Data Science and Engineering
https://doi.org/10.1007/s41019-022-00189-1

RESEARCH PAPERS

A Risk Estimation Mechanism for Android Apps based on Hybrid Analysis

Ha Xuan Son[1] · **Barbara Carminati**[1] · **Elena Ferrari**[1]

Received: 8 April 2022 / Revised: 9 June 2022 / Accepted: 6 July 2022

Abstract
Mobile apps represent essential tools in our daily routines, supporting us in almost every task. However, this assistance might imply a high cost in terms of privacy. Indeed, mobile apps gather a massive amount of data about individuals (e.g., users' profiles and habits) and their devices (e.g., locations), where not all are strictly needed for app execution. According to privacy laws, apps' providers must inform end-users on adopted data usage practices (e.g., which data are collected and for which purpose). Unfortunately, understanding these practices is a complex task for average end-users. The result is that they install apps without understanding their privacy implications. To support users in making more privacy-aware decisions on app usage, we propose a risk estimation approach based on an analysis of the app's code. This analysis adopts a hybrid strategy, exploiting static and dynamic code analyses. Static analysis aims at discovering which personal data an app is collecting to determine whether the target app is asking more than required. This gives the first estimation of the app's risk level. In addition, we also perform a dynamic analysis of the target app's code. This further analysis helps determining whether the collected personal data is consumed locally on the mobile device or sent out to external services. If this happens, the risk level has to be increased, as personal data are more exposed. To prove the proposal's effectiveness, we run several experiments involving different groups of participants. The obtained accuracy results are promising and outperform those obtained with static analysis only.

Keywords Mobile apps · Privacy risk assessment · Hybrid analysis

1 Introduction

Mobile apps represent essential tools in our daily routines, supporting us in almost every task. The market provides apps offering services that vary from social media/entertainment to health monitoring, just to mention a few. The total number of installed Android apps in the market is approximately 258B in 2022, which equates to 20–60 apps per user's device on average.[1]

The downside of these apps' services is the high cost for privacy individuals might need to pay. Indeed, mobile apps gather a massive amount of information about individuals (e.g., users' profiles and habits) and their devices (e.g., locations). Some of these data are not strictly necessary for the app's functionality execution [1]. To limit the collection of unnecessary information, new privacy laws have been issued recently. As an example, EU GDPR[2] introduces the data minimization principle, which requires collecting and retaining only the personal data necessary for the app's purposes.

Although these efforts resulted in a reduced number of access permissions required by apps [2], individuals are still facing problems in fully understanding the privacy implications of permissions they grant to apps and thus the underlying privacy risks. To cope with this issue, several research

Ha Xuan Son, Barbara Carminati and Elena Ferrari were contributed equally to this work.

✉ Ha Xuan Son
sha@uninsubria.it

Barbara Carminati
barbara.carminati@uninsubria.it

Elena Ferrari
elena.ferrari@uninsubria.it

[1] DiSTA, University of Insubria, 21100 Varese, VA, Italy

[1] https://financesonline.com/number-of-mobile-app-downloads/, https://www.statista.com/statistics/267309/number-of-apps-on-mobile-phones/.

[2] The GDPR document is available at https://edps.europa.eu/data-protection/data-protection/glossary/d_en#data_minimization.

Published online: 29 July 2022

groups have recently started to investigate tools supporting users in taking more privacy-aware decisions on app usage. Some approaches estimate the privacy risk considering the app's requested permissions and the app's description (see Sect. 7 for more details). However, these solutions are not able to detect those malicious apps that gain access to sensitive data by exploiting side channels to bypass permissions [3].

A more promising approach relies on investigating app behavior by analyzing its source code. For instance, in [4] we leverage on static analysis of the app's code to determine the usage of functions/constants used to collect personal information. This determines the app behavior in terms of data collection. Then, the app risk level is defined based on the deviation of its behaviour w.r.t. the "regular" one, that is, the usage of these functions/constants done by the majority of apps with similar business goals.

In this paper, we extend the risk estimation proposed in [4] by also taking into account the app behaviour w.r.t. sharing of personal data to third-parties. More precisely, in addition to determine the collected personal data via *static analysis* (as [4] did), we also analyse the app at run-time, via source's code *dynamic analysis*, to determine whether the collected personal data is consumed locally on the mobile device or sent out to external services. If this happens, the risk level has to be increased, as personal data are more exposed. To take into account the app's behaviour w.r.t both data collection and sharing, in this paper, we propose a new *hybrid analysis*-based risk measure. Moreover, to prove that the hybrid analysis-based risk measure is more effective, we run several experiments with the involvement of different groups of participants (i.e., experts and crowd workers), by comparing static and hybrid analysis-based risk measures. The obtained results show that hybrid analysis received better accuracy (i.e., varying from 83.4 to 87%) than static analysis risk accuracy (i.e., varying from 78 to 80.7%).

The remainder of this paper is organized as follows. Sect. 2 provides the modeling of app's behavior w.r.t. data collection, which is used by the considered risk measures. Sections 3 and 4 introduce the static and hybrid analysis-based risk measures. Details on implemented hybrid-based approach are provided in Sect. 5. Section 6 presents experimental results, whereas related work are discussed in Section 7. Finally, Sect. 8 concludes the paper.

2 Modelling Apps behavior

Both static and hybrid approaches consider how much personal information a target app potentially collects during its execution. The key idea is to estimate the risk by comparing its behavior w.r.t. data collection (called *app signature*) with the common behaviors of applications with a similar business goal (called *category signature*).

2.1 App signature

To model the app's behavior, we first determine which personal information it collects. For this aim, we rely on the static analysis approach proposed in [4], where the app's source code is parsed to only consider those instructions used to collect personal data. As described in [4], these instructions have been selected by reviewing the Google-supported APIs and choosing only those related to seven types of personal data, namely: user location (e.g., city, country), 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 data (e.g., heart rate, step counts, body fat). In total, we identified 66 APIs, with 1360 classes and 13535 functions/constants.[3].

Given an app a, its *app signature* is computed by considering the collection of each of the above-mentioned data type separately. In particular, given a data type dt, the app behaviour w.r.t. dt is defined as a vector V_a^{dt} of n elements, where n is the number of functions/constants able to retrieve information of type dt. An element in V_a^{dt} is set to 1 if a's source code contains the corresponding function/constant; 0, otherwise. The final *app signature* is modelled as a set S_a of seven vectors, one for each data type dt (i.e., $V_a^{location}$; V_a^{places}, V_a^{media}, $V_a^{connection}$, $V_a^{hardware}$, $V_a^{telephony}$, $V_a^{profile}$, $V_a^{health\&fitness}$).

An example of a portion of app signature w.r.t. the media data type for app_1 is given in Fig. 1b, where it is reported that app_1 exploits the following functions/constants: *getDomain*(), *Authority*, *resume*(), *getMaxSpl*(), and *getMinSpl*().

2.2 Category Signature

To model the common behavior of applications with similar business goals, in [4] we compute the *app signature* for a selection of apps belonging to the same category and extract from them a common pattern.

More precisely, given a category *Cat* the corresponding *category signature*, denoted as S_{Cat}, consists of seven vectors, one for each of the considered data type dt, denoted as V_{Cat}^{dt}. Let A be the set of apps selected in *Cat* category, each V_{Cat}^{dt} is generated such that: $V_{Cat}^{\text{dt}}[j]$ is 1 if at least 50% of the corresponding elements in the signatures of apps in A is 1 (i.e., $\|\{V_{\overline{a}}^{\text{dt}}[j] = 1 \mid \overline{a} \in A\}\| > 50\%$); 0, otherwise.

[3] The API taxonomy tree is available at https://github.com/SonHaXuan/Android-App-Risk-Estimation/tree/master/data/StaticAnalysisParseTree.json.

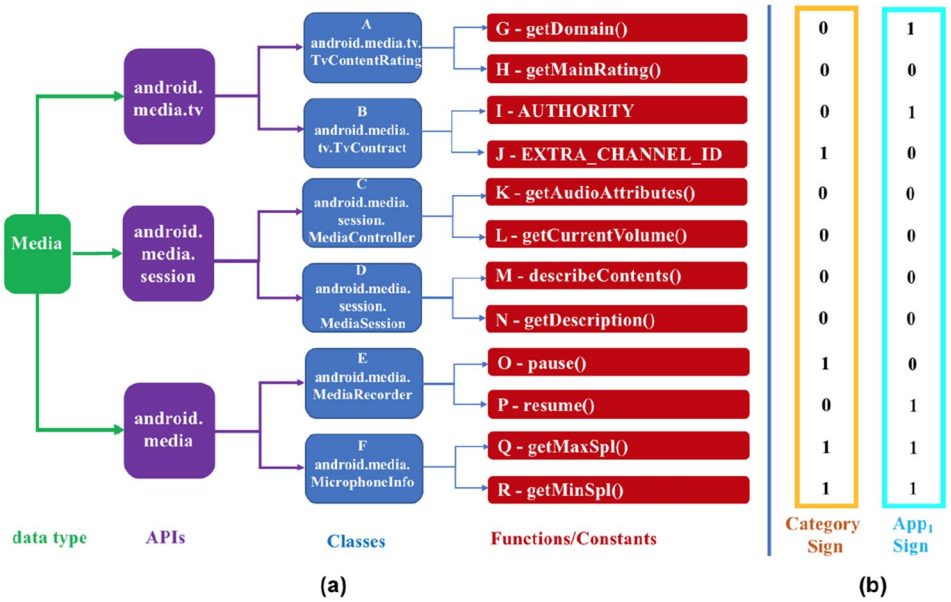

Fig. 1 **a** Portion of the media data type taxonomy; **b** an example of app and category signature

An example of a portion of signature for the entertainment category w.r.t. media data type is given in Figure 1(b).

3 Risk Estimation Based on Static Analysis

The key idea of the approach in [4] is to assess the risk of an app a of category Cat, by comparing its behaviour (i.e., app signature S_a) with the behaviour extracted from the corresponding category (i.e., Cat's signature, S_{Cat}). The rational is that the lower is the deviation of S_a from S_{Cat}, the lower is the risk that a collects unnecessary personal data. Thus, to estimate the risk a function $\mathcal{DF}$ is needed to measure the distance between S_a and S_{Cat}, aka two vectors of 0/1 elements. For this purpose, in [4] we did not exploit a vector distance, like the Hamming distance. Indeed, this type of distance only considers distributions of 0/1, without taking into account the semantics of the corresponding API functions. For example, apps with two very Hamming-similar signatures (e.g., with only 1 bit of difference) may differ in using only one function/constant that collects very different data. In contrast, to consider the API's semantics, [4] builds a distinct taxonomy for each data type, by exploiting the hierarchy of APIs used to collected the corresponding data. Figure 1a presents a portion of the taxonomy for the media data type (*media*): the root node indicates the data type, first-level nodes represent the considered APIs, whereas second-level nodes model their classes. Finally, leaves indicate the functions/constants, that is, the personal data item collected by that function/constant. According to this representation, each element in V_a^{dt} and V_{Cat}^{dt} corresponds to a leaf of the corresponding *dt* taxonomy. Thanks to this taxonomy, the distance between an element in V_a^{dt} and one in V_{Cat}^{dt} is computed by exploiting a semantic similarity measure [5]. In particular, given two elements $V_a^{\text{dt}}[i]$ and $V_{Cat}^{\text{dt}}[i]$, the similarity measure is exploited only when $V_a^{\text{dt}}[i] = 0$ and $V_{Cat}^{\text{dt}}[i] = 1$. Indeed, this is the only deviation meaningful for risk estimation, since $V_a^{\text{dt}}[i] = 1$ implies that a uses a function/constant (i.e., the one in the i-th position) to collect a piece of data that the majority of apps in the same category Cat is not collecting (i.e., $V_{Cat}^{\text{dt}}[i] = 0$).[4] In this case, to understand the importance of the deviation, the collected data (i.e., those corresponding to $V_a^{\text{dt}}[i] = 1$) are compared with the most similar data item collected by the majority of apps in the same category. First, the data items usually required by the apps in the same category are determined (i.e., those elements in V_{Cat}^{dt} with values 1). From these, the item that, based on *dt* taxonomy, is more similar to the one required only by a, i.e., $V_a^{\text{dt}}[i]$, is selected. Hereafter, we refer to this item as the *closest collected data item of* $V_a^{\text{dt}}[i]$, denoted as $ccd(V_a^{\text{dt}}[i])$.

Example 1 Let us consider Fig. 1a, representing a portion of the media data type taxonomy, and the corresponding portion of app_1's signature and its category signature given in Fig. 1b. For simplicity, to each second-level nodes (i.e., classes) and leaves (i.e., functions/constants) we associate a distinct letter used as index (e.g., "A" is associated with android.media.tv.TVContentRanting).

Let us consider the 1-st element in app_1's signature (i.e., G). Since $V_{app_1}^{media}[G]=1$ and $S_{ent}^{media}[G]=0$, we need to find G'

[4] Note that also $V_a^{\text{dt}}[i] = 0$ and $V_{Cat}^{\text{dt}}[i] = 1$ represents a deviation. However, it is not considered relevant in terms of risk estimation since, in this case, a does not collect a data item that the majority does.

closest collected data item, ($ccd(V^{med}_{app_1}[G])$. According to Fig. 1a, this requires to search G's closest collected data among leaves in the subtree rooted at G's father node (i.e., A), that is, H. The value of H is 0; therefore, we examine other leaves by recursively traversing A's parent (`android.media.tv`). Here, we found B node with two leaves I and J. Since J's value is 1, $ccd(V^{med}_{app_1}[G])$ is J.

Once the closest collected data item of $V^{\text{dt}}_a[i]$ has been determined, the risk level associated with the considered deviation is given as the dissimilarity between $V^{\text{dt}}_a[i]$ and $ccd(V^{\text{dt}}_a[i])$, in that more similar are the two data items less risky is the usage of $V^{\text{dt}}_a[i]$ by a. Thus, the risk of an app a, w.r.t a given data type dt, is computed as the sum of these dissimilarities, only in case $V^{\text{dt}}_a[i] = 0$ and $V^{\text{dt}}_{Cat}[i] = 1$, as the following definition states.

Definition 1 Risk of an app w.r.t a data type [4]. 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 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:

$$\mathcal{DF}^{\text{dt}}(a) = \frac{\sum_{i \in V^{\text{dt}}_a} df(V^{\text{dt}}_a[i], V^{dt}_{Cat}[i])}{\|V^{\text{dt}}_a\|} \tag{1}$$

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

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

where

- ccd is the function returning the *closest collected data item of* $V^{\text{dt}}_a[i]$;[5]
- $WP()$ is the Wu & Palmer similarity measure [6] used to compute the semantic similarity between $V^{\text{dt}}_a[i]$ and its closet collected data item.[6]

[5] Starting from the leaf corresponding to $V^{\text{dt}}_a[i]$ in dt's taxonomy, $ccd()$ traverses the taxonomy up to $V^{\text{dt}}_a[i]$'s father node f, and searches among the leaves in the subtree rooted at f an item whose corresponding value in V^{dt}_{Cat} is 1. This process is repeated till finding a data item in V^{dt}_{Cat} with value 1 or reaching the root. In this latter case, $ccd(V^{\text{dt}}_a)$ is set to the root element.

[6] Given two nodes n_1 and n_2 of a taxonomy, Wu & Palmer similarity measure is defined as:

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

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, and $dist(n_1)$ (i.e., $dist(n_2)$) is the length of the path from n_1 to lca (i.e., n_2 to lca).

In the following, we provide an example of risk computation w.r.t. a data type.

Example 2 Let us consider again app_1 signature and the signature of the entertainment category given in Fig. 1a. According to Example 1, the closest collected data item of the first app_1 signature's element set to 1 (i.e., $V^{media}_{app_1}[G]=1$) is node J. We therefore compute the Wu & Palmer similarity value between nodes G and J (with $lca = \}\}android.media.tv''$) as follows:

$$WP(G, J) = \frac{2 \times depth(lca)}{dist(G) + dist(J) + 2 \times depth(lca)} = \frac{2 \times 1}{2 + 2 + 2 \times 1} = 0.333$$

app_1's signature has also four other elements with value 1, namely I, P, Q, and R. Two of them (i.e., Q and R) have the same value of the corresponding element in the entertainment category signature, so their similarity value is 0. For the other two, that is, I and P, the closest collected items are J and O, respectively. Therefore, $WP(I, J) = WP(P, O)=0.667$.

The risk of app_1 w.r.t *media* data type is then computed as follows:

$$\mathcal{DF}^{media}(app_1) = \frac{(1 - 0.333) + (1 - 0.667) + (1 - 0.667) + 0 + 0}{12} = 0.111$$

The risk of an app is then defined by combining the risk values w.r.t each data type dt, as the following definition explains.

Definition 2 Static risk of an app [4]. Let a be a target app and Cat be its category. The static risk of a is defined as:

$$Risk_{static}(a) = \frac{\sum_{\forall \text{dt}} \mathcal{DF}^{\text{dt}}(a)}{7} \tag{3}$$

Example 3 Let us consider again app_1's signature presented in Fig. 1a. Let's assume that app_1 collects only the media data type (*media*). The risk of *app*1 wrt the *media* data type computed in Example 2 is 0.111. The static risk of app_1 is therefore as follows:

$$Risk_{static}(app_1) = \frac{\mathcal{DF}^{med}(app_1)}{7} = \frac{0.111}{7} = 0.016$$

4 Risk Estimation Based on Hybrid Analysis

As described in the previous section, the risk estimation proposed in [4] takes into consideration only the app behavior w.r.t. data collection. With the hybrid approach, we acknowledge that in addition to data collection is also relevant to consider the data usage, particularly whether the collected data are passed to third parties during the app execution. At this aim, we exploit dynamic analysis to determine whether the collected data are used only locally or embedded into communication packets sent to external service (see Sect. 5 for more details). More precisely, given an app, for each function/constant detected by the static analysis, we determine if the corresponding collected data are shared with third parties, transferred to the app server, or elaborated only locally. Then, the percentage of functions/constants that transfer data outside is used as a weight to increase the risk obtained through the static analysis. This is done at the level of data type as the following definition explains.

Definition 3 Hybrid risk of an app w.r.t a data type . Let a be a target app and dt be one of the considered data types. Let $Count(V_a^{\text{dt}} = 1)$ be the number of elements in V_a^{dt} with value equal to 1. Let $Count_{outside}$ be the number of functions/constants in V_a^{dt} that transfer data outside. The hybrid risk of a w.r.t. data type dt is estimated as follows:

$$\mathcal{DF}^{\text{dt}}_{hybrid}(a) = \mathcal{DF}^{\text{dt}}(a) + \frac{Count_{outside}}{Count(V_a^{\text{dt}} = 1)} \tag{4}$$

Example 4 Let us consider again app_1 whose signature has been given in Fig. 1a, and its risk w.r.t the *media* data type ($\mathcal{DF}^{media}(app_1)$) computed in Example 2. Let us assume that 4 of the 5 functions used by app_1 transfer data outside. The risk of app_1 w.r.t the *media* data type is as follows:

$$\begin{aligned}\mathcal{DF}^{media}_{hybrid}(app_1) &= \mathcal{DF}^{media}(app_1) + \frac{4}{5}\\ &= 0.111 + 0.8 = 0.911\end{aligned}$$

Once the hybrid risk value for each data type has been computed, we can compute the hybrid risk of an app, as follows.

Definition 4 Hybrid risk of an app. Let a be a target app. The hybrid risk of a is defined as:

$$Risk_{hybrid}(a) = \frac{\sum_{\forall dt} \mathcal{DF}^{dt}_{hybrid}(a)}{7} \tag{5}$$

Example 5 Let us consider again app_1 whose signature is given in Fig. 1b, and let us assume that app_1 shares only the media data type. The hybrid risk of app_1 is as follows:

$$Risk_{hybrid}(a) = \frac{\mathcal{DF}^{media}_{hybrid}(app_1)}{7} = \frac{0.911}{7} = 0.130$$

5 Implementation and Datasets

This section provides more details on the implemented hybrid analysis. Moreover, we explain how we generate the datasets used to validate the risk measures.

5.1 Hybrid Analysis

Given a target app a, we first perform the static analysis to compute the app signature. For this aim, we retrieve the app Java code by decompiling its Android `apk` files to retrieve the app's `class` files. Then, we parse the obtained code to detect the invoked/declared APIs, classes, functions, and constants, based on the keyword `import` and their activity scopes specified in "{" ... "}". This is done considering only functions/constants related to the seven selected personal data types as described in Sect. 3.[7] We then run the dynamic analysis of the target app a, to determine the data sharing behavior of each function/constant used by the app. More precisely, based on Definition 3, we determine for each data type dt, the number of functions/constants in V_a^{dt} that transfer data outside (i.e., $Count_{outside}$) and the number of functions/constants that appear in app's code (i.e., $Count(V_a^{\text{dt}} = 1)$).

To compute $Count_{outside}$, we exploit the MobiPurpose approach proposed in [7]. Using the approach in [7], we obtain the runtime network traffic between an app a and its server/third parties. The outcome of the network traffic analysis is modeled as `Key-Value` $(K - V)$ pairs, where K is the type of transferred data (a.k.a data item) and V is its value (see [7] for more details).

To determine $Count_{outside}$, we need then to associate each transferred data (i.e., $(K - V)$ pair) with a data type. For this purpose, we analyze each function/constant of a data type and, based on its returned value, we associate a possible data item. Figure 2 shows the result of this process, that is, (a portion of) data items associated with the seven considered personal data types. We then link the data item in the $K - V$ pair to the corresponding functions/constants.

5.2 Datasets

To run the experiments, we build two datasets: the *TrainingSet* used to generate category signatures, and the *TargetSet*

[7] The static analysis code is available at https://github.com/SonHaXuan/IEEE-SmartIoT.

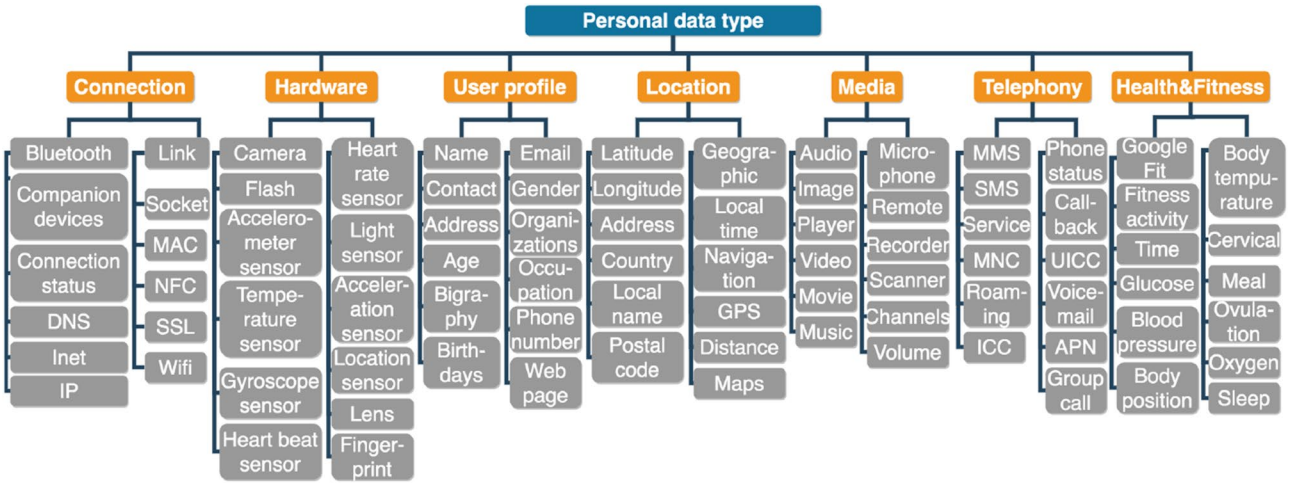

Fig. 2 A portion of the seven personal data types

containing apps to be evaluated with the proposed risk measures.

We exploit the dataset used in [4], which was obtained by downloading in November 2019, some apps from the first 10% of the most-downloaded apps list.[8] In particular, those apps whose APK files were available in the app market.[9] This resulted in 21,784 apps, out of which 18,098 were successfully decompiled (83.08%) for running the static analysis approach presented in [4]. The resulting apps were then divided into two disjoint datasets: the *TrainingSet* with 16,266 apps (90%) and *TargetSet* with 1,832 target apps (10%). W refer the interested reader to [4] for more details on this dataset.

More recently, in November 2021, before running the dynamic analysis, we updated the *TargetSet* by removing those apps that became unavailable in Play Store since November 2019. We removed 453 apps, resulting thus a *TargetSet* with 1,379 apps.

To run the dynamic analysis on apps in the *TargetSet*, we used 3 smartphones with different version of Android OS (i.e., 7, 10, and 12). We run each app from a minimum of 3–5 mins as maximum, depending on the device's hardware and the OS version. Not all apps in the *TargetSet* have generated traffic. This is due to three main reasons: (i) the app works offline; (ii) our dynamic analysis tool did not generate any input that activates requests to the servers; (iii) the app requires login (e.g., Whatsapp) preventing the tool from using the service. During this test, we captured 145,656 packets over seven days. Among those, we removed the traffic with empty-body requests, resulting thus in a final dataset of 92,561 packets. All the collected data are presented in our proof-of-concept prototype available at: https://github.com/SonHaXuan/Android-App-Risk-Estimation.

6 Experiments

The purpose of the experiments is to validate our hybrid risk measure and compare it with the static one proposed in [4], by comparing the risk estimations with those provided by users. To collect user risk evaluation, we developed a web application through which participants to the experiments provide their feedback and risk estimation based on a three-step survey. In the first step, participants read some information related to the target app, namely: the app's description, features, and category. In the second step, participants are asked to rate whether they feel necessary that the target app collects and/or shares a given data item. This is done for each data item (e.g., address, image, audio, video) collected and/or transferred by the app. Participants can express their opinion based on a five-level scale, ranging from `Very unnecessary` to `Very necessary`. This step ensures that each participant carefully considers which data items are indeed collected/shared by the target app and thus makes a 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 levels and collected labels to remove possible random answers.

The last step requires participants to select a risk level for the target app. The level is selected as a grade from a five-point Likert scale, `Very low –Very high`.[10] As a

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

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

[10] We used a five-point Likert scale since it is a good settlement between required users' effort and response quality [8].

Table 1 Precision (Pr), Recall (Re), F1, and Accuracy (Ac) scores for the crowdsourcing-based dataset

	Static risk measure					Hybrid risk measure				
	VL	L	N	H	VH	VL	L	N	H	VH
Pr	71.0%	80.1%	87.7%	76.3%	77.1%	79.5%	84.0%	95.0%	82.5%	88.3%
Re	73.1%	84.8%	67.6%	74.6%	79.4%	86.1%	77.8%	76.0%	75.0%	94.2%
F1	72.0%	82.4%	76.3%	75.4%	78.3%	82.7%	80.8%	84.4%	78.6%	91.1%
Ac	78.04%					83.46%				

Table 2 Precision (Pr), Recall (Re), F1, and Accuracy (Ac) scores for the expert-based dataset

	Static risk measure					Hybrid risk measure				
	VL	L	N	H	VH	VL	L	N	H	VH
Pr	80.3%	82.3%	76.3%	78.5%	83.0%	84.2%	86.1%	91.1%	84.6%	95.1%
Re	76.8%	83.2%	70.3%	76.3%	89.0%	91.4%	93.9%	78.5%	89.2%	95.6%
F1	78.5%	82.8%	73.2%	77.4%	85.9%	87.7%	89.9%	84.3%	86.8%	95.4%
Ac	80.78%					87.03%				

further quality check, we requested each participant to write a short motivation for the selected risk level. The duration of each survey is about 30 minutes.

The following sections describe the participants involved in the experiments and the obtained results.

6.1 Participants

We utilized two groups of participants to better evaluate the performance of our measures and how these are influenced by user characteristics. The first group consists of expert users. These are 59 experts working in the Computer Science field, in both academic (40 users) and industrial institutions (19 users) in different countries (e.g., USA, Italy, Switzerland, Germany, Sweden, France, Cyrus, Greece, Vietnam, Morocco, and Singapore). Among 40 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. Participants have an average age of 29.5 (from 25 to 34).

The second group of participants was 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[11] for the enrollment of participants (called workers), by selecting only those with the best rating in the Microworkers platform. We received 131 participants' feedback; however, we used only 92 of them, because 30 workers do not pass our quality checks (e.g., the worker used answer automation tools); whereas, 9 workers do not join our second experiment, that is, the one to evaluate our risk estimation based on hybrid analysis. The participants are from different countries (e.g., UK, Italy, India, Spain, Portugal, USA), with an average age of 28.6 (from 18 to 52). They have different educational levels (e.g., student, bachelor) and profile (e.g., accountant, teacher, manager). 58% of the participants have a bachelor degree or equivalent, whereas 6% of them have a master degree or a Ph.D. Workers were paid $6 USD for each successful feedback.

6.2 Evaluation of Crowdsourcing-Based Participants

We first compare the static and hybrid risk levels of the crowdsourcing-based dataset (cfr. Definition 4) with the risk level provided by the participants. For each each risk level (i.e. `Very Low (VL)`, `Low (L)`, `Neutral (N)`, `High (H)` and `Very High` (VH)), we compute the F1 score, accuracy (Ac), precision (Pr) and recall (Re) (see [4] for more details on the adopted metrics). Table 1 presents the results for both experiments. Generally, the accuracy of risk levels generated by the hybrid analysis (83.46%) is higher than that of the static analysis (78.04%).

6.3 Evaluation of Expert-Based Participants

Table 2 presents the results obtained by comparing our the static and hybrid risk measures with the risk levels provided by the expert participants. For both groups, the experiments reported in Table 2 confirms that the hybrid risk measure is in line with the participants' feedback, with an accuracy higher than that of the crowdworkers, i.e., static analysis (80.78 vs.78.04%) and hybrid analysis (87.03 vs. 83.46%), respectively.

Comparing the experiments' results for both datasets (i.e., Tables 1 and 2), we observe that the accuracy is improved

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

by the hybrid risk measure for both the participant groups. We explain this improvement by the fact that the hybrid risk measure provides participants with more information to assess the app's risk level. Indeed, we noticed crowdworkers assign the `Neutral` rating when they lack the knowledge/background to assess the risk level of target apps. This has been confirmed by also analysing the participant's comments. As an example, let us consider comments provided by crowdworker **Crow48** for a given app. In the experiment on the static risk measure he/she commented: "It is not enough information to conclude whether the app is risky or not. Neutral is my choice for this app" => `Neutral`. Whereas, in the experiment on hybrid risk measure, he/she commented: "I see this app not very reliable, I would not share my data"=> `High`. A further example is from an expert participant, i.e. **Expert12**, that in the experiment on static risk measure commented: "...I don't understand what this app is expected to do..."=> `Neutral`; whereas, in the experiment on hybrid risk measure he commented, for the same app: "This app does not appear, based on its description, to require advanced access to the phone to function properly. The fact that it requires so many permissions is therefore troubling. Several access categories do not seem directly related to what the app does. Specifically, telephony and media access should not be necessary based on the app description..." => `High`.

The F1 score of the last two classes (`High, Very high`) is also increased in the hybrid risk setting, especially the `Very high` class. The explanation for this improvement is that the participants become aware of the app sharing behaviour, and this increases the assigned risk level. As an example, the expert participant **Expert2** commented: "They transfer data regarding WiFi so they can infer the location. For their use case, they only need internet connection". This aligns well with our hybrid analysis approach, i.e., the more data are shared, the higher the risk (cfr. Def. 1).

7 Related Work

In the following, we review existing proposals for risk analysis of mobile apps. We organize the surveyed approaches based on the adopted techniques.

7.1 Risk Estimation Based on App Metadata Analysis

Several studies detect malicious behavior by analyzing app's metadata available on the Android market (e.g., app description, requested permissions, rating). For instance, Sarma et al. [9] and Peng et al. [10] used probabilistic models to detect malicious apps by comparing the app's functionalities with its required permission. Similarly, Wu et al. [11] developed a framework exploiting deep learning to detect correlations between the app's description and the requested permissions. These correlations assist users in determining whether an app description is accurate. Besides permissions, Chia et al. [12] employed app popularity, user evaluations, and external community ratings to define an app privacy risk. Unfortunately, metadata does not always precisely describe the app actual behaviour w.r.t. personal data collection. Indeed, many studies (e.g., [13]) shown that apps can utilise more permissions than what they state in their metadata. To cope with this limitation, static analysis has been used to analysis apps' behaviour.

7.2 Risk Estimation Based on Static Analysis

Literature offers several proposals exploiting static analysis to examine app's permissions. As an example, Felt et al. [14] determine whether an Android app is over-privileged by analysing the `Android Manifest.xml` file, searching for those permissions that are rarely used. Moreover, Moutaz et al. [15] and Jianmao et al. [16] considered a set of 30 permissions labeled by Google LLC as dangerous,[12] and marked apps using these permissions as risky. Enck et al. [17] proposed a system that identifies if an app uses a dangerous combination of permissions. To this end, authors manually defined a set of permission combinations, such as `WRITE_SMS` and `SEND_SMS`, representing a risk. Via static analysis, they then labelled an app as malicious if it makes use of these combinations. All the above-mentioned approaches determine the app's risk depending only on its requested permissions. However, malicious apps could circumvent the permission system and gain access to protected data by applying side channels [3], such as employing device sensors to uniquely identify users [18], or using the MAC address of the WiFi access point to infer user's location [19].

To overcome this issue, similarly to this proposal, some studies proposed to exploit static analysis of app's code to detect the APIs/libraries usage rather than only simply permission requests. As an example, the approaches proposed by [20–23] assumed an API/library secure (i.e., "regular") when it is used by several apps (as an example, more than 60% in [21]). On this basis, these proposals cluster apps based on their APIs/libraries usage. They label an app as not risky if it belongs to a cluster, that is, the app exploits API/libraries commonly used, or risky if it is an outlier. For instance, Backes et al. [20] focus only on third party libraries to detect risky apps (aka, outlier). Wang et al. [21] considered the APIs exploited by apps of the same category to define regular usage and detect risky apps. Whereas,

[12] The list of dangerous permission is available at: https://developer.android.com/reference/android/Manifest.permission.

leveraging on machine learning classifiers, Zhang et al. [22] developed APIGraph to model app's API usage, for malware detection.

Static analysis has been exploited also by Zhuo et al. [24], and Abhishek et al. [25] to generate a graph representing the data flow among API/classes and the possible data collection. In particular, these approaches build two distinct graphs, the first considering a set of malicious apps and the second one based on a set of apps considered safe. Then, by comparing the two graphs, authors are able to identify whether an app has to be considered dangerous.

However, the main limitation of the above-mentioned approaches is that they provide a binary label (i.e., risky/not risky) for apps, which might not be so practical in assisting users in their decisions.

To cope with this limitation, in [4] we proposed a risk assessment approach able to assign a risk level to a mobile app leveraging on static analysis. As explained also in Section 3, given a target app, the risk level is computed based on the deviation of its behaviour w.r.t personal data collection (app signature) from the common behaviour (called category signature) of apps having the same business goal of the target app. Even if this solution overcome some of the limitations of previous proposals based on static analysis, it does not take into consideration whether apps share the collected data to external parties, which might compromise user privacy. Moreover, static analysis techniques have some limitations in case of code obfuscation [26], code encoding [27], and dynamic loading [28].

7.3 Risk Estimation Based on Hybrid Analysis

In general, dynamic analysis can be used to capture the app's malicious behaviors at runtime, overcoming the main drawbacks of static analysis mentioned in the previous section. For instance, by monitoring the network traffic, it is possible to detect whether an app shares the collected data with third parties [29, 30].

However, dynamic analysis might demand high resources and a long time to perform the analysis. To overcome this limitation, hybrid approaches combining static and dynamic analysis are emerging [31, 32]. As an example, some proposals (e.g., SmartDroid [33]) exploited static analysis to identify user-interface (UI) components to be then dynamically analyzed, rather than evaluating all UI components.

Similarly, other approaches used static analysis to speed up subsequent dynamic analysis, by removing some unnecessary tests in the app's runtime analysis, that do not compromise the final result [34, 35]. Cam et al. [36] developed uitHydroid, a tool to detect the collection of sensitive data by apps. In particular, it exploited static analysis of permissions to determine the types of required data (e.g., address, personal information). Then, uitHydroid applied the dynamic analysis to monitor the actual data flow exiting from the mobile to capture any sharing activity that leaks the collected data to other apps/parties. Similarly, AspectDroid [37] first exploited static analysis to determine the expected collected data based on the required permissions. Via an automated testing environment, AspectDroid then detected the abuse resource behavior, i.e., data collected without the corresponding permission.

A further proposal, Sensdroid [38], exploited the hybrid approach to detect collaboration among apps to collect user data (i.e., side-channel attacks). In particular, it classified permissions into two groups of intents, namely explicit and implicit intent. The explicit-intentioned permissions allow the collection of data that are used only by app's execution without sharing them with other parties. In contrast, implicit-intentioned permissions allow data sharing with other apps/parties. Sensdroid exploited dynamic analysis to determine the implicit-based permissions of the installed apps.

In contrast, Hou et al. [39] introduced a structured heterogeneous information network (called HinDroid) for malware detection. HinDroid models not only API calls but also relationships among them (e.g., API calls belonging to the same code block, having the same package name, or using the same invoke method). Based on this network, HinDroid is able to determine the similarity level between two apps. Malware can be detected by identifying similarities between the target apps and the labelled apps (i.e., malware dataset). Along the same line, Ye et al. [40] focused not only on exploiting relationships between API calls but also relationships between apps within the same ecosystem (i.e., whether they coexist in the same smartphone, they are developed by the same developer, or manufactured by the same company). In addition, [40] proposed the HG-Learning method based on a deep neural network (DNN) classifier to detect anomalous malware behavior. This approach was then extended in [41] to cope with new attack techniques; for instance, Command and Control (CC) malware (e.g., TigerEyeing trojan).

Nevertheless, the above-mentioned approaches focus only on specific data/resource (e.g., SMS [37], API calls [39–41]), or permissions [38, 42] extracted from `Android-Manifest.xml`. In particular, the static analysis is adopted in a coarse-grained manner since these approaches only consider permissions and APIs. Therefore, it is not possible to detect fine-grained personal data leakage (e.g., longitude, latitude, locale). In contrast, in our approach we perform a more fine-grained analysis, as we analyze functions/constants calls. Moreover, the focus of the above-mentioned approaches is binary malware detection, whereas our target is to determine the risk of an app (also a begign one) in terms of leakage of personal information. As such, we keep track of information such as the purpose of data collection and sharing, which are not considered by the above-mentioned proposals.

8 Conclusions

This paper addresses the issue of assessing an app's risk level, by proposing an approach that estimate the risk based on the data collection behavior of an app and its data sharing pattern. More precisely, we have proposed an hybrid analysis approach consisting of two phases i) static analysis of app's code to determine data collection behavior; ii) dynamic analysis of the network traffic to determine data sharing behavior. We experimentally evaluate our approach with both expert users and crowd-workers, and the achieved results are encouraging. In the future, we plan to conduct a more comprehensive experimental assessment. We also plan to extend the approach to detect possible mismatches between an app privacy policy and its actual behavior w.r.t personal data usage. Finally, we plan to test alternative modellings of category and app signatures (e.g., taking also into account the frequency of functions/constants usage).

Acknowledgements 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. An extreme and special gratitude for the helpful support of Mr. Santabarbara Mauro; Dr. Anh-Tu Hoang; and Mr. Ahmed Lekssays. We also express our sincere gratitude to the experts and crowd-workers who joined our survey and provided high-quality feedback.

References

1. Bamberger KA (2020) Can you pay for privacy? consumer expectations and the behavior of free and paid apps. Berkeley Tech LJ 35:327
2. Momen N, Hatamian M, Fritsch L (2019) Did app privacy improve after the GDPR? IEEE Secur Priv 17(6):10–20. https://doi.org/10.1109/MSEC.2019.2938445
3. Reardon J (2019) 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In: 28th USENIX Security Symposium (USENIX Security 19), pp. 603–620
4. Son HX, Carminati B, Ferrari E (2021) A risk assessment mechanism for android apps. In: 2021 IEEE International Conference on Smart Internet of Things (SmartIoT), pp. 237–244. IEEE
5. Chandrasekaran D, Mago V (2020) Evolution of semantic similarity–a survey. In: arXiv preprint arXiv:2004.13820
6. Wu Z, Palmer M (1994) Verb semantics and lexical selection. In: arXiv preprint cmp-lg/9406033
7. Jin H (2018) Why are they collecting my data? inferring the purposes of network traffic in mobile apps. Proc ACM Interact Mob Wear Ubiquitous Technol 2(4):1–27
8. Sachdev SB, Verma HV (2004) Relative importance of service quality dimensions: A multisectoral study. J Services Res **4**(1)
9. Sarma BP (2012) Android permissions: a perspective combining risks and benefits. In: Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, pp. 13–22
10. Peng H (2012) Using probabilistic generative models for ranking risks of android apps. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 241–252
11. Wu Z, Chen X, Lee SU-J (2020) Fcdp: Fidelity calculation for description-to-permissions in android apps. IEEE Access
12. Chia PH (2012) 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, pp. 311–320
13. Olukoya O, Mackenzie L, Omoronyia I (2020) Security-oriented view of app behaviour using textual descriptions and user-granted permission requests. Comput Secur 89:101685
14. Felt AP (2011) Android permissions demystified. In: 18th ACM Conference on Computer and Communications Security, pp. 627–638
15. Alazab M (2020) Intelligent mobile malware detection using permission requests and api calls. Future Gener Comput Syst 107:509–521
16. Xiao J (2020) An android application risk evaluation framework based on minimum permission set identification. J Syst Softw 163:110533
17. Enck W (2014) Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans Comput Syst (TOCS) 32(2):1–29
18. Simon L et al. (2016) Don't interrupt me while i type: Inferring text entered through gesture typing on android keyboards. Priv Enhancing Technol, 136–154
19. Commission FT et al. (2016) Mobile advertising network inmobi settles ftc charges it tracked hundreds of millions of consumers' locations without permission. In: press release (June 22), https://tinyurl.com/h83c2be
20. Backes M (2016) Reliable third-party library detection in android and its security applications. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pp. 356–367
21. Wang H (2015) 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, pp. 71–82
22. Zhang X (2020) Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware. In: Proceedings of the Conference on Computer and Communications Security, pp. 757–770
23. Zhu S (2020) Measuring and modeling the label dynamics of online anti-malware engines. In: 29th USENIX Security Symposium (USENIX Security 20), pp. 2361–2378
24. Singh AK et al (2019) Experimental analysis of android malware detection based on combinations of permissions and api-calls. J Comput Virol Hacking Tech 15(3):209–218
25. Ma Z (2019) A combination method for android malware detection based on control flow graphs and machine learning algorithms. IEEE Access 7:21235–21245

26. Yang Y (2018) Droidward: an effective dynamic analysis method for vetting android applications. Cluster Comput 21(1):265–275
27. Yuan Z (2014) Droid-sec: deep learning in android malware detection. In: Proceedings of ACM Conference on SIGCOMM, pp. 371–372
28. Schütte J (2015) Condroid: Targeted dynamic analysis of android applications. In: 2015 IEEE 29th International Conference on Advanced Information Networking and Applications, pp. 571–578. IEEE
29. Shuba A, Markopoulou A (2020) Nomoats: towards automatic detection of mobile tracking. Proc Priv Enhancing Technol 2020(2):45–66
30. Yurekten O, Demirci M (2021) Sdn-based cyber defense: a survey. Future Gener Comput Syst 115:126–149
31. Gajrani J (2020) Effectiveness of state-of-the-art dynamic analysis techniques in identifying diverse android malware and future enhancements. In: Advances in Computers vol. 119, pp. 73–120. Elsevier,
32. Wang W (2020) Botmark: automated botnet detection with hybrid analysis of flow-based and graph-based traffic behaviors. Inf Sci 511:284–296
33. Zheng C (2012) Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications. In: ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 93–104
34. Devecsery D (2018) Optimistic hybrid analysis: Accelerating dynamic analysis through predicated static analysis. In: 23rd International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 348–362
35. Palit T (2021) Dynpta: Combining static and dynamic analysis for practical selective data protection. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 1919–1937 . IEEE
36. Cam NT (2019) Detecting sensitive data leakage via inter-applications on android using a hybrid analysis technique. Cluster Comput 22(1):1055–1064
37. Ali-Gombe AI (2018) Toward a more dependable hybrid analysis of android malware using aspect-oriented programming. Comput Secur 73, 235–248
38. Shrivastava G, Kumar P (2019) Sensdroid: analysis for malicious activity risk of android application. Multimed Tools Appl 78(24):35713–35731
39. Hou S, Ye Y, Song Y, Abdulhayoglu M (2017) Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1507–1515
40. Ye Y, Hou S, Chen L, Lei J, Wan W, Wang J, Xiong Q, Shao F (2019) Out-of-sample node representation learning for heterogeneous graph in real-time android malware detection. In: 28th International Joint Conference on Artificial Intelligence (IJCAI)
41. Fan Y, Ju M, Hou S, Ye Y, Wan W, Wang K, Mei Y, Xiong Q (2021) Heterogeneous temporal graph transformer: An intelligent system for evolving android malware detection. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 2831–2839
42. Cam NT (2016) Android security analysis based on inter-application relationships. Information Science and Applications (ICISA) 2016. Springer, Singapore, pp 689–700