Abandon All Hope Ye Who Enter Here: A Dynamic, Longitudinal Investigation of Android's Data Safety Section

Ioannis Arkalakis
Technical University of Crete
iarkalakis@tuc.gr

Michalis Diamantaris
Technical University of Crete
mdiamantaris@tuc.gr

Serafeim Moustakas
Technical University of Crete
seraf.moustakas@gmail.com

Sotiris Ioannidis
Technical University of Crete
sioannidis@tuc.gr

Jason Polakis
University of Illinois Chicago
polakis@uic.edu

Panagiotis Ilia
Cyprus University of Technology
panagiotis.ilia@cut.ac.cy

Abstract

Users' growing concerns about online privacy have led to increased platform support for transparency and consent in the web and mobile ecosystems. To that end, Android recently mandated that developers must disclose what user data their applications collect and share, and that information is made available in Google Play's *Data Safety* section.

In this paper, we provide the first large-scale, in-depth investigation on the veracity of the Data Safety section and its use in the Android application ecosystem. We build an automated analysis framework that dynamically exercises and analyzes applications so as to uncover discrepancies between the applications' behavior and the data practices that have been reported in their Data Safety section. Our study on almost 5K applications uncovers a pervasive trend of incomplete disclosure, as 81% misrepresent their data collection and sharing practices in the Data Safety section. At the same time, 79.4% of the applications with incomplete disclosures do not ask the user to provide consent for the data they collect and share, and 78.6% of those that ask for consent disregard the users' choice. Moreover, while embedded third-party libraries are the most common offender, Data Safety discrepancies can be traced back to the application's core code in 41% of the cases. Crucially, Google's documentation contains various "loopholes" that facilitate incomplete disclosure of data practices. Overall, we find that in its current form, Android's Data Safety section does not effectively achieve its goal of increasing transparency and allowing users to provide *informed* consent. We argue that Android's Data Safety policies require considerable reform, and automated validation mechanisms like our framework are crucial for ensuring the correctness and completeness of applications' Data Safety disclosures.

1 Introduction

Android is the most prevalent mobile operating system, powered by an ecosystem of diverse devices and millions [9] of applications. Throughout its lifetime, Android has undergone major changes aiming to address its underlying security and privacy issues. In particular, one of its most critical components that has received much scrutiny from the research community is the permission system, and a large body of work has explored the permission system's limitations and proposed modifications and countermeasures [46, 69, 74, 76]. This has resulted in Android's permission system evolving significantly over time and deviating from its original design.

While early Android versions required users to accept *confusing blocks of information* (i.e., permissions) prior to installing an application [39], later versions allowed users to accept or reject permission requests at runtime. Prior to installing an app, Android users can access information pertaining to the app's permissions in the Google Play store. This permission list is automatically generated during the app's vetting process by parsing the app's Manifest file. Nonetheless, many users may still not fully grasp how permissions work and what information is collected by a granted permission. In fact, Eling et al. [34] showed that users are more likely to deny a permission request when a detailed description of the accessed personal information is provided. Moreover, a study on the runtime permission model [70] found that misunderstandings are common among users, and that most are unable to accurately infer the resources that are protected and the actions that are permitted by each permission group.

To further simplify permissions, and following Apple's Privacy Nutrition Labels for iOS [61], Google recently introduced the *Data Safety* section which allows applications to disclose the personal data that they collect and share. The Data Safety section, which came into effect in July 2022 [11], requires developers to complete a comprehensive Data Safety form and declare all the data that their application collects or shares. While Google Play's permission list section was removed from the official store [18], considerable pushback from the developer community showed that the permission list is useful and shortly after was reinstated [16]. Currently, both the Data Safety information and permission list remain available in the Google Play store.

Even though the information provided in the Data Safety section can help users better understand how their personal data is being used, it relies solely on the developer to declare what data the application will collect and share. Essentially, this approach considers developers to be truthful and transparent (which may not always be the case), as well as fully cognizant of the data their app collects. In reality, this can be complicated by embedded third-party libraries, since developers may not be fully aware of the embedded code's functionality and its privacy implications [41]. As developers already spend considerable effort handling third-party libraries' privacy issues [75], being responsible for ensuring the accuracy of the Data Safety section can be challenging.

To make matters worse, the official Data Safety documentation [11] outlines a set of different policies that determine when data collection and sharing should be disclosed in the Data Safety section. We find that these policies are relaxed in many cases and, in general, can be vague and confusing. This not only inhibits users from being informed about the collection and handling of their data, but may also affect developers' ability to accurately declare all relevant information. Importantly, these policies also create "*loopholes*" that allow developers to misuse the system and deliberately avoid declaring certain pieces of information.

Motivated by these observations, we design a methodology and analysis pipeline that searches for discrepancies between the information declared in the Data Safety section on Google Play and the actual information that Android apps collect and share, as well as the extent to which such violations occur. To that end, we develop a framework that collects applications with their metadata and Data Safety sections from the Play Store, and dynamically exercises them to identify the information that each app accesses and shares over the network. Our framework monitors all API methods that lead to sensitive personal information or device data, which we have manually mapped to the corresponding Data Safety labels, and dynamically traverses the apps to trigger these calls. Furthermore, our tool identifies and interacts with in-app consent dialog boxes, and tests them under five different scenarios that explore how apps' data practices are affected by user consent.

Using our framework we conduct the first, to our knowledge, automated analysis of Android's Data Safety section. Our in-depth, longitudinal analysis reveals a multitude of discrepancies and widespread violation of data transparency. We uncover severe inaccuracies in apps' Data Safety sections, with 69.2% of apps not fully disclosing the data they collect and 43.1% the data that they share. Overall, we find that 81% of the tested apps contain at least one discrepancy in their Data Safety section, and that issues persisted for an entire year after the Data Safety section was made mandatory. Furthermore, we find that while in 59% of the cases the discrepancies originate from embedded third-party libraries, app developers are almost equally complicit in not accurately disclosing their own data practices. The implications of our findings are further exacerbated by the fact that applications not only violate Google's Data Safety policies, but also violate Google's policy governing consent requirements. We argue that Google's vague and ambiguous policies can lead to confusion, while also enabling misuse and the surreptitious exfiltration of personal data, thus significantly undermining the potential benefits of the Data Safety section.

The key contributions of our work are the following:

- We analyze the official Data Safety documentation and identify instances of relaxed policies that can confuse users and/or developers, which also enable misuse scenarios with severe ramifications. We then create a mapping of Data Safety labels to the corresponding function calls, which is necessary for analyzing them within the broader context of Android privacy research.
- We develop a dynamic analysis framework and a pipeline that automatically identifies discrepancies between apps' runtime behavior and their Data Safety section, under different user consent scenarios.
- We conduct an extensive longitudinal investigation of Data Safety discrepancies in popular Android apps. Our study uncovers alarming practices and rampant violations of data transparency and user consent. We have disclosed our findings to Google, and will facilitate additional research by publicly sharing our code and data.[1]

2 Background & Motivation

Google Play includes a dedicated section for each app that informs users about the data that the app collects and shares. The Data Safety section consists of three parts, specifically the *Data Collected*, *Data Shared* and *Security Practices* parts. The *Data Collected* part includes information about the data that the respective app collects, while the *Data Shared* part shows which of the collected data is being shared with third parties [11]. Google Play displays the Data Safety section information regarding the data that is collected and shared by using a list of predefined labels (e.g., `Location, Contacts, Photos or videos`). The *Security Practices* part informs users whether the app uses encryption when transmitting data, if data deletion is possible, and whether the app has been independently reviewed against a global security standard [36]. Moreover, the Data Safety section is mandatory for all apps, and all relevant information needs to be provided by developers when submitting their apps to Google Play, even if no data will be collected or shared.

The Play Console Help [1] provides guidelines for developers, and outlines the policies to be followed for completing an app's Data Safety section. However, our analysis of the policies described in the "User Data" part of the documentation [37], which determine when data collection and sharing should be disclosed, are vague in places and can enable problematic practices. For instance, in the case of *Data Collected*, the documentation specifies that it concerns data transmitted

[1] https://github.com/GiannisArk/USENIX24_DataSafety

off a user's device, with the exception of (i) data that is accessed and only processed locally on the user's device, and (ii) data that is sent off the user's device but is end-to-end encrypted (i.e., messaging apps). A crucial detail is that data that is transmitted off device but only processed "ephemerally" does not need to be disclosed in the Data Safety section.

Moreover, in the case of *Data Shared* policies that concern data transmitted from an app to a third party, the documentation presents several types of data transfers that do not need to be disclosed. These include data transfers to (i) "service providers" that process data on behalf of the developer, (ii) "legal purposes" such as legal obligation or government requests, (iii) "user-initiated action or prominent disclosure and user consent", and (iv) "anonymous data" that can no longer be associated with an individual user.

These policies are overly permissive and potentially unclear, as they rely on complicated terminology and definitions, which may allow apps to exploit "*loopholes*", as was also noted recently by Mozilla [12]. Even more problematic is the policy that allows data transfers based on "user-initiated action or prominent disclosure and user consent" without the need to disclose these transfers. In other words, developers and third-parties can access and transmit user data without declaring it in the Data Safety section if the user consents through the in-app user dialog at runtime. However, prior work has highlighted the use of dark patterns in runtime consent dialogues [72], which can affect users' decision making [52].

In summary, our analysis of Google's policies reveal various opportunities for misinformation and incomplete disclosure of applications' data practices. This motivates our in-depth investigation of Android applications' data practices and their correlation to the corresponding Data Safety section disclosures by app developers. We believe that all data regulated by privacy laws (e.g., GDPR) needs to be declared in the Data Safety section when collected or shared by apps, and Google should not be introducing additional loopholes and exemptions based on arbitrary heuristics. While exemptions to such regulations exist, the well-defined exemption criteria defined by GDPR do not typically cover the nature of the Android apps that we have analyzed, or the specific behaviors that we have uncovered. Throughout the paper we consider as a *discrepancy* any case where data is accessed or shared at runtime without having been disclosed in the app's Data Safety section. Consequently, we use the term data leak, when referring to apps sharing information without disclosing it.

3 System Design and Implementation

Here we describe our methodology for identifying data that is being collected and shared by Android apps, and provide details regarding our framework's implementation. Figure 1 provides an overview of its components and analysis pipeline. Our system consists of several components that allow us to (i) monitor API methods (e.g., framework functions like the Telephony API) and map them to Data Safety labels (i.e., types of information) for identifying discrepancies in the *Data Collected* segment, and (ii) monitor network traffic for identifying discrepancies in the *Data Shared* segment of the Data Safety section. Our framework builds upon existing tools [32], which were modified and extended to match our needs.

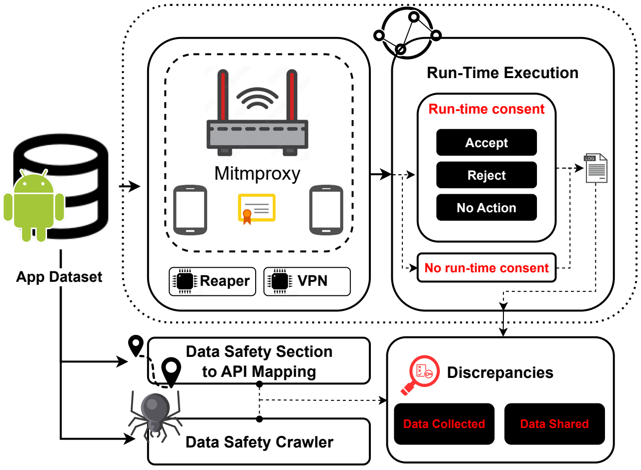

Figure 1: Overview of our framework and analysis pipeline.

Android API monitoring. We monitor API methods by utilizing Reaper [32], a dynamic analysis system that traces permissions requested by apps and non-permission-protected functions that yield Personally Identifiable Information (PII). Since permission mappings do not exist for Android 12 (which was the latest stable version during the time of our experiments), we used the permission mappings provided by DYNAMO [27] for Android 10 as a starting point for constructing our own list of permission-protected functions.

Furthermore, due to the constant evolution of permission-protected APIs and their permission specification, we also followed the methodology employed by APER [77] for identifying any missing or false entries for our target version. Specifically, we traversed all the Java source files for Android 12 and identified all the API methods that have the `@RequiresPermission` annotation tag to specify their permission requirements. Our final list of permission mappings includes all the permission-protected functions of DYNAMO and those that have been identified statically from the source code and the documentation. When encountering differences in the permission annotation of functions between DYNAMO and the permission annotation identified through the Java source files, we used the permission annotation of the latter. Moreover, as apps and third-party libraries are prone to accessing PII from functions that are not permission-protected, our framework also incorporates known PII functions (see Appendix A) from prior work (e.g., [32, 63]).

Data Safety section to API Mapping. The Data Safety section uses predefined labels (e.g., `Location`, `Messages`, `Photos or videos`, `Contacts`) as a high-level, user-

Table 1: Data Safety labels mapped to the functions called by apps in our datasets (coarse- and fine-grained categories).

Category	Sub-categories
App info and performance	Diagnostics, Other app performance data
Location	Approximate location, Precise location
Personal info	Name, Phone number
Photos or videos	Photos
Audio files	Music files
Files and docs	
Contacts	
App activity	
Device or other IDs	

friendly way of describing the information that each Android app can collect and share. Since our system monitors access to function calls (regardless of them being permission-protected or not), we followed a specific process for mapping each function call to the corresponding Data Safety labels that accurately describe the information accessed, thus also indirectly categorizing the collected and shared data. As a first step in this process, we automatically exercised all the apps in our datasets, and out of the 6,955 function calls that our system monitors, we selected 295 functions that were actually accessed by the apps we exercised. This significantly narrowed down the functions that require Data Safety labeling, to only those that were actually accessed by the apps in our datasets. Next, we manually assigned Data Safety labels to these 295 functions and removed any functions that do not require a Data Safety label (e.g., `SystemVibrator.vibrate()`). We took into consideration the relevant permissions and examined the official Data Safety developer guideline documentation [30] and the Java source code of the respective function calls. For example, `getLastKnownLocation()` and `getCurrentLocation()` are mapped to the Location, Precise location and Approximate location labels. Our Data Safety mappings consist of a list of 94 functions that are mapped to 17 different Data Safety labels, with 9 of the labels corresponding to coarse-grained categories and 8 to fine-grained subcategories (e.g., category: App info and performance, with sub-categories: Diagnostics, Other app performance data). All the labels used in our mappings are shown in Table 1. We note that Table 1 does not include labels that relate to personal user identifiers such as name, surname and email, which we detect through network traffic monitoring.

To ensure the accuracy of our manual labeling, this process was performed independently by two of the authors, who subsequently compared their function-label mappings. In cases where the two researchers' mappings did not perfectly align, the two researchers conferred and re-examined the specific cases along with a third co-author. Overall, only a few mismatches occurred between the two labellers and these were cases of assigning labels of a different granularity (e.g., Location and Precise Location). These were all rectified during the re-examination process, based on a majority vote. More specifically, out of the 295 functions examined, the two labellers had a precise agreement (i.e., all assigned labels match) for 242 functions (82.03%). These also include cases where the labellers decided that no Data Safety label is required, and marked it as *NoLabel*. In 8 cases (2.71%) the researchers had assigned some matching labels, but not all labels perfectly matched. For example, for the `getLine1Number()` one researcher assigned the Personal info and Phone number labels, and the other assigned only the Personal info label. After conferring, both labels were assigned to this function. Furthermore, in 44 cases (14.91%) only one of the two labellers assigned a label, while the other one marked them as *NoLabel*. These were cases of uncertainty, and were all discussed and addressed in the presence of a third researcher. Finally, we had a single case of a label category disagreement, for `getDefaultOutgoingPhoneAccount()`, where one labeller assigned the Contacts label and the other the Personal info one. Since this function returns a `PhoneAccount` object we decided that the appropriate label should be Contacts.

Runtime execution. We utilize Reaper's UIHarvester [32] to traverse apps and retrieve all the elements displayed on the device's screen. We have extended UIHarvester to retrieve not only the native Android components but also any web components, since applications may display content inside a WebView element. As stated in the Data Safety's privacy and security practices [36], developers do not need to declare any information in the *Data Shared* part of the Data Safety section if the app requests a runtime user consent (not to be confused with Android's runtime permissions) according to the requirements of Google Play's User Data policy [37].

Our runtime execution methodology takes this policy into consideration and identifies runtime consent elements (e.g., GDPR cookie consent banner) while exercising the app, enabling our subsequent differential analysis. Specifically, our framework identifies any runtime consent elements that exist, and performs two different executions of the app: one for accepting (i.e., providing consent) and one for declining. Additionally, we check whether the accept or reject button has been successfully selected by restarting the app and checking if the runtime consent dialogue persists. In cases where the app does not have a reject button or terminates after the reject button is selected without allowing any further interaction, we terminate the analysis for this app (Figure 1 - No Action). For every execution, we log all the information that the app collects and shares. Interestingly, we found a significant number of apps that contain discrepancies in the Data Safety that also violate the consent requirements of Google Play's User Data policy [37] by collecting and sharing sensitive info even if the user rejects the runtime consent, as we detail in Section 4.

Identifying consent dialogues. Our study analyzes Android apps for data violations and discrepancies in their Data Safety section by taking into consideration the app's request for runtime consent, therefore creating different scenarios for each app execution, based on the user's selection. As such, our system includes a component that identifies *View* and *We-*

bView elements that display runtime consent dialogues, using a string-matching-based approach with a predefined list of keywords. We populated this list of keywords by manually interacting with 100 apps that contain runtime consent dialogues. We verified that our list of keywords is sufficient for identifying runtime consent dialogues by verifying them against a *different* random set of 100 apps. In our manual verification we found that we are able to identify all runtime consent dialogues, even in cases where the app shows a pop-up dialog that informs the user about the data they collect by providing a link to their privacy policy, but without providing an agree or reject button. Moreover, as runtime consent dialogues may require the user to click different checkboxes before being able to press the accept button, we also incorporated this functionality into our module.

Data Safety crawler. We created a Python-based crawler that parses the Google Play web page on a daily basis and logs all the information provided by the Data Safety section for each application in our dataset starting July 29, 2022. Our crawler also logs additional information for each app, such as the developer's name, permissions, and number of downloads.

Discrepancies. Our framework dynamically analyzes apps and logs the data that they collect and share using runtime instrumentation. In our study, we process each app's logs and identify discrepancies between the runtime behavior and the data declared in the Data Safety section. While we crawl each app's Data Safety section on a daily basis, we compare the results of the runtime execution with the information provided in the app's Data Safety section on the day that we downloaded the app's apk file. Furthermore, as we detail in our analysis in Section 4, we find that the same discrepancies tend to persist across different app executions for long periods of time. Overall, we identify *Data Collected* discrepancies by monitoring function calls in the operating system, and *Data Shared* discrepancies by monitoring network traffic.

Testbed configuration. Our experimental setup consists of two Google Pixel devices (Pixel 4a and Pixel 6) running Google's AOSP version 12, and a proxy server using `mitm-proxy` [26]. Magisk [78] was used to root the devices and install additional modules. The devices were configured with the LSPosed [51] framework that supports the latest Android versions and is compatible with Reaper's Xposed [2] modules. We intercepted network traffic by configuring the devices with the mitm proxy's root certificate, by installing it into the Android's system store. Moreover, we use Objection [10] to detect and disable SSL Pinning. At runtime, our framework installs and analyzes each application individually (e.g., install, analyze, clear app data, uninstall), and approves all the runtime permissions that the apps may request, using the "`adb install -g`" option. We modified UIHarvester [32] to be compatible with our devices' API and interacted with each app for five minutes using a breadth-first traversal strategy. Finally, we implemented a module for automating the login process by using Google's Single Sign-On whenever possible.

4 Measurements and Investigation

Here we present the findings from our large-scale study of the Data Safety section in the Android ecosystem.

App dataset. We created our application dataset based on free apps downloaded from the official Google Play market. We selected *up to* the top 250 apps (or as many as were available) from 20 different categories. Since our goal is to investigate how developers adhere to Google's Data Safety policies and how the data collected and shared may change over time, we gathered and analyzed four different versions for each app based on the time of download (i.e., September 2022, March/June/September 2023). Overall, we downloaded 4,986 unique apps from Google Play using the Raccoon [3] framework and performed a total of 21,202 executions across the different scenarios and VPN experiments.

The experiments performed in September 2022 were preliminary, aiming for an initial exploration of Data Safety discrepancies, and were performed using a prototype version of our system. These experiments only explored *Data Collected* discrepancies. We decided not to include new apps across datasets for our experiments and findings to remain consistent. Nonetheless, some apps that existed in the `Sep22` dataset were no longer available in the official Google Play Store in March and June 2023, and are not included in the respective datasets. Also, 225 apps have been removed from Google Play in September 2023 (out of which 213 had discrepancies during March's analysis). Therefore, `Sep22` dataset contains 4,986 apps, `Mar23` and `Jun23` datasets contain 4,157 apps and `Sep23` contains 3,953 from the original dataset. While we observe similar behaviors across all four datasets, our analysis here focuses mainly on `Sep23`, as that provides the most up-to-date view of how the Data Safety section is being used.

VPN dataset. We downloaded a subset of 100 apps from our initial dataset and analyzed them in several countries using a VPN service. As differences may exist between versions of an app distributed in different countries due to legislation, for each of these apps we visited the appropriate Google Play market for different countries and downloaded the apk file and crawled the information in the respective Data Safety section. Overall, our VPN-based analysis includes apps distributed in Canada, United States, Brazil, Iceland, Germany, Ukraine, Greece, Estonia, Kenya, Russia and India. Even though apps may be able to detect when a VPN service is used (e.g., GPS coordinates, nearby WiFi access points) and even prevent us from analyzing them (i.e., by geoblocking [45]), we did not detect such constraints for the majority of apps that we analyzed using VPN. Specifically, we only faced geoblocking restrictions in 18 apps (14 apps in Estonia, 2 apps in Iceland, 1 app in India and 1 app in Ukraine). Our VPN experiments took place between April 15 and May 2, 2023.

Data Safety section discrepancies. Table 2 summarizes our findings about apps that contain discrepancies between the run-time execution and their Data Safety section for different

Table 2: Apps with discrepancies for September 2022 (preliminary experiment), March, June and September 2023. *Any Discr* denotes apps with at least one type of discrepancy. *Data Collected* and *Data Shared* denote apps with discrepancies in the respective Data Safety section. Percentages are based on the total number of apps in the corresponding scenario.

		Preliminary exp.	March 2023			June 2023			September 2023		
Scenario		***Data Collected*** (September 2022)	***Any Discr***	***Data Collected***	***Data Shared***	***Any Discr***	***Data Collected***	***Data Shared***	***Any Discr***	***Data Collected***	***Data Shared***
All		2,365 (47.43%)	3,363 (80.89%)	2,871 (69.06%)	1,797 (43.23%)	3,354 (80.68%)	2,878 (69.23%)	1,749 (42.07%)	3,204 (81.46%)	2,731 (69.44%)	1,735 (44.11%)
(i) No runtime consent		2,065 (48.46%)	2,545 (79.18%)	2,169 (67.48%)	1,384 (43.06%)	2,579 (79.11%)	2,209 (67.76%)	1,378 (42.27%)	2,543 (80.02%)	2,156 (67.84%)	1,418 (44.62%)
runtime consent:	***(ii) Accept***	300 (41.37%)	815 (86.42%)	701 (74.33%)	402 (42.63%)	773 (86.17%)	669 (74.58%)	364 (40.58%)	657 (84.77%)	575 (74.19%)	300 (38.71%)
	(iii) Reject	26 (3.58%)	89 (9.44%)	78 (8.27%)	41 (4.34%)	67 (7.47%)	62 (6.91%)	27 (3.01%)	62 (8%)	58 (7.48%)	20 (2.58%)
	(iv) Reject not found	228 (24.07%)	570 (60.44%)	474 (50.26%)	275 (29.16%)	566 (63.1%)	477 (53.17%)	254 (28.31%)	468 (60.38%)	397 (51.22%)	210 (27.1%)
	(v) Reject & app exited	50 (6.89%)	78 (8.27%)	69 (7.31%)	40 (4.24%)	79 (8.8%)	67 (7.47%)	44 (4.9%)	77 (9.93%)	63 (8.13%)	42 (5.42%)

run-time consent scenarios and across three time periods. *All* lists all apps with discrepancies irrespectively of the run-time consent scenario (i.e., we observed discrepancies in at least one of the scenarios). *Any Discr* denotes apps with at least one discrepancy without distinguishing between data being collected or shared, while *Data Collected* and *Data Shared* denote apps with discrepancies in the respective Data Safety section. Overall, we find that 81% ($\sigma = 0.33$) of apps have discrepancies, and that the percentages are high across all datasets (`Mar23`: 80.89% - `Jun23`: 80.68% - `Sep23`: 81.46%). These discrepancies persist for an entire year after Data Safety was made mandatory, highlighting the dire need for an effective automated validation mechanism that will ensure the accuracy and completeness of the Data Safety section.

Level of non-compliance. Figure 2 presents the extent of non-compliance for apps analyzed in March, June and September 2023. For determining compliance we identify which labels each app should have declared in the Data Safety section, based on their run-time behavior, in order to not have any discrepancies. Essentially, this number is calculated by considering both the labels that have been already declared and those that lead to discrepancies. As can be seen, 794 (19.1%), 803 (19.32%) and 729 (18.54%) apps are fully compliant as they do not have any discrepancies, indicating that they declare all the information that they collect and share in the Data Safety section. On the other hand, we found 579 (13.93%), 567 (13.64%) and 504 (12.81%) apps that appear to be entirely non-compliant, across the three datasets, as they do not declare any information, and hence all functions that access or transmit user or device information result in discrepancies. The remaining apps (i.e., 2,784 (66.67%), 2,787 (67.04%) and 2,700 (68.65%) respectively) have declared some information, but are not yet fully compliant as they still have discrepancies. Overall, apps with discrepancies have an average compliance level of 63.9% (`Mar23`: 63.16% - `Jun23`: 63.68% - `Sep23`: 64.85% – $\sigma = 0.71$). Based on these findings we argue that in the majority of cases Android's Data Safety section can not be trusted, as it contains inaccuracies or, one could argue, misinformation that will actively mislead users about what data will be collected and shared. As such, even cautious users may be unable to make truly informed decisions when granting permissions to apps.

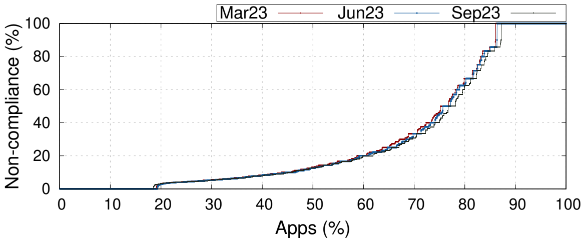

Figure 2: Level of non-compliance for apps in the 3 datasets. Apps in each dataset are sorted in ascending order based on their level of non-compliance.

In-app consent. As apps embed third-party library code and display content from the web, we have also analyzed our datasets with regards to the run-time consent dialogues that the apps present and the possible user options. Overall, our analysis is based on five scenarios, as shown in Table 2. The first scenario refers to (i) apps that do not present any run-time consent dialogue at all, while the remaining four scenarios provide a run-time consent dialogue and consider the respective users' options. Specifically, these scenarios are: (ii) "*Accept*" in the case where the user provides consent, (iii) "*Reject*" when the user declines to provide consent, (iv) "*Reject not found (No action)*" where the reject button was not found in the device's screen and the analysis terminated, and (v) "*Reject & app exited (No action)*" in the case where the reject button was pressed and the application did not allow further interaction. It is worth noting that all apps listed under the scenarios `scenario-iv` and `scenario-v` contain discrepancies and violate Google's policies concerning the user consent and the consent requirements.

We observe that the vast majority of apps (`Mar23`: 79.18% - `Jun23`: 79.11% - `Sep23`: 80.02% – $\mu = 79.43\%$, $\sigma = 0.41$) contain at least one discrepancy in their Data Safety sections and do not have a run-time consent dialogue (`scenario-i`). While our original assessment of Google Play's User Data policies [37] was that they are more permissive than they should be, due to allowing data transfers based on"*user-initiated action or prominent disclosure and user consent*", our analysis reveals that the situation is more problematic than we had expected it to be. Apart from omitting to disclose in their Data Safety section information regarding the data that they collect and share, these apps do not inform the user about run-time data accesses at all, and do not give them the ability to consent (or deny consent). Specifically, we observe that in all datasets the percentage of apps with discrepancies in *Data Collected* (Mar23: 67.48% - Jun23: 67.76% - Sep23: 67.84% – $\mu = 67.69\%$, $\sigma = 0.15$) and *Data Shared* (Mar23: 43.06% - Jun23: 42.27% - Sep23: 44.62% – $\mu = 43.31\%$, $\sigma = 0.97$) is still significantly high.

For apps that provide a run-time consent dialogue and the user agrees to the consent (scenario-ii), we found that 85.79% of the apps (Mar23: 86.42% - Jun23: 86.17% - Sep23: 84.77% – $\sigma = 0.72$) access and share data not declared in the Data Safety section. While these apps fall in line with Google's policies [11], we believe that such instances of relaxed policies can be abused and mislead users about the data that the app collects and shares, and that this information needs to be disclosed in the appropriate Data Safety section.

For apps that provide a consent dialogue but the user declines the consent (scenario-iii), we found that 8.3% of the apps (Mar23: 9.44% - Jun23: 7.47% - Sep23: 8% – $\sigma = 0.83$) contain a discrepancy in their Data Safety section. We note that apart from having discrepancies between the run-time execution and the Data Safety section, these apps also violate the "*User-initiated action or prominent disclosure and user consent*" policy. We identified 61.31% of apps (Mar23: 60.44% - Jun23: 63.1% - Sep23: 60.38% – $\sigma = 1.26$) in scenario-iv that also violate Google's policy governing consent requirements [37]; this policy states that consent "*must be granted by the user before your app can begin to collect or access the personal and sensitive user data*". These apps present a run-time consent dialogue but they do not display an agree button and, to make matters worse, collect data even if the user does not interact with and exits the app. Finally, we found 9% of apps (Mar23: 8.27% - Jun23: 8.8% - Sep23: 9.93% – $\sigma = 0.69$) in scenario-v that collect data before terminating, after the user has pressed the reject button. These apps contain Data Safety discrepancies and also violate both of the aforementioned consent policies.

Overall, for the three "Reject" scenarios (scenario-iii, scenario-iv, scenario-v), we found that 78.61% ($\sigma = 0.54$) of the apps that present a consent dialogue have discrepancies in their data collection and sharing practices (Mar23: 78.15% - Jun23: 79.37% - Sep23: 78.31%). While apps should always respect users' run-time choice concerning accessing and sharing of their data, our experiments reveal that this is not always the case, thus allowing apps to exploit loopholes in Google's current policies that govern data disclosure.

Data Collected discrepancies. Table 6 in Appendix A breaks down the discrepancies between apps' run-time behavior and their Data Safety labels for every run-time scenario across all datasets, and provides aggregated results for the functions that lead to the discrepancies along with the respective Android permissions. We observe that discrepancies may originate from functions that require "dangerous" permissions, such as ACCESS_(COARSE\FINE)_LOCATION, READ_SMS, READ_PHONE_NUMBERS, READ_EXTERNAL_STORAGE, and READ_PHONE_STATE. Moreover, we observe a large number of apps that access location information without declaring any of the corresponding Data Safety labels: Location, Approximate location, or Precise location. This is also true for 45-69 apps (across datasets) accessing precise location, where the user had no interaction with the app (scenario-iv), and 15-20 apps (across datasets) in which the user declined to provide run-time consent and the app did not allow further interaction yet still collected the location information (scenario-v).

Personal & sensitive data. A large number of apps access sensitive personal information that falls into the Data Safety labels of "Device or other IDs" and "App info and performance" without declaring them. Previous work has shown that this type of data can be used to accurately track users (e.g., [32, 47, 63]). Our experiments show that this behavior became more common between March - September 2023 in apps that do not provide a run-time consent dialogue (scenario-i). Additionally, we found one app that collects and shares the user's contacts in every dataset. We also observe apps accessing functions that yield persistent device identifiers (e.g., IMEI/MEID, IMSI, SIM, build serial), which requires the READ_PRIVILEGED_PHONE_STATE permission in the latest Android versions. Even though this permission can be only granted to apps signed with the platform key and privileged system apps [5] (starting from Android API 29), several persistent identifiers (e.g., IMEI) can still be accessed without this permission if, for example, the app is the default SMS role holder [13]. We did not set apps as the default handler, and in our devices' version these calls are automatically declined and no value is returned; however, this will not be the case for older Android versions. Moreover, we found a small but non-negligible number of apps (i.e., 15) that access extremely sensitive user data such as photos, documents, audio files and contacts without declaring the appropriate labels.

API Methods. We identified and analyzed which functions apps are accessing without declaring the appropriate *Data Collected* labels. Figure 3 shows the number of apps that access functions that resulted in a *Data Collected* discrepancy in the Sep23 dataset across all run-time consent scenarios. We observe that the majority of functions are called by apps that do not have a run-time consent dialog. Some of these functions return device specific identifiers (e.g., getDeviceID(), getIMEI(), getNetworkOperator()) while many apps use functions that reveal the user's location (e.g., getLastknownLocation()). Figure 3(c) shows the aggregated results of apps accessing API methods resulting in Data Safety discrepancies for all run-time scenarios where the consent is rejected (i.e., scenarios iii, iv and v). Surprisingly, we observe that the user's decision concerning the run-time consent (i.e., accept or reject) makes almost no difference to whether the user's data is being collected and not disclosed in the Data Safety section,

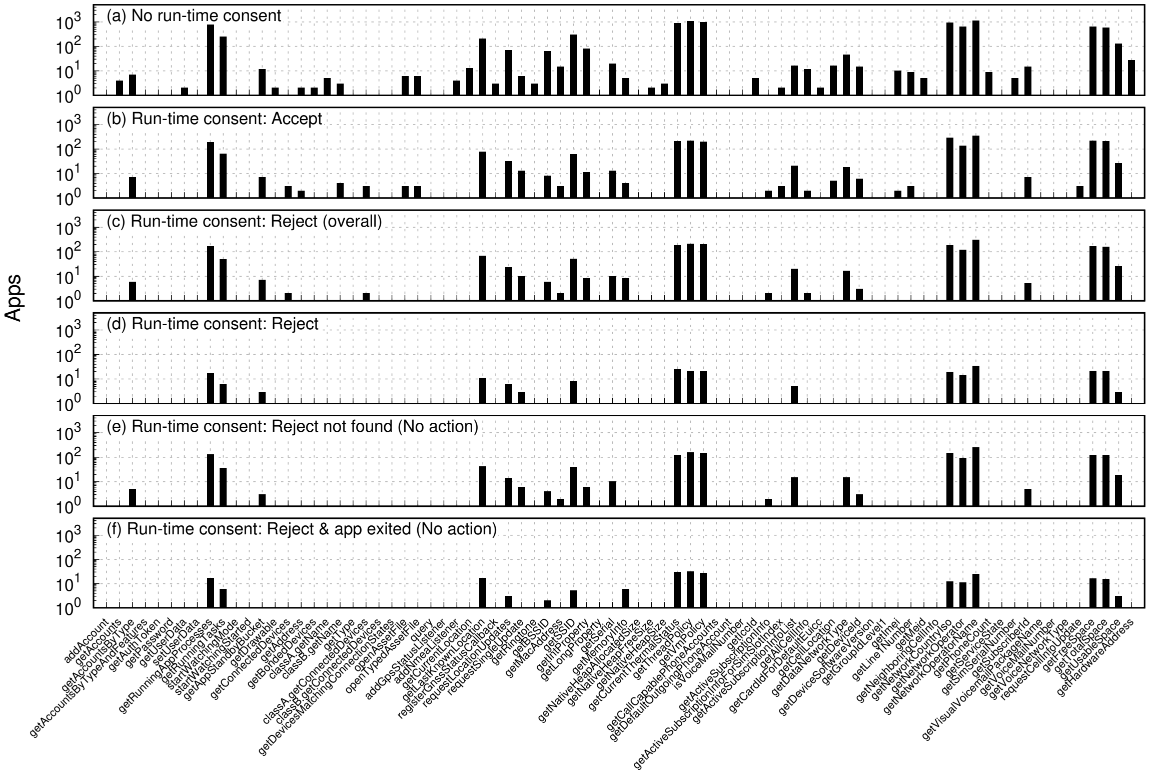

Figure 3: Apps accessing functions that resulted in Data Collected discrepancies for all run-time scenarios for `Sep23`.

as apps ignore the user's reject option (Figure 3(c)). However, for completeness, in Figure 3 we also present apps with discrepancies for all the different reject scenarios (Figure 3(d), Figure 3(e) and Figure 3(f)).

Moreover, we found many apps that use the `getCurrentThermalStatus()` function without declaring the "App info and performance" Data Safety label. This function returns a value that represents the thermal status of the device. Even though similar device characteristics have previously been used for tracking devices (e.g., battery API [59]), verifying whether this is the case here, falls outside the scope of our work. We also observe apps that do not disclose access to the `getSSID()` method which returns the service set identifier of the current WiFi network. Previous work [56] has demonstrated that the list of nearby access points (e.g., the `getScanResults()`) can accurately locate users and, as such, Android protected these functions with the location permission. As mobile devices frequently change access points, third-parties can periodically access this function and gather the list of linked access points in order to locate users. Additionally, we observe apps accessing the BSSID identifier using the `getBSSID()` method, which allows linking devices that share the same access point, thus enabling cross-device tracking [65]. Overall, we identified apps accessing 13,004, 12,860, and 11,645 functions without declaring the respective Data Safety labels in the `Mar23`, `Jun23` and `Sep23` datasets, respectively.

Persistent discrepancies. We investigated whether apps with discrepancies have been corrected between March's and June's version. We found that only 81 apps corrected their discrepancies and are compliant with Data Safety policies. Unfortunately, discrepancies in 3,282 apps remain the same across a period of almost three months. To make matters worse, we identified 72 apps that did not have any discrepancies in March, but had discrepancies in June's analysis. Accordingly, we identified 179 and 270 apps with discrepancies not declaring information in the *Data Collected* and *Data Shared* sections in September 2023 respectively, but having the appropriate labels in March 2023. We believe that these results could indicate confusion amongst developers and the information they are required to disclose. Google should provide better guidelines for completing the Data Safety section and enforce penalties for apps that fail to comply.

Data Shared discrepancies. Our system captures and logs network traffic during the execution of each Android app and identifies *Data Shared* discrepancies following a string-matching based approach that attempts to match device-specific information and keywords to the network traffic logs. For this, we developed a mock application that executes all the functions that we found in our preliminary experiments that can result in *Data Collected* discrepancies, and ran this mock application with both of our testbed devices. We construct a list of device-specific information for our devices (e.g., Advertising ID, GSF ID), and specifically crafted data values (e.g., email, name, surname), so that our system can detect when these identifiers are being exfiltrated over the network.

Since the values sent over the network can be in an encoded form, our system checks for common transformations during the matching process (e.g., `base64`, `HEX`, `ROT13`, `SHA-1`, `SHA256`, `MD5`, `RIPEMD-160`, `Whirlpool`, and `BLAKE2`). Even though this approach is common for identifying network leaks [23, 31, 58], it is not able to handle all cases of leaked data, as data may be encrypted or heavily obfuscated. As such, our results for *Data Shared* discrepancies present a *lower bound*, and we consider an exhaustive investigation using differential analysis (e.g., following the approach of Continella et al. [25]) as future work. Nevertheless, our experiments still reveal that 42.07% - 44.11% of the apps in our three datasets contain at least one discrepancy in the *Data Shared* section.

Violations. Table 2 shows that many apps share data without declaring it in the Data Safety section. Similar to prior work that found GDPR violations in Android apps [58], we also identified cases of apps leaking data (which has not been declared in the Data Safety section) without the user's consent. We observe that apps violate the Data Safety policies, and also violate and ignore the user's choice for data sharing, resulting in major data leakage without the user's knowledge or consent. To this end, we argue that Google should revise the current Data Safety policies and mandate that any data being collected or shared must be declared without exception.

Data leaks. We identified that apps share without the appropriate disclosure various device characteristics, such as the AdvertisingID, BSSID, BuildNumber, DeviceName, Google Services Framework ID, MCC+MNC and SSID, as well as sensitive user information including Address, Contact Numbers, Email, Latitude, Longitude, Name and Surname, to 16,840 different domains. This information is extremely sensitive as it can be used to accurately track and deanonymize users. On average, apps with *Data Shared* discrepancies leak between two and three PIIs to different domains.

We use McAfee's real-time database [54] to classify all the domains that the apps share data with. Figure 4 shows the number of apps that share undisclosed information to different domain categories. We observe that the majority of apps send data to domains classified as "Internet Services", "Software/ Hardware" and "Content Server". While it is likely that these apps use cloud providers to perform some kind of operation on the data they collect, it is infeasible to actually verify what happens with the data after it has left the user's device. We also found many domains classified as "Web Ads" that may use such data for advertising and retargeted ads. This indicates that embedded third-party ad libraries may be responsible for undisclosed data sharing, as we further elaborate later on. Interestingly, we found 12 apps sending PIIs to 12 domains over HTTP even though their *Data Safety* section states that data is sent over a secure connection.

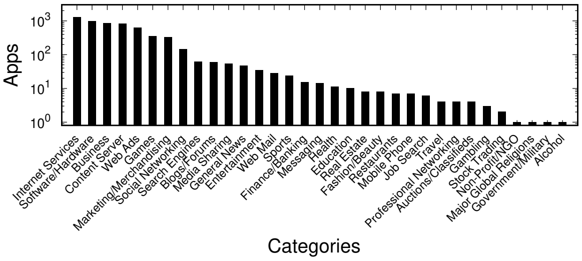

Figure 4: Apps with Data Sharing discrepancies sending data to different domain categories.

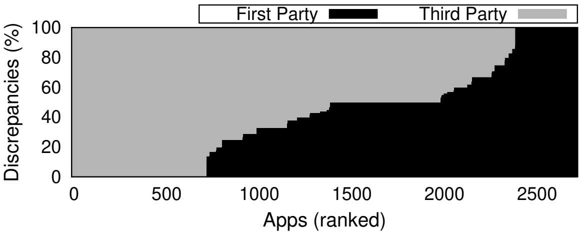

Figure 5: Percentage of discrepancies originating from first-party functionality, for the `Sep23` dataset.

Origin of discrepancies. Motivated by prior work [24, 32, 69] that differentiated between the permissions requested by the actual app and those requested by third-party libraries, we employed a previously proposed stacktrace-analysis methodology [32, 64] to infer which part of the application is responsible for the discrepancies we observe. As the stacktrace reveals the path to the source file and the package name responsible for the function call, we compiled a list of third-party libraries from [20, 48, 68] and distinguished between function calls originating from first-party and third-party functionality. Figure 5 shows the percentage of discrepancies for 2,731 apps in the September 2023 dataset, between the first-party and third-party functionality. We observe that while most of the discrepancies originate from third-party functionality (59%), the percentage of apps with discrepancies originating from first-party functionality (41%) is almost as common. Additionally, we found that for 12.34% of the apps, discrepancies originate only from the first-party functionality, while for 26.62% discrepancies originate only from third parties.

Our analysis shows that 91 libraries are responsible for the discrepancies originating from third-party functionality. The most popular among them is `android.gms` and is used by 1,163 apps, while `facebook`, `applovin` and `unity3d` are used by 828, 678 and 623 apps, respectively. Other popular libraries we found are `ironsource`, `fyber`, `vungle`, `adcolony`, `appsflyer`, `amazon` and `adjust`, all of which are used by more than 100 apps. In total we identified 11,183 function calls that originate from the aforementioned 11 libraries. On the other hand, 58 of the 91 libraries (63.73%) are used by five apps or less, further demonstrating the importance of third-party libraries accurately documenting the

Table 3: Apps with discrepancies across countries.

Country	Data Collected	Data Shared	Country	Data Collected	Data Shared
DEU	80 (80.00%)	81 (81.00%)	CAN	79 (79.00%)	85 (85.00%)
EST	69 (80.23%)	71 (82.56%)	IND	79 (79.80%)	85 (85.86%)
ISL	80 (81.63%)	77 (78.57%)	KEN	79 (79.00%)	83 (83.00%)
UKR	80 (80.81%)	83 (83.84%)	RUS	80 (80.00%)	83 (83.00%)
BRA	80 (80.00%)	82 (82.00%)	USA	79 (79.00%)	82 (82.00%)

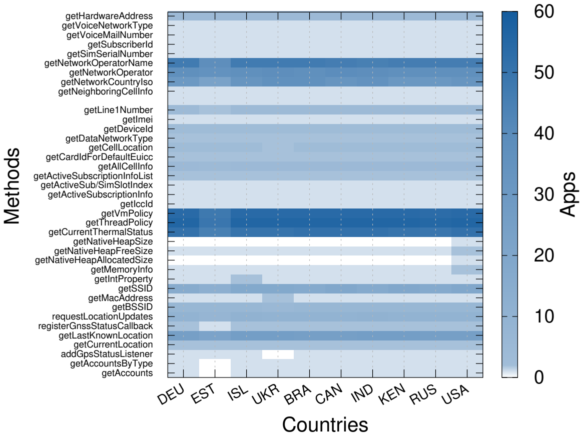

Figure 6: Data Collected discrepancies per country.

information they collect and enabling developers to declare this information appropriately in the Data Safety section.

There has been a long-standing debate regarding the privacy issues that arise form third-party libraries [41, 55, 75] and who is responsible for solving these issues. In that regard, we believe that Google's decision regarding developers being responsible for ensuring the accuracy of the Data Safety section places them in a precarious situation. This has also been argued previously [75] due to developers already struggling with handling the legal requirements for embedded libraries, as they have to invest a lot of time and effort to understand how libraries and ad networks handle privacy issues. Therefore, we believe that third-party libraries should explicitly mention in their documentation the data they collect so they can easily be listed by developers in the Data Safety section. Concerning discrepancies originating from first parties, we observe that developers fail to adhere to Google's policies concerning data transparency in almost half of the cases. As we can not assign (nor disprove) malicious intent, we argue that while developers need to be more vigilant when handling personal data, it is imperative that Google adopts an effective, automated validation mechanism for ensuring the completeness and accuracy of the data disclosed in Data Safety sections.

VPN experiments. Apps served through Google Play across countries contain different app versions based on the regulations and legislation of each country. In this experiment we downloaded and analyzed 100 apps from 10 different countries using a VPN service. The apps were downloaded from the respective Google Play store for each country. Table 3 lists the number of apps with discrepancies analyzed in each country. We found that the majority of apps contain discrepancies independent of the geolocation of the device. Furthermore, as apps follow specific policies based on each country's data protection laws (e.g., CCPA, PIPEDA, GDPR) and may have different regulations and enforcement concerning run-time consent, we analyzed the *Data Collected* and *Data Shared* discrepancies for the same set of 100 apps per country. Figure 6 shows the number of apps and the functions that resulted in *Data Collected* discrepancies, while Figure 7 shows the number of apps and the leaked data that resulted in *Data Shared* discrepancies. We observe that apps across all countries access functions that return device characteristics, as well as sensitive personal information such as the user's name, email, contacts and location. Moreover apps in Iceland tend to have more *Data Collected* discrepancies compared to other countries, while apps in Estonia have the least. Finally, we observe that India has the highest percentage of apps with *Data Shared* discrepancies.

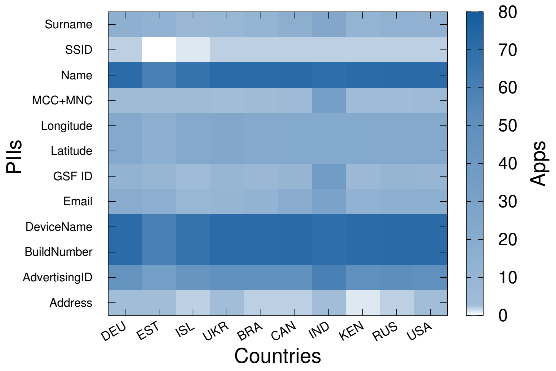

Figure 7: Data Shared discrepancies per country.

4.1 Case studies

Here we further detail notable examples of apps that do not accurately disclose their data practices in the Data Safety.

Popular Apps. Table 4 shows Data Safety discrepancies for popular apps that have more than 500 million downloads, across all three datasets. *Data Collected* denotes the number of Data Safety labels that were not disclosed, and the number of functions that resulted in the discrepancies. *Data Shared* denotes the number of undisclosed Data Safety labels and PIIs that have been sent over the network. We find discrepancies in popular apps from a wide range of categories, including social media (Linkedin, Instagram, Bigo, Lago, TikTok, X, Pinterest), messaging apps (Skype, Telegram, Snapchat), Microsoft Apps (Excel, Word, OneNote), mobile browsers (Chrome, Opera Mini) and games (Free Fire, PUBG MOBILE).

Lago and Bigo are the apps with the most *Data Collected* discrepancies that also leak PIIs without being disclosed in the *Data Shared* section. They access 9 and 13 functions respectively, that return device identifiers and characteristics (e.g.,

Table 4: Discrepancies in popular apps.

⇓ **DLs**	**Application**	***Data Collected*** (Labels - Methods)			***Data Shared*** (Labels - PIIs)		
		`Mar23`	`Jun23`	`Sep23`	`Mar23`	`Jun23`	`Sep23`
10B+	Chrome	0	0	0	1 - 1	0	2 - 6
1B+	Truecaller ID	1 - 1	0	0	0	0	0
1B+	X (Twitter)	0	0	0	0	0	1 - 2
1B+	Instagram	0	0	0	0	0	1 - 1
1B+	Subway Surfers	0	0	0	0	0	1 - 1
1B+	TikTok	0	0	0	1 - 2	1 - 2	1 - 2
1B+	Free Fire	0	0	0	1 - 2	1 - 2	1 - 2
1B+	Snapchat	0	0	0	1 - 2	1 - 1	1 - 3
1B+	Link to Windows	0	0	0	1 - 2	1 - 2	1 - 2
1B+	Dropbox	0	0	0	1 - 3	1 - 3	1 - 2
1B+	LinkedIn	0	0	0	1 - 3	1 - 3	2 - 6
1B+	Microsoft Excel	0	0	0	1 - 2	1 - 3	2 - 6
1B+	Skype	0	0	0	2 - 6	1 - 2	1 - 2
1B+	SHAREit	2 - 1	2 - 1	2 - 1	1 - 3	1 - 3	1 - 2
1B+	Microsoft Word	0	0	0	1 - 2	1 - 2	2 - 6
1B+	Telegram	1 - 2	2 - 3	2 - 3	0	1 - 2	0
500M+	Lago	5 - 10	5 - 10	4 - 9	1 - 3	1 - 3	1 - 3
500M+	Bigo Live	4 - 11	4 - 11	4 - 13	1 - 3	1 - 2	1 - 2
500M+	Pinterest	0	0	0	2 - 4	2 - 5	2 - 4
500M+	Microsoft OneNote	1 - 6	1 - 6	1 - 6	2 - 5	1 - 2	2 - 6
500M+	HP Print Serv.	1 - 1	1 - 1	1 - 1	0	0	0
500M+	Opera Mini	2 - 5	2 - 5	2 - 5	0	0	0
500M+	Hill Climb Racing	2 - 4	2 - 4	2 - 4	1 - 3	1 - 3	1 - 2
500M+	PUBG MOBILE	0	0	0	1 - 4	1 - 4	1 - 4
500M+	Uber	1 - 2	1 - 2	1 - 2	1 - 1	0	0

`getMacAddress`, `getDeviceId`, `getSubscriberId`) while also sharing data like the BuildNumber and DeviceName. Microsoft apps (e.g., Word, Excel) leak up to six PIIs including sensitive information such as the Email, Name and Surname. Opera Mini and SHAREit access the `getLastKnownLocation` method without declaring the appropriate Location Data Safety label. Uber fails to disclose the Files and Docs *Data Collected* label. Furthermore, we found that even though several apps do not have *Data Collected* discrepancies, they still share information (e.g., DeviceName, GSF ID, Email, Name and Surname) without disclosing it in the *Data Shared* section. For instance, Skype accesses sensitive information and discloses it in the *Data Collected* section, but fails to declare that the data is also sent over the network. The same applies for all apps that do not have *Data Collected* discrepancies but have *Data Shared* discrepancies. While certain apps may adhere to the relaxed Data Safety policies [11] concerning the requirements for declaring data in the Data Safety section, we strongly believe that such cases not only mislead users but also undermine the Data Safety section's purpose, which is to improve transparency and enable *informed* consent where users can easily understand how their data is being used.

Google Apps. During our March analysis, we identified three Google apps that had discrepancies in their Data Safety section. To further investigate how common this is across apps offered by Google, we manually analyzed all available Google apps [6] during July 2023. Out of 151 Google apps, 44 were not compatible with our device's API, while eight apps could not run on our devices. Out of the remaining 99 Google apps, we found that 47 of them had discrepancies in their *Data Collected* and *Data Shared* sections. Specifically, we found data belonging to 12 Data safety labels (Phone number, Contacts, Device or other IDs, Location, Approximate location, Precise location, Personal info, Files and docs, App info and performance, Other app performance data, Diagnostics, App activity) that were not declared. Such an example is `com.google.android.dialer`, which accesses the user's location using `requestLocationUpdates()` yet fails to declare the appropriate labels. Regarding undeclared data being shared, we found 24 apps sharing data that fall in the category of "Device or other IDs" and include the BuildNumber, Google Services Framework ID, DeviceName, MCC+MNC and AdvertisingID. Interestingly, some of these apps leak data even if the user rejects the run-time consent. Finally, we checked the origin of discrepancies and did not find any that originate from third-party functionality. The list of Google apps with discrepancies can be found here [7]. This further highlights the need for more stringent Data Safety disclosure policies being enforced, and the removal of potential loopholes that enable opaque data collection and sharing practices.

Independent Security Reviewed Apps. Developers can declare in the Data Safety section that their app has been independently validated against a global security standard (i.e., OWASP's MASVS). This is an optional review that is paid by developers and performed by third-party organizations. Google states that the independent security review may not necessarily verify the accuracy and completeness of the Data Safety declarations and that the developer is solely responsible for making complete and accurate declarations [8]. Nonetheless, we checked OWASP's MASVS and found that the MSTG-STORAGE-12 mobile security testing guide describes how to statically or dynamically identify privacy-related information that is being disclosed (or not) [60]. In our analysis we identified only a small number of apps that have gone through an independent security review and some of them still contain discrepancies in their Data Safety section (`Mar23`: 16/34, `Jun23`: 8/26, `Sep23`: 9/27). Interestingly, certain apps showed the independent security review in their Data Safety section in March but no longer listed it in June and/or September.

5 Discussion, Limitations and Future Work

In this section we further discuss our findings and how these issues could be mitigated, and also present the limitations of our dynamic analysis study, some of which are inherited by different components that our framework incorporates.

Mitigation. The Data Safety section has significant potential for enabling more transparent data practices, and Google plans to further incorporate Data Safety's information into certain run-time permission dialogues in their upcoming release of Android [4]. Our experimental findings revealed that 81% of the analyzed apps contain Data Safety discrepancies and many also violate policies governing run-time consent. Moreover, we believe that certain relaxed and vague policies can confuse users and developers alike, while allowing invasive entities to surreptitiously exfiltrate personal data without the user's knowledge or consent. Our study reveals the extent of Data Safety's misrepresentation of data collection and sharing practices, and highlights the need for reformations that will ensure that all data collected or shared is disclosed without exception. We consider the following directions crucial for improving data transparency in the mobile ecosystem:

Universal standard app disclosure. Android and iOS markets should adopt a universal data privacy disclosure system that accurately, completely, and clearly describes how apps and developers collect and share users' data.

Coherent and comprehensive data policies. All data collected and shared should be disclosed to enhance transparency and protect users from misleading information. Third-party libraries and developers must collaborate to ensure the effectiveness of these policies.

Validation mechanism. Official app markets must not rely solely on the developer's honesty, and must frequently review app's run-time behavior to ensure that it is consistent with the information disclosed in the Data Safety section through automated means. Reports should be publicly available and violations should be penalized by the market. We will open source our framework to facilitate such initiatives and additional research in this space.

Broader implications. The findings of our study are alarming, as they demonstrate to what extent Android applications are collecting and sharing user information without properly disclosing their practices. Surprisingly, this often occurs without the users' knowledge or consent, and in violation of relevant policies and regulations. Our research serves as a cautionary tale to users, aiming to raise awareness about the data Android applications collect and share, as well as the discrepancies that exist in apps' Data Safety disclosures. Overall, we envision that the increased awareness by end users and policy makers will lead to higher demand for transparency in the ecosystem, and hope that it will prompt additional regulations mandating more privacy-preserving practices. This includes Google refining their Data Safety policies, and investing in the deployment of appropriate compliance, review and validation mechanisms. Moreover, we anticipate additional pressure towards developers for properly following the Data Safety disclosure guidelines and accurately reporting the data collected and shared, in compliance with all relevant regulations and policies regarding data collection, disclosure and consent. Our study also sheds light on third-party libraries' questionable practices of incomplete data disclosure, thereby educating app developers about problematic third parties; more widespread awareness about regulatory violations from third-party libraries may facilitate initiatives for better ensuring compliance across the app ecosystem.

Permission mappings. Prior work (e.g., [14, 19, 21, 27] has invested significant effort in providing accurate mappings of permissions to functions. While we used state-of-the-art permission mappings as the starting point for creating the mappings used in our study (since mappings did not exist for version 12 of Android), and further verified them using the methodology employed in [77], it is possible that permission inaccuracies exist due to the limitations employed in their methodology for analyzing the Android framework.

Data Safety label mappings. Our mapping between Data Safety labels and API methods may not be complete, as no official documentation exists that maps the labels to function calls, and our labeling methodology may not be exhaustive. Specifically, we rely on the completeness of the Android Permission Mappings provided by state-of-the-art systems [27, 77]. For non-permission-protected calls to Data Safety labels, we manually identified and labeled such cases using the development guidelines, known PIIs, and by looking at the respective Android classes. Due to the vast amount of API methods that our framework monitors, we only assigned labels to functions that have been accessed during runtime analysis. Finally, our analysis does not include Data Safety labels that do not correspond to an API method, as we cannot monitor such cases (e.g., religious beliefs). Nonetheless, we believe that our mappings are useful to the Android community and are publicly available to facilitate additional research in this space. An alternative approach would be to use NLP techniques for processing API documentation and mapping them to potential labels. While this can be challenging as the documentation may not provide sufficient information, we consider this an interesting future direction.

Encrypted network traffic. We identified data being shared over the network by analyzing HTTPS traffic and using string matching. We searched for values of interest in plaintext format and common encodings. However, we cannot identify shared values that have been encrypted or heavily obfuscated. Differential analysis can be used for identifying leaks even in such cases, by observing deviations in the resulting network traffic [25]. We consider an exhaustive investigation using such techniques as part of our future work.

Consent dialogues. We identify and interact with in-app consent dialogues using a keyword-based approach. While our verification process highlighted the effectiveness of our approach, there may exist cases that cannot be handled by our system (e.g., the decline button being marked with an X).

Stacktraces, libs & obfuscation. Stacktrace-analysis is a common technique, which we use to differentiate Data Safety discrepancies between first and third-party functionality. In our analysis we compiled a comprehensive list of third-party library package names from prior work [20, 48, 68]. However, package names that are not included in our list will be classified as first parties. Furthermore, an obfuscated library package name will be assigned by default to the first party, which may lead to an over-reporting of discrepancies by the first party. While we cannot automatically classify obfuscated package names, such cases are rare [32].

Ethics & disclosure. Our experiments were conducted on our own devices using test accounts; we did not affect actual users or collect user data. Given the scale of our analysis, we shared our findings with Google as they already have mechanisms in place for auditing and notifying apps. We submitted a report to DDPRP, and they suggested submitting our report directly to the Android security team. Subsequently, we submitted a detailed report of our research, findings and video demonstrations to the security team; they informed us that they will conduct an investigation and share their findings with us. In summary, our work does not inflict any form of harm on users – instead, it has the opportunity to result in major benefits for users as our findings can incentivize and guide changes to the Data Safety section that will further enhance data transparency and user consent.

6 Related Work

The Data Safety section is a relatively recent Android mechanism that has not received sufficient scrutiny from the research community. To the best of our knowledge, this paper presents the first in-depth investigation of the Data Safety section and its use in the Android application ecosystem. We conduct a novel dynamic, large-scale, longitudinal study that uncovers privacy violations by correlating the run-time behavior of apps with the data declared in their Data Safety section. Here, we discuss prior work on the accuracy of privacy labels in the mobile ecosystem, the efficiency of in-app user consent dialogues and the liability for apps' privacy issues.

Data Safety section. In a recent report, Mozilla manually compared the privacy policies and terms-of-service of 40 popular Android apps (20 free, 20 paid) to their Data Safety section. Their comparison showed that the Data Safety section does not match the data practices outlined in the privacy policies and terms-of-service. While their focus is orthogonal to ours, their findings also highlight shortcomings of the Data Safety section. We provide a detailed comparison in Appendix A. In another study, Khandelwal et al. [40] explored how developers' practices evolve over time, using app metadata, by taking Data Safety section snapshots from 1.1M apps. They found that as of May 31, 2023, only 46.8% of the apps had privacy labels. Additionally, they found that developers have difficulties with the new Data Safety mechanism and some of the challenges they face include Google's guidelines being unclear, confusing policies due to frequent changes, and Data Safety form options being too complicated. In regards to the methodology employed, [40] uses static analysis to identify apps that include third-party libraries and, thus, deduces that many apps under-report their data sharing practices. While this approach reaches similar conclusions to ours, it suffers from the inherent limitations of static analysis (e.g., inability to handle encryption, dynamic code loading). They provide statistics based on app metadata and, as such, provide a coarse-grained analysis. On the contrary, our study employs dynamic analysis for detecting data collection and sharing practices on a per function-basis, by utilizing fine-grained and manually-curated mappings between Data Safety labels and function calls. We believe that these prior studies provide an important and complementary view of the Data Safety ecosystem using different methodologies, and contribute to the ongoing body of research pushing for better validation mechanisms being deployed by Google.

iOS privacy labels. Xiao et al. [79] proposed Lalaine, an automated system for iOS that combines dynamic and static analysis, natural language processing and network monitoring to identify inconsistencies between apps' run-time execution and their privacy labels. Their evaluation in 5,102 apps showed that the majority of apps neglect to disclose data, and the incomplete or incorrect guidelines from third-party SDKs may lead to non-compliant labels. While their work presents similarities to our work, significant differences exist between the two studies that stem from the differences between Android and iOS, and the differences between Apple's Privacy Labels and Google's Data Safety section. For instance, their system hooks iOS system APIs and matches the return values to network traffic to identify cases of data collection, since "data collection" refers to transmitting data off the user's device according to Apple [17]. This is in contrast to Google's Data Safety section that distinguishes between *Data Collected* and *Data Shared*. Another major difference between Apple's and Google's policies is that run-time consent does not interfere with Apple's Privacy Labels' disclosure requirements. In contrast, Google's Data Safety decision to incorporate run-time consent complicates the process of validating apps' data practices while also introducing additional obstacles to users being able to make informed decisions.

Koch et al. [43] explored how iOS privacy labels are used and whether developers adhere to the declared labels. They monitored network traffic for 1,687 apps without performing any app interaction and found that several apps contact known trackers, transmit data and violate their labels. They concluded that simply declaring privacy labels is not sufficient for data transparency. ATLAS [38] uses an ensemble-based classifier for predicting privacy labels from the privacy policies. Their study suggests that 88.0% of apps had at least one discrepancy between the privacy policy and labels. Li et al. [50] studied how quickly app developers create and update privacy labels and found that 51.6% of apps do not have a label. In [49] the authors examined the usability and understandability of Apple's privacy nutrition labels and identified confusion amongst developers regarding Apple's documentation. Rodriguez et al. [67] compared Android and iOS privacy labels for 822 apps that exist in both operating systems. They found inconsistencies between privacy labels disclosed across platforms with only 3.2% of the apps being consistent.

Consent violations. A plethora of prior work [22, 44, 53, 58, 62, 71, 80]) has explored run-time consent violation in mobile apps. In [57] the authors showed that apps deceive users into accepting all data sharing and transmit data even when users have opted out, therefore violating GDPR's consent requirements. Reyes et al. [66] found that the majority of the apps and the embedded third-party SDKs contain potential COPPA violations. POLICHECK [15] identifies policy inconsistencies in mobile apps and identifies the entity (first- or third-party) that receives the privacy sensitive data. Koch et al. [42] analyzed privacy consent dialogues in both Android and iOS, and found that apps transmit traffic before the consent and even after rejecting the consent. Additionally, they found that only a small percentage of apps give the user some form of a choice when they are presented with a consent dialog. MOWCHECKER [33] statically identifies data violations from third-party libraries when user's run-time choice is to opt-out from data collection. Subsequently, their study urges that third-party libraries and mobile apps need to collaborate for respecting the users' withdrawal choices.

Privacy responsibilities. While several studies [28, 29, 32, 35, 63, 73] have investigated the excessive permission usage and leakage of private information from third-party libraries, who is responsible for privacy issues has been debated. A recent study [41] showed that libraries are responsible for several privacy issues including ad fraud. On the contrary, Tahaei et al. [75] found that the information presented by ad networks to developers complies with legal regulations and they are responsible for handling privacy regulations.

7 Conclusion

Google's new Data Safety process is an important step towards increased platform support for data transparency that will allow users to easily understand how apps process their data. Unfortunately, in its current implementation, bad practices and relaxed policies can confuse both end-users and developers, while also creating opportunity for abuse and the surreptitious exfiltration of data. In this paper we developed a system that automatically identifies discrepancies between apps' run-time behavior and what data developers disclose in Data Safety. Our system analyzes and identifies discrepancies in applications based on different run-time consent scenarios. Our subsequent in-depth analysis spanning across an entire year demonstrated severe inaccuracies in apps' Data Safety sections. To make matters worse, we uncovered that applications not only violate Google's Data Safety policies, but also violate Google's policy governing consent requirements leading to several alarming findings. Consequently we proposed a set of guidelines that should be adopted towards achieving better data transparency. We hope that our study will facilitate additional research pushing for better validation mechanisms that protect users from misleading information.

Acknowledgements

We thank the reviewers and shepherd for their helpful feedback. This project was supported by Horizon 2020 (GA No 101092950, 101093051, 101021659) and the National Science Foundation (CNS-2211574, CNS-2143363). The views in this paper are only those of the authors and may not reflect those of the funding bodies, the US Government or the NSF.

References

[1] Play Console Help. https://support.google.com/googleplay/android-developer#topic=3450769.

[2] Xposed framework. https://repo.xposed.info.

[3] Raccoon - APK downloader, 2018. http://www.onyxbits.de/raccoon.

[4] Data safety information is more visible. https://tinyurl.com/y2wex7c3, 2023.

[5] Device identifiers, 2023. https://source.android.com/docs/core/connect/device-identifiers.

[6] Google Android Play Store Apps, 2023. https://github.com/petarov/google-android-app-ids.

[7] Google apps with discrepancies, 2023. https://pastebin.com/vcqn50ii.

[8] Independent security review. https://tinyurl.com/26p25s9x, 2023.

[9] Number of available applications in the google play store from December 2009 to June 2023, 2023. https://tinyurl.com/bdzbp32r.

[10] Objection - Runtime Mobile Exploration, 2023. https://github.com/sensepost/objection.

[11] Provide information for Google Play's data safety section - play console help, 2023. http://tinyurl.com/yckxutbk.

[12] See No Evil: Loopholes in Google's Data Safety Labels Keep Companies in the Clear and Consumers in the Dark, 2023. https://tinyurl.com/2hhmcfbe.

[13] Telephonymanager: getimei(), 2023. https://tinyurl.com/mr3tcuw6.

[14] Yousra Aafer, Guanhong Tao, Jianjun Huang, Xiangyu Zhang, and Ninghui Li. Precise Android API protection mapping derivation and reasoning. In *CCS '18*.

[15] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. Actions speak louder than words: Entity-sensitive privacy policy and data flow analysis with policheck. In *USENIX Security '20*.

[16] AndroidDevelopers. Android Developers announcing reinstating app permission in Google Play. http://tinyurl.com/426vewvh, 2022.

[17] Apple. App privacy details on the App Store. https://tinyurl.com/3rtzm6ex, 2023.

[18] arsTECHNICA. After public outcry, Google will reinstate Play Store app permissions list. http://tinyurl.com/2h3pndks, 2022.

[19] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: Analyzing the Android permission specification. In *CCS '12*.

[20] Michael Backes, Sven Bugiel, and Erik Derr. Reliable third-party library detection in Android and its security applications. In *CCS '16*.

[21] Michael Backes, Sven Bugiel, Erik Derr, Patrick McDaniel, Damien Octeau, and Sebastian Weisgerber. On demystifying the Android application framework:Re-Visiting Android permission specification analysis. In *USENIX Security '16*.

[22] Duc Bui, Yuan Yao, Kang G. Shin, Jong-Min Choi, and Junbum Shin. Consistency analysis of data-usage purposes in mobile apps. In *CCS '21*.

[23] Quan Chen, Panagiotis Ilia, Michalis Polychronakis, and Alexandros Kapravelos. Cookie swap party: Abusing first-party cookies for web tracking. In *WWW '21*.

[24] Saksham Chitkara, Nishad Gothoskar, Suhas Harish, Jason I. Hong, and Yuvraj Agarwal. Does this app really need my location?: Context-aware privacy management for smartphones. *IMWUT '17*.

[25] Andrea Continella, Yanick Fratantonio, Martina Lindorfer, Alessandro Puccetti, Ali Zand, Christopher Kruegel, and Giovanni Vigna. Obfuscation-resilient privacy leak detection for mobile apps through differential analysis. In *NDSS '17*.

[26] Cortesi, Aldo and Hils, Mayimilian and Kriechbaumer, Thomas. mitmproxy. https://mitmproxy.org.

[27] Abd Elhamed M. Dawoud and Sven Bugiel. Bringing balance to the force: Dynamic analysis of the Android application framework. In *NDSS '21*.

[28] Soteris Demetriou, Whitney Merrill, Wei Yang, Aston Zhang, and Carl A Gunter. Free for all! assessing user data exposure to advertising libraries on android. In *NDSS '16*.

[29] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on Android. In *CCS '17*.

[30] Android Developers. Review how your app collects and shares user data. https://tinyurl.com/482yeb6b.

[31] Michalis Diamantaris, Serafeim Moustakas, Lichao Sun, Sotiris Ioannidis, and Jason Polakis. This sneaky piggy went to the Android ad market: Misusing mobile sensors for stealthy data exfiltration. In *CCS '21*.

[32] Michalis Diamantaris, Elias P. Papadopoulos, Evangelos P. Markatos, Sotiris Ioannidis, and Jason Polakis. Reaper: Real-time App Analysis for Augmenting the Android Permission System. In *CODASPY '19*.

[33] Xiaolin Du, Zhemin Yang, Jiapeng Lin, Yinzhi Cao, and Min Yang. Withdrawing is believing? detecting inconsistencies between withdrawal choices and third-party data collections in mobile apps. In *SP '24*.

[34] Nicole Eling, Siegfried Rasthofer, Max Kolhagen, Eric Bodden, and Peter Buxmann. Investigating users' reaction to fine-grained data requests: A market experiment. In *HICSS '16*.

[35] Michael C Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *WISEC '12*.

[36] Google Play Help. Understand app privacy & security practices with Google Play's Data safety section. https://tinyurl.com/3v9728f5, 2022.

[37] Play Console Help. Google Play's User Data policy - Prominent Disclosure & Consent Requirement. https://tinyurl.com/yfcn4pr3, 2022.

[38] Akshath Jain, David Rodriguez, Jose M. del Alamo, and Norman Sadeh. Atlas: Automatically detecting discrepancies between privacy policies and privacy labels. In *EuroS&PW '23*.

[39] Patrick Gage Kelley, Sunny Consolvo, Lorrie Faith Cranor, Jaeyeon Jung, Norman Sadeh, and David Wetherall. A conundrum of permissions: Installing applications on an Android smartphone. In *USEC '12*.

[40] Rishabh Khandelwal, Asmit Nayak, Paul Chung, and Kassem Fawaz. Unpacking privacy labels: A measurement and developer perspective on Google's Data Safety section. In *USENIX Security '24*.

[41] Joongyum Kim, Jung-hwan Park, and Sooel Son. The abuser inside apps: Finding the culprit committing mobile ad fraud. In *NDSS '21*.

[42] Simon Koch, Benjamin Altpeter, and Martin Johns. The ok is not enough: A large scale study of consent dialogs in smartphone applications. In *USENIX Security '23*.

[43] Simon Koch, Malte Wessels, Benjamin Altpeter, Madita Olvermann, and Martin Johns. Keeping privacy labels honest. In *PoPETS '22*.

[44] Konrad Kollnig, Pierre Dewitte, Max Van Kleek, Ge Wang, Daniel Omeiza, Helena Webb, and Nigel Shadbolt. A fait accompli? an empirical study into the absence of consent to third-party tracking in Android apps. In *SOUPS '21*.

[45] Renuka Kumar, Apurva Virkud, Ram Sundara Raman, Atul Prakash, and Roya Ensafi. A large-scale investigation into geodifferences in mobile apps. In *USENIX Security '22*.

[46] Ilias Leontiadis, Christos Efstratiou, Marco Picone, and Cecilia Mascolo. Don't kill my ads!: Balancing privacy in an ad-supported mobile application market. In *HotMobile '12*.

[47] Christophe Leung, Jingjing Ren, David Choffnes, and Christo Wilson. Should you use the app for that? comparing the privacy implications of app-and web-based online services. In *IMC '16*.

[48] Li Li, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. An investigation into the use of common libraries in Android apps. In *SANER '16*.

[49] Tianshi Li, Kayla Reiman, Yuvraj Agarwal, Lorrie Faith Cranor, and Jason I Hong. Understanding challenges for developers to create accurate privacy nutrition labels. In *CHI '22'*.

[50] Yucheng Li, Deyuan Chen, Tianshi Li, Yuvraj Agarwal, Lorrie Faith Cranor, and Jason I Hong. Understanding ios privacy nutrition labels: An exploratory large-scale analysis of app store data. In *CHI EA '22*.

[51] LSPosed. LSPosed Framework. `https://github.com/LSPosed/LSPosed`, 2022.

[52] Dominique Machuletz and Rainer Böhme. Multiple purposes, multiple problems: A user study of consent dialogs after GDPR. *Proceedings on Privacy Enhancing Technologies*, 2020, apr 2020.

[53] Célestin Matte, Nataliia Bielova, and Cristiana Santos. Do cookie banners respect my choice?: Measuring legal compliance of banners from iab europe's transparency and consent framework. In *SP '20*.

[54] McAfee. Customer URL Ticketing System, Check Single URL. `https://sitelookup.mcafee.com/`, 2023.

[55] Abraham H. Mhaidli, Yixin Zou, and Florian Schaub. We can't Live Without Them! App developers' adoption of ad networks and their considerations of consumer risks. In *SOUPS '19*.

[56] Le Nguyen, Yuan Tian, Sungho Cho, Wookjong Kwak, Sanjay Parab, Yuseung Kim, Patrick Tague, and Joy Zhang. Unlocin: Unauthorized location inference on smartphones without being caught. In *PRISMS '13*.

[57] Trung Nguyen, Michael Backes, and Ben Stock. Freely given consent? Studying consent notice of third-party tracking and its violations of GDPR in Android apps. In *CCS '22*.

[58] Trung Tin Nguyen, Michael Backes, Ninja Marnau, and Ben Stock. Share first, ask later (or never?) Studying violations of GDPR's Explicit Consent in Android Apps. In *USENIX Security '21*.

[59] Lukasz Olejnik, Steven Englehardt, and Arvind Narayanan. Battery status not included: Assessing privacy in web standards. In *IWPE '17*.

[60] OWASP. Mobile App User Privacy Protection. `https://tinyurl.com/hw8wrbnu`, 2023.

[61] OWASP. Privacy Nutrition Labels - Transparency is the best policy. `https://www.apple.com/privacy/labels/`, 2023.

[62] Federica Paci, Jacopo Pizzoli, and Nicola Zannone. A comprehensive study on third-party user tracking in mobile applications. In *ARES '23*, 2023.

[63] Elias P Papadopoulos, Michalis Diamantaris, Panagiotis Papadopoulos, Thanasis Petsas, Sotiris Ioannidis, and Evangelos P Markatos. The long-standing privacy debate: Mobile websites vs mobile apps. In *WWW '17*.

[64] Andrea Possemato and Yanick Fratantonio. Towards HTTPS everywhere on Android: We are not there yet. In *USENIX Security '20*.

[65] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the Android permissions system. In *USENIX Security '19*.

[66] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, Serge Egelman, et al. "Won't somebody think of the children?" Examining COPPA compliance at scale. In *PoPETS '18*.

[67] David Rodriguez, Akshath Jain, Jose M del Alamo, and Norman Sadeh. Comparing Privacy Label Disclosures of Apps Published in Both the App Store and Google Play Stores. In *EuroS&PW '23*.

[68] Jordan Samhi, Marco Alecci, Tegawendé F Bissyandé, and Jacques Klein. A dataset of android libraries. *arXiv preprint arXiv:2307.12609*, 2023.

[69] Jaebaek Seo, Daehyeok Kim, Donghyun Cho, Insik Shin, and Taesoo Kim. FLEXDROID: enforcing in-app privilege separation in android. In *NDSS '16*.

[70] Bingyu Shen, Lili Wei, Chengcheng Xiang, Yudong Wu, Mingyao Shen, Yuanyuan Zhou, and Xinxin Jin. Can systems explain permissions better? understanding users' misperceptions under smartphone runtime permission model. In *USENIX Security '21*.

[71] Rocky Slavin, Xiaoyin Wang, Mitra Bokaei Hosseini, James Hester, Ram Krishnan, Jaspreet Bhatia, Travis D Breaux, and Jianwei Niu. Toward a framework for detecting privacy policy violations in Android application code. In *ICSE '16*.

[72] Than Htut Soe, Cristiana Teixeira Santos, and Marija Slavkovik. Automated detection of dark patterns in cookie banners: how to do it poorly and why it is hard to do it any other way. *arXiv preprint arXiv:2204.11836*.

[73] Sooel Son, Daehyeok Kim, and Vitaly Shmatikov. What mobile ads know about mobile users. In *NDSS '16*.

[74] Mengtao Sun and Gang Tan. Nativeguard: Protecting Android Applications from Third-party Native Libraries. In *WiSec '14*.

[75] Mohammad Tahaei and Kami Vaniea. "Developers Are Responsible": What Ad Networks Tell Developers About Privacy. In *CHI EA '21*.

[76] Fabo Wang, Yuqing Zhang, Kai Wang, Peng Liu, and Wenjie Wang. Stay in your cage! A sound sandbox for third-party libraries on Android. In *ESORICS '16*.

[77] Sinan Wang, Yibo Wang, Xian Zhan, Ying Wang, Yepang Liu, Xiapu Luo, and Shing-Chi Cheung. Aper: evolution-aware runtime permission misuse detection for Android apps. In *ICSE '22*.

[78] John Wu. A Magic Mask to alter System Systemless-ly. `https://github.com/topjohnwu/Magisk`.

[79] Yue Xiao, Zhengyi Li, Yue Qin, Xiaolong Bai, Jiale Guan, Xiaojing Liao, and Luyi Xing. Lalaine: Measuring and characterizing non-compliance of apple privacy labels. In *USENIX Security '23*.

[80] Sebastian Zimmeck, Peter Story, Daniel Smullen, Abhilasha Ravichander, Ziqi Wang, Joel R Reidenberg, N Cameron Russell, and Norman Sadeh. Maps: Scaling privacy compliance analysis to a million apps. *PoPETS '19*.

A Appendix

Additional Technical Details. Table 5 details the APIs that return Personally Identifiable Information, which are monitored by our framework. Functions may be permission-protected or not depending on the Android version.

Table 5: APIs returning Personally Identifiable Information.

Class	**Function**
AdvertisingIdClient	getAdvertisingIdInfo
TelephonyManager	getNetworkCountryIso, getNetworkOperator, getNetworkOperatorName
ContentProvider	query
SubscriptionInfo	getCardId, getCarrierId, getCarrierName, getCountryIso, getMcc getDisplayName, getMccString, getMnc, getMncString, getSimOperator
BatteryManager	getIntProperty, getLongProperty

Breakdown of discrepancies. Table 6 shows the discrepancies between apps' run-time behavior and Data Safety labels for every run-time scenario across all datasets, and provides aggregated results for the functions that lead to the discrepancies along with the respective Android permissions.

Comparison to Mozilla's findings. While we cannot directly compare our results to Mozilla's study [12], due to differences in the methodology and the time of their analysis (i.e., we do not have the exact apk version they analyzed or the date of the Data Safety section), we identified their 20 free apps and included them in our analysis. As our main analysis is based on March 2023 and their report was published in February 2023, we found that for three apps we have not found any discrepancies and Mozilla graded these apps as "OK". Furthermore, UC Browser has the most discrepancies and Mozilla could not grade it, due to the app not filling out the form. We found that during March's analysis UC Browser did not declare anything in the Data Safety section. During June's analysis, the app disclosed the labels Device or other IDs and App info and performance in the Data Collected section. We found that the app collects sensitive information from 14 functions and fails to declare the labels Location, Precise location and Phone number in the Data Collected. Moreover, we found that in June's analysis the app does not disclose Data Shared info, but leaks the BuildNumber and the DeviceName that require the label Device or other IDs. We further analyzed the app and found that apart from sharing the BuildNumber and the DeviceName it also shares (without disclosing) the GSF ID and the MCC+MNC. Additionally, for 10 apps that Mozilla graded as "Poor" or "Needs Improvement", in our analysis a month later we did not find any discrepancies. Even though we did not analyze the privacy policies and the terms of use of these apps (as it falls outside the scope of our work), we believe that both studies provide an important and complementary view of the complicated and contentious design of Google's Data Safety mechanism.

Table 6: Breakdown of discrepancies between run-time behavior and the Data Safety section for different run-time scenarios. *API Methods* show the number of methods that yielded the discrepancies. *Apps* show the number of apps with discrepancies.

Scenario	Data Safety Label	API Methods			Permissions	Apps		
		Mar23	Jun23	Sep23		Mar23	Jun23	Sep23
No run-time consent	Location	11	11	10	ACCESS_(COARSE\FINE)_LOCATION, READ_PHONE_STATE	130	121	107
	Approximate location	6	6	5	ACCESS_(COARSE\FINE)_LOCATION, READ_PHONE_STATE	145	133	120
	Precise location	9	9	9	ACCESS_(COARSE\FINE)_LOCATION	252	252	237
	Personal info	13	14	13	READ_(PHONE\PRIVILEGED_PHONE)_STATE, READ_SMS, READ_PHONE_NUMBERS, GET_ACCOUNTS	1,319	1,323	1,235
	Phone number	1	1	1	READ_PHONE_NUMBERS, READ_PHONE_STATE, READ_SMS	10	11	9
	Photos or videos	1	1	1	READ_EXTERNAL_STORAGE	2	3	2
	Photos	1	1	1	READ_EXTERNAL_STORAGE	2	3	2
	Audio files	1	1	1	-	4	5	3
	Files and docs	3	3	3	READ_EXTERNAL_STORAGE	9	10	9
	Music files	1	1	1	-	4	5	3
	Contacts	1	1	1	READ_PHONE_STATE	1	1	1
	App activity	2	2	2	-	793	802	758
	Other app perf/nce data	3	3	3	-	781	804	746
	Device or other IDs	24	25	24	BLUETOOTH_CONNECT, READ_(PHONE\PRIVILEGED_PHONE)_STATE, LOCAL_MAC_ADDRESS, ACCESS_FINE_LOCATION	771	792	718
	Diagnostics	7	7	8	WATCH_APPOPS	963	893	913
	App info and perf/nce	12	12	13	WATCH_APPOPS, PACKAGE_USAGE_STATS	1,136	1,155	1,087
Accept	Location	8	9	6	ACCESS_(COARSE\FINE)_LOCATION, (READ\MODIFY)_PHONE_STATE	46	49	33
	Approximate location	4	5	3	ACCESS_(COARSE\FINE)_LOCATION, READ_PHONE_STATE	54	58	41
	Precise location	8	8	8	ACCESS_(COARSE\FINE)_LOCATION, MODIFY_PHONE_STATE	116	117	84
	Personal info	10	10	9	READ_PHONE_NUMBERS, READ_PHONE_STATE, GET_ACCOUNTS, READ_SMS	448	428	369
	Phone number	3	3	1	READ_PHONE_NUMBERS, READ_PHONE_STATE, READ_SMS	6	6	3
	Photos or videos	1	1	1	READ_EXTERNAL_STORAGE	4	3	1
	Photos	1	1	1	READ_EXTERNAL_STORAGE	3	2	1
	Files and docs	3	3	3	READ_EXTERNAL_STORAGE	6	6	4
	App activity	2	2	2	-	256	240	185
	App info and perf/nce	10	10	9	WATCH_APPOPS, PACKAGE_USAGE_STATS	275	254	211
	Other app perf/nce data	3	3	3	-	272	262	237
	Device or other IDs	18	17	17	BLUETOOTH_CONNECT, READ_(PHONE\PRIVILEGED_PHONE)_STATE, LOCAL_MAC_ADDRESS, ACCESS_FINE_LOCATION	228	200	155
	Diagnostics	4	4	3	WATCH_APPOPS	280	256	202
Reject	Location	3	1	1	ACCESS_(COARSE\FINE)_LOCATION	5	4	2
	Approximate location	3	1	2	ACCESS_(COARSE\FINE)_LOCATION	5	5	4
	Precise location	4	3	3	ACCESS_(COARSE\FINE)_LOCATION	12	13	13
	Personal info	4	4	4	READ_PHONE_STATE, GET_ACCOUNTS	49	39	37
	Files and docs	2	2	0	-	1	1	0
	App activity	2	2	2	-	19	21	17
	App info and perf/nce	7	8	8	PACKAGE_USAGE_STATS	28	23	22
	Other app perf/nce data	3	3	3	-	29	26	23
	Device or other IDs	4	4	3	READ_PHONE_STATE, BLUETOOTH_CONNECT, LOCAL_MAC_ADDRESS	28	23	15
	Diagnostics	3	2	3	-	28	22	25
Reject not found (No action)	Location	5	5	4	ACCESS_(COARSE\FINE)_LOCATION	29	29	20
	Approximate location	3	3	3	ACCESS_(COARSE\FINE)_LOCATION	33	33	22
	Precise location	6	6	4	ACCESS_(COARSE\FINE)_LOCATION	69	66	45
	Personal info	8	8	7	READ_PHONE_STATE, GET_ACCOUNTS	306	306	262
	Phone number	3	3	0	READ_PHONE_NUMBERS, READ_PHONE_STATE, READ_SMS	2	2	0
	Photos or videos	1	1	1	READ_EXTERNAL_STORAGE	2	2	1
	Photos	1	1	1	READ_EXTERNAL_STORAGE	2	2	1
	Files and docs	3	1	3	READ_EXTERNAL_STORAGE	3	2	2
	App activity	2	2	2	-	182	164	130
	App info and perf/nce	9	9	9	PACKAGE_USAGE_STATS	215	199	158
	Other app perf/nce data	3	3	3	-	161	162	139
	Device or other IDs	17	16	15	BLUETOOTH_CONNECT, READ_(PHONE\PRIVILEGED_PHONE)_STATE, LOCAL_MAC_ADDRESS, ACCESS_FINE_LOCATION	143	128	102
	Diagnostics	4	4	3	WATCH_APPOPS	183	169	125
Reject & app exited (No action)	Location	3	3	3	ACCESS_(COARSE\FINE)_LOCATION	6	9	8
	Approximate location	3	3	2	ACCESS_(COARSE\FINE)_LOCATION	9	13	12
	Precise location	4	4	5	ACCESS_(COARSE\FINE)_LOCATION	15	20	18
	Personal info	5	4	2	READ_PHONE_STATE, GET_ACCOUNTS	25	26	27
	Files and docs	0	2	0	-	0	1	0
	App activity	2	2	2	-	22	21	18
	App info and perf/nce	9	9	9	PACKAGE_USAGE_STATS	33	32	31
	Device or other IDs	10	9	7	READ_(PHONE\PRIVILEGED_PHONE)_STATE, BLUETOOTH_CONNECT, LOCAL_MAC_ADDRESS	27	20	15
	Other app perf/nce data	3	3	3	-	14	12	20
	Diagnostics	3	3	3	-	25	29	30