Withdrawing is believing? Detecting Inconsistencies between Withdrawal Choices and Third-party Data Collections in Mobile Apps

Xiaolin Du*¶, Zhemin Yang*¶, Jiapeng Lin*, Yinzhi Cao†, and Min Yang*
* Fudan University, {xldu20, yangzhemin, linjp23, m_yang}@fudan.edu.cn
† Johns Hopkins University, yzcao@cs.jhu.edu
¶ Co-first authors

***Abstract*—Popular privacy regulations such as General Data Protection Regulation (GDPR) often allow consumers to withdraw from providing data, e.g., the famous right to opt-out. Modern computer software, e.g., mobile applications (apps), often provide withdrawal interfaces, which stop data collection—e.g., from third-party ads and analytics libraries—to respect users' withdrawal decisions. While such interfaces are marked as "withdrawal", their correlated withdrawal decisions are often inconsistent with the apps' actual data collection behavior, especially from third parties, which is defined as withdrawal inconsistency in the paper.**

Prior works have either studied website withdrawal inconsistency or privacy leaks of mobile apps. However, the mobile withdrawal inconsistency problem is different yet more complex than those in websites due to the diversity in mobile withdrawal interface and the variety of private information. At the same time, none of the existing works detecting privacy leaks of mobile apps understand users' withdrawal decisions let alone correlate them with withdrawal behaviors.

In this paper, we design and implement a novel approach, called MOWCHECKER, to detect mobile apps' inconsistencies in third-party data collection. The key insight is that withdrawal choices should have either a control-flow dependency on personal information flow or a data-flow dependency on withdrawal APIs provided by third-party data collection libraries. Our evaluation of MOWCHECKER on real-world Android apps reveals 157 manually-confirmed, zero-day withdrawal inconsistencies. We have responsibly reported them to app developers and received 23 responses with two being fixed.

1. Introduction

Popular privacy regulations, such as General Data Protection Regulation (GDPR) [1] and California Consumer Privacy Act (CCPA) [2], often allow consumers to exclude themselves—or called withdraw—from providing personal information, e.g., following the famous right to opt-out. Software communities have hence reacted accordingly: For example, mobile applications (apps) often provide withdrawal interfaces to collect users' choices to withdraw. Then, mobile apps will stop third-party data collection, e.g., those powered by ads or analytics libraries like Firebase Analytics [3], to respect users' withdrawal decisions.

While intuitively simple, it is often complicated yet challenging to withdraw from data collection in practice especially when third-party libraries are involved. Specifically, after receiving users' withdrawal choices, mobile apps need to either stop third-party data collection themselves or inform third-party libraries to stop collection. Many mobile apps may forget to perform such stopping or informing third-party, leading to an inconsistency in what the user believes about withdrawing and what the app actually performs. Such inconsistency is thus defined as third-party withdrawal inconsistency of mobile apps, or for short *withdrawal inconsistency* without ambiguity in the paper.

Prior works have studied withdrawal inconsistencies and their implications on privacy. On one hand, one popular research direction is to measure withdrawal inconsistencies for websites. For example, Bui et al. [4] show that online trackers embedded as part of websites may exhibit data practices inconsistent with the host websites' stated policies. Rather than focusing on inconsistencies, researchers have also studied websites' withdrawal choices (e.g., users' perception) [5], [6], [7], [8], [9], [10]. However, mobile withdrawals are different yet more complicated than the counterparts on websites. More specifically, mobile withdrawal choices are challenging to extract because they are often embedded inside multiple user interfaces, e.g., via Android activities. Then, mobile personal data is usually more complex yet diversified involving those collected by mobile sensors as compared with cookies, the usual target on the World Wide Web.

On the other hand, prior works have studied privacy leaks [11], [12], [13], [14], [15], [16], [17], [18], [19], [20] of mobile apps. For example, Pan et al. [13] studied the correlation between user consents and mobile apps' actual privacy behaviors. However, withdrawal behaviors usually involve dedicated interfaces and special third-party libraries. That is, none of the prior works can extract or understand withdrawal decisions made by mobile users and withdrawal behaviors of mobile apps, let alone study the consistency between these two. To sum up, it remains an open problem to study the mobile withdrawal inconsistency problem despite existing research on mobile privacy and website withdrawals.

In this paper, we design and implement a novel approach, called MOWCHECKER,[1] to detect inconsistencies between mobile apps' withdrawal choices and third-party data collections. The key insight of MOWCHECKER is that withdrawal choices made by users should have either a control-flow dependency upon the personal information flow or a data-flow to withdrawal APIs provided by third-party data collection libraries. Therefore, MOWCHECKER detects both withdrawal interfaces and data-flows related to personal information and then makes correlations on control- and data-flow dependencies to determine inconsistencies.

More specifically, MOWCHECKER has three steps: (i) withdrawal interface detection, (ii) personal information flow detection, and (iii) inconsistency check. First, MOWCHECKER performs application layout analysis and semantic checking to identify withdrawal options in mobile apps. Second, MOWCHECKER statically discovers data-flows related to personal information collection. Lastly, MOWCHECKER correlates data collection behaviors with withdrawal choices by mining their control and data dependencies. Then, MOWCHECKER performs an ontology-based consistency check to reconcile the different levels of granularity between the withdrawal expectations and actions and report inconsistencies.

Our evaluation shows that MOWCHECKER effectively detects withdrawal inconsistencies with an overall precision of 82.86% and a recall of 90.63%. We also evaluate MOWCHECKER on 25,725 most popular Android apps and 50,000 randomly selected long-tail apps collected from the Google Play Store. MOWCHECKER finds 157 manually-confirmed, zero-day[2] withdrawal inconsistencies, which all lead to violations of privacy regulations. We have responsibly reported all these violations to app developers and so far have received 23 responses and two violations have already been fixed.

Contributions. We summarize our contributions as follows:

- We conducted the first systematic study on the understudied, *withdrawal inconsistency* problem in the mobile ecosystem to reveal the inconsistencies between withdrawal choices and third-party data collections.
- We built an automated yet effective analysis tool, called MOWCHECKER, to detect inconsistencies between mobile apps' withdrawal choices and third-party data collections.
- We manually verified real-world mobile withdrawal inconsistencies identified by MOWCHECKER and notified affected developers with gathered insights from their responses.

2. Overview

In this section, we first describe background related to third-party data collection and withdrawal, and then present our threat model and a motivating example.

1. MOWCHECKER is an abbreviation of Mobile Withdrawal Checker.

2. Borrowing a similar concept for vulnerabilities, "zero-day" here means that the inconsistency is previously unknown to either app developers or the general public.

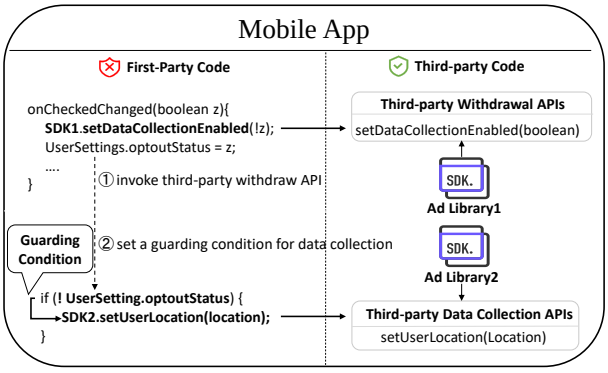


Figure 1: Methods to withdraw from third-party data collection.

2.1. Background

Privacy regulations often allow users to request service providers to stop specific data practices from collecting their personal information, thus called data withdrawal in the paper. Examples of such data withdrawal include but are not limited to the right to opt-out and the right to object. For example, the Article 21 in GDPR supports the "right to object", which enables users to withdraw the processing (*i.e.,* collection, use, disclosure *et al.*) of their personal information for direct marketing. Then, Article 7 allows consumers to revoke consent for the processing of their personal data beyond fulfilling a contractual obligation or business transaction, which is also a form of exercising data withdrawal. Being similar to but slightly different from GDPR, CCPA specifically emphasized withdrawal of the sale[3] of personal information, which allows consumers to request business to stop selling or sharing their personal information to third parties.

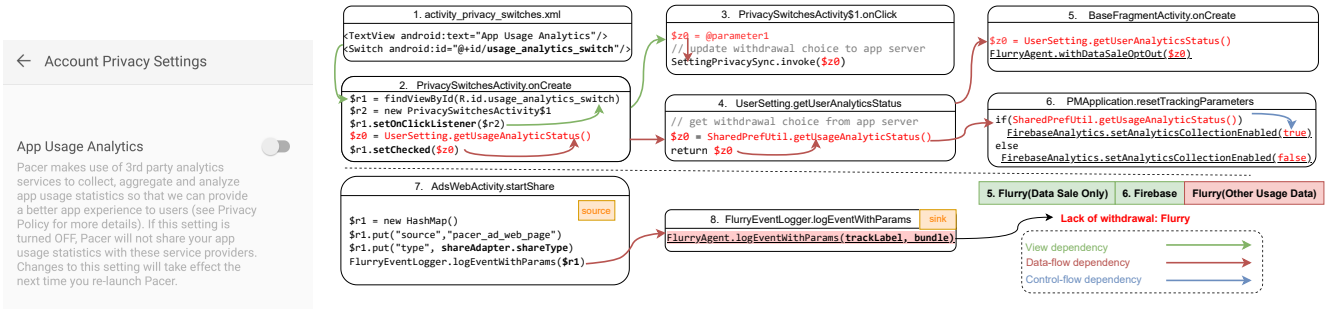
2.2. Third-party Data Collection and Withdrawal

Third-party data collection is the practice of a third-party library like ads and analytics, embedded as part of a mobile app, to collect personal information at mobile devices and send it to the third-party server for aggregation and analysis. There are two types of third-party data collection and corresponding withdrawal methods.

Withdrawal for Host-uncooperated Data Collection. One popular third-party data collection [11], [12], [22], [23] is that the third-party library inherits host app's permission and collects users' sensitive data directly via mobile APIs without further host cooperation beyond library integration. For example, Firebase Analytics [3] will automatically collect some user properties and usage data as long as they are integrated in mobile apps [24].

The withdrawal of such data collection usually requires the host app to inform the third-party library

3. Here the "sale" is defined as *sharing consumer's personal information for monetary or other valuable consideration* [21]



(a) Withdrawal interface of the Pacer app.

(b) Simplified code blocks of the Pacer app.

Figure 2: A motivating example of a zero-day withdrawal inconsistency in Pacer, a popular health tracker app.

via a pre-defined interface. Method ① in Figure 1 shows such an example: The app informs Ad Library1 to stop both the host-uncooperated and host-cooperated data collection via invoking the withdrawal interface `SDK1.setDataCollectionEnabled(false)`. Besides, the app can choose not to initialize the third-party library to stop any data collection by this library.

Withdrawal for Host-cooperated Data Collection. The other third-party data collection method is that the host mobile app provides users' personal information to a data collection interface provided by the third-party library. For example, the aforementioned Firebase Analytics library can collect users' in-app activity information via an API called `FirebaseAnalytics.logEvent(key, bundle)`.

While some libraries also support withdrawal of such data collection via a pre-defined interface. The other popular method adopted by host apps is the setting of a guarding condition (e.g., as part of a *if* statement) to control data collection. Method ② in Figure 1 shows that the host app stops sharing user's location to Ad Library2 via a guarding condition `UserSetting.optoutStatus`.

2.3. Threat Model and A Motivating Example

There are three parties in our threat model: user, host mobile app, and third-party data collection library. We assume that users and third-party data collection libraries are trusted. That is, as long as the host mobile app follows the correct withdrawal practice provided by third-party libraries, we assume that third-party libraries follow their promise to stop data collection. It is an orthogonal problem to study the behavior of third-party libraries, which involve server activities.

At the same time, the host mobile app is potentially either buggy (e.g., forgetting to withdraw data collection) or malicious (e.g., intentionally missing the withdrawal). We do not differentiate these two behaviors as they are related to app developers' intention. Instead, we focus on the consequence, i.e., whether there exists an inconsistency between withdrawal choices and behaviors.

A Motivating Example. Now we present a motivating example with a zero-day withdrawal inconsistency. The inconsistency locates in a popular health tracker app, called Pacer, which has 10,000,000+ installs and an average rating of 4.8 stars. Figure 2 describes the inconsistency. To ensure the comprehensibility of the example, we have simplified the class signatures and code logic in Figure 2(b). The Pacer app provides a withdrawal option with a description called "App Usage Analytics" in Figure 2(a). However, the withdrawal option does not stop a third-party library, called Flurry, from collecting users' personal information such as gender, home domain, and various in-app interactions, which leads to the inconsistency and thus a potential violation of user's right under privacy regulations. More specifically, Figure 2(b) shows that the data collection happens in Block 7, which is not controlled by the withdrawal interface in Block 1. Instead, the withdrawal action only stops the data collection of another library Firebase Analytics and stops the data sale of Flurry. For example, the Firebase Analytics is informed in Block 6 to stop all analytic data collection, and Flurry in Block 5 to stop data sale upon the withdrawal selection.

3. Methodology

In this section, we present our system architecture and then the details of each component of MOWCHECKER.

3.1. Overall Architecture

The overall system architecture of MOWCHECKER is shown in Figure 3. First, MOWCHECKER detects withdrawal interfaces of a target mobile app using application layout analysis and semantic checking. The collected interfaces are represented as a triple of withdrawal option, personal data, and data collection entity. Second, MOWCHECKER statically detects data-flows related to personal information collection of third-party libraries. The collected data-flows are represented as a triple consisted of flow path, personal data and data collection entity. Lastly, MOWCHECKER checks the consistency between data-flows and withdrawal option specified in the interface. Specifically, MOWCHECKER correlates data-flows with withdrawal interfaces and then checks the

consistency between the flows and the option choice in the interface.

Detection of the Motivating Example. Now we illustrate how MOWCHECKER detects this zero-day withdrawal inconsistency using our motivating example in Figure 2 following the aforementioned three steps. Figure 2(b) shows all the related control-, data-, and view-dependencies that are useful for the detection.

First, MOWCHECKER identifies withdrawal options from the mobile app by analyzing the structure and semantics of layout files. Specifically, MOWCHECKER first labels the Switch node in Block 1—i.e., a stateful Android UI widget—as a candidate option. Then, MOWCHECKER extracts its surrounding description texts (*i.e.,*"App Usage Analytics") and determines the candidate option to be an withdrawal option since the phase "Usage Analytics" is identified as a data sharing statement according to its semantics.

Second, MOWCHECKER detects data-flows related to third-party data collections considering both host-uncooperated and cooperated data collection. Blocks 7–8 show an example of the host-cooperated data collection of Flurry. The usage data (especially web page information of ads in this case) collection of Flurry, starts from Block 7, *i.e.,* the source, and flows to the third-party data collection API (*i.e.,* `FlurryEventLogger.logEventWithParams(...)`), *i.e.,* the sink, in Block 8. MOWCHECKER takes the data collection APIs as sinks, and then performs backward data-flow analysis to identify such data-flows. MOWCHECKER also found data-flows of other third-party libraries, such as Firebase Analytics, via similar approaches.

Third, MOWCHECKER mines actual withdrawal behaviors and finally checks their consistency with the withdrawal expectations. A challenge here is that app often caches user's withdrawal choice in local storage or app server for future retrieval and use, and the one-way taint analysis usually fails to capture the withdrawal choices taken from the data storage. MOWCHECKER adopts bi-directional taint analysis to identify variables or methods that can represent user's current withdrawal choice, called *withdrawal fields*.

On one hand, MOWCHECKER performs a forward taint analysis. That is, MOWCHECKER first locates the listener registration statement (Block 2). Then, it starts from the parameter of corresponding event callback method (Block 3), which can reflect user's newly changed withdrawal choice. The withdrawal choice is finally updated to the app server inside the method `SettingPrivacySync.invoke(...)`. On the other hand, MOWCHECKER takes the parameter of `setChecked(...)` method as the taint source of the backward taint propagation. During the taint process, the methods `UserSetting.getUsageAnalyticStatus()` and `SharedPrefUtil.getUsageAnalyticStatus()` are both labeled as withdrawal fields. The data-flow dependency can also be identified in this process. For example, Block 5 and Block 6 show the explicit and implicit data-flow dependency between withdrawal choices and third-party withdrawal APIs (e.g., `FirebaseAnalytics.setAnalyticsCollection-Enabled(...)`).

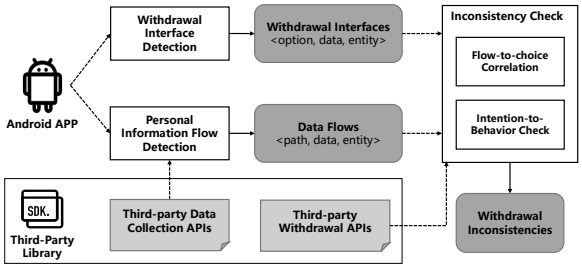


Figure 3: Overall architecture of MOWCHECKER.

After withdrawal choice identification, MOWCHECKER finds guarding conditions to identify the control-flow dependency between withdrawal choices and third-party data collections.

3.2. Withdrawal Interface Detection

This module automatically detects withdrawal interfaces of mobile apps. Our key observation for this module is that mobile withdrawal interfaces are often stateful UI widgets allowing users to switch between withdrawal and confirmation and having texts around to describe what data collection to withdraw Therefore, MOWCHECKER leverages two components (*i.e., Stateful Option Discoverer* and *Withdrawal Semantic Checker*) to identify withdrawal options in mobile apps.

Stateful Option Discoverer. MOWCHECKER, or particularly the stateful option discoverer, statically extracts all stateful UI widgets inside a given APK file as candidate withdrawal options. Specifically, MOWCHECKER first disassembles the Android APK file using apktool [25] and extracts its resource files, including layout files in the `res/layout` directory, definition files for PreferenceScreens in the `res/xml/` directory and string resources files in the `res/values` directory. The layout files are in XML whose vocabulary describes different UI widgets. For example, MOWCHECKER generates a UI widget of switch type for an XML node `<Switch android:text="@string/ALLOW_TRACKING_STR">`. Moreover, the text descriptions for the settings are in the widget node's attributes (e.g., `android:text="@string/ALLOW_TRACKING_STR"`) or in those of other text widgets (e.g., `TextView`) nearby. Then, MOWCHECKER checks widget types listed in Table 1 and their text descriptions and extracts candidate withdrawal options from the resource files. There are two things worth noting here. First, widget types in Table 1 are summarized manually based on all possible stateful Android UI widgets as listed in Android developers documentations [26]. Second, some apps may dynamically generate widgets and draw them in a view. The number of such widgets are

Table 1: Stateful Android UI widgets.

Widget Name	Switch, SwitchCompat, SwitchPreference, SwitchPreferenceCompat, ToggleButton, CompoundButton, RadioButton, CheckBox, CheckBoxPreference, selector, Spinner, Dialog

Table 2: Different forms of action words used by MOWCHECKER.

Type	Word
Active Verb	disclose, distribute, exchange, give, provide, rent, report, sell, send, share, trade, transfer, transmit, access, collect, gather, know, obtain, receive, save, store, get
Passive Verb	disclosed, distributed, exchanged, given, provided, rent, reported, sell, sent, shared, traded, transferred, transmitted, accessed, got, collected, gathered, knew, obtained, received, saved, stored
Noun/Gerund	disclosure, distribution, exchange, giving, providing, renting, report, sale, sending, sharing, trade, transferring, transmission, access, collection, gathering, knowing, receiving, saving, storing

relatively small (14.29% out of 100 apps) based on our manual inspection. We will consider them as our future work.

Withdrawal Semantic Checker. MOWCHECKER checks the semantics of each identified stateful UI option to further decide whether it is used for withdrawal. Specifically, MOWCHECKER analyzes and decides whether the descriptions of the candidate withdrawal options contain data collection/sharing statements so that users can know what data collection will be stopped if they choose to withdraw. MOWCHECKER represents each data collection/sharing statement as triples *(withdrawal option, data object, entity)*, i.e., the withdrawals of the collection of data object by the entity. MOWCHECKER then identifies UI options with data collection/sharing statements using patterns of the grammatical structures between data objects, entities, and different forms of action verb that represent sharing or collection.

Then, MOWCHECKER analyzes each candidate option. MOWCHECKER first uses a dependency parser [27] to transform the description sentence into a dependency parsing tree, which describes the grammatical connections between different words. Because withdrawal descriptions usually contain data collection/sharing phrases rather than sentences, MOWCHECKER utilizes three manually-summarized grammatical patterns corresponding to three voices of the action verb (*i.e.,* active verb, passive verb and noun/gerund form) as shown in Figure 4 for the analysis. MOWCHECKER leverages different forms of action words (listed in Table 2) as known anchors to locate the data objects and entities by traversing the parsing tree. Then, MOWCHECKER filters out the phrases irrelevant to privacy data objects by further checking whether the identified data object is privacy-related by calculating the semantic distance between the identified data object noun phrase and a set of privacy keywords [18]. After the filtering, if the candidate option contains at least one data collection/sharing phrases, MOWCHECKER annotates it as an withdrawal option for further analysis.

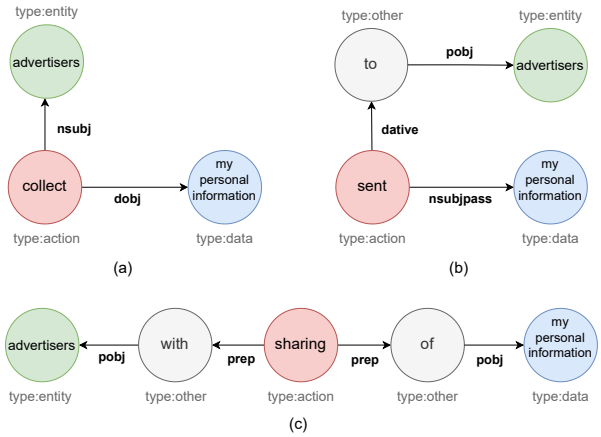


Figure 4: Examples for different grammatical patterns. (a) Active verb based: "do not let advertisers collect my personal information" (b) Passive verb based: "do not let my personal information be sent to advertisers" (c) Noun/Gerund based: "stop the sharing of my personal information with advertisers".

3.3. Personal Information Flow Detection

The personal information detection module statically discovers data-flows for both host-uncooperated and cooperated data collections from third-party libraries. Each data-flow is represented as a triple of flow path (source→sink), personal data and data collection entity.

Identification of flow sink, personal data and collection entity. MOWCHECKER uses a semi-automatic approach to extract third-party data collection APIs from SDK's Javadoc or Javadoc-like documentations. Such APIs are used later for sinks of flow path, and data entity as well as personal data related to APIs are also recorded. Here are the detailed procedures. Because such documentations tend to be highly structured, with well specified API names, argument lists, and descriptions, MOWCHECKER builds a parser to extract these information for each API. Next, MOWCHECKER leverages regular matching to identify APIs whose names or descriptions contains data collection/sharing verbs, and extracts the data objects to be collected for these APIs. Then, MOWCHECKER aligns these data objects with a set of privacy keywords [18] by calculating their semantic similarity to find out APIs related to privacy data collection. MOWCHECKER records each identified API as a triple *(API signature, personal data, data collection entity)*. Finally, we manually checked these APIs and removed the false positives which are not used to share personal data to third-party libraries (e.g., `ErrorInfo.setLocation(...)` is used to set where the error occurs rather than the user's current location).

Flow path detection. MOWCHECKER detects two types of flows for host-uncooperated and cooperated data collection. On one hand, MOWCHECKER refers to existing approaches, such as FlowDroid [14], to find personal information flows of third-party libraries related to host-uncooperated data collection in mobile apps. On the other hand, MOWCHECKER takes third-party data collection APIs as sinks and performs

backward data-flow analysis to detect data-flows from the host app to these sinks for host-cooperated data collection. Finally, MOWCHECKER combines the two types of data-flows to construct the full set of third-party libraries' personal information flows.

Note that MOWCHECKER does not specify the sources of these data-flows for host-cooperated data collection since the data may come from user input or the app server, which are hard to identify. That is, the purpose of MOWCHECKER is to identify a path from first-party code to third-party data collection APIs. The invoked data collection API can indicate the collected personal data and data collection entity for the data-flow.

3.4. Inconsistency Check

In this module, MOWCHECKER correlates personal information flows with withdrawal choices and then identifies those flows that are controlled by the withdrawal choices. Then, it performs an ontology-based consistency check to reveal the inconsistencies between what the user believes about withdrawing and what the app actually performs.

3.4.1. Flow-to-Choice Correlation. After mobile apps collect users' withdrawal choice, they need to either stop third-party data collection themselves or inform third-party libraries to stop collection. To detect such withdrawal behaviors, the key insight is that withdrawal choices should have either a control-flow dependency on personal information flow or a data-flow dependency on withdrawal APIs provided by third-party data collection libraries. Therefore, MOWCHECKER first identifies all variables representing user's current withdrawal choice in the mobile app. For convenience, we refer to such variables as *withdrawal field* in this work. Then, it performs data-flow and control-flow analysis to mine the correlation between the withdrawal fields and third-party personal information flows/withdrawal APIs.

Specially, we observe that the host app often caches user's withdrawal choice in local storage or app server for future retrieval and use. MOWCHECKER utilizes a multi-source, bi-directional taint analysis approach to detect stored withdrawal choices. As shown in Figure 5, MOWCHECKER first identifies initial fields of the withdrawal option via checking Android-specific methods. Then, MOWCHECKER performs forward taint propagation to identify more withdrawal fields in the app, which starts from the initial withdrawal fields and ends at third-party withdrawal APIs or some data storage operations (e.g., network request APIs). The backward taint analysis is performed to complement the identification of withdrawal fields. As the host app need to fetch cached withdrawal field to set the checked state of the withdrawal option (e.g., via `setChecked(boolean)` method), MOWCHECKER takes the withdrawal fields rendered in the withdrawal interface as taint source. Then MOWCHECKER performs backward taint analysis to find the path from app storage to the withdrawal interface and labels all tainted values in the path as withdrawal fields.

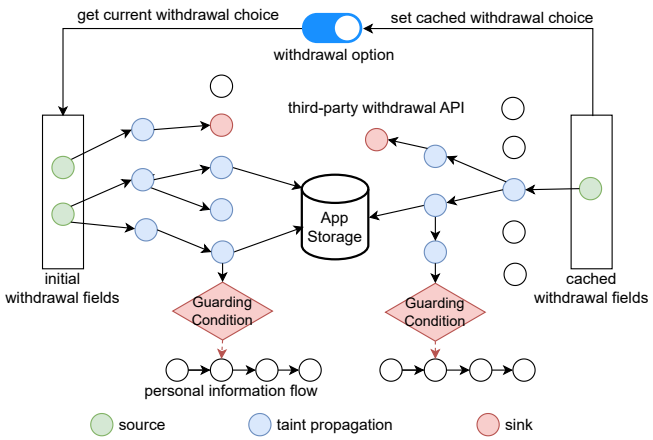


Figure 5: Multi-source, bi-directional flow-to-choice correlation.

The taint propagation process is iteratively executed on new identified withdrawal fields until it reaches a fix point. After that, MOWCHECKER performs correlation analysis based on the identified withdrawal fields by checking the guarding conditions of data-flows and data-flows of third-party withdrawal APIs.

Here, we present the details of (a) withdrawal taint sources identification, (b) taint propagation and (c) correlation analysis in this process.

(a) Withdrawal taint sources identification. For the forward taint analysis, MOWCHECKER identifies three types of initial withdrawal fields which can be used by app developers to get the current checked state of the withdrawal option:

• **Parameter of the event callback method.** The callback mechanism for Android widget is the most common way to obtain the newly changed state when users perform some GUI actions. In detail, GUI actions such as user's clicks of a ToggleButton or Switch, are passed to app code via the registered event callback methods, e.g., the method `onCheckedChanged(View, boolean)` will be called when the checked state of a compound button has changed.

To identify such withdrawal fields, MOWCHECKER extracts the registered callback methods for each withdrawal option from both codes and layout files. Specifically, it first checks the `findViewById(...)` invoke statements in app code to locate view objects for each withdrawal option. Then, it identifies all those event listener registration statements (e.g., `setOnClickListener(...)`) on these view objects and then gets the callbacks by extracting the name of argument class. For registration in layout files, MOWCHECKER parses the app's layout files and extracts the values of those event attributes (e.g., onClick attribute). Finally, MOWCHECKER identifies the target parameters of the callback methods via parameter type and labels them as initial withdrawal fields.

• **State-retrieval method.** Some stateful UI widgets provide state-retrieval methods for developers to get their current state, such as the `CheckBox.isChecked()` method.

Similar to the first case, MOWCHECKER first locates the view objects for each withdrawal option. Then it uses the def-use chains to find the invoking statements of these state-retrieval methods on the view objects and labels the definition boxes of these invoking statements as initial withdrawal fields.

• **Registered storage key.** Specifically, Preference widgets such as SwitchPreference support developers to associate the checked state of the widget with a SharedPreferences key by specifying the `android:key` attribute in layout files. For example, for the withdrawal widget `<SwitchPreference android:key="@string/PREF_OPTIN_KEY"/>`, developers can get the current state of the Switch widget by calling `SharedPreferences.getBoolean(R.String.PREF_OPTIN_KEY)`. MOWCHECKER parses the layout files and extracts the values of the *android:key* attribute as initial withdrawal fields for each withdrawal option.

For the source of the backward taint analysis, MOWCHECKER focuses on Android methods which are used to set the checked state of the UI widgets (e.g., `CheckBox.setChecked(boolean)`), which are manually collected from Android documentations. Then MOWCHECKER parses the app code to find invoking statements of these methods on the withdrawal view objects. The parameters of the invoking statements are labelled as taint sources to perform further backward taint analysis.

(b) Taint propagation. Starting from identified taint sources, MOWCHECKER performs forward and backward taint propagation to find all withdrawal fields in the app, with both explicit data-flow and implicit data-flow dependency considered. MOWCHECKER maintains a set called `FieldSet` to store all tainted withdrawal fields and the initial withdrawal fields are added to it at the beginning of the taint process. Then it iteratively performs taint propagation for each unvisited field in `FieldSet` to add new tainted fields, until it reaches a fixed point. Specially, MOWCHECKER utilizes two additional strategies to build the link between data storage and retrieval operations.

On one hand, MOWCHECKER models some key-value based data storage area such as SharedPreference. Specifically, MOWCHECKER leverages the storage key to recover the data-flow between data storage and retrieval operations. For example, as shown in Figure 6, via invoking `sh.edit().putBoolean("optInStatus", z)`, the withdrawal field is put to SharedPreference using the key `"optInStatus"`. To identify the withdrawal fields fetched from the data store, MOWCHECKER records the storage key and checks its references in the code to identify data-retrieval operations from the data store using this key. Thus the return value of the invoking statement `sh.getBoolean("optInStatus", true)` will be tainted.

On the other hand, we observe that developers usually use pair methods such as setter/getter for data storage and retrieval. In the shown example, `setUserOptInStatus` and `getUserOptInStatus` are a pair of methods to store and get user's withdrawal choice. Therefore, during the taint process, if MOWCHECKER finds the tainted value is passed to a method matching the setter patterns (e.g., `setUserOptInStatus`) summarized in Table 3, it checks whether corresponding getter method exists in the same class, and if exists, MOWCHECKER labels it as withdrawal field as well as new taint source.

Table 3: Pair methods patterns.

Setter Patterns	set.*, put.*, save.*, store.*, write.*, update.*, upload.*
Getter Patterns	get.*, restore.*, retrieval.*, read.*, is.*, has.*, load.*

(c) Correlation analysis. For each withdrawal option, MOWCHECKER records a set of correlated data-flows (named as A-`withdrawal`), which stands for the actual withdrawal behaviors of the withdrawal option. Based on the identified withdrawal fields, MOWCHECKER correlates data-flows with withdrawal choice in the following two ways.

• **Identify guarded data-flows.** Given a data-flow $f_{source} ... f_k ... f_{sink}$, for any f_k, we define a conditional statement c_e—at least one branch of which does not contain f_k—as a guarding condition. A data-flow is guarded by the withdrawal choice if at least one withdrawal field is used as its guarding condition. To check for guarded data-flows, MOWCHECKER first collects all conditional statements ($IfStmt$ in Soot) where the condition is the reference of an withdrawal field. Then given a data-flow, MOWCHECKER checks each conditional statements to see whether it is a guarding condition for the data-flow. If a guarding condition exists, MOWCHECKER labels the data-flow as guarded and adds it to A-`withdrawal`.

• **Identify correlated withdrawal APIs.** MOWCHECKER checks whether third-party withdrawal APIs have data-flow or control-flow dependency on the withdrawal fields. Note the third-party withdrawal APIs are manually annotated from libraries' documentations and we also annotate the control scope (*i.e.,* data collection of which personal data can be stopped via this API) for each withdrawal API. Based on these withdrawal APIs, MOWCHECKER first locates invoking statements for each of them and extracts its call path using constructed control-flow graph. Then, if the parameter of the invoking statement has been tainted in the taint process, it means there exists data-flow dependency between withdrawal choice and the withdrawal API. For the control-flow dependency, MOWCHECKER checks whether an withdrawal field is used as a guarding condition for invoking the withdrawal API. In both ways, user's withdrawal choice has been passed to third-party library via the withdrawal API, and the third-party library will stop corresponding data collections as they declare. Thus, for each correlated withdrawal API, MOWCHECKER adds the data-flows in its control scope to A-`withdrawal`.

3.4.2. Intention-to-Behavior Consistency Check. In this module, MOWCHECKER identifies which data-flows are

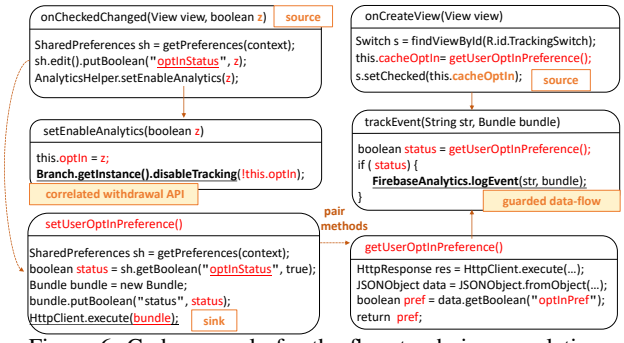


Figure 6: Code example for the flow-to-choice correlation.

expected to be stopped (called E-withdrawal) for each withdrawal option. Then, MOWCHECKER checks the consistency between E-withdrawal and the controlled data-flows (*i.e.,* A-withdrawal) and reports inconsistencies.

In the Withdrawal Interface Detection module (see Section 3.2), each withdrawal option is represented as an withdrawal triple *(option O, data object D_e, entity E_e)*. MOWCHECKER refines the withdrawal triple into a set of data-flows expected to withdraw. Such refinement is non-trivial since the data object and entity declared in withdrawal statements are usually coarse-grained, for example, for the withdrawal statement *"do not share my personal information with advertisers"*, MOWCHECKER checks which third-party libraries integrated in this app are advertisers and which data-flows are collecting personal information. To address this problem, MOWCHECKER employs the data ontology and entity ontology [18] to map coarse-grained data objects and entities to fine-grained data-flows.

Recall that MOWCHECKER has identified all third-party data-flows in the app, which covers the data collection of all target third-party libraries and we call it F. Each data-flow in F is represented as a triple (flow path p, data object d, entity e). MOWCHECKER then maps the withdrawal triple (O, D_e, E_e) to data-flows to get E-withdrawal by the following rule: Let D represent the total set of data objects and E represent the total set of data collection entities, respectively. For each data-flow $f \in F$, $f = (p, d, e)$ where $d \in D, e \in E$, if $d \sqsubseteq_o D_e$ and $e \sqsubseteq_o E_e$, then $f \in E$-withdrawal. Here $x \sqsubseteq_o y$ denotes that if x and y are synonyms or term x is subsumed under the term y according to the data ontology and entity ontology.

Finally, MOWCHECKER determines the withdrawal inconsistencies for each withdrawal option by comparing the actual withdrawal behaviors A-withdrawal with expected withdrawal behaviors E-withdrawal. The withdrawal inconsistency is formally defined and checked as follows:

Given E-withdrawal and A-withdrawal for an withdrawal option, if $\exists f \in E$-withdrawal, $f \notin A$-withdrawal, then the data-flow f is an *unguarded data-flow*. For an withdrawal option O, if $\exists f \in E$-withdrawal, f is an *unguarded data-flow*, then there exists *withdrawal inconsistency* for the option O.

4. Implementation

Now we discuss the implementation of MOWCHECKER in this section. MOWCHECKER is implemented in Python and Java, containing 16,289 Lines of Code (LoC) in total, excluding any third-party libraries, such as SpaCy dependency parser [27] and Soot [28]. We then discuss details of each component in MOWCHECKER.

First, we use apktool to decompile Android apk files. We write 1,253 lines of Java codes to extract stateful UI widgets as well as their description nodes from the layout files extracted from decompiled apk files. Then, MOWCHECKER uses pygtrans package [29], a Python wrapper of Google Translate API, to translate non-English texts into English. For semantic checking, MOWCHECKER leverages the SpaCy NLP library to parse and create the dependency trees for withdrawal descriptions.

Second, we adopt FlowDroid, a precise and efficient Java-implemented static analysis system, to discover personal information flows for host-uncooperated data collection. The backward data-flow analysis for host-cooperated data collection is also based on the control-flow graph constructed by FlowDroid.

Lastly, the flow-to-choice correlation module operates on Jimple intermediate representation (IR) [30], a typed 3-address IR suitable for optimization and easy to understand. We use Soot framework [28] to transform an app into Jimple codes, and write a Soot-based Java program in 10,510 lines of codes to perform the flow-to-choice correlation. For consistency check, we write 3,239 lines of Python codes to check the consistency between data-flows and withdrawal options.

Then, we summarize the dataset produced and consumed by each stage of MOWCHECKER as below.

Third-party Data Collection Libraries. We take the most popular third-party data collection libraries as study target. To identify them, we refer to the list of *Most Used SDKs For Android Mobile Apps* on AppTopia [31]. We select the top 200 most popular libraries since the remaining SDKs are less popular and integrated in less than 10,000 Google Play apps. Next, we exclude 88 libraries which are labeled as Development Tool or Development Platform (e.g., Google Gson SDK [32]), since they do not collect personal data. The remaining 112 unique third-party libraries are then used as the third-party library dataset for data-flow detection and withdrawal inconsistency check, which covers a variety of functional categories, such as analytics, advertising and monetization.

Third-party Data Collection APIs. As discussed, MOWCHECKER semi-automatically extracts third-party data collection APIs as flow sinks. MOWCHECKER uses the regex *"[set|put|log|track|share|store|write|add|record|send](\W*\w*)*"* for pattern matching. Then we manually check these APIs and remove the false positives which are not used to share personal data to third-party libraries. As a result, we gathered 3,053 third-party data collection APIs for all the 112 third-party data collection libraries.

Table 4: Examples of third-party withdrawal APIs.

API	Params	Description
setAppOptOut	true	You can enable an app-level opt out flag that will disable Google Analytics across the entire app.
setCollectIMEI	false	Withdrawal of collection of IMEI.
setEnabled	false	You can enable and disable tracking with the Adjust SDK using the setEnabled method.
updateCCPA--Status	false	Updates the data-gathering consent status of the user. If the state is true, the user has consented to us gathering data about their device.

Third-party Withdrawal APIs. We manually annotate withdrawal APIs from third-party library's documentations, including developer documentation, tutorial, API reference, and Javadoc. We label an API as an withdrawal API if its description states that it can be used to control the collection of certain kinds of data. We then identify its control scope according to its description and usage guide. Two privacy professionals manually reviewed documentations for the 112 third-party libraries and finally identified 169 third-party withdrawal APIs. Some examples of the withdrawal APIs identified are listed in Table 4. These withdrawal APIs are then used by the Inconsistency Check module to identify actual withdrawal behaviors.

5. Evaluation

In this section, we run a series of experiments to evaluate the performance of MOWCHECKER. We run all the experiments on two Ubuntu 18.04 LTS 64-bit servers with Intel Xeon Gold 5218, 2.30 GHz (40 cores/207GB memory, 40 cores/378GB memory respectively). Our evaluation answers the following Research Questions (RQs):

- **RQ1:** What are MOWCHECKER's precision, recall, and accuracy in identifying withdrawal inconsistencies?
- **RQ2:** What is MOWCHECKER's analysis time in identifying withdrawal inconsistencies?
- **RQ3:** How many zero-day withdrawal inconsistencies are reported by MOWCHECKER?

5.1. Dataset

We collected two datasets, totaling 75,725 apps, which contain both high-profile and long-tail apps on the Play Store. Specifically, we crawled the top free high-profile apps in January 2023 from the Google Play store based on the AppTopia statistics [33] in Germany and US region respectively. In detail, we picked the top 500 under each category, unless some categories do not have a full list of 500 apps. After removing duplicates, our crawler obtained 25,725 high-profile apps in total. To get long-tail apps, our crawler randomly crawled free Android apps available in the store from the Google Play store by using a list of search keywords in the AOL Query Log dataset [34], which resulted in one million apps. Then, we excluded those already in the high-profile set and whose downloads are less than 10,000, and randomly sampled 50,000 distinct apps from the remaining apps as our long-tail app dataset.

Analysis Statistics. The analysis is performed in parallel and has a timeout of one hour to analyze each app. The sizes of apps range from 9.6KB to 563MB. Finally, we successfully analyzed 70,364 (92.92%) apps, including 23,001 high-profile apps and 47,363 long-tail ones. The remaining 5,361 apps either ran out of time or failed to be analyzed by Soot.

5.2. RQ1: Precision, Recall, and Accuracy

This research question answers MOWCHECKER's effectiveness in analyzing real-world Android apps, particularly in terms of precision, recall, and accuracy. Because there is no existing ground truth for such inconsistencies, we manually compare existing withdrawal options and their behavior via dynamic testing and reverse engineering and label them as either consistent or inconsistent. Specifically, we randomly sample 100 withdrawal options and ask two computer science students for independent labeling. Whenever they have a conflict, they resolve it by discussing it with a security expert. Here is the detailed methodology. The analysts first install the mobile app on real Android devices, and then manually explore the app to interact with the target withdrawal option. They monitor the app's third-party data collection behaviors and withdrawal behaviors by hooking third-party data collection APIs and withdrawal APIs. If data collection activities persist even after withdrawing, and the corresponding withdrawal API is not invoked to halt data transmission, the option is labeled as inconsistent. Since some data collection activities are hard to trigger (e.g., collection of diagnostic reports when app crashes), analysts investigate app's code to reveal withdrawal behaviors as a supplement. In addition, some apps cannot be dynamically tested due to the need for special credentials, payment, area, or device restrictions. In practice, both students unanimously annotated all withdrawal options, spending around 80 hours in total. Now let us describe the ground truth results in Table 5. Among the 100 withdrawal options, 85 withdrawal options are successfully inspected using this approach and annotators cannot decide whether the rest 15 has consistent withdrawals because of reasons mentioned above, such as the requirement of special credentials or payment. 32 options out of 85 do not comply with users' withdrawal choice to stop corresponding third-party data collection, while 53 are consistent withdrawals.

Then, we measure the true positives (TP), false positives (FP), true negatives (TN) and negatives (FN) false negatives (FN) by comparing the results of MOWCHECKER with manually annotated ground truth. We regard withdrawal inconsistency as positive data, and consistent withdrawal as negative data. TP indicates withdrawal options labelled by both MOWCHECKER and human analysts as inconsistent, while FP are labelled by MOWCHECKER to be inconsistent but analysts determine them to be consistent. TN denotes withdrawal options labelled by both MOWCHECKER and

Table 5: Manually-annotated Ground Truth. (Annotators cannot decide on 15 withdrawal options because of various reasons such as the requirement of payment, special credentials, and device.)

# of Withdrawal Options	# of Consistent Options	# of Inconsistent Options	# of Undecidable Options
100	53	32	15

Table 6: Overall performance of MOWCHECKER against the Ground Truth.

TP	TN	FP	FN	Precision	Recall	Accuracy	F1-Score
29	47	6	3	82.86%	90.63%	89.41%	86.57%

human analysts as consistent and FN are inconsistent withdrawal options according to our ground truth but labelled as consistent by MOWCHECKER.

Table 6 shows that MOWCHECKER achieves 82.86% precision and 90.63% recall among 85 verifiable withdrawal options. In detail, MOWCHECKER identifies 35 withdrawal inconsistencies from the 85 withdrawal options while 29 of them are true positives. For the false positives, three of them are caused by the incomplete identification of withdrawal fields in apps, since some customized callbacks or code obfuscation in mobile apps affect the integrity of static taint propagation. Two of them are considered false positives of identifying withdrawal options. For example, the option is a redundant option in the app which will not be presented to users under any circumstances. Thus, no corresponding withdrawal behavior is performed behind this option, which is justified. And the remaining one is due to the misinterpretation of the withdrawal option. The main reason for the three false negatives is that some mobile apps have heavy code obfuscation, which causes MOWCHECKER to fail to detect some third-party personal information flows. In this case, the set of data-flows that are expected to be stopped is empty while the actual situation is not, resulting in the option being wrongly judged as consistent.

5.3. RQ2: Analysis Time Distribution

This research question evaluates the runtime performance overhead of MOWCHECKER. In total, it took around 10 days for MOWCHECKER to finish all the tasks including withdrawal interface detection, personal information flow detection and inconsistency check on all the 75,725 apps. Note that the analysis is performed in parallel and has a timeout of one hour to analyze each app. Finally, only 2.09% of these apps ran out of time. In addition, if no withdrawal option is detected in an app, the latter two modules will not run, thus we evaluate the runtime performance of MOWCHECKER on apps with withdrawal options detected.

Figure 7 shows the overall and break-down analysis time distribution. Overall, the analysis time per app ranged from 17 seconds to about 44 minutes, with an average of 286 seconds and a medium of 178 seconds. Among the three components, the personal information flow detection and inconsistency check module consume more time since they

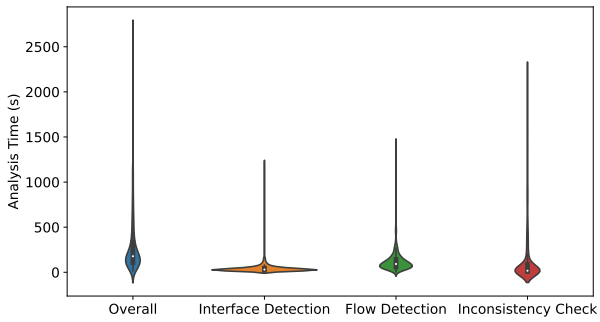


Figure 7: The analysis time distribution of MOWCHECKER.

Table 7: Statistics of withdrawal inconsistencies on the high-profile and long-tail dataset.

Dataset	# of apps	# of apps with withdrawal options	# of apps with withdrawal inconsistency	% of withdrawal inconsistency
High-profile	23,001	1,592	804	50.50%
Long-tail	47,363	2,243	710	31.65%
Total	70,364	3,835	1,514	39.48%

involve data-flow analysis which is usually time-consuming. In most cases, MOWCHECKER can finish analysis in a matter of minutes, while some outliers indicate that some apps take longer, often due to their large size and high code complexity. To summarize, the results indicate that each component of MOWCHECKER is efficient, as is the overall approach.

5.4. RQ3: Withdrawal Inconsistency Statistics

This research question answers the number of zero-day withdrawal inconsistencies MOWCHECKER discovered and their statistics. We first show the overall statistics of MOWCHECKER's detection results and then describe the statistics of manually verified results excluding any false positives. First, Table 7 shows that MOWCHECKER identified 1,514 apps with withdrawal inconsistencies, including 804 in high-profile apps (top 500 of each category in Google Play) and 710 in long-tail apps. In detail, MOWCHECKER locates 3,835 apps with withdrawal options in 70,364 real-world mobile apps, while 39.48% of them incompletely or failed to follow users' choices to stop third-party data collection. Our results indicate that although some developers already realized the necessity to provide withdrawal options to users, they may not effectively halt the corresponding data collection in practice.

Second, we conduct a more in-depth analysis of reported withdrawal inconsistencies to reveal further details about this issue. Considering the potential impact of our tool's imperfect accuracy on the measurement study results, we carefully selected the top 200 apps under each category in our high-profile dataset for closer scrutiny and manually verified any instances of withdrawal inconsistency in these apps. Following this, we present the results of withdrawal inconsistency statistics, MOWCHECKER identified 297 apps with withdrawal inconsistencies out of a total of 6,740 apps. Subsequently, we conducted a manual analysis of these apps, and locate 157 withdrawal inconsistencies. The remaining

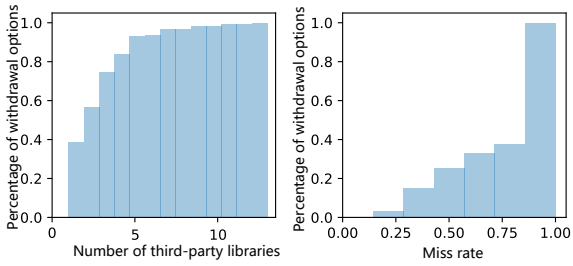


Figure 8: Cumulative distribution function (CDF) of miss count and miss rate of third-party libraries.

apps cannot be analyzed for various reasons such as the requirement of device by a specific vendor.

Degree of Inconsistency. Some withdrawal options may involve multiple third-party libraries, and in case of inconsistency, they may only stop the data collection of some of these libraries, or even none at all. We then assess the inconsistency level of withdrawal options for third-party libraries, specifically analyzing the extent to which the withdrawal behavior of the target libraries is absent. Figure 8 illustrates the cumulative distribution function (CDF) of missed third-party libraries for withdrawal options, as well as the miss rate. Surprisingly, although over 75% of the withdrawal options involve fewer than three third-party libraries, some apps have more than 10 third-party libraries that are not withdrawn. Further investigation of these withdrawal options reveals that they often have overly broad definitions without sufficient clarification (e.g., *"do not sell personal information"*). Consequently, in practice, these options frequently fail to prevent personal data from being shared with multiple third parties.

Additionally, we find that some options have a miss rate of 100%, which means the app does not stop any third-party data collection as expected. Our further examination finds some of them are deceptive since they do nothing other than save the user's withdrawal status for rendering the UI, while some presented as seemingly optional actually require mandatory consent. We provide a few case studies of such violations in Section 6.

Affected Personal Data. Personal data that continues to be collected by third parties, even after the user has explicitly refused, may be considered illegal data collection under regulations. Figure 9 shows the distribution of such third-party data collection. We then analyze the affected personal data and third-party libraries to show the severity of such withdrawal inconsistencies.

We find that the most affected personal data type is usage data, which has not been well studied in prior works but may pose significant privacy risks to users. Such data mainly comes from user interactions with the app and can be used to track user behavior and preferences. For example, user's browsing and search history in a news&magazines app may reveal users' interests as well as some sensitive information such as political beliefs. Mobile apps may share such information with third-party libraries for targeted advertising or in exchange of some other valuable services, which can be invasive and unwanted for users. While these data is

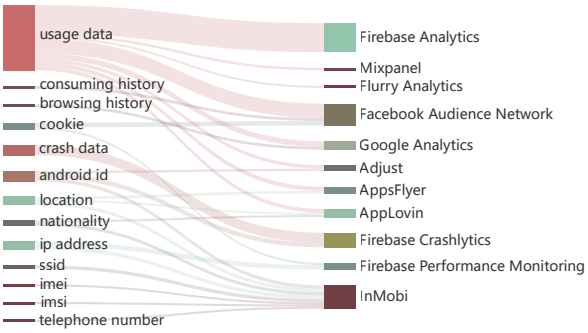


Figure 9: Distribution of unguarded third-party data-flows.

not protected by Android permissions, users lose the real control over this kind of privacy in the case of withdrawal inconsistency, and thus face serious privacy risks.

Affected Third-party Libraries. The withdrawal inconsistencies result in third-party libraries collecting and processing user data without the user's consent, which potentially violates the relevant requirements for user consent in laws and regulations such as GDPR. This exposes these third-party libraries to the risk of being punished by regulations. From the statistics, it is found that withdrawal inconsistencies mainly involve Analytics libraries (e.g., Firebase Analytics) and Advertising libraries (e.g., InMobi). These libraries are widely used in mobile apps and have been shown to collect and process large amounts of user data [12].

6. Case Study

In this section, we illustrate several case studies for the withdrawal inconsistencies reported by MOWCHECKER, to better understand their possible causes. Consistent withdrawals are usually challenging to achieve, as they depend on the proper coordination between the host app and third-party libraries. During our validation process, we observed different types of withdrawal inconsistency, and we illustrate each one with a representative case here.

Deceptive Withdrawal. MAPS.ME, a popular Map app with 50M+ downloads, provides an withdrawal option for users to withdraw the collection of crash data, with a description as *"Crash report. We may use your data to improve MAPS.ME experience"*. However, MOWCHECKER finds that the app still collects such crash data despite an withdrawal choice from users. Specifically, MAPS.ME only changes the UI state but does not perform any actual actions to stop data collection. The app does provide a wrapper method (*i.e.,* isCrashlyticsEnabled()) to get the current withdrawal choice, but the method is not used.

Incomplete Withdrawal. Work Log is a popular Productivity app with 1M+ downloads and an average rating of 4.7. MOWCHECKER detects that the app provides an withdrawal option called *"Google Analytics"*. The option stops sharing customized usage data with Firebase Analytics via guarding its data-flows. However, the automatic data collection of Firebase Analytics [24] is not stopped since the app does not call the provided withdrawal API. During our developer

notification, the app developers stated that they do not know such automatic data collection by third-party libraries and thus are not aware of the inconsistency.

Withdrawal Scope Mismatch. Pacer Pedometer & Step Tracker, a Health and Fiteness app with 10M+ downloads, i.e. our motivating example, has an withdrawal option with a description as *"App Usage Analytics"*. In this app, Firebase Analytics and Flurry Analytics are both integrated to collect usage data. When users choose to withdraw, MOWCHECKER finds that data collection of Firebase Analytics is disabled via calling the withdrawal API. However, the withdrawal of Flurry is not appropriately enforced, which has a scope mismatch with its stated withdrawal description. Specifically, the app calls an withdrawal API of Flurry, called `withDataSaleOptOut(true)`, which only prevents Flurry from sharing data with other parties, but not collecting user data from their device [35].

7. Developer Notification

We have responsibly notified the developers of affected mobile apps about all withdrawal inconsistencies that have been manually validated. This serves two main objectives: first, to inform them of the identified withdrawal inconsistencies and enable them to address this issue; and second, to gain insights into the underlying reasons that caused the withdrawal inconsistency in the first place.

We are aware of the ethical concerns when conducting the developer notification. During the notification process, we emphasize that we will not collect any personal information from developers. We also inform developers of the purpose of the study and follow best practices established by prior works [36], [37], [38] allowing developers to withdraw. This approach ensures that our research is conducted in an ethical and transparent manner, while also respecting the privacy and autonomy of the developers involved.

7.1. Notification Campaign

To notify the affected app developers, we extracted the email addresses they submitted to Google Play. To ease the overhead of handling our reports, we briefly explained the *withdrawal inconsistency* problem and put details in an attached report. Besides, three questions were asked, including if they are aware of such inconsistency, if they know how to withdraw third-party data collection, and what their plans are to remedy this problem or any proposal for support (see Appendix A for the email template).

We performed a notification campaign to inform the affected app developers, which was finished before March 23, 2023. Overall, 157 unique report emails were sent and 170 of them were successfully sent. The failed ones are due to various reasons such as invalid email addresses and inboxes of the recipients being full.

7.2. Developer Response

In total, 66 unique responses were received, where 43 of them are automated replies. Among the remaining 23 manual responses, five confirmed our report along with feedback towards our questions. One of them acknowledged the inconsistency and mentioned that they were developing a new compliant version. Two responses said they were not aware of this issue but promised to investigate and fix it. In addition, the rest two responses said that they had already fixed the withdrawal inconsistency issue in their latest version of the app. Note that the response rate was relatively low. We notice that one possible reason can be the consideration of protecting certain business secrets or their internal security restrictions. For instance, two of the app developers investigated our report but said that they were unable to disclose any information regarding our queries. Most of the manual responses neither confirmed nor denied our report, but said "We'll forward this report to the appropriate team" or stated that they required further investigation within their respective companies. Note that there may be multiple back-and-forth processes during the notification campaigns, as app developers may require additional details and support to address the withdrawal inconsistency problem.

8. Discussion & Limitations

In this section, we discuss some issues related to the implementation and evaluation, as well as the limitations of our tool.

Third-Party Data Collection Libraries. In the current implementation, we have focused solely on withdrawal inconsistencies related to the top 200 popular third-party libraries, due to the impracticality of manually annotating withdrawal APIs for all third-party libraries. However, this approach is still reasonable and has a negligible impact on the tool's results. On the one hand, we have covered the mainstream third-party libraries, and the remaining libraries are less popular, with the 201st ranked SDK integrated into fewer than 10,000 Google Play apps. On the other hand, our tool can be easily extended, as users can add new third-party libraries through the configuration file when needed.

Roles in Data Processing. The GDPR establishes two roles for various parties: the Data Controller and the Data Processor, which are akin concepts to Business and Service Provider as defined in CCPA. The Data Controllers are parties that determine the purposes and means of the processing of personal data, while the Data Processors process data on behalf of the Data Controllers. As per the definitions provided by the GDPR, the responsibility for stopping the processing of personal data lies with the Data Controllers, but not with the Data Processors such as Firebase Analytics. In an application, a third-party library may serve as a Data Processor. However, for the withdrawal interface is provided by the Data Controller, which is the first party app code instead of the Data Processor, such as third-party libraries. In other words, a user is asking the Data Controller instead

of the Data Processor to withdraw data processing. This aligns with "Right to Object to Processing of Personal Data" in Article 21 of GDPR. Thus, MOWCHECKER reports a withdrawal inconsistency whatever the roles of third party libraries are.

False Positives. Due to the limitations of static analysis, MOWCHECKER has some false positives. For instance, MOWCHECKER may overlook some apps' withdrawal solutions due to factors such as Java reflection, code obfuscation, or inadequate pattern coverage, resulting in false positives. Nonetheless, our evaluation results indicate that MOWCHECKER maintains practical precision. While dynamic analysis may seem like a more intuitive method for detecting withdrawal inconsistencies, it is not applicable in our study for the following reasons. Firstly, withdrawal options are deeply concealed within apps, making it difficult to trigger them through dynamic testing. Secondly, the trigger conditions for third-party data collections are complex, and it is challenging to guarantee adequate coverage.

False Negatives. Since we only consider withdrawal options defined in layout files, the withdrawal options presented in WebView or with dynamically-retrieved description texts are not detected by MOWCHECKER, potentially resulting in the omission of some violations. Assessing end-to-end false negatives for withdrawal inconsistencies is challenging. On the one hand, some withdrawal options may only appear under specific conditions (e.g., in certain regions), making it difficult to establish a ground truth for identifying withdrawal options. On the other hand, some non-reported withdrawal options may involve code in other languages such as JavaScript, which differs significantly from app-customized withdrawal mechanisms, making it difficult to evaluate their consistency. Due to these limitations, we only evaluate false negatives of withdrawal inconsistencies for reported withdrawal options. Therefore, our work only revealed the "lower bound" of the withdrawal inconsistencies in the wild.

Scalability of MOWCHECKER. The detection accuracy of MOWCHECKER may be affected by the Android UI widgets, Source-Sink APIs and the withdrawal APIs. MOWCHECKER detects withdrawal interfaces with stateful Android UI widgets which are less prone to becoming outdated. The manual curation of all possible Android UI widgets is a one-time effort that is solely related to Android versions. Our analysis reveals that the types of Android UI widgets have stayed the same from Version 4.2 until the present, spanning an 11-year period. In addition, MOWCHECKER inherits the Source-Sink APIs provided by FlowDroid, which may affect the accuracy since they are outdated. Therefore, one future work revolves around the updating of outdated APIs used in MOWCHECKER. Finally, the withdrawal APIs used in MOWCHECKER may also need to be updated regularly. Manual annotation requires an average of only 15 minutes per third-party library.

Third-party Withdrawal Frameworks. We notice that some mobile apps utilize third-party withdrawal frameworks, such as the mobile Consent Management Platform (CMP) SDK, to offer withdrawal interfaces. These interfaces are commonly defined in WebView or configuration files, which are not captured by our current solution. Moreover, the efficacy of these withdrawal interfaces depends on the implementation of the withdrawal framework, rather than the mobile app that is the focus of our research. Therefore, we consider this to be a potential area for future investigation.

9. Related Works

In this section, we begin by presenting related works on privacy regulation compliance analysis in mobile apps. Next, we discuss existing research on mobile privacy leaks.

9.1. Privacy Regulation Compliance

With more and more privacy regulations (e.g., GDPR and CCPA) went into effect, lots of works have made efforts to study the privacy regulation compliance in the wild.

Opt-out related study. When considering data withdrawal, previous studies mainly focused on the right to opt-out with the analysis on the existence and usability of opt-out options on websites, while little research has been done to measure the violation of withdrawals in mobile apps. Specifically, lots of works [5], [6], [7], [9], [10], [39], [40], [41] paid their attention to cookie notices, which is the mainstream way for websites to inform users about their data practices and enable users to opt-out/opt-in. To extract opt-out options, Sathyendra *et al.* [42] and Bannihatti *et al.* [43] made efforts to automatically extract opt-out statements from website privacy policies. They based on the observation that the privacy policy text describing opt-out options often includes hyperlinks and trained a logistic regression classifier to automatically detect opt-outs in privacy policy.

One popular research direction—which is closely related to our work—is to measure opt-out inconsistencies for websites. Bui *et al.* [4] showed that online trackers embedded as part of websites may exhibit data practices inconsistent with the host websites' stated opt-out policies. Sanchez *et al.* [6] evaluated the effectiveness of rejecting cookie notices by comparing the number of cookies before and after the user rejected them and found rejecting tracking is often ineffective. Moreover, Matte *et al.* [41] focused on cookie banners implemented by Consent Management Providers (CMPs), who respect IAB Europe's Transparency and Consent Framework (TCF) [44] to collect and disseminate user consent to third parties. They inspected 1,426 websites that use CMPs and found that 5% of them store a positive consent even if the user has explicitly opted out.

While these works mainly focus on withdrawal of cookie tracking on websites, mobile withdrawal options are more challenging to extract because they are often embedded inside multiple user interfaces, e.g., via Android activities. Then, mobile personal data is usually more complex yet diversified involving those collected by mobile sensors as compared with cookies.

Regulation violations in mobile apps. A line of works study legislation violations of mobile apps after GDPR and

CCPA went into effect. In particular, Nguyen *et al.* [36] performed a large-scale measurement study on Android apps in the wild to understand the current state of the violation of GDPR's explicit consent. Santhanam *et al.* [45] worked to detect app user information left on servers after account deletion, which checked app's compliance of the "right to be forgotten". Kollnig *et al.* [46] revealed the absence of consent to third-party tracking in Android apps. Nguyen *et al.* [47] and Koch *et al.* [48] performed large-scale studies into consent notices of third-party tracking in Android apps in the wild to understand the current practices and the current state of GDPR's consent violations. While they mainly focused on consent notices that pop up on app's start activity, we are concerned with withdrawal options which are orthogonal directions.

9.2. Mobile Privacy Leaks

Many works detected privacy collection or leakage in mobile apps as well as integrated third-party libraries using static program analysis [14], [49], [17], [20], [11], [23], [22] or dynamic analysis [15], [16], [50], [51]. More recently, another line of works works assessed whether an app's data practice (e.g., data collection and sharing with third-party) is consistent with what is disclosed to users, either presented to users in app UI or in its privacy policy [52], [53], [13], [54], [55], [56], [18], [19]. They leveraged natural-language processing (NLP) techniques, such as patterns of the grammatical structures to extract sharing and collection statements from app's descriptions or privacy polices [55], [19], [56], or they extract semantics from texts and images in UI to infer user's intention [13], [54]. PoliCheck [19] performed an entity-sensitive flow-to-policy consistency analysis while builds on top of PolicyLint [18] to identify data collection and sharing statements. To identify data collection behavior, most of these works only checked data-flows of permission-based privacy, while GUILeak [56] paid attention to the flow-to-policy consistency for user-input data.

Being different from them, MOWCHECKER checks the consistency between withdrawal choices and third-party data collections, where user intention presented in the UI and actual withdrawal behavior both need to be considered. However, withdrawal behaviors usually involve dedicated interface and special third-party libraries. More importantly, we pay more attention to reveal the control or data-flow dependency between withdrawal choices and third-party data-flows, which faces additional challenges. That is, none of prior works can understand withdrawal decisions made by mobile users and withdrawal behaviors of mobile apps let alone study the consistency between user choice and mobile apps' withdrawal behaviors.

10. Conclusion

Many mobile apps provide withdrawal interfaces for users to exercise their withdrawal rights. However, some apps still transmit data to third-party libraries even when users have explicitly withdrawn data collection via such interfaces, which is defined as an withdrawal inconsistency. In this paper, we design and implement a tool, called MOWCHECKER, to automatically identify withdrawal inconsistencies in mobile apps. Our insight is that withdrawal interfaces have control- or data-flow dependencies with the actual data collection behaviors. Then, our evaluation of MOWCHECKER on real-world Android apps reveals 157 manually-confirmed, zero-day withdrawal inconsistencies. Our work shows the urgent need for better collaboration between third-party libraries and mobile apps to respect users' withdrawal choices. We hope that MOWCHECKER will help future researchers as well as developers to better protect users' withdrawal rights and fill the gap between law enforcement and software development.

11. Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments that helped improve the quality of the paper. This work was supported in part by the National Key Research and Development Program (2021YFB3101200), National Natural Science Foundation of China (62172104, 62172105, 61972099, 62102093, 62102091), and National Science Foundation (CCF2317185). Zhemin Yang was supported in part by the Funding of Ministry of Industry and Information Technology of the People's Republic of China under Grant TC220H079. Dr. Yinzhi Cao was supported in part by Johns Hopkins Catalyst Awards. Min Yang is the corresponding author, and a faculty of Shanghai Institute of Intelligent Electronics & Systems and Engineering Research Center of Cyber Security Auditing and Monitoring.

References

[1] T. E. Union. (2022) General data protection regulation(gdpr). https://gdpr.eu/.

[2] C. D. of Justice. (2022) California consumer privacy act (ccpa). https://www.oag.ca.gov/privacy/ccpa.

[3] Google. (2022) Firebase analytics. https://firebase.google.com/docs/analytics.

[4] D. Bui, B. Tang, and K. G. Shin, "Do opt-outs really opt me out?" in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 425–439.

[5] D. Bollinger, K. Kubicek, C. Cotrini, and D. Basin, "Automating cookie consent and gdpr violation detection," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2893–2910.

[6] I. Sanchez-Rola, M. Dell'Amico, P. Kotzias, D. Balzarotti, L. Bilge, P.-A. Vervier, and I. Santos, "Can i opt out yet? gdpr and the global illusion of cookie control," in *Proceedings of the 2019 ACM Asia conference on computer and communications security*, 2019, pp. 340–351.

[7] X. Hu and N. Sastry, "Characterising third party cookie usage in the eu after gdpr," in *Proceedings of the 10th ACM Conference on Web Science*, 2019, pp. 137–141.

[8] H. Habib, Y. Zou, A. Jannu, N. Sridhar, C. Swoopes, A. Acquisti, L. F. Cranor, N. Sadeh, and F. Schaub, "An empirical analysis of data deletion and opt-out choices on 150 websites," in *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, 2019, pp. 387–406.

[9] C. Utz, M. Degeling, S. Fahl, F. Schaub, and T. Holz, "(un) informed consent: Studying gdpr consent notices in the field," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 973–990.

[10] M. Degeling, C. Utz, C. Lentzsch, H. Hosseini, F. Schaub, and T. Holz, "We value your privacy... now take some cookies: Measuring the gdpr's impact on web privacy," *arXiv preprint arXiv:1808.05096*, 2018.

[11] X. Liu, J. Liu, S. Zhu, W. Wang, and X. Zhang, "Privacy risk analysis and mitigation of analytics libraries in the android ecosystem," *IEEE Transactions on Mobile Computing*, vol. 19, no. 5, pp. 1184–1199, 2019.

[12] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich, P. Gill *et al.*, "Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem," in *The 25th Annual Network and Distributed System Security Symposium (NDSS 2018)*, 2018.

[13] X. Pan, Y. Cao, X. Du, B. He, G. Fang, R. Shao, and Y. Chen, "FlowCog: Context-aware semantics extraction and analysis of information flow leaks in android apps," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1669–1685.

[14] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *Acm Sigplan Notices*, vol. 49, no. 6, pp. 259–269, 2014.

[15] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, pp. 1–29, 2014.

[16] M. Sun, T. Wei, and J. C. Lui, "Taintart: A practical multi-level information-flow tracking system for android runtime," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 331–342.

[17] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel, "Iccta: Detecting inter-component privacy leaks in android apps," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 280–291.

[18] 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 *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 585–602.

[19] 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 *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 985–1002.

[20] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "Appcontext: Differentiating malicious and benign mobile app behaviors using context," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 303–313.

[21] B. C. L. P. LLP. (2022) Definitions of ccpa. https://ccpa-info.com/home/1798-140-definitions/.

[22] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Investigating user privacy in android ad libraries," in *Workshop on Mobile Security Technologies (MoST)*, vol. 10, 2012, pp. 195–197.

[23] S. Demetriou, W. Merrill, W. Yang, A. Zhang, and C. A. Gunter, "Free for all! assessing user data exposure to advertising libraries on android." in *The 23th Annual Network and Distributed System Security Symposium (NDSS 2016)*, 2016.

[24] Google. (2022) Automatically collected events. https://support.google.com/firebase/answer/9234069.

[25] iBotPeaches. (2020) Apktool - a tool for reverse engineering android apk files. https://github.com/iBotPeaches/Apktool.

[26] Google. (2021) Android developer documentation. Google. [Online]. Available: https://developer.android.com/docs/

[27] M. Honnibal and I. Montani, "spacy: Industrial-strength natural language processing in python," https://spacy.io/, 2017, accessed on 25 March 2023.

[28] P. Lam, E. Bodden, O. Lhoták, and L. Hendren, "The soot framework for java program analysis: a retrospective," in *Cetus Users and Compiler Infastructure Workshop (CETUS 2011)*, vol. 15, no. 35, 2011.

[29] P. community, "pygtrans." 2021, accessed: February 15, 2023. [Online]. Available: https://pypi.org/project/pygtrans/

[30] R. Vallee-rai and L. Hendren, "Jimple: Simplifying java bytecode for analyses and transformations," 01 2004.

[31] Apptopia. (2022) Most used sdks for android mobile apps. https://apptopia.com/top-charts/top-sdks/google-play/all.

[32] Google. (2022) Google gson sdk. https://github.com/google/gson.

[33] [Apptopia, "Top android apps." 2022, accessed: February 15, 2023. [Online]. Available: https://apptopia.com/store-insights/top-charts/google-play.

[34] G. Pass, A. Chowdhury, and C. Torgeson, "A picture of search," in *Proceedings of the 1st international conference on Scalable information systems*, 2006, pp. 1–es.

[35] Yahoo. (2019) Ccpa summary. Yahoo. [Online]. Available: https://developer.yahoo.com/flurry/docs/analytics/privacyregulation/ccpa/

[36] T. T. Nguyen, M. Backes, N. Marnau, and B. Stock, "Share first, ask later (or never?) studying violations of gdpr's explicit consent in android apps," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3667–3684.

[37] F. Li, Z. Durumeric, J. Czyz, M. Karami, M. Bailey, D. McCoy, S. Savage, and V. Paxson, "You've got vulnerability: Exploring effective vulnerability notifications." in *25th USENIX Security Symposium (USENIX Security 16)*, vol. 16, 2016.

[38] B. Stock, G. Pellegrino, C. Rossow, M. Johns, and M. Backes, "Hey, you have a problem: On the feasibility of large-scale web vulnerability notification." in *25th USENIX Security Symposium (USENIX Security 16)*, vol. 16, 2016.

[39] A. Dabrowski, G. Merzdovnik, J. Ullrich, G. Sendera, and E. Weippl, "Measuring cookies and web privacy in a post-gdpr world," in *International Conference on Passive and Active Network Measurement*. Springer, 2019, pp. 258–270.

[40] M. Nouwens, I. Liccardi, M. Veale, D. Karger, and L. Kagal, "Dark patterns after the gdpr: Scraping consent pop-ups and demonstrating their influence," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.

[41] C. Matte, N. Bielova, and C. Santos, "Do cookie banners respect my choice?: Measuring legal compliance of banners from iab europe's transparency and consent framework," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 791–809.

[42] K. M. Sathyendra, S. Wilson, F. Schaub, S. Zimmeck, and N. Sadeh, "Identifying the provision of choices in privacy policy text," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 2774–2779.

[43] V. Bannihatti Kumar, R. Iyengar, N. Nisal, Y. Feng, H. Habib, P. Story, S. Cherivirala, M. Hagan, L. Cranor, S. Wilson *et al.*, "Finding a choice in a haystack: Automatic extraction of opt-out statements from privacy policy text," in *Proceedings of The Web Conference 2020*, 2020, pp. 1943–1954.

[44] I. A. Bureau, "Transparency and consent framework," https://github.com/InteractiveAdvertisingBureau/GDPR-Transparency-and-Consent-Framework, 04 2018, accessed on 2023-03-26.

[45] P. Santhanam, H. Dang, Z. Shan, and I. Neamtiu, "Scraping sticky leftovers: App user information left on servers after account deletion," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2145–2160.

[46] K. Kollnig, P. Dewitte, M. Van Kleek, G. Wang, D. Omeiza, H. Webb, and N. Shadbolt, "A fait accompli? an empirical study into the absence of consent to third-party tracking in android apps," in *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, 2021, pp. 181–196.

[47] T. T. Nguyen, M. Backes, and B. Stock, "Freely given consent? studying consent notice of third-party tracking and its violations of gdpr in android apps," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2369–2383.

[48] S. Koch, B. Altpeter, and M. Johns, "The ok is not enough: A large scale study of consent dialogs in smartphone applications," in *32st USENIX Security Symposium (USENIX Security 23)*, 2023.

[49] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: Towards automating risk assessment of mobile applications," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 527–542.

[50] W. You, B. Liang, W. Shi, P. Wang, and X. Zhang, "Taintman: An art-compatible dynamic taint analysis framework on unmodified and non-rooted android devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 1, pp. 209–222, 2017.

[51] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "Android taint flow analysis for app sets," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*, 2014, pp. 1–6.

[52] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "Appintent: Analyzing sensitive data transmission in android for privacy leakage detection," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, 2013, pp. 1043–1054.

[53] H. Fu, Z. Zheng, A. K. Das, P. H. Pathak, P. Hu, and P. Mohapatra, "Flowintent: Detecting privacy leakage from user intention to network traffic mapping," in *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2016, pp. 1–9.

[54] S. Xi, S. Yang, X. Xiao, Y. Yao, Y. Xiong, F. Xu, H. Wang, P. Gao, Z. Liu, F. Xu *et al.*, "Deepintent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2421–2436.

[55] L. Yu, X. Luo, J. Chen, H. Zhou, T. Zhang, H. Chang, and H. K. Leung, "Ppchecker: Towards accessing the trustworthiness of android apps' privacy policies," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 221–242, 2018.

[56] 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*, 2018, pp. 37–47.

Appendix A. Email Notification Template

Dear $developer team,
We are a team of academic researchers from $affiliation, conducting a research project on analyzing inconsistency between privacy setting and third-party data collections in mobile apps.

* Please note that this email is part of an academic research project and is not meant to sell any products or services.*

Based on our analysis, your app (pkgName) has provided a privacy setting to users which declares to stop some data collections of third-party libraries (we call it withdrawal). However, we found your app didn't stop these third-party data collections as you declared when users made their decision. That is, your app still shares user data with third-party libraries even after the user explicitly withdraws, which may lead to a violation of privacy regulations.

In order for you to inspect the problem, we have prepared a detailed report on such potential violation that we have detected (please refer to the attachment for details).

Please note that we don't offer a conclusive legal assessment or consultancy on an individual app's compliance. Instead, we aim to enable developers to address the issues before other parties might take any legal actions. As this email is part of a research project in which we are trying to understand why such inconsistency exists, it would be immensely helpful to provide us with feedback regarding your app.

(1) Were you aware of such inconsistency? Do you have a clear idea of which third-party data collection should be stopped in order to be consistent with your privacy settings?

(2) During the development process, were you aware of the methods to withdraw third-party data collection? And are there specific reasons why you fail to stop corresponding data collection?

(3) Are there any changes you plan to apply to remedy the outlined issues? What type of support (e.g., documentation or automated tools) can we provide would benefit you to address such issues?

If you have further questions or wish not to receive any further communication, please contact us, and we will promptly follow the request.

Best regards,
$Research Team

Disclaimer: This study is part of a research project of the $affiliation. The collected information will be used for scientific purposes only. Your responses are pseudonymous. We do NOT collect any personal information, publicize or perform any actions against your apps, and your company.

Appendix B. Meta-Review

B.1. Summary

This papers proposes a new tool named MOOChecker which leverages a static taint flow analysis to first detect opt-out buttons and the determine both code and data flows which divert based on that choice. Together with a set of known APIs which relate to opt-out in 200 major libraries, the authors conduct an analysis of real-world apps. In doing so, they found around 40% of 3.8k which use opt-out options to have inconsistencies, i.e., do not opt-out for all used libraries even if the user chooses so.

B.2. Scientific Contributions

- Independent Confirmation of Important Results with Limited Prior Research
- Creates a New Tool to Enable Future Science
- Identifies an Impactful Vulnerability

B.3. Reasons for Acceptance

1) The MOOChecker tool finds 176 manually-confirmed, zero-day opt-out inconsistencies. It therefore expands on prior work in the space of GDPR/CCPA violations.
2) MOOChecker provides a valuable basis for future work, in particular related given that prior work has operated in dynamic fashion which lacks coverage and interaction with the apps.
3) The paper highlights a problem space for future work, both in terms of vetting libraries and aiding developers.

B.4. Noteworthy Concerns

One major concern shared by several reviewers is the potential for over-reporting of findings. This is because unless there is a clear usage of the data by the recipient (i.e., they are a data controller rather than merely a processor), consent might not be required or not be revokable. Nevertheless, the PC felt the paper's technical contribution is meaningful to illuminate the problem space further.