PPAndroid-Benchmarker: Benchmarking Privacy Protection Systems on Android Devices

Saeed Ibrahim Alqahtani
University of Surrey, UK
Taibah University, Saudi Arabia
s.alqahtani@surrey.ac.uk

Shujun Li
University of Surrey, UK
shujun.li@surrey.ac.uk
http://www.hooklee.com/

ABSTRACT

Mobile devices are ubiquitous in today's digital world. While people enjoy the convenience brought by mobile devices, it has been proven that many mobile apps leak personal information without user consent or even awareness. That can occur due to many reasons, such as careless programming errors, intention of developers to collect private information, infection of innocent apps by malware, etc. Thus, the research community has proposed many methods and systems to detect privacy leakage and prevent such detected leakage on mobile devices. This is a to do note at margin While it is obviously essential to evaluate the accuracy and effectiveness of privacy protection systems, we are not aware of any automated system that can benchmark performance of privacy protection systems on Android devices. In this paper, we report PPAndroid-Benchmarker, the first system of this kind, which can fairly benchmark *any* privacy protection systems *dynamically* (i.e., in *run time*) or *statically*. PPAndroid-Benchmarker has been released as an open-source tool and we believe that it will help the research community, developers and even end users to analyze, improve, and choose privacy protection systems on Android devices. We applied PPAndroid-Benchmarker in dynamic mode to 165 Android apps with some privacy protection features, selected from variant app markets and the research community, and showed effectiveness of the tool. We also illustrate two components of PPAndroid-Benchmarker on the design level, which are *Automatic Test Apps Generator* for benchmarking static analysis based tools and *Reconfigurability Engine* that allows any instance of PPAndroid-Benchmarker to be reconfigured including but not limited to adding and removing information sources and sinks. Furthermore, we give some insights about current status of mobile privacy protection and prevention in app markets based upon our analysis.

KEYWORDS

Android, mobile apps, privacy leakages, privacy protection, benchmarking, performance evaluation, static analysis, dynamic analysis

ARES '17, Reggio Calabria, Italy

978-1-4503-5257-4/17/08…$15.00
DOI: 10.1145/3098954.3098984

ACM Reference format:
Saeed Ibrahim Alqahtani and Shujun Li. 2017. PPAndroid-Benchmarker: Benchmarking Privacy Protection Systems on Android Devices. In *Proceedings of ARES '17, Reggio Calabria, Italy, August 29 – September 1, 2017,* 10 pages.
DOI: 10.1145/3098954.3098984

1 INTRODUCTION

Mobile devices' usage has increased rapidly over the recent years. The wide range of mobile devices' possession has been accompanied by the rise of mobile apps e.g. Google Play is now offering over 3 million apps as the time of this writing [1]. Mobile users can enjoy a variety of apps, which provide appropriate services and attractive features by making use of smart devices' capabilities.

Undoubtedly, the advance in mobile technologies opens the door for new privacy issues and concerns. The consequences of privacy violations on individuals have led research and industry to focus heavily on user privacy. In some cases, malicious identities may exploit users' data to steal or uncover personal information about them. In some other cases, intruders may misuse collected sensitive data to financially or socially harm users. Moreover, data can be used by companies to identify personal information about users without their consent. Consequently, the research community has proposed many solutions to overcome privacy and security obstacles. Many protection mechanisms and detective approaches have been offered to detect and prevent mobile privacy leakages.

However, existing solutions have limitations. For instance, static analysis based methods mostly require access to source code. On the other hand, dynamic analysis based approaches are time-consuming and normally cannot cover all possible privacy leakages [10]. Another common limitation is that many of these tools use a limited and static list of information sources, leaving other sources uncovered.

For each new privacy protection solution, there is always a problem of how to evaluate its performance against existing solutions. Similarly, given a number of candidate solutions, a user has the need to know which solution is the best for her specific needs. For researchers, a proper benchmarking system is also desired so that insights about how to improve exist solutions can be gained and the performance of any new solution can be properly evaluated. While such benchmarking systems are very important, surprisingly, we could not find any such systems in the research or commercial worlds, not mentioning open-source tools. Instead, currently researchers and developers either depend on bespoke performance evaluation apps or collections of test apps to conduct such benchmarking tasks (e.g. DroidBench [2, 23]).

In this paper, we present PPAndroid-Benchmarker, a system for benchmarking mobile privacy protection systems on Android devices, which to our best knowledge is the first of this kind. PPAndroid-Benchmarker is designed to be oblivious to details of mobile privacy protection systems, and can detect performance of combined mobile privacy protection apps in run time – this is why we use the term "systems" rather than "apps". It is effectively a system 1) simulating different kinds of tester-configurable privacy leakage activities; 2) capturing what leakage attempts were successful; 3) supporting a high level of automation for the whole benchmarking life-cycle; and 4) providing a good user interface for testers to configure benchmarking tasks. Moreover, this paper also highlights two additional components of PPAndroid-Benchmarker at the design level (which have not been implemented in our prototype): 1) an Automatic Test Apps Generator for benchmarking static analysis based privacy protection systems; and 2) a Reconfigurability Engine allowing PPAndroid-Benchmarker to be reconfigured such as adding and removing information sources and sinks. Although we implemented a prototype system for Android OS only, the framework is generic enough to be applied to other mobile operating systems such as iOS. We tested PPAndroid-Benchmarker with 165 selected Android privacy protection apps, and report our findings and some insights about current status and future directions of mobile privacy protection and prevention tools.

The rest of the paper is organized as follows. Section II discusses related work. The design of our system is described in Section III. Section IV presents the experimental set-up we followed and Section V illustrates the results and analysis. Section VI discusses the implications of the findings and some limitations. Lastly, Section VII concludes the paper and illustrates some future work.

2 RELATED WORK

This section is divided into two subsections, mobile privacy protection techniques and mobile privacy benchmarking systems. In the former subsection, several privacy protection approaches are presented for the purpose of understanding how privacy protection systems work. The latter subsection illustrates a few similar systems and related works from research.

2.1 Application Analysis Based Approaches

The research community has proposed various techniques of mobile application analysis many of which are based on static, dynamic and hybrid analysis of computer programs.

In static analysis based approaches, the analysis is carried out statically (i.e., without running the app) with (source code analysis) or without source code (binary code analysis). Almost all mobile detection tools that follow this approach retrieve the target app's source code using a decompiler, and rely on its precision. For instance, **ScanDal** [15] proposed by Kim et al. is an example of the static analysis category, and there are many others [13, 27]. ScanDal implements a static analysis to detect data leakages by converting a given app package from Dalvik byte-code to a pre-defined intermediate language. Using abstract interpretations, suspicious flows can be detected. Kim et al. analyzed 90 free apps, and they found 11 of them leak private data. They reported many location leakages to remote ad servers, namely AdMob and AdSenseSpec. Besides location data, many apps were found to leak IMEI to their app servers.

On the other hand, dynamic analysis is based on observing dynamic behaviors of the target app in run time. By performing different kinds of dynamic analysis techniques such as data flow analysis (DFA) in mobile platforms, dynamic tools monitor sensitive data sources and detect data leaks once the target app is executed. **TaintDroid** [7] is one of the most well-known tools in this category. Many other tools are built on top of **TaintDroid** by adding new features, such as MockDroid [4], TISSA [28] and AppFence [14]. TaintDroid, applies a specific dynamic analysis technique called taint tracking to detect potential data leaks. Sensitive information sources are identified and tainted. Using DFA, TaintDroid monitors how apps handle tainted information and notify the user if sensitive data is leaked outside the device. One problem of TaintDroid and some other dynamic analysis based mobile privacy protection tools is that it requires changes to mobile operating systems, which may not be possible or difficult in some applications. To overcome this problem, Rastogi et al. proposed **Uranine** [21], a system for instrumenting existing apps so that taint tracking can be done without changes to mobile operating systems.

In hybrid analysis methods, static and dynamic analysis techniques are combined to improve privacy leakage detection [18]. For instance, **SmartDroid** [26] combines static and dynamic analysis in a two-level process. At the first level, it utilizes a static path selector to identify activity switch paths through analyzing activity and function call graphs. At the second level, SmartDroid implements a dynamic analysis to traverse each UI element and explore the UI interaction paths towards sensitive APIs.

In addition to code analysis and dynamic app behavior monitoring, the app package manifest file is another important source of information for inferring potential privacy problems in mobile apps. The app package manifest file contains permission labels and some other useful information. Mobile platforms apply a permission framework that allows users to control mobile apps' access to sensitive information and resources such as geo-locations and help them to make proper decisions prior to app installation. Researchers have proposed many methods of identifying suspicious apps by analyzing permissions they request. Enck et al. proposed **Kirin**, a system that can detect potentially malicious apps by matching some pre-defined security rules with information in the package manifest file [8]. A similar system called **Stowaway** was proposed by Felt et al. [11], which identifies malicious apps based on over-privileges. They analyzed 956 apps from Google Play where they found that most common unnecessarily requested permissions are accessing the Internet and reading phone state.

In another work [12], Gates et al. proposed a permission analysis based approach to producing risk signals and risk scores for apps to help users identify risky apps. Zhang et al. presented **VetDroid** [25] which falls in the same category. It is a tool that reconstructs sensitive apps behaviors from a permission use perspective. It applies a dynamic analysis approach and presents a systematic framework in constructing permission use behaviours.

Yet another way of capturing suspicious behaviors is by monitoring and analyzing network traffic associated with an app. For instance, **AntMonitor** [16] is a traffic monitoring system for passive monitoring, collection, and analysis of network traffic of Android

devices. Using a VPN service and a special interface, AntMonitor can analyze all outgoing traffic from apps to hosts, as well as incoming packets from hosts. AntClient, a component of AntMonitor running from the user's device, lets the user know if installed apps are leaking her personally identifiable information. **AGRIGENTO** [6] is another example falling in this category. This system is mainly designed to detect privacy violations using a black-box differential analysis technique. By creating a network behavior baseline of each app and modifying information sources, AGRIGENTO can capture privacy leaks (i.e., deviations from the baseline behavior) in the resulting network traffic.

Some researchers also paid attention to justifiable sensitive information transmission as part of an app's functionality. For instance, Chen and Zhu proposed **DroidJust** [5], an automated technique that can differentiate justifiable sensitive information transmission and privacy leakage by malicious apps. In [9], Fan et al. proposed four metrics (possibility, severity, crypticity, and manipulability) to quantitatively analyze privacy leak behaviour. The authors showed the effectiveness of the proposed metrics in revealing apps' characteristics in several aspects.

2.2 Mobile Privacy Benchmarking Systems

In order to help users decide what tools to use and how to use them, privacy protection tools need to be carefully validated and their performance compared with each other. Surprisingly, while there is a lot of work on developing mobile privacy protection tools, there is very little work on benchmarking mobile privacy protection tools. This subsection focuses on DroidBench, the closest work comparing to the system we propose in this paper. In addition, we also introduce some other loosely related work.

DroidBench [2] is a collection of different Android apps from various categories that deliberately leak many types of data. Additionally, some apps without data leakages are also included to allow detection of false positives. DroidBench tests if Android privacy protection tools could detect such privacy leaks when they occur or the privacy-leaking apps before any leakage attempts take place.

Stanford SecuriBench [19] is very similar to DroidBench but include Java web-based applications rather than Android apps. It consists of eight open-source Web-based Java applications that have intentional security flaws. They purposely suffer from a number of vulnerabilities, including SQL injections, cross-site scripting, HTTP splitting and path traversal attacks. This tool is meant to serve as test cases for researchers and practitioners to study solutions against such applications with security flaws.

There have been some efforts in benchmarking ad hoc privacy protection tools or techniques on mobile platforms. In **Anti-TaintDroid** (also called ScrubDroid) [22] Sarwar et al. investigated the limitations of TaintDroid in tracking sensitive information leaks on Android mobile devices. Multiple attacks on TaintDroid were demonstrated using a number of generic classes of anti-taint methods.

3 DESIGN AND IMPLEMENTATION

This section illustrates the design and some implementation details of the proposed benchmarking system.

3.1 Overall Design

The main purpose of PPAndroid-Benchmarker is to evaluate privacy leakage detection applications in an automated manner. As stated earlier, the Android platform is targeted in this work. PPAndroid-Benchmarker is composed of three basic components, the benchmarker app, the drop-in server and the PC-based mobile device manager (MDM). Figure 1 shows the architecture of PPAndroid-Benchmarker. Firstly, the benchmarker is programmed to simulate leakages of different types of private information. It has a profile creator that allows the user to define different benchmarking tasks. Secondly, the drop-in server is used to receive leaked information from the benchmarker. Lastly, the MDM handles automatic installation and uninstallation of tested apps during the benchmarking process.

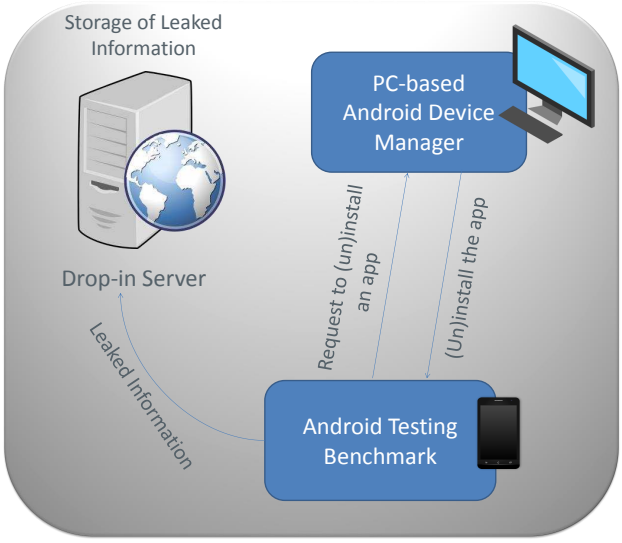

Figure 1: PPAndroid-Benchmarker's overall design.

3.2 PPAndroid-Benchmarker Components

Figure 2 shows how PPAndroid-Benchmarker's different components interact with each other as data flows between them, where the user is also shown as a "component" as his/her interactions with several components are needed. In the following, we explain the three key components with greater details.

The **Profile Creator** allows a user (who wants to tests some privacy protection apps) to create the actual benchmarking task (i.e., a benchmarking profile). Any profiles created can be stored in a profile database so that they can be reused in future. The profile database can also retrieve app-related information from an **Apps Repository** which will also work with the **MDM** (to be explained below). The Profile Creator interacts with the user to collect information about a benchmarking profile and feed the created profile to the **Benchmarker App** for execution. The profiles are stored as XML files to make them accessible from other components and external applications more easily. The Profile Creator can be made part of the Benchmarker App or be implemented as a PC-based

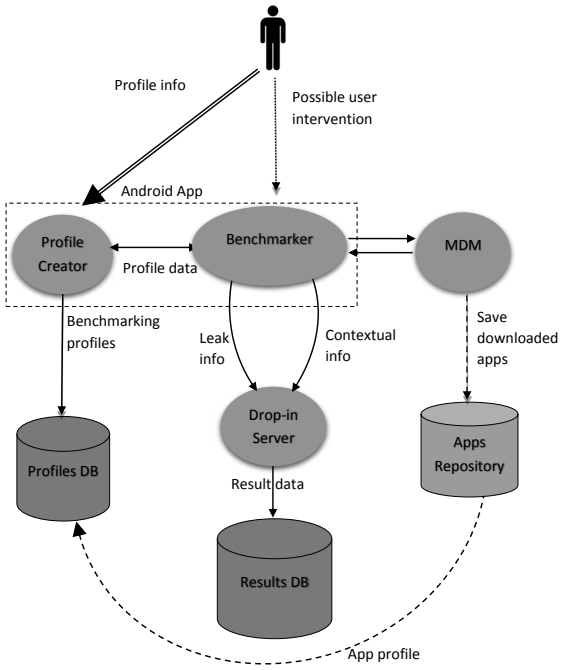

Figure 2: PPAndroid-Benchmarker's components and data flow map.

application which communicate with the Benchamarker App via USB or a wireless channel.

The **Benchmarker App** is the core of PPAndroid-Benchmarker and its main purpose is to simulate leakages of different types of private information. It is programmed to collect a variety of private information from the hosting mobile device. Thus, this app needs to be granted with all required permissions in order to access all data sources. This is not an issue as the benchmarker is used for testing purposes only, and can run on a dedicated testing device or within an emulator. The Benchmarker App is programmed to leak information to a **Drop-in Server**, which is a web server set up to simulate an attacker's information collection server. The Drop-in Server is designed to collect all needed information to create results of each benchmarking profile, which are stored in a **Results Database** for further (off-line) analysis. The Benchmarker App is connected to the MDM in order to facilitate the automation of the benchmarking process.

Furthermore, the Benchmarker App is facilitated with many anti-tainting tricks. Many designed dynamic analysis tools apply taint tracking technique, e.g., TaintDroid, MockDroid, AppFence and many others. Therefore, we added several tricks, which are constructed to bypass dynamic taint tracking. In our current implementation, we include the following tricks reported in Anti-TaintDroid [22], which were verified to be still valid for the Android and TaintDroid versions we tested.

- simple encoding trick
- shell command trick
- file+shell hybrid trick
- time keeper trick
- count-to-X trick
- file length trick
- clipboard trick
- exception/error trick
- remote control trick
- file last modified trick

The **Mobile Device Management (MDM)** is a PC-based Android device manager handling *automatic* installation and uninstallation of each tested privacy protection app during the testing process. We implemented the MDM to communicate with the Benchmarker App via a TCP port, although other communication channels can also be used. When it receives an app download link, it downloads it and installs it to the mobile device using ADB (Android Device Bridge). After the benchmarking process ends at the Benchmarker App's side, the MDM will receive a request of uninstalling the tested app.

Drop-in Server: In order to simulate the complete information leakage process, a sink is required to allow information to go out of the mobile device. In PPAndroid-Benchmarker, the Internet connection is used as the sink. To simulate the case of information leaked through the Internet, we need to set up a server that receives leaked information. A number of server-side scripts (written in PHP in our implementation) are used to handle received leakage information, some are used to receive leaked files, and some others to create the results as XML files.

We implemented a prototype of PPAndroid-Benchmarker including all the above components. At the time of this writing, our prototype supports the following information sources:

- **Device IDs**: IMEI, IMSI, Android_ID
- **Personal data**: SMS messages, contacts, call logs and browsing history
- **Sensor data**: camera, microphone, accelerometer axes data, last known geolocation by GPS device or the ISP (Internet service provider)
- **Files** on the mobile device's external memory storage

The above list is not supposed to be complete, but used as a representative start of our prototype system. Adding more sources is a matter of improving the tool itself. To simplify our prototype system, we do not apply any advanced processing of information leaked (e.g. encryption and steganography) other than some tricks specially added to circumvent taint tracking techniques. Adding more advanced information processing operations to information leaked will not be difficult, but require a proper interface to allow easy reconfiguration (see below).

Our design also considers two other major conceptual components, the **Automatic Test Apps Generator** and the **Reconfigurability Engine**, which have not been implemented in our current prototype system yet but will be added in future versions of PPAndroid-Benchmarker prototype.

3.3 Automatic Test Apps Generator

PPAndroid-Benchmarker is designed to benchmark privacy protection apps more in a dynamic way. To support benchmarking of static analysis tools, an automatic **Test Apps Generator** can be developed to allow generation of apps with different privacy

leakage capabilities. The Test Apps Generator will take the source code of the Benchmarker App as the source and the user's descriptions of the test apps wanted, and generate a number of apps with requested privacy leakage capabilities. The process of generating test apps can contain random factors so that a large number of test apps can be generated, which will produce much more test cases for static analysis based tools than other solutions can provide. The generated apps (in the form of apk files) can be used to benchmark any static analysis based privacy protection systems. This component can be achieved in several ways such as embedding a compiler that can automatically convert the source code of the Benchmarker App into a subset representing the needed privacy leakage profile and then compile the resulting source code to a mobile app. The compiler may be implemented as part of the MDM as well.

3.4 Reconfigurability Engine

Any instance of PPAndroid-Benchmarker can only cover a limited number of sources and sinks and limited settings for benchmarking profiles. To allow extension of supported features and reconfiguration of the system (including removing some unwanted features), a **Reconfigurability Engine** can also be developed.

A major part of the Reconfigurability Engine is addition and removal of sources and sinks. This can be achieved by defining a dynamic list of sources and sinks for PPAndroid-Benchmarker to process. The dynamic list needs to support both descriptions of sources and sinks and also code for accessing the sources and sinks. One way of supporting such a dynamic list is to have an XML file for the sources and another one for the sink, and the binary code for accessing each source and sink is provided in the form of an executable plug-in following a defined API. Another way of achieving this is to provide source code of new sources and sinks directly with description files, and a compiler is used to re-compile the whole system into a new instance of PPAndroid-Benchmarker.

Another part of the Reconfigurability Engine is changing how the system behaves e.g. how to automatically configure some privacy protection apps requiring human intervention, which can be achieved by defining other configuration files or APIs so that different types of plug-ins can be added.

Another major part of the Reconfigurability Engine is to add and reconfigure more information processing operations and tricks against static and dynamic analysis techniques. Our prototype has included a number of tricks mainly for testing TaintDroid. Adding more will require a different type of API and plug-in system so that any operation can be added between any pair of source and sink, which will need to work along with the API/plug-in systems for sources and sinks.

3.5 Interaction between Components

The interactions between different components can be explained by how a typical benchmarking task looks like. At the beginning, the user will fill a test profile to tell the benchmarker about details of the test. It will include information such as types of leaked data, mobile device specifications, privacy apps to be tested and others. The profiles are written in XML as stated. The profile creator keeps XML files in the Profiles Database in order to be used for analysis. Then the Benchmarker App starts communicating with the MDM to request installing each tested app. The MDM searches for the required app and installs it. MDM keeps records of installed apps in the Apps Repository for future use. Once the Benchmarker App receives a signal of starting the test, it will initiate leakage attempts to the Drop-in Server. The latter keeps the results as XML profiles in the Results Database.

4 EXPERIMENTAL SETUP

In this section, the method we followed to set up the experiment is explained. It starts with describing how tested apps were collected, followed by how special apps were handled. Lastly, an explanation of variant settings and implementation is provided.

4.1 Selection of Privacy Protection Apps

The first step of the experiment was to identify and collect Android apps with some real-time privacy protection features. These tools were collected from variant sources. Many of them were gathered from Google Play store as it is the main source for Android apps. The following steps were taken to collect the apps. Firstly, Google's search engine was used to look for related privacy apps. Many keywords were used in this step. For example, we used "privacy", "security", "private", "protection", "leak", "dynamic analysis", "static analysis", "leakages" and several others. Secondly, some major third-party Android markets have been explored such as Amazon Appstore, GetJar, Slide ME, F-Droid, Samsung Galaxy Apps, AppsLib, Mobogenie and a few others. Thirdly, many related apps have been collected from cyber security product vendors and service providers. The list of these vendors were taken from AV-Comparatives website [3]. There are around 50 mobile security companies like AVG, AegisLab, Bitdefender, etc. Lastly, some data leakage protection and Android forensics tools are included such as TaintDroid, NowSecure forensic tool [20], PrivacyProtecter app reported in [17]. We wanted to test all privacy protection apps we found but not all of them are available or provide real-time protection. At the time of this writing, in total 165 privacy protection apps have been collected and tested. According to the sources, these tools can be categorized as follows:

- Apps dedicated for privacy protection (from Google Play),
- security apps with privacy protection features (from Google Play),
- apps from third-party markets,
- security vendors' apps (those not covered in the above categories),
- privacy related apps developed by researchers.

Functionally speaking, those tools can be categorized into three different groups:

(1) apps that try to detect privacy violations at installation time,
(2) apps that detect privacy violations based on blocking access to sensitive information sources,
(3) real-time dynamic monitoring tools requiring changes to the Android system.

4.2 Testing Procedure and Settings

In our experiment, the testing procedure covers three different scenarios: fully automated testing without user intervention, semi-automated testing with user intervention, and testing access-related analysis apps. For the fully automated scenario, the tester (as a user) is involved to select target apps for testing and define the benchmarking tasks only. For the semi-automated scenario, the tester is also involved in the process of installing process because some apps require manual configuration before they can run properly. For the last scenario, the Benchmarker App will attempt to access private sources first. Then, if the access is granted, the app will proceed with the actual leakage.

PPAndroid-Benchmarker has been equipped with some options to increase the configuration power. For instance, the user can set time-outs for the evaluation test. Moreover, the user can set wait times between data acquisition by the benchmarker and data leaking attempts.

The architecture of PPAndroid-Benchmarker is intended to work in an automated manner as much as possible including automatically downloading and installing a target app. However, download links of some apps cannot be automatically determined, so the apk files must be provided manually by the user Moreover, some apps are marked for manual configuration and initialization. For instance, some apps require manual registration, accepting terms or connecting to the cloud. To ensure all target apps were tested with the most appropriate configuration settings, we tested settings with and without user intervention manually and then labelled each app with the best setting accordingly.

While testing apps which require *Root* access, we noticed that there is a need of different settings. In this case, PPAndroid-Benchmarker is programmed to also record if access to each target data source is blocked. That allows PPAndroid-Benchmarker to know if a recorded failure of a privacy leakage attempt was blocked at the access level or at a later stage.

Most apps can be tested with PPAndroid-Benchmarker without a special treatment. However, there are a few tools that must be tested using a different procedure or special settings. For example, TaintDroid does not work as a stand-alone application. It involves building a custom-built operating system on the tested device/emulator. Thus, a customized version of Android was built to test TaintDroid against our system in an emulator. Accordingly, we tested Anti-TaintDroid (ScrubDroid) in the same environment, too.

In our experiment, we chose not to test apps based on traffic analysis. Testing such apps requires some significant changes to the architecture and procedures of PPAndroid-Benchmarker, and we leave this as our future work.

4.3 Special Benchmarking Profiles

In this subsection, we discuss some special benchmarking profiles we used in our experiment.

Baseline Benchmarking Profile: The Android system itself may already has some privacy protection mechanism so that some privacy leakage attempts can be detected and blocked at the operating system level. In our experiment, we always ran a baseline profile without any third-party privacy protection app first. When each privacy protection app was tested, only those new successes in detecting privacy leakages were counted. For the Android version we ran our experiment, none of the privacy leakage attempt was detected by the Android system, but this may change in future versions.

Probing Phase: As mentioned above, some apps may require user intervention. A probing phase was therefore added to test if an app behaves differently with and without user intervention. Figure 3 shows how this phase was conducted.

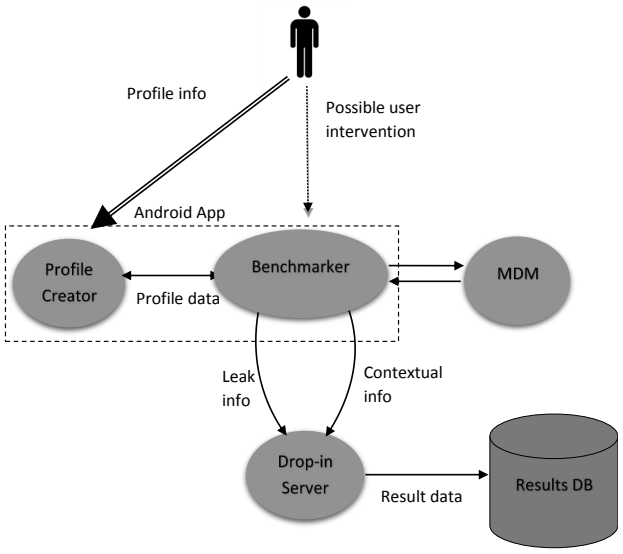

Figure 3: PPAndroid-Benchmarker in probing phase.

Access-Based Apps: Some apps block privacy leakages by blocking access to sensitive information sources. Therefore, a special benchmarking task is added to test if a privacy leakage attempt is blocked at the source access level or afterwards. The Benchmarker App will access each sources in the benchmarking profile and check if there is a response from the tested app, and if the access goes through it will proceed with the actual leaking attempt.

5 RESULTS & ANALYSIS

This section presents the analysis results gathered from testing collected apps and tools. We report the results for the three functional categories of privacy protection apps, respectively, since the behaviors of apps in each category are similar.

5.1 Probing Phase Results

During the study, we noticed that some apps need to be configured before using to ensure all included privacy protection means and settings are enabled. A small experiment was conducted for nine apps, selected from distinct categories. Seven out of nine asked for user input in variant ways. For example, some asked the user to accept terms, to slide a few ad pages, to register on-line or to wait for configuration. We also calculated the time spent to set up each app after installation. Table 3 shows the findings of this small pilot study. As a consequence, the actual benchmarking tasks were

Table 1: Testing Installation-Time Privacy Apps

Apps Category	Sensitive Information Source								
	Device IDs	Accelerometer	Contacts	Location	SMS	Files	Web History	Call Log	Camera
Installation-time apps	F	F	F	F	F	F	F	F	F

'F' means the tested app *failed* in detecting the leakage attempt.

Table 2: Testing Privacy Apps that Require Root Access

App Name	Vendor/Developer	Sensitive Information Source								
		Device IDs	Accelerometer data	Contacts	Loc. details	SMS	Files	Web History	Call Logs	Camera
SECUREit	Lenovo	F	F	B	B	B	F	B	B	F
X Privacy	Marcel Bokhorst	B+	F	B	B	B	F	B	B	B
LBE Privacy Guard	Lamian	B	F	B	B	B	F	F	B	B
360 Mobile Safe	Qihoo 360	B	F	B	B	B	F	F	B	B
PrivacyProtecter	Li et al.	B	F	F	F	F	F	F	F	B

'F' means the tested app *failed* in detecting the leakage attempt.
'B' indicates that the privacy system *blocked* access to the information source.
'B+' refers that the privacy system is able to go beyond blocking access via faking the leaked information.

conducted in two approaches depending on if user intervention is needed: fully automated without user intervention and semi-automated with user intervention.

Table 3: Probing Apps Requirement of User-Intervention

App No.	Set-up time in seconds	Interaction required	Type of interaction
1	16.1	Yes	Accept terms/ Connect to cloud
2	26.5	Yes	Accept policy/ Configurations
3	8.5	No	-
4	22.5	Yes	Slide some ads/ Configurations
5	8.8	Yes	Accept policy/ Configurations
6	13.0	Yes	Accept terms/ Upgrade offer
7	76.3	Yes	Accept terms/ Set a code/ Register
8	57.4	Yes	Click start/ Setting/ Slide ads
9	18.0	No	-

5.2 Benchmarking Results

Static analysis based apps: A majority of the mobile privacy protecting applications collected from different app markets belong to the first category. Apps in this group are only capable of inspecting privacy-related features of an app at installation time. They either apply a permission analysis, statically analyze installed apps or upload examined app to a sandbox to test it dynamically. Some tools notify the user of reported malicious apps if installed. Nevertheless, none of them reported the Benchmarker App of PPAndroid-Benchmarker as a malicious app. Therefore, all tested apps did not block or detect any of the leaking attempts. Table 1 clarifies that where 'F' indicates the failure of tested apps to detect privacy violation. The results are expected and demonstrated PPAndroid-Benchmarker worked as designed.

In this group, approximately %75 of the apps were tested semi-automatically with user intervention and the others tested fully automatically.

Privacy apps requiring Root access: The second category covers apps that block access to defined sensitive data sources on mobile devices. All apps in this group require the Root access in order to reject other apps' requests of accessing sensitive information sources. In our testing, five commercial apps were found to provide such a functionality: *SECUREit*, *X Privacy*, *Pdroid Privacy Protection*, *LBE privacy guard* and *360 Mobile Safe*. The last two apps are only available in Chinese. One research app developed by Li et al. [17] also falls into this category.

Table 2 shows the result of testing apps in this category. Those apps are different in terms of how much they protect the user. As illustrated, each app has its own fixed list of predefined private

Table 4: Testing TaintDroid against PPAndroid-Benchmarker

Anti-Taint Trick	Sensitive Information Source								
	Device IDs	Accelerometer	Contacts	Location	SMS	Files	Web History	Call Log	Camera
None	S	S	S	S	S	F	S	F	S
simple encoding	F	F	F	F	F	F	F	F	-
shell command	F	F	F	F	F	F	F	F	-
file+shell hybrid	F	F	F	F	F	F	F	F	-
time keeper	F	F	F	F	F	F	F	F	-
count-to-X	F	F	F	F	F	F	F	F	-
file length	F	F	F	F	F	F	F	F	-
clipboard	F	F	F	F	F	F	F	F	-
exception/error	F	F	F	F	F	F	F	F	-
remote control	F	F	F	F	F	F	F	F	-
file last modified	F	F	F	F	F	F	F	F	-
direct buffer	S	S	S	S	S	F	S	F	-
lookup table	S	S	S	S	S	F	S	F	-

'F' means the tested app *failed* in detecting the leakage attempt.
'S' means the tested app *succeeded* in detecting the leakage attempt.
'-' anti-taint tricks cannot be applied to camera photos, as they are designed to process strings only.

information sources. The table shows how each app responded to privacy leaks of variant sources. 'F' indicates that the app did not react against that leakage, where 'B' means the attempt was blocked. *X Privacy* clearly is more diverse than others in covering many sensitive sources. It is also capable of faking the mobile device identifier IMEI (represented as 'B+' in Table 2). This app is the only one in this group able of providing an option beyond blocking attempts. Hence, *X Privacy* gives three options when the IMEI is being accessed: block, fake, and allow. All the apps in this group were tested in the semi-automated with user interaction mode on a rooted device. The reason is that they need to be granted with root access to function. Additionally, most of them require the user to set up the app and interact with some interfaces before they are ready.

Real-time privacy monitoring apps: The last category covers only a few apps developed by researchers. For this category we tested TaintDroid since it is the basis of many other solutions. Table 4 illustrates the result of benchmarking TaintDroid with our system. It shows that TaintDroid succeeded in detecting privacy leakages in real time, as 'S' shows in the table, for its predefined list of sensitive information sources. Each time we tried to leak a piece of sensitive information, TaintDroid triggered a notification showing the tainted tracked data alongside with the leaking app and some other details. Some information sources, notably 'files', 'call log' and 'camera', are not included in TaintDroid's list, so the privacy leakage attempts were not detected. When the information leakage was attempted using anti-tainting tricks, most of leakage attempts were not detected, especially for the 'direct buffer' and 'lookup table' tricks.

TaintDroid in this work was tested in a customized emulator since it requires a modified version of Android operating system to run. We tested TaintDroid in a fully automatically manner since no user intervention is required in the benchmarking process.

6 DISCUSSIONS

In brief, our experiments proved that PPAndroid-Benchmarker is efficiently and correctly working as designed. The overall results from the benchmarking experiment can be summarized as follows: static analysis based apps tested all failed to detect any real-time leakage attempts, privacy apps that require Root access could block access to some sensitive information sources, and TaintDroid (representing dynamic analysis based methods) could detect most of the leakage attempts except for three information sources tested but it failed when most anti-tainting tricks were applied.

As a general observation, the more "intrusive" a privacy protection tool is, the more powerful it can do to detect and prevent privacy leakage attempts. By "intrusive" we mean how much changes a tool requires from the operating system, ranging from the most intrusive to the least: rebuilding the operating system, requesting Root access to the operating system, hooking into the operating system for checking new apps at the installation time. However, even for the most intrusive tool, TaintDroid, there are still privacy leakages that cannot be detected. This implies that the best approach is for the operating system itself to provide native support on privacy leakage detection and prevention, so that the intrusiveness will not be an issue any more. If that happens, PPAndroid-Benchmarker will still be able to benchmark the built-in privacy protection mechanism since it simple simulates what malicious

apps are doing. As a matter of fact, in our current implementation, a baseline benchmarking profile is always run without any third-party privacy protection apps so that we know what privacy leakage attempts can be detected by the Android system itself.

Some could argue, why PPAndroid-Benchmarker does not give a numeric or categorical rating for tested apps. Having a way of rating privacy protection mobile apps in terms of their general performance can be very useful for end users, and is a topic for our future research. A possible way of giving scores or ratings through our system is to count how many sensitive sources are protected by the tested tool, but some issues need to be carefully considered. First, not all sensitive information leakages are considered privacy violation [24]. Some sensitive information is transmitted out of the device for the sake of providing better services for the user. For instance, collecting users' precise location can help an app give more personalized and contextualized recommendations, and collecting some user details help mobile app vendors to provide extra functionalities and more personalized services. Another issue is that it can be very challenging to cover all means of collecting sensitive information sources from the mobile device, so any ratings based on a limited coverage can be inaccurate or biased. In addition, it is not trivial to give a single rating for so many different information sources and ways to leak information. It may be beneficial to produce multiple ratings representing different aspects of the privacy protection level. For each rating, there is also issues around how to aggregate results of all covered information sources and sinks together, which is not trivial, either. Therefore, rating mobile privacy protection apps could be a very complicated task, and requires further work beyond this paper's scope.

One of the key design goals of the presented work is to increase the level of automation to streamline the benchmarking processes. However, since Google Play's Terms of Services does not allow automated download of mobile apps, that feature cannot be fully achieved for our prototype. The system has to run in a way such that the tester is asked to provide the apk file of the tested app. Nevertheless, PPAndroid-Benchmarker currently mitigates this problem by building an **Apps Repository** so that known apps can be retrieved directly from the database without any user intervention. Note that this limitation is not a technical issue but a legal one. If PPAndroid-Benchmarker is adopted by Google, this issue will go away naturally.

As we mentioned before, due to the need to set up and configure initial settings, it is not possible to fully automate the benchmarking process without any human intervention. However, given enough information about what kinds of human interventions are needed, such interventions can be avoided by adding needed human interactions into the Apps Repository and then using some system services to emulate the actions done by human users. For instance, some system services for accessibility purposes have provided ways to automate clicks of buttons on the user interface, which may be incorporated in further versions of PPAndroid-Benchmarker.

To allow further development and validation of PPAndroid-Benchmarker by the wider research community, we decided to release our current implementation as an open-source tool at https://github.com/SaeedAlqahtani/PPAndroid-Benchmarker. As mentioned before, we plan to add some new components and APIs into the current implementation. We also welcome other researchers and developers to contribute to PPAndroid-Benchmarker, by conducting more tests, adding new add-ons to it and porting it to other mobile operating systems.

7 CONCLUSION & FUTURE WORK

In this paper, we present PPAndroid-Benchmarker, a benchmarking system for evaluating performance of mobile privacy protection apps. This system allows to benchmark privacy protection apps designed for the Android platform. To the best of our knowledge, PPAndroid-Benchmarker is the first of its kind. We tested PPAndroid-Benchmarker on 165 privacy protection apps belonging to three different functional categories to demonstrate PPAndroid-Benchmarker's effectiveness in evaluating their performance in detecting privacy leakage attempts. We believe that the experiment we conducted is also the first of the kind as many previous efforts are about benchmarking privacy risks of normal apps rather than privacy protection apps. The results showed that real-time dynamic monitoring tools like TaintDroid is the best approach, which is not a surprise since such tools require the most changes to the underlying operating system.

For future work, there are a number of improvements we can make on PPAndroid-Benchmarker. The most important components to add to our prototype are the Test Apps Generator and the Reconfigurability Engine discussed in Sections 3.3 and 3.4. Adding these two components is not technically difficult, although we will need to decide carefully how to make them more usable to end users of PPAndroid-Benchmarker. Another interesting feature to add is to incorporate the system with an **Off-line Analyzer**. This component can be designed to collect both testing profiles and benchmarking results with the aim of producing more visualized results to facilitate understanding of tested privacy protection apps and comparing their performance. It can also produce one or more ratings to reflect the level of privacy protection of each tested app. Furthermore, another future work is to add a benchmarking profile for dynamic behaviour apps. Some privacy protection apps have dynamic behaviors. There are some privacy protection apps connected to a cloud and update their data every while. Other apps may have an intelligent way of adapting their behaviors, e.g. it may use machine learning techniques and improve its responses to privacy leakage attempts of malicious apps. To properly benchmark such apps, PPAndroid-Benchmarker need to run the benchmarking task for a significantly long period of time and capture a number of snapshots of the tested app's behavior, and then see if some changes can be observed.

ACKNOWLEDGMENTS

We would like to thank authors of [22] and [17] for providing more information about their work. Thanks also go to Haiyue Yuan of the University of Surrey, for his ideas and help in testing some of the apps. Saeed Alqahtani's work is financially sponsored by a PhD scholarship of the Taibah University, Saudi Arabia.

REFERENCES

[1] AppBrain. 2017. Android Statistics: Number of Android applications. http://www.appbrain.com/stats/number-of-android-apps. (2017). Accessed: June 17, 2017.

[2] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. 49, 6 (2014), 259–269. Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14).

[3] AV-Comparatives. 2015. List of Mobile Security Vendors. http://www.av-comparatives.org/list-mobile/. (2015). Accessed: 2015-10-02.

[4] Alastair R Beresford, Andrew Rice, Nicholas Skehin, and Ripduman Sohan. 2011. MockDroid: trading privacy for application functionality on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications (HotMobile 2011)*. ACM, 49–54.

[5] Xin Chen and Sencun Zhu. 2015. DroidJust: Automated functionality-aware privacy leakage analysis for Android applications. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 5.

[6] Andrea Continella, Yanick Fratantonio, Martina Lindorfer, Alessandro Puccetti, Ali Zand, Christopher Kruegel, and Giovanni Vigna. 2017. Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis. (2017).

[7] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2014. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *ACM Transactions on Computer Systems* 32, 2 (2014), Article No. 5.

[8] William Enck, Machigar Ongtang, and Patrick McDaniel. 2009. On Lightweight Mobile Phone Application Certification. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS 2009)*. ACM, 235–245.

[9] Lejun Fan, Yuanzhuo Wang, Xiaolong Jin, Jingyuan Li, Xueqi Cheng, and Shuyuan Jin. 2013. Comprehensive Quantitative Analysis on Privacy Leak Behavior. *PloS ONE* 8, 9 (2013), e73410.

[10] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. 2015. Android Security: A Survey of Issues, Malware Penetration, and Defenses. *IEEE Communications Surveys & Tutorials* 17, 2 (2015), 998–1022.

[11] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. 2011. Android Permissions Demystified. In *Proceedings of 2011 18th ACM Conference on Computer and Communications Security (CCS 2011)*. ACM, 627–638.

[12] Christopher S. Gates, Ninghui Li, Hao Peng, Bhaskar Sarma, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. 2014. Generating Summary Risk Scores for Mobile Applications. *IEEE Transactions on Dependable and Secure Computing* 11, 3 (2014), 238–251.

[13] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. 2012. Riskranker: scalable and accurate zero-day android malware detection. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 281–294.

[14] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. 2011. "These Aren't the Droids You're Looking For": Retrofitting Android to Protect Data from Imperious Applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS 2011)*. ACM, 639–652.

[15] Jinyung Kim, Yongho Yoon, Kwangkeun Yi, and Junbum Shin. 2012. ScanDal: Static Analyzer for Detecting Privacy Leaks in Android Applications. *Proceedings of 2012 Workshop on Mobile Security Technologies (MoST 2012)* (2012).

[16] Anh Le, Janus Varmarken, Simon Langhoff, Anastasia Shuba, Minas Gjoka, and Athina Markopoulou. 2015. AntMonitor: A System for Monitoring from Mobile Devices. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data*. ACM, 15–20.

[17] Shancang Li, Junhua Chen, Theodoros Spyridopoulos, Panagiotis Andriotis, Robert Ludwiniak, and Gordon Russell. 2015. Real-Time Monitoring of Privacy Abuses and Intrusion Detection in Android System. In *Human Aspects of Information Security, Privacy, and Trust: Third International Conference, HAS 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015. Proceedings (Lecture Notes in Computer Science)*, Vol. 9190. Springer, 379–390.

[18] Jialiu Lin. 2013. *Understanding and capturing people's mobile app privacy preferences*. Ph.D. Dissertation. Carnegie Mellon University. http://cmuchimps.org/uploads/publication/paper/132/understanding_and_capturing_people_s_mobile_app_privacy_preferences.pdf

[19] V. Benjamin Livshits and Monica S. Lam. 2005. Finding Security Errors in Java Programs with Static Analysis. In *Proceedings of the 14th USENIX Security Symposium*. USENIX Association, 271–286.

[20] NowSecure. 2015. The Mobile App Security Company. https://www.nowsecure.com/. (2015). Accessed: 2015-10-20.

[21] Vaibhav Rastogi, Zhengyang Qu, Jedidiah McClurg, Yinzhi Cao, and Yan Chen. 2015. Uranine: Real-time Privacy Leakage Monitoring without System Modification for Android. In *Security and Privacy in Communication Networks: 11th EAI International Conference, SecureComm 2015, Dallas, TX, USA, October 26-29, 2015, Proceedings (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering)*, Vol. 164. Springer, 256–276.

[22] Golam Sarwar, Olivier Mehani, Roksana Boreli, and Mohamed Ali Kaafar. 2013. On the Effectiveness of Dynamic Taint Analysis for Protecting against Private Information Leaks on Android-based Devices. In *Proceedings of 2013 10th International Conference on Security and Cryptography (SECRYPT 2013)*. 461–468.

[23] Secure Software Engineering at Paderborn University and TU Darmstadt. 2017. DroidBench - Benchmarks. https://blogs.uni-paderborn.de/sse/tools/droidbench/. (2017). Accessed: June 18, 2017.

[24] Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X Sean Wang. 2013. AppIntent: Analyzing Sensitive Data Transmission in Android for Privacy Leakage Detection. In *Proceedings of 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS 2013)*. ACM, 1043–1054.

[25] Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, X Sean Wang, and Binyu Zang. 2013. Vetting Undesirable Behaviors in Android Apps with Permission Use Analysis. In *Proceedings of the 2013 Conference on ACM Computer & Communications Security (CCS 2013)*. ACM, 611–622.

[26] Cong Zheng, Shixiong Zhu, Shuaifu Dai, Guofei Gu, Xiaorui Gong, Xinhui Han, and Wei Zou. 2012. 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 (SPSM 2012)*. ACM, 93–104.

[27] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. 2012. Hey, you, get off of my market: detecting malicious apps in official and alternative android markets.. In *NDSS*, Vol. 25. 50–52.

[28] Yajin Zhou, Xinwen Zhang, Xuxian Jiang, and Vincent W Freeh. 2011. Taming Information-Stealing Smartphone Applications (on Android). In *Trust and Trustworthy Computing: 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22-24, 2011. Proceedings*. Lecture Notes in Computer Science, Vol. 6740. Springer, 93–107.