A Risk Assessment Mechanism for Android Apps

Ha Xuan Son
DiSTA, University of Insubria
Varese, Italy
sha@uninsubria.it

Barbara Carminati
DiSTA, University of Insubria
Varese, Italy
barbara.carminati@uninsubria.it

Elena Ferrari
DiSTA, University of Insubria
Varese, Italy
elena.ferrari@uninsubria.it

***Abstract*—Mobile apps have become an integral part of our daily lives in that they can be used for accessing a variety of services everywhere, being smart IoT one of the most important domain. However, despite the many benefits that the use of mobile apps provide, there are also risks related to the usage of personal information. Understanding the privacy implications of installing an app could be very difficult, especially for non skilled users. To cope with this issue, in this paper, we provide a risk estimation approach based on apps' static analysis. The output of the static analysis is then used to determine how much the personal data usage pattern of an app diverges from that of apps with the same purpose and this is in turn used to determine the app privacy risk. To prove that the proposed risk estimation measure is effective, we run several experiments with the involvement of different groups of participants, obtaining an accuracy varying from 79% to 82%.**

***Index Terms*—Mobile apps, Privacy risk assessment, static analysis.**

I. Introduction

Our daily activities heavily rely on apps running, and, as a result, we have an app that can support us in almost every task. Internet of Things (IoT) is one of the most fast growing domain for mobile apps development, since apps are heavily used for the remote control of smart gadgets, like wearables (e.g., wristwatches, eyeglasses), medical devices, sensors, infrared foot-traffic counters, to mention a few. The result is that users have on average from 20 to 60 apps installed on their mobile devices.[1]

Although this trend has enormous advantages in that users can benefit from different services almost everywhere, the downside is that apps gather a massive amount of information about individuals (e.g., users' profile and habits) and their devices (e.g., locations) that may leak personal information. However, not all this information is indeed strictly needed for the app execution. Privacy regulations, such as the European GDPR,[2] have been designed with the aim to limit the request of unnecessary information. At this purpose, GDPR introduces the data minimization principle, which requires to collect and retain only that personal data which is necessary for the app's purposes. Although these efforts have produced the effects of reducing the set of required permissions to some extent, still users are largely unaware of the privacy implications of some permissions they grant to apps.

To cope with the privacy risk of installing apps, individuals have to be able to determine which data an app is collecting. Ideally, they could read and analyze the app's privacy policy, where one expects to read which data the app collects, for which usage, and how such data are handled at the provider side. Individuals could also carefully check the list of permissions they grant when installing the app. However, these tasks might be too tricky for average users. For this reason, several research groups have recently started to investigate tools supporting users in taking more privacy-aware decisions on app usage. Some approaches propose to determine the risk only based on the app's requested permissions and the app's description (see Section VI for more details). However, these solutions are not robust against malicious apps able to circumvent the permission system and gain access to protected data by applying side channels [1].

To overcome this problem, this paper proposes to rely on the static analysis of the app's code to provide a more robust risk estimation. For this purpose, we reviewed the Google-supported APIs to determine the functions/constants used to collect personal information. We have identified 66 APIs and more than 13K functions/constants. Then, given a target app, we parsed its code searching for the usage of these functions/constants. This determines the app's behavior in terms of data collection. Risk estimation is then based on how the identified behavior is different from the "regular" one, that is, the functions/constants usage of the majority of apps. In particular, acknowledging that apps behave differently based on their business goals, we build a different regular behavior for each distinct app category.

Other approaches have been proposed to identify risky apps via static analysis, that we review in Section VI. With respect to these approaches, our proposal is different in two main aspects. The first is that we consider API usage at a fine-grain level, by targeting functions/constants to model the app behavior. The second is that existing approaches just label an app as risky or not. This binary estimation might not be so practical in supporting users in their decisions. In contrast, we estimate the risk as a measure of the deviation of the target app's behavior from the one built on the corresponding category. To prove that the proposed risk estimation measure is effective, we run several experiments with the involvement of different groups of participants (i.e., experts and crowd workers). The obtained accuracy varies from about 79% with the crowd workers group to 82% with the experts group.

[1]https://www.simform.com/the-state-of-mobile-app-usage/, https://www.statista.com/statistics/267309/number-of-apps-on-mobile-phones/
[2]https://eur-lex.europa.eu/eli/reg/2016/679/oj

The remainder of this paper is organized as follows. Section II provides the definitions of the behavior of an app and of an app's category. Section III presents our risk measure, whereas details on static analysis are provided in Section IV. Experimental results are presented in Section V. Related work are discussed in Section VI. Finally, Section VII concludes the paper.

II. Apps behavior in personal data usage

To build the behavior of an app, we focus on how it requests personal data and how the pattern of its requests (called *app signature* in what follows) is different from the "regular" one. To determine the "regular behavior", we analyze which personal data the majority of the apps with the same business goal (e.g., social network, shopping, sports, etc.) requires. We then estimate to what extent the target app signature diverges from the usual access pattern of the corresponding group of similar apps (denoted in what follows as *category signature*).

To determine an app signature, we analyze the app's source code to detect API functions or constants that exploit personal data. Instead of dynamic code analysis, we use a static analysis approach, since it is easier to be deployed on a large number of apps (cfr. Section IV). Static analysis is fine-grain in that we consider functions/constants usage for each API, rather than only classes and APIs usage. In this way, we are able to precisely determine which personal data an app collects.

A. App signature

To determine which personal data an app a collects, we rely on static analysis of its source code. In parsing the source code, we are interested only in those instructions used to collect personal data. As such, we have conducted a review of Google-supported APIs for determining the functions/constants used to collect personal information. In particular, we target seven types of personal data, namely: user location (e.g., city, country), public places (i.e., public places where users have been); media (e.g., users' image, video, audio); connection (e.g., wifi name, used to infer user location in case of public wifi, activity on Bluetooth, NFC); hardware (e.g., camera, USB devices); telephony (e.g., contacts info, phone number); user profile (e.g., birthday, gender, name); and health and fitness (e.g., heart rate, step counts, body fat). For each of these seven data types, we identified all APIs that have at least a function/constant collecting personal data of that type.

In total, we identified 66 APIs. The results of the analysis is reported in Table I, where, for each data type, we present only some of the corresponding collected data, and the number of analyzed APIs, classes and functions/constants.

Given an app a, we model its behavior with respect to the collection of data of a given type dt as a vector V_a^{dt}, containing an element for each distinct function/constant able to retrieve a data of dt type. V_a^{dt}'s elements have value 1, if the corresponding function/constant is present in a's source code; 0, otherwise. As such, the behavior of an app w.r.t personal data usage is represented as a set S_a of seven vectors, one for each data type dt, to which we refer to as *app signature*.

Example 1. *The first five rows of Table II show the signatures of five different apps. For the sake of simplicity, each row presents only a portion of the vector associated with the location data type, i.e.,* $V^{location}$. *In particular, we focus on five functions related to: address, city, country, postal code, and last location.*

B. Category signature

To estimate the risk of an app, we compare it against what is expected to be its typical behavior w.r.t personal data usage. Acknowledging that apps behave differently based on their business goals, we generate a different behavior for each distinct category available in app stores. In this paper, we exploited the categories available at Google Play store.

To build this behavior, for each of the considered category, we analyzed a set of apps belonging to the same category, say *Cat*. As it will be described in Section IV, this set, referred to as training dataset, is generated by selecting apps from the top 10% of the most-downloaded apps in a given category. We run the static analysis on the training dataset to generate a set of app signatures, one for each app in the training dataset. Then, we extract from this set some common patterns, representing the typical behavior for *Cat*, denoted in what follows as *category signature*. More precisely, given a category *Cat* the corresponding signature S_{Cat} consists of 7 vectors, one for each data type dt, denoted in what follows as V_{Cat}^{dt}. V_{Cat}^{dt} is generated by considering all $V_{\overline{a}}^{dt} \in S_{\overline{a}}$, for each app $\overline{a}$ of the training dataset. Elements of V_{Cat}^{dt} are set to 1, if at least 50% of the values of the corresponding elements in $V_{\overline{a}}^{dt}$ computed on all apps in the training dataset are set to 1, they are set to 0, otherwise.

For instance, the last row in Table II presents the signature of the Sport category, built assuming as training dataset only the 5 apps whose signatures are listed in the first five rows of the table.

III. Risk estimation

To estimate the risk of a target app a belonging to a category *Cat*, we compare its signature S_a, with *Cat*'s signature S_{Cat}. According to our approach, the lower is the deviation of S_a from S_{Cat}, the lower is the risk that a collects not-necessary personal data. As such, we need a distance function $\mathcal{DF}$ able to compare the two signatures S_a and S_{Cat}. More precisely, for each dt in the 7 considered data types, we first evaluate separately $\mathcal{DF}$ on a's vector $V_a^{dt} \in S_a$ and the corresponding vector $V_{Cat}^{dt} \in S_{Cat}$. The final risk estimation is then given by averaging the obtained 7 distance values.

In the following, we first present the distance function $\mathcal{DF}$ for a single data type.

A. Distance function for a single data type

Given a data type dt and a target app a, a naive way to measure the distance between V_a^{dt} and V_{Cat}^{dt} is to compute the Hamming distance among the two vectors. This would help to determine the mismatch between the signature of the target app a and the corresponding category signature. However, this

Table I
EXAMPLES OF COLLECTED DATA FOR EACH DATA TYPE, THE NUMBER OF ANALYZED APIS, CLASSES, AND FUNCTIONS/CONSTANTS

Data type	**Collected data**	**APIs**	**Classes**	**Functions/Constants**
Location	Longitude, latitude, address, public places	6	92	1189
Media	Image, media metadata, audio, video	24	588	5362
Connection	IP address, wifi, bluetooth, http, nfc	13	329	3215
Hardware	Device name, camera, mobile sensors	8	84	1146
Telephony	Send/receive SMS, contact, phone number	8	127	1637
User profile	Birthday, gender, name, organization	2	58	354
Health&fitness	Fitness activities, height, weight, body fat	5	82	632

Table II
EXAMPLES OF A PORTION OF THE SIGNATURES OF FIVE APPS IN THE SPORT CATEGORY AND RELATED CATEGORY SIGNATURE

	getAddress()	**getCity()**	**getCountry()**	**getPostalCode()**	**getLastLocation()**
App 1 signature	1	1	1	1	0
App 2 signature	1	0	1	1	0
App 3 signature	1	0	1	1	1
App 4 signature	1	0	1	1	0
App 5 signature	1	0	1	1	0
Sports category signature	1	0	1	1	0

is not enough for our scenario, since we have to consider that not all the deviations from the regular behavior have the same importance, as the following example clarifies.

Example 2. *Let us consider the signature of the first and third app in Table II (i.e., first and third row of the table). They both have only one element different from the corresponding category signature represented in the last row, that is: `getCity()` and `getLastLocation()`. As such, the two apps have the same Hamming distance. However, we can notice that, in the first app, the anomaly is due to a data item (i.e., city name) which is similar to another information considered normal for the sports category (i.e., postal code). In contrast, the third app collects a data item (i.e., last location) which is not similar to any of the other information considered normal for the sports category, according to the category signature. Therefore, the third app must be considered more risky than the first one.*

In order to understand the relevance of a deviation, we need to be able to catch the semantic similarity between different data items collected by apps. At this purpose, for each considered data type, we build a taxonomy exploiting the hierarchy of APIs related to that data type. In the taxonomy, the root node denotes the data type (e.g., location), whereas first-level nodes represent the APIs considered for that data type, whereas second-level nodes model the classes belonging to each API in the first-level. Finally, leaves indicate the functions/constants belonging to the class represented by the parent node, that is, the personal data items collected by that function/constant. Each element in V_a^{dt} and V_{Cat}^{dt} correspond to a leaf of the corresponding dt taxonomy.

Example 3. *Figure 1 shows a portion of the taxonomy built for the location data type, where the root node is called* location info*, first-level nodes represent the considered APIs (e.g., `location.places` API), second-level nodes model the classes belonging to those APIs (e.g., `Place`, `Location` classes), whereas leaves represent the functions/constants that allow the collection of location data (e.g., `getName()`).*

Given a data type dt and a target app a beloging to category Cat, the proposed distance function $\mathcal{DF}$ takes as input vectors V_a^{dt} and V_{Cat}^{dt} and compares each pair of corresponding elements, that is, $V_a^{dt}[i]$ and $V_{Cat}^{dt}[i]$. In particular, $\mathcal{DF}$ associates a risk value to all possible combinations of values of $V_a^{dt}[i]$ and $V_{Cat}^{dt}[i]$. Hereafter, we analyze the four possible cases by discussing the risk value we propose for each of them. We recall that $V_a^{dt}[i] = 1$ means that the target app collects a data item of type dt by exploiting the function/constant corresponding to the i-th position in the vector. Similarly, $V_{Cat}^{dt}[i] = 1$ implies that the majority of the apps in *Cat* in the training dataset collect the same data item.

$V_a^{dt}[i] = 1$ and $V_{Cat}^{dt}[i] = 1$. This implies that both the target app a and the majority of the apps in the same category belonging to the training dataset exploit the function/constant corresponding to the vector's i-th position. There is no deviation of the target app a from the category behavior, as such, the distance function $\mathcal{DF}$ associates a 0 risk value.

$V_a^{dt}[i] = 0$ and $V_{Cat}^{dt}[i] = 0$. Similarly to the previous case, there is no deviation of a's behavior w.r.t. the one of the apps in the same category. Therefore, the risk is set to 0.

$V_a^{dt}[i] = 0$ and $V_{Cat}^{dt}[i] = 1$. This represents the case where the majority of apps in the same category of a collect a data item that the app a does not request. Even if this is a deviation of a from the category behavior, we do not consider this as risky since, in terms of personal data collection, a does not collect a data item that similar apps collect. Therefore, function $\mathcal{DF}$ assigns a 0 risk value for this case.

$V_a^{dt}[i] = 1$ and $V_{Cat}^{dt}[i] = 0$. In this case, a requests a data item that is not normally required by similar apps. As such, a non-zero risk value has to be assigned for this case. However, as Example 2 highlights, not all deviations are equally relevant. To understand the importance of the mismatch, we first search among the data items that are usually required by the apps

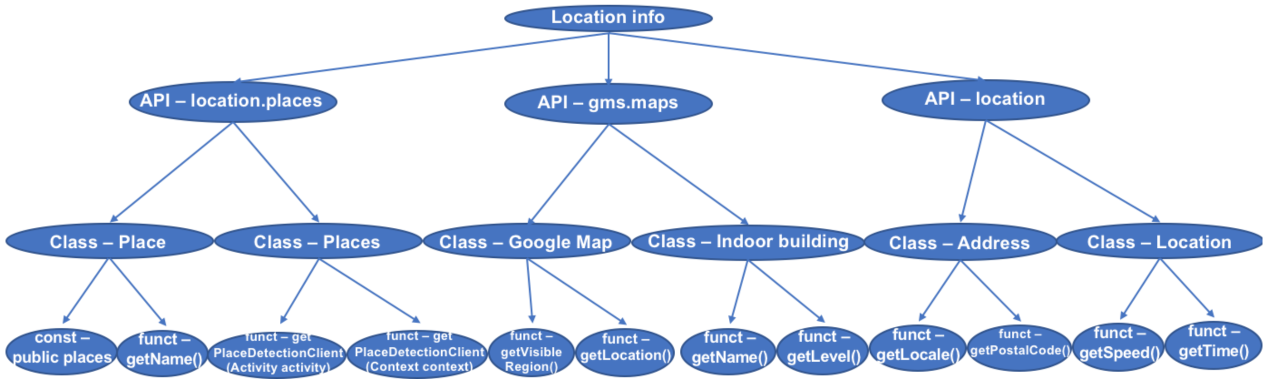

Figure 1. A portion of the location taxonomy

in the considered category (i.e., those elements in V_{Cat}^{dt} with values 1) the item that, based on dt taxonomy, is more similar to $V_a^{dt}[i]$. Hereafter, we refer to this item as the *closest collected data item of* $V_a^{dt}[i]$, denoted as $ccd(V_a^{dt}[i])$. The risk value is then estimated based on the similarity between $V_a^{dt}[i]$ and $ccd(V_a^{dt}[i])$. More similar are the two data items less risky is the collection of $V_a^{dt}[i]$ by a. This is the case, for instance, of data items `getCity()`, `getLastLocation()` (Exp. 2).

Therefore, in order to estimate the risk value of a target app, we need to introduce: a function $ccd()$ that, given a data item, returns its closest collected data item; and a similarity function $sim()$, to estimate the distance between the two on a taxonomy. The two functions are presented in what follows.

Given $V_a^{dt}[i]$, to find its closest collected data item we exploit the dt taxonomy, where $V_a^{dt}[i]$ represents a leaf node. The purpose of $ccd()$ is to traverse the taxonomy to find an element that in V_{Cat}^{dt} is set to 1 and is the closest element for $V_a^{dt}[i]$. In particular, starting from the leaf corresponding to $V_a^{dt}[i]$, $ccd()$ traverses the taxonomy up to $V_a^{dt}[i]$'s father node f, and searches among the leaves in the subtree rooted at f an item whose corresponding value in V_{Cat}^{dt} is set to 1.[3] If the search fails, $ccd()$ goes up to f's father to search among leaves in the subtree rooted at it. This is repeated till finding a data item in V_{Cat}^{dt} with value 1 or reaching the root. In this latter case, $ccd(V_a^{dt})$ is set to the root element.

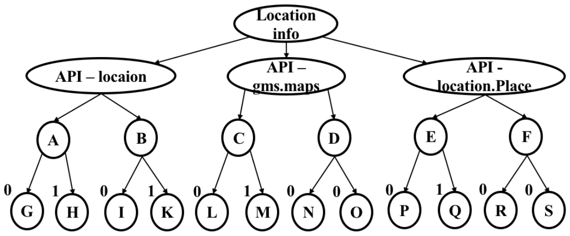

Figure 2. A sample of the location data type taxonomy

Example 4. *Figure 2 describes a portion of the taxonomy for the location data type. For the sake of simplicity, labels of second-level nodes and leaves have been replaced by a single letter. Let us first consider an app app_1 with signature* $\{1,0,0,0,0,0,0,0,1,0,0,1\}$. *Let us assume that the signature associated with app_1's category is* $\{0,1,0,1,0,1,0,0,0,1,0,0\}$. *For simplicity, in Figure 2, we reported the category signature values on the corresponding leaves of the taxonomy tree. Let us consider the 1-st element in app_1's signature (i.e., $V_{app_1}^{location}[1]$=1) corresponding to node G in the taxonomy. According to Figure 2, the corresponding element in the category signature is equal to 0. This implies to search G's closest collected data item. At first, we look among leaves in the subtree rooted at G's father node (i.e., A), that is, H. Since H's value is 1, ccd($V_{app_1}^{location}[1]$) is H.*

[3]If more than one leaf has value 1, $ccd()$ returns the first one. Indeed, according to the adopted similarity measure, leaves in the same subtree have the same distance w.r.t. a target node.

In contrast, $sim()$ computes the semantic similarity between two nodes (i.e., $V_a^{dt}[i]$ and $ccd(V_a^{dt}[i])$). At this purpose we exploit dt taxonomy, having $V_a^{dt}[i]$ and $ccd(V_a^{dt}[i])$ be two leaves of it (or a leaf and the root). Literature offers several methods to measure the similarity among terms of a taxonomy [2]. Among them, we select the Wu&Palmer similarity measure [3]. This measure takes into account the leaves' lowest common ancestor node, which, in our taxonomy is a class or API, as such the distance takes into account whether the two functions/constants belong or not to the same class/API. More precisely, given two leaf nodes n_1 and n_2 in a taxonomy, Wu&Palmer similarity is defined as follows:

$$WP(n_1,n_2) = \frac{2*depth(lca)}{dist(n_1)+dist(n_2)+2*depth(lca)} \quad (1)$$

where lca is the lowest common ancestor between n_1 and n_2, $depth$ (lca) is the length of the path from lca to the root of the tree, whereas $dist(n_1)$ (i.e., $dist(n_2)$) is the length of the path from n_1 to lca (i.e., n_2 to lca).

Example 5. *Considering again Example 4, the Wu&Palmer*

similarity between nodes G and H is:

$$WP(G,H) = \frac{2 \times depth(A)}{dist(G) + dist(H) + 2 \times depth(A)}$$
$$= \frac{2 \times 2}{1 + 1 + 2 \times 2} = \frac{4}{6} = 0.667$$

app_1 signature has also elements $V^{location}_{app_1}[8] = 1$ and $V^{location}_{app_1}[12] = 0$ having values different from those of the category signature (see elements P and S in Figure 2). For both of them, the closest collected item is Q. Therefore, $WP(P,Q)$= 0.667, and $WP(S,Q)$=0.333.

B. Risk estimation for a target app

To compute the risk of a target app, we first compute the risk for each single data type dt considered in the app signature. We then combine the computed risk values to obtain the app risk level.

Definition 1. Risk based on a single dt. *Let a be a target app and Cat be its category. Let dt be one of the considered data types, and let $V^{dt}_a \in S_a$ and $V^{dt}_{Cat} \in S_{Cat}$ be the vectors corresponding to dt in a's signature S_a and in Cat's signature S_{Cat}, respectively. The risk of app a w.r.t. data type dt is estimated as follows:*

$$\mathcal{DF}^{dt}(a) = \frac{\sum_{i \in V^{dt}_a} df(V^{dt}_a[i], V^{dt}_{Cat}[i])}{|V^{dt}_a|} \quad (2)$$

where $df(V^{dt}_a[i], V^{dt}_{Cat}[i])$ is computed as follows:

$$\begin{cases} 1 - WP(V^{dt}_a[i], ccd(V^{dt}_a[i])) & if V^{dt}_a[i] = 1, V^{dt}_{Cat}[i] = 0; \\ 0 & otherwise. \end{cases}$$

Example 6. *Let us consider again app_1, assuming for the location data type the taxonomy in Figure 2. The signatures of app_1 and of its category are reported in Table II. The risk of app_1 w.r.t Location is as follows:*

$$\mathcal{DF}^{Location}(app_1) = \frac{WP(G,H) + WP(P,Q) + WP(S,Q)}{12}$$
$$= \frac{0.667 + 0.667 + 0.333}{12} = 0.1389$$

Once the risk values for each of the considered seven data types have been computed, we compute the final risk value of an app.

Definition 2. Risk of an app. *Let a be a target app, and Cat be its category. The risk of a is defined as:*

$$Risk(a) = \frac{\sum_{\forall dt} \mathcal{DF}^{dt}(a)}{7} \quad (3)$$

IV. Implementation and datasets

Given an app a, we first retrieve the app's Java code by decompiling its Android `apk` files. We exploit the `apktool` tool to obtain the `dex` files, and then the `dex2jar` tool to obtain a conventional `jar` file.[4] This permits retrieving the app's `class` files. Finally, we obtain the Java code corresponding to each app by using the `jadx` tool.[5] Then, we parse the obtained `Java code` to detect the invoked/declared APIs, classes, functions, and constants, based on the keyword `import` and their activity scope specified in `"{" ... "}"`.

The amount of functions/constants to be analyzed is very high (i.e., 17,117). To have a more efficient analysis, we focused only on functions/constants related to the seven selected personal data types (cfr. Table I). This results in 13,535 functions/constants used in the static analysis to generate the apps' signature.

To run the experiments, we create two datasets: the training set ($TrainingSet$), used to build the category signatures, and the set of target apps ($TargetSet$), used to evaluate the proposed approach. To generate both of them, in November 2019, we downloaded some apps from the first 10% of the list of the most-downloaded apps for each of the considered categories,[6] in particular, those whose APK files were available in the app market.[7] This resulted in 21,784 apps,on which we run the static analysis. Among them, 18,098 were successfully decompiled (83.08%).[8] The whole reverse process required about 519.54 hours, i.e., approximately 85.86 seconds per app.

The resulting apps were then randomly divided into two disjoint datasets, namely training set (90%) and target set (10%). Table III shows, for each category, the number of apps in the training set and in the target set, respectively.

Table III
TRAINING AND TARGET SET

	Category	Training set	Target set
1	Food and drink	437	49
2	Health and fitness	2558	285
3	Map and navigation	622	70
4	Music and audio	563	63
5	Social networks	162	18
6	Beauty	547	61
7	Business	1039	116
8	Shopping	1071	119
9	Entertainment	1588	177
10	Finance	385	43
11	Medical	1521	170
12	Tools	1683	188
13	Sport	2002	203
14	Travel	989	110
15	Education	1434	160

V. Experimental evaluation

The purpose of our evaluation is to validate the proposed risk measure against risk users' perspective. Therefore, we acquire user feedback to check if users consider risky those apps collecting not appropriate data (out of scope wrt app category). We developed a web application to show to each participant a predefined set of apps. At first, participants are

[4]http://ibotpeaches.github.io/Apktool/ http://github.com/pxb1988/dex2jar

[5]https://github.com/skylot/jadx

[6]https://www.appbrain.com/stats/google-play-rankings/

[7]APK files have been downloaded at https://apkpure.com/

[8]Manual inspection revealed that failures are due to code obfuscation.

asked to read information about these apps (i.e., description, category name) that we consider useful to judge the risk level. Before asking to associate a risk level with a target app, we ask participants to rate whether they feel necessary or not that the target app collects a given data item. This is done for each single data item (e.g, address, image, audio, video) required by the app. Participants can express this with a label, ranging from `Very unnecessary` to `Very necessary`. This step ensures that each participant carefully considers which data items are indeed collected by the target app, and thus makes a more conscious judgment of the app's risk level. Moreover, the collected information is used to verify the quality of participant feedback, in that we manually inspected all feedback to check consistency between associated risk level and collected labels to remove possible random answers.

In the last step, participants are asked to select a risk level for the target apps (by selecting a value from a five-point Likert scale. `Very low - Very high`)[9] As a further quality check, we have asked each participant to write a short motivation for the selected risk level. The time for each survey is about 40 minutes.

A. Participants

We use two groups of participants to better understand the performance of our approach and how they are impacted by user characteristics. The first group has been selected to test our risk measure with expert users. At this purpose, we collected the feedback of 64 experts, all of them working in the Computer Science field. These experts work in both academic (45 users) and industrial institutions (19 users) in different countries (e.g., USA, Italy, Switzerland, Germany, Sweden, France, Cyrus, Greece, Vietnam, Morocco, and Singapore). Participants have an average age of 28.9 (from 24 to 37). Among the 45 participants working in academic institutions, 14 are lecturers/professors, whereas the others are Ph.D students and post-docs. For the participants working in industry, they are junior and senior developers.

The second group of participants has been selected to ensure the involvement of a good number of participants of different nationalities, ages, and educational levels. At this purpose, we exploit the Microworkers crowdsourcing platform[10] for the enrollment of participants (called workers). We select only the workers with the best rating in Microworkers platform.

We received 131 participants' feedback; however, we used only 101 of them, since 30 workers' contributions do not pass our quality checks (i.e., the worker used answer automation tools). The participants came from different countries (e.g., UK, Italy, India, Spain, Portugal, USA), with an average age of 29.4, being the oldest worker 52, and the youngest 18. They have different educational levels (e.g., student, bachelor) and background (e.g., accountant, teacher, manager). 60% of the participants were reported to have a bachelor degree or equivalent, whereas 6% of them had a master degree or a Ph.D. Workers were compensated $4.5 for each successful feedback.

[9]We used a five-point Likert scale, since it is a good trade-off between required users' effort and response quality [4].

[10]https://www.microworkers.com/

B. Metrics

Since we consider 5 risk levels (i.e., `Very Low, Low, Neutral, High` and `Very High`), we exploit a 5X5 confusion matrix to measure the effectiveness of our measure (see Table IV). The columns describe the risk levels estimated by our approach, the rows represent possible actual level given by the experiment participants. Finally, the cells denote error value (*E*) or true positive (*TP*). From the confusion matrix, we define the evaluation metrics given in Table V.

C. Experimental settings

We run a set of preliminary experiments to determine to what extent the information we provide to users to judge the risk of an app impacts the user final decision. We conduct three experiments, by varying the provided information, and we analyze its impact on the selected app risk level. In all the experiments, we use a fixed set of apps (10 apps), randomly selected from the *Targetset*.

In exp_0, for each target app, each participant is informed only on the requested personal data. In contrast, in exp_1, we provide to each participant the information provided in exp_0 as well as the category to which the target app belongs to. Finally, in exp_2, we provide all the information provided in exp_1, plus the description of the target app as well as its features.[11]

The number of participants is the same for all the settings, that is, 15 Microworkers and 15 experts. We start from exp_0 and then we go on with exp_1 and exp_2 to see whether having more information about the target app influences the user feedback. To limit the case that workers/experts can remember their decisions in previous experiments, we run the three experiments ten days apart.

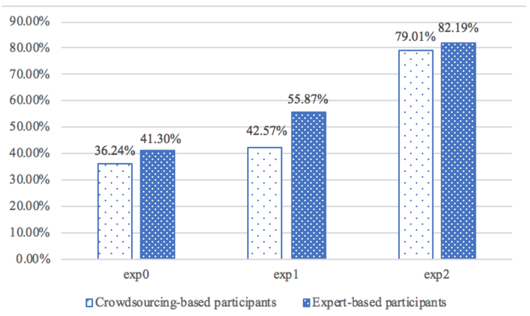

Figure 3. Crowdsourcing-based and expert-based accuracy for exp_0, exp_1 and exp_2

Figure 3 shows that the accuracy for the expert-based is higher than that of the crowdsourcing-based dataset for all the three settings. In addition, the greater increase in accuracy is obtained by changing from exp_1 to exp_2 setting. Moreover, also the feedback we received from participants show that they do not consider the information provided in the first

[11]App features denote the main service characteristics provided by the app.

Table IV
CONFUSION MATRIX

	Estimated value: VL	**Estimated value:** L	**Estimated value:** N	**Estimated value:** H	**Estimated value:** VH
Actual value: VL	TP_{VL}	$E_{V,L}$	$E_{VL,N}$	$E_{VL,H}$	$E_{VL,VH}$
Actual value: L	$E_{L,VL}$	TP_L	$E_{L,N}$	$E_{L,H}$	$E_{L,VH}$
Actual value: N	$E_{N,VL}$	$E_{N,L}$	TP_N	$E_{N,H}$	$E_{N,VH}$
Actual value: H	$E_{H,VL}$	$E_{H,L}$	$E_{H,L}$	TP_H	$E_{H,VH}$
Actual value: VH	$E_{VH,VL}$	$E_{VH,L}$	$E_{VH,N}$	$E_{VH,H}$	TP_{VH}

Table V
METRICS DEFINITION

Acc	(TP_{VL} + TP_L + TP_N + TP_H + TP_{VH}) / # samples
Pre_{VL}	TP_{VL}/(TP_{VL}+$E_{L,VL}$+$E_{N,VL}$+$E_{H,VL}$+$E_{VH,VL}$)
Pre_L	TP_L/(TP_L+$E_{VL,L}$+$E_{N,L}$+$E_{H,L}$+$E_{VH,L}$)
Pre_N	TP_N/(TP_N+$E_{VL,N}$+$E_{L,N}$+$E_{H,L}$+$E_{VH,N}$)
Pre_H	TP_H/(TP_H+$E_{VL,H}$+$E_{L,H}$+$E_{N,H}$+$E_{VH,H}$)
Pre_{VH}	TP_{VH}/(TP_{VH}+$E_{VL,VH}$+$E_{L,VH}$+$E_{N,VH}$+$E_{H,VH}$)
Re_{VL}	TP_{VL}/(TP_{VL}+$E_{VL,L}$+$E_{VL,N}$+$E_{VL,H}$+$E_{VL,VH}$)
Re_L	TP_L/(TP_L+$E_{L,VL}$+$E_{L,N}$+$E_{L,H}$+$E_{L,VH}$)
Re_N	TP_N/(TP_N+$E_{N,VL}$+$E_{N,L}$+$E_{N,H}$+$E_{N,VH}$)
Re_H	TP_H/(TP_H+$E_{H,VL}$+$E_{H,L}$+$E_{H,L}$+$E_{H,VH}$)
Re_{VH}	TP_{VH}/(TP_{VH}+$E_{VH,VL}$+$E_{VH,L}$+$E_{VH,N}$+$E_{VH,H}$)
$F1_C$	2*(Pre_C*Re_C)/(Pre_C+Rec_C), $C \in \{VL, L, N, H, VH\}$

two settings enough for evaluating the risk of a target app.[12] Therefore, we run our subsequent experiments by considering exp_2 setting.

D. Results

We compare the risk level computed by our approach with the risk level given by the participants. for both datasets.

Table VI reports the result for both datasets. Generally, the accuracy of experts (82.188%) is higher than that of the crowdsourcing dataset (79.009%). For both the datasets this experiment confirms that our risk measure is usually in line with the participants' feedback. We also measure the F1 score (cfr. Table V) for each class (i.e. `Very Low, Low, Neutral, High` and `Very High`) for both datasets. As we can see from Table VI, we have the lowest value for the F1 score for `Neutral` in both the datasets. The reason can be that users assign the `Neutral` rating when they lack the knowledge/background to assess the risk level of the target apps. In contrast, F1 low value for `Very low` level for the crowdsourcing-based dataset can be explained by the little awareness that not skilled users have on the privacy risks related to Android app.

VI. RELATED WORK

For instance, Sarma et al. [5] and Peng et al. [6] used probabilistic models to detect malicious apps, based on the requested permissions and their categories. Wu et al. [7] developed a framework that leverages on deep learning to identify correlations between the app's description and the requested permissions. These correlations assist users in determining whether an app description is accurate. In addition to permissions, Chia et al. [8] exploited app popularity, user ratings, and external community ratings to determine an app privacy risk. However, metadata do not accurately describe the actual behavior of an app w.r.t. the consumption of personal data. Indeed, many studies (e.g., [9]) have showed that apps can actually exploit more permissions than what they provide in their metadata. To take into account this issue, our approach exploits both metadata (e.g., app category) as well as input from the static analysis.

Other proposals focus on the requested permissions. For instance, Felt et al. [10] determined whether an Android app is over-privileged via static analysis. They classified an app as over-privileged if it requires permissions that are never actually used. Moutaz et al. [11], and Jianmao et al. [12] considered the set of 30 permissions provided by Google LLC [13], by labelling as critical those that have significant security/privacy impact. If any app requests these critical permissions, it is labelled as risky. Enck et al. [14] developed a system that detects whether an app makes use of risky combination of permissions. To do this, they manually defined a set of permission combinations, such as `WRITE_SMS` and `SEND_SMS`, to be considered risky. Then, they performed static analysis to label an app as malicious if it makes use of these combinations. All the above mentioned approaches determined the app's risk based only on its requested permissions. However, these solutions are not robust against malicious apps able to circumvent the permission system and gain access to protected data by applying side channels [1], such as using device sensors to uniquely identify users [15] or using the MAC address of the WiFi access point to infer user's location [16]. To overcome this limitation, our solution relies on static analysis to detect APIs/libraries usage.

Other proposals have also attempted to identify malicious apps and the leakage of private information by analysing the API usage. For example, the approaches proposed by [17], [18], [19], [20], [21] are based on the assumption that APIs/libraries are secure (i.e., "regular") when pervasively used by many apps (e.g., more than 60% in [19]). Exploiting this assumption, they cluster apps based on their APIs/libraries usage. They label an app as not risky, if it belongs to a cluster, or risky in case it is an outlier. Zhuo et al. [22] and Abhishek et al. [23] exploited static analysis to generate a graph representation of the data flow among API classes and possible data collection. These is done on a set of malicious apps

[12]For example: **Crow29**:*"In my opinion, this app is a privacy violation, so my choice is "High". Because it requires a lot of medical info. However, if it is a healthcare app, my answer is "Low"..."*points out that knowing the app category (which is provided by exp_2 and exp_1) is fundamental for assessing its risk level.

Table VI
Comparison of our metric with crowdsourcing-based and expert participants feedback

	Crowdsourcing-based dataset					Expert-based dataset				
	VL	L	N	H	VH	VL	L	N	H	VH
Precision	71.724%	78.947%	87.654%	82.589%	76.884%	79.412%	82.775%	77.778%	83.333%	83.019%
Recall	69.799%	84.071%	68.269%	79.399%	82.703%	80.597%	83.981%	63.636%	78.125%	93.617%
F1	70.748%	81.429%	76.757%	80.963%	79.688%	80.000%	83.373%	70.000%	80.645%	88.000%
Accuracy	79.009%					82.188%				

and a set of apps considered to be safe. The two graphs are then compared to determine the set of APIs to be considered dangerous. The above mentioned approaches provided a binary definition of risky/not risky app. This might not be so practical in supporting users in their decisions. In contrast, we estimate the risk as a measure of the deviation of the target app's behavior from the one built on the corresponding category. Moreover, differently from the above-mentioned papers, our proposal considers app behaviour w.r.t data collection at a fine-grain level, considering more than 13K functions/constants usage rather than simply APIs/classes. Additionally, our focus is not malicious apps detection, rather providing a user a risk estimation wrt the app's personal data consumption.

VII. Conclusions

This paper has addressed the challenging issue of estimating the privacy risk related to installing an Android app. We have proposed an approach based on the static analysis of the app's code, able to combine the need to design a scalable solution (since most of the computation can be done offline) with the ability to provide a fine-grain assessment of the app's personal data usage. We have tested our approach by using both expert feedback as well as feedback obtained via a crowdsourcing platform. The achieved experimental results are promising. In the future, we plan to perform a more extensive experimental evaluation and compare our method with others mentioned in Sect. VI. We also plan to extend our approach to be used to detect mismatches between an app privacy policy and its actual behavior w.r.t personal data usage.

Acknowledgments

This work has received funding from RAIS (Real-time analytics for the Internet of Sports), Marie Skłodowska-Curie Innovative Training Networks (ITN), under grant agreement No 813162 and from CONCORDIA, (Cybersecurity Competence Network) supported by H2020 Research and Innovation program under grant agreement No 830927. The content of this paper reflects only the authors' view and the Agency and the Commission are not responsible for any use that may be made of the information it contains.

References

[1] J. Reardon *et al.*, "50 ways to leak your data: An exploration of apps' circumvention of the android permissions system," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 603–620.
[2] D. Chandrasekaran and V. Mago, "Evolution of semantic similarity–a survey," *arXiv preprint arXiv:2004.13820*, 2020.
[3] Z. Wu and M. Palmer, "Verb semantics and lexical selection," *arXiv preprint cmp-lg/9406033*, 1994.
[4] S. B. Sachdev and H. V. Verma, "Relative importance of service quality dimensions: A multisectoral study." *Journal of services research*, vol. 4, no. 1, 2004.
[5] B. P. Sarma *et al.*, "Android permissions: a perspective combining risks and benefits," in *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, 2012, pp. 13–22.
[6] H. Peng *et al.*, "Using probabilistic generative models for ranking risks of android apps," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 241–252.
[7] Z. Wu, X. Chen, and S. U.-J. Lee, "Fcdp: Fidelity calculation for description-to-permissions in android apps," *IEEE Access*, 2020.
[8] P. H. Chia, Y. Yamamoto, and N. Asokan, "Is this app safe? a large scale study on application permissions and risk signals," in *Proceedings of the 21st international conference on World Wide Web*, 2012, pp. 311–320.
[9] O. Olukoya, L. Mackenzie, and I. Omoronyia, "Security-oriented view of app behaviour using textual descriptions and user-granted permission requests," *Computers & Security*, vol. 89, p. 101685, 2020.
[10] A. P. Felt *et al.*, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 627–638.
[11] M. Alazab *et al.*, "Intelligent mobile malware detection using permission requests and api calls," *Future Generation Computer Systems*, vol. 107, pp. 509–521, 2020.
[12] J. Xiao, S. Chen, Q. He, Z. Feng, and X. Xue, "An android application risk evaluation framework based on minimum permission set identification," *Journal of Systems and Software*, vol. 163, p. 110533, 2020.
[13] G. LLC, "Request app permissions," https://developer.android.com/training/permissions/requesting#perm-groups, accessed: 2021-02-10.
[14] W. Enck *et al.*, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, pp. 1–29, 2014.
[15] L. Simon *et al.*, "Don't interrupt me while i type: Inferring text entered through gesture typing on android keyboards," *Proceedings on Privacy Enhancing Technologies*, pp. 136–154, 2016.
[16] F. T. Commission *et al.*, "Mobile advertising network inmobi settles ftc charges it tracked hundreds of millions of consumers' locations without permission," *press release (June 22), https://tinyurl.com/h83c2be*, 2016.
[17] M. Backes *et al.*, "Reliable third-party library detection in android and its security applications," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 356–367.
[18] A. Kharaz *et al.*, "Unveil: A large-scale, automated approach to detecting ransomware," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 757–772.
[19] H. Wang *et al.*, "Wukong: A scalable and accurate two-phase approach to android app clone detection," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015, pp. 71–82.
[20] X. Zhang *et al.*, "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *Proceedings of the Conference on Computer and Communications Security*, 2020, pp. 757–770.
[21] S. Zhu *et al.*, "Measuring and modeling the label dynamics of online anti-malware engines," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2361–2378.
[22] A. K. Singh *et al.*, "Experimental analysis of android malware detection based on combinations of permissions and api-calls," *Journal of Computer Virology and Hacking Techniques*, pp. 209–218, 2019.
[23] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," *IEEE access*, vol. 7, pp. 21 235–21 245, 2019.