# PTPDroid: Detecting Violated User Privacy Disclosures to Third-Parties of Android Apps

Zeya Tan
*School of Computer Science and Engineering*
*Nanjing University of Science and Technology*
Nanjing, China
tanzeya@qq.com

Wei Song*
*School of Computer Science and Engineering*
*Nanjing University of Science and Technology*
Nanjing, China
wsong@njust.edu.cn

***Abstract*—Android apps frequently access personal information to provide customized services. Since such information is sensitive in general, regulators require Android app vendors to publish privacy policies that describe what information is collected and why it is collected. Existing work mainly focuses on the types of the collected data but seldom considers the entities that collect user privacy, which could falsely classify problematic declarations about user privacy collected by third-parties into clear disclosures. To address this problem, we propose PTPDroid, a flow-to-policy consistency checking approach and an automated tool, to comprehensively uncover from the privacy policy the violated disclosures to third-parties. Our experiments on real-world apps demonstrate the effectiveness and superiority of PTPDroid, and our empirical study on 1,000 popular real-world apps reveals that violated user privacy disclosures to third-parties are prevalent in practice.**

***Index Terms*—Android app, privacy policy, third-party entities, violation detection, taint analysis, empirical study**

## I. Introduction

Mobile applications (apps) are becoming increasingly pervasive. By March 2022, the apps in Google Play Store have surpassed 2.9 million [1], and the Android platform has held 83% of the smartphone OS market share [2]. With the increasing market share and user acceptance, the risk of user privacy leak and misuse becomes a tricky problem, because apps increasingly use private information such as users' locations, network information, and unique device information to better provide customized services. Although Android operating system applies a user-based permission system [3], [4] to limit the sensitive information an app can obtain, data leakage and misuse are still possible [5], [6], which could be due to the low granularity of the permissions [7] and the ambiguity of the phrases presented to users when they install an app [8]. Hence, to protect user privacy, regulators, such as the U.S. Federal Trade Commission (FTC), ask the developers to provide the privacy policy in natural languages to enumerate how applications collect, use, and share personal information [9]. App developers are subject to a host of privacy requirements that they have to comply with when disclosing their app privacy policies.

Despite these efforts on advocating privacy policies, like any software documentation, there are possibilities for the privacy policies to become inconsistent with the app code. Privacy policies can be written by the persons who are not the developers (e.g., lawyers), or the app code can change while the privacy policy does not update accordingly. Such inconsistencies regarding an end user's personal data, intentional or not, can have legal repercussions [10], [11]. It is therefore a good practice for Android apps to accurately and completely state in their privacy policies what and how users' private data is used and for what purposes. However, this is not an easy task for the developers or app vendors to achieve this, especially those from small start-ups.

A recent thread of research begins to study the consistency between privacy policies and Android apps [12], [13], [14], [15]. The goal of these studies is to help developers be aware of the violated disclosures in their privacy policies, and help end users choose privacy-friendly apps. Conceptually, these studies use a combination of static analysis and natural language processing to check the flow-to-policy consistency, that is, to determine whether an app's behavior is consistent with what is declared in the privacy policy.

While such prior studies have led to promising results, they do not differentiate the entities (first-party entities and third-party entities) that collect the data. However, third-party entities are inevitably involved with the increasing interactions between apps and the emerging app-in-app paradigm [16]. Therefore, the requirement that developers disclose the third-party entities with which they are sharing information is grounded in regulations, such as GDPR [17] and CCPA [9]. PoliCheck [18], the sole work considering entities, confirms that entity-insensitive models may falsely classify 38.4% of apps as having privacy-sensitive data flows consistent with their privacy policies when one employs the policy declarations discussing first-party collections to determine the reasonableness of data flows to third-parties in the app code. For accuracy, PoliCheck builds upon AppCensus [19], a dynamic analysis tool to detect data flows that send user private information to remote servers. However, AppCensus only considers a part of data types (media, app list, calendar are not considered). More importantly, the data flows obtained by AppCensus is limited to the observed executions, and thus some disclosures to third-parties may be missed.

To remedy the shortcomings of existing work, we pro-

* Corresponding author.

pose an entity-sensitive flow-to-policy consistency checking approach PTPDroid. A salient merit of PTPDroid is that it can statically obtain more comprehensive data flows that user privacy are collected by third-parties. PTPDroid tracks sensitive data by starting at a pre-defined source (e.g., an API method returning location information) and then following the data until it reaches a given sink (e.g., a method sending the information to the network). For the obtained data flows, PTPDroid further determines the data types and third-party entities, and finally classifies the data flows into four types of disclosures, including *clear disclosures*, *vague disclosures*, *omitted disclosures* , and *incorrect (denied) disclosures* (see Section III). We use PTPDroid to study the flow-to-policy consistency of 1,000 real-world Android apps and find that violated user privacy disclosures to third-parties are prevalent in practice. In those apps, only 3.8% (38/1,000) of them clearly state the third-party entities and the shared data types in their privacy policies, and 19% (190/1,000) of them omit or deny the actual collection of user privacy data by third-parties in their privacy policies.

In this study, we make the following contributions:

- We propose a flow-to-policy consistency checking approach PTPDroid, which is dedicated to comprehensively detect the violated user privacy disclosures to third-parties in the privacy policy.
- Based on PolicyLint [20] and FlowDroid [21], we implement PTPDroid as an open-source tool.
- We evaluate the effectiveness and efficiency of PTPDroid and show its superiority over the state-of-the-art with real-world apps. Our empirical study on 1,000 popular real-world Android apps reveals the severity of the violated user privacy disclosures to third-parties in practice.

The rest of this paper proceeds as follows. Section II reviews the background knowledge. Section III presents our approach. Section IV evaluates our approach. Section V reviews the related work and Section VI concludes the paper.

## II. Background

In this section, we introduce the background knowledge.

### A. Privacy Policy

Privacy policy serves as the primary means to communicate with users regarding which and how user private information is accessed, collected, stored, shared, used/processed, and the purposes of the information collection and sharing. From privacy policies, users can learn about the behaviors in apps that are not transparent to them. For example, Figure 1 exhibits the content of the privacy policy of an app (*SHEIN*)[1], which covers all aspects that a privacy policy needs to state. There are two most crucial parts. The first part describes which private information will be collected by the app itself (first-party) and the reason why the information is used for. The second part demonstrates which private information will be shared with which third-party entities. In this paper, we focus on the second part, that is, we only analyze whether the description of private information collected by third-parties in the privacy policy is consistent with the relevant behavior in the app code.

[1]https://us.shein.com/

It is worth mentioning that the third-party entities we refer to in this paper only include advertisers, analytics providers, and service providers, which have specific names. Since policy declarations cannot be used for automated reasoning, we need natural language processing to extract from the policy the formal representations of the data sharing declarations [20]. In this paper, we represent data sharing declarations as a three-tuple (*entity, action, data type*) where the entity performs action (collects, not collect) on the data type. For instance, the declaration "We may share cookies with *AppsFlyer* to facilitate our provision of the service." can be represented by the tuple (*AppsFlyer*, collects, cookie).


Fig. 1. The content of the privacy policy of *SHEIN*.

### B. Ontology

Privacy policies may disclose data collection using a coarse-grained description rather than the concrete data flows. For example, a privacy policy may specify (*advertiser*, collects, information) to disclose the data flow (*AdMob*, collects, phone number). To match the policy declaration to the data flow, it is important to be able to build the *synonym* and *subsumptive* relationships between phrases. Such relationships are often encoded into an *ontology*, which is a rooted directed acyclic graph where nodes are ontology phrases and edges are marked with the relationship between those phrases.

An ontology is a formal description of entities and their properties, relationships, and behaviors [22], [23]. PTPDroid uses two ontologies to represent the hierarchical classification of data types and third-party entities, respectively. The hierarchical nature of an ontology allows for transitive relationships that can be used for mapping strings to phrases indirectly.

## III. PTPDroid

As shown in Figure 2, PTPDroid takes the privacy policy and APK file of an app as input, and yields three kinds of violated disclosures (about user privacy sharing with third-parties) in the privacy policy as output, including: 1) *vague disclosures*, referring to the situations that the declarations

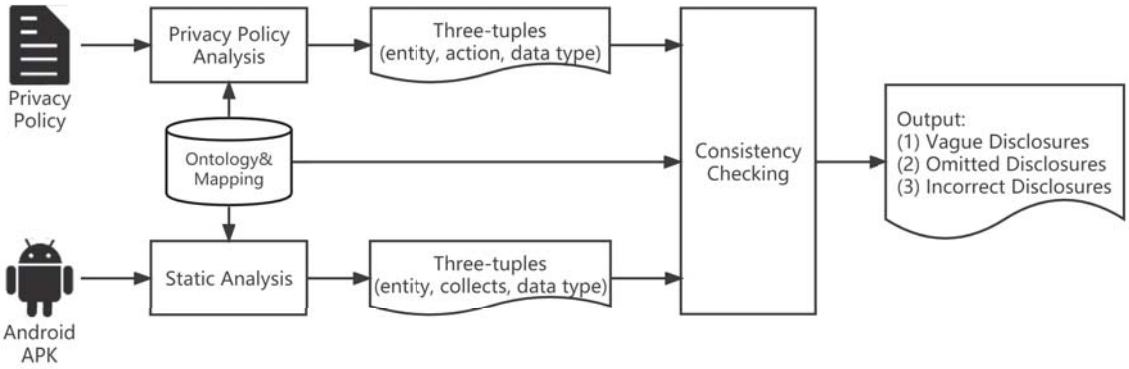


Fig. 2. Workflow of PTPDroid.

about third-parties in the privacy policy use more general phrases to describe the concrete data types or third-party entities in the app code; 2) *omitted disclosures*, referring to the situations that the specific behaviors that disclose the user privacy to third-parties in the app code are not mentioned in the privacy policy; 3) *incorrect (denied) disclosures*, referring to the situations that the privacy policy denies the privacy sharing behaviors to third-parties that actually take place in the app code. The output could be an empty set if there are no such violated disclosures. The workflow of PTPDroid consists of the following four steps:

1) **Ontology and mapping establishment**. In this step, we construct ontologies of private information and third-party entities, respectively. Additionally, the mapping from strings (e.g., method names or variable values) of the relevant private information and third-parties to phrases in the ontologies are established. Through the mapping, all the concerned information obtained from the app code and privacy policy will be converted into phrases in the ontologies.
2) **Privacy policy analysis**. In this step, we analyze the privacy policy to extract all declarations related to the user privacy collected by third-parties. These declarations are converted into three-tuples, in the form of (*entity, action, data type*), to facilitate consistency checking.
3) **Static analysis**. This step inspects the APK file through static taint analysis to detect the data flows that the app shares user private information to third-parties. The relevant strings in the *sink methods* and *source methods* are transformed into phrases in the ontologies via the constructed mapping. A series of three-tuples, in the form of (*entity, collects, data type*), are obtained.
4) **Consistency checking**. This step takes in two sets of tuples obtained from privacy policy analysis and static analysis respectively to detect inconsistency between them. The details of each inconsistency are enumerated, including the type of inconsistency, the relevant privacy policy declaration, and the relevant data flow in the code.

TABLE I
DATA TYPES TRACKED VIA STATIC ANALYSIS

| **Data types** |
|---|
| Ad ID, Android ID, serial number, IMEI, cookie, IP Address, media, app list, contact, calendar, email, account, location, phone number |

### A. Ontology and Mapping Establishment

A common phenomenon in natural languages is *generalization*, in which a more general phrase can be utilized to imply several sub-concepts of the phrase. Since generalization can be applied to both private information and third-party entities, we build the ontologies for user privacy and third-party entities, respectively. While ontologies can represent several different types of relationships, we only need to consider subsumptive ($\supset$) and synonym ($\equiv$) relationships which are sufficient for our work. Based on the forms of strings of the relevant private information and third-party entities represented in the app code, we build several phrase mappings: 1) API methods mapping to phrases; 2) URI strings mapping to phrases; 3) package names mapping to third-party entities; 4) destination domains mapping to third-party entities; 5) IP addresses mapping to third-party entities.

**Ontology construction**. Our analysis method is based on PolicyLint [20], which is the state-of-the-art for privacy policy analysis. We extend its privacy ontology and construct the third-party ontology based on its generated third-party entities.

Table I lists all the data types we considered, which also indicates that our privacy ontology only contains these phrases and their *hypernyms*. Our privacy ontology extends that of PolicyLint to add the data types PolicyLint misses, e.g., media, app list, calendar, etc. We manually add the missing data types and the edges between them and their hypernyms to the privacy ontology.

We choose third-party entities and determine their relationships through `Appbrain`[2], a web site that lists the most popular analytics and advertising organizations. For example,

[2]https://appbrain.com/

TABLE II
WAYS TO SHARE INFORMATION WITH THIRD-PARTIES

| Sharing ways |
|---|
| external storage, intent, network, clipboard, content provider, broadcast |

*Unity* is marked as an advertiser on this site. Therefore, we add in the third-party ontology an edge between node *advertiser* and node *Unity* to represent the subsumptive relationship between them. Since the number of apps is explosive and some apps do not collect user private information, it is not advisable to take all of them in our third-party ontology. For this reason, we use the most common advertisers and analytics providers to build the initial third-party ontology, and add the other third-party entities to the ontology manually when they are required in the subsequent experiments. Altogether, 67 and 89 third-party entities are initialized and finally involved, respectively.

**Mapping establishment**. With the ontologies constructed via the above steps, the strings (e.g., APIs and variables) of the private information and third-parties in the app code could then be mapped to phrases in the ontologies.

An app can collect personal information through three manners. (1) Call sensitive API methods, e.g., `getLastKnownLocation()` is called to obtain the location information. (2) Through content providers [24], e.g., invoking `android.content.ContentResolver.query()` with *content://com.android.calendar* as an argument to access the calendar information. For the two examples above, we can obtain the mapping pairs (`getLastKnownLocation()`, location) and (*content://com.android.calendar*, calendar), respectively. To look for the sensitive API methods and URI strings that access different types of private information, we resort to the data sets in SUSI [25] and Pscout [24]. (3) Request to the app server, e.g., use *org.apache.http.HttpResponse: getEntity(advertising_identifier)* to get the server data. For this, we obtain the sensitive data from the arguments of the API invocation based on keyword similarity, for example, the argument *advertising_identifier* is mapped to Ad ID.

User privacy can be sent to third-parties through several ways, and the common ways (provided by the developers from ByteDance) are listed in Table II. We only consider the first three ways as for the others we cannot identify a specific third-party entity. For the first two ways, we first locate the API invocations relevant to external storage or intent, and then we can obtain the third-parties from the names of the packages that these API invocations belong to. Next, we build synonym lists for these third-party entities. Typically, the package name contains terms that are unique to that third-party entity. For example, the package name `com.appsflyer` contains the third-party name *Appsflyer* as keywords. In this case, strings can be mapped directly to third-party entities via keyword matching. In other cases, for example, since the package name `com.firefox` belongs to *Mozilla*, we manually extend *Mozilla*'s synonym list to {*Firefox*, *Mozilla*}.

If an app sends some information to third-parties through the network-related APIs, the arguments of these API invocations are URLs in the form of destination domains or IP addresses. If the URL is in the form of a domain, we can use either of the two ways to determine the third-party entities. For the domains like http://app.adjust.com, we use keyword extraction to directly obtain the entity *Adjust*. While for the domains like http://www.firefox.com, we manually construct a mapping pair (http://www.firefox.com, *Mozilla*). Note that for the URL that involves only IP address without domain name, we follow PoliCheck [18] to perform a reverse-DNS search to resolve the IP Address as a domain name. If it works, we construct a mapping between the IP Address and the domain name. Otherwise, we discard this IP Address.

### B. Privacy Policy Analysis

To analyze privacy policies, we leverage PolicyLint [20], which enhances prior approaches by extracting entities and negative sentiment declarations. PolicyLint represents a policy declaration as a four-tuple (*actor, action, data type, entity*). For example, the declaration "We may share your location information with advertisers" is represented as (we, share, location, advertiser) by PolicyLint. As our analysis only focuses on the behavior of third-parties, the field *actor* is useless to our analysis. Therefore, we reduce the four-tuple to a three-tuple (*entity, action, data type*). Meanwhile, we abstract all positive actions into "collects" and all negative actions into "not collect". For the example above, the three-tuple obtained by our privacy policy analysis is (*advertiser*, collects, location).

It is worth mentioning that PolicyLint can find inconsistencies in a privacy policy, which are referred to *ambiguous disclosures* [20]. If the privacy policy contains *logical contradictions*, i.e., two diametrically opposed descriptions, we do not conduct the subsequent analysis for these contradictory declarations because they inevitably affect the consistency checking.

### C. Static Analysis

Given the APK file of an app, we first perform the static taint analysis to obtain the data flows that user private information is collected by third-party entities. For the data flows, we then determine the data types and third-parties related to the source methods and sink methods based on the call graph. Finally, we transform the obtained information (strings) into ontology phrases according to our established mapping.

Before going into the details, we use an example of sharing the device identifier to *Facebook* for illustrating our static analysis (cf. Figure 3a). First, `getDeviceId()` is specified as the source method because it gets user data, and `sendDataByPost()` is specified as the sink method, because it sends something outside. Based on the taint analysis, we obtain a data flow from `getDeviceId()` to `sendDataByPost()`. Then, we determine which data type and which entities are involved in this data flow. Based on the pre-established mapping discussed in Section III-A, we know that the data type related to the source method is the

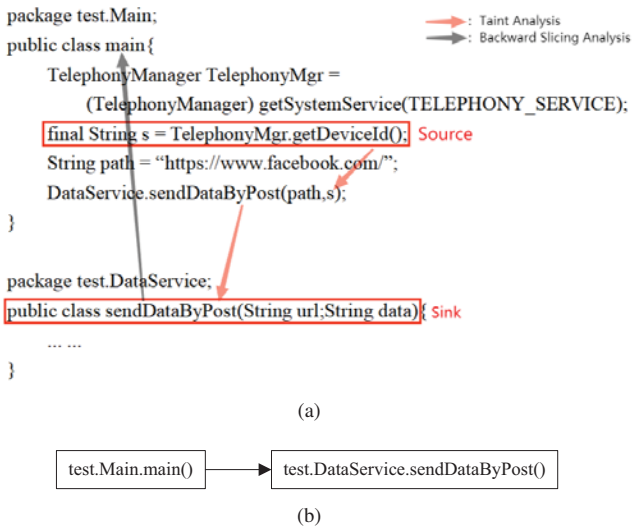


Fig. 3. Illustration of the static analysis: (a) Code snippet; (b) Call graph.

device identifier. To get the entities, we search backward for the variables (whose value reflects third-party entities) in all methods that point to the sink method in the call graph (cf. Figure 3b). In the function `main()` which invokes the sink method (`sendDataByPost()`), we find that a variable $s$ is passed to the sink method, and the variable value https://www.facebook.com indicates that the entity is *Facebook* according to the mapping based on keyword similarity. Finally, we get a three-tuple (*Facebook*, collects, device identifier).

**Static taint analysis**. Based on FlowDroid [21], we develop the static taint analysis module to track the data flows in the app bytecode. Altogether, we choose 78 APIs (from Android SDK) that collect user privacy data as the source methods, and choose 82 APIs that transfer or store Data as the sink methods. The output of Flowdroid is a series of tuples in the form of (*source method, sink method*). Each tuple implies the existence of a path from the source method to the sink method. We extend FlowDroid to also output the signatures of the source method and the sink method. The signatures are used to locate the methods in the call graph. Therefore, the output is converted to a set of four-tuples in the form of (*source method, source signature, sink method, sink signature*).

**Analysis based on call graph**. The results obtained through the static taint analysis may not be directly used, because for some source methods (e.g., `ContentResolver.query()`) and sink methods (e.g., `HttpResponse.execute()`), we cannot get the data types and entities directly from the method names. In this case, we search for the data types and entities in the methods which (in)directly call the source method based on the call graph.

For source methods, there are two situations in which we cannot obtain the data types directly from the API names based on the pre-established mappings from APIs to ontology phrases: (1) APIs access content providers and (2) APIs request to the server. For the former, we first locate the API node in the call graph. Then, from the node, we adopt a reverse DFS (depth-first search) strategy to traverse the call graph to find all the methods which define the URIs of the content providers and passed the URIs to the source method. Since the URIs of content providers have already mapped to phrases in the privacy ontology, we know the data types collected by the source method. For instance, if a method invokes `Uri.parse(content://com.android.calendar)` to get the URI object, *content://com.android.calendar* is recorded to obtain the mapped data type. For the latter, we first find such source methods (e.g., `org.apache.http.HttpResponse: getEntity()`) in the call graph. Then, we obtain the sensitive strings similarly based on the call graph. For example, if the string $advertising\_id$ is found in a method in the call graph and it is passed to the source method, we record $advertising\_id$ to get the mapped data type.

For sink methods, if they are the network-related APIs (in this case, we cannot directly obtain the entities from the package name of the APIs), we also employ a reverse DFS search in the call graph to determine the URL strings (e.g., http://graph.facebook.com). If they access a third-party through intent (e.g., `Intent.setClassName()` or `Intent.setComponentName()`), we record all the strings passed to the methods and obtain the strings related to third-party entities based on keyword similarity. In some scenarios, there exist no useful strings in the method for some reasons, such as requesting the URL at runtime and getting the URL by accessing the internal storage. In these cases, we record the class name of the sink method. If the class name matches a third-party SDK, we regard that the third-party collects user private information.

**Normalization**. Although the strings in the data flows contain information about third-party entities and data types, they also need to be mapped to the phrases in ontologies to facilitate consistency checking. For each of the strings, we first determine whether it contains an ontology phrase by keyword matching (e.g., http://graph.facebook.com contains the ontology phrase *Facebook*). If it does, we directly convert it to the ontology phrase. Otherwise, we convert it to an ontology phrase based on the pre-established mappings. If there is no mapping pair hit, we discard the data flow.

**Filtering first-party entities**. It is worth mentioning that some data flows we detect are relevant to the first-party. Owing to the scope of this paper, those data flows need to be excluded. Therefore, when we analyze an app, we first build a set that represents the first-party. For example, when we analyze the app *Instagram*, the first-party set consists of *Instagram* and *Facebook*. If the strings recorded match one of the first-party entities, we discard such data flows.

### D. Consistency Checking

The privacy policy analysis returns a set of three-tuples summarizing the user privacy data collected by third-parties stated in the privacy policy (summarized in $PolicyResults$), while the static analysis yields a set of three-tuples showing

**Algorithm 1** Consistency Analysis

**Input:** *PolicyResults*: three-tuples (*entity, action, info*) disclosed by privacy policy, *FlowResults*: three-tuples (*entity, collects, info*) obtained from static analysis, Privacy Ontology, Third-party Ontology
**Output:** *ClearDisclosure*, *VagueDisclosure*, *OmittedDisclosure*, *DeniedDisclosure*

```
1: ClearDisclosure, VagueDisclosure, OmittedDisclosure, DeniedDisclosure = ∅;
2: for each FlowRes ∈ FlowResults do
3:     isOmitted = true; isVague = true;
4:     for each PoliRes ∈ PolicyResults do
5:         if PoliRes.entity ⊒ FlowRes.entity ∧ PoliRes.info ⊒ FlowRes.info then
6:             isOmitted = false;
7:             if PoliRes.action == not collect then
8:                 DeniedDisclosure.append(FlowRes, PoliRes);
9:                 isVague = false;
10:            else
11:                if FlowRes.entity≡PoliRes.entity∧FlowRes.info≡PoliRes.info then
12:                    ClearDisclosure.append(FlowRes);
13:                    isVague = false;
14:    if isOmitted == true then
15:        OmittedDisclosure.append(FlowRes);
16:    else
17:        if isVague == true then
18:            VagueDisclosure.append(FlowRes), PoliRes);
```

the sensitive information shared to third-parties in the app code (summarized in $FlowResults$). Algorithm 1 compares each tuple in $FlowResults$ with all tuples in $PolicyResults$ to detect the violated disclosures.

In Algorithm 1, for each tuple $FlowRes$ in $FlowResults$, we maintain two variables $isOmitted$ and $isVague$, which are initialized to true. As we iterate through the tuples in $PolicyResults$, the value of $isOmitted$ is changed to false if a match is found. After we have gone through all the tuples in $PolicyResults$, if the value of $isOmitted$ is true, it implies that no match to $FlowRes$ is found in the privacy policy, and thus $FlowRes$ is an omitted disclosure. If the value of $isVague$ is true, it implies that there exists no clear disclosure or denied disclosure, and thus $FlowRes$ is a vague disclosure. Here, "match" ($\sqsupseteq$) means that the third-party and data type of $PoliRes$ and those of $FlowRes$ are in subsumptive or synonym relationships, respectively. In this situation, if the declaration of $PoliRes$ is with a negative sentiment (not collect), it is a denied disclosure. If the declaration of $PoliRes$ is with a positive sentiment (collects), we then further determine whether it is equivalent to $FlowRes$. If yes, it is a clear disclosure. Otherwise, it is a vague disclosure.

## IV. Evaluation

In this section, we evaluate PTPDroid with large real-world Android apps, and aim to answer the following research questions (RQs).

- **RQ1:** *Efficacy*. What is the accuracy of PTPDroid in analyzing app code and privacy policies, respectively? And, how efficient is the analysis?
- **RQ2:** *Superiority*. Compared to the state-of-the-art, what is the advantage of PTPDroid?
- **RQ3:** *Empirical study*. Are violated user privacy disclosures to third-parties prevalent in practice?

**Data availability**. We implement PTPDroid in Java and publish it as an open-source tool. We use PTPDroid to analyze real-world apps on a computer with an Inter Core i7-4790 3.60GHz CPU and 16 GB of memory, running Windows 8.1, JDK 1.8, and Android 8.0.

### A. Effectiveness and Efficiency

It is challenging to establish the ground truth to evaluate the effectiveness of both static analysis and privacy policy analysis. Since the most popular Android apps are well-tested and long-recognized, we expect that they faithfully and correctly state which private information is shared to third-parties. With this expectation, we seek the most popular apps whose privacy policies clearly state the data flows to third-parties, and utilize their privacy policies as the ground truth for both static analysis and privacy policy analysis.

Our experiment to answer RQ1 proceeds as follows. Firstly, we download the recommended 100 best Android apps[3] (July, 2021) and the top-200 Android apps from Google Play (July, 2021). Since these two sets of apps have some overlap, we obtain 276 apps in total. Secondly, we manually check the privacy policies of the 276 apps, and unfortunately, only 24 apps whose privacy policies clearly state how the user privacy discloses to third-parties are preserved. Of the 24 apps, a few app privacy policies employ tables to state user privacy disclosures to third-parties. For this case, we transform the tables into plain texts of natural language. Thirdly, we employ the 24 apps as the benchmark to conduct the experiment. Specifically, we ask two group members (not authors of this paper) to read the privacy policies of the 24 apps, and to independently write down the three-tuple for each of the item that is relevant to third-parties. If the tuples written by the two group members are inconsistent, the third group member is involved to resolve the differences. In this way, we obtain a tuple set $T$ for each of the 24 apps, and this tuple set is used as the ground truth for both static analysis and privacy policy analysis. Finally, we apply PTPDroid to the 24 apps. Assume that the obtained results (tuple sets) of the static analysis and privacy policy analysis are recorded in $T_s$ and $T_p$, respectively. We adopt the F-score of precision and recall to evaluate the effectiveness of the two analysis. For example, the F-score of the static analysis is obtained as follows: $precision = \frac{|T_s \cap T|}{|T_s|}$, $recall = \frac{|T_s \cap T|}{|T|}$, and $F\text{-}measure = 2\times \frac{precision \times recall}{precision + recall}$.

Table III summarizes the experimental results, where the columns "Recall" and "F-score" list the actual recall and F-score, as well as those (in the brackets) when the false negatives caused by code obfuscation are excluded. The average precision of our static analysis reaches up to 87.9%. For those false positives of the static analysis, most are due to that our construction of third-party ontology is inadequate. For example, we detect that the third-party *Moat* collects the device identifier in more than one app (*FileMaster*, *OneBooster*, and *Ibis*) but this is not declared in the privacy policies. In this case, we find that the privacy policy uses another phrase (such as the organization name of *Moat*) to replace *Moat*, so this can also be regarded that the privacy policy has a vague disclosure. The average recall of our static analysis reaches to

[3]https://www.digitaltrends.com/mobile/best-android-apps/

TABLE III
EXPERIMENTAL RESULTS ANSWERING RQ1

| App | Static analysis | | | | Policy analysis | |
|---|---|---|---|---|---|---|
| | Precision | Recall | F-score | Time | F-score | Time |
| *Duolingo* | 100% | 75% (100%) | 85.7% (100%) | 12.3 m | 100% | 2.8 m |
| *Kinemaster* | 100% | 66.7% (66.7%) | 80.0% (80%) | 10.3 m | 100% | 2.6 m |
| *APnews* | 66.7% | 50% (75%) | 57.2% (70.6%) | 9.1 m | 100% | 1.8 m |
| *Moon+* | 50% | 50% (100%) | 50% (66.7%) | 9.7 m | 100% | 3.4 m |
| *ShopPackage* | 66.7% | 40% (60%) | 50.0% (63.2%) | 12.7 m | 100% | 2.4 m |
| *Kik* | 100% | 33.3% (66.7%) | 50.0% (80.0%) | 5.4 m | 100% | 2.5 m |
| *MicrosoftT∗* | 100% | 33.3% (33.3%) | 50.0% (50.0%) | 6.2 m | 100% | 2.1 m |
| *Indeed* | 100% | 25% (50%) | 40% (66.7%) | 6.5 m | 100% | 2.8 m |
| *Geocaching* | 100% | 25% (50%) | 40% (66.7%) | 7.1 m | 100% | 2.0 m |
| *BodyFast* | 75% | 75% (75%) | 75% (75%) | 8.3 m | 100% | 1.8m |
| *FileMaster* | 88.9% | 53.3% (80%) | 66.6% (84.2%) | 6.2 m | 100% | 3.9 m |
| *Urbandroid* | 100% | 100% (100%) | 100% (100%) | 12.6 m | 100% | 2.4 m |
| *EverMatch* | 80% | 75% (75%) | 77.4% (77.4%) | 9.2 m | 100% | 2.5 m |
| *Headway* | 100% | 40% (100%) | 57.1% (100%) | 5.3 m | 100% | 1.6 m |
| *OneBooster* | 91.6% | 78.6% (85.7%) | 84.6% (88.6%) | 10.5 m | 100% | 2.5 m |
| *Ibis* | 83.3% | 71.4% (71.4%) | 76.9% (76.9%) | 8.7 m | 100% | 3.3 m |
| *Kptncook* | 100% | 75% (100%) | 85.7% (100%) | 11.3 m | 100% | 2.7 m |
| *Reality* | 100% | 100% (100%) | 100% (100%) | 17.3 m | 100% | 3.6 m |
| *Ashley* | 100% | 40% (40%) | 57.1% (57.1%) | 8.1m | 100% | 2.4 m |
| *GetContact* | 66.7% | 40% (80%) | 50.5% (72.7%) | 5.9 m | 100% | 2.3 m |
| *Blob* | 85.7% | 75% (81.3%) | 80.0% (83.4%) | 12.1 m | 100% | 2.2 m |
| *Inffy* | 80% | 80% (80%) | 80% (80%) | 9.9 m | 100% | 2.3 m |
| *PhoneOpt∗* | 92.3% | 80% (86.7%) | 85.7% (89.4%) | 12.3 m | 100% | 2.7 m |
| *PowerfulC∗* | 83.3% | 66.7% (80%) | 73.6% (81.6%) | 11.4 m | 100% | 1.9 m |
| **Average** | 87.9% | 60.3% (76.5%) | 68.9% (79.6%) | 9.5 m | 100% | 2.5 m |

60.3%. If we exclude third-parties that are obfuscated in the app code, the average recall can reach to 76.5%. There are two main reasons for the false negatives of the static analysis. First, some third-parties (e.g., websites) can only be obtained at runtime. Second, our static analysis (also the other static analysis techniques) can only analyze the app code of the client side, whereas user privacy can be sent to third-parties by the app code of the server side. The average F-score of the static analysis reaches to 68.9% and if third-parties obfuscated in the code are excluded, the average F-score can increase to 79.6%. These indicate that our static analysis of the data flows to third-parties tends to be effective. The average F-score of the privacy policy analysis is 100%, which demonstrates that our privacy policy analysis that extends PolicyLint is also effective. The average time cost of the static analysis and privacy policy analysis is 9.5 minutes (m) and 2.5 minutes, respectively. Thus, it takes about on average 12 minutes in total to analyze a large Android app.

### B. Comparison and Superiority

To answer RQ2, we compare PTPDroid with PoliCheck [18], the-state-of-the-art tool that can detect violated user privacy disclosures to third-parties. PoliCheck employs PolicyLint [20] to process privacy policies and AppCensus [19] to analyze the app code. Since we also use PolicyLint to analyze privacy policies, we compare PTPDroid with PoliCheck only based on the results of code analysis. Note that AppCensus (thus PoliCheck as well) is not open-source or publicly available, we can only retrieve its analysis results in `AppSearch`[4]. We randomly select 48 popular apps AppCensus analyzed, covering various types and functionalities. We download these apps of the same versions from androidapksfree.com. For the results of Appcensus, only the obtained data flows that are relevant to third-parties are preserved. In addition, we remove data flows

[4]https://search.appcensus.io/

TABLE IV
EXPERIMENTAL RESULTS ANSWERING RQ2

| App | AppCensus | PTPDroid | Intersection |
|---|---|---|---|
| *Accuweather_v6.1.2* | 2 | 11 | 2 |
| *Askfm_v4.38.4* | 0 | 27 | 0 |
| *Bitmoji_v10.61.105* | 1 | 2 | 1 |
| *CalorieCounter_v19.6.10* | 4 | 12 | 4 |
| *Camfrog_v7.0.4.55* | 5 | 10 | 5 |
| *Dailymotion_v1.36.12* | 6 | 4 | 3 |
| *Diskdigger_v1.0* | 0 | 0 | 0 |
| *eBay_v5.38.0.14* | 1 | 6 | 1 |
| *Edmodo_v10.6.1* | 1 | 3 | 1 |
| *FieldTrip_v2.0.9* | 0 | 0 | 0 |
| *FileCommander_v6.2.33122* | 1 | 6 | 1 |
| *FilmoraGo_v3.1.4* | 1 | 4 | 1 |
| *Firefox_v68.3.0* | 0 | 2 | 0 |
| *HAGO_v2.12.7* | 2 | 4 | 2 |
| *Hulu_v3.62.1.307830* | 2 | 6 | 2 |
| *GooglePlayGames_v5.13.7466* | 0 | 0 | 0 |
| *Instagram_v123.0.0.21.114* | 1 | 1 | 1 |
| *JioChat_v3.2.7* | 1 | 2 | 1 |
| *Maps-Navigate_v10.4.1* | 0 | 0 | 0 |
| *Medscape_v5.0* | 4 | 5 | 3 |
| *Mercado_v9.47.5* | 1 | 3 | 1 |
| *MemeGenerator_v4.557* | 6 | 10 | 3 |
| *Netflix_v7.14.0* | 1 | 2 | 1 |
| *Nova_v6.2.3* | 0 | 0 | 0 |
| *Periscope_v1.24.18.69* | 5 | 8 | 5 |
| *Pinterest_v7.38.0* | 4 | 8 | 3 |
| *Pocket_v6.7.15.7* | 1 | 3 | 1 |
| *Prisma_v3.1.4.381* | 4 | 9 | 3 |
| *Psiphon_v244* | 1 | 3 | 0 |
| *Quora_v2.5.13* | 0 | 2 | 0 |
| *SHAREit_v4.7.48* | 1 | 5 | 1 |
| *Snapchat_10.45.7.0* | 0 | 4 | 0 |
| *Skype_v8.55.0.123* | 0 | 3 | 0 |
| *Steam_v2.3.11* | 0 | 0 | 0 |
| *Tango_v6.16.240967* | 2 | 12 | 2 |
| *TikTok_v9.6.0* | 3 | 10 | 3 |
| *Twilight_v10.3* | 0 | 0 | 0 |
| *Twitter_v7.93.2* | 1 | 2 | 1 |
| *UCBrowser_v12.11.3.1204* | 2 | 5 | 2 |
| *Viber_v12.1.0.11* | 2 | 10 | 2 |
| *Wattpad_v8.45.0* | 4 | 14 | 4 |
| *Waze_v4.52.3.4* | 0 | 1 | 0 |
| *WeChat_v6.7.3* | 0 | 0 | 0 |
| *Windfinder_v3.8.2* | 0 | 4 | 0 |
| *WWE_v4.0.24* | 2 | 3 | 2 |
| *Xbox_v1910.1023.1856* | 1 | 2 | 1 |
| *Yelp_v10.28.0* | 3 | 10 | 3 |
| *YouNow_v15.9.7* | 4 | 10 | 4 |
| Aggregate | 80 | 248 | 70 |

where the entities are the third-parties (e.g., those employ advertisers) of the third-parties (e.g., advertisers) of the app. The comparison results are summarized in Table IV.

As shown in Table IV, AppCensus detects a total of 80 data flows that third-parties collect user privacy, while PTPDroid detects 248 such data flows. There are 70 data flows that are detected by both tools. Of all 48 apps, PTPDroid detects more data flows in 34 of them and the same data flows in eight of them. However, for the remaining six apps, AppCensus detects more data flows than PTPDroid does. For the 10 data flows that PTPDroid fails to detect, we analyze their source code and find the reasons which are summarized in Table V: Among the 10 undetected data flows, three are due to code obfuscation, five are because the third-parties request private information directly from the app server side, and the last two impute to the inherent limitations of our static analysis. The first two reasons are further explained as follows.

**Code obfuscation**. Some undetected data flows are due to

TABLE V
DETAILS OF THE DATA FLOWS UNDETECTED BY PTPDROID

| App | Code obfuscation | Request to server | Others |
|---|---|---|---|
| *Dailymotion* | 1 | 1 | 1 |
| *Medscape* | | 1 | |
| *Memegenerator* | | 2 | 1 |
| *Pinterest* | 1 | | |
| *Prisma* | 1 | | |
| *Psiphon* | | 1 | |
| Aggregate | 3 | 5 | 2 |

TABLE VI
THE DISTRIBUTION OF DIFFERENT DISCLOSURES TO THIRD-PARTIES ACROSS APPS AND DATA FLOWS

| | Clear | Vague | Omitted | Denied |
|---|---|---|---|---|
| Apps | 38 | 719 | 178 | 12 |
| Data flows | 324 | 2,389 | 605 | 33 |

code obfuscation, which is used to prevent app decompilation by rewriting various elements of the app code to meaningless strings. For example, in app *Pinterest*, the third-party library *Crashlytics* is obfuscated into $com.b.a.a$. As a consequence, the relevant data flow is not pinpointed by PTPDroid. The same situation occurs to *Prisma* and *Dailymotion*, where the third-party libraries *Amplitude* and *Branch* are obfuscated into $o0Q10$ and $a.a$, respectively.

**Request to server**. As aforesaid, for the APIs that request to the app server, we cannot get the data type directly from the API names. In this case, we search for the variables which can map to data types from the arguments (in)directly passed to the source methods. Nonetheless, this does not always work. For example, in app *Medscape*, we do find the data flow that the third-party *Appboy* requests information from the server, but we do not find the arguments that can be mapped to any data type. Consequently, we have to discard the data flow.

The experimental results indicate that although our static analysis misses some data flows due to reasons like code obfuscation, PTPDroid still detects 168 more data flows than PoliCheck (AppCensus) does, because the latter is limited to the observed exploration (major cause) and does not consider some data types (minor cause). These results justify the necessity ofPTPDroid. Although PTPDroid shows advantages, it and PoliCheck are complementary in practice.

Additionally, we also compare our entity-sensitive consistency model with the entity-insensitive consistency model [12], [13], [14], [15]. The result shows that for the 248 data flows detected by PTPDroid, under the entity-sensitive consistency model, 45 of them are clear disclosures, 198 are vague disclosures, and 5 are omitted disclosures. Whereas if the entity-insensitive consistency model is adopted, 215 are clear disclosures and 33 are vague disclosures, with 68.5% of the data flows are incorrectly classified. The experimental results demonstrate that the entity-sensitive consistency model more precise than the entity-insensitive consistency model.

TABLE VII
THE TOP-10 THIRD-PARTY ENTITIES IN 1,000 APPS

| Entity | Category | #Apps | Proportion |
|---|---|---|---|
| *Facebook* | Advertiser | 322 | 32.2% |
| *Admob* | Advertiser | 214 | 21.4% |
| *Unity* | Advertiser | 147 | 14.7% |
| *Moat* | Analytics | 113 | 11.3% |
| *Crashlytics* | Analytics | 86 | 8.6% |
| *Vungle* | Advertiser | 74 | 7.4% |
| *Adcolony* | Advertiser | 68 | 6.8% |
| *AppsFlyer* | Analytics | 57 | 5.7% |
| *Adjust* | Analytics | 57 | 5.7% |
| *Applovin* | Advertiser | 53 | 5.3% |

### C. Empirical Study on 1,000 Commercial Apps

To investigate how severe violated user privacy disclosures to third-parties are in practice, we further apply PTPDroid to 1,000 commercial Android apps randomly selected and downloaded from the app store of Google Play. These 1,000 apps cover various categories and functionalities, and with sufficient downloads. The privacy policies of these 1,000 apps are all in English. The empirical results are summarized in Table VI and we gain the following primary findings.

**Primary results**. For a total of 3,351 data flows to third-parties across 947 apps, there are 11 distinct data types shared with 88 different third-party entities. Overall, device identifiers are the most frequently collected data type, which accounts for 87.5% (2,931/3,351) of data flows to 93.2% (82/88) of unique third-parties. Locations are at the second place, which account for 9.2% (308/3,351) of data flows. For the data flows, *Facebook* is the most common advertiser recipient, which accounts for 13.7% (458/3,351) of data flows involving four unique data types and across 32.3% (323/1,000) of apps. *Moat* is the most popular analytics providers, which accounts for 6.1% (205/3,351) of data flows across 11.3% (113/1,000) of apps. The top-10 third-party entities (and their statistics) with which the 1,000 apps frequently share user data is summarized in Table VII. Surprisingly, we find that 19% (190/1,000) of apps contain at least one omitted disclosure or denied disclosure. The remainder of this section discusses the concrete findings of our empirical study.

> **Finding 1**: Only 3.8% (38/1,000) of apps detail all cooperating third-party entities in their privacy policies. And only 9.7% (324/3,351) of data flows to third-parties are explicitly declared by the declarations in the privacy policies that pinpoint the exact transmitted data types and exact third-party entities.

**Clear disclosures**. In total, only 324 data flows to third-parties are clear disclosures. Of these, 144 data flows detected from 83 apps are just used as examples declared in the privacy policies. For instance, in app *Life360*[5], its privacy policy shows

[5]https://play.google.com/store/apps/details?id=com.subsplash.thechurchapp.life360church

only one example (*Arity*) that collects user private information: "One of the 3rd parties who we share data with is Arity. Arity collects the following via the app and/or your mobile device. Some of the information they may collect includes: Geolocation and movement data; Mobile device information and application analytics, including IP address and device identifiers." Although a clear disclosure can be derived from this declaration, our detected data flows to the third-parties *Amplitude* and *Appsflyer* are vague disclosures. Consequently, such declarations are not entirely clear. This indicates that the app vendors (developers) do not realize the necessity or importance of enumerating all third-party entities in their privacy policies.

**Finding 2**: 71.9% (719/1,000) of apps contain at least one vague disclosure. Moreover, 8.1% (81/1,000) of apps use the term "third-party" generally to refer to all third-party entities.

**Vague disclosures**. A total of 2,389 data flows (declarations) in the privacy policies use vague terms to blur the specific third-party entities, the concrete data types, or both. Among the 2,389 data flows, 57.3% (1,369/2,389) are disclosed using vague terms to refer to concrete data types, and 42.7% (1,020/2,389) are disclosed using vague terms to refer to specific entities. For vague representations of third-party entities, 88.2% (634/719) are represented by the functionalities (services) provided by the third-parties, such as advertisement and data analysis. 9.4% (225/2,389) of data flows replace some specific third-party entities with the term "third-party", which raises the concern that these apps do not comply with the mandate of GDPR on specificity of disclosures. For example, the app *iHeartRadio*[6] shares device identifiers to *Amazonad*, *Appboy*, and *Adobe*, but its privacy policy only states: "We may share your personal information with our external third-party service providers.".

For some of apps, the private information collected by the third-parties can be obtained from some links in privacy policies, but is not directly declared in privacy policies. For example, a part of privacy policy of the app *Evermatch*[7] is shown in Figure 4. Despite the fact that there is no clear legal issues to use third-party links in privacy policies, users need to browse several pages to get the privacy data collected by third-parties (many users may give up), which can be regarded as vague/hidden disclosures from users' perspective. Even for the users who browse the third-party links, they are not sure whether all or partial data mentioned in the links are collected. We note that most of apps with vague disclosures share two or fewer unique data types. As a result, it is feasible for the developers to explicitly disclose the exact data types shared with the concrete entities because the developers who use the third-party libraries are likely and should be aware the data collected by the libraries.

[6]https://play.google.com/store/apps/details?id=com.thisisglobal.player.heart
[7]https://sites.google.com/view/evermatch

b. Third-party advertising services. Evermatch uses

Facebook - https://www.facebook.com/policy.php

Google Admob - https://policies.google.com/privacy

AppLovin - https://www.applovin.com/privacy/

MyTarget - https://target.my.com/optout/

MoPub - https://www.mopub.com/legal/privacy/

UnityAds - https://unity3d.com/legal/privacy-policy

Fig. 4. Part of privacy policy of *Evermatch*.

**Finding 3**: 17.8% (178/1,000) of apps omit to declare the user private information to third-parties in their privacy policies. Such apps may also have confusing sentences in privacy policies.

**Omitted disclosures**. Of the 605 omitted disclosures, the most frequently omitted data type is device identifiers (549/605 = 90.7%) and the most frequently omitted third-party entities is *Unity* (54/605 = 8.9%). For instance, the app *Decision Roulette*[8] shares device identifiers to *Appboy* and *Adwizz*, but this is not declared in its privacy policy. The large number of omitted disclosures to third-parties reflect that either the developers do not know the data types collected by the third-party libraries they embed into their apps or they do not think omitted disclosure is a serious problem.

**Finding 4**: 1.2% (12/1,000) of apps' privacy policies involve denied disclosures to third-parties. And, such privacy policies are often difficult to understand.

**Denied disclosures**. A total of 12 apps contain denied disclosures to third-parties in their privacy policies, and these apps contribute to only 1.0% (33/3,351) of the data flows. This shows denied disclosures are not so severe in practice. One such example is a children's app called "*FamilyAlbum - Easy Photo & Video Sharing*"[9]. We detect the data flows in its code that it shares device identifiers to *Facebook* and *Appsflyer*, while its privacy policy explicitly states: "Even if the statistical data is anonymized, we will not sell or provide the user's data to a third party." Obviously, its privacy policy contains denied disclosures. On the other hand, its privacy policy also states that the first-party may collect device identifiers, and if we do not differentiate the entities, these data flows will be misclassified as clear disclosures, which illustrates the importance of considering third-party entities. In addition, most privacy policies with denied disclosures are difficult to understand, which indicates that many developers may not take privacy policies seriously.

[8]https://play.google.com/store/apps/details?id=com.fireshooters.roulette
[9]https://mitene.us

TABLE VIII
VENDOR CONFIRMATION OR CORRECTION

| App | Developer confirmation or the corrected privacy policy |
|---|---|
| *Breeze* | We do use 3rd party services and 3rd party advertising in our apps. |
| *Familyalbum* | We cooperate with facebook to provide advertising service. |
| *Trainman* | https://www.trainman.in/static/privacy-policy.html |
| *DStv* | https://www.dstv.com/en-za/legal/privacy-cookie-notice/ |
| *Nyari Kopi* | https://dashboard.createappasia.com/users/appPrivacyPolicy |
| *BetterSleep* | https://www.ipnos.com/privacy-policy/ |
| *Fitbit* | https://www.fitbit.com/global/us/legal/privacy-policy |
| *iHealth* | https://cdn.ihealthlabs.com/policy/iHealth-Privacy-Policy.html |

**Finding 5**: Only 2.3% (23/1,000) of apps may not share user privacy data with third-parties. In addition, 3.0% (30/1,000) of apps state third-party sharing in their privacy policies, yet we do not detect such data flows.

**No third-party sharing**. Altogether, only 53 apps of the 1,000 apps do not involve any data flows to third-parties in the observed client-side behavior. 44.2% (23/53) of those apps whose privacy policies do not state user privacy sharing to third-parties. Therefore, assuming that their privacy policy accurately reflects both the client-side and server-side behaviors, only 23 apps do not share data with third-parties. While the other 56.6% (30/53) of apps do not contain any data flows to third-parties in the observed client-side behavior, their privacy policies contain the declarations about third-party sharing. This could be due to the limitation of the static analysis, for example, code obfuscation and the fact that user private information can be shared by the server side.

**Vendor feedback**. For those apps whose privacy policies involve violated disclosures, we try to contact their vendors to verify our findings. We get a few confirmations and some of them have already corrected their privacy policies recently (their previous privacy policies can be found in our data sets), which are summarized in Table VIII. Frankly, it is uneasy to get vendors' confirmation because this is related to whether they disobey the laws concerned, and most vendors are reluctant to answer positively.

**Summary**. Our empirical study reveals that while sharing user private information with third-party entities has become a common practice in Android ecosystems, the app vendors have neither paid enough attention nor noticed the importance of clearly disclosing this in app privacy policies.

### D. Threats to Validity and Limitations

**Internal validity**. The major threat to the first experiment answering RQ1 comes from the ground truth used for the evaluation, because the privacy policies of the 24 popular Android apps might not be perfect or not be necessarily consistent with the app code. Our conclusions are based on partial observations that carry a probability of an error, and *Bayesian data analysis* may help to reduce the threat. In addition, for the construction of third-party ontology, we only consider the popular third-parties and the third-parties involved in our first two experiments, which could affect the results of the empirical study on 1,000 commercial apps.

**External validity**. To answer RQ1 and RQ2, only 24 and 48 apps are used, respectively. And, our empirical study is based on the 1,000 real-world commercial apps. Since there are more than three million apps available on Google Play, the number of apps for the empirical analysis is small. Fortunately, both the 48 and 1,000 apps are randomly selected, and are with different functionalities and categories, we believe that the results are generalized to a larger population of apps.

**Limitations**. The reported experimental/empirical results are just an under-approximation of the reality due to the following limitations of our work.

1) Our analysis is limited to predefined third-parties, while other actors might be harvesting personal data this way.
2) Our analysis is limited to in-app flow analysis, but app back-ends operating in data centers may also share sensitive information with third-parties.
3) PTPDroid does not consider user input data from app GUI [15], which may also involve user privacy.
4) The precision and recall of PTPDroid is far from perfect. For instance, we use keyword matching to map API methods to ontology phrases, even if topic modelling is used, the matching is affected by code obfuscation.

## V. RELATED WORK

In this section, we briefly review several strands of researches related to our work.

**Privacy policy analysis**. Various studies concentrate on extracting the useful information from the privacy policy to make it easier for users to understand. Privee [26] employs natural language processing for deriving a set of binary questions from privacy policies to increase privacy transparency. Hermes [27] utilizes topic modeling to reduce the ambiguities in privacy policies to facilitate users. PrivacyCheck [28] and Polisis [29] analyze the privacy policies to automatically extract their graphical summaries representing what and how information is used. The difference is that the former leverages data mining techniques, whereas the latter uses deep learning. PI-Extract [30] trains a large data set to improve the users' reading comprehension of policy texts.

Analyzing the usability of privacy policies [31], [32], [33] is another well-researched area. Some studies [34], [35] find that the disclosures are increasing while readability is decreasing in privacy policies over the years. Some surveys [36], [37] investigate privacy policy shortcomings in specific sectors, such as healthcare and finance. A recent study [38] finds that some privacy policies fail to disclose the presence of common tracking technologies and third-parties. All of these studies show that most of the privacy policies on the market are more or less problematic.

Other researches focus on finding conflicting declarations in privacy policies. Breaux et al. [39] propose a formal language Eddy to analyze the privacy policy of multi-tier systems to find the conflicts between the policies regarding data collection, usage, retention and transfer. PolicyLint [20] considers the entities and negative sentiment declarations, which can better pinpoint conflicts.

**Privacy policy generation**. From the perspective of developers, it is a challenging task to write a privacy policy that correctly reflects an app's privacy practices and keeping the policy up-to-date as the app evolves over time. Therefore, some studies concentrate on generating privacy policies automatically to facilitate developers. For example, Liccardi et al. [40] develop a framework to generate policies based on questionnaires filled by the developers. PAGE [41] is a plugin for the Eclipse IDE that can be used to create privacy policies based on questionnaires during the development process. However, purely questionnaire-based generators can lead to inaccurate representations of an app's privacy practices if the questions are not answered accurately or completely.

There is some work [42], [43] that leverages the source code to generate privacy policy. AutoPPG [43] extracts code from Android apps to create short text snippets for the corresponding app behavior. A corpus of policies collected in the wild is used for generating snippets of the form (*subject, verb, object*). PrivacyFlash Pro [42] creates legally compliant policies by mapping app analysis results and questionnaire answers to standardized legal templates.

Some other work uses code analysis to generate templates with app privacy settings [44] and security descriptions [45]. Although these templates and descriptions are not directly used as privacy policies, they can be helpful for users to understand and adjust the privacy and security settings of an app.

**Flow-to-policy consistency**. Some early studies focus on analyzing policy-code consistency of specific app categories, such as those designed for families or children [46], [47]. While these approaches potentially work for a category of apps, they are severely limited in the analysis accuracy for broader categories. A deal of recent work [12], [13], [14] uses similar approaches to detect privacy policy violations in Android apps based on Android's API (application program interface) calls. These approaches are useful in identifying leaks when the API calls collect personal information from a mobile device. GUILeak [15] extends the taint sources to include the sensitive data entered by users through an app's user interface. However, since it is too costly to build the privacy ontology for each app category, GUILeak only considers three app categories: finance, health, and dating. Yu et al. [48] develop an approach for detecting policy violations with more advanced data flow models, which performs the cross-verification among privacy policy, byte code, description, and permissions. Unfortunately, the above effort does not differentiate first-party entities and third-party entities that collect user privacy, which may falsely classify violated third-party flows as clear disclosures.

PoliCheck [18], the closest work to ours, is an entity-sensitive tool to detect inconsistency between privacy policies and Android apps. PoliCheck makes advancement over prior work by considering DNS domains of entities (first-parties and third-parties) for comprehensive analysis. However, PoliCheck considers only a small number of data types and may miss a large number of data flows due to the limitation of dynamic analysis. In contrast to PoliCheck, PTPDroid is based on static analysis and takes all of the data types accessed by the sensitive API methods into account. IoTPrivComp [49] is another entity-sensitive work, but it only focuses on flow-to-policy consistency of IoT apps.

**Taint analysis**. PTPDroid leverages data flow analysis to check whether the private information accessed from a certain method transmits to a third-party. PTPDroid is based on FlowDroid [21], the state-of-art static taint analysis tool for Android apps. Other static analysis techniques include CHEX [50], IccTA [51], VulHunter [52], and EdgeMiner [53]; EdgeMiner is used in some work on consistency analysis [12], [13], [14]. Dynamic data flow analysis techniques, such as TaintDroid [5], AppCensus [19], and CopperDroid [54], detect the privacy leak at runtime. By using Monkey [55], in addition to differentiating first-parties and third-parties data flows, Guamán et al. [56] also consider the countries where the recipient servers are located. Based on this, they conduct GDPR compliance assessment for cross-border private data transfers in Android apps. In general, static analysis can obtain more comprehensive data flows, while dynamic analysis can ensure the realness of the detected data flows.

## VI. Conclusions

For most Android users, privacy threats from mobile apps are arguably a greater risk than malware. Recently, many approaches focus on checking the consistency between the app code and the privacy policy. However, prior work seldom differentiates the entities collecting the data. To this end, we propose PTPDroid, an entity-sensitive flow-to-policy consistency checking technique to detect violated privacy disclosures to third-parties. PTPDroid achieves this by utilizing static analysis to obtain data flows in the app code and using natural language processing to extract declarations in the privacy policy. Our experimental evaluation on a benchmark of 24 popular commercial Android apps demonstrates the effectiveness and efficiency of PTPDroid, and the comparison experiment on 48 Android apps proves the advantage of PTPDroid. Moreover, our empirical study on 1,000 commercial Android apps based on PTPDroid shows that vague, omitted, and denied user privacy disclosures to third-parties are prevailing in reality, and we suggest app vendors to take this matter seriously.

The future work can consider to automatically synthesize the privacy disclosures to third-parties, and extend our approach to IoT apps by taking more data types into account.

## Data Availability

Our open-source tool PTPDroid can be found in GitHub[10]. All apps used for answering RQ1-RQ3 in our experiments and empirical study are publicly available[11].


## Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant No. 61761136003.


[10]https://github.com/wsong-nj/PTPDroid
[11]https://doi.org/10.5281/zenodo.5442986

## REFERENCES


[1] Statista, "Google play statistics," 2021. [Online]. Available: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store

[2] IDC, "International data corporation (idc) smartphone os market share 2021 q1," 2021. [Online]. Available: https://www.idc.com/promo/smartphone-market-share

[3] L. Zhao, N. W. H. Chan, S. J. Yang, and R. W. Melton, "Privacy sensitive resource access monitoring for android systems," in *Proceedings of the 24th International Conference on Computer Communication and Networks, ICCCN'15, Las Vegas, NV, USA, August 3-6*. IEEE, 2015, pp. 1–6.

[4] S. Yang, Z. Zeng, and W. Song, "Permdroid: automatically testing permission-related behaviour of android applications," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA'22, Virtual Event, South Korea, July 18 - 22*. ACM, 2022, pp. 593–604.

[5] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. D. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, pp. 5:1–5:29, 2014.

[6] J. Reardon, Á. Feal, P. Wijesekera, A. E. B. On, N. Vallina-Rodriguez, and S. Egelman, "50 ways to leak your data: An exploration of apps' circumvention of the android permissions system," in *Proceedings of the 28th USENIX Security Symposium, USENIX Security'19, Santa Clara, CA, USA, August 14-16*. USENIX Association, 2019, pp. 603–620.

[7] S. Matsumoto and K. Sakurai, "A proposal for the privacy leakage verification tool for android application developers," in *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication, ICUIMC '13, Kota Kinabalu, Malaysia - January 17 - 19*. ACM, 2013, p. 54.

[8] P. G. Kelley, L. F. Cranor, and N. M. Sadeh, "Privacy as part of the app decision-making process," in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, CHI'13, Paris, France, April 27 - May 2*. ACM, 2013, pp. 3393–3402.

[9] CCPA, "California consumer privacy act." 2018. [Online]. Available: https://oag.ca.gov/privacy/ccpa

[10] FTC, "Ftc path case helps app developers stay on the right, er, path," 2013. [Online]. Available: https://goo.gl/JKgJT4

[11] ——, "In the matter of snapchat," 2014. [Online]. Available: https://www.ftc.gov/enforcement/cases-proceedings/132-3078/snapchat-inc-matter

[12] R. Slavin, X. Wang, M. B. Hosseini, J. Hester, R. Krishnan, J. Bhatia, T. D. Breaux, and J. Niu, "Toward a framework for detecting privacy policy violations in android application code," in *Proceedings of the 38th International Conference on Software Engineering, ICSE'16, Austin, TX, USA, May 14-22*. ACM, 2016, pp. 25–36.

[13] S. Zimmeck, Z. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, S. Wilson, N. M. Sadeh, S. M. Bellovin, and J. R. Reidenberg, "Automated analysis of privacy requirements for mobile apps," in *Proceedings of the 24th Annual Network and Distributed System Security Symposium, NDSS'17, San Diego, California, USA, February 26 - March 1*. The Internet Society, 2017.

[14] L. Yu, X. Luo, X. Liu, and T. Zhang, "Can we trust the privacy policies of android apps?" in *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN'16, Toulouse, France, June 28 - July 1*. IEEE Computer Society, 2016, pp. 538–549.

[15] X. Wang, X. Qin, M. B. Hosseini, R. Slavin, T. D. Breaux, and J. Niu, "Guileak: tracing privacy policy claims on user input data for android applications," in *Proceedings of the 40th International Conference on Software Engineering, ICSE'18, Gothenburg, Sweden, May 27 - June 03*. ACM, 2018, pp. 37–47.

[16] H. Lu, L. Xing, Y. Xiao, Y. Zhang, X. Liao, X. Wang, and X. Wang, "Demystifying resource management risks in emerging mobile app-in-app ecosystems," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS'20, Virtual Event, USA, November 9-13*. ACM, 2020, pp. 569–585.

[17] GDPR, "The eu general data protection regulation," 2018. [Online]. Available: https://eugdpr.org

[18] B. Andow, S. Y. Mahmud, J. Whitaker, W. Enck, B. Reaves, K. Singh, and S. Egelman, "Actions speak louder than words: Entity-sensitive privacy policy and data flow analysis with policheck," in *Proceedings of the 29th USENIX Security Symposium, USENIX Security'20, August 12-14*. USENIX Association, 2020, pp. 985–1002.

[19] AppCensus, "Appsearch," 2019. [Online]. Available: https://search.appcensus.io/

[20] B. Andow, S. Y. Mahmud, W. Wang, J. Whitaker, W. Enck, B. Reaves, K. Singh, and T. Xie, "Policylint: Investigating internal privacy policy contradictions on google play," in *Proceedings of the 28th USENIX Security Symposium, USENIX Security '19, Santa Clara, CA, USA, August 14-16*. USENIX Association, 2019, pp. 585–602.

[21] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, and P. D. McDaniel, "Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'14, Edinburgh, United Kingdom - June 09 - 11*. ACM, 2014, pp. 259–269.

[22] T. Gruber, "Ontology," in *Encyclopedia of Database Systems, Second Edition*, L. Liu and M. T. Özsu, Eds. Springer, 2018.

[23] M. B. Hosseini, T. D. Breaux, R. Slavin, J. Niu, and X. Wang, "Analyzing privacy policies through syntax-driven semantic analysis of information types," *Inf. Softw. Technol.*, vol. 138, p. 106608, 2021.

[24] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: analyzing the android permission specification," in *Proceedings of the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18*. ACM, 2012, pp. 217–228.

[25] S. Rasthofer, S. Arzt, and E. Bodden, "A machine-learning approach for classifying and categorizing android sources and sinks," in *Proceedings of the 21st Annual Network and Distributed System Security Symposium, NDSS'14, San Diego, California, USA, February 23-26*. The Internet Society, 2014.

[26] S. Zimmeck and S. M. Bellovin, "Privee: An architecture for automatically analyzing web privacy policies," in *Proceedings of the 23rd USENIX Security Symposium, USENIX Security'14, San Diego, CA, USA, August 20-22*. USENIX Association, 2014, pp. 1–16.

[27] J. W. Stamey and R. A. Rossi, "Automatically identifying relations in privacy policies," in *Proceedings of the 27th Annual International Conference on Design of Communication, SIGDOC '09, Bloomington, Indiana, USA, October 5-7*. ACM, 2009, pp. 233–238.

[28] R. N. Zaeem, R. L. German, and K. S. Barber, "Privacycheck: Automatic summarization of privacy policies using data mining," *ACM Trans. Internet Techn.*, vol. 18, no. 4, pp. 53:1–53:18, 2018.

[29] H. Harkous, K. Fawaz, R. Lebret, F. Schaub, K. G. Shin, and K. Aberer, "Polisis: Automated analysis and presentation of privacy policies using deep learning," in *Proceedings of the 27th USENIX Security Symposium, USENIX Security'18, Baltimore, MD, USA, August 15-17*. USENIX Association, 2018, pp. 531–548.

[30] D. Bui, K. G. Shin, J.-M. Choi, and J. Shin, "Automated extraction and presentation of data practices in privacy policies," *Proc. Priv. Enhancing Technol.*, vol. 2021, no. 2, pp. 88–110, 2021.

[31] F. Liu, R. Ramanath, N. M. Sadeh, and N. A. Smith, "A step towards usable privacy policy: Automatic alignment of privacy statements," in *Proceedings of the 25th International Conference on Computational Linguistics, COLING'14, August 23-29, Dublin, Ireland*. ACL, 2014, pp. 884–894.

[32] R. Ramanath, F. Liu, N. M. Sadeh, and N. A. Smith, "Unsupervised alignment of privacy policies using hidden markov models," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL'14, June 22-27, Baltimore, MD, USA, Volume 2: Short Papers*. The Association for Computer Linguistics, 2014, pp. 605–610.

[33] E. Costante, Y. Sun, M. Petkovic, and J. den Hartog, "A machine learning solution to assess privacy policy completeness: (short paper)," in *Proceedings of the 11th annual ACM Workshop on Privacy in the Electronic Society, WPES '12, Raleigh, NC, USA, October 15*. ACM, 2012, pp. 91–96.

[34] G. R. Milne and M. J. Culnan, "Using the content of online privacy notices to inform public policy," *Inf. Soc.*, vol. 18, no. 5, pp. 345–359, 2002.

[35] A. I. Antón, J. B. Earp, M. W. Vail, N. Jain, C. M. Gheen, and J. M. Frink, "Hipaa's effect on web site privacy policies," *IEEE Secur. Priv.*, vol. 5, no. 1, pp. 45–52, 2007.

[36] A. Sunyaev, T. Dehling, P. L. Taylor, and K. D. Mandl, "Availability and quality of mobile health app privacy policies," *J. Am. Medical Informatics Assoc.*, vol. 22, no. e1, pp. e28–e33, 2015.

[37] J. D. Bowers, B. Reaves, I. N. Sherman, P. Traynor, and K. R. B. Butler, "Regulators, mount up! analysis of privacy policies for mobile money

services," in *Proceedings of the 13th Symposium on Usable Privacy and Security, SOUPS '17, Santa Clara, CA, USA, July 12-14*. USENIX Association, 2017, pp. 97–114.

[38] R. Amos, G. Acar, E. Lucherini, M. Kshirsagar, A. Narayanan, and J. Mayer, "Privacy policies over time: Curation and analysis of a million-document dataset," in *Proceedings of the Web Conference, WWW'21, Virtual Event / Ljubljana, Slovenia, April 19-23*. ACM / IW3C2, 2021, pp. 2165–2176.

[39] T. D. Breaux, H. Hibshi, and A. Rao, "Eddy, a formal language for specifying and analyzing data flow specifications for conflicting privacy requirements," *Requir. Eng.*, vol. 19, no. 3, pp. 281–307, 2014.

[40] I. Liccardi, M. Bulger, H. Abelson, D. J. Weitzner, and W. E. Mackay, "Can apps play by the COPPA rules?" in *Proceedings of the 12th Annual International Conference on Privacy, Security and Trust, Toronto, ON, Canada, July 23-24*. IEEE Computer Society, 2014, pp. 1–9.

[41] M. Rowan and J. Dehlinger, "Encouraging privacy by design concepts with privacy policy auto-generation in eclipse (page)," in *Proceedings of the 2014 Workshop on Eclipse Technology eXchange, ETX'14, Portland, OR, USA, October 20 - 24*. ACM, 2014, pp. 9–14.

[42] S. Zimmeck, R. Goldstein, and D. Baraka, "Privacyflash pro: Automating privacy policy generation for mobile apps," in *Proceedings of the 28th Annual Network and Distributed System Security Symposium, NDSS'21, virtually, February 21-25*. The Internet Society, 2021.

[43] L. Yu, T. Zhang, X. Luo, L. Xue, and H. Chang, "Toward automatically generating privacy policy for android apps," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 4, pp. 865–880, 2017.

[44] X. Chen, H. Huang, S. Zhu, Q. Li, and Q. Guan, "Sweetdroid: Toward a context-sensitive privacy policy enforcement framework for android OS," in *Proceedings of the on Workshop on Privacy in the Electronic Society, Dallas, TX, USA, October 30 - November 3*. ACM, 2017, pp. 75–86.

[45] M. Zhang, Y. Duan, Q. Feng, and H. Yin, "Towards automatic generation of security-centric descriptions for android apps," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS'15, Denver, CO, USA, October 12-16*. ACM, 2015, pp. 518–529.

[46] R. Tang, D. Shao, S. Bressan, and P. Valduriez, "What you pay for is what you get," in *Proceedings of the 24th International Conference on Database and Expert Systems Applications, DEXA'13, Prague, Czech Republic, August 26-29, Part II*, vol. 8056. Springer, 2013, pp. 395–409.

[47] I. Reyes, P. Wijesekera, J. Reardon, A. E. B. On, A. Razaghpanah, N. Vallina-Rodriguez, and S. Egelman, ""won't somebody think of the children?" examining COPPA compliance at scale," *Proc. Priv. Enhancing Technol.*, vol. 2018, no. 3, pp. 63–83, 2018.

[48] L. Yu, X. Luo, C. Qian, S. Wang, and H. K. N. Leung, "Enhancing the description-to-behavior fidelity in android apps with privacy policy," *IEEE Trans. Software Eng.*, vol. 44, no. 9, pp. 834–854, 2018.

[49] J. Ahmad, F. Li, and B. Luo, "Iotprivcomp: A measurement study of privacy compliance in iot apps," in *Proceedings of the 27th European Symposium on Research in Computer Security, ESORICS'22, Copenhagen, Denmark, September 26-30, Part II*, ser. Lecture Notes in Computer Science, vol. 13555. Springer, 2022, pp. 589–609.

[50] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "CHEX: statically vetting android apps for component hijacking vulnerabilities," in *Proceedings of the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18*. ACM, 2012, pp. 229–240.

[51] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. L. Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. D. McDaniel, "Iccta: Detecting inter-component privacy leaks in android apps," in *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering, ICSE'15, Florence, Italy, May 16-24, Volume 1*. IEEE Computer Society, 2015, pp. 280–291.

[52] C. Qian, X. Luo, Y. Le, and G. Gu, "Vulhunter: Toward discovering vulnerabilities in android applications," *IEEE Micro*, vol. 35, no. 1, pp. 44–53, 2015.

[53] Y. Cao, Y. Fratantonio, A. Bianchi, M. Egele, C. Kruegel, G. Vigna, and Y. Chen, "Edgeminer: Automatically detecting implicit control flow transitions through the android framework," in *Proceedings of the 22nd Annual Network and Distributed System Security Symposium, NDSS'15, San Diego, California, USA, February 8-11*. The Internet Society, 2015.

[54] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "Copperdroid: Automatic reconstruction of android malware behaviors," in *Proceedings of the 22nd Annual Network and Distributed System Security Symposium, NDSS'15, San Diego, California, USA, February 8-11*. The Internet Society, 2015.

[55] Google, "The monkey ui android testing tool," 2020. [Online]. Available: http://developer.android.com/tools/help/monkey.html

[56] D. S. Guamán, J. M. del Álamo, and J. C. Caiza, "GDPR compliance assessment for cross-border personal data transfers in android apps," *IEEE Access*, vol. 9, pp. 15 961–15 982, 2021.