Lalaine: Measuring and Characterizing Non-Compliance of Apple Privacy Labels

Yue Xiao[1], *Zhengyi Li*[1], *Yue Qin*[1], *Xiaolong Bai*[2], *Jiale Guan*[1], *Xiaojing Liao*[1], *Luyi Xing*[1]

[1]*Indiana University Bloomington,* [2]*Orion Security Lab, Alibaba Group*

Abstract

As a key supplement to privacy policies that are known to be lengthy and difficult to read, Apple has launched app privacy labels, which purportedly help users more easily understand an app's privacy practices. However, false and misleading privacy labels can dupe privacy-conscious consumers into downloading data-intensive apps, ultimately eroding the credibility and integrity of the labels. Although Apple releases requirements and guidelines for app developers to create privacy labels, little is known about whether and to what extent the privacy labels in the wild are correct and compliant, reflecting the actual data practices of iOS apps.

This paper presents the first systematic study, based on our new methodology named *Lalaine*, to evaluate data-flow to privacy-label (*flow-to-label*) consistency. *Lalaine* analyzed the privacy labels and binaries of 5,102 iOS apps, shedding light on the prevalence and seriousness of privacy-label non-compliance. We provide detailed case studies and analyze root causes for privacy label non-compliance that complements prior understandings. This has led to new insights for improving privacy-label design and compliance requirements, so app developers, platform stakeholders, and policy-makers can better achieve their privacy and accountability goals. *Lalaine* is thoroughly evaluated for its high effectiveness and efficiency. We are responsibly reporting the results to stakeholders.

1 Introduction

In December 2020, Apple launched its app privacy labels, which purportedly help users better understand an app's privacy practices before they download the app on any Apple platform. Without meaningful, accurate information, Apple's tool of illumination and transparency may become a source of consumer confusion and harm — a concern raised in the United States congress [11]. False and misleading privacy labels can dupe privacy-conscious consumers into downloading data-intensive apps, ultimately eroding the credibility and integrity of the labels. A privacy label without credibility and integrity also may dull the competitive forces encouraging app developers to improve their data practices.

Apple defines a set of compliance requirements and provides guidelines for app developers to create privacy labels. In particular, the Apple privacy label is not a direct abstraction of the app's privacy policy: Apple defines a four-layer structure for privacy label (i.e., with first layer — four data usages, second layer — six purposes , third layer — 14 data types, and fourth layer — 32 data items, see § 2.1). Hence, it is imperative to understand whether and to what extent the privacy labels in the wild are correct and compliant, reflecting the actual data practices of iOS apps. In response to this emerging privacy call, prior work studied challenges from developers' perspectives [77, 78] (e.g., how promptly developers update privacy labels) and end-users' perspectives [102], and offered tools [66] to help generate accurate privacy labels. A concurrent work [74], studied 1,687 iOS apps to understand privacy labels' correctness. However, fully inspecting compliance of iOS privacy labels at scale is challenging, which requires sophisticated analysis of app data-flow to privacy-label (*flow-to-label*) consistency, entailing to tackle the multiple-layer semantics of privacy labels (e.g., general but vague data types, purpose prediction, § 5.3.2) and rigorous definition of what are *flow-to-label* inconsistencies in the privacy label context (§ 4), which we aim to address in this paper.

Methodology for privacy-label compliance check. We present a new, automatic methodology called *Lalaine* to check the compliance of privacy labels for iOS apps at scale. Specifically, *Lalaine* checks the consistency between disclosure statements in privacy labels and actual data flows or practices of iOS apps (see the modeling in § 4). Essential is to formally define the inconsistency model for privacy labels, which cannot directly adopt prior inconsistency models for privacy policies [50, 53] due to privacy labels' different structure and semantics (e.g., with four layers, emphasizing data-usage purposes versus the party/entity that receives the data, see § 2.2).

Based on our new inconsistency model that formalizes key issues in privacy label compliance (§ 4), we design and implemented *Lalaine*, an automated, end-to-end system by adapting and synthesizing a set of innovative techniques including automatic iOS app UI execution, natural-language processing (NLP), and dynamic and static binary analysis (Figure 1). Notably, based on Apple, a data item is deemed "collected" only if it is transmitted to the Internet [47]. Hence, for a precise compliance check, *Lalaine* includes dynamic end-to-end app execution to find out the actual data "collection". Considering the known, limited scalability of dynamic analysis, *Lalaine* proposes a novel, optimized strategy by filtering apps of most interest: (1) apps that involve sensitive iOS APIs whose return values fall under Apple's framing of the 32 data items for privacy labels (§ 5.2.1) and (2) apps showing inconsistency between their privacy labels

and privacy policies. *Lalaine* further intersected the two app sets to yield three sub-sets with different (non-)compliance characteristics (e.g., apps being flow-to-label inconsistent while flow-to-policy consistent, apps being flow-to-label inconsistent while flow-to-policy inconsistent, flow-to-label consistent while flow-to-policy inconsistent, see § 5.3.1), and sampled a total of 6,332 apps for full dynamic execution. Another key challenge is to infer the vendors' actual purposes for the collection of each data item, which we tackled by adapting a modeling and learning-based approach (§ 5.3.2). Our thorough evaluation shows that *Lalaine* can detect privacy label non-compliance effectively and efficiently (§ 5.4).

Measurement and findings. Looking into the apps with privacy label non-compliance reported by *Lalaine*, we find the pervasiveness of privacy label non-compliance in iOS apps, with a serious impact on credible and transparent disclosure of app privacy practices. More specifically, among 5,102 iOS apps being fully tested, 3,423 apps came with non-compliant privacy labels: 3,281 of them neglected to disclose data and purposes, 1,628 contrarily specified purposes, and 677 apps inadequately disclose purposes. We characterize the findings with case studies and common cases, such as measurement results of most problematic data types, apps types, etc. (§ 6).

Also important are the root causes of privacy label non-compliance, as discovered in our study, which include opaque data collection from diverse third-party partners, and misleading privacy label disclosure guidance, etc. (detailed in § 7). Our findings characterize new challenges for designing, creating and regulating precise and proper privacy labels and based on which we come up with actionable recommendations for multiple stakeholders (§ 7).

Responsible disclosure. We are reporting all findings (non-compliant privacy labels) to Apple and related vendors.

Contributions. We summarize the key contributions below.

- We performed a systematic study on Apple privacy label compliance, in particular focusing on inconsistency between privacy labels and app data flows/practices for the first time. Our results shed lights on the prevalence, seriousness of privacy-label non-compliance and challenges for developers to achieve the emerging compliance goal.
- We introduce a first methodology with end-to-end implementation called *Lalaine* that can automatically assess the "flow-to-label" inconsistency. *Lalaine* is based on our new, formally defined inconsistency model for privacy labels (by adapting prior inconsistency models for privacy policies). *Lalaine* can be used by app developers for achieving compliance goals and by app stores for vetting purpose. We will release *Lalaine*.
- We provide detailed case studies and analyze root causes for privacy label non-compliance that complements recent understandings on privacy labels. This has led to new insights for improving privacy-label design and compliance requirements, so app developers, platform stakeholders, and policy-makers can better achieve their privacy and accountability goals.

2 Background

2.1 Privacy Labels of iOS Apps

As is illustrated in Figure 3, privacy label of an iOS app is a "nutrition label"-like privacy disclosure which lists what data is collected from the iOS app and how it is used. Privacy label has a specific focus on data collection, to help app users better understand how apps handle privacy-sensitive data. As indicated in [47], "collect" refers to transmitting data off the device in a way that allows app developers and/or their third-party partners to access it for an extended period. In addition, Apple asked developers to provide privacy labels following certain key requirements [47]: (1) The app developer are responsible for the disclosure for data collection of the whole app, including those collected by third-party partners; the app developer are required to disclose *all* purposes for collecting a data. (2) The app developers need to keep privacy labels accurate and up to date.

Taxonomy of privacy label. The privacy disclosure of the privacy label follows a four-layer taxonomy [47]: data usage, purpose, data type and data item. At the highest layer, these labels fall into four categories of data usage: *Data Used to Track You*, *Data Linked to You*, *Data Not Linked to You*, and *Data Not Collected*. Under each category, Apple defines six purposes except for the category *Data Not Collected*: *Third-Party Advertising*, *Developer's Advertising or Marketing*, *Analytics*, *Product Personalization*, *App Functionality*, and *Other Purposes*. For each purpose, the privacy label lists what data types (e.g., *Contact Info*) and specific data items (e.g., *Email Address, Phone Number, Physical Address, etc.*) are being collected. The taxonomy of privacy label defines 14 data types and 32 data items [47] covering various privacy-sensitive data ranging from personally identifiable information (PII) to health information.

2.2 Consistency Model

A consistency model is defined as a contract between regulators and rule-followers. In the privacy domain, a consistency model measures to what extent privacy practices (e.g., data collection behaviors in mobile apps) execute the rules (e.g., a privacy statement indicating which party collects what data) specified by the associated privacy disclosures (e.g., privacy policies) [50, 53]. More specifically, a disclosure can correctly and completely indicate critical information in a privacy practice if the privacy practice follows all rules in the disclosure. Otherwise, an inappropriate disclosure occurs, indicating that certain rules are violated by the privacy practice. The consistency model summarizes such inappropriate disclosures into several categories, according to the type of the rules obeyed by the privacy practice.

Commonly-used inconsistency detection logic. An inconsistency detection logic captures the difference between a data flow in a privacy practice and its associated rules in the

Table 1: Comparison with prior consistency models.

Method		PoliCheck [50]	PurPliance [53]	Our work
Disclosure		Privacy Policy	Privacy Policy	Privacy Label
Disclosure Representation		$\{s \mid s:(e,c,d)\}$	$\{s \mid s:((e,c,d),(d,k,q))\}$	$\{s \mid s:(d,q)\}$
Data Flow Representation		$\{f \mid f:(e,d)\}$	$\{f \mid f:(e,d,q)\}$	$\{f \mid f:(d,q)\}$
Definition of flow-f-related disclosure		$\mathbb{S}_f = \{s:(e_s,c_s,d_s) \mid d_f \sqsubseteq d_s \wedge e_f \sqsubseteq e_s\}$	$\mathbb{S}_f = \{s:((e_s,c_s,d_s),(d_s,k_s,q_s)) \mid e_f \sqsubseteq e_s \wedge d_f \sqsubseteq d_s \wedge q_f \sqsubseteq q_s\}$	$\mathbb{S}_f = \{s:(d_s,q_s) \mid d_f \sqsubseteq d_s \vee d_f \sqsupset d_s\}$
Inconsistent Disclosure Type	Omitted Disclosure	$\mathbb{S}_f = \emptyset$	$\mathbb{S}_f = \emptyset$	$\mathbb{S}_f = \emptyset$ (neglect disclosure); $\mathbb{S}_f \neq \emptyset \wedge q_f \notin Q_f^S \wedge Q_f^S \nsubseteq Q_f\}$ (contrary disclosure); $\mathbb{S}_f \neq \emptyset \wedge q_f \notin Q_f^S \wedge Q_f^S \subset Q_f\}$ (inadequate disclosure)
	Incorrect Disclosure	$\exists s \in \mathbb{S}_f$ s.t. $c_s =$ not_collect	$\exists s \in \mathbb{S}_f$ s.t. $c_s =$ not_collect $\vee k_s =$ not_for	N/A

privacy disclosure. A rule or a data flow can typically be formalized as a triplet in the form of (e,c,d), where e is subject, c is predicate and d is object. Here, the predicate can be along with positive or negative sentiments. For example, as shown in Table 1, PoliCheck [50] defines the subject e as *platform entities* (e.g., first-party or third-party), the object d as *data objects* (e.g., email address), and the predicate c as *data collection action* with positive (i.e., collect) and negative (i.e., not collect) sentiments. In addition, PurPliance [53] considered the data usage (i.e., purpose) in the consistency model, and extended the representation of a rule in privacy disclosures to be $((e,c,d),(d,k,q))$ where k is the predicate associated with data usage (e.g., "used for" and "not used for"), q is the purpose of data usage (e.g., advertising for third party).

Previous works study the inconsistency between privacy disclosure and data flows whose predicate sentiments c,k are opposite. More specifically, the prior inconsistency detection logic [50, 53] generally are defined between two representations with *different* predicate sentiments c and k, *correlated* entities e, data objects d and data usage q, i.e., with the definitions of flow-f-related disclosure $\mathbb{S}_f = \{s:(e_s,c_s,d_s) \mid d_f \sqsubseteq d_s \wedge e_f \sqsubseteq e_s\}$ [50] or $\mathbb{S}_f = \{s:((e_s,c_s,d_s),(d_s,k_s,q_s)) \mid e_f \sqsubseteq e_s \wedge d_f \sqsubseteq d_s \wedge q_f \sqsubseteq q_s\}$ [53]. The correlation between e,d,q of two representations is modeled using four semantic relations: synonym ($d \equiv d'$), approximation ($d \approx d'$), hyponym ($d \sqsubset d'$), and hypernym ($d \sqsupset d'$).

It is worth noting that in privacy label, the data usage purposes of a privacy-sensitive data item are presented only with the positive sentiment. In our study, we analyze inconsistency disclosure with specific focuses on data usage purpose (see Section 4) instead of predicate sentiment.

Types of Inconsistency Disclosure. Previous studies [50, 53] summarize several inconsistency types to inspect to what extent data flows can be disclosed by a privacy disclosure (see Table 1). The typical inconsistency types include omitted disclosure and incorrect disclosure [50, 53]. *Omitted disclosure* indicates a data flow which is not discussed by any policy statements (i.e., $\mathbb{S}_f = \emptyset$). And *incorrect disclosure* recognizes a data flow if a policy statement indicates that the flow will not occur (i.e., a negative sentiment collection statement) and there is not a contradicting positive sentiment statement (i.e., $\nexists s \in \mathbb{S}_f$ s.t. $c_s =$ collect $\vee k_s =$ for). In this paper, we did not include *logical contradictions*, which is defined on a pair of policy statements in the disclosure that have contradictions. This is because that compared with privacy policy, privacy label is a better-structured privacy disclosure where the logic contradicted privacy statement does not exist. Since logical contradiction is out of scope of this work, the prior type of ambiguous disclosure [50, 53] will not be discussed in this paper. In contrast to the previous works on "flow-to-policy" consistency, to analyze flow-to-label consistency, we need to focus on inconsistency of data usage purpose. Accordingly, we formally define three types of inconsistency for privacy labels : *neglect disclosure, contrary disclosure and inadequate disclosure* (Section 4) to fine-grain the type of *Omitted disclosure* in PurPliance [53].

2.3 Scope of Problem

Our study focuses on the inconsistency between privacy label and data flows in iOS apps to check the non-compliance. Between the privacy policy and privacy label, we perform a new measurement study to understand their divergence. We did not directly compare data flows with privacy policies since such a task has been conducted extensively in the literature [50, 69, 86, 103].

Regarding the four layers of privacy labels (§ 2.1), our study investigated whether the data items (the fourth layer) and purposes (the second-layer) have been disclosed *correctly* and *completely*.

For data types (the third layer), each data type covers several data items (e.g., *Identifiers* covers two data items: User ID and Device ID) defined by Apple. We did not scrutinize the first layer. This is because that for some data (e.g., vendor-specific data items like Apple device ID), linking data to the user's identity (e.g., via their account) are usually taken place at the server-side, which may not be precisely identified from client-side behavior. For example, although both Apple apps *Contacts* and *Apple news* collect a device identifier, based on their privacy labels, the *Contacts* app does not link it to user identity, while *Apple news* does. However, it can be hard to differentiate those two from client side.

Scope of data items. Apple defined 32 privacy-sensitive data items (the fourth layer in privacy labels), which if collected, should be disclosed in privacy labels [47] (e.g., Device ID, Precise Location, referred to as *l-data item*s in our

study). *l-data items* originate from three sources: (1) Apple system-level API (e.g., Device ID, Location, Contacts); (2) app-level code including third-party libraries (e.g., User ID, Advertising Data, Crash Data); (3) user input through graphical interface (e.g., Sensitive Info including sexual orientation, religious or philosophical beliefs, etc.). In § 5, we propose complementary static and dynamic analysis techniques, called *Lalaine*, aiming at capturing data items from all three sources for compliance check of privacy labels.

3 Related Work

App privacy labels and other short-form privacy disclosures. P3P [37] specification studied in [55, 59, 67, 88], served as predecessor of privacy label, allows websites to express their privacy practices in a standard machine-readable format [36]. Privacy nutrition labels have been proposed or studied in the literature [32, 61, 62, 71, 72, 73]. Prior works perform user study to understand challenges from developers' perspectives [77] and end-users' perspectives [102]. Li et al. [78] studied how promptly developers create and update privacy labels. Gardner et al. [66] developed a tool combining static analysis and interactive wizards to help iOS developers generate accurate privacy labels. News reports [3, 31, 44, 46] showcased inaccurate privacy labels of iOS apps or suspected their compliance quality, which partially motivated our systematic study. Koch et al. [74], concurrent with our study, conducted a no-touch traffic analysis of 1,687 iOS apps to study privacy labels' correctness. Our work significantly complements theirs [74] in key aspects such as coverage: no-touch inspection without UI interactions cannot thoroughly trigger app execution, leading substantial false negatives; [74] only considered one kind of inconsistency —"Neglect Disclosure" and did not analyze data collection purposes — key part of privacy labels. Compared with previous works, we perform a systematic study on Apple privacy label compliance by investigating three types of data-flow to privacy-label inconsistencies with formal definitions, end-to-end tools, uncovering root causes of non-compliance and actionable takeaways.

Privacy compliance check on mobile apps. Privacy compliance check [48, 50, 51, 54, 81, 82, 93, 100, 101, 103, 104] are evolving from coarse-grained analysis to complex but fine-grained analysis, from the data-level consistency, which checks whether the return value of sensitive API is described in the policy [93, 104], to data-purpose-level consistency [53]. Recent work (e.g., PoliCheck [50]) take into account the entities (third-party vs first-party) of the personal data and proposed an entity-sensitive consistency model. PurPliance [53] propose a new consistency model by extending the PoliCheck [50] through incorporating data usage purposes. However, PurPliance's ontology didn't cover all data items defined by Apple and its taxonomy of data-usage purpose didn't differentiate the purpose between entities, which can not be directly applied to our study. To fill this gap, we crafted a mapping table considering both purpose and entity in the privacy policy and enhanced the ontology (see § 5.2.2). Besides, essential for analyzing privacy labels, we define a new consistency model (with comparison in § 4), enabling three types of inconsistency analysis between disclosures in privacy labels and data practices in iOS apps.

4 New Inconsistency Model for Privacy Label

In this paper, we propose a new inconsistency model essential for analyzing *privacy label*, which includes three types of inconsistent disclosure based on whether the usage purposes of a certain data object are *omitted*, *contrary*, or *inadequate* in the privacy label. Compared with previous works [50, 53] which defined the consistency model on the predicate sentiment (see Section 2), our consistency model takes full advantage of the succinct expression of the privacy label to fine-grain the *ommitted disclosure issues* and rationalize them using inconsistent data usage purposes. More specifically, we aggregate privacy statements in privacy labels and data flows in an iOS app with the correlated data objects (i.e., $\mathbb{S}_f = \{s : (d_s, q_s) | d_f \sqsubseteq d_s \vee d_f \sqsupset d_s\}$), and identify three types of inconsistencies between the data usage portfolios disclosed in privacy labels and actual data flows in iOS apps. We summarize the comparisons with the previous consistency models in Table 1. Below we introduce the formal representation of the privacy label, data flows, and the semantic relationship between data objects.

Definition 1 (Privacy Label Representation)**.** Given the set of data items D and data usage purposes Q, an app's privacy label is modeled as a set of tuples $S(D,Q) = \{s | s : (d_s, q_s), d_s \in D, q_s \in Q\}$. Each tuple s represents a privacy statement for a certain privacy-sensitive data item d_s, which discloses one purpose q_s that d_s is supposed to be used for (see Figure 3). Note that one data item can be associated with multiple data usage purposes, and thus different privacy statements can be associated with the same data item; e.g., "Device ID" can be used for both *Analytics* and *Third-Party Advertising* purposes.

Definition 2 (Data Flow Representation)**.** An individual data flow f in an app is represented as a tuple $f = (d_f, q_f)$, where $d_f \in D$ is a privacy-sensitive data item and q_f is the usage purpose of d_f in flow f. In mobile apps' data collection, a certain data item can be associated with multiple usage purposes in different flows. We define the purposes in all flows with the same data item with f as the purpose portfolio of the flow $Q_f = \{q_{f'} | f' = (d_{f'}, q_{f'}), d_{f'} = d_f\}$.

Definition 3 (Semantic Relationship)**.** Similar to prior work [50, 53], we use an ontology of data items to capture the relationship between data items (e.g., `DeviceInfo` is a hypernym of `DeviceID`). Given an ontology o and two terms u, v, we denote $u \equiv_o v$ if u, v are *synonyms* with the same semantic meaning. Otherwise, if u is a general term and v is a specific term whose semantic meaning is included in u, we denote $v \sqsubset u$ or $u \sqsupset v$. In this case, u is called a *hypernym* of v, and v is called a *hyponym* of u. $u \sqsubseteq_o v$ is equivalent with $u \sqsubset_o v \vee u \equiv_o v$.

Definition 4 (Flow-relevant Privacy Label)**.** The privacy disclosure $s = (d_s, q_s)$ for data item d_s is relevant to the flow $f = (d_f, q_f)$ (denoted as $s \simeq f$) if and only if $d_s \sqsubseteq_o d_f$ or $d_s \sqsupset_o d_f$. Let $\mathbb{S}_f = \{s | s \in S \wedge s \simeq f\}$ denote the set of flow-f-relevant privacy label. Let $Q_f^S = \{q_s | s \in \mathbb{S}_f\}$ denote all data usage purposes in flow-f-relevant privacy label, which is also called the purpose portfolio of the privacy label relevant to f.

Given a flow f, by comparing its purpose q_f, its purpose portfolio Q_f, and the purpose portfolio of its relevant privacy label Q_f^S, we reveal the flow-to-label inconsistencies and categorize them into three types: *Neglect Disclosure*, *Contrary Disclosure*, and *Inadequate Disclosure*.

Inconsistency 1 ([$\nVdash$] Neglect Disclosure)**.** The privacy label S is a *neglect disclosure* with regard to a flow f if there exists no flow-f-relevant privacy label in S:

$$\mathbb{S}_f = \emptyset \;\Rightarrow\; S \nVdash f.$$

Intuitively, Neglect Disclosure means that a data item is collected but not disclosed in the apps' privacy label.

Inconsistency 2 ([$\nvDash$] Contrary Disclosure)**.** The privacy label S is a *contrary disclosure* with regard to a flow f if the flow-f-relevant privacy label $\mathbb{S}_f$ exists while the purpose of the flow is not in $\mathbb{S}_f$ and there exists at least one purpose in $\mathbb{S}_f$ which is not in the flow's purpose portfolio Q_f:

$$\mathbb{S}_f \neq \emptyset \wedge q_f \notin Q_f^S \wedge Q_f^S \nsubseteq Q_f\} \;\Rightarrow\; S \nvDash f.$$

Intuitively, Contrary Disclosure means that the data collection purposes declared in the app privacy label are not all consistent with the actual purposes (e.g., a declared purpose of Third-Party Advertising compared to the actual purpose of App Functionality).

Inconsistency 3 ([$\nvdash$] Inadequate Disclosure)**.** The privacy label S is an *inadequate disclosure* with regard to a flow f if the flow-f-relevant privacy label $\mathbb{S}_f$ exists while the purpose of the flow is not in $\mathbb{S}_f$ and the purpose portfolio in $\mathbb{S}_f$ is a *proper subset* of the flow's purpose portfolio:

$$\mathbb{S}_f \neq \emptyset \wedge q_f \notin Q_f^S \wedge Q_f^S \subset Q_f\} \;\Rightarrow\; S \nvdash f.$$

Intuitively, Inadequate Disclosure indicates that the privacy label correctly discloses *partial* rather than all usage purposes of the data objects related with d_f. Note that in our application (§ 6.3), it is possible that the usage purposes extracted from the data flows are not complete, while this will not erroneously result in false positives of inadequate disclosure.

5 Methodology

5.1 Overview

Architecture. Our approach relies on the extraction of data-purposes pair from dynamic code behavior and analysis of inconsistency with its corresponding privacy label. In particular, the design of *Lalaine* includes three major components: data collection and preprocessing, Static Assessment Framework (*SAF*, § 5.2), Dynamic Assessment Framework (*DAF*, § 5.3), as outlined in Figure 1. First, *Lalaine* collects a large set of 366,685 iOS unique apps from Apple app store (U.S.) from October 29, 2021 to April 26, 2022, with their privacy labels and privacy policies simultaneously collected. We used the privacy label closest to the app download time and checked the version history to make sure the app version is the latest one listed in the privacy label page. Second, the *SAF* analyzes the app binaries and their privacy labels and policies, and outputs two app sets: (1) apps that use the iOS system APIs whose return data, if collected by the apps, should be disclosed in iOS privacy labels — denoted as *API* including 161,262 apps; (2) apps whose privacy label and privacy policy have inconsistent data practice disclosures (regarding data collection and usage purposes) — denoted as *PP* including 53,376 apps. Last, based on the apps of interest (i.e., sets *API* and *PP*), the *DAF* includes a dynamic analysis pipeline which performs end-to-end execution (fully automated app UI execution, dynamic instrumentation, and network monitoring). The pipeline strategically takes a subset of the apps for full execution to maximize the precision of detection for privacy-label non-compliance while maintaining the scalability of the study (detailed in § 5.3). Based on the context that features the data practices (e.g., API caller, stack traces, server endpoints, Web requests, and responses) collected by the pipeline, *DAF* infers the actual data collected and purposes of collection, and aligns them, abstracted as tuples $(data, purpose)$, to the 32 data items and 6 purposes defined by Apple. *DAF* finally reports privacy label non-compliance by comparing those tuples with disclosures in the vendors' privacy statements.

Data collection. In our research, we collected 366,685 iOS apps for privacy label compliance checks. Specifically, we reuse the more than one million Android app names from prior works [97], assuming that the iOS and Android versions of an app might share similar names. This approach turned out to be efficient: we collected 485,024 unique app Bundle IDs from the Apple app store (U.S.). Given those Bundle IDs, we ran an app crawler on 10 iPhones to download and decrypt those apps (see technical details [30]). In this way, we collected 366,685 iOS apps, which cover 27 app categories. Meanwhile, for each iOS app, we also collect its privacy label and privacy policy.

5.2 Static Assessment Framework

As mentioned earlier (§ 5.1), the *SAF* takes in our data set (app binaries with their privacy labels and privacy policies) and finds out two sets of apps, i.e., *API* (apps using the iOS system APIs returning *l-data items*) and *PP* (apps with inconsistency between privacy label and privacy policy). Those apps will be further validated through dynamic analysis in § 5.3.

5.2.1 Sensitive-API Analyzer

Although Apple provides a list of 32 data items (e.g., *Device ID*, *Health*, called *l-data items*) whose collection are expected

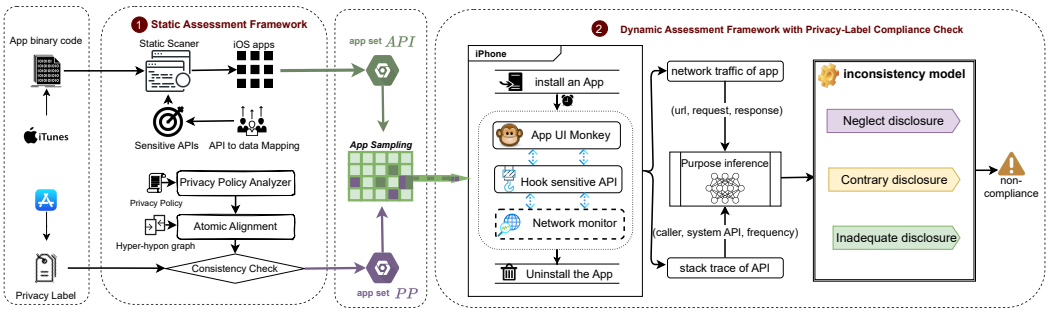


Figure 1: Overview of *Lalaine*.

to be disclosed in privacy labels, such information is insufficient and cannot be directly used to find out whether an app involves/collects such data. The gap is that we lack a precise and comprehensive mapping between the data items and corresponding iOS system APIs that return the data — we aim to identify apps that access the data by inspecting their invocation of those APIs. Preparing such a mapping is non-trivial due to (1) the substantial amount of iOS system APIs (e.g. more than 200,000 public APIs on the recent iOS 15.5) whose documents often lack descriptions about the return values or are even completely missing [42]; (2) the *l-data item* being too general without a clear definition of its scope. For example, the data item *Device ID* is general and unique on iOS, including *the identifier for advertisers* (IDFA), *the identifier for vendors* (IDFV), and possibly many others, and it is never apparent for app developers whether accessing, for example, IDFV (or calling specific APIs such as `identifierForVendor` [25], the iOS API that returns IDFV (see § A.1)), should be disclosed in privacy labels (see measurement results in § 6).

***l-data item* to iOS-API mapping (l-mapping).** To enable a privacy-label compliance tool like *Lalaine*, we propose and release an *l-data item* to iOS-API mapping (called *l-mapping*). First, similar to [93], we gather a set of candidate APIs by utilizing the enhanced ontology (§ 5.2.2) that provides subsumptive relations between low-level technical terminology and high-level privacy terms. Specifically, if privacy-label data items subsume the data value returned by API, then this API will be regarded as a candidate. Then, to validate the APIs, we manually inspected iOS API documents (with development of end-to-end proof-of-concept apps to check the return values of the APIs) and summarized a list of iOS APIs corresponding to five types of *l-data items*: Device ID, Location, Contacts, Health, Performance Data. We release l-mapping online [30].

Static scanner. To support a large-scale study, we developed a lightweight, effective static analysis to screen apps that call iOS system APIs recorded in the *l-mapping*. *Lalaine* considers that an app does not collect, for example, IDFA (a Device ID), if the corresponding iOS API `-[ASIdentifierManager advertisingIdentifier]` cannot be found in the app's binary code. More specifically, our static analysis is adapted for the unique function call mechanism of Objective-C [34] and Swift [39]. In these languages, API names are composed of two parts: a class name (e.g., *ASIdentifierManager*) and a selector name (e.g., *advertisingIdentifier*). The invocation of an API is compiled to an instruction-level call to the iOS-unique *objc_msgSend* function with the API's selector name (a string) passed in as the second argument [56, 57, 96]. Essentially, that is, an API's selector name string will appear in the binary if the API is called by the app. Hence, our static scanner searches the selector names in the apps' binaries (using the *grep* command [22]) and filters out apps that do not invoke APIs in the *l-mapping*.

Evaluation. Through manual confirmation by domain experts of iOS and privacy, we ensured our l-mapping was precise (each API returns a *l-data item*). For best coverage, we further manually checked 293 iOS frameworks (iOS 15.5) that are likely to include APIs related to the five *l-data items* and confirmed that our l-mapping was comprehensive. Regarding the static scanner, prior approaches [57, 64, 92, 96] with analysis of control flows are too heavy-weight for our task (e.g., 54 minutes per app [96]); in contrast, our *SAF* spent 20 hours to scan 366,685 apps, also precluding manual analysis of apps at this scale for cost efficiency. To evaluate coverage, we randomly sampled 20 apps that were filtered out by *SAF*, manually exercise UI and functionalities for 10 minutes for each app, and in the meantime, used Frida to hook the iOS APIs in the l-mapping. In this experiment, *SAF* showed high coverage (100%) for static screening of APIs in l-mapping. Further, potential false-positives (e.g., if non-system APIs share names with an iOS API) in *SAF*'s static scanner introduces no impact for *Lalaine*'s end-to-end accuracy: the dynamic-testing (§ 5.3.1) guarantees precise hooking of APIs of interest (using class-name and API-name) based on Frida [20].

5.2.2 Privacy Label-to-Policy Consistency Checker

To identify apps with inconsistent privacy label and privacy policy, in this subtask, we aim at aligning data objects and their usage purposes mentioned in privacy label and privacy policy for inconsistency check. More specifically, we use PurPliance [53] to retrieve data items and the purpose of data

usage from the privacy policy, and map them into the taxonomy of the privacy label with 32 data items and 6 purposes (see § 2). Below we elaborate on the mapping process.

• *Data object alignment.* Apple defined 32 specific data items as the taxonomy of privacy label (Section 2). However, given an app's privacy policy, the extracted data object d_p can not be simply matched to the data item d_l of privacy label using keyword matching. This is because the semantic levels of d_p and d_l are sometimes different. For example, considering the sentence extracted from privacy policy *"we will collect Android ID, IMEI, IDFA ..."*, here the sensitive data objects are *Android ID, IMEI, IDFA*. However, those data objects are not matched to the 32 data items in privacy label, but they are semantically subsumed to *"Device ID"* defined by Apple which represents any device-level ID.

To fill this gap, we enhanced the data object ontology in [49] to determine the semantic relationship (*Synonym*, *Hypernym*, *Hyponym*) between d_p and d_l and align data objects in different granularity. More specifically, to recover their subsumption relationships, we use data object ontology in [49], which is a graph-based data structure where each node represent a data object and each edge represents a relationship among hyponym and hypernym (e.g., "personal information" subsumes "your email address"). However, the ontology in [49] is built on the corpus of the privacy policy, there miss some nodes and edges defined by the privacy label taxonomy. For example, Apple defined *Emails or Text Messages* includes *subject line*, *sender*, *recipients* and etc. Those data objects and their subsumption edges are not in the current ontology. Hence, we manually enlarged the ontology to cover such new edges (e.g., emails or text messages ⇒ subject line). In total, we added 75 nodes and 193 edges to the original ontology. We release the ontology online [30]. Given the ontology, we align d_p and d_l if d_l is a synonym, hypernym, or hyponym of d_p.

• *Purpose alignment.* It's non-trivial to fill the gap between the data usage purpose mentioned in the privacy policy and those in the taxonomy of the privacy label because the taxonomy of data-usage purpose in [53] did not differentiate the purpose between first-party and third-party. For example, the *Advertising* purpose should be differential from the entity (i.e., first-party or third-party) in the privacy label: first-party advertising is aligned with *"Developer's Advertising or Marketing"* while third-party advertising is corresponding to *"Third-Party Advertising"* based on Apple's definition of purposes. Hence, we crafted a mapping table considering both purpose and entity in the privacy policy, as shown in Table 2.

• *Comparison.* After data object and purpose alignment, privacy policy statements can be represented as (data, purposes). As the privacy label is a structured data in the form of (data, purposes), we then compare the 2-tuples of privacy policy and privacy label using the definition of three inconsistent disclosure in § 4.

Table 2: Purpose Alignment

Privacy Policy		**Privacy Label**
Purpose	*Entity*	*Purpose*
Functionality	First Party	App Functionality Product Personalization
Advertising	First Party	Developer's Advertising or Marketing
	third Party	Third-Party Advertising
Analytics	-	Analytics
Marketing	First party	Developer's Advertising or Marketing
Other	-	Other Purposes

5.3 Dynamic Assessment Framework with Privacy-Label Compliance Check

Based on the apps of interest found by *SAF* (i.e., *API* and *PP*), the *DAF* performs automatic end-to-end execution on a strategically sampled subset of the apps to maximize the precision of the privacy-label non-compliance detection while maintaining the scalability of the study. *DAF* includes two key components: (1) a dynamic analysis pipeline that automatically runs the apps with dynamic instrumentation and network monitoring (§ 5.3.1) and (2) a compliance checker that reports inconsistencies between the actual data practices and privacy labels (§ 5.3.2).

5.3.1 Dynamic Analysis Pipeline

App sampling. From our data set of 366,685 apps (§ 5.1), *SAF* yields 161,262 apps denoted as *API*, and 53,376 denoted as *PP*. Limited by the scalability of dynamic analysis, we narrow down the test scope while preserving the generalization of results by performing a strategic sampling over the apps. Specifically, we perform an intersection of *API* and *PP*, yielding three app sets: $API \setminus PP$, $API \cap PP$, $PP \setminus API$, which will help characterize different levels of privacy-label noncompliance. We sampled apps from each of the three sets and obtained 6,332 apps for full testing using *DAF*:

• $API \setminus PP$. These are apps in set *API* but are not in set *PP*. Intuitively, these are apps that invoke iOS system APIs accessing *l-data item* while their privacy labels and privacy policies are consistent (regarding disclosure of *l-data item* usage/collection, see § 6). Data practices of apps in $API \setminus PP$, can be inconsistent with both the privacy labels and privacy policies. In our study, $API \setminus PP$ includes 140,944 iOS apps, from which we sampled 4,593 for dynamic analysis (see below).

• $API \cap PP$. $API \cap PP$ are apps that invoke iOS system APIs accessing *l-data item* while their privacy labels and privacy policies are inconsistent. Data practices of apps in $API \cap PP$, even if inconsistent with privacy labels, can be consistent with privacy policies, or vice versa (consistent/inconsistent with privacy labels/policies, see measurement results in § 6). In our study, $API \cap PP$ includes 20,318 iOS apps, from which we sampled 662 for dynamic analysis.

• $PP \setminus API$. $PP \setminus API$ are apps with inconsistencies between privacy labels and privacy policies, while they do not apparently access *l-data item*s (i.e., their code is not found to invoke iOS system APIs returning *l-data item*s). Notably,

Lalaine considers that the apps might (1) leverage runtime techniques [68, 97] (such as reflection) that evade the static screening (§ 6); (2) access *l-data item*s that originate from the app-level code or third-party libraries. For best coverage, our dynamic analysis inspects all possible *l-data item*s from the network traffic of the apps (see below). Data practices of apps in $PP \setminus API$, can be inconsistent with either privacy labels or privacy policies, or both of them. $PP \setminus API$ includes 33,058 apps, from which we sampled 1,077 for dynamic analysis.

App UI execution. Our pipeline automatically installs each app (using the *ideviceInstaller* command [15]) and schedules it to run on a set of jail-broken iPhones. Specifically, similar to common practices [76, 80, 83, 89, 91] on Android, we leverage an open-source, off-the-shelf UI execution tool, called *nosmoke* [41], to generate actions and automatically trigger the dynamic execution of an app, such as clicks or swipes, through the user interface (UI). Notably, *nosmoke* is designed to identify textual information and types of UI elements using the OCR techniques [94] from screenshots. Specifically, it identifies actionable UI elements in the current app window and executes target actions based on a simple configuration (e.g., clicks for a button or edit for a input, see the configuration details in [30] based on depth-first search (DFS) algorithm). In our pipeline, we set the maximum depth of the UI window stack as 5, and the maximum number of actions on one window as 15; each app is scheduled to run for three minutes and then uninstalled. With 4 iPhones concurrently running, *Lalaine* could execute about 600 apps each day.

Evaluation. Compared to sophisticated, manual UI execution that can properly handle app-specific text inputs, automatic UI execution can have lower coverage. As an evaluation, with 100 sampled apps, we manually executed each app with exhaustive UI interactions for up to three minutes (same period as *DAF*), and compared the coverage with *DAF* based on two metrics: *DAF* covered 61.01% of unique network traffic (unique to server API endpoints) and 43.34% of unique call-stack traces (unique to iOS APIs hooked) of manual UI execution. Hence, although *Lalaine* reported many non-compliance of privacy labels (§ 6), it could still be a low bound of results, indicating the seriousness of the problem. Our further evaluation on 150 apps also shows that running nosmoke for three minutes (our configuration) yields a 90% of network traffic compared to a longer time such as 10 minutes (see details in [30]).

Dynamic instrumentation and network monitoring. Similar to common approaches [63, 75, 95], during the app execution, *Lalaine* leverages *Frida* [20], a dynamic instrumentation tookit, to hook iOS systems APIs returning *l-data item*s (see the l-mapping in § 5.2.1). In particular, we inspect the APIs' argument values and return values (using the *onEnter(args)* and *onLeave(retval)* APIs of Frida [20] respectively), and obtain the call-stack traces (using the *Thread.backtrace([context, backtracer])* API [20]). *Lalaine* matches the APIs' return values with network traffic to confirm the data collection. Similar to [70, 85, 89, 90, 91], *Lalaine* adopted a popular network monitoring tool *Fiddler* [33] to decrypt and inspect app traffic.

• *Mapping call-stack traces to network-traffic.* We came up with a mapping (denoted as *c-mapping*: $caller \rightarrow endpoint$) between the system API caller (class/function) from call traces and endpoints in network traffic. First, similar to prior approaches [50, 58, 87], *DAF* performed a case-insensitive match of the return value of system APIs to content values in network traffic, and uniquely mapped 13,038 (31.8%) call traces to corresponding network traffic. In the app *com.yinzcam.venues.unitedcenter*, for example, the system API *identifierForVendor* is only invoked by the caller function *[YCPushService registrationJSONDataForTags:]* and its return value is only transmitted to the endpoint *yinzcam*. We add such a mapping to *c-mapping*.

• *Enhancing the mapping for high coverage.* Second, we enhanced *c-mapping* using *cocoapods* [10], a library repository for iOS apps with over 92,000 libraries. Specifically, we observed that sensitive data are often collected by third-party libraries being agnostic to individual apps (e.g., the caller function *[FlurryLocationInfoSource onqueue_dataProviderDidUpdate:] in flurry library* transmits user location to the *flurry* server). To gather such information, we crawled all libraries in *cocoapods* with their metadata (*podspec.json* file of each library includes its owner and domain) as a knowledge base. For example, by searching the caller `KVAAdapter: valueForContext:touchlessBool:waitBool: completionHandler:` that invoked the system API *advertisingIdentifier*, *Lalaine* found the caller was in the KochavaTrackeriOS_4.3.1 SDK with domain *kochava.com* [29]. Finally, *c-mapping* allows Lalaine to map 34,098 (83.17%) call traces to traffic that transmits l-data (see limitation in § 5.5).

5.3.2 Privacy-Label Compliance Check

Overview. Based on Apple, an *l-data item* is deemed "collected" by a vendor only if it is transmitted to the Internet [47]. For our compliance check, *Lalaine* first infers the *l-data item* that is actually collected based on both network traffic and evidence found in dynamic instrumentation. Another key challenge is to infer the vendors' actual purposes for the collection of each *l-data item*s (e.g., *App functionality*, *Third party advertising*, falling under categories defined by Apple), which we tackled by adapting a modeling and learning-based approach. *Lalaine* finally reports whether the tuple (*l-data item*, purpose) representing actual data practices is consistent with the app's privacy labels (see the consistency model in § 4).

Inference of *l-data item* collection. Prior approaches [70, 89, 90] to identify privacy data from network traffic are often limited to specific data types. Tailored to *l-data item*s, *DAF* extends prior value-matching approaches [50, 58, 87] that match sensitive data in traffic to data observed in dynamic instrumentation (data returned by iOS system APIs, see § 5.3.1). For example, in traffic of the app *TED* [40], *DAF* found the value

`9F0B31E6-980B-4468-9797-0B1F1A8FA56E` that matched the IDFA (a Device ID) returned by the iOS API *advertisingIdentifier* (returning IDFA, an Device ID). Like common approaches [86, 87, 91], *DAF* also employed common data transformations/encoding schemes (MD5, SHA1, SHA-256, SHA-512, URL-encoding, adding hyphens and underscores, and quotes, etc.) to improve coverage.

For *l-data item*s that may not be returned by system APIs (§ 2.3), *DAF* searches them in network traffic based on a crafted list of 197 keywords that may identify *l-data item*s. For example, the keyword *device_identifier* found in network requests of many apps indicates that the collected data is a *Deivce ID* (the traffic, often being Web API traffic, includes key/value pairs); *DAF* used the keyword *blood_type* to find data collection from traffic aligned to the *l-data item Heath*. To collect the keywords, we inspected the network traffic of 259 iOS apps (113 from Apple and 146 from high-profile vendors), each running for ten minutes using *SAF* and 15 minutes with exhaustive manual interactions, to collect a comprehensive list of keywords (see details in [30]).

Evaluation. We randomly selected 450 network requests from 136 apps. Our manual inspection of all data in the traffic shows a precision of 99.8% in *Lalaine*'s inference of l-data items.

Inference of *l-data item* collection purposes. In our study, we analyze a set of high-profile iOS apps and their privacy labels, and identify their features for purpose prediction. More specifically, we select features tailored to the purpose categories and definitions from Apple, particularly those considered to be robust, in the sense that missing these features might disable the classifier to correctly differentiate different purposes. To this end, we extended MobiPurpose [70] to 13 features, categorized into three groups: external app information, traffic information, and call trace information. The external app information is extracted from the app download page on the Apple store, including *Bundle ID, App name, Company name*. We also leverage the context information found in dynamic instrumentation (e.g., call stack traces) and semantics in network traffic to abstract a six-tuple: $(caller, systemAPI, frequency, endpoint, request, response)$ for each *l-data item*. Such an abstraction encodes essential context information for *Lalaine* to infer the collection purpose, which can be mapped to those defined by privacy labels. A description of these features is provided in [30]. In our study, we use 6 new features geared towards data collector, data receiver and collection behavior to find the signal of the purpose of this data collection behavior. For example, we calculate API frequency based on the observation that frequent invocation might indicate a suspicious intention (e.g., tracking the user's precise location multiple times in one second). This feature is effective for our task, which yielding a performance improvement of 1.436%. The details about feature extraction, feature example, feature importance, and implementation details are in [30].

Evaluation. We elaborate on the model evaluation below.

Table 3: Evaluation of Purpose Identifier.

Purpose	Precision	Recall	F1
Analytics	0.97	0.98	0.98
App Functionality	0.92	0.97	0.94
Developer's Advertising or Marketing	0.95	0.77	0.85
Product Personalization	0.86	0.68	0.76
Third-Party Advertising	0.97	0.98	0.98
Macro Avg	0.94	0.88	0.90
Weighted Avg	0.95	0.95	0.95

• *Dataset.* We collected and manually validated 259 high-profile iOS apps as groundtruth (113 developed by Apple and 146 from other high-profile vendors). Each app goes through the dynamic analysis pipeline (§ 5.3.1) complemented with ten-minute manual app usage, and then we obtained the six-tuple for each *l-data item* found in app traffic. These six-tuples are manually aligned with (*l-data item*, purpose) pairs extracted from privacy labels to establish ground-truth. In total, four annotators labeled 2,958 pieces of traffic which contact 184 distinct domains and achieved 0.923 agreement score calculated by Fleiss's kappa [65]. The annotation criteria are detailed in [30].

The groundtruth dataset consists of 2,958 samples in five data usage purpose (1,035 as *Analytics*, 739 as *App Functionality*, 445 as *Developer's Advertising or Marketing*, 141 as *Product Personalization*, and 598 as *Third-Party Advertising*). We use 75% samples as the training set to train the model and 25% samples as the testing set to evaluate our approach. The class labels (i.e., purposes) in the training set and the testing set share the same marginal distribution.

• *Experiment results*. The inconsistent disclosure model (see § 4) focuses to reveal the data usage purposes in practice which are not reflected by the associated privacy label. Therefore, the false positives in purpose identification (i.e., the predicted purpose does not occur in practice) will result in false alarms in inconsistent disclosure. To avoid false alarms as possible, we tune the hyper-parameters of our model towards high precision, which indicates less false positives in each purpose, during the cross-validation on the training set. Then we evaluate Purpose Identifier on the testing set and the results are shown in Table 3. The proposed model achieves high precision (i.e., 0.97, 0.92, 0.95, 0.86, and 0.97) and tolerable recall (i.e., 0.98, 0.97, 0.77, 0.68, and 0.98) for each purpose. Further, we run our model on 40,999 unique pieces of network traffic with *l-data item* detected from 5,102 (source of application data), and manually verify the prediction of 1,000 randomly selected samples, which achieves 94.3% accuracy.

Discussion. To rank the results of *Lalaine* based on their validity, for a sample x_i, *Lalaine* will output a detection result along with a confidence score p_i. In our study, we mainly consider the potential errors introduced by *Lalaine*'s purpose inference module, and calculate $p_i = \max_k F_k(t_i)$, where $F_k(t_i)$ is the softmax probability of predicting an input of purpose

inference module t_i as a purpose k. This is because the machine learning based purpose inference module is a source of errors while the pattern-based *l-data item* inference module yields a unified error of 0.2% for all of its outputs. Our study showed that 85.7% of violations flagged by *Lalaine* have confidence scores larger than 90%, while the average confidence score is 92.4%. We ranked potential violations based on their confidence scores, and selected top-100 flagged apps with highest confidence scores for manually validation, yielding an accuracy of 100%.

5.4 Evaluation of the Entire *Lalaine* System

Experiment settings. We ran *Lalaine* on four iPhones (iOS versions: 12.4.1, 13.4, 13.7, 14.8.1), two Mac minis with Apple M1 chip with 8-core/16-core CPUs, and two Macbook Pros with Apple M1 chip and 16-core CPU.

Evaluation results. To evaluate the overall effectiveness of *Lalaine*, we first collect ground truths. We randomly selected 100 apps from 6,332 apps sampled in § 5.3.1, manually interacted with those apps, and went through all possible UIs. Meanwhile, network traffic and call-stack traces are recorded by *fiddler* and *Frida*. In total, 81 apps out of 100 apps were successfully tested, generating 939 system API invocations and 1,362 network requests with responses. Further, we leverage domain experts to manually inspect the generated network traffic and stack traces to summarize the data and their corresponding purposes, then, and identified 122 non-compliant cases, denoted as (d,q) pairs where d is the data and q is the disclosed purpose in its privacy label, associated with 64 apps.

On the ground-truth dataset, *Lalaine* generated 407 system API invocations and 831 network requests with responses. *Lalaine* reported 75 non-compliant $(data, purpose)$ pairs (49 apps), showing a precision of 96% and a recall of 60.6% (for non-compliant data/purposes pairs), or a precision of 93.87% and a recall of 76.5% (for non-compliant apps). On average, it took 185 seconds (180 seconds for executing an app and 5 seconds for inconsistency analysis) to investigate one app.

• *Falsely detected inconsistency.* The three false positives, i.e., three reported (d, q) pairs, come from falsely inferred purposes. We found it challenging to distinguish product personalization from app functionality as they usually shared some similar features (e.g., sent data to its own domain). We elaborate on the falsely detected cases in [30]. We discuss all possible causes of false positives in § 5.5.

• *False negatives.* We observed two causes for the missed cases, both due to the limitation of off-the-shelf UI automation tool: (1) compared with manually/fully executing apps, *Lalaine* can only invoke 43.34% of system APIs and generate 61.01% of network traffic (see component evaluation in § 5.3.1); (2) many apps requires login (username/password, two-factor authentication, CAPTCHA) to be fully executed where *Lalaine* falls short. We discuss all possible causes of false negatives in § 5.5.

• *Apps in the omitted group.* Apps omitted in *Lalaine*'s sampling are those that do not involve iOS system APIs accessing *l-data item* items and their privacy policy is consistent with privacy label, denote as a set $\neg(API \cup PP)$ (172,365 apps in our dataset). Although these apps might have smaller likelihood of privacy label issues than other groups (§ 5.3.1), we randomly sampled 400 apps and performed a full testing using *Lalaine*. We found 9.75% of the apps have non-compliant privacy labels (compared to 72.26%, 66.46%, 69.45%, in the other sets $API \setminus PP$, $API \cap PP$, $PP \setminus API$ respectively), among which 80% are neglected disclosure. We observed that most leaked data are *User ID* and *Device ID* which are generated by app-level code including third-party libraries. The others originated from user input through the graphical interface (e.g., Phone Number).

5.5 Limitation

False negatives can arise for a few reasons. First, *Lalaine* is built on a set of off-the-shelf dynamic analysis tools (e.g., Frida in jailbroken environments, Fiddler for network inspection), and inheriting their limitations. *Lalaine* could not fully run for a portion of apps (1,230/6,332, see § 6) that (1) use SSL Certificate Pinning [23] to hinder traffic monitoring using Fiddler, (2) apply *"Block Frida Toolkits"* [5] to prohibit code instrumentation, (3) prevent themselves from running on jail-broken phones. Also, *Lalaine*'s automatic app interaction based on depth-first search may not traverse all UI paths, due to (1) configuration (3 minutes per app), (2) sophisticated user inputs (e.g., login) needed. Also, in traffic analysis, *Lalaine* may omit cases if the apps use customized encryption/obfuscation/encoding before data transfer. Further, sophisticated obfuscation to hide invocation of public system APIs can impact Lalaine's coverage. However, prevalence of obfuscation in iOS apps was low (0-8.17% in [96, 98]); prior obfuscation focuses on application-level code [68, 79, 98, 99] (e.g., function names of app code or third-party libraries) and private APIs [56]. Hence, we consider the current state of iOS app obfuscation to pose limited impact on *Lalaine*. Last, the purpose prediction model may introduce false negatives.

Despite multiple sources of false negatives,*Lalaine* includes multiple strategies to help ensure that the issues reported are most likely true and thus warrant developer investigation/action (93.87% precision, see § 5.4). Particularly, (1) at the *Lalaine* pipeline, we manually validated results of several components, ensuring the correctness of l-mapping and data ontology before they are used; (2) *Lalaine* tuned the hyperparameters of the purpose prediction model towards high precision; (3) *Lalaine* utilized a strict value/pattern match to find data items from network traffic instead of an ML-based method (99.8% precision, see § 5.3.2). Nonetheless, false positives can still occur for a few reasons. Above all is the challenge with inference of data-collection purposes. For example, it can be difficult to completely differentiate "An-

alytics" from "Developer's Advertising or Marketing", and we leveraged a set of features (e.g., SDK, endpoints, server responses) and ground truths (§ 5.3.2) to best infer purposes. In this process, failure of feature extraction could impact performance; for example, c-mapping enabled *Lalaine* to map 83% of call traces to traffic (see § 5.3.1), with missing cases caused by (1) obfuscated caller class/method names (e.g., QxxcvxMdXVGIxX:xfxxMmJcqGxlxJ); (2) uncommon or deprecated callers not found in known libraries. Last, the value match in traffic can lead to false cases if the values are too short (the coarse location if with just four digits may match other data types such as version number).

6 Inconsistency Results and Analysis

Running *Lalaine* on 6,332 iOS apps and their privacy labels, we show that the inconsistencies are prevalent. Altogether, 5,102 iOS apps have been fully tested, while the other apps crash due to their resistance to run in jail-broken environment, Frida hooking or traffic monitoring. Among the tested apps, *Lalaine* reveals that 3,281 of them neglect to disclose data and purposes (§ 6.1), 1,628 apps contrarily specify purposes (§ 6.2), and 677 apps inadequately disclose purposes (§ 6.3).

6.1 Neglect Disclosure

Neglect Disclosure indicates that the app developer collects certain data without disclosure. Among 5,102 apps, we found 3,281 apps with neglect disclosure. In total, 11,726 data objects are neglected in the privacy label of these apps. Among these non-compliant apps, 2,346 apps lie in $API \setminus PP$ (the data disclosure in the privacy label is consistent with that in the privacy policy), indicating that both the privacy label and privacy policy may neglect the data in app code behavior. Also, 238 non-compliant apps lie in $API \cap PP$, where the privacy policy is reliable while the privacy label fails to reflect the code behavior. Besides, 434 non-compliant apps lie in $PP \setminus API$, indicating that they harvest data from user inputs, generated by app-level code, or bypassed static screening.

• *Data types and purposes.* To understand the disclosure of what kind of data with which purposes are prone to be ignored by app developers, we plot the number of apps neglecting data under each purpose, as shown in Figure 2a. The results show that *Diagnostic data*, such as device metadata (e.g., *is_jailbroken, localized_Model, GPU_type, sensors_signal, screen_size, os_version, etc*), is the most likely to be omitted. Although such data are usually used for technical diagnostics, they can be monetized by the data broker. For example, the traffic sent to a data broker *broker.datazoom.io* consists of 12 device-related data along with user events in the request body, which can be used to profile a user. Further, the *User ID*, *Device ID* and *Location* data are also prone to be omitted by developers. Those data are commonly used for third-party advertising, analytics, or marketing. In our study, we also observe data that come from user input, like *Email Address, Phone Number* to support the App Functionality purpose, are commonly omitted by app developers.

• *Endpoints of omitted data.* To examine the endpoint that the data are leaked to, we present a stacked histogram of the top 15 endpoints, ranked by the frequencies of collecting each targeted data, as shown in Figure 4. The top three endpoints (googleapis.com, doubleclick.net, graph.facebook.com) all belong to third-party libraries. This indicates that the app developers are generally unclear about the data practice of third-party partners (§ 7).

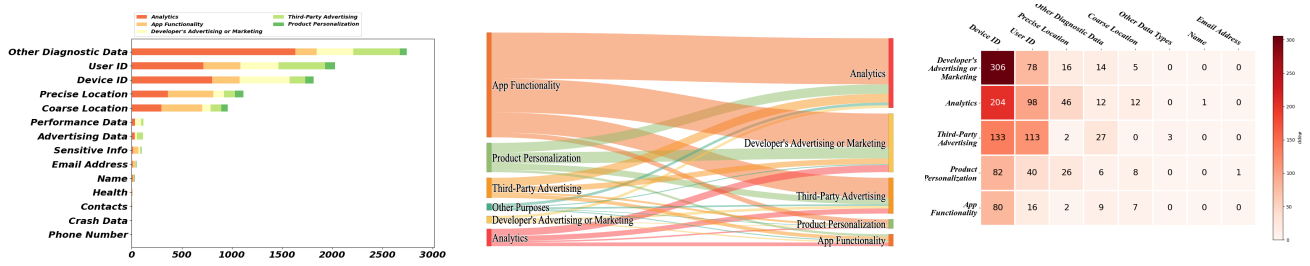
• *App categories.* To examine which app categories are most likely to neglect data disclosure, we investigate the distribution of non-compliant apps according to their app categories. As shown in Figure 5, the game apps are most prone to neglect disclosure. We observe 97.15% of omitted data are leaked to third-parties in game apps. We also found youth-using education apps with neglect disclosure, especially with the omitted data types of Device ID, User ID, and Precise Location. The results aligned with prior work that examined similar privacy issues in youth-targeted Android apps [60, 91]. In our study, we observe that 81.41% of omitted data are leaked to third-parties in education apps.

6.2 Contrary Disclosure

Lalaine reports 1,628 apps incorrectly labeling the purposes of 2,935 data objects. Among those apps, we found 973, 202, and 314 from $API \setminus PP$, $API \cap PP$, and $PP \setminus API$ respectively.

• *Data types and purposes.* The app developers are most likely to falsely disclose the other four purposes to *App Functionality*. In our study, we observe that 347 app developers distrustfully disclose purpose as *App Functionality*, while the data is actually used for advertising. Such incorrect disclosure may intentionally deceive users to download and use their app by claiming they have a more acceptable reason to collect user data. As reported in [52], users were willing to share their location data to help cities plan bus routes or to get traffic information (*App Functionality*), but reluctant to share the same data for ads (*Advertising*) or maps showing one's travel patterns (*Analytics*). As to data types, similar to neglect disclosure, we found the data items most commonly to be incorrectly disclosed are Device ID, User ID, and Coarse Location.

• *App categories.* By examining the app category with the most contrary disclosure, we found that lifestyle apps (related to fitness, dating, food, music, travel, etc) always falsely disclose the purpose of Developer's Advertising and Marketing as App functionality. This is probably due to the rich functionalities (e.g., finding the next new song, restaurant, or destination, etc) encompassed in such apps; also, the app developers may carelessly consider most data collection operation as a part of app functionality.



(a) (b) (c)

Figure 2: (a) The Distribution of neglected data apps. Each stack represents different data type. The colors on the spectrum show the actual purposes of the apps. (b) The flow between the claimed purposes in the privacy label and actual purposes. The sources on the left denote the purposes in privacy label. The sinks on the right represent the actual purposes of apps. The widths of the flow show the relative portions of apps with contrary disclosure. (c) The distribution of inadequate disclosure in different data types and purposes. Darker color shows the data and purpose more apps fail to adequately disclose in their privacy labels.

6.3 Inadequate Disclosure

Inadequate Disclosure means the developer has already declared one or multiple purposes for a specific data item but fails to disclose all of them. The scanning results of *Lalaine* show that 677 apps inadequately label the purposes of 1347 data objects, which indicated that developers intend to inform users of data usage but may lack of capability to disclose data practices in the app comprehensively. Among those non-compliant apps, we found 402, 74, and 133 from $API \setminus PP$, $API \cap PP$, and $PP \setminus API$ respectively. As shown in Figure 2c, the most prevailing inadequate purposes are *Developer's Advertising or Marketing*, *Analytics*, followed by *Third-party Advertising*, ranked by numbers of unique apps with inadequate disclosure. App Functionality and Product Personalization are less prone to be inadequately disclosed. The results indicate that developers are more familiar with data practices in their own code and less clear about data collection from other vendors integrated into the same app. As to data types, similar to neglect disclosure and contrary disclosure, we found the most common data items to be inadequately disclosed are *Device ID, User ID, and Precise Location.* As to the distribution of app categories, detailed in [30], we observed similar trend with Neglect Disclosure.

6.4 Characterizing Severity of Violations

We break down all reported inconsistencies based on the data types (see details in [30]). To better characterize severity of each, we further categorizes *l-data item*s into three groups: Sensitive PII, PII, non-PII (based on definition by Department of Homeland Security [16]), with each group's relative severity upon a privacy-label violation ranging from high to low. Notably, by inspecting 5,102 apps, each of the 14 *l-data item*s we studied observed inconsistencies. The top five *l-data item*s associated with most number of non-compliant apps are Other Diagnostic Data, User ID, Device ID, Precise Location, and Coarse Location. As discussed in § 7.1, a key cause is that Apple's definition of *many l-data item*s are too general and vague. For example, 376 apps and some third-party SDKs (e.g., iron-Source [28]) may not have considered IDFV as a device ID; it may not be clear what data are Sensitive Info, or Other Diagnostic Data (see § 7.1); 140 apps incorrectly disclose the collection of precise location as coarse location.Further, under each data type, we consider the inconsistency types Neglect Disclosure, Contrary Disclosure and Inadequate Disclosure can have a relative severity from high to low.

7 Root Cause Analysis with Case Study

While an important cause of privacy label non-compliance can be app developers' neglect, in this section, we also analyze other possible, major causes with empirical studies.

7.1 Lack of Specification by Apple

Apple's ambiguous privacy label requirements can be a major cause. For some of Apple's 32 general *l-data item*s, it can be very confusing whether or not a specific data *is a* certain *l-data item*, for example, *Other Diagnostic Data*, *Sensitive Info*, or *Performance Data*. To mitigate the problem, by analyzing Apple's own apps and high-profile apps (259 apps in ground-truth set § 5.3.2), our ontology (§ 5.2.2) and specific data types (e.g., device_dimension for *Other Diagnostic Data*) are released online [30]. For the vague *l-data item*s *Sensitive Info* and *Performance Data*, Apple provided just examples, although ideally one may favor a comprehensive list. For the *l-data item Health*, Apple indicates [47] that iOS APIs in three frameworks (Clinical Health Records, HealthKit, Movement Disorder) can return *Health* data, without a specific/comprehensive list of the APIs. For such *l-data item*s, non-compliance reported by *Lalaine* can be just a lower bound. Further, we break down all non-compliance found by *Lalaine* based on the *l-data item*s (released online [30]).

7.2 Misleading Third-Party Disclosure Guides

To help app developers fill out privacy label properly, some third-party SDKs released guidelines (dubbed "privacy label guidelines") about what privacy label items the app developers need to disclose should their apps integrate the SDKs

(the SDK may collect the data with/without the app developers' awareness). However, our study shows that incomplete or incorrect guidelines are common from popular SDKs and may easily lead to non-compliant privacy labels for iOS apps. Notably, our findings complements the recent human subject study [77] which showed that iOS app developers were not aware of the guidelines provided by third-party SDKs.

Cause attribution. Since both the app developers and SDK providers may share part of disclosure responsibility for data collection by SDKs, *a challenge* is how to attribute a noncompliance to right parties, based on which we may generalize improved practices and responsibility models for privacy label guidance. The key is to understand whether app developers (1) are aware of the data collection and (2) have the option/-configurations to control (enable/disable) SDK data practices.

18 SDKs out of the 30 most popular SDKs in our dataset (2,679 iOS apps) offered privacy label guideline. We examined their data collections (what data under which configurations were collected, known by building sample apps using those SDKs), and the configurations they offered to app developers. We found that the SDKs come with three kinds of configuration practices: 1) The SDKs compulsively collect *l-data item*s and the app cannot disable the collection. 2) The SDKs by default collect the data although the app developers can disable the collection through configurations. 3) The SDKs do not collect the data by default unless app developers explicitly configure them to. For each practice, see real examples in Appendix § A.2.

For practice 1 and 2, the SDK vendors' privacy-label guidance is designed to properly disclose data collection behaviors in SDK [8, 17, 35]. For practice 3, we consider that app developers are easily aware of the SDK's collection; still the guidance should be precise and not deny the collections. We found that the guidance of 11 SDKs (out of 18) are either inadequately (N=3) or incorrectly (N=7) disclosing data collection practices, or using vague statements (N=1). For example, we observe that *Flurry Analytics SDK* collected precise location data by default from 76 non-compliant apps. However, *Flurry* didn't suggest app developers declare the collection of user location in its guide [19]. Although the collection of location data can be disabled by developers, collecting such by default without noticeable statements is stealthy, which further would cause apps inaccurately disclose privacy labels. We provide additional examples in § A.4, and list the SDKs with their practices (1-3) in [30] , including the number of non-compliant apps related to these SDKs.

7.3 Abusing Apple's Requirement

For apps with web views [43], Apple required that *"Data collected via web traffic must be declared, unless you are enabling the user to navigate the open web"* [47]. That is, data collection inside app web-views must be disclosed, while if the app redirects users to open browsers for data collections, app developers are not required to disclose it. In our study, we observed that some apps exploit this disclosure requirement, to collect private user data while likely evading the privacy-label disclosure requirement. For example, the privacy label of the financial app *Banco ABC Brasil Personal* [1] only discloses collection of *Identifiers* and *Diagnostics*. However, once users open this app, it redirects users to the browser along with the users' precise geo-coordinates appended to the URL, sending to its server *abrasuaconta.abcbrasil.com.br*. More seriously, the webpage further sent the precise location data to 50 unique URLs belonging to 9 different domains including *px.ads.linkedin.com, trc.taboola.com, google-analytics.com*. The similar finding of the CVS Pharmacy app was reported in [13] on Android in 2017. In our study, we found 169 apps passed 1,148 privacy-sensitive data items (e.g., *Deivce ID, Location data, Diagnostics data*) through browsers to 124 unique endpoints, leading to plausible bypassing of privacy-label disclosure requirement based on Apple's current definition (see technical approach in Appendix § A.3).

7.4 Diverse, Opaque Third-Party Partners

Mobile apps extensively incorporate third-party services (e.g., analytics, advertising, app monetization, or single-sign-on SDK), whose data collection is required to be disclosed in privacy labels by the app developers [47]. The recent human-subject study [77] showed that developers could have limited awareness of third-party libraries' data use, likely leading to non-compliant privacy labels. In our study, to understand the non-compliance indeed caused by opaque third-party data collection in the wild, we characterize the non-compliance with respect to different kinds of third-party service, which can serve as a finer-grained analysis to complement prior works. Specifically, for apps we found with non-compliant privacy labels (§ 6), we re-examine the app traffic and extract domain names associated with data transmission to third-party partners. Next, we look up the domain owners using domain WHOIS [45] and Crunchbase [12]. Further, we use the corporate categories in Crunchbase [12] to classify those third-party partners into three categories: *data broker*, *service provider*, and *advertising, analytics, and marketing*. We characterize the nuances of their non-compliance with different third-party partners, which can help platform owners and policy-makers pinpoint issues with third-parties and improve regulations. See details at Appendix § A.5.

7.5 Recommendations for Stakeholders

Based on the causes above, we give recommendations for multiple stakeholders to improve privacy label compliance.

• *Apple.* Apple with the community should provide (1) a comprehensive ontology of sensitive data items tailored to the iOS ecosystem instead of general definitions of *l-data item*s, (2) a mapping between sensitive data items and corresponding system APIs (even with a ontology, it is still non-trivial for app developers to correctly correlate them to the huge amount

of system APIs). With the latter, developers can easily know the *l-data item* to disclose if specific APIs are used.

• *SDK vendors.* We recommend SDK vendors to improve data transparency and guidance accuracy for privacy labels. Specifically, the guidance should clearly and accurately disclose the three kinds of practices: what data are collected (1) compulsively, (2) by default with option to disable, (3) unless explicit configuration by apps. SDK vendors' guidance should avoid confusing description such as "may collect" (§ A.4).

• *App developers.* App developers should scrutinize SDKs adopted to understand their data practices, their privacy label guidance if available, best understand Apple's requirements and other public policies, and if possible, adopt tools [66] or analysis reports such as *Lalaine*'s.

8 Conclusion

We developed *Lalaine* by adapting a set of innovative techniques including automatic app UI execution, NLP, and dynamic and static binary analysis. Running on 5,102 iOS apps, *Lalaine* found 3,423 non-compliant ones. Our research further uncovers a set of root causes including opaque data collection from diverse third-party partners and misleading privacy label disclosure guidance. Our study brings new insights into proper privacy label design, implementation, and regulation, essential to improve privacy compliance assurance.

Acknowledgement

We thank the undergraduate students Jack Ruocco, Sultan Aloufi, Keegan Allred, and Zixiao Pan, and graduate students Feifan Wu, Chaoqi Zhang at Indiana University (IU) for their efforts in data collection and annotation and cases studies. IU authors are supported by IU faculty startup funding, Indiana University's IAS Collaborative Research Award, and graduate teaching assistantships at IU's department of computer science. This works is not possible without support of IU.

References

[1] App. https://apps.apple.com/us/app/banco-abc-brasil-personal/id1363240193.

[2] App. https://apps.apple.com/us/app/Centre%20Laser%20Pro/id1551154071.

[3] Apple's app 'privacy labels' are here—and they're a big step forward. https://www.wired.com/story/apple-app-privacy-labels/.

[4] appmakerRank. https://influencermarketinghub.com/12-best-app-makers/#:~:text=Two%20of%20the%20best%20app,it%20at%20absolutely%20no%20cost.

[5] block frida toolkits. https://www.appdome.com/how-to/mobile-fraud-prevention/prevent-android-ios-malware/block-frida-toolkits/.

[6] Branch API. https://help.branch.io/developers-hub/docs/attribution-api.

[7] Branch api documentation. https://help.branch.io/developers-hub/reference/attribution-api-1#post-body-parameters.

[8] Branch SDK guidance. https://help.branch.io/using-branch/docs/answering-the-app-store-connect-privacy-questions.

[9] CCPA. https://oag.ca.gov/privacy/ccpa.

[10] cocoapods. https://cocoapods.org/.

[11] Concerns of Privacy Label Accuracy. https://energycommerce.house.gov/sites/democrats.energycommerce.house.gov/files/documents/Apple%20Letter%20re%20App%20Privacy%20Label%202-2021.pdf.

[12] Crunchbase. https://www.crunchbase.com/.

[13] CVS Discretely Shares Your Location with 40+ Other Sites. https://blog.appcensus.io/2017/08/25/cvs-discretely-shares-your-location-with-40-other-sites/.

[14] Data broker. https://oag.ca.gov/data-brokers.

[15] Device Installer. https://github.com/libimobiledevice/ideviceinstaller.

[16] DHS Definition of data. https://www.dhs.gov/privacy-training/what-personally-identifiable-information.

[17] Facebook SDK guidance. https://developers.facebook.com/blog/post/2020/10/22/preparing-for-apple-app-store-data-disclosure-requirements/.

[18] Flurry API. https://developer.yahoo.com/flurry/docs/analytics/gettingstarted/technicalquickstart/ios/.

[19] Flurry SDK Guidance. https://www.flurry.com/blog/flurry-guidance-apple-app-privacy-details/.

[20] Frida API. https://frida.re/docs/javascript-api/#script.

[21] Google Ads. https://ads.google.com/.

[22] Grep. https://linuxhint.com/grep_command_linux/.

[23] Identity Pinning. https://developer.apple.com/news/?id=g9ejcf8y.

[24] IDFA. https://developer.apple.com/documentation/adsupport/asidentifiermanager/1614151-advertisingidentifier?language=objc.

[25] IDFV. https://developer.apple.com/documentation/uikit/uidevice/1620059-identifierforvendor?language=objc.

[26] IDFV Definition. https://developer.apple.com/app-store/user-privacy-and-data-use/.

[27] Inmobi. https://www.inmobi.com/.

[28] Ironsource SDK. https://www.is.com/.

[29] KochavaTrackeriOSGK SDK. https://github.com/CocoaPods/Specs/blob/master/Specs/8/e/1/KochavaTrackeriOSGK/3.1.4/KochavaTrackeriOSGK.podspec.json.

[30] Lalaine. https://sites.google.com/view/privacylabel/home.

[31] Many apple's new privacy nutrition labels were false. https://www.washingtonpost.com/technology/2021/01/29/apple-privacy-nutrition-label/.

[32] Mobile-App Privacy Nutrition Labels Missing Key Ingredients for Success. https://cacm.acm.org/magazines/2022/11/265814-mobile-app-privacy-nutrition-labels-missing-key-ingredients-for-success/fulltext.

[33] Network traffic monitor. https://www.telerik.com/fiddler.

[34] Object C. https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html.

[35] One Signal SDK guidance. https://documentation.onesignal.com/docs/apple-app-privacy-requirements.

[36] P3P. https://www.w3.org/TR/P3P/.

[37] Platform for Privacy Preferences Project. https://www.w3.org/P3P/.

[38] Singular API. https://support.singular.net/hc/en-us/articles/8586543222683-Singular-SDK-Integration-for-Cordova.

[39] Swift. https://developer.apple.com/swift/.

[40] TED. https://apps.apple.com/us/app/TED/id376183339.

[41] UI automation tool. https://github.com/macacajs/NoSmoke.

[42] Unclear iOS API documentation. https://apple.slashdot.org/story/19/10/28/173216/apple-your-developer-documentation-is-garbage.

[43] Webview. https://developer.apple.com/documentation/webkit/wkwebview?language=objc.

[44] What we learned from apple's new privacy labels. https://www.nytimes.com/2021/01/27/technology/personaltech/apple-privacy-labels.html.

[45] WHOIS. https://www.whois.com/.

[46] Why some like apple's new privacy labels, despite their flaws. vox.com/recode/22285020/apple-privacy-nutrition-labels-ios-14.

[47] App privacy details on the App Store, 2021. https://developer.apple.com/app-store/app-privacy-details/.

[48] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. Policylint: investigating internal privacy policy contradictions on google play. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 585–602, 2019.

[49] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. PolicyLint: Investigating internal privacy policy contradictions on google play. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 585–602, Santa Clara, CA, August 2019. USENIX Association.

[50] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. Actions speak louder than words:{Entity-Sensitive} privacy policy and data flow analysis with {PoliCheck}. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 985–1002, 2020.

[51] Travis D Breaux and Ashwini Rao. Formal analysis of privacy requirements specifications for multi-tier applications. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 14–23. IEEE, 2013.

[52] AJ Bernheim Brush, John Krumm, and James Scott. Exploring end user preferences for location obfuscation, location-based services, and the value of location. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, pages 95–104, 2010.

[53] Duc Bui, Yuan Yao, Kang G Shin, Jong-Min Choi, and Junbum Shin. Consistency analysis of data-usage purposes in mobile apps. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2824–2843, 2021.

[54] Yi Chen, Mingming Zha, Nan Zhang, Dandan Xu, Qianqian Zhao, Xuan Feng, Kan Yuan, Fnu Suya, Yuan Tian, and Kai Chen. Demystifying hidden privacy settings in mobile apps. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 570–586. IEEE, 2019.

[55] Lorrie Faith Cranor, Serge Egelman, Steve Sheng, Aleecia M McDonald, and Abdur Chowdhury. P3p deployment on websites. *Electronic Commerce Research and Applications*, 7(3):274–293, 2008.

[56] Zhui Deng, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu. iris: Vetting private api abuse in ios applications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 44–56, 2015.

[57] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Pios: Detecting privacy leaks in ios applications. In *NDSS*, pages 177–183, 2011.

[58] Serge Egelman. Taking responsibility for someone else's code: Studying the privacy behaviors of mobile apps at scale. In *2020 {USENIX} Conference on Privacy Engineering Practice and Respect ({PEPR} 20)*, 2020.

[59] Serge Egelman, Lorrie Faith Cranor, and Abdur Chowdhury. An analysis of p3p-enabled web sites among top-20 search results. In *Proceedings of the 8th international conference on Electronic commerce: The new e-commerce: innovations for conquering current barriers, obstacles and limitations to conducting successful business on the internet*, pages 197–207, 2006.

[60] Anirudh Ekambaranathan, Jun Zhao, and Max 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*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[61] Pardis Emami-Naeini, Yuvraj Agarwal, Lorrie Faith Cranor, and Hanan Hibshi. Ask the experts: What should be on an iot privacy and security label? In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 447–464. IEEE, 2020.

[62] Pardis Emami-Naeini, Henry Dixon, Yuvraj Agarwal, and Lorrie Faith Cranor. Exploring how privacy and security factor into iot device purchase behavior. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[63] Farnood Faghihi, Mohammad Zulkernine, and Steven Ding. Camodroid: An android application analysis environment resilient against sandbox evasion. *Journal of Systems Architecture*, 125:102452, 2022.

[64] Johannes Feichtner, David Missmann, and Raphael Spreitzer. Automated binary analysis on ios: A case study on cryptographic misuse in ios applications. In *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 236–247, 2018.

[65] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[66] Jack Gardner, Yuanyuan Feng, Kayla Reiman, Zhi Lin, Akshath Jain, and Norman Sadeh. Helping mobile application developers create accurate privacy labels. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 212–230. IEEE, 2022.

[67] Julia Gideon, Lorrie Cranor, Serge Egelman, and Alessandro Acquisti. Power strips, prophylactics, and privacy, oh my! In *Proceedings of the Second Symposium on Usable privacy and security*, pages 133–144, 2006.

[68] Michael C Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pages 101–112, 2012.

[69] Jin Han, Qiang Yan, Debin Gao, Jianying Zhou, and Robert H Deng. Comparing mobile privacy protection through cross-platform applications. 2013.

[70] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I Hong. Why are they collecting my data? inferring the purposes of network traffic in mobile apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):1–27, 2018.

[71] Patrick Gage Kelley, Joanna Bresee, Lorrie Faith Cranor, and Robert W Reeder. A" nutrition label" for privacy. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, pages 1–12, 2009.

[72] Patrick Gage Kelley, Lucian Cesca, Joanna Bresee, and Lorrie Faith Cranor. Standardizing privacy notices: an online study of the nutrition label approach. In *Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 1573–1582, 2010.

[73] Patrick Gage Kelley, Lorrie Faith Cranor, and Norman Sadeh. Privacy as part of the app decision-making process. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 3393–3402, 2013.

[74] Simon Koch, Malte Wessels, Benjamin Altpeter, Madita Olvermann, and Martin Johns. Keeping privacy labels honest. *Proceedings on Privacy Enhancing Technologies*, 4:486–506, 2022.

[75] Arash Habibi Lashkari, Andi Fitriah A Kadir, Laya Taheri, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *2018 International Carnahan Conference on Security Technology (ICCST)*, pages 1–7. IEEE, 2018.

[76] Kyungmin Lee, Jason Flinn, Thomas J Giuli, Brian Noble, and Christopher Peplin. Amc: Verifying user interface properties for vehicular applications. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 1–12,

2013.

[77] Tianshi Li, Kayla Reiman, Yuvraj Agarwal, Lorrie Faith Cranor, and Jason I Hong. Understanding challenges for developers to create accurate privacy nutrition labels. In *CHI Conference on Human Factors in Computing Systems*, pages 1–24, 2022.

[78] Yucheng Li, Deyuan Chen, Tianshi Li, Yuvraj Agarwal, Lorrie Faith Cranor, and Jason I Hong. Understanding ios privacy nutrition labels: An exploratory large-scale analysis of app store data. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–7, 2022.

[79] Bin Liu, Bin Liu, Hongxia Jin, and Ramesh Govindan. Efficient privilege de-escalation for ad libraries in mobile apps. In *Proceedings of the 13th annual international conference on mobile systems, applications, and services*, pages 89–103, 2015.

[80] Bin Liu, Suman Nath, Ramesh Govindan, and Jie Liu. {DECAF}: Detecting and characterizing ad fraud in mobile apps. In *11th USENIX symposium on networked systems design and implementation (NSDI 14)*, pages 57–70, 2014.

[81] Yuhong Nan, Min Yang, Zhemin Yang, Shunfan Zhou, Guofei Gu, and XiaoFeng Wang. Uipicker: User-input privacy identification in mobile applications. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 993–1008, 2015.

[82] Yuhong Nan, Zhemin Yang, Xiaofeng Wang, Yuan Zhang, Donglai Zhu, and Min Yang. Finding clues for your secrets: Semantics-driven, learning-based privacy discovery in mobile apps. In *NDSS*, 2018.

[83] Suman Nath, Felix Xiaozhu Lin, Lenin Ravindranath, and Jitendra Padhye. Smartads: bringing contextual ads to mobile apps. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 111–124, 2013.

[84] Marten Oltrogge, Erik Derr, Christian Stransky, Yasemin Acar, Sascha Fahl, Christian Rossow, Giancarlo Pellegrino, Sven Bugiel, and Michael Backes. The rise of the citizen developer: Assessing the security impact of online app generators. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 634–647. IEEE, 2018.

[85] Damilola Orikogbo, Matthias Büchler, and Manuel Egele. Crios: Toward large-scale ios application analysis. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 33–42, 2016.

[86] Abbas Razaghpanah, Rishab Nithyanand, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Mark Allman, Christian Kreibich, Phillipa 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.

[87] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 603–620, 2019.

[88] Robert W Reeder. *Expandable Grids: A user interface visualization technique and a policy semantics to support fast, accurate security and privacy policy authoring*. PhD thesis, Carnegie Mellon University, 2008.

[89] Jingjing Ren, Martina Lindorfer, Daniel J Dubois, Ashwin Rao, David Choffnes, and Narseo Vallina-Rodriguez. A longitudinal study of pii leaks across android app versions. In *Proceedings of the 25th Network and Distributed System Security Symposium (NDSS 2018)*, 2018.

[90] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 361–374, 2016.

[91] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, Serge Egelman, et al. "won't somebody think of the children?" examining coppa compliance at scale. In *The 18th Privacy Enhancing Technologies Symposium (PETS 2018)*, 2018.

[92] Julian Schütte and Dennis Titze. lios: Lifting ios apps for fun and profit. In *2019 International Workshop on Secure Internet of Things (SIOT)*, pages 1–10, 2019.

[93] Rocky Slavin, Xiaoyin Wang, Mitra Bokaei Hosseini, James Hester, Ram Krishnan, Jaspreet Bhatia, Travis D Breaux, and Jianwei Niu. Toward a framework for detecting privacy policy violations in android application code. In *Proceedings of the 38th International Conference on Software Engineering*, pages 25–36, 2016.

[94] Ray Smith. An overview of the tesseract ocr engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, volume 2, pages 629–633. IEEE, 2007.

[95] Benfano Soewito and Agung Suwandaru. Android sensitive data leakage prevention with rooting detection using java function hooking. *Journal of King Saud University - Computer and Information Sciences*, 34(5):1950–1957, 2022.

[96] Zhushou Tang, Ke Tang, Minhui Xue, Yuan Tian, Sen Chen, Muhammad Ikram, Tielei Wang, and Haojin Zhu. {iOS}, your {OS}, everybody's {OS}: Vetting and analyzing network services of {iOS} applications. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2415–2432, 2020.

[97] Jice Wang, Yue Xiao, Xueqiang Wang, Yuhong Nan, Luyi Xing, Xiaojing Liao, JinWei Dong, Nicolas Serrano, Haoran Lu, XiaoFeng Wang, et al. Understanding malicious cross-library data harvesting on android. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 4133–4150, 2021.

[98] Pei Wang, Qinkun Bao, Li Wang, Shuai Wang, Zhaofeng Chen, Tao Wei, and Dinghao Wu. Software protection on the go: A large-scale empirical study on mobile app obfuscation. In *Proceedings of the 40th International Conference on Software Engineering*, pages 26–36, 2018.

[99] Pei Wang, Dinghao Wu, Zhaofeng Chen, and Tao Wei. Protecting million-user ios apps with obfuscation: motivations, pitfalls, and experience. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 235–244, 2018.

[100] Xiaoyin Wang, Xue Qin, Mitra Bokaei Hosseini, Rocky Slavin, Travis D Breaux, and Jianwei Niu. Guileak: Tracing privacy policy claims on user input data for android applications. In *Proceedings of the 40th International Conference on Software Engineering*, pages 37–47, 2018.

[101] Le Yu, Xiapu Luo, Xule Liu, and Tao Zhang. Can we trust the privacy policies of android apps? In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 538–549. IEEE, 2016.

[102] Shikun Zhang, Yuanyuan Feng, Yaxing Yao, Lorrie Faith Cranor, and Norman Sadeh. How usable are ios app privacy labels? *UMBC Faculty Collection*, 2022.

[103] Sebastian Zimmeck, Peter Story, Daniel Smullen, Abhilasha Ravichander, Ziqi Wang, Joel R Reidenberg, N Cameron Russell, and Norman Sadeh. Maps: Scaling privacy compliance analysis to a million apps. *Proc. Priv. Enhancing Tech.*, 2019:66, 2019.

[104] Sebastian Zimmeck, Ziqi Wang, Lieyong Zou, Roger Iyengar, Bin Liu, Florian Schaub, Shomir Wilson, Norman Sadeh, Steven Bellovin, and Joel Reidenberg. Automated analysis of privacy requirements for mobile apps. In *2016 AAAI Fall Symposium Series*, 2016.

Appendix A

A.1 IDFV Interpretation.

IDFA is a unique identifier assigned by Apple to a user's device that allows an app to track user behavior across other companies' apps [24]. The IDFV is more convoluted and essentially considered as a privacy-sensitive ID in our research since it enables cross-app user tracking: based on Apple, *"it is useful for analytics across apps from the same content provider and may not be combined with other data to track a user across apps and websites owned by other companies unless the app has been granted permission to track"* [26].

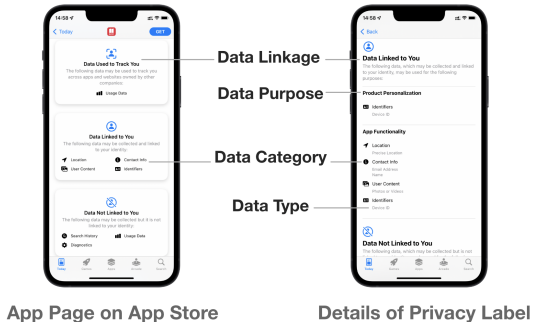


Figure 3: Privacy label views on App Store. The left is the privacy label card views from the app details page. On the right is the details of privacy label, which will appear after clicking any of the card views on the left page.

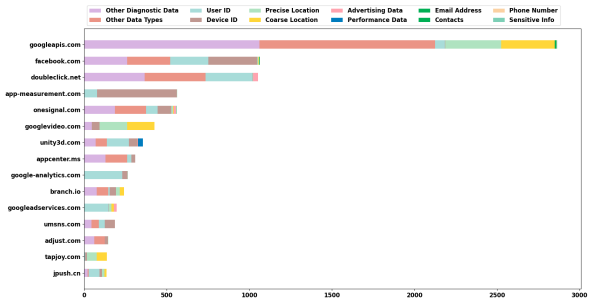


Figure 4: The top 15 endpoints that the omitted data are most likely to leak to. Each stack represents different endpoints. The colors on the spectrum show the types of data

A.2 SDK Configuration Practices

We found that the SDKs come with three kinds of configuration practices: 1) The SDKs compulsively collect *l-data items* and the app cannot disable the collection (e.g., the Branch SDK compulsively collects device IP addresses once apps get installed/launched/closed for deep linking and session attribution [6]). 2) The SDKs by default collect the data although the app developers can disable the collection through configurations (e.g., the Flurry SDK defaulted the collection of users' precise location, which can be disabled by configuring the API*[Flurry trackPreciseLocation:NO]* [18]). 3) The SDKs do not collect the data by default unless app developers explicitly configure them to (e.g., app developers can configure Singular SDK's Events and Attributes (e.g., *sng_attr_location, sng_attr_location_address_region_or_province, sng_attr_latitude, sng_attr_longitude*) [38] to send the users' coarse location to the Sentry back-end).

A.3 Detect apps abusing Apple's requirement

To understand how pervasive such abusive behaviors are, we use regex `(iPhone).*AppleWebKit (?!.*Version).*Safari` to match the *"userAgent"* in HTTP headers of app traffic, to find out traffic redirecting users to browsers (e.g., Safari). Note that the user agent strings of mobile Safari include the word "Version", whereas the user-agent strings of iOS web view (*uiWebView*) do not. Hence, the above regex will not include traffic in web views.

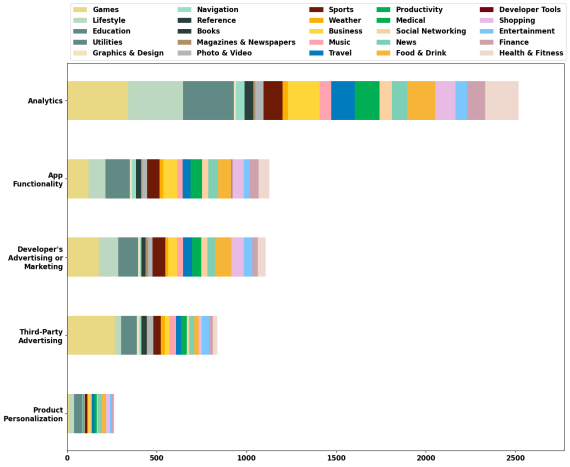


Figure 5: The distribution of iOS apps with neglected disclosure in their privacy labels. Each stack represents different actual purpose. The colors on the spectrum show the categories of the apps.

A.4 Three types of inappropriate Guides

• *Incomplete guidelines.* The incomplete guidelines mean the SDK providers did not disclose all the data and its configuration practices. For example, in our study, we observe that *Flurry Analytics SDK* collected precise location data by default from 76 apps. However, in its privacy-label guidance, it didn't suggest app developers declare the collection of user location (but only "Device ID", "Product Interaction", and "Other Usage Data") [19]. Although the collection of location data can be disabled by developers, collecting such by default without noticeable statements is stealthy, which further would cause apps inaccurately disclose privacy labels.

• *Incorrect guidelines.* In our study, we also found that the SDK providers falsely claimed their data collection practices. For example, in the guideline of Branch SDK [8], it claimed that location and Diagnostics data are not collected. However, we observed that the device's IP address [1] was compulsively sent to the endpoint *api2.branch.io*. Based on Apple's definition [47], the IP address, which can be traced back to the user's ISP, belongs to the precise or coarse location depending on the resolution; the IP address used to measure technical diagnostics belongs to diagnostic data. However, none of them are disclosed.

[1]The branch SDK required the app developers to provide the device's public IP address by configuring the request header as "-H "X-IP-Override: xx.xx.xx.xx (public IP) [7].

• *Vague statements.* The vague guidelines describe the data collection practices in an uncertain tone by using an auxiliary verb (e.g., may, could, might). For example, all the data "collect" actions from Facebook SDK guidance [17] are modified by auxiliary verb *may*, e.g., *"We may receive and process certain contact, location, identifier, and device information associated with Facebook users and their use of your application."* However, the identifier and coarse location are collected once the app developers integrate it, except app developers disable the configuration options. In our dataset, we found that over 90% traffic transmitted to the Facebook domain contain the identifier and coarse location data, indicating app developers probably even did not notice the existence of the "disable" options. Moreover, we indeed found that 694 iOS apps (out of 908), which integrated Facebook SDK, did not disclose the data collection practices of Facebook SDK.

A.5 Diverse, Opaque Third-Party partners

We characterize the nuances of apps non-compliance with different third-party partners they integrated as follows:

• *Service provider.* In our study, non-compliance of 854 apps is caused by opaque data collection by third-party service providers. Contrary to the previous common understanding which attributes privacy non-compliance to opaque third-party disclosure guides, we find that even when the third-party libraries declare the collection and usage of data, such information sometimes cannot be leveraged by app developers to create compliant privacy label. The top service provider associated with most (250) non-compliant apps is OneSignal, which helps apps implement push notifications and in-app messaging. As an instance, an app, called *IUOE 513*, providing general news, training, and course information for their members in the union, utilizes the OneSignal SDK to increase user stickiness. However, some sensitive information (e.g., social_security_number, employee_ID, job_site_location) are sent to *api.onesignal.com* without disclosure in the app's privacy label. We found that OneSignal actually provides configurable APIs *OneSignal.sendTag("key", value: "value")* for app developer to share user information. That is, any information collected by OneSignal is explicitly provided by the app developers (by calling the API of OneSignal SDK), and thus known to the app developers. Despite the transparency, the app's privacy label missed the disclosure related to OneSignal.

• *Advertising, analytics, marketing and their affiliates.* Based on the definition in CCPA [9], the advertising, analytics, and marketing SDKs are entities that do not qualify as either data collectors or service providers that can obtain the personal information of a consumer from a business. The app developers usually integrate those SDKs for app monetization, user behavior measurement, scaling marketing campaigns, etc. In our study, we observed 2,086 non-compliant behaviors from those advertising and analytics SDKs in iOS apps. For example, the app *"CloudMall"*, providing coupons and price comparison, declared only to collect *Diagnostics data* for analytics purpose in its privacy label. However, we observed that it sent various user data to different advertising and analytics SDKs (e.g., IDFA/IDFV to Adjust SDK, User ID, and Email Address to Google Analytics SDK).

• *Data broker.* Data brokers are businesses that knowingly collect and sell the personal information of consumers with whom they do not have a direct relationship. A data broker, required by California law, needs to register with the Attorney General. A list of 469 registered data brokers [14] is provided by the State of California Department of Justice[2]. To check whether the endpoint is owned by a data broker, we first obtain the company name by using Crunchbase API and then look up the above list to determine if the data receiver is a data broker. In our study, we found that 273 non-compliant apps leak sensitive data to 22 data brokers. The top four data brokers are *unity3d*, *adcolony*, *kochava*, and *taboola* (see details in [30]).

• *App maker.* According to Apple's privacy label requirements *"...(Apps) need to identify all of the data you or your third-party partners collect.."* [47], apps are responsible for the disclosure of data collection by in-app third-party libraries. However, many app owners or developers delegate the app development to app makers, who operate an app builder business to automatically and instantly create apps or landing pages. In such cases, the actual owners of these apps are never in the position to specify privacy labels, while the compliance of the privacy label relies largely on the awareness and the honesty of app makers. In our study, we observed 9 app makers that distributed 133 non-compliant apps on behalf of the app developers/owners. Specifically, we observed that all 15 non-compliant apps generated by Appy Pie, a top self-service app maker ranked by [4], stated *Data not Collected* in privacy labels (meaning they do not collect data). However, the privacy policies and app descriptions of these apps disclose certain data collection. For instance, the app *Centre Laser Pro* [2], an app for beauty treatments, was built by Appy Pie with the privacy label stating *Data Not Collected*. However, we observed that coarse location obtained from the IP address was sent to *chatbottest.appypie.com*; also, the precise geographical precise location data along with the user identifier were sent to the endpoint *api.appexecutable.com* which is owned by Appy Pie to perform in-app notification for developer's marketing and advertising purpose. It also sent usage information (e.g., log time, disk space) to *analytics.appypie.com* for analytics purpose. Even worse, Appy Pie also embeds advertising networks such as Google Ads [21] and inMobi [27] into apps. Those ad networks also collect and transmit device information (e.g., device ID, screen size, etc) along with other advertising data. Served as a supplement to prior work [84] that reveals severe security issues of app makers, our findings demonstrate the privacy issues brought by app makers.

[2]We acknowledge that this list is not comprehensive as some data brokers (e.g. Anamoly 6, Babel Street) are not registered, and the laws around the registry still need better enforcement.