Mojtaba Eskandari*, Maqsood Ahmad, Anderson Santana de Oliveira, and Bruno Crispo

Analyzing Remote Server Locations for Personal Data Transfers in Mobile Apps

Abstract: The prevalence of mobile devices and their capability to access high speed internet has transformed them into a portable pocket cloud interface. Being home to a wide range of users' personal data, mobile devices often use cloud servers for storage and processing. The sensitivity of a user's personal data demands adequate level of protection at the back-end servers. In this regard, the European Union Data Protection regulations (*e.g.*, article 25.1) impose restriction on the locations of European users' personal data transfer. The matter of concern, however, is the enforcement of such regulations. The first step in this regard is to analyze mobile apps and identify the location of servers to which personal data is transferred. To this end, we design and implement an app analysis tool, PDTLoc (Personal Data Transfer Location Analyzer), to detect violation of the mentioned regulations. We analyze 1,498 most popular apps in the EEA using PDTLoc to investigate the data recipient server locations. We found that 16.5% (242) of these apps transfer users' personal data to servers located at places outside Europe without being under the control of a data protection framework. Moreover, we inspect the privacy policies of the apps revealing that 51% of these apps do not provide any privacy policy while almost all of them contact the servers hosted outside Europe.

Keywords: Personal Data, Privacy, Mobile Apps, Cloud, Information Flow Analysis

DOI 10.1515/popets-2017-0008
Received 2016-05-31; revised 2016-09-01; accepted 2016-09-02.

***Corresponding Author: Mojtaba Eskandari:** DISI, University of Trento, Italy and Fondazione Bruno Kessler, Trento, Italy, E-mail: mojtaba.eskandari@unitn.it
Maqsood Ahmad: DISI, University of Trento, Italy, E-mail: maqsood.ahmad@unitn.it
Anderson Santana de Oliveira: SAP Labs, France, E-mail: anderson.santana.de.oliveira@sap.com
Bruno Crispo: DISI, University of Trento, Italy and DistriNet, KULeuven, Belgium, E-mail: bruno.crispo@cs.kuleuven.be

1 Introduction

The number of smartphone users has rapidly increased over the past decade. International Data Corporation (IDC) reported 1.43 billion smartphone shipments in 2015 and anticipated a steadily rise to 1.92 billion shipments in 2020 [35]. In Europe alone, according to the *Ericsson ConsumerLab's* report, there were 475 million user subscriptions in 2014 and this number is estimated to reach 815 million subscriptions in 2020 [18].

Smartphones often store personal data such as contacts, financial information, photos, location, etc. Essentially, "personal data" refers to any information relating to an identified or identifiable natural person *i.e.*, data subject [34]. It is a common practice for mobile apps to collect, process and transfer personal data to back-end servers (cloud) for further processing and storage. Cloud service provisioning usually is independent of the service provider's location; thus, it raises the issue of identifying in which jurisdiction, personal data is stored and processed. Data protection regulations, such as article 25.1 of the European Union Data Protection Directive (`DPD'25.1`) [24, 34], restrict personal data transfer to particular jurisdictions. DPD'25.1 prohibits the transfer of personal data to any country that does not ensure an adequate level of protection. This principle drove the creation of the *EU-US Safe Harbor* agreement in July 2000 [19].

Years later, an Austrian privacy activist, Maximillian Schrems, sued Facebook Ireland claiming that the company made his personal information available to the US intelligence agencies without any consent or notification[10]. As consequence of that filed case, the European court of Justice decided to invalidate the Safe Harbor agreement [14]. This decision caused numerous debates on the legal aspects of transferring and processing European Citizens' personal data to outside the European Economic Area (EEA) [4, 44].

In February 2016, the European Commission and the United States agreed on a new framework for transatlantic data flows, the *EU-US Privacy Shield* [2]. The new arrangement imposes stronger obligations on companies in the US in order to protect the European users' personal data. Moreover, it provides more robust

BY-NC-ND

monitoring and enforcement mechanism that allows the European users to raise any inquiry or complaint in this context with a dedicated new Ombudsman.

Several studies highlighted that mobile applications actively collect and exfiltrate personal data from smartphones [3, 16, 23]. The main concern here is how to enforce jurisdictional regulations such as DPD'25.1 in mobile apps. As the first pace to tackle the problem, we design and implement *PDTLoc* (Personal Data Transfer Location Analyzer), which employs both static and dynamic analysis techniques to infer whether the apps violate DPD'25.1. PDTLoc inspects mobile applications and extracts information about the collected personal data and the jurisdictions of the remote servers to which the data is transferred. In order to investigate the current state of the privacy protection of the European smartphone users with regard to DPD'25.1, we used PDTLoc to analyze the 1, 498 most popular apps in the EEA. We obtained evidence confirming that 16.5% (that is 242 apps), transfer data outside the EEA whitout user consent. This signifies that these apps collect and transfer personal data to servers located outside the EEA escaping the control of a data protection framework (*e.g.*, Safe Harbor), thus violating the users' data protection rights. We also analyzed the privacy policies provided by the app developers. One striking finding is that 51% of the most used apps in Europe do not provide any privacy policy. Furthermore, out the apps providing a privacy policy, only 53 apps (3.5% of all) have Safe Harbor certification, whose agreement was anyway declared invalid, as we mentioned above. Perhaps the situation will be clarified when the EU-US Privacy Shield framework will be finalized.

Contributions:

- We design and implement PDTLoc, an Android app analysis tool that employs a backward program slicing technique to detect DPD'25.1's violation by mobile apps [34].
- We collect a dataset of 1, 498 Android apps which are the most used apps in the EEA. We analyze these apps using PDTLoc to investigate the recipient server locations of the users' personal data. To our knowledge, this is the first study of the kind conducted so far.
- In order to demonstrate the gravity of the problem, we also analyze the privacy policies of the apps in the dataset in order to check if the data controller and the processing locations are clearly identified.

Organization: Section 2 provides a formal description of the problem. Section 3 discusses the background of data flow analysis. We explain the design and implementation of PDTLoc in Section 4 and discuss the empirical analysis setup and the results in Section 5 and Section 6, respectively. Section 7 provides suggestions for better data governance in the EU. Section 8 states the limitations of our work and Section 9 reports the related work. Finally, Section 10 concludes the paper.

2 Problem Statement

Let $A = \{a_0, a_1, a_2, \dots, a_n\}$ be a set of android apps. There is a jurisdictional regulation denoted by $R = \{l_0, l_1, l_2, \dots, l_q\}$, which restricts $a_i \in A$ to transfer data to particular locations *i.e.*, $l_0 ... l_q$. Let $S = \{s_0, s_1, s_2, \dots, s_m\}$ be the set of all servers used by A to store and process data. Each $s_j \in S$ is situated in a physical location indicated as s_j^l. There is a `T(a,s)` function showing the app a transfers personal data to the remote server s. Formally speaking, we have to enforce the regulation R on the set A by using Equation 1:

$$\forall\ a \in A\ \exists\ s \mid\ T(a,s) \Rightarrow s^l \in\ R \qquad (1)$$

The problem we solve is to determine, with a particular level of certainty, whether $a_i \in A$ violates R. For each app, we have to discover the list of the remote servers to which it transfers data; then, we need to find their locations and match them against the allowed list, R, to discover violations.

In the analysis we perform in this work, the major goal is to expose the status of privacy protection with respect to DPD'25.1 by the most popular mobile apps in the EU. More specifically, we analyze the type of personal data accessed by these apps, the number of apps that collect/transfer personal data to servers over the network, and the locations of the recipient servers.

Caveat: In this work, we do not try to assert the compliance of the apps but rather to detect if they are likely violating user's privacy rights concerning international data flows limited by the DPD'25.1.

3 Data Flow Analysis

The purpose of our data flow analysis is to understand how an app retrieves and transfers personal data. In the context of the problem, we need to infer whether a user's personal data leaves the boundary of the app and to identify the geographical location of the data recipient. Transferring personal data consists of a data flow between the Android framework APIs called to access personal information; such as device Id, location, contacts, calendar, photos, etc., *i.e.*, "source APIs"; and the APIs that provide potential transfer points; such as network, files, log, etc. *i.e.*, "sink APIs". Source and sink APIs are discussed in detail later in Section 4. An automated data flow analysis tool detects data flows between the source and sink APIs. We can perform such analysis both statically, *i.e.*, extracting information from the bytecode/source code; and dynamically, *i.e.*, running an app on a device/emulator and monitoring its behavior. Here we describe the fundamental concepts regarding static and dynamic analysis that form the basis of our approach.

– **Backward Program Slicing:** In the bytecode representation of a program, two types of data structures are used for storing and performing operations on data, *i.e.*, stack and register. Operations are performed on stack or register variables using program instructions (I). In this text, we use registers (r) as we perform analysis on Android apps and Android is based on a register based virtual machine. Backward program slicing is a data flow analysis technique that, with respect to a register r used at point P in a program, considers all the instructions I that can be executed before P and have a direct or indirect effect on the value of r at P. The combination of r and P, a certain API call in our case, forms a slicing criterion, whereas the set of instruction I that effect the value of r at P is called a backward slice. For instance, Line 2–6 and Line 9 represent a backward slice corresponding to the variable `Sum` used at Line 9 (Point P) in Listing 1.

  Listing 1. Backward Slicing Example

```
1 ...
2 int i, sum, count;
3 i = 1;
4 sum = 0;
5 for( ; i != 50; i++){
6   sum += i;
7     count = count * 100 / i;
8 }
9 System.out.println( "Sum = " + Sum); // point P
```

  As a backward slice usually starts from a sink API, an inspection of a backward slice corresponding to a particular register, can provide information about its source and, thereby, infer the data flow path between the source and sink APIs. The source, here, is represented by the API that retrieves the personal information and the sink is represented by the API that transfers the information to the outside world through the Internet.

  Such data flow paths can also be inferred by other analysis techniques outlined in the literature (Section 9), such as [6]. These techniques, generally, identify access to sources of personal information and then track its flow in the program. Since the basic motivation of this work is to find the location of the recipients of personal information, starting the analysis from the sink APIs yields a better performance by filtering out the apps that do not transfer personal information to the outside world. Therefore, we use backward program slicing and effectively avoid analyzing those apps which might access personal data, but do not send it outside.

– **Dynamic Tracking:** Dynamic analysis is the process of executing an app and observing its behavior. Tracking the behavior exhibited by an app at execution time, usually, involves monitoring the system resources accessed by the app, such as filesystem, network, telephony, etc. Since, this work focuses on personal data transfer over the network, we monitor the outgoing traffic.

Both of these techniques, static and dynamic, come with their respective pros and cons. Static analysis is able to reach all possible data flows in the source code and not only those executed in a specific run of the app. On the other hand, there are programming features, such as various types of code and data obfuscation, reflection, and dynamic code loading, that yields an incomplete result. Reflection is a programming feature that enables apps to operate on strings, *i.e.*, instantiate objects of a class, invoke its methods and access/modify its fields where the class, method and field names are represented by strings that may not be readily available for a static analyzer[36]. Similarly, dynamic code loading allows an app to extend its code base after installation[13]. Therefore, it is impossible for a static analysis tool to fully analyze such cases of dynamic nature. On the other hand, dynamic analysis is able to overcome these limitations in many cases. However, the app must be executed to trigger the critical data flows for dynamic analysis to capture sensitive behavior, which is a challenge for dynamic analysis tools. There are limitations with each technique; however, if combined, static and dynamic analysis techniques can complement each other in designing a more effective data flow analysis system.

4 Our Approach: *PDTLoc*

In order to effectively detect violation from DPD'25.1 in popular mobile apps, we design PDTLoc, a tool that takes advantage of both static and dynamic analysis.

4.1 Overview

Figure 1 shows an overview of the basic blocks and the workflow of PDTLoc. PDTLoc consists of three major modules: a static analysis, a dynamic analysis, and a location investigator module. Both the static and the dynamic analysis modules take an `.apk` file; extract a list of accessed personal data; and a list of server names, URLs and IP addresses to which the personal information is sent. The dynamic analysis module complements the static module and it is only activated when the given app uses reflection. The lists of URLs extracted by both the modules may wary due to different nature of these analysis techniques. Therefore, we consider a union set of both the lists.

The lists of URLs along with the identifiers of the respective personal information, which is sent to these URLs, are stored in a repository for further analysis. The Location Investigator module (shown in Figure 1) reads URL/IP addresses from the repository and creates a list of the server locations where the app sends the collected personal data.

4.2 Static Analysis Module

PDTLoc's static analysis module, represented by module (1) in Figure 1, takes an `.apk` file and extracts the URLs/IP addresses of the destination servers. APK is an archive file format that represents the Android app and contains all the compiled code and the compiled/raw resources. Android apps are usually written in Java, compiled into Dalvik bytecode and then all the compiled classes are packed into a `classes.dex` file. Therefore, we have to analyze this file to understand the behavior of the app.

To analyze the `classes.dex` file, the static analysis module extracts and translates it into Smali code by employing ApkTool [39]. Smali code is a disassembled representation of the Dalvik bytecode [22]. We use Smali disassembly over Java because the decompilation process is more prone to be thwarted by obfuscation, whereas the disassembly is more resilient [37]. The *Static Check* component inspects the Smali code for the use of reflection in order to pass the app to the dynamic analysis module. The *Backward Slicer* performs backward program slicing on the Smali files to discover the information flow to certain sink APIs. This component employs an extension of SAAF (Static Android Analysis Framework for Android apps) that is able to extract the backward slices corresponding to a given sink API [25].

Table 1 lists the sink APIs along with the corresponding parameters of interest used in our analysis. We carefully analyzed the lists of sink and source APIs provided in the literature, such as [6, 42], and considered only those sink APIs that take the name or IP address of a particular server and transfer data to it. The *Backward Slicer* receives these APIs in the form of an `.xml` file referred to as BackTrack Patterns (*shortly* BT Patterns). A BT Pattern provides information about the API, such as class name, method name, position of the parameter in the parameters list and its type. For example, Listing 2 instructs the *Backward Slicer* to backtrack parameter `0`, which is of type `Ljava/lang/String;`, of `setURI` method of class `org/apache/http/client/methods/HttpPost`. Similarly, information about the rest of the APIs in Table 1 is also provided to the *Backward Slicer*.

Listing 2. BT Pattern Example

```
<backtracking-pattern
  active="true" class="org/apache/http/client/methods
     /HttpPost"
    description="Apache HTTP POST" method="setURI"
     parameters="Ljava/lang/String;"
    interesting="0" />
```

The *Backward Slicer* spots the position of a given BT pattern in the Smali files, backtracks the target parameter and extracts the corresponding code slice. A code slice contains all the code statements that have a direct/indirect impact the register holding the value of the target parameter. Listing 3 depicts an example of a backward slice corresponding to the API `java/net/URL;-><init>`. Similar slices representing each of the BT patterns are extracted in the form of BT report and provided to a *Slice Analyzer*.

The *Slice Analyzer* component traverses the slices and extracts URLs/IP addresses to which user's personal data might be transferred. Its major role is to analyze the slices for certain data extraction patterns that represent access to personal data. A typical data extraction pattern consists of a class name, a method name, and a parameter as listed in Table 2. This list includes only the APIs provided by the framework, used to acquire a user's personal information. However, it does not consider other methods of acquiring user per-

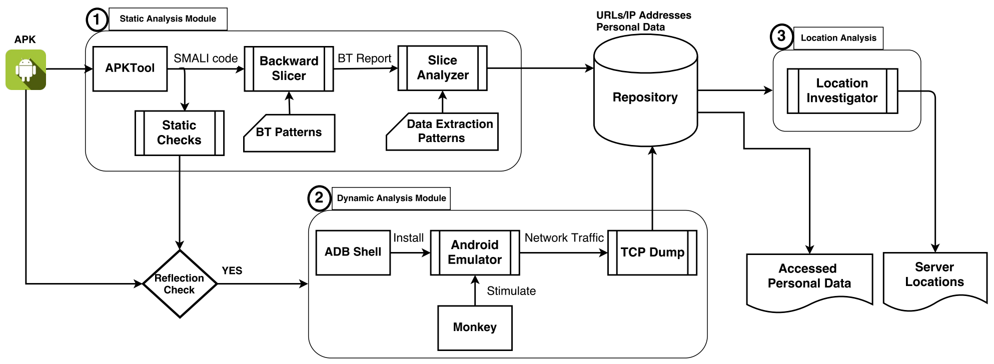

Fig. 1. A general overview of PDTLoc.

Class	Method	Parameter
`javax/net/ssl/SSLSocketFactory`	`createSocket`	`host, port`
`android/net/Uri`	`parse`	`uri`
`java/net/URL`	`<init>`	*
`java/net/Socket`	`setRequestProperty`	`key, value`
`org/apache/http/client/methods/HttpGet`	`<init>, setURI`	`uri`
`org/apache/http/client/methods/HttpPost`	`<init>, setURI, setEntity`	`uri`
`org/apache/http/client/methods/HttpPut`	`<init>, setURI`	`uri`
`java/io/OutputStream`	`write`	*
`java/io/Writer`	`write`	*

Table 1. The sinks of personal information.
*: All possible parameters.

sonal data, such as data input through text fields or that stored on files on the device, etc. At this stage, PDTLoc can only guarantee the existence of such data flow paths and flags them suspicious with respect to their potential violation of DPD'25.1. Finally, the *Slice Analyzer* generates a mapping of the personal data accessed by the app and the corresponding server-locations to which the app might transfer the personal data, and stores it in a repository for further processing.

4.3 Dynamic Analysis Module

We dynamically analyze the apps using reflection (around 90% of the apps in this study) that can potentially conceal data flows when only statically analyzed [45]. Therefore, based on the static checks, the PDTLoc's dynamic analysis module, (module ② in Figure 1), is activated in case of the app making use of reflection. The dynamic analysis module utilizes a number of tools provided as part of the Android SDK. It executes the given app, monitors its network traffic, and captures the URLs/IP addresses to which the personal data is transferred. It employs `adb` to manage the dynamic analysis process that follows certain steps for each app:

– Launch the emulator and configure WiFi and GPS.
– Run the TCP-Dump tool to monitor the network traffic [38].
– Install the app and unlock the emulator.
– Launch the app with `Monkey` to stimulate it [21]. `Monkey` injects random events into the app including touch, drag, type, change the screen orientation, etc.
– After completion of the execution, URLs/IP addresses and the parameters (*i.e.*, the transmitted data) are extracted from the TCP-Dump, stored in the repository and all the data is erased from the emulator in order to make it ready for next app analysis.

This process is repeated for each app that uses reflection. Since the goal of using the dynamic analysis is to complement the static analysis, the results are stored in the repository as a union set of both the analysis modules. The dynamic analysis module captures all those data flows (including those involving in reflection, native code, dynamic code loading, etc.) that are properly executed during the analysis run. In order to ex-

Listing 3. Backward Slice Example

```
const-string v0, "facebook.com"
sput-object p0, Lcom/facebook/Settings;->facebookDomain:Ljava/lang/String;
sput-object v0, Lcom/facebook/Settings;->facebookDomain:Ljava/lang/String;
sget-object v0, Lcom/facebook/Settings;->facebookDomain:Ljava/lang/String;
const-string v0, "https://graph.%s"
...
move-result-object v4
invoke-direct {v3, v4}, Ljava/net/URL;-><init>(Ljava/lang/String;)V
```

Class	Method	Parameter	Parameter example
android/content/ContentResolver	query	uri	content://media/external/video/media content://sms/inbox content://com.android.browser/history
android/net/Uri	parse	uri	
android/content/Context	getSystemService	name	location connection wifi netstats batterymanager
android/telephony/TelephonyManager	getAllCellInfo getCellLocation getDeviceId getSimCountryIso ...	*	

Table 2. The sources of personal information

tract the personal information traveling through those data flows, we analyzed the URLs by looking for particular patterns like `"lat=[\.\-0-9]*"`, `"city=[a-Z]*"`, `"deviceIds=[0-9]*"`, `"macAddress=[0-9a-f]*"`, etc.

4.4 Location Investigator

The fundamental purpose of PDTLoc is to analyze an app and tell if it sends user's personal data to servers hosted at locations outside the jurisdiction defined by the given policies, *e.g.*, the EU DPD'25.1. Therefore, we need to investigate the physical locations of the machines represented by the extracted URLs/IP addresses. The PDTLoc's third module is *Location Investigator* (module ③ in Figure 1). This module reads the URLs/IP addresses of the remote servers from the repository and determines their physical locations. There are a number of online databases that bind IP addresses (or server names) to their corresponding geographical locations. We configure the Location Investigator to use IPaddressAPI.com [28]. The Location Investigator retrieves the locations using this online IP-Location service and reports a mapping of URLs and geographical locations. It also marks those locations which are outside the declared jurisdiction.

5 Empirical Analysis

This section describes the criteria and the procedure of dataset collection followed by the experimental setup and the evaluation goals.

5.1 Dataset Collection

Since this work considers the analysis of the transfer of European users' personal data outside the EEA as a case study (*i.e.*, violating DPD'25.1), we have targeted the popular mobile apps in the EEA. For the app selection, we relied on AppFigures which is an app tracking platform that monitors the downloads and sales of the apps from Google and Apple app stores [5]. We downloaded AppFigures's list of the 400 most popular apps for each EEA state. We identified 1,498 distinct android apps for the entire EEA and downloaded them. We use android apps because they have over 80% of the market share [27], also for the availability of analysis tools and their simple downloading mechanism. However, we searched for the apps in our list, on iTunes in order to check their availability for iOS. In the dataset we have collected, 80% of apps are available for both Android and iOS and 20% are available only for Android. Therefore, our research results are meaningful to iOS users as well; assuming the destination cloud servers are the same for both OS, which is reasonable.

We developed a crawler to download `.apk` files. It browses the given marketplace website for the download link of each app and downloads it. Moreover, since some marketplaces do not provide the download link instantly, the crawler manages to switch to another marketplace to ensure that the app is downloaded. We submitted all the downloaded `.apk` files to VirusTotal which is an online service to analyze suspicious files and URLs [41]; it marked them as benign.

5.2 Experimental Setup

As PDTLoc consists of a static and a dynamic analysis module, we designed the experiment in such a way to know the results from both modules separately as well as their combined results. The static module analyzes all the apps in the dataset, whereas the dynamic module analyzes those apps which pass the static reflection check.

Static analysis module configuration: This module analyzes all 1, 498 downloaded `.apk` files. We used a desktop computer with an `Intel Core i5 3.20 GHz` CPU and 8 *GB* memory running `Ubuntu 15.10` for the analysis. The static analysis took roughly 38 hours on this machine.

Dynamic analysis module configuration: The dynamic module analyzes those apps that are marked for the use of reflection. We call it Auto-Dynamic as it uses `Monkey` to stimulate the apps. It employs `Android Lollipop 5.0.1` on its emulator and the `Monkey` tool is configured to inject 700 random events into each app. We executed the experiment on the same machine and it took about 6 days to analyze all the given apps.

5.3 Evaluation Goals

The experiments are designed to answer the following research questions:

- **RQ1. Accessed Data:** What are the types of personal data accessed/collected by the apps?
- **RQ2. Data Transfer:** What are the locations of the remote servers to which the collected personal data is transferred?
- **RQ3. DPD'25.1 Violation:** How many of the popular apps used in the EEA, violate DPD'25.1?

Notice that there may be exceptions where transfers outside of the EEA are authorized, such as Binding Corporate Rules [1]. Data subjects need to be informed about the adoption of such legal mechanisms, through the terms of service and privacy policy. We took this in consideration when considering violations, looking for information about the agreements and certifications by the app developer when available.

6 Results and Discussions

We analyzed statically all 1, 498 apps and out of these apps, 1, 472 (98%) apps use reflection; therefore, we analyzed them also dynamically.

This section reports the analysis results and discusses them in the light of the consequent privacy concerns. The supporting data, the full list of the analyzed apps and the study's conclusions are accessible via this link: `http://bit.do/pdtloc`.

6.1 Personal Data Accessed

Pieces of personal data stored on a user's device are categorized into three broader groups as shown in Table 3. These groups, Content; Device; and Network, represent user data stored on the device; device status data; and network data, respectively.

Figure 2 provides a graphical representation of the number of apps that access the various types of personal data (**RQ1**). According to these results, device status data, marked as "Device", such as device id, notifications and power information, etc., as shown in Table 3, is accessed by almost all apps. Further examination revealed that 75% of the apps request device location. Similarly, network information is of interest to 65% of the apps. What is alarming here is that over 70% of the apps read "Content", which carries sensitive personal information.

6.2 Contacted Servers

The static analysis and the dynamic analysis module extracted, in total, 135 *K* and 21 *K* valid URLs/IP addresses, respectively. The number of URLs extracted by the static analysis module is much more as compared to those extracted by the dynamic analysis. The disparity in these numbers further endorses that static analysis

Category	Information
Content	Calendar
	Contacts
	Audio
	Video
	Image
	Files
	MMS & SMS
	Call log
	System settings
	User dictionary
Device	Device ID
	Online accounts
	Power state
	System alarm
	Device location
	Telephony services
Network	MAC Address
	Proxy settings
	Network Status
	Network connectivity
	Network usage
	history and statistics

Table 3. Types of personal data in each category.

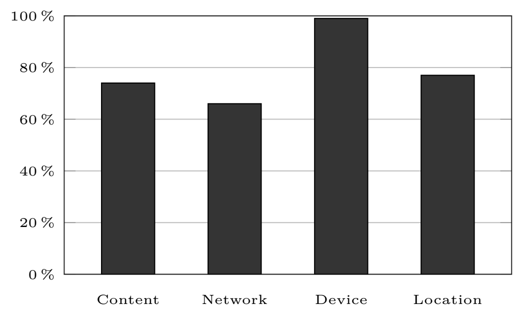

Fig. 2. Type distribution of personal data collected by the analyzed apps.

provides an over-approximation of the program and extracts URLs which might not be contacted in an actual program execution, whereas the dynamic analysis extracts only those URLs which are contacted by the app in a single run. However, the presence of any of these URLs in the executable of an app provides a potential data transfer point and cannot be ignored. Since the purpose of dynamic analysis is to widen the analysis range, we use a combination of the URLs/IP addresses extracted by both the modules. Figure 3 illustrates the value of the dynamic analysis module to the static analysis results; where the red line shows the number of URLs found in a particular app and the blue line represents the number of new unique URLs discovered only by dynamic analysis in the same app. For certain apps

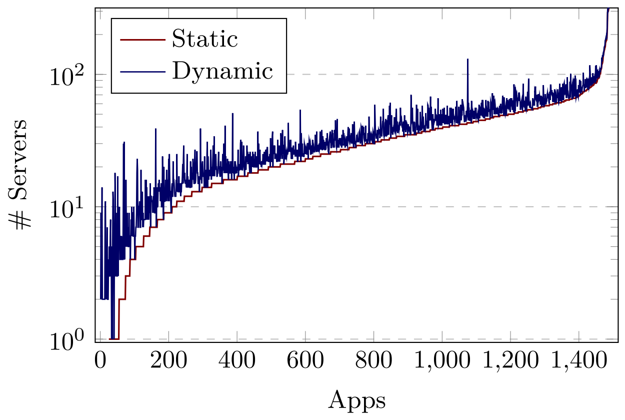

Fig. 3. This graph shows that how effective is the auto-dynamic analysis technique in covering the blind spots of the static analysis technique.

the number of new URLs/IP addresses discovered by the dynamic analysis in comparison to the static analysis is much higher than the others possibly because of heavy use of reflection.

It is important to mention here that the relation between servers and URLs is one-to-many, *i.e.*, on each server there can be multiple resources represented by different URLs. Therefore, the number of servers an app contacts is considerably less than the number of URLs.

Moreover, PDTLoc could extract data flow paths only for a portion of all the URLs due to known limitation of static and dynamic analysis. For clarity, we refer to the data flows captured during the dynamic analysis as observed data flows. Figure 4 provides the number of servers and apps for which at least a personal data transfer is observed. Overall for 505 (34%) apps, transfer of personal data is observed among which 295 (20%) of the apps transfer personal data outside the EEA. Similarly, 401 servers are the recipients of data transferred by these apps among which 213 are located outside the EEA.

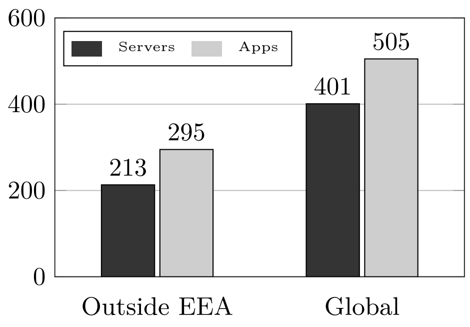

Fig. 4. Number of servers and apps engaged in actual data transfer

6.3 Server Locations

Figure 5 illustrates the distribution of locations for servers engaged in the transmission of personal data (**RQ2**). As it reveals, only 23% of the servers are hosted in the EEA and the majority of the servers (67%) is in the US. Therefore, it is expected that the major portion of personal data to travel outside the EEA.

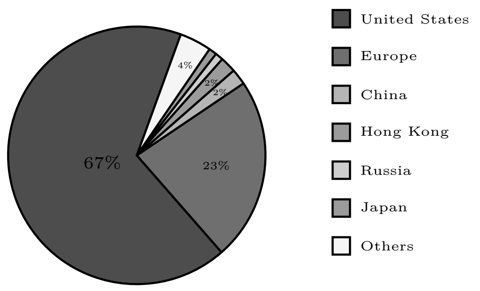

Fig. 5. The distribution of the locations to which the European users' personal data collected by mobile apps travels.

The main focus of this work is to provide a location analysis of the servers contacted by the apps in our dataset. Figure 6 shows a graphical representation of the country-wise distribution of servers based on the number of apps. It illustrates that a reasonable portion of the apps contact (observed and potential data transfer) servers outside the EEA and US, especially China, Japan, India and Russia.

As most of the analyzed apps contact servers outside the EEA, it is interesting to know the number of apps transferring data only to a certain location/country. In this regard, Figure 7 illustrates the number of apps exclusively contacting servers located in the EEA, US, EEA & US and any other country. It shows that none of the apps perform exclusive data transfer to servers located outside the EEA and US. Only 12 (less than 1%) apps contact servers located only inside the EEA. In contrast, the number of apps contacting servers exclusively in the US is reasonably higher, *i.e.*, 892 apps. This implies that most of these apps either belong to the US-based companies or having their data centers located in the US. Similarly, the number of apps exclusively contacting servers in the EEA & US is 232. A similar reasoning applies to these apps as well where the apps either communicate to servers in the US or their local counterparts in the EEA.

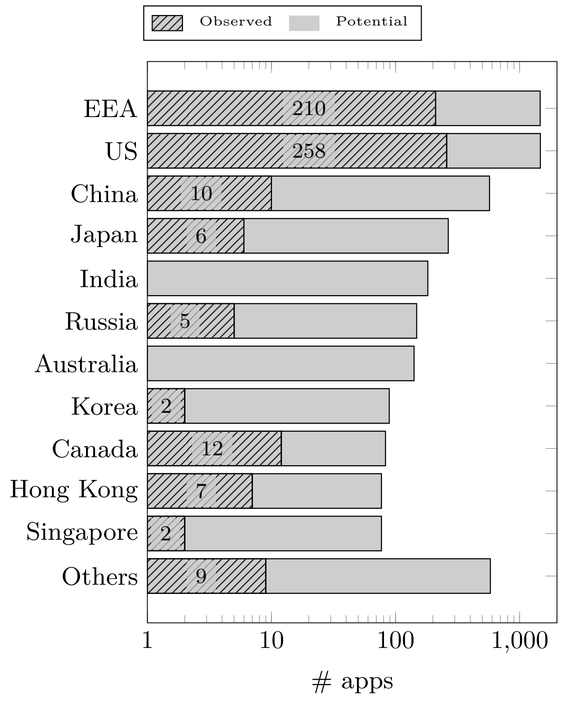

Fig. 6. The target countries per apps.

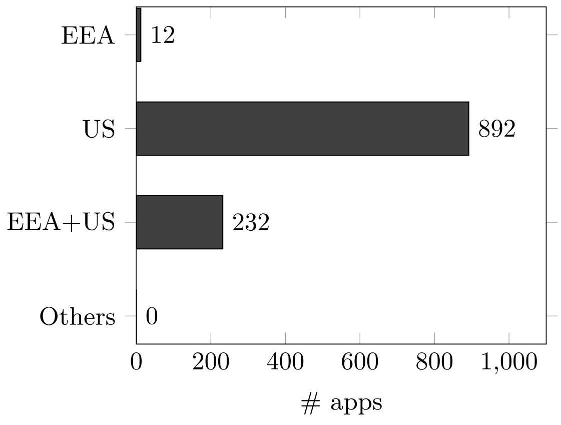

Fig. 7. The number of apps that transfer the personal data exclusively to the EEA, US and other locations.

6.4 Privacy Discussion

The new agreement between the EU and the US, the EU-US Privacy Shield, provides stronger obligation on the US based companies dealing with EU personal data. However, similar to the Safe Harbor, the EU-US Privacy Shield also control only a portion of the entities (service providers, apps) involved in personal data collection/transfer to US and other countries. Figures 8 and 9 depict the results of the analysis we have done on the mobile apps' privacy policy and terms of use. More than half of the most used apps in Europe, 51%, do not provide any privacy policy as shown in Figure 8. They simply do not tell their users what they do to the personal data they collect and where they store and process them.

In the app analysis, we observed that 7% (108) of the apps transfer personal data outside the EEA while do not provide any privacy policy; thus, this is a violation of the DPD'25.1 regulation (**RQ3**). Moreover, the analysis results reveal that 50% of the apps contact servers (potentially transfer personal data) outside the EEA and since these apps do not provide a privacy policy, they should anyways be considered suspicious.

Among all the apps providing privacy policy (49%), we observed that 13% transfer personal data to the non EEA based servers (*e.g.* , the US, China, Russia, etc.) while only 3.5% of them holding safe harbor certification (Figure 9). We concludes that 9.5% (134) of the apps certainly violate DPD'25.1 (**RQ3**) since users did not provide consent for those international data flows, and the available privacy policies are transparent about the data processing locations. Additionally, when we consider the apps which do not provide privacy policy and transfer personal data outside EEA 7% (108), in total, we confirm that 16.5% (242) of the analyzed apps violate this regulation.

One of the major challenges for the Privacy Shield Agreement is that even if we assume that its enforcement will be practical, it will cover only a small portion of mobile apps dealing with European users personal data. The European Data Protection watchdogs would need to have a more proactive role in inspecting compliance with the Data Protection Regulations, in particular for widely used mobile apps.

7 Improving Transparency and Compliance

A number of actions are necessary to improve the control over trans-border personal data flows. It is important that the data protection authorities in Europe demand transparency from the application providers about the location of the data processing. This needs to be explicit in privacy policies. It is vital to clarify which parties have access to personal data and for which purpose. In our study, we observed some applications transferring sensitive personal data items to multiple servers across the globe. In addition to the jurisdictional issue, as all countries do not offer the same level of privacy protection to individuals, it is not possible to state that all those servers belong to the data controller, or even if the data controller is aware of them. Data Protection Authorities (DPAs) need to be proactive in protecting the privacy rights of individuals, by identifying international data which is not compliant with the regulations and agreements in place. The market place provider needs to make privacy policies mandatory, that is the minimum acceptable action that Google, Apple and Microsoft need to take, to mention the main companies who control mobile application marketplaces today. Ideally, the apps should display certification seal, but it is possible to go further. Research on machine readable and automated privacy policy enforcement has shown it is possible to offer more transparency and control to data subjects [7, 8, 11]. Moreover, the marketplace must have a mechanism to promptly remove applications signaled as non-compliant by DPAs or by the users. Furthermore, it is not difficult to implement user notification features on the mobile OS, such that users can remove those non-compliant applications from their devices. On the other hand, it is extremely hard to reclaim the data that has already been leaked.

We are planing to provide PDTLoc as an online service. End-users, marketplaces and security agencies can utilize such service to perform privacy analysis of mobile apps.

8 Limitations

PDTLoc determines the physical location of the remote servers by employing a third party service *e.g.*, `IPAddressAPI.com`; thus, it relies on the information provided by this service.

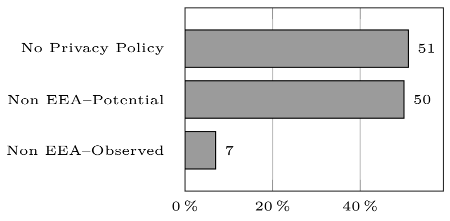

Fig. 8. The apps that do not provide any privacy policy.

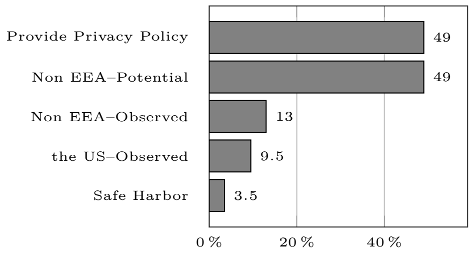

Fig. 9. The apps providing privacy policy.

Our server location analysis of the apps is based on the 1st hop server and do not consider if the personal data might be transferred to another server, e.g., App1 transfers data to `abc.com` and then the data is transferred from `abc.com` to `xyz.com`. In this case, PDTLoc only considers the transfer of data to the 1st server. It is impossible to trace data transfer once they are released to a server without collaboration of the target server. However, the mere transfer of the data towards another jurisdiction without explicit consent by the data subject and for which no international agreements are in place, already represents a violation. Therefore, PDTLoc only considers the transfer of data to the first hop server. Moreover, some applications might behave differently depending upon the location of the device they are running on. Since we have performed dynamic analysis in only one location (i.e., Italy), there is no guarantee about the behavior of such apps elsewhere.

The source and sink APIs considered in this work is a representative list of APIs which can be used by apps to retrieve personal data and transfer it over the network. However, there are other methods to receive personal data and transfer it outside which is not considered in the analysis, e.g., apps can coordinate with each other to acquire and transfer data. Furthermore, we focus on only benign apps rather than malware that usually employ more sophisticated and stealthy methods to exfiltrate personal information.

The data flow paths extracted by the static analysis module only indicate the existence of potential personal data transfer in the app, but do not ensure if the app actually transfers personal data outside. However, even the existence of such data flow paths enables the app potentially violate DPD'25.1 and are, therefore, flagged in this work.

9 Related Work

Literature shows a number of research publications and tools which try to solve the problem of privacy leakages in Android apps. They focus on a wide range of private user data and are based on different strategies. Here we briefly discuss some of them in the context of our problem.

9.1 Static Privacy Leak Detection

A number of static analysis approaches have been proposed in literature which can serve to detect privacy leakage in Android apps. Based on the model of the Android framework, CHEX is an approach that performs data flow analysis to detect component hijacking vulnerabilities [31]. In principle, the same approach can be used to detect also privacy leakages.

Scandal [29] tries to detect leakage of private information, such as location information and phone identifiers, using media including Network, Files and SMS. It is based on identifying data flow using abstract semantics of the applications. Although, it provides a concrete representation of the data flow, it consumes a lot of resources and would therefore suffer from performance and scalability issues.

Androidleaks is a WALA based solution to detect privacy leakages in Android apps [20] [26]. It uses a system dependence graph to perform taint analysis.

Based on bytecode analysis of Android apps, DroidAlarm is a tool designed to counter privilege escalation by detecting capability leaks [47]. It uses control flow graphs to detect and extract capability leak paths from to sensitive sources to public interfaces. However, it only supports Android 2.2 which is quite outdated.

AmanDroid is an inter component-data flow analysis framework for Android apps [42]. It is an extensible tool implemented in Scala and based on an intermediate representation of Dalvik bytecode. Amandroid performs data flow analysis by constructing an inter component data flow analysis graph. It provides a plugin for taint analysis which captures data flow between various sources and sinks of information. The sources and sinks are easily configurable in Amandroid's taint analysis plugin. Theoretically, these sort of tools are ideal for detecting privacy leakage. However, we practically tried it on some apps and it could not detect some very obvious data flows.

Bodden *et al.* presents, Flowdroid, one of the most sophisticated static analysis tool for Androd [6]. It is a Soot based tool which performs data flow analysis on a representation of Java bytecode called Jimple [40]. They also publish a benchmark of applications, known as DroidBench, which can be used to test data flow analysis tools.

Epicc is another static analysis tool which focuses on privacy leakage considering inter component communication and inter app data flows [32]. They provide a cover for privacy leakage between various components of an app and among multiple apps, which most of the static analysis tools do not consider. To make it a complete package, Didfail and IccTa are two other tools which combine Flowdroid and Epicc [9] [30]. They utilize the object/field/context sensitivity of Flowdroid and the inter-component data flow detection Epicc to construct superior tools. This chain of static analysis tools, however, is based on Soot that was designed for Java applications and some times fails to analyze Android apps.

As a matter of fact, some of these static analysis tools capable of detecting privacy leakage can be adopted to be used in our work. However, we preferred performing analysis on Smali code as it provides a direct representation of the Dalvik bytecode. Therefore, we used an extension of SAAF that performs analysis on Smali code and is based on backward program slicing of apps. The extension of SAAF overcomes some of its limitations, such as handling data flow through intents.

9.2 Dynamic Privacy Leak Detection

As static analysis usually suffers from over-approximation and, therefore, a higher number of false alarms, dynamic analysis solutions provide the answer.

SmartDroid detects sensitive APIs in an app, creates a static activity switch path and control flow paths leading to these sensitive APIs and dynamically executes these paths to generate trigger inputs which could be used to detect privacy leakage [46]. They rely on instrumentation of framework services to ensure dynamic execution.

TaintDroid is one of the most widely cited tools in Android dynamic privacy leakage detection [17]. It is based on tainting sensitive information and tracking it towards sensitive sources. Similarly, Droidbox is another tool which detects privacy leakage in Android apps by executing them in an emulator [15]. However, such tools require a dynamic triggering solution to effectively execute portions of the code that leak private information.

To counter this problem, some tools provide their own triggering solution along with privacy leakage detection, e.g., AppsPlayground, AppIntent, etc. [33, 43].

However, most of these tools still suffer from code coverage issues and increasing the code coverage when analyzing Android apps is an open research problem. Moreover, Shauvik *et al.* performed an analysis based study of the state-of-the-art open sourced test input generation tools for Android applications [12]. Surprisingly, random exploration strategies based tools performed far better than the other model based and systematic tools.

Since even the more sophisticated tools do not yield considerable improvement in the code coverage and unnecessarily complicate the process, we use the standard application exerciser provided with the Android SDK,*i.e.*, the Monkey tool, in the dynamic analysis module.

10 Conclusions

This paper makes a substantial contribution in the analysis of trans-border personal data flows. It is a major debate that may impact how the regulatory framework around the digital economy will evolve. We have highlighted the main concerns in personal data transfers by in principle non-malicious applications, and shown a considerable number of them fail to comply with the EU personal data protection regulation, in the first study of the kind, up to our knowledge. While PDTLoc has been suitable in this case, we believe it can be extended to analyze other information flow properties as well.

Acknowledgment

This work has been partly supported by the EU under grant 317387 SECENTIS (FP7-PEOPLE-2012-ITN).

We thank Nicolas Dolgin for his comments on a previous version of this work.

References

[1] European commission - overview on binding corporate rules. *http://ec.europa.eu/justice/data-protection/international-transfers/binding-corporate-rules/index_en.htm*, 2016.

[2] European Commission - press release: EU-US Privacy Shield. *http://europa.eu/rapid/press-release_IP-16-216_en.htm*, 2016.

[3] Jagdish Prasad Achara, Franck Baudot, Claude Castelluccia, Geoffrey Delcroix, and Vincent Roca. Mobilitics: Analyzing privacy leaks in smartphones. *ERCIM News*, 2013(93), 2013.

[4] Tina Amirtha. Safe Harbor was for EU privacy: But how safe is US data in Europe? *http://www.zdnet.com/article/safe-harbor-was-for-eu-privacy-but-how-safe-is-us-data-in-europe/*, 2015.

[5] AppFigures. A tracking platform to monitor the sales and downloads of apps. *http://AppFigures.com*.

[6] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *ACM SIGPLAN Notices*, volume 49, pages 259–269. ACM, 2014.

[7] Monir Azraoui, Kaoutar Elkhiyaoui, Melek Önen, Karin Bernsmed, Anderson Santana Oliveira, and Jakub Sendor. *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance: 9th International Workshop, DPM 2014, 7th International Workshop, SETOP 2014, and 3rd International Workshop, QASA 2014, Wroclaw, Poland, September 10-11, 2014. Revised Selected Papers*, chapter A-PPL: An Accountability Policy Language, pages 319–326. Springer International Publishing, Cham, 2015.

[8] Walid Benghabrit, Hervé Grall, Jean-Claude Royer, Mohamed Sellami, Monir Azraoui, Kaoutar Elkhiyaoui, Melek Önen, Anderson Santana Oliveira, and Karin Bernsmed. *Cloud Computing and Services Sciences: International Conference in Cloud Computing and Services Sciences, CLOSER 2014 Barcelona Spain, April 3–5, 2014 Revised Selected Papers*, chapter From Regulatory Obligations to Enforceable Accountability Policies in the Cloud, pages 134–150. Springer International Publishing, Cham, 2015.

[9] Johnathon Burket, Lori Flynn, Will Klieber, Jonathan Lim, and William Snavely. Making DidFail Succeed: Enhancing the CERT Static Taint Analyzer for Android App Sets. 2015.

[10] Mary Carolan. Data protection commissioner to investigate max schrems claims. *http://www.irishtimes.com/news/crime-and-law/courts/high-court/data-protection-commissioner-to-investigate-max-schrems-claims-1.2398728*, 2015.

[11] F. Di Cerbo, D. F. Some, L. Gomez, and S. Trabelsi. Ppl v2.0: Uniform data access and usage control on cloud and mobile. In *TEchnical and LEgal aspects of data pRivacy and SEcurity, 2015 IEEE/ACM 1st International Workshop on*, pages 2–7, May 2015.

[12] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. Automated Test Input Generation for Android: Are We There Yet?(E). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 429–440. IEEE, 2015.

[13] Fred Chung. Custom Class Loading in Dalvik. http://android-developers.blogspot.it/2011/07/custom-class-loading-in-dalvik.html.

[14] Court of Justice of the European Union. The court of justice declares that the commission's us safe harbour decision is invalid. *http://curia.europa.eu/jcms/upload/docs/application/pdf/2015-10/cp150117en.pdf*, 2015.

[15] Anthony Desnos and Patrik Lantz. Droidbox: An android application sandbox for dynamic analysis (2011). *https://code.google.com/p/droidbox*, 2014.

[16] Serge Egelman, Adrienne Porter Felt, and David Wagner. Choice architecture and smartphone privacy: There'sa price for that. In *The economics of information security and privacy*, pages 211–236. Springer, 2013.

[17] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014.

[18] Ericsson. Europe mobility report appendix. *http://www.ericsson.com/res/docs/2014/emr-november2014-regional-appendices-europe.pdf*, 2014.

[19] European Court of Justice. Commission Decision of 26 july 2000 pursuant to directive 95/46/ec of the european parliament and of the council on the adequacy of the protection provided by the safe harbour privacy principles and related frequently asked questions issued by the us department of commerce. *Official Journal L 215 , 25/08/2000 P. 0007 - 0047 URL: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32000D0520:EN:HTML*, 2000.

[20] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. *AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale*. Springer, 2012.

[21] Google. Monkey Tool. *http://developer.android.com/tools/help/monkey.html*, 2015.

[22] Ben Gruver. Smali/Baksmali Tool. *https://github.com/JesusFreke/smali/wiki*, 2015.

[23] Dominik Herrmann and Jens Lindemann. Obtaining personal data and asking for erasure: Do app vendors and website owners honour your privacy rights? *CoRR*, abs/1602.01804, 2016.

[24] Paul De Hert and Vagelis Papakonstantinou. The proposed data protection Regulation replacing Directive 95/46/EC: A sound system for the protection of individuals. *Computer Law & Security Review*, 28(2):130–142, 2012.

[25] Johannes Hoffmann, Martin Ussath, Thorsten Holz, and Michael Spreitzenbarth. Slicing Droids: Program Slicing for Smali Code. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1844–1851, New York, NY, USA, 2013. ACM.

[26] IBM. Watson libraries for analysis. *http://wala.sourceforge.net/wiki/index.php*.

[27] IDC Press Release. Smartphone os marketshare. *http://www.idc.com/prodserv/smartphone-os-market-share.jsp*.

[28] IPaddressAPI.com. An ip location api solution. *http://www.ipaddressapi.com/*, 2015.

[29] Jinyung Kim, Yongho Yoon, Kwangkeun Yi, Junbum Shin, and SWRD Center. ScanDal: Static analyzer for detecting privacy leaks in android applications. *MoST*, 12, 2012.

[30] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. IccTA: Detecting inter-component privacy leaks in Android apps. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 280–291. IEEE Press, 2015.

[31] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. Chex: statically vetting android apps for component hijacking vulnerabilities. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 229–240. ACM, 2012.

[32] Damien Octeau, Patrick McDaniel, Somesh Jha, Alexandre Bartel, Eric Bodden, Jacques Klein, and Yves Le Traon. Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis. *Effective Inter-Component Communication Mapping in Android with Epicc: An Essential Step Towards Holistic Security Analysis*, 2013.

[33] Vaibhav Rastogi, Yan Chen, and William Enck. AppsPlayground: automatic security analysis of smartphone applications. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 209–220. ACM, 2013.

[34] European Parliament. Directive 95/46/ec of the european parliament and of the Council of 24 october 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *http://eur-lex.europa.eu/eli/dir/1995/46/oj*.

[35] IDC Press Release. Worldwide smartphone market will see the first single-digit growth year on record, according to idc. *http://www.idc.com/getdoc.jsp?containerId=prUS40664915*, 2015.

[36] Brian Cantwell Smith. *Procedural Reflection in Programming Languages*. PhD thesis, Massachusetts Institute of Technology, Laboratory for Computer Science, 1982.

[37] David Sounthiraraj, Justin Sahs, Garret Greenwood, Zhiqiang Lin, and Latifur Khan. Smv-hunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps. In *In Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS'14*. Citeseer, 2014.

[38] The Tcpdump Group. TCP-Dump. *http://www.tcpdump.org/*, 2015.

[39] Connor Tumbleson and Ryszard Wiśniewski. APK tool - a tool for reverse engineering android apk files. *http://ibotpeaches.github.io/Apktool/*.

[40] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. Soot-a Java bytecode optimization framework. In *Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research*, page 13. IBM Press, 1999.

[41] VirusTotal. Free online virus, malware and url scanner. *https://www.virustotal.com*.

[42] Fengguo Wei, Sankardas Roy, Xinming Ou, et al. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1329–1341. ACM, 2014.

[43] Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X Sean Wang. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1043–1054. ACM, 2013.

[44] Sara Zaske. Germany's privacy leaders gather to discuss suspending us safe harbor. *http://www.zdnet.com/article/germanys-privacy-leaders-gather-to-discuss-suspending-us-safe-harbor/*, 2015.

[45] Yury Zhauniarovich, Maqsood Ahmad, Olga Gadyatskaya, Bruno Crispo, and Fabio Massacci. Stadyna: addressing the problem of dynamic code updates in the security analysis of android applications. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 37–48. ACM, 2015.

[46] Cong Zheng, Shixiong Zhu, Shuaifu Dai, Guofei Gu, Xiaorui Gong, Xinhui Han, and Wei Zou. Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, pages 93–104. ACM, 2012.

[47] Yibing Zhongyang, Zhi Xin, Bing Mao, and Li Xie. DroidAlarm: an all-sided static analysis tool for Android privilege-escalation malware. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 353–358. ACM, 2013.