ANALYSIS OF INFORMATION FLOWS IN ANDROID

TOOLS FOR COMPLIANCE WITH ACCOUNTABILITY

Date of publication: 07/03/19

CONTENTS

I. EXECUTIVE SUMMARY

Applications present on mobile phones can manage data such as photos, emails or calendar, can access certain data generated by integrated sensors in the device or connected to it, such as geo-localisation of the user's vital signs and certain identifiers used by the hardware, operating system, services and other applications, what is called the device's digital signature (see the study Fingerprint or Digital Device Fingerprint published by the AEPD). These personal data may be processed internally by the applications, although they may also be communicated internally to other applications within the same device or to external entities (e.g. a data analysis server).

The versatility of data, processing and the potential of data communications in the Android model elevates the potential risk of illegitimate exploitation of personal data by third parties.

The Data Controller for processing carried out by a mobile application is obliged to inform the user through privacy policies, notifications or descriptions published in app stores, and the effective implementation of the service must comply with the limits of this information, of the legitimacy of the processing and general GDPR guarantees. The reality is that the Data Controller for the processing of the data of an app is not always the direct and exclusive developer of same, but based on third party libraries, subcontracting or agreements and/or the execution of the third party environment, which is why there is a potential loss of control over the implementation of said processing and an increase in complexity to approach the aforementioned requirements for compliance with data protection.

The developers of mobile applications, the managers who subcontract development, distributors or repositories of apps are obliged to ensure that the apps they make available to users are in line with privacy policies and advertising services with adequate guarantees. That means applying the principle of Accountability through the application of Default Privacy measures minimising the processing of data, the extension thereof, retention and accessibility and measures of Privacy in Design, selecting those components that most respect privacy.

In order to meet these obligations, this document offers a study of the existing techniques for the analysis of personal information in mobile applications that are executed in Android operated devices. First, we look at the execution environment of these applications, their fundamental components, the different actors involved in processing the data and a brief description of the data lifer cycle. We then turn to the principal techniques and tools used for analysis of personal information flows and describe them. These include static analysis of code, analysis of execution and communication analysis.

II. INTRODUCTION

OBJECTIVES AND SCOPE

This study is conducted within a framework of collaboration between the Universidad Politécnica de Madrid (UPM) and the Spanish Data Protection Agency (AEPD), with the scope to identify the existing techniques and tools to detect personal data flows in software for mobile devices.

The objectives of the study are particularly focussed on:

- Defining the context and conceptual framework of the detection of the personal data communications in applications executed on an Android operating system.
- Demonstrating the elevated risk in the mobile application environment of leaks of personal data and the need to carry out an evaluation of data flows that takes into account the life cycle of this information and allow for the possible impact assessment of said data flow to be carried out in relation to privacy and people's right to data protection.
- Studying the existing techniques for the detection and analysis of personal information flows in Android Applications.

This document is the first outcome produced by this UPM-AEPD collaboration.

III. PRIVACY AND PERSONAL DATA PROTECTION IN ANDROID APPLICATIONS

ANDROID APPLICATION ENVIRONMENT

This section presents the principal elements and actors of the Android applications environment, locating them in the context of the privacy and protection of personal data.

Figure 1 describes the relationship between those elements and actors.

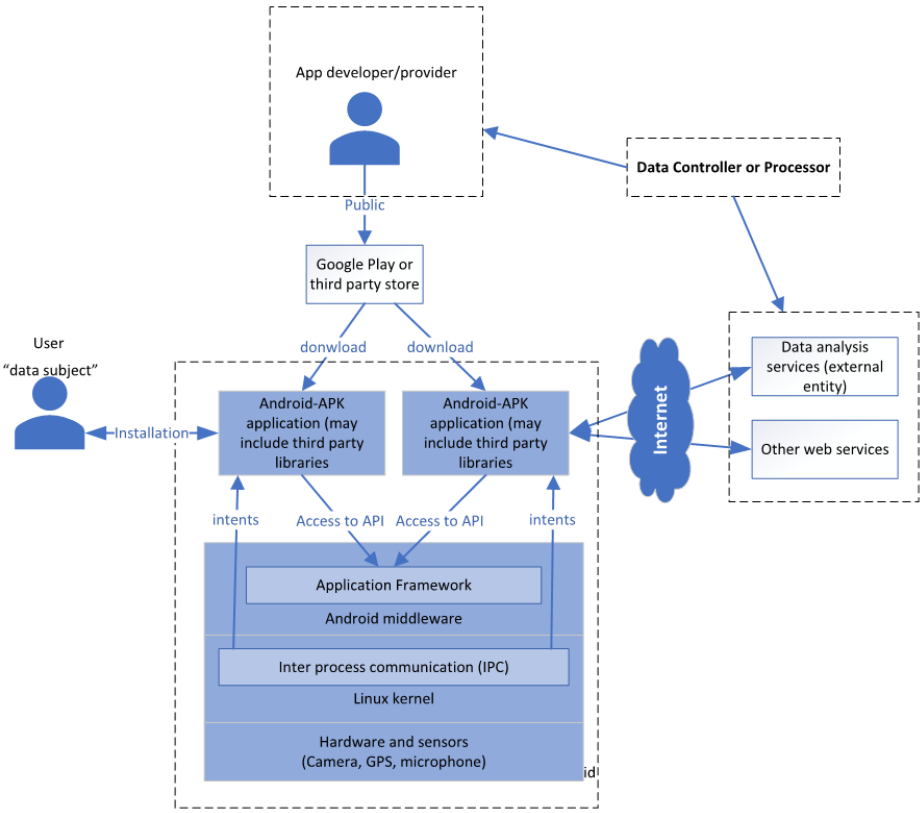

Figure 1. Android Ecosystem

Developers of Android applications primarily use Java as their programming language. To facilitate development, Android provides a set of software libraries (Android Framework) which contains the basic components for building applications and software interfaces (APIs - *Application Programming Interfaces*) for accessing the services of the operating systems (e.g. Bluetooth administration service) and the data generated by certain resources of the device (e.g. sensor data). Moreover, a large number of applications also use third party libraries with different purposes, like adding functionalities or monetizing their applications (e.g. libraries for personalised announcements).

Android Framework libraries contain four basic components for building applications[1]:

- Activities: represent the user interfaces (e.g. an interface to send an email).
- Services: allow for the execution of prolonged tasks or remote connections on background (e.g. a service for downloading files).
- Content providers: manages data persistently stored(e.g. a supplier of content administers the creation, reading, modification and elimination of registers in an internal database of the device).
- Broadcast receiver: allow messages sent by the system or other applications (e.g. receive notification that a downloaded has ended). Two applications can communicate between them (IPC - Inter Process Communication) through what are known as "intents". For example, a health and wellness application can transmit a message to another application such as a social media site that the user has run 10km in 30 minutes.

Once the development of the application is complete, these are compiled obtaining a code in DEX (Dalvik Executable) format and compressed along with other resources necessary for the execution of the APKs (Android Packages). The APKs are published in the official Google Play store[2] or third party stores such as Uptodown[3] or APK Pure[4].

Mobile phones are a source of personal data, given they are used by the user to carry out their daily activities. The applications can manage data that are personal by nature (e.g. photos, audio notes, emails, diary entries and lists of contact of a user) but are also capable of accessing certain data generated by internal (e.g. geo-localisation[5], application use logs) or external (e.g. pulse obtained using a pulse monitor) resources/sensors. Added to this is the fact that the hardware, operating system, services and applications internally use global identifiers that are capable of identifying (and tracking) the users of the devices. The General Data Protection Regulation (GDPR)[6] defines personal data as "any information concerning an identified or identifiable natural person" ("the data subject"), therefore, the mobile applications that collect and process the aforementioned data must comply with the requirements set out in this regulation.

The appropriate processing[7] of personal data requires clear definition of the main roles involved and their responsibilities. The GDPR contains the following definitions:

[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] Not only the geo-localisation obtained from GPS but also inferred from other data such as WiFi network identifiers (SSID).

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

[7] In accordance with the GDPR processing refers to any operation or set of operations using personal data (e.g. Collection, use, dissemination, storage and erasure).

- Data subject: identified or identifiable natural person whose data are subject to processing, and generally also the user of the device.
- Data Controller: the person or entity that determines the purposes of the processing of personal data and who must ensure that data protection requirements are met even when delegating a particular processing task to an external entity with previously defined purposes.
- Data processor: person or entity that processes personal data by delegation of the Data Controller and following said Controller's terms and conditions.

In the mobile ecosystem, the user of the device is the data subject and, at the same time, the user of applications with a purpose that should be in line with the purpose or purposes set by the Data Controller. The supplier of services may be the Data Controller if it is the one that determines the purpose, or may be the Data Processor if it only processes data when contracted to do so by the Data Controller. For example, in a music streaming app, the user of the app is the data subject, the organisation that operates the application is the data controller and the data processor is the provider of the cloud services where the data controller hosts their services.

PERSONAL DATA LIFE-CYCLE IN ANDROID APPLICATIONS

The phases of the life-cycle of personal information are defined generalising the components of an application like in all relations with external elements. The application is comprised of all the Android components that have been defined in a file (MANIFEST.xml) which includes the APK. This clarification is essential as, for example, a transmitter/diffuser will take place only if certain personal data have been sent outside the own components of the application being assessed. Figure 2 Illustrates the phases of the life cycle of personal data in an Android application, including:

- Collection: a component of the application receives or accesses personal data sources outside the domain of the application. Access to the majority (although not all) of the resources/data must be declared in the MANIFEST.xml file, requiring the user's prior acceptance[8]. Among the potential sources of personal data are the following:
    - The user: Users provide personal information directly to applications through forms. Some applications use registration forms to request, for example, name, address, telephone, and other personal information from users.
    - Sensors: Mobile devices incorporate or may opperate, using different sensors (e.g. GPS, camera, microphone, WiFi, sensors for health and physical health, etc.) that generate a considerable quantity of personal data (e.g. geo-localisation, photos and personal audio notes, temperature, heart rate, etc.) which applications can access.
    - Applications: The personal data belonging to other applications on the device constitute another source of information that can be accessed

[8] On versions of Android prior to Android 6.0 Marshmallow (version 23 of the Android API) all the permissions required by an application were requested upon installation and were granted/denied as a block. From Android 6.0 on, each permission was requested individually when the app attempted to access the resource.

by an application through mechanisms of communication between processes (e.g. through intents).

- Environment: Mobile devices store identifiers, metadata, logs of use of the applications (e.g. frequency of use, time, type of applications, etc.) or their configurations (e.g. configurations or WiFi networks) that can be accessed by the applications.

- Use: a component of the application will process certain personal data. For example, the geo-localisation data originally represented by latitude and longitude can be mapped to the "city" by said latitude and longitude. This transformation is done through code in the application, with no need to transfer data.
- Transmission/diffusion: a component of the application sends certain personal data outside the application. For example, the geo-localisation data is sent to another application (even within the same device), the server of the provider for storage, or a third-party server for storage.
- Storage: the application persistently stores certain personal data in storage formats that are accessible through the application's Content Providers. The application has full control of the data stored and may maintain a private space accessible only through the application or may make them available to other applications.
- Erasure: the application erases certain personal through the application's Content Providers.

This study focuses on the collection and dissemination of personal data, in particular to highlight information flows towards entities external to the application itself.

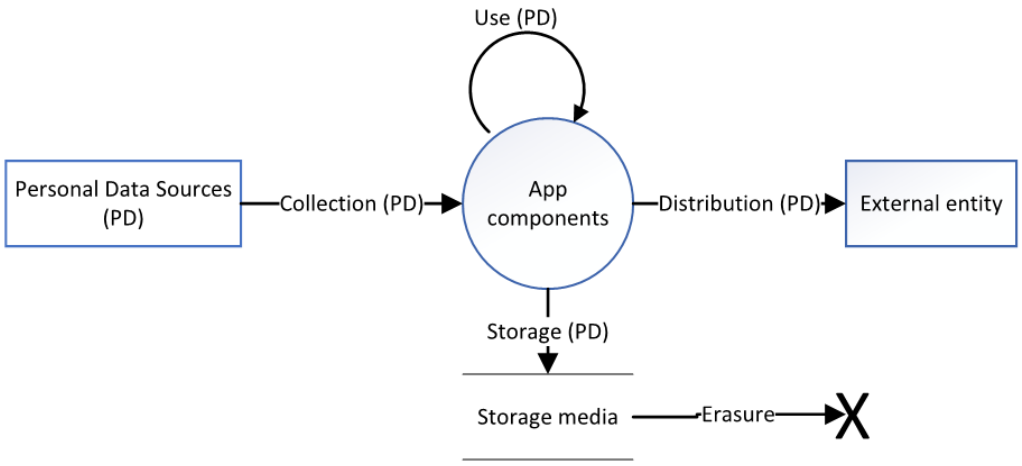

Figure 2. Life cycle of personal data

DECLARATION OF PRIVACY ON ANDROID APPLICATIONS

In the mobile ecosystem different channels have been used to inform the different actors with respect to the processing of the personal data. The informal textual

descriptions inform of the functionalities of the application and they are usually available in the app stores. Privacy policies inform of the practices of the Data Controller and the rights of the user in a detailed manner. The warnings/notifications inform the user of the intent of the application to access sensitive data, requesting their permission:

These channels have different recipients and purposes

- Textual descriptions: Textual descriptions describe the functionality of the application and on occasion also include information with respect to the processing of personal data. These descriptions are written by the providers of the applications and published in the stores, seeking to attract the attention of final users. Descriptions are written in easy-to-understand language as recipients of this channel are the users, who decide whether or not to install an application and then read the description and assess whether or not they meet the expectations. According to [1] the textual descriptions have greater presence than the warnings and privacy policies, however, as the same study showed, not all descriptions include information with respect to personal data. For example, less than 30% of the textual descriptions inform in relation to the use of geo-localisation.
- Privacy Policy: Privacy Policies are documents, usually rather extensive, that describe the practices of the Data Controller and the rights of users in relation to the processing of personal data. In practice they have legal purposes and must include all the information in accordance with the regulation, in particular the GDPR and the Spanish LOPDGDD. They are usually presented via a link which the user can visit before installing the application. Even though they present the user with detailed information, prior studies present empirical evidence[9] that demonstrate that they are very complex to understand, using technical and legal terms that are complicated for users, and are usually ignored [2]. Despite this not being the purpose of privacy policies, they often seem to be more geared towards demonstrating compliance with legal requirements before regulatory authorities, rather than being geared towards the final users. The AEPD has published ten golden rules that serve as a guide for the correct preparation of Privacy policies and a Guide for compliance with the duty to inform under the framework set out in the GDPR, the latter considering the possibility of providing the user or data subject with the necessary information in layers, detailing the necessary information to meet the requirements of the GDPR.
- Warnings/notifications: Warnings or notifications are dialogue boxes which include short texts to inform the users of the processing of resources/data that may be sensitive to their privacy. For example, access to geo-localisation requires the explicit consent of the user. Even though privacy warnings are the most direct channel of information, there still exist aspects of usability that should be improved [3].

IV. DATA FLOW DETECTION TECHNIQUES

There are three main groups of data flow detection and characterisation techniques for software programs: static analysis of the program, dynamic analysis of the process

[9] For example, this study [4] shows that only 20% of users admit to having ready a privacy policy.

in execution and communications analysis. Some consider communication analysis to be part of dynamic analysis as it requires the execution of the program.

Static analysis takes the source code as the entry or intermediate point of a program, examines it without running it and makes an approximation of "all" the possible flows of the execution or the set of possible values calculated in different points of the program. On the other hand, dynamic analysis allows for the behaviour of the application of the program to be analysed in-running. This is achieved by generating a finite set of events that stimulate the program, capture and store the registries generated and, based on them, makes an assessment of the properties of interest. The analysis of the communications puts the focus on communications made, analysing both the metadata (e.g. Recipients) and the data transmitted.

These techniques are used for the detection of data leaks and have different characteristics in terms of completeness[10] and soundness[11]. On the one hand the main advantage of static analysis is the completeness with respect to the flows detected, given that, theoretically, it is capable of detecting all of them (high completeness). Nevertheless, the main disadvantage is that it is not required and may generate false positives (low soundness). For example, this may be because there are instructions that are never executed. On the other hand, the main advantages of dynamic analysis is the low rate of false positives (high soundness) given that the analysis is not based on execution flows inferred but on flows that, effectively, are occurring. Nevertheless, dynamic analysis could lead to false negatives if, for any reason, all the possible flows are not executed (low completeness). Dynamic analysis, therefore, can only offer an indication of the lower limit of the leaks.

This imbalance between completeness and precision of static and dynamic analysis techniques has been identified quite clearly in the literature, which is why it is useful to carry out analysis using approaches that combine both techniques.

Below we describe the main techniques for static and dynamic analysis, together with some tools that allow for application on Android platforms.

STATIC ANALYSIS TECHNIQUES FOR CODE

Static analysis studies the code of the program to identify the sources and exit points of personal data, and infer all the possible ways to execute the flows of these data, building models of the state of the program and determining all the possible states. Due to the fact that there are multiple possibilities of execution, it has been decided to build an approximate model of the states of the program. The consequence of having an approximate model is the loss of information and precision in the analysis but with the advantage that the results are generalizable, because the built model represents a description of the software behaviour, regardless of the entries and the context in which it is run.

[10] Completeness, in our context, refers to the extent to which, theoretically, all of the potential flows/leaks are detected (that is, there are no false negatives). Although in the effort, false positives can be generated.

[11] Soundness, in our context, refers to the extent to which all the potential leaks detected are true leaks (that is, there are no false positive). Although in the effort, false positives can be generated.

Identification of sources and exit points

In this context, the sources of personal information are those channels within the developer's reach for access to personal data, while exit points of personal data are those channels through which personal data may filter beyond the domain of the application. For example, in an application that accesses the list of contacts of a mobile phone, sending it to an external server, the source is the fragment of the code that reads the list of contacts and the exit point is the fragment of code that sends the list to the server. The effectiveness of static analysis is significantly determined by the complexity of the list of sources and exit points identified, given that, for example, each source or exit point not identified could lead to a false negative (would not be considered in the analysis).

There are various alternatives to identify the sources and exit points included in the code of a program, the most interesting being:

- Permission analysis: On Android, calls to API methods are the main mechanism through which an application accesses the resources that can be considered sources or exit points. If the resource is considered sensitive, then it requires the user's permission and must be declared in the MANIFEST.xml file. For example, if an application needs to obtain the geo-localisation data of the mobile device, it needs the ACCESS_FINE_LOCATION or ACCESS_COARSE_LOCATION permission and if it needs to send an SMS it needs the SEND_SMS permission. At present there are various sets of data and tools that provide the correspondence between permissions and the methods that require them. Thus, once the requested permissions are known by an application, it can determine that Android API methods require these permissions and attempt to locate them in the application code. However, this technique presents difficulties, because:
    1. The Android API updates periodically, leaving the existing correspondences obsolete, and
    2. Data exist that, together with others, can be privacy-sensitive and not require access permission.
- Use of machine learning: This approach uses machine learning techniques to identify other similar methods based on an initial list of sources and exit points. The main advantage is that it can cover a greater range of sources and exit points as it is not focused only on those that require permission. Moreover, it may be extended to cover new versions of Android.

Identification tools for sources and exit points.

Below is a non-exhaustive list of some of the tools available:

- **PScout**

PScout [4] generates a correspondence between the calls to Android API and the permissions in three phases: identification of APK permissions, generation of graphs of calls and scope analysis. It is based on the assumption that most methods that provide access to sensitive information are protected by permissions. Nevertheless one must remember that privacy-sensitive resources that can be accessed without permission do indeed exist.

- **AndroidLeaks**

AndroidLeaks [5] presents another approach to identify sources and exit points through Android permissions and is very similar to PScout. While this approach is easy to implement for the sources, it is more complicated for exit points, which is why, ultimately, AndroidLeaks resorts to a manually prepared list of exit points.

- **Susi**

This tool [6] uses machine learning to identify sources and exit points. What's more, it also classifies the methods into a number of categories according to the type of sensitive information they handle and the way they transmit it. After categorisation, the fact that all categories contain more than one method shows that there is often more than one way to recover a specific piece of data, and that there are multiple forms of transmission.

- **Merlin**

Merlin [7] uses machine learning to find unidentified sources and exit points. According to the original document, Merlin's rate of false positives is 6% for sources and 26% for exit points. This approach, however, can only identify those sources and exit points that are used in at least one of the applications of the analysis set, overlooking less commonly used methods.

Static analysis of information flow

Once the potential sources and exit points are identified, the objective of the next step is to detect, using static analysis of information flow, which of these are "connected". That is, to determine whether the personal data obtained from a source reaches an exit point, producing a real leak.

The main static analysis techniques include:

- Control flow analysis: The program is modelled on a control flow graph (CFG), where each node represents a basic block of code (sentence of instruction) and each link between them indicates possible control flow between two nodes. The objective is to find all the theoretical routes of the execution of a program.
- Data flow analysis: It allows for the set of possible values that a program manages at a determined point of execution (data flow graph or DFG). This technique is based on control flow analysis, given that to determine the value of the variables at a specific point it needs to be certain of the order of the operations executed by the program (that is, its control flow graph). On the other hand, the effectiveness of control flow analysis will increase insofar as it knows the values of the variables to a certainty, seen as the result of the control sentences depends on them.
- Taint analysis: This is a special type of data flow analysis that monitors the information over the route of the execution of the program. The data of interest are marked with a symbol (commonly called a tag or stain) in the source and it is propagated through the execution routes of the program, to see if it emerges at any of the exit points. That means that if, in addition to

associating levels of privacy sensitivity both to the sources and the exit points, we can detect whether private information can reach public or undesired places. Annotations to the code are normally required to indicate the variables to be tagged.

Flow analysis tools

Just as in the case of the tools for static analysis, below is a non-exhaustive list of tools:

- **Soot**

Soot [8] generates intermediate code for Java code and executable Android code. This intermediate code has been specifically created to facilitate static analysis of the code, therefore it allows for a wide range operations to be carried out (e.g. the creation of call graphs).

- **FlowDroid**

FlowDroid [10] is an open source tool for static analysis of Android applications (and also Java), specifically for taint analysis. Flowdroid reduces the program to an intermediate representation that models the life cycle of the components of Android applications. Flowdroid can only detect intra-process data flows.

- **Epicc**

Epicc [11] complements Flowdroid through intra-process flow detection.

- **DidFail**

DidFail [12] combines FlowDroid and Epicc to sweep the data flows of a set of applications both between components and within each component. DidFail has two phases of analysis: first to determine the data flow of each application and the conditions under which they are possible; two, based on the results in the first phase, list the potentially dangerous flows activated by the applications as a set.

- **IccTa**

IccTA [13] carries out single-phase analysis. It is more accurate than DidFail because it is more sensitive to context and makes fewer overestimations of tagged data that reach exit points. IccTA and DidFail are very similar and were developed simultaneously, but are independent projects.

- **CHEX**

CHEX [14] detects the vulnerabilities in the information flow between components.

- **LeakMiner**

Similar to Flowdroid, Leakminer [15] is based on Soot for the call graph generation and implements the life cycle of Android components. Even though this tool can

analyse an application in a couple of minutes, the analysis is not sensitive to the context, which can lead to a high rate of false positives.

- **AndroidLeaks**

AndroidLeaks [5] performs taint analysis. However, it is not very accurate in the analysis given that it does not refine its sensitivity when it comes to tainting sensitive data, leading to a high number of false positives.

- **ScanDroid**

ScanDroid [16] focuses on flow analysis between components and the flow of information between applications, which lays down the challenge of relating the components with their respective recipients in other applications.

- **DroidSafe**

DroidSafe [17] is a tool that allows for static analysis of data flows of Android applications. What is worth highlighting here is that it shows a higher rate of detection in comparison to previous tools, based on the BenchDroid test bench.

- **JoDroid**

JoDroid [18] is an extension of the analysis tool JOANA to support analysis of Android applications, using data flow and control flow analysis techniques. The analysis starts with annotations in the source code of the application.

DYNAMIC ANALYSIS TECHNIQUES

Dynamic analysis techniques allow for the behaviour of the application of the program to be analysed while it is running. This is achieved by generating a finite set of events that stimulate the program, inspecting its behaviour during execution and storing the necessary registries for analysis and assessment. The following significant categories can be distinguished for our purposes.

- Taint analysis: This technique stains/tags the data that come from different sources and transitively applies tags which are then propagated over the course of the application variables, files and messages between processes. When the tagged data are transmitted outside the domain of the application (e.g. are sent by Internet), these data are registered alongside other information of interest, such as the destination of the data. The leading dynamic taint analysis tool is TaintDroid [19] which performs tagging at variable, method, message and file level. The main advantage of this approach is its consistency in detecting data leaks. However, it has the following disadvantages: it is vulnerable to flow control attacks (applications that use implicit flows to filter information), therefore it can increase the number of false positives; it requires considerable time to analyse an application and therefore is not suitable for assessing applications on a large scale. Finally, given that it needs to modify the operating system, there may be some incompatible applications.
- Monitoring access to personal data resources. This technique instrumentalizes the Android operating system to allow real time monitoring

of applications' access to resources relating to aspects of privacy. Applications are not modified, thus a mobile device with an operating system instrumented for monitoring any extension can be used. For example all the source/access point methods that access sensitive resources can be instrumented so that every time an application accesses these resources, registries are saved for subsequent analysis. The main advantages of this approach lie in that (1) it is not necessary to modify applications to assess and (2) it allows for the detection of source/exit point methods that really access sensitive resources. On the other hand, the main inconvenience is that it needs considerable time to analyse an application, making it unsuitable for analysing applications on a large scale.

Event generation techniques

Given that dynamic analysis is based on the real running of the application, the interaction between the user and the application, or the simulation thereof, is essential. This interaction is achieved through the manipulation (real or otherwise) of said applications, which is why both manual and automatic approaches exist.

- Manual: Within the manual techniques there exist two options: (1) a user uses the application to assess and record the use to play it back or (2) employ a group of people to use and record that use of the application. The first can be carried out using the *Selendroid* [19] tool. The second can be carried out using crowdsourcing but requires acquiring and configuring the devices with the applications of interest.
- Automatic: Automatic techniques are primarily based on the generation of (pseudo) random events that emulate the interaction of a user with the application. The development platform for Android already has a program called Exerciser Monkey [20] which performs this function. The application allows for certain control over the events that are generated and it is possible to replicate them as it uses a pseudo-random algorithm. Nevertheless it has the inconvenience that the events generated by this tool differ to certain extent from the real interaction events that a human would carry out.

The manual approaches should only be considered where the number of applications to analyse is relatively low. On the contrary, costs are higher, both in terms of resources and in comparison to automatic execution and pseudo-random execution. When the number of applications to be analysed is higher, the benefits of both crowdsourcing and the random execution are higher than recorded manual execution, so the decision should be between those two.

Moreover, the manual techniques are more efficient when it comes to approximating the real behaviour of a user, manipulating the application, but they are not very scalable. On the other hand, automatic techniques are scalable, but at the same time they are capable of generating fewer real activity events as the actions are performed in a pseudo-random manner.

TRAFFIC ANALYSIS TECHNIQUES

Traffic analysis techniques allow for the communications executed by an application to be examined in order to determine whether any type of personal information has been transmitted therein and the characteristics of such transmission (recipient, location in the world, etc.). For traffic analysis, four phases will be necessary: the

generation of events in the applications to assess what generates traffic, the interception of this traffic and its decryption where necessary and analysis, per se, of the information that is being transmitted. While the event generation techniques were already described in a previous section, the techniques for the three remaining phases are described in the following subsections.

Traffic interception techniques

Shows two traffic interception techniques commonly used in the context of mobile application together with the parameter of interest evaluated. These are described in more detail in the following sections.

Technique	Extendible	Extra infrastructure	Latency increase
VPN	No	No	No
Proxy	Yes	Yes	Yes

Table 1. Traffic interception techniques

A VPN (Virtual Private Network) allows for the creation of a private network over a public network by establishing virtual tunnels in establishing the connections [21]. For the use of VPN on Android, an API is offered which allows for the capture of traffic at IP level [21] [22].

When establishing a VPN on the mobile device it is not necessary to display extra infrastructure nor configure a network or mobile device so that traffic is captured, therefore it will not necessarily lead to an associated increase of latency. Nevertheless, there is the inconvenience that it is not extendible, as the implementation of the VPN will depend on the mobile device and, specifically, on the API that Android offers.

A proxy is a system that functions as an intermediary between two other systems on a network, receiving the packages sent from the origin and retransmitting them to the destination. The implementation of a proxy is independent of the mobile device which is connected to the network, although it is necessary to configure the network or the mobile device in such a manner that all traffic flows through the proxy. It will also lead to an increase in the latency of the connections.

On the other hand, given that the implementation of the proxy is independent of the mobile device, this is a solution that is very extendible to mobile devices with versions of Android that are also different. To intercept traffic from another mobile device, it is only necessary to ensure that they traffic flows through a proxy without implementing the implementation.

Traffic decryption techniques

shows the techniques used to decrypt traffic, indicating qualitatively the difficulty of implementation and the feasibility of implementing it.

Technique	Difficulty	Availability

Faking certificates	Easy	4 < Android < 7
DH_anon interception	Medium	Improbable
APK editing	Difficult	Yes

Table 2. Traffic decryption techniques

The simplest way to intercept encrypted traffic (TLS/SSL) is by **faking certificates.** Specifically, a root certificate is installed by an own Certification Authority (CA) on the mobile device that operates as a client so that when a false certificate is received from the client so that when a fake certificate is received from the Proxy or VPN, it trusts it [21] [23] [24].

This will function provided no used is made of a technique called certificate planning [24], in which the application does not trust all the certification authorities [whose root certificates are installed on the mobile device], but that it preselects one or a specific set of them. From Android 7 on, the standard configuration of applications has employed *certificate pinning* [25] [26].

Another way of intercepting TLS/SSL is to modify the start of the agreement or handshake between the client and the server so that it uses the technique known as **DH_anon** for the keys agreement that will be used in the communication. This method is vulnerable to attack from a Man in the Middle as it does not authenticate the server [27], but it is usually deactivated by default in most TLS clients.

One way of overcoming certificate pinning is APK editing, so that our CA is one of those preselected to trust [28] [29]. This method is very effective, as the developer has no possible defence against the modification of the code, but it is significantly more complex than the previous solutions.

Information analysis techniques

It is possible to distinguish between three principal categories of data that can be obtained through the capture and analysis of traffic. These categories correspond more or less to the location of the packet in which the data are to be transmitted. These categories are: information on the domain or IP address (Internet Protocol) to which the request is sent, information in the headers of the packet and information in the body of the packet.

The domain to which the request is addressed does not usually provide excessive information, but can give clues as to the behaviour of the application, especially in terms of its privacy. Certain domains correspond to advertising companies or trackers, which would imply that a request to one or more domains corresponds to a certain degree of vulnerability of the user's privacy. For an analysis of this environment, see reference [23].

The HTTP headers are also an element through which it is possible to transfer user information and therefore it is necessary to analyse them. An example might include cookies that allow for sweeping a user through various domains.

Finally, most information transmitted will be sent through the body of HTTP requests, understanding that the queries of GET requests are parallel to the body in the

PUT and POST requests. This way, it is possible to transmit a large quantity of data, and very varied characteristics.

It is possible to send multimedia information, sets of codified bytes of multiple forms or any information of the mobile device that can be represented in plain text. Each of these three subcategories presents its own difficulties for detection and analysis.

The multimedia information is detected seeking headers and magic numbers corresponding to known audio, video and image codifications. This technique can produce false negatives in the event that the coding used has not been contemplated, does not contain a magic number or recognisable header or if the packet has been sliced in suitable way [24].

The information sent in plain text is the easiest to detect and analyse. Where we have the information the sources of the app are capable of accessing it is easy to search for that information in the body of each request.

Also, sending that information as byte sets can cause major complications for analysis. On the one hand, those data can be encrypted or coded but on the other hand there is no reference as to what those data might mean.

Tools

Table 3 shows a summary of the most important tools that implement one or several of the techniques described in the previous subsections.

Tool	VPN	Proxy	Own CA	DH_anon	APK editing	TCP stack reset	TLS interception
Android VPN	Yes	No	IN	IN	No	Yes	IN
Mitmproxy	No	Yes	Yes	IN	No	No	Yes
Tcpdump	No	Yes	No	No	No	Yes	No
Ssldump	No	Yes	Yes	IN	No	Yes	Yes
Meddle	Yes	No	-	-	-	-	-
WireShark	No	-	-	-	No	No	No
Frida	No	No	No	No	Yes	-	-
Apktool	No	No	No	No	Yes	-	-

Table 3. Traffic analysis tools (*IN = Implementation Needed*)

- AndroidVPN is the most widely used solution for different traffic analysis applications for Android mobile phones [21] [22] [23] [24] [30]. It is necessary

to install a root certificate issued by an own CA on the mobile device so that it trusts the VPN. All applications developed for API 24 level (Android 7) will not trust the certificate unless it modifies the application of the set of standard Android certificates [25] [26].

This tool does have a number of inconvenient features however. 1) it is not extendible for mobile devices that run on a system other than Android; 2) it is executed in the user's mobile device, preventing it from using other VPNs and consuming the battery; 3) it is not very stable as it depends fundamentally on Android, which could terminate the underlying process, and; 4) it is necessary to reset the HTTP/TCP protocol stack and the packets delivered to the VPN are layer 3 [31].

It also has a series of advantages. 1) If trust in the certificate is not a problem, it is very easy to use 2) Requires no complication with extra infrastructure as it runs on the mobile device itself 3) Ensures that all traffic from the mobile device is captured.

- Mitmproxy is a mature and extendible HTTP and HTTPS traffic interception tool. [32]. It is necessary to install an own CA root certificate on the mobile device so that it trusts the VPN. All applications developed for API 24 level (Android 7) will not trust the certificate unless it modifies the application of the set of standard Android certificates [25] [26].

  There is the inconvenience that, as a proxy external from the mobile device, it is necessary to design a network that ensures that all mobile traffic passes through the proxy [34]. On the other hand, it has a series of advantages: 1) if trust in the certificate is not a problem it is very easy to use; 2) it is not necessary to reset the HTTP/TCP battery, and; 3) it allows for the separation of the mobile interception device, which is why it doesn’t depend on the first system.

- Tcpdump is a tool widely used for analysing system packets [35]. It does not allow for the decryption of HTTPS traffic which is why the process must be carried out using other mechanisms. Its main advantage is the extensive use of the tools and, therefore, the extensive existing documentation.

  Nevertheless, there are a number of drawbacks: 1) it is necessary to design a network to ensure that all the mobile traffic passes through the proxy and 2) it is necessary to resent the HTTP/TCP stack as the packets delivered are layer 3.

- Ssldump [36] is a sister tool to tcpdump. The main difference is that it does allow the decryption of TLS/SSL traffic but it also needs the client to accept the certificate, much like the first two tools.

- WireShark is a popular [37] traffic analyser with a graphic interface that can be used to capture live traffic or anlayse the reuslt of a previous capture stored in a PCAP file [37]. The advantages and disadvantages are similar to tcpdump, with the difference that it is not necessary to reset the TCP stack and adding the use of a GUI, making it more complicated to automate.

- Meddle is a VPN for Android that is primarily focussed on use with the application ReCon. It is not available for use in applications developed for people external to ReCon [38]. In terms of advantages and inconveniences it

is similar to AndroidVPN with the exception that it is not necessary to reset the TCP/HTTP stack.

- Frida is a tool that allows for the code to be injected into applications for a greater variety of operating systems. It could be used specifically to overcome the pinning of certificates that implement some of the applications [39].

  For applications developed for API 21 or higher it will not be necessary to use Frida as, in theory, modifying the application's MANIFEST.xml file could be exceeded without any problems. For other applications it could be necessary to use Frida, when the pinning of certificates is implemented directly in the application.

- Apktool is a dismantling tool for Android applications [40] that follows the APK packaging. It is very useful to decompile an application and be able to modify it or analyse its code. It can be used in conjunction with Frida.

V. CONCLUSIONS

This study shows the existence of a large number of possible personal data flows in mobile applications, which implies a potentially high risk of communication to third parties of personal data without the knowledge of the user themselves or the owner of the data (the data subject). This is possible due to the high availability of sensors incorporated into devices and the high number of unique identifiers that are intrinsic to the device itself, that facilitate the collection of personal data attributable to the user of the mobile device. Moreover, the great capacity for connectivity of mobile devices and the large variety of agents that intervene in the mobile application environment only serve to increase this risk.

Developers of mobile applications, the managers that subcontract development and distributors or app repositories have a duty or proactive responsibility, as set out in the GDPR, to ensure that their products and services comply with the personal data protection levels established in the regulation. Therefore, they must carry out analysis and/or audits using tools and methodologies such as those presented in this study to determined that the apps made available to users are in line with the privacy policies, and any other information provided through the textual descriptions, warnings and notifications with the adequate guarantees required by the GDPR and the regulations adapting Spanish Law to the GDPR, as in the case of the LOPDGDD[12].

There are a plethora of techniques and tools that can be employed to determine whether an application collects and transmits personal data. As the study shows, it is necessary to combine the results of all of these tools for effective analysis.

The potential personal data flows detected through static analysis techniques should be confirmed with dynamic analysis tools that allows for the interception and analysis of real traffic of the app. Both the static analysis tools and dynamic analysis tools analysed are freely available to any user.

As an identification tool for sources and exit points, the conclusion is [41] that Susi is capable of locating not only all the sources and exit point found by other tools, but also finding other new ones that had been previously overlooked. The flow analysis tool

[12] Organic Law 3/2018, of 5 December, on Personal Data Protection and the guarantee of digital rights

Soot allows for a call graph of the applications to be obtained while FlowDroid can detect the connections between the sources and the exit points

For investigations for which manual execution is not the optimal option, we can use Exerciser Monkey, which allows for the execution of pseudorandom executions. Other research has also shown the efficiency of crowdsourcing in terms of the capacity to analyse a large number of different applications on different devices and used by real users.

In terms of intercepting traffic, the use of a proxy gives the advantage of physically separating the execution phase and the interception phase, making it possible to user other security tools outside the mobile environment.

This document is the product of collaborative work between the AEPD and José María del Álamo, Danny Santiago Guamán, Lucas María Tomé and Esperanza Zamora from the Universidad Politécnica de Madrid.

VI. REFERENCES

[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,» [Online]. 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,» [Online]. Available: http://selendroid.io/.

[20] «UI/Application Exercise Monkey,» [Online]. 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,» [Online]. Available: https://developer.android.com/training/articles/security-config.

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

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

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

[29] «Bypassing Certificate Pinning on Android for fun and profit,» [Online]. 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,» [Online]. Available: https://developer.android.com/reference/android/net/VpnService.

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

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

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

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

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

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

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

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

[40] «Apktool,» [Online]. 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,» [Online]. Available: https://developer.android.com/distribute/best-practices/develop/runtime-permissions.

[43] «Android Developers: Permissions,» [Online]. 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.