# ANÁLISIS DE LOS FLUJOS DE INFORMACIÓN EN ANDROID

# HERRAMIENTAS PARA EL CUMPLIMIENTO DE LA RESPONSABILIDAD PROACTIVA

Fecha de publicación: 07/03/19

# ÍNDICE

# I. RESUMEN EJECUTIVO

Las aplicaciones presentes en los teléfonos móviles pueden manejar datos como las fotografías, correos o la agenda de actividades, pueden acceder a ciertos datos generados por sensores integrados en el dispositivo o conectados a él, como la localización o los signos vitales de los usuarios, y a ciertos identificadores usados por el hardware, sistema operativo, servicios y otras aplicaciones, lo que se denomina firma digital del dispositivo (ver el estudio Fingerprint o Huella Digital del Dispositivo publicado por la AEPD). Estos datos personales pueden ser procesados internamente por las aplicaciones, aunque también pueden ser comunicados internamente a otras aplicaciones dentro del mismo dispositivo o hacia entidades externas (ej. un servidor de análisis de datos).

La versatilidad de datos, tratamientos y la potencialidad de las comunicaciones de datos en el modelo Android eleva el riesgo de una posible explotación no legítima de datos personales por terceros.

El responsable del tratamiento realizado por una aplicación móvil tiene la obligación de informar al usuario a través políticas de privacidad, notificaciones o descripciones publicadas en las tiendas de aplicaciones, y la implementación efectiva del servicio ha de ajustarse a los límites de esa información, de la legitimación para el tratamiento y las garantías generales del RGPD. La realidad es que el responsable del tratamiento de los datos de una app no siempre será el desarrollador directo y exclusivo de esta, se basará en librerías de terceros, subcontrataciones o acuerdos y/o en la ejecución en el entorno de un tercero, por lo que hay una potencial pérdida de control sobre la implementación de dicho tratamiento y un aumento de la complejidad para abordar los mencionados requisitos de cumplimiento en materia de protección de datos.

Los desarrolladores de aplicaciones móviles, los responsables que subcontraten dichos desarrollos y distribuidores o repositorios de apps tienen la obligación de asegurar que las apps que ponen a disposición de los usuarios están de acuerdo a las políticas de privacidad y los servicios publicitados con las garantías adecuadas. Es decir, aplicar los principios de Responsabilidad Proactiva a través de la aplicación de medidas de Privacidad por Defecto, minimizando el tratamiento de datos su extensión, conservación y accesibilidad, y de Privacidad desde el Diseño, seleccionando aquellos componentes más respetuosos para la privacidad.

Para poder cumplir con estás obligaciones, en este documento se presenta un estudio sobre las técnicas existentes para el análisis de flujos de información personal en aplicaciones móviles que se ejecutan en terminales con sistema operativo Android. En primer lugar, se presenta el entorno de ejecución de estas aplicaciones, sus componentes fundamentales, los diferentes actores involucrados en el tratamiento de los datos y una breve descripción del ciclo de vida de los datos. Posteriormente, se describen las principales técnicas y herramientas para análisis de flujos de información personal, que incluyen el análisis estático de código, análisis de ejecución, y análisis de comunicaciones.

# II. INTRODUCCIÓN

## OBJETIVOS Y ALCANCE

Este estudio se lleva a cabo en el marco de colaboración entre la Universidad Politécnica de Madrid (UPM) y la Agencia Española de Protección de Datos (AEPD), cuyo alcance es identificar las técnicas y herramientas existentes para detectar flujos de información personal en software para dispositivos móviles.

En lo que respecta a los objetivos, este estudio se centra especialmente en:

- Definir el contexto y marco conceptual de la detección de las comunicaciones de información personal en las aplicaciones que se ejecutan sobre el sistema operativo Android.
- Poner de manifiesto el elevado riesgo de fugas de información personal que se presenta en el entorno de aplicaciones móviles y la necesidad de realizar una evaluación de los flujos de datos que tenga en cuenta el ciclo de vida de esta información y permita llevar a cabo una posible evaluación de impacto de dicho flujo de datos con relación a la privacidad y el derecho a la protección de datos de las personas
- Estudiar las técnicas existentes para el detección y análisis de flujos de información personal en aplicaciones Android.

El presente documento es el primer resultado de la colaboración UPM-AEPD.

# III. PRIVACIDAD Y PROTECCIÓN DE DATOS EN LAS APLICACIONES ANDROID

## ENTORNO DE LAS APLICACIONES ANDROID

En esta sección se presentan los principales elementos y actores del entorno de las aplicaciones Android situándolos en el contexto de la privacidad y protección de datos personales. La Figura 1 describe la relación entre dichos elementos y actores.

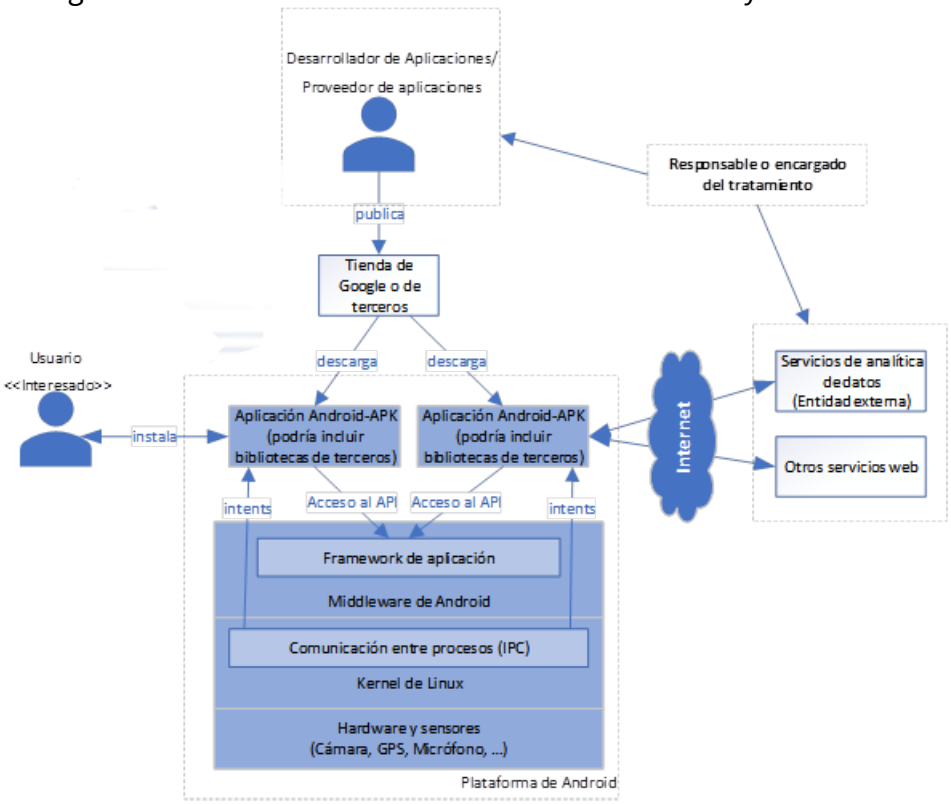


Figura 1. Ecosistema Android

Los desarrolladores de aplicaciones para Android usan principalmente Java como lenguaje de programación. Para facilitar el desarrollo, Android proporciona un conjunto de bibliotecas software (*Framework* de Android) que contienen los componentes básicos para construir las aplicaciones, e interfaces software (APIs - *Application Programming Interfaces*) para acceder a servicios del sistema operativo (ej. servicio de administración de *bluetooth*) y a los datos generados por ciertos recursos del dispositivo (ej. datos de los sensores). Además, un gran número de aplicaciones también usan bibliotecas de terceros con diferentes finalidades, como agregar funcionalidades o monetizar sus aplicaciones (ej. bibliotecas para anuncios personalizados).

Las bibliotecas del Framework de Android contienen cuatro componentes básicos para construir aplicaciones[1]:

- Actividades: representan las interfaces de usuario (ej. una interfaz para enviar un correo electrónico).
- Servicios: permiten ejecutar tareas prolongadas o realizar conexiones remotas en segundo plano (ej. un servicio para descarga de archivos).
- Proveedores de contenido: administran datos almacenados persistentemente (ej. un proveedor de contenidos administra la creación, lectura, modificación y eliminación de registros de una base de datos interna del dispositivo).
- Receptores de mensajes: permiten recibir los mensajes difundidos por el sistema u otras aplicaciones (ej. recibir la notificación de que una descarga ha finalizado). Dos aplicaciones pueden comunicarse entre ellas (IPC – *Inter Process Communication*) a través de lo que se conoce como *intents*. Por ejemplo, una aplicación de salud y bienestar puede transmitir un mensaje a otra aplicación como una red social indicando que el usuario ha corrido 10km en 30 minutos.

Una vez terminado el desarrollo de las aplicaciones, estas son compiladas obteniendo un código en formato DEX (*Dalvik Executable*) y comprimidas junto con otros recursos necesarios para su ejecución en paquetes APKs (*Android Packages*). Los APKs son publicados en la tienda oficial de Google Play[2] o tiendas de terceros como Uptodown[3] o APK Pure[4].

Los teléfonos móviles son una fuente de datos personales, dado que son utilizados por un usuario para llevar a cabo sus actividades cotidianas. Sus aplicaciones pueden manejar datos que son personales por naturaleza (ej. las fotografías, notas de audio, correos, actividades de la agenda y lista de contactos de un usuario), pero también son capaces de acceder a ciertos datos generados por recursos/sensores internos (ej. la localización[5], registros de uso de las aplicaciones) o externos (ej. el pulso cardiaco obtenido de una pulsera de monitorización). A esto hay que sumarle que el hardware, sistema operativo, servicios y aplicaciones manejan internamente identificadores globales que podrían permitir identificar (y rastrear) a los usuarios de los dispositivos. El Reglamento General de Protección de Datos (RGPD)[6] establece que son datos personales "*toda información sobre una persona física identificada o identificable («el interesado»)*", por lo tanto, las aplicaciones móviles que recopilan y procesan los datos mencionados anteriormente deberían cumplir con los requerimientos definidos en este reglamento.

[1] https://developer.android.com/guide/components/fundamentals?hl=es-419

[2] https://play.google.com/store/apps?hl=en

[3] https://uptodown-android.uptodown.com/android

[4] https://apkpure.com/es/apkpure-app.html

[5] No únicamente la localización obtenida del GPS sino también inferida de otros datos como los identificadores de las redes wifi (SSID).

[6] https://eur-lex.europa.eu/legal-content/ES/TXT/HTML/?uri=CELEX:32016R0679&from=EN

El tratamiento[7] adecuado de los datos personales requiere una clara identificación de los principales roles involucrados y sus responsabilidades. El RGPD define los siguientes:

- Interesado: persona física identificada o identificable cuyos datos son objetos del tratamiento, con carácter general coincide con el usuario del dispositivo.
- Responsable del tratamiento: es la persona o entidad que determina los fines de tratamiento de los datos personales y es quien tiene que asegurarse que los requerimientos de protección de datos sean cumplidos aun cuando delegue en una entidad externa una tarea de procesamiento particular con finalidades definidas previamente.
- Encargado del tratamiento: persona o entidad que trata datos personales por delegación del responsable del tratamiento y siguiendo los términos y condiciones de dicho responsable.

En el ecosistema móvil, el usuario del dispositivo es el interesado y es a su vez usuario de aplicaciones con una finalidad que debería con la finalidad o finalidades fijadas por el responsable del tratamiento. El proveedor de servicios puede ser responsable del tratamiento si es quien determina el propósito, o encargado del tratamiento si únicamente trata los datos por encargo del responsable. Por ejemplo, en una aplicación de música en streaming el usuario de la aplicación es el interesado, la organización que explota la aplicación es el responsable del tratamiento y el encargado del tratamiento es el proveedor de los servicios en la nube donde el responsable aloja sus servicios.

### CICLO DE VIDA DE DATOS PERSONALES EN LAS APLICACIONES ANDROID

Las fases del ciclo de vida de la información personal se han definido generalizando los componentes de una aplicación como un todo en relación con elementos externos. La aplicación está formada por todos los componentes Android que han sido definidos en un archivo (MANIFEST.xml) que se incluye en el paquete de instalación de la aplicación (APK). Esta aclaración es esencial ya que, por ejemplo, una transmisión/difusión tendrá lugar solamente si ciertos datos personales han sido enviados fuera de los componentes propios de la aplicación que se está evaluando. La Figura 2 ilustra las fases del ciclo de vida de los datos personales en una aplicación Android, que incluye:

- Recolección: un componente de la aplicación recibe o accede a fuentes de datos personales que están fuera del dominio de la aplicación. Los accesos a la mayoría (aunque no todos) de recursos/datos deben ser declarados en el fichero MANIFEST.xml, y requieren la aceptación previa del usuario[8]. Entre las fuentes de potenciales datos personales resaltamos las siguientes:

[7] De acuerdo al RGPD tratamiento se refiere a cualquier operación o conjunto de operaciones sobre los datos personales (ej. recolección, utilización, difusión, almacenamiento, y supresión).

[8] En versiones de Android previas a Android 6.0 Marshmallow (versión 23 de la API de Android) todos los permisos requeridos por una aplicación se solicitaban al instalarla, y se otorgaban/denegaban en bloque. A partir de Android 6.0 cada permiso se solicita individualmente, en el momento que la aplicación intenta el acceso al recurso.

  - El propio usuario: Los usuarios proveen información personal directamente a las aplicaciones a través de formularios. Algunas aplicaciones suelen usar formularios de registro para solicitar, por ejemplo, el nombre, dirección, teléfono y otra información personal de los usuarios.
  - Sensores: Los dispositivos móviles incorporan o pueden interoperar con sensores de diferente naturaleza (ej. GPS, cámara, micrófono, Wifi, sensores para salud y estado físico, etc.) que generan una cantidad considerable de datos personales (ej. localización, fotografías y notas de audios personales, temperatura, ritmo cardíaco, etc.) a los que las aplicaciones pueden acceder.
  - Aplicaciones: Los datos personales pertenecientes a otras aplicaciones del dispositivo constituyen otra fuente de información que puede ser accedida por una aplicación a través de mecanismos de comunicación entre procesos (ej. mediante *intents*).
  - Entorno: Los dispositivos móviles almacenan identificadores, metadatos, registros de uso de las aplicaciones (ej. frecuencia de uso, hora, tipo de aplicaciones, etc.) o sus configuraciones (ej. configuración de las redes wifi) que pueden ser accedidos por las aplicaciones.

- Utilización: un componente de la aplicación realizará un tratamiento de ciertos datos personales. Por ejemplo, lo datos de localización originalmente representados por la latitud y longitud puede mapearse a la "ciudad" apuntada por dicha latitud o longitud. Esta transformación se hace mediante código en la aplicación, sin necesidad de transmitir los datos.
- Transmisión/difusión: un componente de la aplicación envía ciertos datos personales fuera de la aplicación. Por ejemplo, los datos de localización son enviados a otra aplicación (incluso dentro del mismo dispositivo), al servidor del proveedor para almacenarse, o a un servidor de terceros para su tratamiento.
- Almacenamiento: la aplicación almacena persistentemente ciertos datos personales en medios de almacenamiento que son accesibles a través de los Proveedores de Contenido pertenecientes a la aplicación. Es decir que la aplicación tiene control sobre los datos almacenados y podría mantenerlos en un espacio privado accesible solo por la aplicación o puede ponerlos a disponibilidad de otras aplicaciones.
- Supresión: la aplicación suprime ciertos datos personales a través de los Proveedores de Contenido pertenecientes a la aplicación.

Este estudio se centra en las fases de recolección y difusión de datos personales, en particular para detectar los flujos de información hacia entidades externas a la propia aplicación.

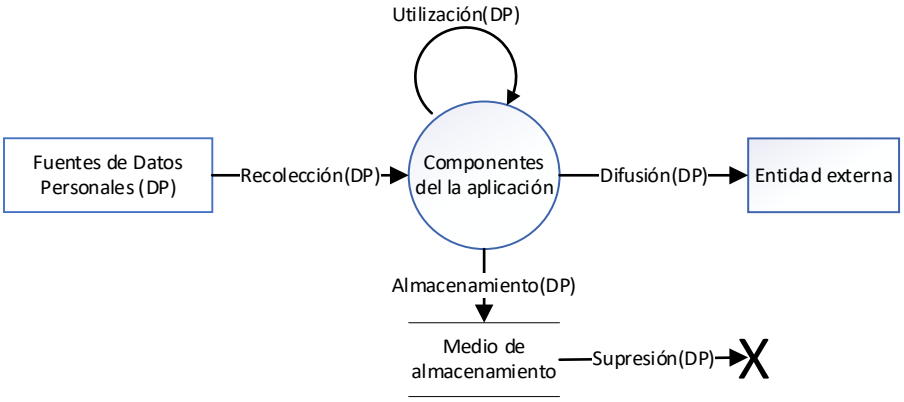


Figura 2. Ciclo de vida de los datos personales

## DECLARACIÓN DE LA PRIVACIDAD EN LAS APLICACIONES ANDROID

En el ecosistema móvil se han empleado diferentes canales para informar a los diferentes actores con respecto al tratamiento de los datos personales. Las descripciones textuales informales informan de la funcionalidad de la aplicación, y suelen estar disponibles en las tiendas de aplicaciones. Las políticas de privacidad informan de forma detallada de las prácticas del responsable y los derechos de los usuarios. Los avisos/notificaciones informan al usuario del intento de acceso de la aplicación a datos sensibles, solicitando su permiso.

Estos canales tienen destinatarios y propósitos diferentes:

- Descripciones textuales: Las descripciones textuales describen la funcionalidad de la aplicación y en ocasiones también incluyen información respecto al tratamiento de datos personales. Estas descripciones son escritas por los proveedores de las aplicaciones y publicadas en las tiendas, buscando ganar la atención de los usuarios finales. Las descripciones incluyen un lenguaje fácil de entender ya que los destinatarios de este canal son los usuarios, quienes deciden si instalar o no una aplicación luego de leer su descripción y evaluar si cumple sus expectativas. Según [1], las descripciones textuales tienen más presencia que los avisos y las políticas de privacidad; sin embargo, como el mismo estudio señala no todas las descripciones incluyen información respecto al tratamiento de datos personales. Por ejemplo, señala que menos del 30% de las descripciones textuales informan acerca del uso de la localización.
- Políticas de privacidad: Las políticas de privacidad son documentos, usualmente extensos, que describen las prácticas del responsable y los derechos de los usuarios acerca del tratamiento de sus datos personales. En la práctica tienen propósitos legales y deben incluir toda la información de acuerdo a la normativa, en particular el RGPD y la LOPDGDD. Se suelen presentar mediante un enlace que el usuario puede seguir, antes de la instalación de la aplicación. Aunque presentan información detallada al usuario, estudios previos presentan evidencias empíricas[9] que demuestran

[9] Por ejemplo en este estudio [4] se muestra que solo el 20% de usuarios admite haber leído una política de privacidad.

que son muy complejas de entender, utilizando términos técnicos y legales complicados para los usuarios, y normalmente son ignoradas [2]. A pesar de no ser la finalidad de las políticas de privacidad, en ocasiones parecen ser un canal orientado a las autoridades de control, para demostrar el cumplimiento de requerimientos legales más que estar orientadas a los usuarios finales. La AEPD tiene publicado un decálogo que sirve de guía para la elaboración de políticas de privacidad correctas y una guía para el cumplimiento del deber de informar bajo el marco que establece el RGPD, este último documento plantea la posibilidad de proporcionar al usuario o interesado la información necesaria mediante capas en las que se detalle la necesaria información para abordar los requisitos del RGPD.

- Avisos/notificaciones: Los avisos o notificaciones son cuadros de diálogo que incluyen textos cortos para informar a los usuarios acerca del tratamiento de recursos/datos que podrían resultar sensibles a la privacidad. Por ejemplo, el acceso a la localización geográfica requiere el consentimiento explícito del usuario. Aunque los avisos de privacidad son el canal de información más directo, aún existen aspectos de usabilidad que deben ser mejorados [3].

## IV. TÉCNICAS DE DETECCIÓN DE FLUJOS DE DATOS

Existen tres grupos de técnicas para la detección y caracterización de flujos de datos en programas software: análisis estático del programa, análisis dinámico del proceso en ejecución, y análisis de las comunicaciones. En ocasiones, las última se considera dentro del análisis dinámico, ya que requiere la ejecución del programa.

El análisis estático toma como entrada el código fuente o intermedio de un programa, lo examina sin ejecutarlo, y realiza una aproximación de "todos" los posibles flujos de ejecución o del conjunto de posibles valores calculados en diferentes puntos del programa. Por otro lado, el análisis dinámico permite analizar el comportamiento de la aplicación durante su ejecución real. Esto se consigue generando un conjunto finito de eventos que estimulan el programa, capturan y almacenan los registros generados, y en base a ellos se realiza la evaluación de las propiedades de interés. El análisis de las comunicaciones pone el foco en las comunicaciones realizadas, analizando tanto los metadatos (p.ej. destinatarios) como los datos transmitidos.

Estas técnicas son usadas para la detección de fugas de datos y poseen características distintas respecto a las propiedades de completitud[10] y consistencia[11]. Por un lado, la principal ventaja del análisis estático es la completitud respecto a los flujos detectados, dado que teóricamente es capaz de detectarlos a todos (alta completitud). No obstante, la principal desventaja es que no es preciso y puede incurrir en falsos positivos (baja consistencia), por ejemplo, porque haya instrucciones que nunca se ejecuten. Por otro lado, una de las principales ventajas del análisis dinámico es la baja tasa de falsos positivos (alta consistencia), puesto que el análisis no se basa

[10] Completitud (*completeness*) en nuestro contexto significa que teóricamente todos los potenciales flujos/fugas son detectados (es decir, no hay falsos negativos). Aunque en ese esfuerzo se puedan generar falsos positivos.

[11] Consistencia (*soundness*) en nuestro contexto significa que todas las potenciales fugas detectados son verdaderamente fugas (es decir, no hay falsos positivos). Aunque aquello implique que se puedan generar falsos negativos.

en flujos de ejecución inferidos sino en flujos que efectivamente están ocurriendo. No obstante, el análisis dinámico podría incurrir en falsos negativos, si por algún motivo no se ejecutan todos los flujos posibles (baja completitud). Por ello, el análisis dinámico sólo puede ofrecer indicación del límite inferior de las fugas.

Este desequilibrio entre completitud y precisión de las técnicas de análisis estático y dinámico ha sido claramente señalado en la literatura, razón por la cual resulta conveniente emplear, para el análisis, enfoques que combinen ambas técnicas.

A continuación, se describen las principales técnicas para análisis estático y dinámico, junto con algunas herramientas que permiten su aplicación en plataformas Android.

### TÉCNICAS DE ANÁLISIS ESTÁTICO DE CÓDIGO

En el análisis estático se estudia el código del programa para identificar las fuentes y sumideros de datos personales, e inferir todos los posibles caminos de ejecución y flujos de esos datos, construyendo modelos de estado del programa y determinando todos los estados posibles. Debido a que existen múltiples posibilidades de ejecución, se opta por construir un modelo aproximado de los estados del programa. La consecuencia de tener un modelo aproximado es la pérdida de información y de precisión en el análisis, pero cuenta con la ventaja de que sus resultados son generalizables, porque el modelo construido representa una descripción del comportamiento del programa, independientemente de las entradas y el contexto en que este se ejecute.

#### Identificación de fuentes y sumideros

En nuestro contexto, las *fuentes* de información personal son aquellos canales que el desarrollador tiene a su alcance para el acceso a los datos personales, mientras que los *sumideros* de información personal son los canales que podrían filtrar datos personales fuera del dominio de la aplicación. Por ejemplo, en una aplicación que accede a la lista de contactos del dispositivo móvil y la envía a un servidor externo, la *fuente* es el fragmento de código que lee la lista de contactos y el sumidero es el fragmento de código que la envía al servidor. La efectividad del análisis estático está significativamente determinada por la completitud de la lista de fuentes y sumideros identificados, puesto que, por ejemplo, cada método de fuente o sumidero no identificado puede dar lugar a falsos negativos (no se estaría considerando en el análisis).

Existen varias alternativas para la identificación de las fuentes y sumideros incluidas en el código de un programa, siendo las más interesantes:

- Análisis de permisos: En Android, las llamadas a los métodos de la API son el principal mecanismo a través del cual una aplicación accede a recursos que pueden ser considerados fuentes o sumideros. Si el recurso es considerado sensible, entonces requiere el permiso del usuario, y se debe declarar en el fichero MANIFEST.xml. Por ejemplo, si una aplicación requiere obtener los datos de localización del dispositivo móvil, necesita el permiso ACCESS_FINE_LOCATION o ACCESS_COARSE_LOCATION, y si requiere enviar un SMS necesita el permiso SEND_SMS. En la actualidad, existen varios conjuntos de datos y herramientas que proporcionan la correspondencia entre permisos y métodos que los requieren. Así, conocidos los permisos solicitados por una aplicación se puede determinar qué métodos de la API de

Android requieren esos permisos, e intentar localizarlos en el código de la aplicación. Sin embargo, esta técnica presenta dificultades ya que:

1. la API de Android se actualiza periódicamente, dejando obsoletas las correspondencias existentes, y
2. existen datos que, en conjunto con otros, pueden ser sensibles la privacidad y no requieren permisos de acceso.

- Uso de aprendizaje automatizado: Este enfoque emplea técnicas de aprendizaje automatizado (*machine learning*) para, a partir de una lista inicial de fuentes y sumideros, identificar otros métodos similares. Su principal ventaja es que puede cubrir un mayor rango de fuentes y sumideros ya que no se centra únicamente en aquellos que requieren permisos. Además, puede extenderse para cubrir nuevas versiones de Android.

**Herramientas de identificación de fuentes y sumideros**

Una lista, no exhaustiva, de herramientas disponibles es la siguiente:

- **PScout**

PScout [4] genera una correspondencia entre las llamadas a la API de Android y los permisos, en tres fases: identificación de permisos del APK, generación de grafos de llamadas y análisis de alcance. Se parte del supuesto de que la mayoría de los métodos que dan acceso a información sensible a una aplicación están protegidos por permisos. No obstante, se debe tener en cuenta que existen recursos sensibles a la privacidad que pueden ser accedidos sin permisos.

- **AndroidLeaks**

AndroidLeaks [5] presenta otro enfoque para identificar fuentes y sumideros a través de los permisos Android, muy similar a PScout. Si bien este enfoque es trivial de implementar para las fuentes, es más complicado para los sumideros, por lo que finalmente AndroidLeaks recurre a una lista de sumideros elaborada manualmente.

- **Susi**

Esta herramienta [6] utiliza aprendizaje automatizado para identificar fuentes y sumideros. Además, clasifica los métodos en diversas categorías, según el tipo de información sensible que manejan o la forma en que se transmite. Tras la categorización, el hecho de que todas las categorías contienen más de un método muestra que a menudo hay más de una forma de recuperar un determinado dato, y que existen múltiples formas de transmisión.

- **Merlin**

Merlín [7] utiliza aprendizaje automatizado para encontrar fuentes y sumideros no identificados. Según el documento original, la tasa de falsos positivos de Merlín es del 6% para las fuentes y del 26% para los sumideros. Este enfoque, sin embargo, sólo puede identificar aquellas fuentes y sumideros que se utilizan en al menos una de las aplicaciones del conjunto de análisis, pasando por alto los métodos más raramente utilizados.

### Análisis estático de flujo de información

Una vez identificadas las fuentes y sumideros potenciales, el siguiente paso tiene por objetivo detectar, mediante análisis estático de flujo, cuáles están "conectados". Es decir, determinar si los datos personales obtenidos de una fuente alcanzan un sumidero y realmente se produce una filtración.

Las principales técnicas de análisis estático incluyen:

- Análisis de flujo de control: Modela el programa como un grafo de control de flujo (CFG - *Control Flow Graph*), donde cada nodo representa un bloque básico de código (sentencia o instrucción) y cada vínculo entre ellos indica un posible flujo de control entre dos nodos. El objetivo es encontrar todos los caminos teóricos de ejecución de un programa.
- Análisis de flujo de datos: Permite calcular el conjunto de valores posibles que un programa maneja en un punto determinado de ejecución (DFG – *Data Flow Graph*). Está técnica se apoya en el análisis de flujo de control, dado que para determinar el valor de las variables en un punto dado se requiere conocer con certeza el orden de las operaciones ejecutadas por el programa (es decir, su grafo de flujo de control). Por otro lado, la efectividad de un análisis de control de flujo aumentará en la medida en que se conozca con exactitud los valores de las variables, ya que de ellos depende el resultado de las sentencias de control.
- Análisis de marcado (taint analysis): Es un tipo especial de análisis de flujo de datos que realiza el seguimiento de información a lo largo del camino de la ejecución del programa. Los datos de interés son marcados con un tipo (comúnmente denominado mancha o *tag*) en la fuente, y se propaga a través de todas las rutas de ejecución del programa, para ver si desemboca en algún destino sumidero. Es decir, si además se asocian niveles de sensibilidad (de privacidad) tanto a las fuentes como a los sumideros, se puede detectar si información privada puede alcanzar lugares públicos o lugares no deseados. Normalmente requiere anotaciones el código para indicar las variables a marcar.

### Herramientas para análisis de flujo

Al igual que en el caso de las herramientas de análisis estático, se presenta una lista no exhaustiva de herramientas:

- **Soot**

Soot [8] genera código intermedio para código Java y código ejecutable Android. Este código intermedio ha sido creado específicamente para facilitar el análisis estático de código, por lo tanto permite llevar a cabo un amplio conjunto de operaciones (ej. la creación de grafos de llamadas).

- **FlowDroid**

FlowDroid [10] es una herramienta de código abierto para análisis estático de aplicaciones Android (también de Java), en concreto de análisis de marcado. Flowdroid reduce el programa a una representación intermedia que modela el ciclo de vida de los

componentes de las aplicaciones de Android. Flowdroid puede detectar únicamente flujos de datos intra-procesales.

- **Epicc**

Epicc [11] complementa a Flowdroid mediante la detección de flujos inter-procesales.

- **DidFail**

DidFail [12] combina FlowDroid y Epicc para rastrear los flujos de datos de un conjunto de aplicaciones tanto entre componentes como dentro de cada componente. DidFail tiene dos fases en el análisis: primero, determina el flujo de datos de cada aplicación y las condiciones bajo las cuales son posibles; segundo, basándose en los resultados de la primera fase, enumera los flujos de datos potencialmente peligrosos habilitados por las aplicaciones en su conjunto.

- **IccTa**

IccTA [13] realiza el análisis en una sola fase. Tiene mayor precisión que DidFail porque tiene mayor sensibilidad al contexto y realiza menos sobreestimaciones de los datos marcados que alcanzan sumideros. IccTA y DidFail son muy similares y fueron desarrollados simultáneamente, pero son proyectos independientes.

- **CHEX**

CHEX [14] detecta vulnerabilidades en el flujo de la información entre componentes.

- **LeakMiner**

Similar a Flowdroid, Leakminer [15] se basa en Soot para la generación de los grafos de llamadas e implementa el ciclo de vida de los componentes de Android. Aunque esta herramienta puede analizar una aplicación en un par de minutos, el análisis no es sensible al contexto, lo que da lugar a una alta tasa de falsos positivos.

- **AndroidLeaks**

AndroidLeaks [5] realiza análisis de marcado. Sin embargo, es poco preciso en el análisis puesto que no refina su sensibilidad a la hora de marcar los datos sensibles, dando lugar a un elevado número de falsos positivos.

- **ScanDroid**

ScanDroid [16] se centra en el análisis del flujo entre componentes y el flujo de información entre aplicaciones, lo que plantea el reto de relacionar los componentes con sus respectivos receptores en otras aplicaciones.

- **DroidSafe**

DroidSafe [17] es una herramienta que permite realizar análisis estático de flujos de datos en aplicaciones Android. Lo que cabe destacar de este trabajo es que presenta

una tasa de detección mayor en comparación con las herramientas anteriores, en base al banco de pruebas BenchDroid.

- **JoDroid**

JoDroid [18] es una extensión a la herramienta de análisis JOANA para soportar análisis de aplicaciones Android, usando técnicas de análisis de flujo de datos y técnicas de análisis de control de flujo. El análisis parte de anotaciones en el código fuente de la aplicación.

### TÉCNICAS DE ANÁLISIS DINÁMICO

Las técnicas de análisis dinámico permiten analizar el comportamiento de una aplicación durante su ejecución real. Esto se consigue generando un conjunto finito de eventos que estimulan el programa, inspeccionando el comportamiento del programa durante la ejecución, y almacenando los registros necesarios para su análisis y evaluación. Se pueden distinguir las siguientes categorías significativas para nuestros propósitos:

- Marcado (*tanto analysis*): Esta técnica etiqueta/marca los datos que provienen de distintas fuentes y transitivamente aplica etiquetas a medida que los mismos se propagan a lo largo de las variables de la aplicación, archivos y mensajes entre procesos. Cuando los datos marcados son transmitidos fuera del dominio de la aplicación (ej. son enviados a través de Internet), se registran estos datos junto con otros de interés, como el destino de los datos. La herramienta más representativa del análisis dinámico de marcado es TaintDroid [19] que realiza un marcado a nivel variable, método, mensaje y archivo. La principal ventaja de este enfoque es su consistencia para detectar fugas de datos. Sin embargo, tiene las siguientes desventajas: es vulnerable a los ataques de control de flujo (es decir, las aplicaciones pueden usar flujos implícitos para filtrar información) por tanto puede incrementar el número de falsos positivos; se requiere un tiempo considerable para analizar una aplicación por lo tanto no es adecuado para evaluar aplicaciones a gran escala; finalmente, dado que se requiere modificar el sistema operativo puede haber aplicaciones incompatibles.
- Monitorización de acceso a recursos con acceso a datos personales: Esta técnica instrumenta el sistema operativo Android para permitir la monitorización en tiempo real del acceso de las aplicaciones a los recursos relacionados con aspectos de la privacidad. No se modifican las aplicaciones, de esta manera se puede usar un dispositivo móvil con el sistema operativo instrumentado para monitorizar cualquier aplicación. Por ejemplo, se puede instrumentar todos los métodos fuente/sumidero que acceden a los recursos sensibles, de forma que cada vez que una aplicación acceda a estos recursos se almacenarán registros para su posterior análisis. Las principales ventajas de este enfoque radican en que (1) no se requiere la modificación de las aplicaciones a evaluar y (2) permite detectar los métodos fuente/sumidero que realmente acceden a recursos sensibles. Por otro lado, el principal inconveniente es que se requiere un tiempo considerable para analizar una aplicación, por lo que no es adecuado para evaluar aplicaciones a gran escala.

### Técnicas de generación de eventos

Dado que el análisis dinámico se basa en la ejecución real de la aplicación, la interacción de un usuario con la aplicación o una emulación de ella es imprescindible. Esta interacción se logra por medio de la manipulación (real o no) de dichas aplicaciones, para lo cual existen enfoques manuales y automáticos.

- Manuales: Dentro de las técnicas manuales existen dos opciones: (1) un usuario utiliza la aplicación a evaluar y se graba para poder repetirlo, o (2) emplear un grupo de personas que será grabada durante el uso de la aplicación. La primera se puede llevar a cabo haciendo uso de la herramienta *Selendroid* [19]. La segunda se podría llevar a cabo mediante crowdsourcing, pero requiere adquirir y adecuar los dispositivos con las aplicaciones de interés.
- Automáticas: Las técnicas automáticas se basan principalmente en la generación (pseudo) aleatoria de eventos que emulan la interacción de un usuario con la aplicación. La plataforma de desarrollo para Android ya cuenta con un programa llamado Exerciser Monkey [20] que realiza esta función. La aplicación permite tener cierto control sobre los eventos que se generan y es posible replicarlos ya que usa un algoritmo pseudo aleatorio. No obstante, tiene el inconveniente que los eventos generados por esta herramienta difieren en cierta medida de los eventos de interacciones reales que realizaría un humano.

Los enfoques manuales sólo se deben considerar cuando el número de aplicaciones a analizar es reducido. Por el contrario, su coste es más alto, tanto en tiempo como en recursos, en comparación con la ejecución automática y (pseudo) aleatoria. Cuando el número de aplicaciones a analizar crece, tanto el crowdsourcing como la ejecución aleatoria presentan grandes beneficios respecto de la ejecución manual grabada, por lo que la decisión deberá realizarse entre ambas.

Además, las técnicas manuales son más efectivas a la hora de generar eventos que se aproximen al comportamiento real de un usuario manipulando la aplicación, pero son muy poco escalables. En cambio, las técnicas automáticas son escalables, pero a su vez son capaces de generar muchos menos eventos de actividades reales ya que las acciones se realizan de manera (pseudo) aleatoria.

### Técnicas de análisis de tráfico

El análisis de tráfico permite examinar las comunicaciones llevadas a cabo por una aplicación, para determinar si en ellas se está transmitiendo algún tipo de información personal, y las características de la transmisión (destinatario, lugar del mundo, etc.). Para el análisis de tráfico serán necesarias cuatro fases: la generación de eventos en las aplicaciones a evaluar que a su vez generen tráfico, la intercepción de este tráfico y su descifrado en aquellos casos que sea necesario, y el análisis propiamente dicho de la información que está siendo transmitida. Mientras las técnicas de generación de eventos ya fueron descritas en la sección anterior, las técnicas para las tres fases restantes se describen en las siguientes subsecciones.

### Técnicas de intercepción de tráfico

La Tabla 1 muestra las dos técnicas de intercepción de tráfico comúnmente empleadas en el contexto de las aplicaciones móviles junto con los parámetros de interés evaluados. Estas son descritas en más detalle en las siguientes secciones.

| Técnica | Extensible | Infraestructura extra | Aumento de latencia |
|---|---|---|---|
| **VPN** | No | No | No |
| **Proxy** | Si | Si | Si |

Tabla 1. Técnicas de intercepción de tráfico

Una VPN (*Virtual Private Network*), red privada virtual, permite la creación de una red privada sobre una red pública mediante el establecimiento de túneles virtuales en el establecimiento de las conexiones [21]. Para el uso de una VPN en Android se ofrece una API que permite la captura de tráfico a nivel IP [21] [22].

Al establecerse la VPN en el propio dispositivo móvil no es necesario desplegar una infraestructura extra ni configurar la red o el dispositivo móvil para que el tráfico sea capturado, por lo cual no tendrá asociado necesariamente un aumento de la latencia. No obstante, tiene el inconveniente que no es extensible, ya que la implementación de la VPN dependerá del dispositivo móvil y en concreto de la API que ofrece Android.

Un proxy es un sistema que funciona como intermediario entre otros dos sistemas en red, recibiendo los paquetes enviados por el origen y retransmitiéndolos al destino. La implementación del proxy es independiente del dispositivo móvil que esté conectado a la red, aunque es necesario configurar la red o el dispositivo móvil de forma que todo el tráfico fluya por el proxy. También tendrá asociado un aumento de la latencia en las conexiones.

Por otro lado, dado que la implementación del proxy es independiente del dispositivo móvil, esta es una solución muy extensible a dispositivos móviles con versiones de Android también diferentes. En caso de desear interceptar el tráfico de otro dispositivo móvil distinto solo es necesario asegurarse que su tráfico fluye por el proxy, sin cambiar la implementación.

### Técnicas de descifrado de tráfico

La Tabla 2 muestra las técnicas empleadas para descifrar el tráfico, indicando cualitativamente la dificultad de implementación y la factibilidad de implementarlo.

| Técnica | Dificultad | Disponibilidad |
|---|---|---|
| **Suplantación de certificados** | Fácil | 4 < Android < 7 |
| **Intercepción DH_anon** | Medio | Improbable |

| | | |
|---|---|---|
| **Edición de APK** | Difícil | Si |

Tabla 2. Técnicas de descifrado de tráfico

La manera más sencilla de interceptar tráfico cifrado (TLS/SSL) es mediante la **suplantación de certificados**. En concreto, se instala un certificado raíz emitido por una Autoridad de Certificación propia (CA – *Certification Authority*) en el dispositivo móvil que funciona como cliente, de forma que al recibir un certificado falso del proxy o VPN confíe en ella [21] [23] [24].

Esto funcionará siempre y cuando no se haga uso de una técnica denominada *certificate pinning* [24], en la cual la aplicación no confía en todas las autoridades de certificación (cuyos certificados raíz están instalados en el dispositivo móvil), sino que preselecciona una o un conjunto específico de ellas. A partir de Android 7 la configuración estándar de las aplicaciones hace uso de *certificate pinning* [25] [26].

Otra forma de interceptar tráfico TLS/SSL es modificando el inicio del acuerdo (*handshake*) entre cliente y servidor de forma que hagan uso de la técnica conocida como ***DH_anon*** para el acuerdo de las claves que usarán en la comunicación. Este método es vulnerable a un ataque por *Man in the Middle* dado que no autentica al servidor [27], pero suele estar desactivado por defecto en gran parte de los clientes TLS.

Una forma de superar el *certificate pinning* es mediante la **edición del APK**, de forma que considere que nuestra CA es una de las preseleccionadas para confiar [28] [29]. Este método es muy efectivo, ya que ante la modificación del código el desarrollador de la aplicación no tiene defensas posibles, pero es notablemente más complejo que las soluciones anteriores.

### Técnicas de análisis de la información

Es posible distinguir entre tres categorías principales de datos que pueden obtener a través de la captura y análisis del tráfico. Estas categorías corresponden más o menos al lugar del paquete en el cual vayan dichos datos al transmitirlos. Dichas categorías son: información respecto del dominio o dirección IP (*Internet Protocol*) a la que la petición va dirigida, información en las cabeceras del paquete, e información en el cuerpo del paquete.

El dominio al cual va dirigida la petición no suele aportar excesiva información, pero puede dar pistas sobre el comportamiento de la aplicación, especialmente en cuanto a la privacidad de la misma. Ciertos dominios corresponden a empresas de anuncios o de *trackers*, lo que implicaría que una petición a uno de dichos dominios corresponde a cierto grado de vulneración de la privacidad del usuario. Para un análisis de este entorno ver la referencia [23].

Las cabeceras HTTP también son un elemento a través del cual es posible transferir información del usuario, y por lo tanto es necesario analizarlas. Un ejemplo podrían ser las cookies que permitiesen rastrear a un usuario a través de varios dominios.

Finalmente, la mayoría de la información transmitida será enviada a través del cuerpo de las peticiones HTTP, entendiendo que las *queries* de las peticiones GET son un paralelo al cuerpo en las peticiones PUT y POST. De esta forma es posible la transmisión de una gran cantidad de datos, y de características muy variadas.

Es posible enviar información multimedia, conjuntos de bytes codificados de múltiples formas, o cualquier información del dispositivo móvil que se pueda representar en texto plano. Cada una de estas tres subcategorías presentan dificultades propias para su detección y análisis.

La información multimedia se detectará buscando cabeceras y números mágicos correspondientes a codificaciones conocidas de audio, video e imagen. Esta técnica puede producir falsos negativos en el caso de que la codificación usada no haya sido contemplada, que no contenga un número mágico o cabecera reconocible, o que el paquete haya sido troceado de manera apropiada [24].

La información enviada en texto plano es la más sencilla de detectar y analizar. En caso de contar con la información a la que las fuentes de la aplicación son capaces de acceder es trivial buscar en cada petición si dicha información está contenida en su cuerpo.

Por otro lado, el envío de la información como conjuntos de bytes genera las mayores complicaciones para el análisis. Por un lado, dichos datos pueden encontrarse cifrados o codificados, por otro lado, no se cuenta con ninguna referencia respecto de qué pueden significar dichos datos.

### Herramientas

La Tabla 3 muestra un resumen de las herramientas más significativas que implementan una o varias de las técnicas descritas en las subsecciones anteriores.

| Herramienta | VPN | Proxy | Propio CA | DH_anon | Edición de apk | Rehacer stack TCP | Intercepción TLS |
|---|---|---|---|---|---|---|---|
| **Android VPN** | Si | No | NI | NI | No | Si | NI |
| **Mitmproxy** | No | Si | Si | NI | No | No | Si |
| **Tcpdump** | No | Si | No | No | No | Si | No |
| **Ssldump** | No | Si | Si | NI | No | Si | Si |
| **Meddle** | Si | No | - | - | - | - | - |
| **WireShark** | No | - | - | - | No | No | No |
| **Frida** | No | No | No | No | Si | - | - |
| **Apktool** | No | No | No | No | Si | - | - |

Tabla 3. Herramientas para análisis de tráfico (*NI = Necesario implementarlo*)

- AndroidVPN es la solución más usada por las distintas aplicaciones de análisis de tráfico para teléfonos móviles Android [21] [22] [23] [24] [30]. Es necesario

instalar un certificado raíz emitida por una CA propia en el dispositivo móvil para que éste confíe en la VPN. Todas aquellas aplicaciones desarrolladas para el nivel de API 24 (Android 7) no confiarán en el certificado a menos que se modifique la aplicación o el conjunto de certificados estándar de Android [25] [26].

Esta herramienta acarrea una serie de inconvenientes: 1) no es extensible para dispositivo móviles que corran un sistema operativo diferente de Android; 2) se ejecuta en el dispositivo móvil del usuario, evitando que use otros VPNs y consumiendo su batería; 3) no es muy estable, ya que depende fundamentalmente de Android, que podría terminar el proceso subyacente, y; 4) es necesario rehacer la pila de protocolos HTTP/TCP ya que los paquetes entregados a la VPN son de la capa 3 [31].

También tiene una serie de ventajas. 1) Si la confianza en el certificado no es un problema resulta muy fácil de usar 2) No requiere de ninguna complicación con infraestructura extra ya que corre en el propio dispositivo móvil 3) Asegura que se atrapa todo el tráfico que sale del dispositivo móvil.

- Mitmproxy es una herramienta de intercepción de tráfico HTTP y HTTPS madura y extensible [32]. Es necesario instalar un certificado raíz de una CA propia en el dispositivo móvil para que éste confíe en el proxy [33]. Todas aquellas aplicaciones desarrolladas para el nivel de API 24 (Android 7) no confiarán en el certificado a menos que se modifique la aplicación o el conjunto de certificados estándar de Android [25] [26].

  Tiene el inconveniente que al ser un proxy externo al dispositivo móvil es necesario diseñar una red que asegure que todo el tráfico del móvil pasa a través del proxy [34]. Por otro lado, tiene una serie de ventajas: 1) si la confianza en el certificado no es un problema resulta muy fácil de usar; 2) no es necesario rehacer la pila HTTP/TCP, y; 3) permite separar el dispositivo móvil de la intercepción, por lo cual esta no depende del sistema del primero.

- Tcpdump es una herramienta muy utilizada para el análisis de paquetes en un sistema [35]. No permite el descifrado de tráfico HTTPS por lo cual ese proceso debería hacerse usando otros mecanismos. Su principal ventaja es el amplio uso de la herramienta y por ende la amplia documentación existente.

  No obstante, conlleva una serie de inconvenientes: 1) es necesario diseñar una red que asegure que todo el tráfico del móvil pasa a través del proxy y 2) es necesario rehacer la pila HTTP/TCP ya que los paquetes entregados son de la capa 3.

- Ssldump [36] es una herramienta hermana de tcpdump. La diferencia principal es que sí permite el descifrado de tráfico TLS/SSL, pero también necesita que el cliente acepte su certificado al igual que las dos primeras herramientas.

- WireShark es un popular analizador de tráfico con una interfaz gráfica que puede usarse para capturar el tráfico en vivo o analizar el resultado de una captura anterior almacenada en un fichero PCAP [37]. En ventajas e inconvenientes es similar a tcpdump, con la diferencia de que no es necesario rehacer la pila TCP y añadiendo el uso de una GUI, por lo que es más complicado de automatizar.

- Meddle es una VPN para Android que principalmente está enfocado para su uso con la aplicación ReCon. No se encuentra disponible para su uso en aplicaciones desarrolladas por personas ajenas a ReCon [38]. En ventajas e inconvenientes es similar a AndroidVPN con la salvedad de que no es necesario rehacer la pila TCP/HTTP.
- Frida es una herramienta que permite inyectar código en aplicaciones para una gran variedad de sistemas operativos. En concreto podría ser usada para sobrepasar el *pinning* de certificados que implementen algunas de las aplicaciones [39].

  Para las aplicaciones desarrolladas para el nivel de API 24 o superior no será necesario utilizar Frida ya que, en teoría, modificando el fichero MANIFEST.xml de la aplicación se podría sobrepasar sin problemas. Para otras aplicaciones puede ser necesario utilizar Frida, cuando el *pinning* de certificados venga implementado directamente en la aplicación.
- Apktool es una herramienta de desensamblaje para aplicaciones Android [40], que sigue el empaquetamiento APK. Es muy útil para descompilar una aplicación y poder modificarla o analizar su código. Se puede usar en conjunto con Frida.

## V. CONCLUSIONES

Este estudio muestra la existencia de un elevado número de posibles flujos de datos personales en las aplicaciones móviles, lo que implica un riesgo potencialmente alto de comunicación a terceros de datos personales sin el conocimiento del propio usuario o titular de los datos (el interesado). Esto es posible debido a la alta disponibilidad de sensores incorporados en los dispositivos, así como el elevado número de identificadores únicos que son intrínsecos al propio dispositivo, que facilitan la recopilación de datos personales directamente atribuibles al usuario del dispositivo móvil. Además, la gran capacidad de conectividad de los dispositivos móviles y la gran variedad de agentes que intervienen en el entorno de aplicaciones móviles no hacen sino aumentar este riesgo.

Los desarrolladores de aplicaciones móviles, los responsables que subcontraten dichos desarrollos y distribuidores o repositorios de apps tienen una obligación, la de responsabilidad proactiva señalada en el RGPD, para asegurar que sus productos y servicios cumplen con la protección de los datos de carácter personal establecida en la normativa. Por lo tanto, han de llevar a cabo análisis y/o auditorías, utilizando herramientas y metodologías como las presentadas en este estudio, para determinar que las apps que están poniendo a disposición de los usuarios están de acuerdo a las políticas de privacidad, y cualquier otra información que se facilita a través de descripciones textuales, avisos y notificaciones, con las garantías adecuadas que exige el RGPD y las normativas que adaptan el derecho español al RGPD, como es el caso de la LOPDGDD[12].

Existe un conjunto elevado de técnicas y herramientas para determinar si una aplicación recoge y transmite información personal. Como se recoge en el estudio, es necesario combinar los resultados de todas ellas para conseguir un análisis efectivo.

[12] Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales

Los flujos potenciales de datos personales detectados mediante técnicas de análisis estático deben confirmarse con herramientas de análisis dinámico que permitan interceptar y analizar el tráfico real de la aplicación. Tanto las herramientas de análisis estático como las herramientas de análisis dinámico analizadas son herramientas disponibles libremente para cualquier usuario.

Como herramienta de identificación de fuentes y sumideros, se concluye [41] que Susi es capa de localizar no solo todas las fuentes y sumideros encontradas por otras herramientas, sino que encuentra otras nuevas que hasta el momento habían sido pasadas por alto. La herramienta de análisis de flujo Soot permite realizar el grafo de llamadas de las aplicaciones y con FlowDroid se pueden detectar las conexiones entre fuentes y sumideros.

Para las investigaciones en las que la ejecución manual no sea la opción óptima, se puede utilizar Exerciser Monkey, que permite automatizar ejecuciones pseudoaleatorias. En otros trabajos de investigación también se ha mostrado la eficacia del crowsourcing, en cuanto a la capacidad de analizar un elevado conjunto de aplicaciones en distintos dispositivos y utilizados por usuarios reales.

En cuanto a la intercepción del tráfico, el uso de un proxy aporta la ventaja de separar físicamente la fase de ejecución de la de intercepción, dando de esta manera la posibilidad de utilizar otras herramientas de seguridad fuera del entorno móvil.

En la realización del presente documento ha participado la AEPD y José María del Álamo, Danny Santiago Guamán, Lucas María Tomé y Esperanza Zamora por la Universidad Politécnica de Madrid.

## VI. REFERENCIAS

[1] F. Schaub, R. Balebako, and L. F. Cranor, «Designing Effective Privacy Notices and Controls,» *IEEE Internet Computing,* vol. 21, nº 3, p. 70–77, 2017.

[2] A. Felt, E. Ha, S. Egelman, and A. Haney, «Android permissions: User attention, comprehension, and behavior,» *Proc. of SOUPS,,* p. 1–14, 2012.

[3] T. Watanabe, M. Akiyama, T. Sakai, and H. Washizaki, «Understanding the Inconsistency between Behaviors and Descriptions of Mobile Apps,» nº 11, p. 2584–2599, 2018.

[4] K. Z. Y. H. Z. &. L. D. Au, «PScout: analyzing the Android permission specification.,» de *ACM Conference on Computer and Communications Security.*, 2012.

[5] C. &. C. J. &. E. J. &. C. H. Gibler, «AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale,» de *Lecture Notes in Computer Science*, 2012, pp. 291-307.

[6] S. A. a. E. Bodden., «Reviser: efficiently updating IDE-/IFDS-based data-flow analyses in response to,» de *Proceedings of the 36th International Conference on Software Engineering*, 2014.

[7] B. &. N. A. &. R. S. &. B. A. Livshits, «Merlin: Specification Inference for Explicit Information Flow Problems,» 2009.

[8] Lam, Patrick & Bodden, Eric & Lhoták, Ondvrej & Hendren, Laurie, «The Soot framework for Java program analysis: a retrospective,» 2011.

[9] T. Watson, «Watson libraries for analysis,» [En línea]. Available: http://wala.sourceforge.net/wiki/index.php/Main_Page.

[10] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Damien Octeau, Patrick McDaniel, and Yves Le Traon, «Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps.,» Department of Computer Science and Engineering, 2014.

[11] P. M. S. J. A. B. E. B. D. Octeau, «Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis,» de *Proceedings of the 22nd USENIX Security Symposium (USENIX Security 13)*, 2013.

[12] L. F. W. K. J. L. W. S. a. W. S. Jonathan Burket, «Making didfail succeed: Enhancing the cert static taint analyzer for android app sets.,» CERT Division, 2015.

[13] A. B. J. K. Y. L. T. S. A. S. R. E. B. D. O. a. P. M. L. Li, «I know what leaked in your pocket: Uncovering privacy leaks on android apps with static taint analysis.,» 2014.

[14] L. Lu et al., «Chex: Statically vetting android apps for component hijacking vulnerabilities.,» 2012.

[15] Z. Yang and M. Yang., «LeakMiner: Detect Information Leakage on Android with Static Taint Analysis,» *Software Engineering (WCSE), 2012 Third World Congress on,* pp. 101-104, 2012.

[16] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, «Scandroid: Automated security certification of android applications.,» 2010.

[17] Michael I. Gordon, Deokhwan Kim, Jeff Perkins, Limei Gilham, Nguyen Nguyen, and Martin Rinard., «Information-flow analysis of android applications in DroidSafe,» Massachusetts Institute of Technology, 2015.

[18] J. Graf, M. Hecker, and M. Mohr, «Jodroid: Adding android support to a static information flow control tool,» de *Proceedings of the 8th Working Conference on Programming*

*Languages*, 2015.

[19] «Selendroid,» [En línea]. Available: http://selendroid.io/.

[20] «UI/Application Exercise Monkey,» [En línea]. Available: https://developer.android.com/studio/test/monkey.

[21] SONG, Yihang; HENGARTNER, Urs., «Privacyguard: A vpn-based platform to detect information leakage on android devices.,» de *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2015.

[22] LE, Anh, et al., «AntMonitor: A system for monitoring from mobile devices,» de AntMonitor: A system for monitoring from mobile devices. EnProceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data, 2015.

[23] RAZAGHPANAH, Abbas, et al, «Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem,» 2018.

[24] PAN, Elleen, et al, «Panoptispy: Characterizing Audio and Video Exfiltration from Android Applications,» *Proceedings on Privacy Enhancing Technologies,* vol. 2018, nº 4, 2018.

[25] «Network Security Configuration,» [En línea]. Available: https://developer.android.com/training/articles/security-config.

[26] «Intercepting HTTPS traffic of Android Nougat Applications,» [En línea]. Available: https://serializethoughts.com/2016/09/10/905/.

[27] «RFC 5246 Anonymous Key Exchange,» [En línea]. Available: https://tools.ietf.org/html/rfc5246#appendix-F.1.1.1.

[28] «Prevent bypassing of SSL certificate pinning in iOS applications,» [En línea]. Available: https://www.guardsquare.com/en/blog/iOS-SSL-certificate-pinning-bypassing.

[29] «Bypassing Certificate Pinning on Android for fun and profit,» [En línea]. Available: https://medium.com/@felipecsl/bypassing-certificate-pinning-on-android-for-fun-and-profit-1b0d14beab2b.

[30] REN, Jingjing, et al, «Recon: Revealing and controlling pii leaks in mobile network traffic,» de *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, 2016.

[31] «VpnService,» [En línea]. Available: https://developer.android.com/reference/android/net/VpnService.

[32] «mitmproxy,» [En línea]. Available: https://mitmproxy.org/.

[33] «About Certificates,» [En línea]. Available: https://docs.mitmproxy.org/stable/concepts-certificates/.

[34] «Transparent Proxy,» [En línea]. Available: https://docs.mitmproxy.org/stable/concepts-modes/#transparent-proxy.

[35] «Tcpdump,» [En línea]. Available: http://www.tcpdump.org/.

[36] «Using ssldump to Decode/Decrypt SSL/TLS Packets,» [En línea]. Available: https://packetpushers.net/using-ssldump-decode-ssltls-packets/.

[37] «Wireshark,» [En línea]. Available: https://www.wireshark.org/.

[38] «Meddle,» [En línea]. Available: https://vpn.meddle.mobi/.

[39] «Using Frida to Bypass Snapchat's Certificate Pinning,» [En línea]. Available: https://labs.nettitude.com/tutorials/using-frida-to-bypass-snapchats-certificate-pinning/.

[40] «Apktool,» [En línea]. Available: https://ibotpeaches.github.io/Apktool/.

[41] S. &. A. S. &. B. E. Rasthofer, «A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks,» de *Network and Distributed System Security Symposium*, 2014.

[42] «Runtime Permissions - Android Developers,» [En línea]. Available: https://developer.android.com/distribute/best-practices/develop/runtime-permissions.

[43] «Android Developers: Permissions,» [En línea]. Available: https://developer.android.com/guide/topics/permissions/overview.

[44] J. Kim, Y. Yoon, K. Yi, and J. Shin, «Scandal: Static analyzer for detecting privacy leaks in android applications,» 2012.