitir la generación de la base de datos en caso de perdida (siempre que se tuviesen los ficheros JSON). A continuación se enumeran y describen las tareas principales:

- *Leer JSON y crear base de datos:* Este caso se podría considerar como el paso inicial. Tal y como se explicó en la sección 5.2, el fichero generado no está correctamente formado. Esta opción corregiría el fichero, leería su información y la

insertaría en una base de datos SQLite.

- *Comprobar un permiso en la base de datos:* Para un uso futuro se permite obtener la información relacionada a un cierto permiso. En el caso de este proyecto solo se emplea para realizar sencillas comprobaciones pero se podría integrar en una herramienta de análisis automático.

- *Confrontar un permiso en ambas bases de datos:* Para validar los resultados obtenidos durante el proceso de análisis y una vez generada la base de datos se puede proceder a la comparación de resultados. El funcionamiento de la herramienta se basa principalmente en una comparación de la clase y el método de dos elementos, uno de Pscout y otro de este proyecto.

  Debido a que los resultados de Pscout son mucha más favorables, la información principal está obtenida de dicho proyecto. De esta forma, el permiso también dependerá de Pscout aunque, a pesar de esto, al existir una coincidencia de nombre de clase y nombre de método en la búsqueda de un determinado permiso, se premiará el mapeo obtenido de los ficheros JSON.

La parte más importante de la herramienta sería la comparación de los resultados. A pesar de haber eliminado los resultados redundantes sigue habiendo resultados que no se consideran de utilidad ya que son funciones internas de Android como pueden ser los métodos *loop()* de la clase *Looper* o *run()* de un *Thread*. A pesar de ello, si aparece una coincidencia es que Pscout también tiene esa información en sus resultados por lo que no se consideró relevante su eliminación.

A continuación se van a mostrar resultados obtenidos en consultas a distintos permisos. De esta forma, se quiere mostrar la información de Pscout para un permiso y demostrar cómo se encuentra en la base de datos de este proyecto. Al principio de la ejecución se muestra si se han encontrado permisos disponibles en este proyecto que no están en Pscout. Si se comprueba para el permiso BLUETOOTH se obtiene:

```
[*] 0 rows found with android.permission.BLUETOOTH in PScoutDB
[*] 32 rows found with android.permission.BLUETOOTH in JsonDB
```

Al parecer, el proyecto obtiene información para el permiso BLUETOOTH que Pscout no tiene lo que se debe a que los métodos relacionados con ese permiso están mapeados con el permiso DUMP. Estos resultados transmiten buenas sensaciones de cara a los resultados finales del proyecto.

Por otro lado, se van a analizar los resultados obtenidos y que se consideran relevantes, evitando los mapeos de aquellas funciones "ruidosas". Para la consulta al permiso BLUETOOTH realizada anteriormente se obtiene lo siguiente:

```
[..]
Methodname: getState
Classname: android/bluetooth/BluetoothAdapter
Filename: BluetoothAdapter.java
```

```
Signature: ()I
Permission: android.permission.DUMP

Methodname: isEnabled
Classname: android/bluetooth/BluetoothAdapter
Filename: BluetoothAdapter.java
Signature: ()Z
Permission: android.permission.DUMP
[..]
```

Los métodos *getState()* e *isEnabled()* pertenecen a la clase *BluetoothAdapter*. Esta clase es la empleada por el desarrollador para acceder al Bluetooth del dispositivo. En la muestra anterior se observa que aparecen con el permiso DUMP debido a que se priorizan los resultados de Pscout pero la búsqueda se realiza sobre el permiso BLUETOOTH por lo que se consideran mapeadas correctamente.

Por otro lado, uno de los permisos más importantes a la hora de analizar malware es el permiso SEND_SMS. Si se comprueba ese permiso con la herramienta no se obtiene ningún resultado. Esto se debe a que ni Pscout ni este proyecto tienen un mapeo correcto para ese permiso. En el caso de Pscout, el método *sendTextMessage()* está mapeado con el permiso DUMP y este proyecto no tiene información relacionado con ese permiso. Pscout mantiene su limitación pero, en el caso de este proyecto y debido a que se depende de las pruebas que se realizan en el CTS se llega a la conclusión de que la inexistencia de este permiso se deba a dos posibilidades: el CTS no prueba este aspecto, bastante improbable, o que el dispositivo en el que se realizó el test no tuviese una tarjeta Subscriber Identity Module (SIM).

De esta forma, a pesar de no obtener los mejores resultados a los que se podía aspirar, los datos elaborados con el analizador dinámico de permisos pueden servir para una prueba de concepto siguiendo la línea de integración de resultados de la que se partió desde un principio y que pretendía emplear la herramienta ARTDroid, comentada en la sección 2.4.3. Finalmente, no se ha podido emplear este proyecto por no estar disponible aunque, a pesar de ello, se encontró otra herramienta similar y que permite llegar al mismo punto. Esta herramienta es Frida que ha sido comentada en la sección 2.4.3.

6.4. Utilidad de los resultados obtenidos

Debido a los problemas encontrados durante el proyecto, la necesidad de buscar *workarounds* y el tiempo limitado se han obtenido unos resultados útiles pero no completos teniendo en cuenta la limitación impuesta por Pscout. Desde el inicio del proyecto lo que se pretendía obtener fue un mapeo que permitiese instrumentar herramientas de análisis de aplicaciones de forma dinámica. Así, se ha podido preparar una prueba de concepto con los resultados útiles obtenidos y con otros obtenidos para satisfacer el objetivo final de la prueba.

6.4.1. Prueba de Concepto

Para mostrar lo que se podría realizar con un buen mapeo de permisos, como se pueden emplear los resultados obtenidos y lo que permiten realizar herramientas de instrumentación dinámica como ARTDroid o Frida se han preparado diferentes pruebas con una aplicación Android sencilla pero que serían perfectamente útiles con aplicaciones disponibles en el mercado.

Como se comentó anteriormente en la sección 2.3.2, existe una limitación impuesta por la ofuscación del código ya que Androguard no es capaz de obtener la información necesaria. A pesar de ello, se han publicado herramientas capaces de recuperar una aplicación sin ofuscar partiendo de una ofuscada por lo que, en un futuro, se podría continuar con la incorporación de estas utilidades. Al realizar la prueba se ha conseguido aplicarla tanto en Android Virtual Device (AVD) como en un dispositivo real rooteado lo que es importante de cara al resultado final.

Con la facilidad que ofrece Frida para interceptar invocaciones y el análisis rápido que permite Androguard se ha desarrollado una utilidad que se ha incluido en la herramienta Apkdissector(4.1). Con ella, es posible analizar una aplicación sin ofuscar, obtener sus clases junto a sus métodos y, con esta información, generar los ficheros JavaScript y Python para alterar sus métodos. Por otro lado, en el caso de tener un buen mapeo de permisos, sería posible aplicar lo mismo a las propias clases que ofrece Android.

Para demostrar el funcionamiento de las herramientas de este tipo, se han realizado varios ejemplos con objetivos diferentes. En un primer lugar se ha tratado de interceptar las invocaciones a métodos propios de Android, como *getDeviceId()* del *TelephonyManager* o *isEnabled* del *BluetoothManager*. Se ha querido emplear el método *isEnabled* del Bluetooth ya que aparece correctamente mapeado como se comenta en el capítulo 5 y relacionar de forma directa este proyecto con Frida.

Por otro lado, se ha realizado algo similar con métodos de las propias clases de la aplicación como la modificación de una URL o el bloqueo por código PIN necesario para acceder a la información de la aplicación.

Debido al tiempo limitado del que se disponía no se ha podido profundizar demasiado en el código dinámico de instrumentación para Frida por lo que no se modifica el código de cada método. Lo único que se realiza es enviar un mensaje desde el servidor incluido en el dispositivo hacia el cliente, es decir, el fichero Python, con el fin de notificar el *hook* de un método.

Las pruebas anteriores tratan de demostrar y concienciar sobre que el flujo de ejecución de una aplicación no siempre tiene que ser el considerado por el desarrollador. Por ejemplo, si una aplicación cuenta con los permisos de INTERNET y SEND_SMS se podría interceptar la llamada a cualquier método y subscribir al usuario a un servicio premium o descargar cualquier tipo de *malware* en su dispositivo. Este es el motivo que lleva a la necesidad de obtener un buen mapeo de permisos y, con ello, elaborar herramientas de análisis que puedan predecir si una aplicación podría suponer un riesgo para el usuario. Con el motivo de interceptar llamadas a la librería de Android, el mapeo de permisos sería muy útil ya que no sería necesario analizar previamente la aplicación sino que se podría instrumentar directamente sobre las clases del propio sistema.

En el anexo G se muestra el código necesario para modificar el comportamiento de los métodos junto al de la aplicación Android.

Capítulo 7

Conclusiones y trabajo futuro

Este proyecto lo empecé durante mi estancia Erasmus en Turín, Italia. Por motivos relacionados con la convalidación de asignaturas se me quedó un Erasmus más relajado y decidí aventurarme en este proyecto. Gracias a esa decisión sigo manteniendo buena relación con mis profesores de allí.

Durante el transcurso del proyecto se consultaron muchos trabajos interesantes. Destaca Pscout con el que llegamos a comentar con los desarrolladores. Al emplear trabajo ajeno en un proyecto se imponen sus limitaciones y, en este caso, al emplearse varios, el proyecto cuenta con dos fundamentales: la limitación del permiso DUMP y el empleo de Frida o ARTDroid que impone la necesidad de emplear un dispositivo root. A pesar de ello, es posible emplear el proyecto tanto en dispositivos reales como en virtuales.

Tras el análisis de Pscout no se tenían los resultados óptimos para integrar con ARTDroid por lo que el proyecto cambió radicalmente. Del desarrollo de una herramienta automática de análisis se pasó al mapeo de permisos de Android modificando totalmente los conocimientos necesarios para llevar a cabo la investigación deseada ya que, a pesar de que Android es *open source*, el Binder añade una capa más oscura. El director del proyecto me facilitó un gran número de referencias junto con libros, sobre todo en las fases iniciales del proyecto.

Teniendo en cuenta que la necesidad de mapeo de permisos en el análisis de aplicaciones Android está presente desde las primeras versiones del sistema operativo de Google, es muy posible que existan más personas intentando obtener una respuesta al permiso DUMP de Pscout y, seguramente, existan también muchos proyectos que no han conseguido unos resultados completos, como le ocurre a este. A pesar de no haber conseguido obtener unos buenos resultados si que se han obtenido fragmentos útiles que se han podido integrar en la herramienta Frida por lo que el objetivo inicial se ha cumplido en algunos casos.

Este proyecto me ha servido para adentrarme en el mundo de la investigación y comprobar que no es nada fácil. En mi opinión, los días más complicados son los de bloqueo o los de comprobar resultados. Tener que despreciar datos por no ser completos y dejar de lado un idea en la que llevas días trabajando. Por ello, me gustaría reconocer

la labor de los investigadores de todas las universidades y de cualquier materia.

Para trabajo futuro se podría dedicar otro proyecto más. El propio sistema de Android y sus versiones provoca que unos trabajos realizados para una determinada versión, si Google modifica enormemente su sistema, ocasiona que con el paso de los años se queden obsoletos, como es el caso de Stowaway. Por otro lado, muchos de los proyectos desarrollados para este sistema operativo tienen limitaciones: la necesidad de root, no considerar Java Reflection, no conseguir un mapeo de permisos íntegro o problemas al elaborar CFG en según qué llamadas. Por ello, una línea de trabajo futuro podría ser la mejora de todos estos aspectos en nuevos proyectos que englobasen los anteriores.

Un trabajo próximo podría ser comprobar la limitación sobre el permiso SEND_SMS ejecutando otra vez la modificación realizada por el AOSP, obteniendo la base de datos y confrontando los resultados con Pscout con Apkdissector y su módulo *Enforcement* pero, en este caso, asegurando disponer de una tarjeta SIM en el dispositivo. También se podrían buscar nuevas líneas de investigación para comprobar a qué se debe el tener información incompleta y si sería posible obtener una mejora continuando por la línea seguida.

Por otro lado, se podría trabajar en la integración de una buena herramienta de CFG en Apkdissector para ofrecer así un analizador estático o abrir nuevas líneas de investigación relacionadas con analizadores dinámicos.

Bibliografía

[AZHL] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: Analyzing the android permission specification. Technical report, Dept. of Electrical and Computer Engineering, University of Toronto, Canada, 27 King's College Cir, Toronto, ON M5S, Canadá. http://www.eecg.toronto.edu/%7Elie/papers/PScout-CCS2012-web.pdf.

[BGH+] Michael Backes, Sebastian Gerling, Christian Hammer, Matteo Maffei, and Philipp von Styp-Rekowsky. Appguard - enforcing user requirements on android apps. Technical report, Saarlan University, Saarbrücken, Germany and Max Planck Institute for Software Systems, 66123 Sarrebruck, Alemania. http://sps.cs.uni-saarland.de/publications/tacas2013.pdf.

[CBC16] CBCnews. Instagram glitch discovery earns $10k facebook bug bounty for boy, 2016. http://www.cbc.ca/news/technology/instagram-bug-bounty-1.3565981.

[Chi] Rodrigo Chiossi. Androidxref. http://androidxref.com/.

[Cop] Copperhead. Copperhead, a hardened open-source operating system based on android. https://copperhead.co/android/.

[Cya] CyanogenCommunity. Cyanogenmod, a customized aftermarket firmware distribution for android devices. https://www.cyanogenmod.org/.

[CZ] Valerio Costamagna and Cong Zheng. Artdroid: A virtual-method hooking framework on android art runtime. Technical report. http://ceur-ws.org/Vol-1575/paper_10.pdf.

[DG] Anthony Desnos and Geoffroy Gueguen. Reverse engineering, malware and goodware analysis of android applications. https://github.com/androguard/androguard.

[Erl] Ulfar Erlingsson. The inlined reference monitor approach to security policy enforcement. Technical report, Cornell University, Ithaca, Nueva York 14850, EE. UU. http://kennarar.ru.is/ulfar/thesis.pdf.

[Gol] Robert Gold. Control flow graphs and code coverage. Technical report, Ingolstadt University of Applied Sciences, Esplanade 10, D-85049 Ingolstadt, Germany. http://matwbn.icm.edu.pl/ksiazki/amc/amc20/amc20411.pdf.

[Gooa] Google. Contextimpl.java. https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/app/ContextImpl.java.

[Goob] Google. Cts, a free commercial-grade test suite. https://source.android.com/compatibility/cts/.

[Gooc] Google. Google developers context class. https://developer.android.com/reference/android/content/Context.html.

[Good] Google. Proguard. https://developer.android.com/studio/build/shrink-code.html.

[Gooe] Google. Running cts tests. https://source.android.com/compatibility/cts/run.html.

[Goof] Google. Setting up cts. https://source.android.com/compatibility/cts/setup.html.

[Goog] Google. Software stack for a wide range of mobile devices. https://source.android.com/.

[Gooh] Android Developers Google. Basic ipc primitive for android. https://developer.android.com/reference/android/os/Binder.html.

[Gua] GuardSquare. Dexguard. https://www.guardsquare.com/en/dexguard.

[HUHS] Johannes Hoffman, Martin Ussath, Thorsten Holz, and Michael Spreitzenbarth. Slicing droids: Program slicing for smali code. Technical report, Ruhr-University Bochum and Friedrich-Alexander-University. https://www.ei.rub.de/media/emma/veroeffentlichungen/2013/04/16/slicing_droids-sac13.pdf.

[KK07] Chad McMillan Kris Kendall. Practical malware analysis, 2007. https://www.blackhat.com/presentations/bh-dc-07/Kendall_McMillan/Presentation/bh-dc-07-Kendall_McMillan.pdf.

[mun16] El mundo. El 92 % de los españoles tiene al menos un smartphone, 2016. http://www.elmundo.es/tecnologia/2016/11/07/582087f0e2704e905c8b45c8.html.

[MVZ13] Federico Maggi, Andrea Valdi, and Stefano Zanero. Andrototal: A flexible, scalable toolbox and service for testing mobile malware detectors. In *Proceedings of the 3rd Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*. ACM, November 2013. https://andrototal.org/.

[Ora] Oracle. The reflection api. https://docs.oracle.com/javase/tutorial/reflect/.

[Rav] Ole André V. Ravnås. Frida: a dynamic code instrumentation toolkit. http://www.frida.re/.

[RLVS] Fernando Ramírez, Francisco López, Daniel Vaca, and Antonio Sánchez. Koodous, a collaborative platform to combine analysis with social interactions. https://koodous.com/.

[SEL] SELinux. Security-enhanced linux in android. https://source.android.com/security/selinux/.

[Sis04] Hispasec Sistemas. Virustotal, 2004. https://www.virustotal.com.

[Sta] Statista. Statistics and facts about smartphones. https://www.statista.com/topics/840/smartphones/.

[Wika] Wikipedia. Android application package. https://en.wikipedia.org/wiki/Android_application_package.

[Wikb] Wikipedia. Software obfuscation. https://en.wikipedia.org/wiki/Obfuscation_(software).

[Wi Ryszard Wiśniewski. A tool for reverse engineering android apk files. https://ibotpeaches.github.io/Apktool/.

[Xat14] Xataka. Y el primer smartphone de la historia fue..., 2014. http://www.xatakamovil.com/movil-y-sociedad/y-el-primer-smartphone-de-la-historia-fue.

Apéndice A

Distribución temporal

En este Anexo se muestra la división en tareas del proyecto así como el tiempo dedicado en cada una de ellas. Hay que remarcar que no son tiempos exactos ya que no se ha contabilizado la dedicación en cada una de las partes. Además, hay partes que no se han incluido debido a las modificaciones del propio proyecto durante su desarrollo y que, por tanto, no se han considerado vinculadas al proyecto final.

En primer lugar, se va a mostrar el gráfico general en la figura A.1 que está dividido en las diferentes fases del proyecto junto con el porcentaje de tiempo dedicado. En este gráfico no se muestran las subtareas realizadas en cada una de las fases ya que la intención era dar una visión global.

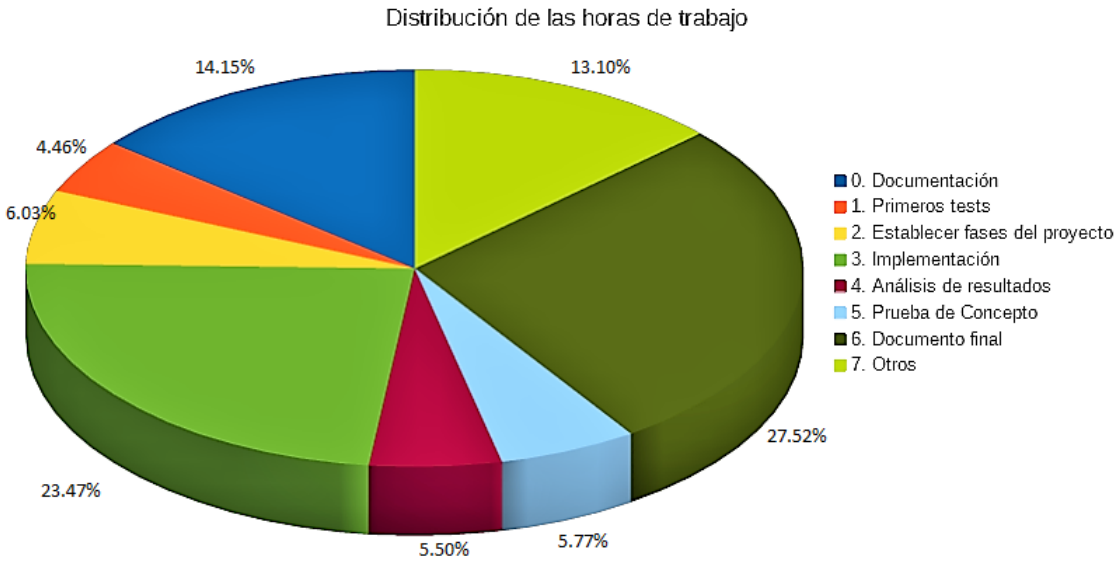

Figura A.1: Distribución de las horas de trabajo

Por otro lado, se muestran unas tablas con toda la información. La tablas con las

tareas se dividen en tres figuras. En la primera de ellas, la figura A.2, hace referencia a las primeras fases del proyecto, es decir, las fases de documentación con los proyectos considerados de interés y las primeras pruebas para familiarizar con el entorno Android ya que no se tenían conocimientos previos que alcanzasen tal profundidad. En la segunda tabla, en la figura A.3, están incluidas las tareas fundamentales del proyecto. Finalmente, en la figura A.4 se muestran las tareas finales relacionadas con la comprobación de resultados, la prueba de concepto, tiempo de redacción del presente documento y otra información de interés.

Fase	Tiempo dedicado (horas)	Porcentaje
0. Documentación	54	14,15 %
0.1 Android internals: A confectioner's cookbook	30	
0.2 AppGuard – Enforcing User Requirements on Android apps	2	
0.3 Slicing Droids: Program Slicing for Smali Code	2	
0.4 Pscout: Analyzing the Android Permission Specification	4	
0.5 On the Effectiveness of API-Level Access Control Using Bytecode Rewriting in Android	2	
0.6 Towards Black Box Testing of Android apps	2	
0.7 I-ARM-Droid: A Rewriting Framework for In-App Reference Monitors for Android applications	2	
0.8 IRM Enforcement of Java Stack Inspection	2	
0.9 Rage Against the Virtual Machine: Hindering Dynamic Analysis of Android Malware	2	
0.10 SuSi: A Tool for the Fully Automated Classification and Categorization of Android Sources and Sinks	2	
0.11 Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces	2	
0.12 ARTDroid: A Virtual-Method Hooking Framework on Android ART Runtime	2	
1. Primeros tests	17	4,46 %
1.1 Test Java Reflection Smali	6	
1.2 Test APIMonitor	4	
1.3 Test Proguard	3	
1.4 Tests Python	4	

Figura A.2: Tareas fases iniciales

Fase	Tiempo dedicado (horas)	Porcentaje
2. Establecer fases del proyecto	23	6,03 %
2.1 Decisión desarrollo scripts de descarga	3	
2.1.1 Análisis de las APIs	3	
2.2 Decisión Desarrollo de Apkdissector	3	
2.2.1 Multithreading	2	
2.2.2 Diseño de las bases de datos	1	
2.3 Decisión desarrollo del analizador dinámico de permisos	15	
2.3.1 Análisis forzado de permisos en Android	15	
2.4 Prueba de Concepto	2	
3. Implementación	89,55	23,47 %
3.1 Bases de datos	3,55	
3.1.1 Base de datos Pscout	3	
3.1.2 Base de datos hashes	0,55	
3.1.2.1 Integración base de datos Pscout	0,5	
3.1.2.2 Añadir tabla para análisis	0,05	
3.2 Implementación scripts de descarga	20	
3.2.1 Implementación script Koodous	16	
3.2.2 Implementación script Andrototal	2	
3.2.3 Implementación script Virustotal	2	
3.3 Implementación de Apkdissector	54	
3.3.1 Androguard	15	
3.3.2 Multithreading	20	
3.3.3 Acceso a bases de datos	10	
3.3.4 Resultados analizador en ficheros JSON	6	
3.3.5 Resultados analizador en base de datos	3	
3.4 Implementación analizador dinámico	12	
3.4.1 Modificación del AOSP	1	
3.4.2 Búsqueda kernel con SELinux permissive	5	
3.4.3 Ejecución CTS	6	

Figura A.3: Tareas del núcleo del proyecto

Fase	Tiempo dedicado (horas)	Porcentaje
4. Análisis de resultados	21	5,50 %
4.1 Comprobación de resultados de Pscout	2	
4.2 Comprobación de resultados tras CTS	4	
4.3 Implementación script de comparación de resultados entre Pscout y el analizador dinámico	15	
5. Prueba de Concepto	22	5,77 %
5.1 Test de prueba con Frida	12	
5.2 Implementación aplicación Android vulnerable	4	
5.3 Implementación ficheros de hook para Frida	6	
6. Documento final	105	27,52 %
6.1 Escritura memoria final	105	
6.1.1 Documento	100	
6.1.2 Gestión de tiempo	5	
7. Otros	50	13,10 %
7.1 Intentos fallidos	23	
7.1.1 Análisis limitaciones Pscout	3	
7.1.2 Intento de integración de Slicing Droids	10	
7.1.3 Intento de uso de ARTDroid	3,5	
7.1.4 Intento de uso de ARTHook	6,5	
7.2 Descarga de aplicaciones	20	
7.2.1 Problemas con claves de la API de Koodous	20	
7.3 Problemas con CTS	7	
Tiempo total:	**381,55**	**100,00 %**

Figura A.4: Tareas finales y resultados

Apéndice B

Diccionario de términos

En este Anexo se muestra una explicación de cada uno de los conceptos empleados en este documento. Se recomienda la lectura del proyecto apoyándose en este apéndice para facilitar su comprensión.

Con el objetivo de dividir los conceptos que componen el documento, se presentan separados en dos secciones: términos de Android y términos adicionales.

B.1. Términos de Android

- **Activity:** Sección de la funcionalidad de una aplicación Android para que el usuario realice una tarea concreta. Debe definirse en el fichero AndroidManifest.
- **Android Application Package (APK):** Formato de las aplicaciones de Android.
- **Android Debug Bridge (ADB):** Herramienta disponible para acceder a un dispositivo Android.
- **Android Interface Definition Language (AIDL):** Interfaz para definir el esquema de comunicación entre procesos.
- **Android Open Source Project (AOSP):** Proyecto *open source* de Android desarrollado por Google.
- **Android Virtual Device (AVD):** Emulación de un dispositivo Android real.
- **Content Provider:** Gestor de acceso a un conjunto de datos estructurado. Se deben definir antes de su uso en el fichero AndroidManifest.
- **Compatibility Test Suite (CTS):** Test ofrecido por Google para los fabricantes con el fin de cumplir la compatibilidad con el Android Open Source Project.
- **Dalvik Executable (DEX):** Ficheros compilados de una aplicación Android.

- **Google Summer of Code (GSOC):** Concurso de Google en el que estudiantes desarrollan un proyecto con el apoyo de un mentor de Google.
- **Service:** Opción que ofrece Android para la realización de tareas largas sin interacción del usuario. Debe definirse previamente en el fichero AndroidManifest.
- **Software Development Kit (SDK):** Conjunto de herramientas de software.

B.2. Términos adicionales

- **Application Programming Interface (API):** Definición de herramientas y funciones para desarrollar una aplicación.
- **Call Flow Graph (CFG):** Grafo útil para gestionar el proceso de invocaciones seguidas por una determinada función.
- **Comma-Separated Values (CSV):** Formato de ficheros de texto compuestos por diferentes campos separados por comas.
- **eXtensible Markup Language (XML):** Formato de fichero de texto con una estructura definida mediante *tags*.
- **First In First Out (FIFO):** Tipo de cola en el que el primer elemento en añadirse es el primero en salir.
- **Graphical User Interface (GUI):** Interfaz gráfica de cualquier aplicación para ser más amigable para el usuario.
- **Hypertext Transfer Protocol Secure (HTTPS):** Protocolo de seguridad basado en certificados para dotar de seguridad a la navegación web.
- **Inline Reference Monitor (IRM):** Modificación de un software o aplicación con el fin de aplicar unas políticas de seguridad.
- **International Mobile Station Equipment Identity (IMEI):** Identificador de un dispositivo móvil.
- **Inter-Process Communication (IPC):** Función básica de un sistema operativo para comunicación de procesos.
- **JavaScript Object Notation (JSON):** Formato de fichero de texto con una estructura definida fácilmente analizable. Similar al formato XML.
- **Ley Orgánica de Protección de Datos (LOPD):** Ley española de protección de datos de carácter personal.
- **Proof of Concept (PoC):** Exposición de los resultados obtenidos para un proyecto.

- **Remote Procedure Call (RPC):** Software para ejecución de funciones de código localizadas en otra máquina.
- **Short Message Service (SMS):** Servicio de envío de mensajes de texto.
- **Subscriber Identity Module (SIM):** Tarjeta empleada para identificar a los usuarios de una red telefónica.
- **Uniform Resource Identifier (URI):** Identificador para localizar un cierto recurso.
- **Uniform Resource Locator (URL):** Identificador de un recurso web.

Apéndice C

Androguard

En este Anexo se muestra un ejemplo de análisis de una aplicación empleando androlyzer.py que permite observar aspectos básicos como pueden ser los permisos o las *activities*. A bajo nivel, Androguard lo que hace es decodificar la aplicación, leer el fichero Manifest, con formato XML, y filtrar la información requerida. Lo bueno de esta herramienta es que ofrece una línea de comandos para hacer las diferentes consultas.

```
In [1]: a,d,dx = AnalyzeAPK("ffdb94d60214020248b8
0e3d464af1bab6e4a0413d7e55cb405cd0d9a97ed43a.apk",
decompiler="dad")

In [2]: a.get_permissions()
Out[2]:
['android.permission.WAKE_LOCK',
 'android.permission.INTERNET',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.ACCESS_NETWORK_STATE',
 'android.permission.WRITE_EXTERNAL_STORAGE',
 'android.permission.WRITE_SETTINGS']

In [3]: a.get_activities()
Out[3]:
['com.ta3alam.ibriya.fiasra3wa9t.Hebrew.Audio95212',
 'com.ta3alam.ibriya.fiasra3wa9t.Hebrew.Dashboard_000',
 'com.google.android.gms.ads.AdActivity',
 'com.google.android.gms.ads.purchase.InAppPurchaseActivity']
```

Apéndice D

Análisis de AppGuard

En el presente Anexo se va a realizar un análisis de una aplicación simple que requiere el IMEI del dispositivo. Para comprobar el proyecto de AppGuard se plantean dos modos de invocación: normal y mediante Java Reflection [Ora]. Para ilustrarlo de forma más clara se muestra una invocación Java y, después, se analiza el código Smali vinculado a dicho fragmento de código una vez compilado.

En el siguiente fragmento de código se observa el tratamiento de una llamada al método *getDeviceId()* de la clase *TelephonyManager* que require tener el permiso *READ_PHONE_STATE* declarado en el AndroidManifest.

```
1  TelephonyManager tm = (TelephonyManager) getSystemService
2      (this.TELEPHONY_SERVICE);
3  imei = tm.getDeviceId();
```

Invocación al método *getDeviceId()*.

A continuación, en el fragmento de código inferior se observa como aparece la anterior llamada en el código smali. La información más importante se encuentra en la primera instrucción, encargada de realizar la invocación. En ella, se muestra que el objeto *TelephonyManager*, creado anteriormente, llama a su método *getDeviceId()*, seguido del tipo de objeto que devuelve y donde lo va a almacenar, es decir, devuelve un String y lo almacena en v1. Las siguiente instrucciones se encargan de establecer el valor obtenido por la operación realizada en la Activity que contiene la llamada, se mueve el registro v1 a v2 (línea 3) y se almacena lo que contiene dicho registro en la variable , de tipo String, en *MainActivity* (línea 4).

```
1  invoke-virtual {v1},Landroid/telephony/TelephonyManager;
2      ->getDeviceId()Ljava/lang/String;
3  move-result-object v2
4  iput-object v2, p0,Lcom/example/sergio/zeroapp/MainActivity;
5      ->:Ljava/lang/String;
```

Invocación al método *getDeviceId()* en lenguaje Smali.

Java ofrece la posibilidad de realizar las llamadas a métodos de una determinada clase de forma genérica. Esto se conoce como *Java Reflection* y, por medio de la clase Method, es posible, partiendo de una objeto determinado, crear un objeto *Method* capaz de obtener a través de un String el método de interés. Para invocarlo se emplea el método *invoke()* lo que dará un resultado igual que si se hace siguiendo la forma común. En el siguiente fragmento de código se observa un ejemplo.

```
1  TelephonyManager tm = (TelephonyManager) getSystemService(
2  this.TELEPHONY_SERVICE);
3  try {
4      Method m = TelephonyManager.class.
5          getMethod("getDeviceId", null);
6      try {
7           imei = (String) m.invoke(tm, null);
8      }
9      [..]
10 }
11 [..]
```

Invocación al método *getDeviceId()* con Java Reflection.

Este procedimiento tiene implicaciones en el bajo nivel ya que, en vez de aparecer en el código smali la invocación a *getDeviceId()*, aparecerá el nombre del método en un registro que se pasará en la invocación lo que supone un problema para los analizadores estáticos. Para ilustrar un ejemplo de sus implicaciones, se va a analizar su comportamiento en las aplicaciones mostradas anteriormente tras ser modificadas por APIMonitor, herramienta pública que sigue el esquema planteado por Appguard.

D.1. Implicaciones para AppGuard

Para ilustrar el tratamiento que hace el *rewriter*, lo mejor es observar directamente el código Smali. En un primer lugar, se va a analizar la aplicación con una invocación sin uso de *Reflection* al método *getDeviceId()*. Si se observa el código del fragmento de abajo aparece la invocación a dicho método tras la modificación del código que aplica APIMonitor: en vez de partir del objeto TelephonyManager que ofrece Android se emplea el modificado por ellos, indicado por emplear el paquete Ldroidbox en la línea 9.

```
1  invoke-virtual {p0, v2}, Lcom/example/sergio/zeroapp/
2      MainActivity;->getSystemService(Ljava/lang/String;
3      )Ljava/lang/Object;
4
5  move-result-object v1
6
7  check-cast v1, Landroid/telephony/TelephonyManager;
```

```
8
9  invoke-static {v1}, Ldroidbox/android/telephony/
10     TelephonyManager;->getDeviceId(Landroid/telephony
11     /TelephonyManager;)Ljava/lang/String;
```

Invocación al método *getDeviceId()* con APIMonitor en código Smali.

Con ello, se concluye que efectivamente interfiere en la llamada original y la modifica. Pero... ¿qué pasaría si se invoca a través de *Reflection*? En la siguiente sección de código se observa el proceso de invocación a *getDeviceId()* con *Reflection*. En la línea 12 se observa el registro en el que se carga el nombre del método en forma de String. Ese registro se emplea más tarde en la línea 16 que es la encargada de crear el objeto *Method*. Ese objeto invoca al método establecido en el registro anterior, visible en la línea 29, que obtendría el IMEI del dispositivo y, por último, en la línea 37, se introduciría el valor en la variable llamada *imei*.

Como se observa en la línea 29, teniendo en cuenta que se ha empleado APIMonitor para asegurar la llamada a *getDeviceId()*, no aparece la llamada a través de DroidBox al método modificado. Esto se debe a que no considera Java Reflection y ha conseguido evitar la invocación segura.

```
1  invoke-virtual {p0, v4}, Lcom/example/sergio/zeroapp/
2      MainActivity;->getSystemService(Ljava/lang/String;
3      )Ljava/lang/Object;
4
5  move-result-object v3
6
7  check-cast v3, Landroid/telephony/TelephonyManager;
8
9  :try_start_0
10 const-class v4, Landroid/telephony/TelephonyManager;
11
12 const-string v5, "getDeviceId"
13
14 const/4 v6, 0x0
15
16 invoke-virtual {v4, v5, v6}, Ljava/lang/Class;->getMethod
17     (Ljava/lang/String;[Ljava/lang/Class;
18     )Ljava/lang/reflect/Method;
19
20 :try_end_0
21 .catch Ljava/lang/NoSuchMethodException;
22 {:try_start_0 .. :try_end_0} :catch_1
23
24 move-result-object v2
```

```
25
26  const/4 v4, 0x0
27
28  :try_start_1
29  invoke-virtual {v2, v3, v4}, Ljava/lang/reflect/Method;
30      ->invoke(Ljava/lang/Object;[Ljava/lang/Object;
31      )Ljava/lang/Object;
32
33  move-result-object v4
34
35  check-cast v4, Ljava/lang/String;
36
37  iput-object v4, p0, Lcom/example/sergio/zeroapp/
38      MainActivity;->:Ljava/lang/String;
```

Invocación al método *getDeviceId()* con Java Reflection y APIMonitor en código Smali.

¿Qué implicaciones tienen estos resultados en el análisis estático de aplicaciones Android? Teniendo en cuenta que es posible saltarse las políticas de seguridad haciendo que no sean relevantes, las conclusiones son claras y negativas. Para evitar este tipo de invocaciones se emplea los CFG, comentados en el capítulo 3.2.

Los CFG permiten analizar el flujo de uso de un determinado registro dentro del código smali por lo que, en caso de encontrar Java Reflection, se podría controlar el registro que contiene el nombre del método que se va a invocar y evitar comportamientos de este tipo dentro de los analizadores estáticos.

Apéndice E

Consultas a la base de datos

A continuación se van a mostrar una serie de consultas a la base de datos creada y poblada con los *scripts* mencionados anteriormente.

E.1. Consulta tabla hashes

En esta consulta se quiere mostrar la estructura de los elementos de la tabla *hashes*. Observando la consulta, se comprueba la versatilidad que ofrece la distinción mediante tipos a la hora de conocer de que base de datos proviene el *hash*, mostrándose junto con un valor que indica si está descargada o no.

```
sqlite> select * from hashes where type = 'andrototal' limit 5;
00002d74a9faa53f5199c910b652ef09d3a7f6bd42b693755a233635c3ffb0f4
|1|andrototal|0
0000350c0792f61ee513f40bd9a42d09144cc6a3c4f2171f812ef415a9a51640
|1|andrototal|0
00017d87a5a7a738061e3e40cf29cc9ac68d114451566d88ad8051015fa9f671
|1|andrototal|0
000198ad556a1f50cc285c76ae8f8d1b2472808f3ccaa0b429a60984912ce88f
|1|andrototal|0
0001bbdf3489bbaa27df4af9d52b77028a1b7315b528a1e6bce8b63b0d3384a8
|1|andrototal|0
```

E.2. Consulta tabla analyzed

Para conocer las aplicaciones que han sido analizadas, es necesario consultar la tabla *analyzed*. En ella, se muestran los permisos que una determinada aplicación contiene en su *Manifest*. De esta forma, se pueden comprobar facilmente los permisos que tiene una determinada aplicación y, posteriormente, realizar estadísticas sobre esos permisos.

```
sqlite> select * from analyzed where hash = '0e06661024b010880f
099f2f6b4a2bdaddd2358b47977a882ebda66851d3e880' LIMIT 5;
```

```
0e06661024b010880f099f2f6b4a2bdaddd2358b47977a882ebda66851d3e880|
com.android.launcher.permission.INSTALL_SHORTCUT
0e06661024b010880f099f2f6b4a2bdaddd2358b47977a882ebda66851d3e880|
android.permission.GET_TASKS
0e06661024b010880f099f2f6b4a2bdaddd2358b47977a882ebda66851d3e880|
android.intent.category.HOME
0e06661024b010880f099f2f6b4a2bdaddd2358b47977a882ebda66851d3e880|
android.permission.RESTART_PACKAGES
0e06661024b010880f099f2f6b4a2bdaddd2358b47977a882ebda66851d3e880|
android.permission.REORDER_TASKS
```

E.3. Consulta tabla pscout

La tabla *pscout* contiene la información que ofrece el proyecto Pscout. Para poblarla se ha elaborado un *parser* de sus ficheros y, a continuación, se ha integrado en la base de datos construida. La estructura es la misma que contiene los ficheros de este proyecto en los que se observa un mapeo de los permisos.

```
sqlite> select * from pscout where permission =
'android.permission.INTERNET' limit 5;
31808|com/android/server/ConnectivityService|reportInetCondition|
(II)V|android.permission.INTERNET|5.1.1

31809|com/android/server/NsdService|getMessenger|
()Landroid/os/Messenger;|android.permission.INTERNET|5.1.1

31810|com/android/server/ConnectivityService|reportBadNetwork|
(Landroid/net/Network;)V|android.permission.INTERNET|5.1.1

31811|com/android/browser/BrowserWebView|<init>|
(Landroid/content/Context;)V|android.permission.INTERNET|5.1.1

31812|com/android/browser/BrowserWebView|<init>|
(Landroid/content/Context;Landroid/util/AttributeSet;
ILjava/util/Map;Z)V|android.permission.INTERNET|5.1.1
```

Apéndice F

Estadísticas de permisos

Permiso	Porcentaje	Cuenta
android.permission.INTERNET	97,43 %	2011
android.permission.ACCESS_NETWORK_STATE	92,97 %	1919
android.permission.WRITE_EXTERNAL_STORAGE	82,03 %	1693
android.permission.READ_PHONE_STATE	71,75 %	1481
android.permission.ACCESS_WIFI_STATE	64,53 %	1332
android.permission.WAKE_LOCK	57,95 %	1196
android.permission.ACCESS_COARSE_LOCATION	41,96 %	866
android.permission.RECEIVE_BOOT_COMPLETED	41,09 %	848
android.permission.GET_TASKS	40,41 %	834
android.permission.VIBRATE	39,92 %	824
android.permission.ACCESS_FINE_LOCATION	37,69 %	778
android.permission.SYSTEM_ALERT_WINDOW	33,67 %	695
com.android.launcher.permission.INSTALL_SHORTCUT	31,59 %	652
android.permission.CHANGE_WIFI_STATE	27,13 %	560
android.permission.SEND_SMS	25,15 %	519
android.permission.WRITE_SETTINGS	23,35 %	482
android.permission.MOUNT_UNMOUNT_FILESYSTEMS	22,92 %	473
android.permission.GET_ACCOUNTS	22,72 %	469
android.permission.RECEIVE_SMS	22,34 %	461
com.google.andrroid.c2dm.permission.RECEIVE	20,40 %	421
android.permission.CHANGE_NETWORK_STATE	19,86 %	410
android.permission.READ_EXTERNAL_STORAGE	19,67 %	406
android.permission.READ_SMS	19,38 %	400
android.permission.CALL_PHONE	15,75 %	325
android.permission.READ_CONTACTS	14,63 %	302
android.permission.CAMERA	14,24 %	294
android.permission.READ_LOGS	14,10 %	291
android.permission.WRITE_SMS	13,91 %	287
android.permission.RESTART_PACKAGES	13,66 %	282

Figura F.1: Tabla de estadísticas de permisos

Apéndice G

Código de la prueba de concepto

G.1. Código Android

G.1.1. MainActivity

```
1  package com.example.t4k3d0wn.exampleapp;
2
3  import android.bluetooth.BluetoothAdapter;
4  import android.bluetooth.BluetoothManager;
5  import android.content.Context;
6  import android.content.Intent;
7  import android.content.pm.PackageManager;
8  import android.support.v7.app.AppCompatActivity;
9  import android.os.Bundle;
10 import android.telephony.TelephonyManager;
11 import android.util.Log;
12 import android.view.View;
13 import android.widget.Button;
14 import android.widget.EditText;
15 import android.widget.TextView;
16 import android.widget.Toast;
17
18 import org.w3c.dom.Text;
19
20 public class MainActivity extends AppCompatActivity {
21
22     private TelephonyManager tm;
23     private BluetoothAdapter ba;
24
25     private Button imeiButton, checkButton, webViewButton,
   ↪  bluetoothButton;
```

```
26      private TextView textIMEI;
27      private EditText editPIN;
28
29      private String imei;
30
31
32      @Override
33      protected void onCreate(Bundle savedInstanceState) {
34          super.onCreate(savedInstanceState);
35          setContentView(R.layout.activity_main);
36
37          textIMEI = (TextView) findViewById(R.id.textView);
38          checkButton = (Button) findViewById(R.id.sendPIN);
39          editPIN = (EditText) findViewById(R.id.editText);
40
41          //Get IMEI button and hide until unlock with PIN
42          imeiButton = (Button) findViewById(R.id.button);
43          imeiButton.setVisibility(View.INVISIBLE);
44
45          webViewButton = (Button) findViewById(R.id.webviewButton);
46          webViewButton.setVisibility(View.INVISIBLE);
47
48          bluetoothButton = (Button) findViewById(R.id.bluetoothButton);
49          bluetoothButton.setVisibility(View.INVISIBLE);
50          bluetoothButton.setOnClickListener(new View.OnClickListener()
    ↪ {
51              public void onClick(View view) {
52                  modifyBluetooth();
53              }
54          });
55
56          tm = (TelephonyManager)
    ↪ getSystemService(Context.TELEPHONY_SERVICE);
57          ba = BluetoothAdapter.getDefaultAdapter();
58      }
59
60      public void changeActivity(View view){
61          Intent intent = new Intent(getApplicationContext(),
    ↪ WebViewActivity.class);
62          startActivity(intent);
63      }
64
65      public void getIMEI(View view){
```

```
66          imei = tm.getDeviceId();
67          Log.e("IMEI",imei);
68          if(!imei.contains("Este")){
69              imei = imei.substring(0,3) + "***************";
70          }
71
72          Toast.makeText(view.getContext(),imei,
    ↪ Toast.LENGTH_LONG).show();
73          updateTV(imei);
74      }
75
76      public void updateTV(String s){
77          textIMEI.setText(s);
78      }
79
80      public void modifyBluetooth(){
81
82          if(ba.isEnabled()){
83              ba.disable();
84          }
85          else{
86              ba.enable();
87          }
88      }
89
90      public void checkPIN(View view){
91          String text = editPIN.getText().toString();
92          if(!text.equals("")){
93              if(text.equals("1234")){
94                  Toast.makeText(view.getContext(), "PIN OK",
    ↪ Toast.LENGTH_LONG).show();
95                  hidePinVerification();
96              }
97              else{
98                  Toast.makeText(view.getContext(), "WRONG PIN",
    ↪ Toast.LENGTH_LONG).show();
99              }
100         }
101     }
102
103     private void hidePinVerification(){
104         editPIN.setVisibility(View.INVISIBLE);
105         checkButton.setVisibility(View.INVISIBLE);
```

```
106             imeiButton.setVisibility(View.VISIBLE);
107             webViewButton.setVisibility(View.VISIBLE);
108             bluetoothButton.setVisibility(View.VISIBLE);
109         }
110
111     }
```

G.1.2. WebViewActivity

```
1   package com.example.t4k3d0wn.exampleapp;
2
3   import android.os.Bundle;
4   import android.support.v7.app.AppCompatActivity;
5   import android.support.v7.widget.Toolbar;
6   import android.view.View;
7   import android.webkit.WebView;
8   import android.widget.Button;
9   import android.widget.EditText;
10
11  public class WebViewActivity extends AppCompatActivity {
12
13      private String url = "";
14      private WebView w;
15      private EditText newurl;
16      private Button refreshButton;
17
18      @Override
19      protected void onCreate(Bundle savedInstanceState) {
20          super.onCreate(savedInstanceState);
21          setContentView(R.layout.webview);
22
23          w = (WebView) findViewById(R.id.webviewId);
24          w.getSettings().setJavaScriptEnabled(true);
25
26          newurl = (EditText) findViewById(R.id.newUrl);
27          refreshButton = (Button) findViewById(R.id.
28              refreshButton);
29      }
30
31      public void refreshWebView(View view){
32          String text = newurl.getText().toString();
33          if(text.contains("http://")){
34              url = text;
35          }
```

```
36          else{
37              url = "http://" + text;
38          }
39          updateURL(url);
40      }
41
42      private void updateURL(String url){
43          w.loadUrl(url);
44      }
45  }
```

G.2. Código Python

```
1   import frida
2   import sys
3
4   package_name = "com.example.t4k3d0wn.exampleapp"
5
6
7   def get_messages_from_js(message, data):
8       print(message)
9       print (message['payload'])
10
11
12  def instrument_load_url():
13
14      hook_code = """
15      Java.perform(function () {
16          // Function to hook is defined here
17          var BA = Java.use('android/bluetooth/BluetoothAdapter');
18
19          // Whenever button is clicked
20          BA.isEnabled.implementation = function (v) {
21              send("hook - isEnabled");
22
23              return true;
24          };
25
26          var TM = Java.use('android/telephony/TelephonyManager');
27          TM.getDeviceId.overload("int").implementation = function(v) {
28              send("hook - getDeviceId");
29
30              return "Este no es tu IMEI";
```

```
31          };
32
33          var MainActivity =
    ↪   Java.use('com.example.t4k3d0wn.exampleapp.MainActivity');
34
35          // Whenever button is clicked
36          MainActivity.checkPIN.implementation = function (v) {
37              // Show a message to know that the function got called
38              send('Call - CheckPIN');
39
40              this.hidePinVerification();
41
42          };
43
44          var WebViewActivity =
    ↪   Java.use('com.example.t4k3d0wn.exampleapp.WebViewActivity');
45
46          // Whenever button is clicked
47          WebViewActivity.refreshWebView.implementation = function (v) {
48
49              // Show a message to know that the function got called
50              send('Call - refreshWebView');
51
52              this.updateURL("http://192.168.1.39");
53          };
54      });
55      """
56      return hook_code
57
58  process = frida.get_usb_device().attach(package_name)
59  script = process.create_script(instrument_load_url())
60  script.on('message',get_messages_from_js)
61  script.load()
62  sys.stdin.read()
```