Assessing Privacy Disclosure Compliance of Android Third-Party SDKs

Mark Huasong Meng, Chuan Yan, Qing Zhang, Zeyu Wang, Kailong Wang,
Sin G. Teo, Guangdong Bai, and Jin Song Dong

***Abstract*—Third-party Software Development Kits (SDKs) are widely adopted in Android app development, to accelerate development pipelines and enhance app functionality effortlessly. However, this convenience raises substantial concerns about unauthorized access to users' privacy-sensitive information, which could be further abused for illegitimate purposes like user tracking or monetization. Our study offers a targeted analysis of user privacy protection among Android third-party SDKs, filling a critical gap in the Android software supply chain. It focuses on two aspects of their privacy practices, including *data exfiltration* and *behavior-policy compliance* (or *privacy disclosure compliance*), utilizing taint analysis and large language models. It covers 158 widely used SDKs collected from two major distribution channels: the official platform and a leading third-party alternative. From these SDKs, we identified 338 instances of personal data exfiltration. On privacy disclosure compliance, our study reveals that more than 30% of the examined SDKs fail to provide a privacy policy to disclose their data handling practices. Among those that provide privacy policies, 37% of them over-collect user data, and 88% falsely claim access to sensitive data. We revisit the latest versions of the SDKs after 12 months. Our analysis demonstrates a persistent lack of improvement in these concerning trends. Based on our findings, we propose three actionable recommendations to mitigate the privacy leakage risks and enhance privacy protection for Android users. Our research not only serves as an urgent call for industry attention but also provides crucial insights for future regulatory interventions.**

***Index Terms*—mobile, privacy assessment, taint analysis.**

I. Introduction

The advent and ubiquity of third-party Software Development Kits (SDKs) in Android app development herald a profound shift in the mobile software development paradigm, transforming efficiency and accelerating innovation. These SDKs provide pre-built reusable code that offers a broad spectrum of functionalities, from fundamental features like user authentication [1] and data encryption [2] to specialized components like analytics [3], advertisement delivery [4], and user interface enhancements [5]. The open-source nature and global reach of the Android ecosystem further fuel the proliferation of these SDKs.

Integrating an SDK into an app is often as simple as declaring dependencies in the build configuration file, thus drastically reducing development time and allowing developers to focus on app-specific features. Nowadays, the majority of Android apps incorporate multiple third-party SDKs, with an average app integrating approximately 8.6 SDKs [6], [7]. However, this widespread dependency also introduces significant security and privacy concerns. The complexity of the software supply chain and the opaque data handling practices of some SDKs can lead to privacy risks and security vulnerabilities, either due to malicious intent or unintentional oversights. These can be exploited by malevolent actors or lead to inadvertent privacy violations, as evidenced by recent high-profile incidents related to third-party SDKs [8], [9].

The privacy issues caused by SDKs could be further magnified in the pre-installed apps on Android devices. These apps are bundled by device manufacturers or network carriers and, therefore, often have an extremely large user base. They come with high privileges and are typically non-removable, such that an SDK embedded by them can stealthily perform higher-privileged operations that normal apps are unable to. An illustrative example is an app named "*Mobile Services Manager*" pre-installed on certain Android devices, which was found to download apps without user consent due to a malicious third-party SDK embedded [10], [11].

Assessing the privacy practices of SDKs involves two key aspects, i.e., *data exfiltration* and *behavior-policy compliance* (or *privacy disclosure compliance*). Data exfiltration concerns the extraction of sensitive data to publicly accessible places, such as the Internet, publicly accessible directories in file systems, and system settings on the device. Although third-party SDK code can often be obtained through reverse engineering, SDKs frequently lack sufficient public documentation, and their behaviors are shaped by highly product- and domain-specific mechanisms that are difficult to faithfully reproduce without essential context. This operational "black-box" nature hinders meaningful assessments, leaving both end users and app developers in the dark and elevating risks. Dishonest or malicious SDKs may disguise themselves as their embedding apps to extensively collect privacy-sensitive data, while the end-users are unable to distinguish which party requests their personal data. Incidents like the X-Mode controversy [12], where user locations were tracked by over 100 apps through embedded SDKs, underscore this point and serve as a stark reminder of the challenges associated with preserving users' privacy in third-party SDKs.

On the regulatory front, data protection regulations have been put in place globally, such as the General Data Protection Regulation (GDPR) [13] and the California Consumer Privacy Act (CCPA) [14], which impose stringent requirements on

M. H. Meng is with University College Dublin, Ireland. C. Yan is with the University of Queensland, Australia. G. Bai is with the City University of Hong Kong, Hong Kong S.A.R. Q. Zhang and Z. Wang are with ByteDance Group, China, K. Wang is with Huazhong University of Science and Technology, China, S. G. Teo is with Agency for Science, Technology and Research, Singapore, and J. S. Dong is with National University of Singapore, Singapore.

0000–0000/00$00.00 © 2021 IEEE

data processors. In response to this, software developers are required to provide a privacy policy to explicitly disclose what personal information is collected and how this data is handled by their software. Dissecting whether the data handling behaviors of SDKs comply with the claims in their privacy policies, which we define as privacy disclosure compliance, becomes an urgent task.

Our Work. In this work, we conduct a large-scale study on the privacy practices of existing third-party SDKs. Our study covers 158 prevalently installed SDKs from two mainstream release platforms, including 139 SDKs from Google Play SDK Index [15] and 19 from CAICT SDK Index [16]. The former is the official source of Android SDKs as a part of Google Play infrastructure, which lists the most widely used commercial SDKs in the Android app ecosystem [17]. The latter describes itself as the largest release platform of mobile SDKs in China, serving as a complementary source of SDKs due to the absence of Google's service in the Chinese market. Based on the statistical data provided by the two platforms, over 100 collected SDKs have been embedded in at least 100 apps with over 10 million installations, demonstrating the broad impact of this study.

Our study comprises two main strategies, i.e., conducting a data exfiltration assessment through static program analysis, and assessing privacy disclosure compliance through interpreting privacy policies. Specifically, we conduct a static taint analysis for tracking the flow of privacy-sensitive data within the SDK. We explore the traces that read and share privacy-sensitive data in public spaces, and accordingly investigate whether these traces can lead to potential privacy-sensitive data exfiltration. For the privacy disclosure compliance assessment, we employ a lightweight yet effective transformer-based language model to analyze the SDKs' privacy policies and thereby identify what data is requested, cross-check the data claimed in privacy policies with the data access behaviors caught during the taint analysis, and determine if an excessive collection or over-claiming issue exists.

Our study reveals critical privacy shortcomings in Android third-party SDKs. Astonishingly, 346 traces are found to read personal data and share it in publicly accessible places. Among these traces, 338 are confirmed to be subject to data leakage. Only 109 out of the 158 examined SDKs provide privacy policies, among which approximately 37% over-collect private data and over 88% falsely claim their data collection scopes, seriously violating global data protection norms. Our re-inspection after 12 months shows no meaningful improvement in these practices over time. We also uncover a novel system settings injection issue: some malicious SDKs exploit pre-installed apps to share privacy-sensitive data into Android OS's public system settings, effectively creating a unique user-unresettable identifier (UUI) accessible to any app. This unprecedented finding underscores the need for improvement in SDK development practices, permission management, and privacy violation detection. To this end, we further propose three key strategic recommendations for mitigation.

Contributions. Our work makes the following key contributions:

- **Large-Scale Privacy Disclosure Compliance Assessment of Android Third-Party SDKs.** To the best of our knowledge, we are the first to conduct a systematic, large-scale assessment of the privacy disclosure compliance of Android third-party SDKs. Our proposed pipeline draws on different sources to construct a comprehensive scope of privacy-related data, and leverages a state-of-the-art static taint analysis engine and a lightweight transformer-based NLI model for privacy policy inference, making scalable and reproducible privacy disclosure compliance assessment feasible.
- **Demystifying the Privacy Disclosure Compliance Landscape at the SDK Level.** Our assessment shows that 34 out of 158 examined SDKs (21.5%) exhibit potential privacy leakage risks. Our further investigation of privacy policies finds that 36.7% of the examined SDKs excessively collect users' privacy-related data in their implementation, and as many as 88.1% over-claim their privacy access. Through our re-inspection, we depict the trends of SDKs' practices in collecting and sharing personal data, and underscore the urgency of tackling privacy risks in the Android ecosystem at a fine-grained SDK level.
- **Discovery of Privacy Risks caused by Pre-installed Privileges.** We find a system settings injection issue in the pre-installed apps, in which a malicious SDK can excessively read personal data and share it in the system settings. This enables an arbitrary pre-installed app with the malicious SDK to create uniquely unresettable identifiers and share them, violating the relevant regulations about privacy protection. It is the first time such issues have been revealed and systematically investigated.

II. Background and Problem Definitions

A. GDPR and Privacy Disclosure Compliance

The General Data Protection Regulation (GDPR) [13], a landmark legislation enacted by the European Union (EU) in 2018, has profoundly transformed the landscape of data privacy laws within the region. Under GDPR, entities processing personal data, defined broadly as any data enabling the direct or indirect identification of individuals, are subject to stringent regulations that underscore the importance of "*lawfulness, fairness, and transparency*". Furthermore, GDPR provides explicit requirements for obtaining valid consent, necessitating that it is freely given, specific, informed, and unambiguous.

Privacy disclosure compliance, in the context of GDPR, necessitates a thorough examination of privacy policies to ensure adherence to the GDPR's comprehensive principles and stipulations. It entails a meticulous inspection of data processing activities, thereby ensuring data is acquired and utilized only for explicit, legitimate purposes. Furthermore, the principle of "*data minimization*" obliges organizations to limit their data processing activities to the minimum necessary, thereby reinforcing the importance of data accuracy and relevance. This process also extends to an examination of storage practices, necessitating the validation that data is retained only for as long as necessary and is safeguarded against unlawful processing, inadvertent loss, or damage. Therefore,

privacy disclosure compliance is essential for both enforcing accountability and unequivocally demonstrating organizational adherence to GDPR principles.

B. Android SDKs and Data Collection Practices

Android third-party SDKs represent a crucial component of the broader Android app development ecosystem. These SDKs offer pre-packaged code and configuration that enable developers to efficiently integrate diverse features into their apps, ranging from user interface elements to complex data analytics capabilities. In this paper, we use the term *third-party SDK* in a narrower sense than *third-party library*. Generic third-party libraries, which have been extensively covered in prior research [18]–[21], are usually referred to as reusable code components with a narrow functional scope. The most widely used libraries on Android include Retrofit, Picasso, Espresso, and Glide [22]. By contrast, an SDK is a vendor-maintained toolkit that bundles one or more libraries together with documentation and configuration to support a specific platform or online services [23], [24]. On Android, such SDKs are typically distributed via a canonical Maven repository [25] and listed in dedicated catalogues, such as the Google Play SDK Index [15]. Representative third-party SDK providers for Android include Google Ad Mobile (GMA), Facebook, and NAVER Map [15].

Third-party SDKs are typically implemented with complex code bases, realize commercial business logic, and often establish data flows that connect to the SDK providers' server over the Internet [26]. For that reason, the data collection practices associated with Android third-party SDKs have been a subject of scrutiny within academic and industry communities. It has been observed that some SDKs, while offering useful functionality, also engage in extensive data collection activities without obtaining explicit acknowledgment or consent from end-users [12], [26]. Furthermore, there are instances where data are transmitted to remote servers that raise concerns about data security and potential misuse of personal information [8]. In light of this, there is an increasing need for a large-scale privacy assessment of SDKs.

In this work, unless otherwise specified, we use the term *SDK* exclusively to refer to Android third-party SDKs that are integrated into Android apps as reusable components. This notion is distinct from the *official Android SDK*, which denotes the platform's development toolkit [27], [28].

C. Objectives and Problem Definitions

This work aims to (1) detect potential exfiltration of users' personal data existing in the third-party SDKs, (2) explore the privacy disclosure compliance of developers in the SDK release, and thereby, (3) unveil the landscape of privacy protection in the Android ecosystem at the SDK level. To this end, we propose a compliance model to systematically assess the privacy practices of Android SDKs with respect to the collection of personal information. We let $\mathbb{D}$ represent a set of diverse data considered as user privacy, and let $\mathbb{S}$ be the set of publicly released third-party SDKs. Then we define what personal data an SDK claims to collect and actually collects as follows.

Compliance Disclosure ($\mathcal{C}$). We let $\mathcal{C}$ denote the compliance disclosure of an SDK $s \in \mathbb{S}$. Specifically, it defines the set of personal data claimed to be accessed in the privacy policy of the SDK s. It shall also be assumed not to leak any personal data to the public.

Implementation-level Practice ($\mathcal{P}$). The compliance practice of an SDK concerns two types of operations in its implementation, i.e., collecting personal data, and sharing the collected personal data. We let $\mathcal{R}_s$ denote the scope of personal data that would be read/collected by an SDK s based on its implementation, in which $\mathcal{R}_s \subseteq \mathbb{D}$. We also let $\mathcal{U}_s$ denote the collected personal data involved in any uploading/sharing operations, given that $\mathcal{U}_s \subseteq \mathcal{R}_s$. Thus, we have the compliance practice of an SDK s as a tuple of its collecting and sharing behaviors, written as $\mathcal{P}_s = (\mathcal{R}_s, \mathcal{U}_s)$.

Next, we formalize three types of compliance issues that are of concern in our investigation.

Type I: Privacy Leakage. Sharing users' privacy in public spaces poses compliance risks, especially when the collection or the sharing operations are not performed with users' consent. Moreover, some regulations like the GDPR have explicitly prohibited any collection of UUIs, amplifying the risks of privacy sharing on the SDK side. Here we define that any pre-recognized personal data d being observed in sharing operations of an SDK s would constitute privacy sharing risk, which we formalize as compliance not being true (i.e., strikethrough in a unary turnstile, $\nvdash$) due to implementation-level practice $\mathcal{P}$. We detail this as follows:

$$\exists s \in \mathbb{S}, \exists d \in \mathbb{D}, d \in \mathcal{U}_s \Rightarrow \nvdash_{\mathcal{P}} \mathcal{C}_s. \tag{1}$$

Besides improper sharing, recent developments in privacy protection regulations also pose stricter requirements for even reading personal data on personal devices. Developers are required not only to declare a list of personal data to be collected but also to ensure that the actual collection operations are always consistent with what has been declared. Next, we formalize two types of inconsistency issues that could happen in personal data collection.

Type II: Excessive Collection. Excessive collection concerns the privacy practice of an SDK that attempts to read more types of personal data than what it requests/claims in its privacy policy. Excessive collection infringes on users' right to be informed, as the SDK collects personal data without users' awareness and consent. We define this type of risk as follows.

$$\exists s \in \mathbb{S}, \exists d \in \mathbb{D}, (d \in \mathcal{R}_s) \wedge (d \notin \mathcal{C}_s) \Rightarrow \mathcal{R}_s \nvdash_{\mathcal{P}} \mathcal{C}_s. \tag{2}$$

We remark that Google has not set a formal definition of "data collection" at the SDK level, as it does for third-party apps [29]. Thus, we refer to the guideline of GDPR Article 13 [13], [30] and take a conservative strategy by considering data accessed by SDKs as *collected* when evaluating the excessive collection, since the data accessed by an SDK can flow into the host apps, which can then consume or send the data out.

Type III: Over-claiming. We assert that an SDK has an over-claiming issue if it is found to claim access to more types of personal data than it actually reads. In other words, over-claiming indicates that an SDK declares its access to data types that it does not actually read. Over-claiming issues, although they may not substantially lead to personal data exfiltration, seriously violate the *data minimization principle* stipulated in Article 5(1)(c) of GDPR. It is defined as follows.

$$\exists s \in \mathbb{S}, \exists d \in \mathbb{D}, (d \notin \mathcal{R}_s) \wedge (d \in \mathcal{C}_s) \Rightarrow \mathcal{R}_s \nvDash_{\mathcal{P}} \mathcal{C}_s. \quad (3)$$

D. Threat Model

We consider that the exfiltration of users' personal data caused by malicious SDKs should follow a unified routine. To achieve such exfiltration, the malicious SDK (the attacker) gains capabilities as follows:

- The malicious SDK is released in a public platform/market and thus, can be embedded in an arbitrary app and installed on an Android device. Without causing an advantage to the attacker, that device is not rooted and is properly maintained with the latest security update installed.
- The malicious SDK accesses users' personal data through the interfaces of the Android OS with permissions granted to the embedding app, i.e., all its collection of users' personal data must be realized on behalf of the embedded app.
- The malicious SDK saves or shares the collected personal data in a public space, i.e., there exists no party (including the embedding app) colluding with the malicious SDK in sharing the collected personal data, or saving it in private storage.

III. Methodology

A. Research Questions and Approach Overview

Our study investigates the privacy disclosure compliance of third-party SDKs. Specifically, we analyze both the privacy policy and the implementation of each SDK, and check if their privacy practice perfectly matches. To this end, our study aims to answer the following three research questions (RQs):

- **RQ1**. Can our privacy policy analysis identify the personal data claimed to be collected by SDKs on a large scale? What is the performance (i.e., precision and recall) of our NLI model-based approach?
- **RQ2**. Can our taint analysis find personal data collection and sharing behaviors from the SDKs? How many tainted traces are detected? Can those tainted traces (really) leak users' privacy?
- **RQ3**. By cross-checking their privacy policies, what is the *status quo* of personal data protection at the SDK level? Do the collected SDKs comply with their privacy policies?

To address the three research questions, we propose a four-phase approach, which is briefly illustrated in Figure 1. In ***Phase 1***, we collect 158 Android SDKs, including their binary files, privacy policies, and metadata from two release platforms. After that, ***Phase 2*** involves analyzing the privacy policies using Natural Language Inference (NLI) models to identify the types of personal data requested by the SDKs. In ***Phase 3***, we perform static taint analysis on the SDKs to examine their data collection behaviors at the implementation level, aiming to identify traces that constitute a complete personal data reading and sharing process. SDKs found to read and share user privacy would be prone to privacy leakage (i.e., type I issues). Finally, in ***Phase 4***, we assess privacy disclosure compliance by cross-checking the data requested in the privacy policies with the actual data collection behaviors, aiming to detect excessive collection (Type II) and over-claiming (Type III) issues. Given the time-intensive nature of static taint analysis and the substantial human efforts required to interpret its outcomes, two authors were involved in the Phase 3 and Phase 4. Specifically, they independently assessed the validity of the tainted traces and cross-validated their respective findings.

B. Scope of Personal Data

Our study aims to cover as many distinct privacy-related data types as possible, with the goal of presenting a comprehensive view of the privacy disclosure compliance landscape at the level of third-party SDKs. To this end, we draw on multiple sources to construct the set of privacy-related data types.

Since most privacy-related data requires explicit permissions for apps to access them, we primarily resort to the official list of Android permissions [31] to define our initial set of personal data, from which we identify 12 types of privacy-related data, such as *device identifiers*, *location*, *contacts*, and *calendars*. Next, we further complement the set with privacy-sensitive items recognized in the "privacy or security change" section of each Android OS release report, for example, the *clipboard* and *app list* were highlighted in the release documents of Android 10 [32] and 11 [33] and are therefore added to the scope of this study. During this step, we also identify a group of user-unresettable identifiers (UUIs) as a subset of device identifiers, such as *IMEI*, *IMSI*, and *ICCID*, whose access by apps has been restricted since Android 10 [32].

In addition, we take into account existing literature on personal information collection [34]–[39] and app analysis [40]–[43] to further refine and complement the list of personal data. We thus include access to a variety of on-device sensors (e.g., *motion* and *light sensors*) and two software-defined identifiers (i.e., *Google Ad ID* and its alternative in the Chinese market, *OAID* [44]) in the list. As a result, we recognize 41 types of privacy-related data. These data types constitute the set $\mathbb{D}$ defined in Section II-C and formulate the scope of our compliance assessment. The full list of recognized personal data is shown in Table I.

Categorization. To facilitate the subsequent presentation and discussion of our findings, we further categorize these 41 data types into five groups according to three criteria:

- We primarily group those personal data that share a similar functionality or purpose. This is to facilitate our privacy policy-based compliance assessment, as many developers may tend to only declare the data collected roughly by functionality (e.g., device identifiers, location) rather than accessing specific data or certain APIs.

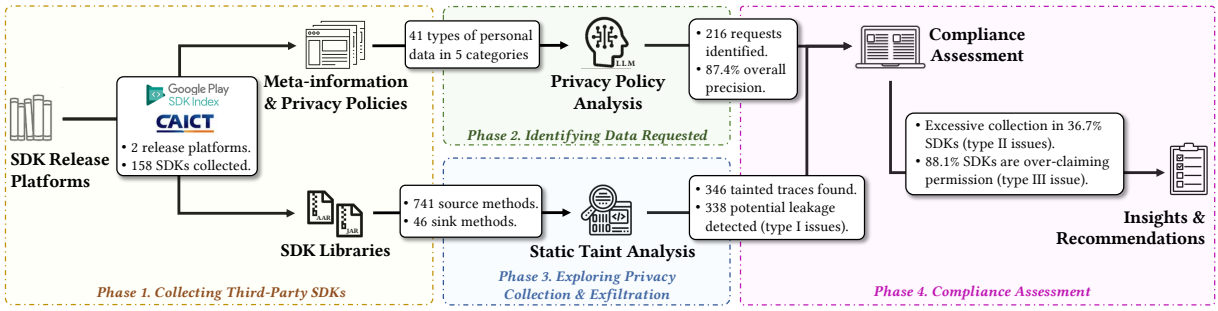


Fig. 1: The workflow of potential data exfiltration exploration and compliance assessment at the SDK level

TABLE I: List of 41 pre-identified personal data types, their permission requirements for API access, and the distribution of the 741 source methods (data types without a annotation do not require permission at dangerous level or above)

Data type & class	Count	Data type & class	Count
(C1) Chip, Cellular & Peripheral		Wifi*	38
Call phone*	41	Wifi MACs†	5
Carrier info†	5	*Subtotal*	*(245)*
Cellbroadcast†	2	**(C3) Location & Sensors**	
External storage*	18	Camera*	54
ICCID†	15	Location*,‡	87
IMEI†	5	Misc. sensors	140
IMSI†	3	*Subtotal*	*(281)*
MEID†	2	**(C4) Media & Software Specific**	
Network info	6	Android ID	2
Phone status*	1	App list†	7
SD card serial*	4	Audio record*	22
Serial†	4	Google Ad ID	3
SIM info†	7	Media files*	1
SMS*	23	OAID	9
Telephone number*	12	Screen record & screenshot†	1
Subtotal	*(148)*	*Subtotal*	*(45)*
(C2) Wireless Communication		**(C5) Misc. Personal Data**	
Bluetooth*	117	Account info*	6
Bluetooth call*	6	Browser bookmarks†	1
Bluetooth ID*	2	Calendar*	2
Bluetooth MAC†	36	Clipboard data†	2
IP address	2	Contact list*	9
SIP service (VOIP)*	21	Contact log*	2
SSID/BSSID*	17	*Subtotal*	*(22)*
Ultra-wideband*	1	**Total**	**741**

* API access requests at least one *dangerous* permission that demands explicit approval from the user at runtime.
† API access request a *system/privileged* permission, or the data itself is app-private. Third-party apps without system-level privilege cannot access through API invocation.
‡ Includes coarse location that is determined by network, and fine location that is jointly determined by network and GPS.

- We then refine our categorization by grouping data that can be programmatically accessed through APIs into the same or similar classes, based on the Android developer's documentation. This is to facilitate our next step's taint analysis as we need to collect as many involved APIs as possible to be the source methods, which we will detail in Section III-E.
- We also consider the API permissions required for collecting personal data into our categorization. Due to the historic changes in Android privacy mechanisms, primarily reflected in the permission rules for calling sensitive APIs, grouping personal data that involves APIs mentioned in the same privacy change can facilitate our discussion and provide insights into any non-compliance observed. A typical example is that the invocation permission of most APIs to read device identifiers, including serial, IMEI, and ICCID, is escalated in Android 10. We accordingly group these aforementioned data into one category.

Based on the criteria listed above, we further categorize them into five classes, namely *chip, cellular & peripheral* (**C1**), *wireless communication* (**C2**), *camera, location, and other sensors* (**C3**), *media and software specific* (**C4**), and *miscellaneous personal data* (**C5**).

Ontology for Semantic Relationships. Considering the request of personal data in privacy policies may not always be written in a precise manner as defined in the developer's documentation, and sometimes certain personal data items have multiple names or colloquial phrases, we refer to relevant research [19], [45] and propose an ontology to capture the semantic relationship among the 41 types of personal data, and consequently to guide our assessment.

First, we manually collect alternative names or colloquial phrases for each type of identified personal data and treat them as *synonyms* ($\equiv$). For example, we find that the advertising ID and GAID are interchangeably used in privacy policy documents to refer to the Google Ad ID. Thus, all the synonymous terms of a personal data type will be considered in our assessment.

Besides that, we define a *hypernymy relationship* ($\sqsubset$) between personal data types in generic and specific terms. For example, we learn that both the user name and email are compositions of the account information of an Android device, and accordingly treat both user name and email as *hyponyms* of the term account information (denoted as `name` $\sqsubset$ `account_info` and `email` $\sqsubset$ `account_info`). In case an SDK developer claims to request the email address of the device owner, we assume the API for reading account information will be invoked, and therefore, both the email and name of the user will be accessed.

We also learned that some SDK developers use the rough term "device identifiers" in disclaiming personal data to collect. Unlike the definition of "`DeviceID`" in Android documentation, device identifiers are often referred to as a broad sense of hardware identifiers in privacy policies, such as the serial number of the smartphone. Thus, we consider device identifiers as a hypernym of diverse hardware identifiers (e.g., `serial` $\sqsubset$ `device_identifiers`). We adopt a conservative stance in this study and, therefore, assume that the involved SDKs claim to request all types of personal

data that fall within the scope of device identifiers, including IMEI, MEID, ICCID, and serial numbers. Furthermore, we assume the term "location" and its synonyms refer to data derived from the SDK's interaction with Android's location services (e.g., `LocationManager`), although a user's location can also be inferred through his/her device's IP address [46]. Our taxonomy considers IP address as a dedicated data type rather than a synonyms of location data mainly due to two reasons: (1) an IP address does not necessarily reflect a device's precise physical location, and can be substantially distorted by network configurations such as VPNs and carrier routing; and (2) the location-related permission does not apply to the access of IP address. Specifically, accessing location through location APIs requests permission at "dangerous" level (e.g., `ACCESS_COARSE_LOCATION` and `ACCESS_FINE_LOCATION`), whereas accessing IP address is often treated as essential networking behaviors [46]. For these reasons, when a policy claims that an SDK collects "location", we assume this corresponds to location-service-based access to minimize ambiguity. An SDK claiming to access users' locations through any manner other than location services without providing an explicit explanation would be considered non-compliant in this study.

C. Phase 1: Collecting Android SDKs

Our study starts by collecting as many Android SDKs as possible from mainstream sources. Since our approach involves cross-checking SDK privacy disclosures with actual data collection behaviors at the implementation level, we gather not only SDK binaries (typically in AAR and JAR formats) but also the privacy policies and relevant metadata provided by the developers, when available. Although third-party SDKs are now a common concept in the Android developer community, Google does not provide a precise and official definition of what constitutes a third-party SDK. As a result, there is no strong industry-wide consensus on where to draw the boundary between a generic library and an SDK [24]. Therefore, we base our SDK dataset on major platforms that explicitly verify and publish third-party SDKs for Android apps.

Sourcing SDKs from Multiple Repositories. We identify the Google Play SDK Index as the primary source for collecting Android SDKs [23]. Additionally, considering the uniqueness of mainland China's Android ecosystem, where Google Play infrastructure is absent, we use CAICT[1] as a supplementary source. CAICT SDK repository is a joint initiative of the Chinese government, academia, and industry to provide a centralized third-party SDK hosting platform since 2020 [16]. As a result, we download the latest version of 158 distinct SDKs as of October 2022, among which 139 are from the Google Play SDK Index and 19 are released on the CAICT website. These 158 constitute our previously defined $\mathbb{S}$ (Section II-C) for the later assessment. We also revisit the latest versions of the same group of SDKs in October 2023 and conduct an additional assessment, which we will detail in Section IV-D.

[1]Acronym of China Academy of Information and Communications Technology

We note that, at the time of this study, neither the Google Play SDK Index nor CAICT had implemented strict regulation or code auditing mechanisms comparable to those of the Google Play App Store. As a result, there is no guarantee that the SDKs from these sources include comprehensive documentation or privacy policies required by regulations such as GDPR.

Crawling Metadata and Privacy Policies. Next, we implement a crawler using Selenium [47] to retrieve metadata and privacy policy links from the two sources. The metadata includes the SDK name, provider (developer's identity), Maven IDs, number of installs, and functional categories. Such information will be jointly used to uniquely identify SDKs during our subsequent analysis. We then collect the privacy policies from the retrieved links. We will use them to extract the list of personal data that the SDK requests.

The retrieval of privacy policies is found to be more complex than collecting metadata due to the lack of a compulsory standard for drafting them. For SDKs from the Google Play SDK Index, we rely on the links provided in the "data safety" section if available, and download the corresponding webpages. Similarly, for SDKs from the CAICT website, we download the webpages from the "SDK privacy policy" link on the SDK details page. We use Google Translate to process all non-English text on these webpages.

Due to the limited availability of privacy policies, we only collected policies for 52 SDKs directly from the two platforms. For those that do not provide privacy policy URLs, we manually searched the Internet using keywords like "disclaimer", "policy", "disclosure", and "guidance" to look for any available privacy policies drafted by the SDKs' developers.

SDK–Privacy Policy Matching. To ensure that each Internet-retrieved privacy policy corresponds to the examined SDK, we applied a verification procedure based on multiple criteria. First, we matched policies using SDK metadata obtained from the release platforms, including the SDK name, SDK identifier (e.g., Maven ID), and the vendor name. Second, we verified that the retrieved policy was hosted on the official domain of the SDK vendor and that its scope covered the corresponding SDK or product family.

We note that some vendors may provide a single privacy policy covering multiple products or services rather than SDK-specific policies. In such cases, we treat the vendor-level policy as applicable to the examined SDK unless the policy explicitly states that it does not apply to the SDK or related software components. This assumption follows common practice in prior studies and reflects how privacy disclosures are typically structured in the mobile ecosystem. Accordingly, only privacy policies whose association with an SDK could be confidently established were included in our assessment. During dataset construction, we did not observe cases where multiple privacy policies explicitly applied to a single SDK.

Using this procedure, we identified privacy policy webpages for an additional 57 SDKs. In total, we obtained valid privacy policies for 109 SDKs (69.0% of all SDKs). We then extracted the text from these policies for further analysis.

Representativeness of the Collected SDKs. We emphasize that our collection of 158 SDKs does not exhaust all third-

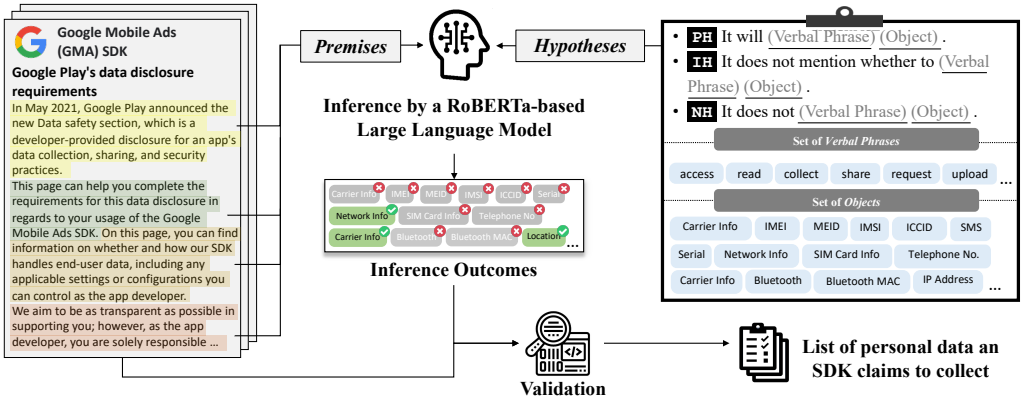


Fig. 2: The workflow of our privacy policy analysis to find out what an SDK claims to collect

party SDKs in the Android ecosystem. However, it already contains many well-known and heavily used SDKs, such as Google Mobile Ads and Baidu Map SDKs. By crawling the "number of installs" metadata from the Google Play SDK Index, we observe that more than 100 SDKs in our collection have been embedded by apps with over 10 million installations (as of November 2023). These observations suggest that, although our SDK collection is not fully exhaustive, our assessment is sufficiently representative to reveal the privacy disclosure compliance landscape of Android third-party SDKs at scale.

D. Phase 2: Identifying Data Requested in Privacy Policies

After collecting the 109 privacy policies, our next step aims to identify the data requested by each SDK from them. In contrast to traditional app privacy policies that are written in *user-oriented* language, SDK privacy policies are more *developer-oriented* and may contain much domain-specific terminology (e.g., ICCID, BSSID) and technical synonyms (e.g., Google Ad ID, which also appears as Advertising ID, or GAID). This uniqueness makes it challenging for existing methods [19], [45], [48]–[51] to extract domain-specific entities accurately. To tackle these challenges, we leverage large language models (LLMs) to infer the data collection practices from the privacy policies.

Adoption of NLI Model. Recent advances in LLMs have led to their adoption in many NLP tasks. In designing our privacy policy analysis, we draw on a recent work [21], which shows that LLM-based methods can achieve similar or even higher performance than conventional NLP techniques (i.e., sentence-level syntactic and semantic analysis) while avoiding labor-intensive but error-prone rule engineering, e.g., constructing verb lists and synonym sets for privacy-related entities. Motivated by these observations, we adopt an LLM-based approach in this work.

We first considered widely used conversational LLMs, such as ChatGPT-4 and Llama 3, in a pilot study of analyzing Google's in-house SDKs. Specifically, we supplied the privacy policy together with hypothesis-style prompts that asked *whether the policy describes the access and/or collection of a particular privacy-related data type*, with the goal of obtaining binary judgments. However, the results were found unsatisfactory. We found that these state-of-the-art conversational large language models (LLMs) often produced open-ended and ambiguous responses rather than providing clear yes or no decisions. We also found that these models tend to produce inconsistent responses when queried multiple times with the same prompt. These LLMs, although extremely powerful in reasoning and natural language generation, lack transparency in decision making, making it difficult to ensure the soundness and reproducibility of our assessment pipeline.

We therefore adopt a more conservative approach by employing a LLM-based natural language inference (NLI) model named `roberta-NLI`[2] to identify requested data in privacy policies [52]. NLI has proven effective in understanding natural language across different domains [53]. In our study, we parse sentences from privacy policies as *premises* and pair them with hypothesis statements instantiated from our 41 pre-identified data types (refer to Section III-B). The NLI model outputs an entailment score for each premise-hypothesis pair, which we use to determine whether the privacy policy discloses the collection of the corresponding data type. Compared with conversational LLMs like ChatGPT, this NLI model-based design provides us with a more transparent and stable decision process for large-scale privacy policy analysis, thereby enhancing the reproducibility of our methodology. We briefly illustrate our approach in Figure 2.

Formulating Prompts. There are two key challenges that arose during the hypothesis formulation. The first challenge is the lack of a canonical writing standard for SDK privacy policies. Privacy policies often use developer-oriented language with technical synonyms, requiring us to manually gather synonyms (e.g., the advertising ID and GAID are interchangeably used to refer to the Google Ad ID) to ensure hypothesis completeness. Besides that, we also tailored hypotheses by matching different verb phrases (e.g., "collect", "read") to each data type and its synonyms to maximize the entailment score when the collection of the inferred data type is mentioned in the privacy policy. As a result, we curate a list of 169 hypotheses covering the 41 pre-identified privacy-

[2]The full name of the model is `roberta-large-snli_mnli_fever_anli_R1_R2_R3-nli`. We shortly write as `roberta-NLI` to save space.

TABLE II: Combination of objects and applicable verbal phrases to form hypotheses

Objects *(Synonyms shown in italic font)*	*Collect*	*Make*	*Read*	*Request*	*Send*	*Take*	*Write*	*Share*
Phone state	•		•	•				•
Carrier info	•		•	•				•
Cellular function	•		•	•				•
IMEI	•		•	•				•
MEID	•		•	•				•
IMSI	•		•	•				•
ICCID	•		•	•				•
Serial	•		•	•				•
Device identifier	•		•	•				•
SIM card information	•		•	•				•
SMS			•		•			•
Message			•		•			•
Phone number	•		•	•				•
SSID	•		•	•				•
BSSID	•		•	•				•
Phone call		•						•
Storage			•				•	•
SD card			•				•	•
Peripheral								•
Network info				•				•
Network state				•				•
Connection status				•				•
Connection info				•				•
Bluetooth				•				•
Wifi				•				•
IP address	•			•				•
MAC address	•			•				•

Objects *(Synonyms shown in italic font)*	*Collect*	*Make*	*Read*	*Request*	*Send*	*Take*	*Write*	*Share*
VOIP call		•						•
UWB				•				•
Ultra-band Broadcasting				•				•
Location	•			•				•
GPS	•			•				•
Camera				•				•
Sensor				•				•
Android ID	•		•	•				•
Google Ad ID	•		•	•				•
Google Advertising ID	•		•	•				•
GAID	•		•	•				•
OAID	•		•	•				•
App list	•			•				•
Package list	•			•				•
Installed apps	•			•				•
Audio				•				•
Recording		•		•				•
Media				•				•
Screenshot		•	•				•	•
Screen			•					•
Account info	•		•	•				•
Username	•			•				•
Email address	•			•				•
Clipboard	•		•	•				•
Contact	•		•	•				•
Calendar	•		•	•				•
Browser bookmark	•		•	•				•
Total combinations								**169**

related data types. A summary of these hypotheses can be found in Table II.

The second challenge is handling the complex context of SDK privacy policies, as the NLI model is not domain-specifically trained. For example, we found common sentence formats in privacy policies, such as "According to GDPR, all collection of device identifiers must be declared in advance." Conventional inference models might incorrectly conclude with high confidence that this implies the *collection of device identifiers*, but it is irrelevant to the actual behaviors of the SDK being analyzed. To address this issue, we created three tailored hypotheses for each combination of verbal phrase and object:

1) **Positive hypothesis** (shown as **PH** in Figure 2, e.g., "It *will* collect your IMEI", "It *will* share your location", etc.).
2) **Negative hypothesis** (**NH**, e.g., "It *does not* collect your IP address", "It *does not* make screenshot", etc.).
3) **Irrelevant hypothesis** (**IH**, e.g., "It *does not mention whether to* collect your phone number").

By applying the 169 hypotheses to the three different tones (i.e., positive, negative, and irrelevant), we compose a list of 507 prompts for the next step of inference by the NLI model.

Inference of Data Requested. Given that three hypotheses have been defined for each combination of verbal phrase and object, we only consider a data type claimed to be collected by an SDK if there exists *at least one* combination of the associated object (e.g., location) and applicable verbal phrase (e.g., request) satisfying the logical condition as follows:

$$(\text{Score}_p > T) \wedge (\text{Score}_p > \text{Score}_n) \wedge (\text{Score}_p > \text{Score}_i)$$

where Score_p indicates the entailment confidence score for the *positive hypothesis*. An entailment inference requires Score_p to be greater than a threshold value, written as T. Simultaneously, Score_p must be greater than the scores for *negative* and *irrelevant hypotheses* (i.e., Score_n and Score_i), indicating that the premise does not only "mention" the collection of that data type but also confirm that collection behavior is "performed" by the SDK itself. All these scores are obtained directly from the NLI model's output and range from 0 to 1.

Ground Truth Construction and Threshold Tuning. Intuitively, to infer a specific type of data being collected, the entailment prediction score (Score_p) is supposed to be greater than 0.5 to ensure it is always higher than the prediction of the neutral and contradiction hypotheses. However, our preliminary experiments observe a large number of false positives by setting the threshold to 0.5. For that reason, we perform a small-scale analysis to tune the threshold value from 0.5 and above until we find an optimal value that minimizes the number of false positive predictions. Specifically, we assume that Google should provide high-quality privacy policies for its own SDKs and therefore, we collected five distinct privacy policies shared by 13 Google in-house SDKs.

We then manually analyzed the declared personal data types in the five privacy policies to construct the ground truth. One author first reviewed each privacy disclosure document and identified statements describing the collection or access of personal data according to the 41 predefined data types in our taxonomy. The annotations were recorded in a structured annotation table by marking the corresponding data types.

A second author then independently reviewed the same privacy policies and verified each annotated data type. When discrepancies occurred, the two authors discussed the cases and reached a consensus. To improve efficiency while maintaining accuracy, we adopted a keyword-guided reading strategy to locate relevant sections describing data collection and handling.

The annotation process was guided by the semantic ontology introduced in Section III-B, which defines synonym and hypernym relationships among the data types. This ontology helps reduce ambiguity when interpreting the wording of privacy policies and ensures consistent identification of data types across annotators.

Using the ground truth obtained through this procedure, we fine-tuned the threshold value and found that the NLI model achieved complete agreement with the ground truth when the threshold was set to 0.73. Therefore, we adopt 0.73 as the optimal threshold in our large-scale privacy policy analysis.

E. Phase 3: Exploring Privacy Collection and Leakages

We leverage the static taint analysis techniques to explore privacy collection behaviors and potential exfiltration in SDKs. We define valid personal data exfiltration as consisting of a *collection action* that queries one or more personal data instances from the OS, and a *sharing action* that shares the obtained personal data, directly or indirectly, to the public spaces, including the Internet, system settings, and the public storage. Thus, we consider that an SDK engages in privacy collection if its implementation contains a data flow originating from a personal data collection action. This privacy collection leads to exfiltration if the data flow ends at a sensitive sharing action.

Sourcing Taint Analysis Tools. For the selection of a taint-analysis engine, our preliminary study considered several

TABLE III: Taint analysis outcomes for the six sampled SDKs

Tool	Tainted traces	Valid traces	Recall	Precision
FlowDroid [54]	0	0	–	–
Mariana Trench [56]	6	1	1.3%	16.7%
AppShark [57]	30	26	36.1%	86.7%

popular open-source tools that are applicable to Android apps, including FlowDroid, Mariana Trench, and AppShark. FlowDroid [54] is a widely adopted static analysis framework in the research community and has been regarded in earlier studies [55] as one of the best-performing analyzers. However, it has some known limitations in certain cases. For example, FlowDroid often fails to model data flows involving reflection and misses data flows propagated through Android-specific callbacks. In contrast, Mariana Trench [56] and AppShark [57] are two later open-source tools that explicitly claim better support for taint analysis in modern Android apps. Motivated by these observations, we conducted a pilot comparison of the three tools.

Comparison of Available Tools. We randomly select three SDKs from each repository to conduct the preliminary study. As a result, the three taint analysis tools were evaluated with six SDKs, namely Facebook Places 7.1.0 (with file hash `3af9bc`), GMS Play Services Map 18.1.0 (`8a00b7`), Yandex Maps Mobile Lite 4.3.1 (`f49896`), Alipay SDK 15.8.12 (`c3458e`), HiPush 6.0.4 (`97f9d3`), and AMap 3D Map 9.6.0 (`bf6a42`). Among which the first three are sampled from the Google Play SDK Index, and the latter three are from the CAICT repository. These six SDKs are later embedded into blank apps and passed to the three candidate tools for static taint analysis.

We run all three analyzers on the six apps that solely embed the sampled SDKs, and manually examine the validity of the caught traces (refer to Section IV-B for validating tainted traces). For all tools, we set the maximum data-flow depth to 30, while keeping all other configuration parameters at their default values.

In parallel, we manually analyzed the data collection behaviors of these apps at the source code level. Specifically, we decompiled the apps, searched for all occurrences of source methods, and then manually inspected the corresponding data flows. Each app was independently examined by two authors, and the final ground truth was obtained by comparing their findings and reconciling any disagreements. We remark that this manual inspection is extremely labor-intensive and its duration is hard to predict. During our inspection of the six SDKs, even two of which (i.e., GMS Play Services Map and HiPush) turned out not to access any privacy-related data – our manual analysis still took three man-days in total.

We summarize the results of our pilot study in Table III. Overall, AppShark achieved the strongest performance on the tested six SDKs. FlowDroid failed to capture data flows that are propagated through callback parameters, while Mariana Trench often timed out due to significantly long analysis time, which appears to be a known issue that has been discussed in the community [58]. Since developing a static analysis tool is not the primary focus of our paper and suitable alternatives already exist, we decided to use AppShark for our taint analysis rather than investing effort in troubleshooting or extending the other two frameworks.

We note that the valid traces identified by our analysis may represent only a small fraction of the SDKs' actual privacy-related behaviors. Many traces are excessively long and complex, making them difficult to recover within a reasonable time budget even for state-of-the-art tools. In our pilot study, we observed only 36.1% recall, given 72 traces were later identified by manual analysis. The absence of a tainted trace should not be interpreted as definitive evidence of privacy disclosure compliance. Meanwhile, large-scale manual inspection of SDK implementations – although it can find missed tainted data flows – is technically infeasible in practice. These considerations motivate us to examine both tainted and non-tainted traces (refer to RQ2) to better characterize the landscape of privacy disclosure compliance.

TABLE IV: A brief statistics of recognized sink methods

Purpose	Class name	Count
Network Transfer	`java.net.HttpURLConnection`	1
	`java.net.URL`	2
	`java.net.URLConnection`	2
	`org.android.spdy.SpdyRequest`	12
	`org.android.spdy.SpdySession`	3
	`org.apache.http.client.methods.HttpGet`	8
	`org.apache.http.client.methods.HttpPost`	8
	`org.apache.http.params.HttpParams`	1
File Saving	`java.io.Writer`	5
	`java.io.FileOutputStream`	1
System Settings	`android.provider.Settings`	3
Total		**46**

Configuring Taint Analysis. We aim to collect all relevant APIs that contribute to a collection action and include them in the *source methods* for taint analysis. To this end, we refer to an existing study [59] which proposes a technique named `SuSi`, which classifies over 900 API methods that can potentially leak users' privacy in Android 4.3 and earlier versions. Considering there have been multiple versions of evolutions since Android 4.3, the API rules have experienced significant change; we first map the APIs identified in [59] with the 41 types of personal data. We iterate through the developers' documents of those APIs and filter out those APIs that are no longer valid in the Android Open Source Project (AOSP) implementation. We then manually visit the historic developer's document since Android 4.3, and jointly refer to the AOSP source code to collect as many APIs that are associated with the new data types as possible. In that way, we identify candidate source methods for biometric sensors, secondary camera, and UWB communications. In the end, we collected 741 source methods.[3] The distribution of 741 methods corresponding to the 41 types of personal data can be found in Table I.

Next, we retrieve all methods that perform a sharing action from Android SDKs. Although there are not many new APIs introduced after Android 4.2 that can be treated as sink methods, we find most sink methods studied in earlier literature including `SuSi` [59], such as logging and SQLite databases, are no longer valid in the latest Android versions and are therefore excluded in this study [60], [61]. In the end, we identify 46

[3]We count the number of methods by unique method signatures in this paper.

TABLE V: Evaluation outcomes of privacy policy analysis (only data types requested are displayed)

Data type & class	#SDKs detected to collect	TP	FP	FN	Precision	Recall	F1-score	Data type & class	#SDKs detected to collect	TP	FP	FN	Precision	Recall	F1-score
C1 Carrier info	12	10	2	3	83.3%	76.9%	80.0%	**C3** Location	28	25	3	4	89.3%	86.2%	87.7%
Device identifiers*	21	21	0	3	100.0%	87.5%	93.3%	Misc sensors	1	1	0	0	100.0%	100.0%	100.0%
Network info	30	25	5	0	83.3%	100.0%	90.9%	Android ID	8	8	0	0	100.0%	100.0%	100.0%
SIM card info	7	7	0	0	100.0%	100.0%	100.0%	App list	4	4	0	0	100.0%	100.0%	100.0%
SMS	7	3	4	0	42.9%	100.0%	60.0%	Audio record	10	10	0	0	100.0%	100.0%	100.0%
Telephone No	5	3	2	2	60.0%	60.0%	60.0%	Google Ad ID	6	6	0	0	100.0%	100.0%	100.0%
C2 BSSID/SSID	4	4	0	0	100.0%	100.0%	100.0%	OAID	2	2	0	0	100.0%	100.0%	100.0%
IP address	47	45	2	1	95.7%	97.8%	96.8%	**C4** Account info	10	4	6	4	40.0%	50.0%	44.4%
WiFi	4	3	1	0	75.0%	100.0%	85.7%	Clipboard	1	1	0	0	100.0%	100.0%	100.0%
C3 Camera	7	5	2	0	71.4%	100.0%	83.3%	**Total**	**214**	**187**	**27**	**17**	**87.4%**	**91.7%**	**89.5%**

* We use a broad sense of device identifiers to represent all hardware UUIs, including IMEI, MEID, IMSI, ICCID, and serial.

methods from 11 classes and then configure them as *sink methods*. We present a brief statistics of the sink methods by their containing classes in Table IV.

The majority of sink methods are relevant to data transfer via network interfaces, reserving 37 out of 46 identified method signatures. Besides that, six methods are recognized because they can share sensitive data via public storage. The remaining three sink methods are counted because they can be used for modifying the system settings, which is known as an undocumented channel to share data among different parties on the device [62]. We note that writing contents into the system settings requires privileged permission, and we will discuss it in Section V.

IV. Evaluation

Our evaluation aims to assess the performance of our proposed NLI model-based privacy policy analysis and explore the personal data collection and potential leakages in the SDKs. After that, we analyze the privacy preservation landscape at the SDK level by cross-checking the data requested in the privacy policies and the data collected in SDKs' implementation (refer to Phase 4 of Figure 1).

Implementation and Experimental Settings. We implement our privacy policy analysis in Python with PyTorch. The NLI model is provided by a Python package named *Transformers*[4]. We run all the analysis on an Ubuntu PC equipped with an Intel Core i7 12700K CPU and 64GB RAM and an NVIDIA RTX 3060 GPU with 12GB VRAM. We set the maximum data flow depth to 30 and the timeout to 300 seconds for our static analysis. Our artifacts can be found at https://doi.org/10.5281/zenodo.18737114.

A. RQ1: Privacy Policy Analysis Performance

We perform NLI model-based analysis on the 109 SDK privacy policies and present the analysis outcomes in Table V. Our analysis identifies 214 data requests from privacy policies. Among the 214 data requests, IP address accounts for the largest share, contributing to 47 (21.9%) occurrences. Network information and location are the remaining data types among the top three. From the perspective of SDKs, the number of data types claimed by an SDK ranges from 0 to 6, with an average of 1.9.

In parallel with the NLI model-based analysis, we follow the manual annotation procedure described in Section III-D to examine the 109 privacy policies. As a result, we identify 187 verified data requests covering 19 data types. These verified requests constitute the ground truth used to evaluate our analysis approach. The results show that our approach can effectively analyze privacy policies at scale, achieving a precision of 87.4%, a recall of 91.7%, and an F1 score of 89.5% (see Table V).

Error Analysis. Our evaluation shows that the account information (6), network information (5), and SMS (4) are the top three data types involved in false positives, i.e., the NLI model mistakenly determined the access of such data from the privacy policies. On the other hand, account information (4), carrier information and telephone number (5 in total), and device identifiers (3) are the data types that appear in the most false negative cases (i.e., our analysis failed to recognize the relevant disclosure). These results indicate the model struggled to infer data collection for certain types.

After manual inspection, we find that two false negatives are caused by the NLI model that is unable to process tables in markdown format[5]. Apart from these two cases, all remaining errors stem from the model's misinterpretation of terms and data type scopes. For example, although our NLI model correctly recognizes SMS, it fails to differentiate SMS and other messaging methods like emails, leading to high-confidence false positives while dealing with sentences like "we may send an email to the users..."[6] as SMS collection. Additionally, our analysis also struggles when inferring dedicated terms in the Android context, but reflects a broader scope based on the knowledge of the pre-trained LLMs. For example, we find that our NLI model failed to distinguish *account information* (i.e., account type and username) and *real names* when processing a sentence writing "... the way users refer to themselves, such as their first or last name."[7], for which the NLI model mistakenly determines that the collection of account information is mentioned, although the privacy policy does not intend to do so.

[4]Available at https://huggingface.co/docs/transformers/index

[5]This happened in analyzing the privacy policy of `com.razorpay.checkout`.

[6]Excerpt from the privacy policy of `com.zendesk.messaging`.

[7]Excerpt from the privacy policy of `clevertap.android.sdk`.

TABLE VI: A brief statistics of the taint analysis results (only data types called and/or tainted are displayed)

Data type & class	Caught access traces: Not tainted	Tainted (Type I)	Total	Data type & class	Caught access traces: Not tainted	Tainted (Type I)	Total
C1 Carrier info	106	0	106	**C2** WiFi MAC	62	64	126
IMEI	92	26	118	*Subtotal*	*(356)*	*(172)*	*(528)*
MEID	0	4	4	**C3** Camera	3	0	3
IMSI	15	5	20	Location	486	130	616
ICCID	20	4	24	Misc sensors	383	0	383
Network info	2	0	2	*Subtotal*	*(872)*	*(130)*	*(1002)*
Serial	82	5	87	**C4** Android ID	83	0	83
SIM card info	14	0	14	App list	128	0	128
SMS	6	0	6	Audio record	11	0	11
Telephone no	33	0	33	Google Ad ID	20	0	20
Subtotal	*(370)*	*(44)*	*(414)*	OAID	68	0	68
C2 Bluetooth	20	0	20	*Subtotal*	*(310)*	*(0)*	*(310)*
Bluetooth MAC	10	0	10	**C5** Account info	2	0	2
BSSID/SSID	220	107	327	Clipboard data	57	0	57
IP Address	23	1	24	Contact list	5	0	5
WiFi	21	0	21	*Subtotal*	*(64)*	*(0)*	*(64)*
				Total	**1972**	**346**	**2318**

B. RQ2: Privacy Collection and Potential Leakages at the SDK Level

Our static taint analysis identified 2318 instances of personal data access across 27 of the 41 pre-identified data types. Out of 158 SDKs, 95 (60.1%) were found to read personal data, with 34 (21.5%) having at least one tainted trace. The analysis revealed that 346 (14.9%) of these accesses were tainted, suggesting potential privacy leakage into public spaces, i.e., system settings, shared storage on device, and Internet. We present the taint analysis outcomes by data types in Table VI.

Distribution of Data Collected & Shared. As shown in Table VI, location and sensors' data (category C3) accounts for the greatest portion of collection actions, contributing 1002 out of 2318 (43.2%) caught collection actions. The remaining personal data categories ordered by their number of caught collection actions are data for wireless communication (C2, 22.8%), data for chip, cellular and peripheral (C1, 17.9%), media and software-specific data (C4, 13.4%), and personal data (C5, 2.8%), respectively.

From a personal data sharing perspective, the access of wireless communication data (C2) constitutes the largest part of the tainted flow, as 172 out of 346 (49.7%) traces are tainted. The numbers of sharing actions involving data in categories C3 and C1 come next, accounting for 37.6% and 12.7% of all sharing actions, respectively. No sharing actions involving data in categories C4 and C5 are observed. From an *identifiability* perspective, we find that most personal data involved in sharing actions are referred to as *personally identifiable information* (PII) in relevant literature [37], [38], [63], [64].

We remark that whether SDKs can collect personal data eventually depends on the permissions and privileges of their embedding apps, and tainted traces do not automatically imply exfiltration in runtime. A manual validation is performed to exclude false-positive tainted traces. We will elaborate more on them in the remainder of this section.

Privacy Leakage (Type I Issues). During our analysis, we observed that static taint analysis struggles to accurately determine the execution paths in conditional and branched code. If any tainted trace is found spanning over an unreachable, dead, or logically infeasible code, we exclude the tainted trace from the statistics of potential leakages. In Figure 3, we provide an example code reverse-engineered from one of our examined SDKs named "`mipush`". We find that it attempts to read the SSID of the connected WiFi, but it is later assessed by us as a false alarm. The SSID reading action (which can be realized by the method `getExtraInfo` (line 7) until Android 10) would only take place if the device is not connected to WiFi (after line 4); therefore, we determine that this is a logically infeasible code regarding personal data access. In this study, we manually exclude the false alarms from the 346 traces and accordingly identify 338 (97.7% precision) logically valid traces, among which each trace indicates an occurrence of privacy leakage issue.

```
1  public static String a(Context context) {
2    ConnectivityManager cm = (ConnectivityManager) context.
         getSystemService(Context.CONNECTIVITY_SERVICE);
3    NetworkInfo info = cm.getActiveNetworkInfo();
4    if (info.isConnected()) { return "wifi"; }
5    ...
6    try {
7      return info == null? "" : info.getTypeName()+"-"+info.
           getSubtypeName()+"-"+info.getExtraInfo().toLowerCase
           ();
8    }
9    catch(Exception e) { return ""; }
10 }
```

Fig. 3: A false positive example of SSID reading (line 7)

For the sharing operations, we observe that 332 out of 338 (98.2%) valid traces are tainted at network interfaces. The six remaining traces are tainted at system setting APIs. Compared to sharing personal data through the Internet, we remark that writing personal data in system settings is a more worrying issue because it exploits the privilege of pre-installed apps and is capable of sharing user privacy even without reading any personal data from the Android OS. We will detail this in Section V.

Potential Missed Leakages. Although our taint analysis pipeline achieves promising precision, we remark that the 338 detected traces may not exhaust all leakage data flows present in the examined SDKs. As indicated by our preliminary study, even state-of-the-art static analysis engines can miss taint flows, i.e., produce false negatives. The main reasons could be twofold. First, some leakage traces may be so long or complex that our adopted analyzer cannot explore them within a reasonable time budget on our experimental hardware. Second, the examined SDKs may also read and share privacy-related data through undocumented code paths, for which identifying all relevant methods is technically challenging, even with manual inspection.

Considering the recall rate obtained from our preliminary study, there may still exist hundreds of additional leakage traces that remain undiscovered. For this reason, our findings should be interpreted as a conservative lower bound on the actual extent of privacy leakage. We further discuss the implications of these potential false negatives as a limitation of our static analysis in Section VIII.

C. RQ3: SDK-Level Privacy Disclosure Compliance

We resort to cross-checking the privacy policies with the actual data collection behaviors of the collected SDKs to

TABLE VII: Assessment of privacy disclosure compliance by data types

Data type and class	Type II (EC)	Type III (OC)	Data type and class	Type II (EC)	Type III (OC)	Data type and class	Type II (EC)	Type III (OC)
C1 Carrier info	10	7	**C2** Bluetooth	4	0	**C4** Android ID	10	0
IMEI	1	6	Bluetooth MAC	2	0	App list	22	1
MEID	0	21	BSSID/SSID	8	2	Audio record	0	9
IMSI	1	18	IP Address	2	44	Google Ad ID	2	3
ICCID	4	16	WiFi	3	0	OAID	0	3
Network info	0	24	WiFi MAC	2	2	Screen record	0	1
Serial	1	6	*Subtotal*	*(21)*	*(48)*	*Subtotal*	*(34)*	*(17)*
SIM card info	1	0	**C3** Camera	0	4	**C5** Account info	1	4
SMS	3	2	Location	11	14	Clipboard data	11	0
Telephone no	1	3	Misc. sensors	7	0	Contact list	1	10
Subtotal	*(22)*	*(103)*	*Subtotal*	*(18)*	*(18)*	*Subtotal*	*(13)*	*(14)*
						Total	**108**	**200**

understand the *status quo* of privacy disclosure compliance at the SDK level. We remark that our assessment only focuses on the 109 out of 158 (69.0%) SDKs that provide privacy policies, and the 27 data types that appear in the data collection behaviors observed from the taint analysis. We present our assessment results in Table VII.

Excessive Collection (Type II Issues). Our evaluation finds that 40 out of 109 (36.7%) SDKs collected excessive personal data compared with their privacy policies (e.g., an SDK collects personal identifiers but does not declare them in its privacy policy). Among the 40 SDKs that excessively collect users' personal data, the number of undeclared data types ranges from 1 to 9. Each SDK has an average of three undeclared data types. As shown in Table VII, 22 out of the 41 pre-identified data types (81.5%) have been excessively collected by the examined SDKs. The app list, clipboard data, and location are the top three most excessively collected data types. Specifically, 22 SDKs (20.2%) collect the app list without any declaration in their privacy policies, followed by clipboard data and location, which are excessively collected by 11 (10.0%) SDKs. By jointly considering data collection behaviors (refer to Table VI), our findings show that the app list and clipboard data are the two most *stealthily collected* data types. Only 4 SDKs collecting the app list have mentioned it in their privacy policies (leading to 22 excessive collection behaviors). None of the 11 caught SDKs have declared to read the clipboard data.

Over-claiming (Type III Issues). Apart from the excessive collection, we find that the over-claiming issue in privacy policies is far more pervasive in the tested SDKs. In our study, 96 out of 109 (88.1%) SDKs are found to claim more data types in privacy policies than what they (really) need. The IP address, network information, and MEID are the top three data types that have been over-claimed in the tested SDK privacy policies. Especially, 44 out of 109 (40.4%) SDKs that provide privacy policies over-claim their access to IP addresses. Such pervasive claims, again, reflect the worrisome landscape of privacy disclosure compliance at the SDK level.

Lessons Learned. Our study reveals two key findings. First, some SDK developers use vague or broad terms like "identifiers" or "device information" instead of specifying the exact data being collected, leading to differing interpretations. For instance, while most developers use "identifiers" to indicate device identifiers, some refer to advertisement identifiers, resulting in overclaims for device identifiers and underclaims

TABLE VIII: Changes in the tainted traces after 12 months (only tainted data types are displayed)

Data type & class	Changes in tainted traces	Data type & class	Changes in tainted traces
C1 Carrier info	▲ +31 (new)†	**C2** *Subtotal*	▼ *-106 (-61%)*
IMEI	▼-17 (65%)	**C3** Camera	▲ +2 (new)
MEID	▼-2 (50%)	Location	▼-94 (72%)
IMSI	▼-2 (40%)	Misc sensors	▲ +20 (new)
ICCID	• +0 (0%)	*Subtotal*	▼ *-72 (55%)*
Serial	▲ +3 (60%)	**C4** Android ID	▲ +34 (new)
SIM card info	▲ +1 (new)	App list	▲ +53 (new)
SMS	▲ +4 (new)	Audio record	▲ +5 (new)
Telephone no	▲ +2 (new)	Google Ad ID	▲ +9 (new)
Subtotal	▲ *+20 (45%)*	OAID	▲ +3 (new)
C2 Bluetooth	▲ +5 (new)	Screen record	▲ +1 (new)
Bluetooth MAC	▲ +3 (new)	*Subtotal*	▲ *+105 (new)*
BSSID/SSID	▼-73 (68%)	**C5** Clipboard data	▲ +17 (new)
IP Address	▲ +9 (900%)	Contact list	▲ +1 (new)
WiFi	▲ +9 (new)	*Subtotal*	▲ *+18 (new)*
WiFi MAC	▼-59 (92%)	**Total**	▼-35 (10%)

† We use "new" to indicate data types observed re-inspection for the first time.

for Android ID or Google Ad ID. Second, although many SDKs declare access to both GPS and IP addresses for location data, our taint analysis shows that most SDKs only use location APIs, leading to over-claiming cases of IP addresses. This lack of clarity can confuse users about what data is actually collected, reducing transparency in privacy policies.

D. Privacy Disclosure Compliance Re-Inspection

On the basis of our large-scale analysis of 158 SDKs collected in October 2022, we re-visited the two data sources and retrieved the latest release versions of the same SDKs as of October 2023, on which we conducted a cross-version analysis. This re-inspection serves two main purposes. First, it allows us to examine whether these SDKs have modified their privacy-relevant behaviors following our attempts to report the findings to the corresponding developers (see Section VI for our ethical disclosure). Second, it enables us to observe potential trends in personal data collection practices as newer Android OS versions have been gradually rolled out. Table VIII summarizes the results of our re-inspection.

Trends of Privacy Collection Behaviors. We find 311 traces tainted from the latest version of 158 SDKs. Compared to our initial assessment, we observed 35 (10.1%) fewer traces that attempt to collect and share user personal data.

The difference between the outcomes of the two assessments mainly falls in the sources of tainted traces. As shown in Table VIII, we observe significant growth in the sharing of data relevant to media and installed software (C4), accounting for over 1/3 of the total caught traces. Among the 105 tainted traces, reading and sharing the list of installed apps ranks the top among all sources, with 53 traces caught. Although the access to data in that category has been pervasively observed in our initial assessment, our re-inspection did not catch any traces of *sharing* such data.

Additionally, we find that the collection behaviors for location and SSID/BSSID have significantly declined, marking the two greatest decreases. There is also an overall reduction in the collection of UUIs, including data types in categories C1

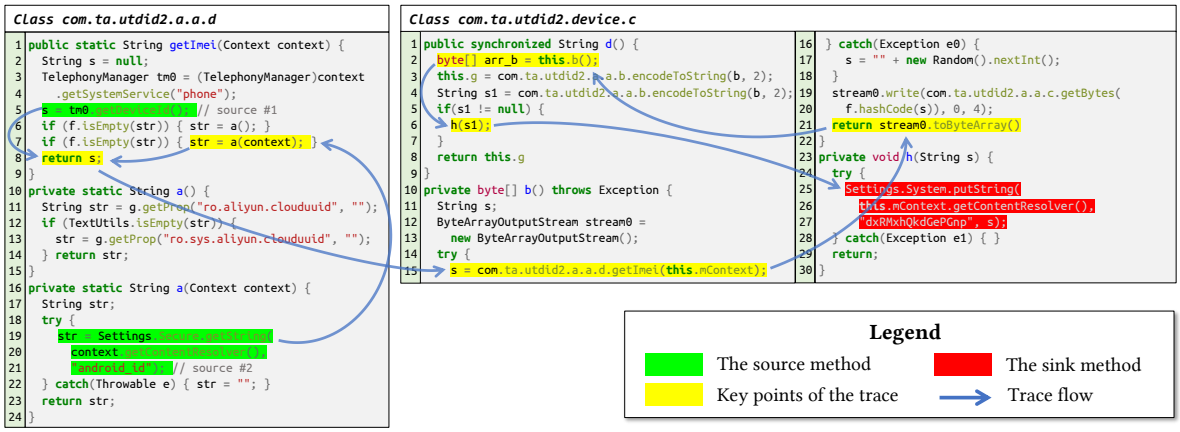


Fig. 4: Information flow demonstrates how the device ID is accessed and saved to the system settings by one of our analyzed SDKs (the obfuscated code is obtained from reverse engineering and has been simplified to save space)

TABLE IX: System setting entries found that contains a UUI

SDK	Entry name	Description of the value
Alicloud Push	`dxCRMxhQkdGePGnp`	IMEI/MEID, serial, or Android ID†, AES encrypted, base64 encoded
	`mqBRboGZkQPcAkyk`	IMEI/MEID, serial, or Android ID†, AES encrypted, base64 encoded
Baidu LBS	`com.baidu.deviceid.v2`	Package name and Android ID, MD5 hashed‡
	`com.baidu.uuid`	Package name and serial or a random UUID†, MD5 hashed‡

† depending on the permissions granted to the embedded app.
‡ inferred description based on the reading logic in the implementation.

and C2. Upon reviewing the historical changes in permission requirements, we note that most data types showing decreased collection behaviors had their access rules significantly updated in Android 10 [32], which imposed strict restrictions on third-party apps' access to personally identifiable information. In contrast, we find the growth of tainted traces for the app list. Although access to the app list has also been restricted in a future release, these results still seriously concern us since the latest version of SDKs implies their persistent attempt to excessively and stealthily collect user personal data.

V. Case Study: System Settings Injection in Pre-Installed Apps

Building on the tainted traces identified in RQ2, we conduct a focused case study on writing privacy to Android system settings, which is an uncommon but particularly concerning sink. During our analysis, we observed several tainted traces from an SDK named "Alicloud push" that terminate at writing to system settings. Specifically, two entries of system settings are involved, as shown in Table IX. Writing into the system settings requires privileged permission, so that ordinary third-party apps are unable to do so. However, we find that SDKs are being embedded by pre-installed apps can write arbitrary values into the system settings even without the user's awareness. We refer to this finding as a *system settings injection issue*, because it can be manipulated by an SDK to create and share user-unresettable identifiers (UUIs).

Although leaking personal data through covert channels has been studied in existing literature [65], the system settings injection issues still seriously concern us as the caught SDK subtly evades accessing highly sensitive UUIs (e.g., IMEI, as discussed in [65]) but attempts to generate UUIs on its own. The caught SDK then takes advantage of its popularity and abuses the "`preinstalled`" privilege by counting on itself to be embedded in pre-installed apps or even privileged system apps. By exploiting this issue, the system settings become a *covert channel* that enables privacy-sensitive data sharing to apps without privileged permission.

Figure 4 illustrates an example of system setting injection from our findings. The source code is obtained through reverse engineering. In order to obtain a UUI, it attempts to invoke a privileged API (line 5, left), query the existence of pre-written system properties (lines 6 and 10-15, left), and read the Android ID (lines 7 and 16-24, left) sequentially, as shown in the figure. Due to the first approach requests the embedded app to be a system app with "`READ_PRIVILEGED_PHONE_STATE`" permission, and the second approach is only viable when there exists a system app on the same device to write the two system properties in advance, we consider both them *infeasible* by default on nowadays Android devices. However, the third approach, i.e., reading Android ID through a system setting table named `SECURE`, does not request special permission. As a result, the SDK can at least obtain the Android ID to uniquely identify the device. It then performs a hash function over the obtained Android ID (lines 19-20, right) and writes the hash value into the system settings with a customized key (lines 25-27, right). This process, although it complies with the Android permission mechanism, transforms the value of *resettable* Android ID into a *de facto* UUI that can be accessed by any app that knows the customized key in the system settings.

Considering the exploitability of the pre-installed privileges, we carry out another round of investigation of all 158 SDKs to find if there exists any other potential privacy infringement by taking advantage of the system settings mechanism. As a result, we managed to find another two entries that have

been read in an SDK named "Baidu LBS" (see Table IX). As their names imply, these two entries are obviously written by third-party privileged apps. We suspect the formation of the two entries should follow a similar routine, and accordingly reverse engineer the involved SDK and present our inferred descriptions of the two entries in Table IX. Unfortunately, we did not find any app to write these entries into the system settings. The identity of those who exploited this system settings issue remains unknown to us.

Impact. We have noticed that the two studied SDKs are widely adopted in third-party apps, particularly in the Chinese market. Therefore, we conduct a small-scale study to explore the impact of the system settings injection issues. We collect devices from 9 OEM manufacturers (8 OEM devices purchased in the Chinese market[8] and 1 Google Pixel device purchased in the US market), download the packages of pre-install apps, and investigate whether the two aforementioned SDKs have been once embedded in the pre-installed apps. We observe a worrisome fact that, except for the Google Pixel devices, all the tested devices have at least one pre-installed app that installs the investigated SDKs. According to the latest market report [66], these eight OEM manufacturers reserve over 66% of the global Android smartphone market share. They also shipped over 94% new Android devices in China in 2023 [67]. This implies that almost all Android smartphones sold in the Chinese market may already pose a privacy leakage risk, based on the assumption that the choice of pre-installed apps remains the same across different models produced by a manufacturer.

VI. Ethics Considerations

We have reported all our findings to the relevant parties, and we have also kept them confidential for at least 90 days. For each SDK that found a compliance issue, we manually searched the contact left by its developers and emailed our findings since our first round of SDK investigation. Although we observed slight mitigation following re-inspection, we did not receive positive responses regarding non-compliance concerns from the developers we attempted to contact. We also expressed our concern to Google about the system settings injection issues (to be detailed in Section V) and suggested removing the pre-installed privileges for writing system settings in future releases. Google replied and suggested that it is at the device manufacturer's discretion to select and grant pre-installed privileges to third-party apps.

VII. Discussion

A. Causes of Privacy Mishandling

Privilege Free-riding. An SDK's access to personal data depends on the permissions granted to the apps it's embedded in. However, third-party apps pre-installed by OEMs can obtain privileged permissions. A malicious SDK in such an app can exploit system settings to deploy privilege free-riding attacks, creating and sharing UUIs without users' awareness. As reading system settings require no permissions, privilege free-riding significantly undermines Google's efforts in restricting third-party app access to UUIs since Android 10.

Insufficient Regulations. It is well known that the release of Android apps is subject to strict regulations, with non-compliance in privacy preservation potentially leading to removal from app stores or fines. However, SDK releases lack similar fine-grained regulations and sufficient legislation to ensure compliance with personal data protection laws. Ironically, even some Google in-house SDKs do not have privacy policies listed on the SDK Index[9]. Currently, both repositories function merely as search engines for available SDKs, without a platform-level compliance checking mechanism to ensure complete permission disclosure.

B. Our Recommendations

Precautions of Privilege Abuse. We argue that the pre-installed privileges open an attack surface for a malicious SDK, specifically when the SDK uses app-differential Android ID to identify a device user, transforms it into a de facto UUI, and provides free access with all apps installed on the victim device. Based on this finding, we advocate that Google restrict pre-installed apps from writing data into the system settings in the future. We also advise OEM manufacturers to minimize the number of pre-installed apps on their devices.

Strengthened Platform Regulation. Currently, both release platforms selected in this study are merely index websites, which do not host the SDK binaries and lack substantial enforcement. Developers voluntarily register SDKs on these platforms. For this reason, we call for strengthened regulation over the usage of Android SDKs. We also advocate that the platforms perform static analysis over the SDK binaries to exhaustively list requested permissions, especially for those at the dangerous level and above. This measure could address the black-box concern for app developers.

Transparent App Development. We advocate for the transparent management of permissions requested by the embedding SDKs. Thus, app developers can have their own discretion in granting permissions without compromising functionality. We are delighted to see Google's efforts in improving the SDK's governance. A new mechanism called *SDK Runtime* has been introduced in Android 13 that provides a dedicated execution environment for SDKs [68]. However, this mechanism is not compulsory and still faces many usability and functionality limitations at the moment this work is being performed. Besides that, how to ensure the cooperation of SDK developers remains a challenging issue, given that many SDKs are released and maintained on non-Google platforms.

VIII. Threats to Validity

A. Internal Limitations

Static-Only Taint Analysis. Our approach relies solely on static taint analysis. While this design is sufficient for uncovering large numbers of potential privacy leakage issues at scale, it cannot automatically prune logically infeasible taint

[8]These devices are manufactured by Honor, Huawei, Lenovo, OnePlus, Oppo, Samsung, Vivo, and Xiaomi.

[9]An example can be found at https://play.google.com/sdks/details/com-google-android-gms-play-services-auth

traces. As a result, some manual validation is still required. Dynamic testing could help filter out such infeasible traces and further improve the precision and efficiency of our analysis. However, designing and integrating such a dynamic technique is beyond the scope of this work, and we elaborate on the relevant practical challenges as an external limitation.

Choice of Taint Analysis Engine. Our study adopts AppShark as the taint-analysis engine based on the outcomes of our preliminary comparison, which evaluates multiple off-the-shelf tools. We therefore treat AppShark as an optimal and practical choice for enabling large-scale analysis, even though advancing the state of the art in taint analysis itself is not our primary goal. We also note that our approach is not tied to AppShark, and alternative engines could be applied by adjusting the analysis configuration. The optimal choice of analyzer may vary across different SDKs and evaluation settings, and may also evolve over time as existing tools are updated and new analysis frameworks are developed.

Coverage of Our SDK collection. Because there is no precise definition of what constitutes an Android third-party SDK and no global registry of such components, it is inherently difficult to enumerate all existing SDKs in the Android ecosystem. As a result, our analysis covers 158 SDKs collected from major repositories, which we believe include many widely used and commercially influential SDKs. The true size of the Android third-party SDK ecosystem remains unknown; therefore, our findings should be interpreted as characterizing a large and representative subset of prevalent SDKs, rather than the complete scope.

Coverage of Sources and Sinks. Our current configuration covers 741 API source methods and 46 sinks. Although we continuously refined this configuration, it may still miss certain data-exfiltration channels as new APIs can be introduced along with OS evolution and emerging hardware. Our results should therefore be interpreted as a conservative lower bound on potential leakage via the APIs we model, rather than an exhaustive enumeration of all possible channels.

Reliance on a Language Model. For policy–code consistency checking, we employ one specific language model-based NLI configuration, including a pre-trained model, prompt templates, and a decision threshold. Although the adopted NLI model offers more stability and transparency in determining entailment than the state-of-the-art conversational LLMs (e.g., Llama and ChatGPT families), our analysis still inherits the intrinsic limitations of the underlying model, including potential knowledge gaps, hallucination, unstable reasoning across paraphrased inputs, and amplification of biases present in its training data. These phenomena can lead to both false positives and false negatives in our classification of disclosures. A systematic comparison of alternative LLMs, prompting strategies, and calibration techniques is left as future work.

B. External Limitations

Practical Constraints on Dynamic Testing. Although dynamic testing can automatically identify and exclude false alarms (e.g., logically invalid taint traces, as shown in Figure 3), performing large-scale dynamic SDK testing is challenging in practice. Many SDKs suffer from inadequate documentation, undisclosed interfaces, and highly event-driven or credential-dependent logic. For instance, some finance and customer-service SDKs require valid licenses, tokens, or server-side integration to exercise their full functionality. Meeting all such functional prerequisites for hundreds of SDKs would require substantial engineering effort and access privileges that are not generally available to researchers. For that reason, our approach relies solely on static taint analysis and regards dynamic testing as a future research direction.

Elusive and Ambiguous Disclosure. Although SDK vendors should clearly disclose the collection of personal data, we observe that many SDK privacy policies employ vague or ambiguous language. Vague statements, overloaded terms, and missing contextual information can leave actual data-collection practice under-specified, which may in turn yield false negatives in our NLI-based analysis. We also find that some vendors maintain a single, shared privacy policy for multiple SDK products with different functionality and across different platforms, further blurring which disclosure apply to which SDK and under what conditions. Addressing these elusive and ambiguous disclosures requires coordinated efforts from SDK vendors and regulators and is beyond the scope of our analysis framework.

IX. Related Works

Privacy Disclosure Compliance Evaluation. Numerous works focus on detecting inconsistencies between the privacy practices of an application and its descriptions or its privacy policies [69]–[76]. This line of research begins with automatic risk assessment of Android apps, such as WHYPER [72] and AutoCog [73], which leverage NLP techniques to evaluate the necessity of required permissions based on apps' descriptions. Gorla et al. [70] study mismatches between app descriptions and app behaviors by clustering Android apps according to their description topics and identifying outliers in each cluster with respect to API usage. More recent work extends the scope of privacy disclosure compliance evaluation to third-party data harvesting, user consent, service misconfigurations, and the quality of privacy policies at the app level. XFinder [39] uses dynamic analysis and NLP techniques to parse terms-of-service and extract data-sharing policies. Nguyen et al. [77] investigate user-consent verification through traffic analysis and ablation experiments. Zhang et al. [78] highlight the issue of analytical service misconfigurations, while Yu et al. [18] scrutinize the apps' privacy policies with regard to the inclusion of third-party libraries (TPLs). Andow et al. [19] propose POLICHECK, which identifies ambiguous or omitted disclosures of third parties. Pan et al. [79] study the automated generation of privacy policies for mobile apps and show that defective compliance is widespread among automatically generated privacy policies.

A recent study by Zhao et al. [20] analyzes the privacy disclosure compliance of Android TPLs through control-flow graph extraction and privacy disclosure analysis. Our work differs from theirs in two key aspects. First, we focus on third-party SDKs, whose implementations and business logic

are more complex than those of generic libraries. In this setting, the strategy of [20] that constructs full control-flow graphs for all public methods via Soot becomes technically challenging and difficult to scale. Second, whereas prior work primarily resorts to predefined keyword lists and language patterns to analyze privacy policies, we employ a pre-trained, ready-to-use transformer-based NLI model, which reduces our dependence on hand-crafted keywords and patterns and is more amenable to supporting additional languages. Another parallel line of work by Xiao et al. [21] investigates the compliance of Apple TPLs by comparing privacy disclosures with dynamic analysis results. In contrast to Apple's ecosystem, which provides a predefined "nutrition-label"-style taxonomy of privacy disclosures, the Android platform lacks a standardized scope of privacy-related data types, leading SDK providers to adopt highly heterogeneous styles when declaring their privacy practices. To address this challenge, we draw on multiple sources to construct the scope of personal data in our compliance assessment.

To the best of our knowledge, we are the first to assess the privacy disclosure compliance of Android third-party SDKs at scale. The scope of personal data we adopt is the most comprehensive among existing works. Our evaluations present new findings that have not been covered before. We also discovered the elevated risk of privacy leakage in the context of pre-installed apps caused by their embedding of SDKs.

Privacy Leakage Assessment. There is a substantial body of literature addressing various aspects of privacy concerns in the Android ecosystem [40], [80], [81]. Meng et al. [62] study user-unresettable identifier safeguards on a wide range of Android devices. He et al. [82] utilize dynamic analysis to explore the leakage of permission-related data from third-party libraries in Android apps. Ekambaranathan et al. [83] investigate data usage and disclosure in children's apps. Liu et al. [84] examine data leakage from nine analytic libraries across 300 apps, using both static and dynamic analyses. Razaghpanah et al. [85] detect third-party advertising and tracking services via dynamic analysis of network traffic data.

X. Conclusion

In this study, we investigate the intricate privacy challenges associated with the widespread use of third-party SDKs in Android apps. Our large-scale analysis involved an evaluation of 158 third-party SDKs, prevalent across Android applications, with an approach intertwining rigorous taint analysis for data collection practices and NLI model-aided interpretation of privacy policies. The findings were startling, with 338 potential privacy leakages detected from 158 SDKs. Besides that, our study reveals that less than 70% of examined SDKs provide privacy policies, among which approximately 37% of SDKs are found indulging in the over-collection of personal data, signaling a clear violation of privacy norms. Our findings suggest the need for stricter regulations, improved development guidelines, and more transparent permission management to reduce privacy risks in Android apps.

References

[1] P. Ruiz, "Authenticating on Android with the AppAuth Library," 2022, (accessed 9 July 2024). [Online]. Available: https://medium.com/androiddevelopers/authenticating-on-android-with-the-appauth-library-7bea226555d5

[2] Android Documentation, "Work with data more securely," 2023, (accessed 9 July 2024). [Online]. Available: https://developer.android.com/topic/security/data

[3] Segment, "Analytics for Android," 2023, (accessed 9 July 2024). [Online]. Available: https://segment.com/docs/connections/sources/catalog/libraries/mobile/android

[4] Android Documentation, "Add Ads to Your Instant App," 2023, (accessed 9 July 2024). [Online]. Available: https://developer.android.com/topic/google-play-instant/guides/advertising

[5] ——, "Develop UI for Android," 2023, (accessed 9 July 2024). [Online]. Available: https://developer.android.com/develop/ui

[6] P. Salza, F. Palomba, D. Di Nucci, C. D'Uva, A. De Lucia, and F. Ferrucci, "Do Developers Update Third-Party Libraries in Mobile Apps?" in *ICPC*, 2018, p. 255–265.

[7] Y. Zhang, J. Dai, X. Zhang, S. Huang, Z. Yang, M. Yang, and H. Chen, "Detecting third-party libraries in android applications with high precision and recall," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 141–152.

[8] blackkite, "Data Breaches Caused By Third-Parties," 2022, (accessed 9 July 2024). [Online]. Available: https://blackkite.com/data-breaches-caused-by-third-parties

[9] P. Mahajan, "3rd Party Libraries: Your Next Data Breach Nightmare," 2022, (accessed 9 July 2024). [Online]. Available: https://www.privado.ai/post/3rd-party-libraries-data-breach

[10] Droid, "How to fix my Android - What Is Mobile Services Manager?" 2023, (accessed 9 July 2024). [Online]. Available: https://www.howtofixmyandroid.com/what-is-mobile-services-manager/

[11] M. Mollah, "What Is Mobile Services Manager? Is It A Threat? How To Fix It?" 2022, (accessed 9 July 2024). [Online]. Available: https://www.socialmediamagazine.org/mobile-services-manager/

[12] J. Keegan and A. Ng, "Over 100 apps that sold location data to sketchy data broker revealed," 2022, (accessed 9 July 2024). [Online]. Available: https://mashable.com/article/app-location-data-sold

[13] The European Parliament, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation) (text with eea relevance)," *Official Journal of the European Union*, 2016.

[14] State of California Department of Justice, "California consumer privacy act (ccpa)," 2023, (accessed 9 July 2024). [Online]. Available: https://oag.ca.gov/privacy/ccpa

[15] Google, "Google play sdk index," 2024, (accessed 9 July 2024). [Online]. Available: https://play.google.com/sdks

[16] China Academy of Information and Communication Technology, "Nationwide sdk management and service platform (translated from chinese)," 2024, (accessed 9 July 2024). [Online]. Available: https://sdk.caict.ac.cn/official

[17] D. Rodriguez, J. M. Del Alamo, C. Fernández-Aller, and N. Sadeh, "Sharing is not always caring: Delving into personal data transfer compliance in android apps," *IEEE Access*, 2024.

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

[19] B. Andow, S. Y. Mahmud, J. Whitaker, W. Enck, B. Reaves, K. Singh, and S. Egelman, "Actions speak louder than words: Entity-Sensitive privacy policy and data flow analysis with PoliCheck," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 985–1002.

[20] K. Zhao, X. Zhan, L. Yu, S. Zhou, H. Zhou, X. Luo, H. Wang, and Y. Liu, "Demystifying privacy policy of third-party libraries in mobile apps," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1583–1595.

[21] Y. Xiao, C. Zhang, Y. Qin, F. F. S. Alharbi, L. Xing, and X. Liao, "Measuring compliance implications of third-party libraries' privacy label disclosure guidelines," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 1641–1655.

[22] K. Rafalski, "Top 22 android libraries for 2025 every developer needs," 2025, (accessed 25 November 2025). [Online]. Available: https://www.netguru.com/blog/android-libraries

[23] Y. Becher, "Build better, safer sdks with google play sdk console," 2024, (accessed 25 November 2025). [Online]. Available: https://android-developers.googleblog.com/2024/05/build-better-safer-sdks-google-play-sdk-console.html

[24] S. Jaiswal, "What is an sdk? sdk vs library vs framework," 2025, (accessed 25 November 2025). [Online]. Available: https://www.geeksforgeeks.org/software-engineering/what-is-an-sdk-sdk-vs-library-vs-framework/

[25] Google, "Sdk eligibility for sdk console," 2022, (accessed 25 November 2025). [Online]. Available: https://support.google.com/googleplay/android-developer/answer/12244916#SDK_eligibility

[26] O. Fich, "Does gdpr apply to mobile apps?" 2022, (accessed 25 November 2025). [Online]. Available: https://cookieinformation.com/resources/blog/does-gdpr-apply-to-mobile-apps/

[27] Google, "Sdk platform tools release notes," 2025, (accessed 25 November 2025). [Online]. Available: https://developer.android.com/tools/releases/platform-tools

[28] P. Oburoh-Kuju, "Understanding android sdk: A comprehensive guide," 2023, (accessed 25 November 2025). [Online]. Available: https://www.amorserv.com/insights/understanding-android-sdk-a-comprehensive-guide/

[29] Google, "Understand app privacy & security practices with google play's data safety section," 2024, (accessed 9 July 2024). [Online]. Available: https://support.google.com/googleplay/answer/11416267?sjid=6208336964583751082-AP

[30] B. Wolford, "A guide to gdpr data privacy requirements," 2024, (accessed 9 July 2024). [Online]. Available: https://gdpr.eu/data-privacy/

[31] Google, "Manifest.permission," 2024, (accessed 28 June 2024). [Online]. Available: https://developer.android.com/reference/android/Manifest.permission

[32] ——, "Privacy changes in Android 10," 2023, (accessed 9 July 2024). [Online]. Available: https://developer.android.com/about/versions/10/privacy/changes

[33] ——, "Privacy in Android 11," 2023, (accessed 9 July 2024). [Online]. Available: https://developer.android.com/about/versions/11/privacy

[34] T. Chen, I. Ullah, M. A. Kaafar, and R. Boreli, "Information leakage through mobile analytics services," in *HotMobile*, 2014, pp. 1–6.

[35] C. Leung, J. Ren, D. Choffnes, and C. Wilson, "Should you use the app for that? comparing the privacy implications of app-and web-based online services," in *IMC*, 2016, pp. 365–372.

[36] E. P. Papadopoulos, M. Diamantaris, P. Papadopoulos, T. Petsas, S. Ioannidis, and E. P. Markatos, "The long-standing privacy debate: Mobile websites vs mobile apps," in *WWW*, 2017, pp. 153–162.

[37] J. Ren, M. Lindorfer, D. J. Dubois, A. Rao, D. Choffnes, and N. Vallina-Rodriguez, "A longitudinal study of pii leaks across android app versions," in *NDSS*, 2018.

[38] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes, "Recon: Revealing and controlling pii leaks in mobile network traffic," in *MobiSys*, 2016, pp. 361–374.

[39] J. Wang, Y. Xiao, X. Wang, Y. Nan, L. Xing, X. Liao, J. Dong, N. Serrano, H. Lu, X. Wang *et al.*, "Understanding malicious cross-library data harvesting on android," in *USENIX Security*, 2021.

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

[41] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale," in *TrustCom*. Springer, 2012, pp. 291–307.

[42] L. Qiu, Z. Zhang, Z. Shen, and G. Sun, "Apptrace: Dynamic trace on android devices," in *ICC*, 2015, pp. 7145–7150.

[43] M. Sun, T. Wei, and J. C. Lui, "Taintart: A practical multi-level information-flow tracking system for android runtime," in *CCS*, 2016, pp. 331–342.

[44] Appsflyer, "Open anonymous device identifier (oaid)," 2025, (accessed 20 November 2025). [Online]. Available: https://dev.appsflyer.com/hc/docs/oaid

[45] D. Bui, Y. Yao, K. G. Shin, J.-M. Choi, and J. Shin, "Consistency analysis of data-usage purposes in mobile apps," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2824–2843.

[46] Google, "Understand & manage your location when you search on google," 2025, (accessed 20 Feb 2026). [Online]. Available: https://support.google.com/websearch/answer/179386

[47] B. Muthukadan, "Selenium with Python," 2018, (accessed 9 July 2024). [Online]. Available: https://selenium-python.readthedocs.io/

[48] B. Andow, S. Y. Mahmud, W. Wang, J. Whitaker, W. Enck, B. Reaves, K. Singh, and T. Xie, "PolicyLint: investigating internal privacy policy contradictions on google play," in *28th USENIX security symposium (USENIX security 19)*, 2019, pp. 585–602.

[49] F. Xie, Y. Zhang, C. Yan, S. Li, L. Bu, K. Chen, Z. Huang, and G. Bai, "Scrutinizing privacy policy compliance of virtual personal assistant apps," in *ASE*, 2022.

[50] L. Yu, X. Luo, X. Liu, and T. Zhang, "Can we trust the privacy policies of android apps?" in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 538–549.

[51] S. Zimmeck, Z. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, S. Wilson, N. Sadeh, S. Bellovin, and J. Reidenberg, "Automated analysis of privacy requirements for mobile apps," in *NDSS*, 2017.

[52] Y. Nie, A. Williams, E. Dinan, M. Bansal, J. Weston, and D. Kiela, "Adversarial NLI: A new benchmark for natural language understanding," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020.

[53] H. Harkous, S. T. Peddinti, R. Khandelwal, A. Srivastava, and N. Taft, "Hark: A deep learning system for navigating privacy feedback at scale," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2469–2486.

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

[55] L. Qiu, Y. Wang, and J. Rubin, "Analyzing the analyzers: Flowdroid/iccta, amandroid, and droidsafe," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 176–186.

[56] Facebook Open Source, "Mariana Trench: Security-Focused Static Analysis for Android and Java Applications," 2024, (accessed 9 July 2024). [Online]. Available: https://mariana-tren.ch/

[57] Bytedance, "Appshark," 2022, (accessed 9 July 2024). [Online]. Available: https://github.com/bytedance/appshark

[58] GitHub, "For large size apk/code it is consuming heavy battery/cpu/memory," 2021, (accessed 30 Nov 2025). [Online]. Available: https://github.com/facebook/mariana-trench/issues/51

[59] S. Rasthofer, S. Arzt, and E. Bodden, "A machine-learning approach for classifying and categorizing android sources and sinks." in *NDSS*, vol. 14, 2014, p. 1125.

[60] Google, "Log Info Disclosure," 2023, (accessed 9 July 2024). [Online]. Available: https://developer.android.com/topic/security/risks/log-info-disclosure

[61] ——, "Application Sandbox," 2022, (accessed 9 July 2024). [Online]. Available: https://source.android.com/docs/security/app-sandbox

[62] M. H. Meng, Q. Zhang, G. Xia, Y. Zheng, Y. Zhang, G. Bai, Z. Liu, S. G. Teo, and J. S. Dong, "Post-gdpr threat hunting on android phones: dissecting os-level safeguards of user-unresettable identifiers," in *NDSS*, 2023.

[63] Y. Shen, P.-A. Vervier, and G. Stringhini, "Understanding worldwide private information collection on android," in *NDSS*, 2021.

[64] Z. Wang, Z. Li, M. Xue, and G. Tyson, "Exploring the eastern frontier: A first look at mobile app tracking in china," in *Passive and Active Measurement: 21st International Conference, PAM 2020, Eugene, Oregon, USA, March 30–31, 2020, Proceedings 21*. Springer, 2020, pp. 314–328.

[65] J. Reardon, Á. Feal, P. Wijesekera, A. E. B. On, N. Vallina-Rodriguez, and S. Egelman, "50 ways to leak your data: An exploration of apps' circumvention of the android permissions system," in *USENIX Security*, 2019.

[66] Statcounter, "Mobile Vendor Market Share Worldwide," 2024, (accessed 9 July 2024). [Online]. Available: https://gs.statcounter.com/vendor-market-share/mobile

[67] ——, "Mobile Vendor Market Share China," 2024, (accessed 9 July 2024). [Online]. Available: https://gs.statcounter.com/vendor-market-share/mobile/china

[68] Google, "SDK Runtime," 2024, (accessed 9 July 2024). [Online]. Available: https://developer.android.com/design-for-safety/privacy-sandbox/sdk-runtime

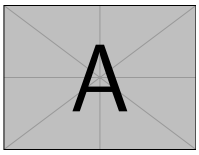
[69] D. Bui, B. Tang, and K. G. Shin, "Detection of inconsistencies in privacy practices of browser extensions," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2780–2798.

[70] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *ICSE*, 2014.
[71] X. Liu, Y. Leng, W. Yang, W. Wang, C. Zhai, and T. Xie, "A large-scale empirical study on android runtime-permission rationale messages," in *VL/HCC*, 2018.
[72] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: Towards automating risk assessment of mobile applications," in *USENIX Security*, 2013, pp. 527–542.
[73] Z. Qu, V. Rastogi, X. Zhang, Y. C. Chen, T. Z. Zhu, and Z. Chen, "Autocog: Measuring the description-to-permission fidelity in android applications," in *CCS*, 2014.
[74] R. Slavin, X. Wang, M. B. Hosseini, J. Hester, R. Krishnan, J. Bhatia, T. D. Breaux, and J. Niu, "Toward a framework for detecting privacy policy violations in android application code," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 25–36.
[75] X. Wang, X. Qin, M. B. Hosseini, R. Slavin, T. D. Breaux, and J. Niu, "Guileak: Tracing privacy policy claims on user input data for android applications," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 37–47.
[76] L. Zhou, C. Wei, T. Zhu, G. Chen, X. Zhang, S. Du, H. Cap, and H. Zhu, "POLICYCOMP: Counterpart comparison of privacy policies uncovers overbroad personal data collection practices," in *USENIX Security*, 2022.
[77] T. T. Nguyen, M. Backes, N. Marnau, and B. Stock, "Share first, ask later (or never?) studying violations of {GDPR's} explicit consent in android apps," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3667–3684.
[78] X. Zhang, X. Wang, R. Slavin, T. Breaux, and J. Niu, "How does misconfiguration of analytic services compromise mobile privacy?" in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1572–1583.
[79] S. Pan, D. Zhang, M. Staples, Z. Xing, J. Chen, X. Xu, and T. Hoang, "Is it a trap? a large-scale empirical study and comprehensive assessment of online automated privacy policy generators for mobile apps," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5681–5698.
[80] K. Zhang and X. Wang, "Peeping tom in the neighborhood: Keystroke eavesdropping on multi-user systems," in *USENIX Security*, 2009.
[81] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt, "Identity, location, disease and more: Inferring your secrets from android public resources," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 1017–1028.
[82] Y. He, B. Hu, and Z. Han, "Dynamic privacy leakage analysis of android third-party libraries," in *2018 1st International Conference on Data Intelligence and Security (ICDIS)*, 2018, pp. 275–280.
[83] A. Ekambaranathan, J. Zhao, and M. Van Kleek, ""money makes the world go around": Identifying barriers to better privacy in children's apps from developers' perspectives," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–15.
[84] X. Liu, J. Liu, S. Zhu, W. Wang, and X. Zhang, "Privacy risk analysis and mitigation of analytics libraries in the android ecosystem," *IEEE Transactions on Mobile Computing*, vol. 19, no. 5, pp. 1184–1199, 2019.
[85] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich, P. Gill *et al.*, "Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem," in *The 25th Annual Network and Distributed System Security Symposium (NDSS 2018)*, 2018.

Mark Huasong Meng is an Assistant Professor at the School of Computer Science, University College Dublin (UCD), Ireland. He received his PhD and Master's degree from National University of Singapore (NUS), and Bachelor of Engineering (Hons.) degree from Nanyang Technological University (NTU) in Singapore. His research interests include trustworthy AI, software engineering, and mobile security.

Chuan Yan is a PhD candidate at the University of Queensland, Australia. His research focuses on detecting issues in the system and third-party application ecosystem based on documentation. His works have been published in leading conferences and journals, including WWW, FSE, ICSE, ASE, and PETs. He is the recipient of ACM SIGSOFT Distinguished Paper Award at ASE 2024.

Qing Zhang is a senior security and privacy expert. Previously, he was an invited visiting scholar at the National University of Singapore. His expertise encompasses Android security, IoT security, and payment security, with a focus on privacy security, reverse engineering, and fuzzing techniques. He holds prestigious certifications such as the International Privacy Technology Certification (CIPT) and the Certified Information Privacy Professional/Europe (CIPP/E). His influential work has been featured at prominent security conferences, including Syscan (2016, 2018), Black Hat (2017, 2021, 2023), and HITB (2017-2022). Between 2016 and 2023, he consistently demonstrated excellence in vulnerability detection, securing numerous first-place awards from major corporations like Samsung, Huawei, Meizu, Chuizi, OPPO, Honor, and OnePlus. His academic contributions are equally noteworthy, with papers presented at top-tier security and networking conferences such as NDSS and MobiCom.



Zeyu Wang

Kailong Wang is an Associate Professor at Huazhong University of Science and Technology (HUST), China. He received his PhD degree from National University of Singapore (NUS) and bachelor's degree with the first class honours from Nanyang Technological University (NTU). He is broadly interested in trustworthy AI, secure and private software engineering. His current research focuses on the safety and ethics assessment as well as interpretability and explainability of Large Language Models.

Sin G. Teo is a Research Scientist at the Institute for Infocomm Research (I2R), A*STAR, Singapore. He earned his PhD from Monash University, Australia. His research sits at the intersection of security, intelligence, and scalable AI systems, with a focus on applied cryptography, data privacy, and software vulnerability analysis. He advances cutting-edge work in agentic AI security, malware detection, and network anomaly classification, leveraging deep learning, federated learning, and adversarial machine learning to build robust, privacy-preserving, and resilient intelligent systems.

Guangdong Bai is an Associate Professor at the City University of Hong Kong. He obtained his PhD degree from National University of Singapore, Singapore, and his M.S. and B.S. degrees from Peking University, China. His research interests include Security and Privacy, Trustworthy AI, Software Engineering and Formal Methods. He serves as an Associate Editor of IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Service Computing, and Neural Networks.

Jin Song Dong is a Professor with the School of Computing, National University of Singapore. He received his PhD degree from the University of Queensland, Australia. His research interests include software engineering, program analysis, formal verification, and model checking. He has been on the editorial board of ACM Transaction on Software Engineering and Methodology, Formal Aspects of Computing, and Innovations in Systems and Software Engineering, A NASA Journal.