# Measuring Compliance Implications of Third-party Libraries' Privacy Label Disclosure Guidelines


Yue Xiao
Indiana University Bloomington
Bloomington, Indiana, USA
cz42@iu.edu

Chaoqi Zhang
Indiana University Bloomington
Bloomington, Indiana, USA
cz42@iu.edu

Yue Qin
Indiana University Bloomington
Bloomington, Indiana, USA
qinyue@iu.edu

Fares Fahad S Alharbi
Indiana University Bloomington
Bloomington, Indiana, USA
fafaalha@iu.edu

Luyi Xing
Indiana University Bloomington
Bloomington, Indiana, USA
luyixing@indiana.edu

Xiaojing Liao
Indiana University Bloomington
Bloomington, Indiana, USA
xliao@indiana.edu



## ABSTRACT

Privacy label disclosure guideline, which specifies the data usage practices of third-party libraries (TPL), is a valuable resource for iOS app developers to accurately complete their iOS privacy labels. This is particularly important given the mandatory requirement for all apps on the App Store to disclose their data practices via privacy labels. However, it is essential to ensure the accuracy and compliance of these guidelines to ensure that accurate TPL data usage has been provided to app developers. Despite the significance of these guidelines, there is little understanding of how accurate and compliant they are in reflecting the actual data practices of third-party libraries used in iOS apps. To address this issue, our study implements a tool called *Colaine* to automatically check the compliance of privacy label disclosure guidelines, taking into account the configurable data practices in TPLs. *Colaine* analyzed 107 TPLs associated with 1,605 different configurations, shedding light on the prevalence and seriousness of privacy label disclosure guideline non-compliance issues.



## CCS CONCEPTS

• **Security and privacy → Software and application security**; **Systems security**.

## KEYWORDS

Privacy Compliance Check; Mobile Supply Chain; User Privacy; Third-party Libraries; Consistency Model; Dynamic Analysis

**ACM Reference Format:**

Yue Xiao, Chaoqi Zhang, Yue Qin, Fares Fahad S Alharbi, Luyi Xing, and Xiaojing Liao. 2024. Measuring Compliance Implications of Third-party Libraries' Privacy Label Disclosure Guidelines. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3658644.3670371



Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.



*CCS '24, October 14–18, 2024, Salt Lake City, UT, USA*



© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.



ACM ISBN 979-8-4007-0636-3/24/10
https://doi.org/10.1145/3658644.3670371


## 1 INTRODUCTION

Third-party libraries (TPL) or Software Development Kits (SDKs) play a crucial role in the mobile supply chain, and it is essential for them to provide comprehensive information about their data collection practices. This ensures transparency and privacy for downstream app developers who integrate these TPLs into their mobile applications. However, previous research has highlighted that data practices of TPLs often lack transparency for app developers, leading to challenges in fully disclosing data practices in their apps and posing non-compliance risks [52, 58, 74, 76]. To address this issue, many TPL vendors, such as Facebook, Google, and Flurry Analytics, have released *TPL privacy disclosure guidelines* to ensure TPL privacy transparency for downstream app developers. Unlike *app-level* privacy disclosures (i.e., iOS Privacy Label [5], Google Play Data Safety Section [19], mobile app privacy policies), TPL privacy disclosure guidelines are designed to assist app developers to understand the data usage of TPLs and make informed decisions, regarding the integration of TPLs in their mobile apps, ultimately ensuring better privacy compliance and accountability.

A prominent example of TPL privacy disclosure guideline is *privacy label disclosure guideline* (Figure 1), which specifies data usage practices of a TPL to assist app developers in accurately completing iOS privacy labels for their apps. Following Apple's recent mandate requiring all apps to disclose their privacy labels and data practices, including those collected by TPLs, app developers are increasingly relying on the TPLs' privacy label disclosure guidelines to align with Apple's policy requirements. Our study (§ 3) reveals that among 305 leading TPLs, of which 99% of iOS apps in the Apple Store use at least one [22], 107 (35%) have provided privacy label disclosure guidelines. Meanwhile, prior work [76] involving 18 TPLs identified instances where incorrect privacy label disclosure guidelines led to non-compliant iOS app privacy labels, posing privacy risks for both app developers and end-users. Given these findings, it becomes imperative to conduct a comprehensive investigation into the compliance implications of privacy label disclosure guidelines and their associated TPLs on a larger scale, aiming at deepening our understanding of the privacy risks and accountability issues associated with non-compliance in TPL data disclosure.

However, fully inspecting compliance of privacy label disclosure guidelines provided by TPL vendors at scale poses challenges in precisely defining inconsistencies within the diverse functionalities

| Data Item | Operation | Configuration |
|---|---|---|
| Production Interaction | collect by default but can be disabled | You can disable tracking production interaction data by `Branch.setTrackDisabled(true)`. |
| Advertising data | not collect by default but can be enabled | You can enable collect advertising data by setting `AppsFlyerAdRevenue.start()`. |
| User ID | Yes | - |
| Sensitive Info | No | - |

**Figure 1: An example of privacy label disclosure guideline.**

and sophisticated data usage practices of TPLs, as well as automating their analysis. Specifically, configurable TPLs offer flexibility for app developers to customize the behavior of the library based on their specific requirements. The configuration process typically involves using configurable APIs to set parameters that control how the TPL handles user data. This ability to configure a TPL introduces configurable data usage practices (e.g., disabling a by-default data collection). However, the analysis of configurable data usage practices requires the assessment of TPL's configurable-data-flow to privacy label disclosure guideline (flow-to-guideline) consistency (§ 3), and a rigorous definition of flow-to-guideline inconsistencies in the context of privacy label disclosure guideline (§ 3.4.1). Hence, in this study, we aim to address two key research questions: (1) How to formalize a consistency model that takes into account the diverse and configurable data usage practices of TPLs? (2) How to design an automatic TPL analysis method to uncover the actual data collection practices under different privacy configurations?

**Methodology**. In this paper, we present an automatic methodology called *Colaine* to verify the compliance of privacy label disclosure guidelines for iOS TPLs on a large scale. Specifically, *Colaine* focuses on checking the consistency between the disclosed data practices in the privacy label disclosure guideline and the actual configurable data flow within the TPL when integrated into iOS applications.

To address the first research question, we formalize a consistency model, considering the configurable data usage practices (§ 3.4.1). Such a model is able to measure to what extent a configurable data practice executes a rule (e.g., a privacy statement indicating which party collects what data) specified by the associated privacy label disclosure guidelines. Accordingly, we define an *operation-level* inconsistent disclosure based on whether a configurable data practice is under-claimed by the disclosure guideline, and three types of *configuration-level* inconsistent disclosure based on whether the configurations applied to a certain data object are *omitted*, *contrary*, or *inadequate* in the privacy label disclosure guideline.

Regarding the second research question, we employ a set of techniques including configuration patch generation, natural language processing (NLP), and iOS app dynamic analysis techniques to analyze configurable data flow of TPLs (§ 3). More specifically, to analyze TPL's configurable-data-flow, we design a novel configuration patch-based TPL wrapper app generation and testing technique. It uses NLP techniques to identify configurations affecting data usage practice from TPL API documentations, and then automatically constructs machine-readable configuration patches for those configurations. Such configuration patch specifies a set of instructions that dictate, given a configuration, how a wrapper app should be modified to achieve a desired configurable data usage practice. Those patches further guide TPL's wrapper app generation and dynamic analysis, considering all possible configurable data usage practices. Our evaluation on the crafted groundtruth dataset with 11 TPLs and their associated privacy label disclosure guidelines shows that *Colaine* can detect privacy label disclosure guideline non-compliance effectively and efficiently (§ 3.5).

**Large-scale measurement and discoveries**. Running *Colaine* on 107 TPLs and their privacy label disclosure guidelines, which consists of 1,605 different TPL configurations, *Colaine* identified non-compliant privacy label disclosure guidelines in 47 distinct TPLs spanning 8 different categories, including Advertising, Analytics, and Engagement. Among them, 37 incorrectly disclose data collection operations, 8 have missing configuration information, 3 present invalid configuration settings, and 2 do not disclose comprehensive configuration information. The non-compliant TPLs have been incorporated into 82.26% iOS apps. As an example, CleverTap, a notable TPL, has been integrated with 71.7K iOS apps. In addition, we observed over 1K apps integrated with more than one non-compliant TPLs. It underscores the widespread issue of non-compliance privacy label disclosure guidelines among iOS TPLs.

Additionally, we noted significant privacy implications stemming from non-compliant privacy label disclosure guidelines. This particularly leads to iOS app developers failing to disable data collection in a TPL, which in turn results in inaccurate privacy disclosures at the app level. As a prominent example, we conducted a detailed analysis of nine privacy-conscious apps that attempted to disable data collection in CleverTap. Our reverse-engineering efforts revealed that eight out of these nine apps using CleverTap had non-compliant privacy labels. They failed to disclose the collection of user data, such as Email, UserID, and Sensitive Information. More specifically, according to CleverTap's privacy label disclosure guideline [2], app developers can disable user data collection of "email" and "gender" from CleverTap by a configuration setting `[[CleverTap sharedInstance] setOptOut: YES]`. However, our analysis found that this configuration does not stop user data collection. Additionally, the privacy label disclosure guideline inaccurately claims that the *UserID* is only collected when developers explicitly configure `NSDictionary *profile` to send this data, while CleverTap automatically generates and collects *UserID* for each user upon initialization (§ 4). We observed that these nine privacy-conscious app developers attempted to use the specified configuration to stop CleverTap's data collection, indicating in their privacy labels that they did not collect data like email and gender. However, due to the ineffective configuration settings, they inadvertently provided incorrect privacy disclosures.

**Responsible disclosure**. We have communicated our findings to all TPL vendors, except for 8 whose contact details were unobtainable. Of the 39 vendors contacted, 22 have responded to us. Among these, 10 have initiated corrective measures on their privacy label guidance or API documentation, while the remaining 12 have committed to further investigating the issues.

**Contributions**. We summarize the contributions as follows.

- We performed a systematic study on non-compliance issues of iOS TPL privacy label disclosure guidelines, particularly focusing on configurable data practices and their associated privacy statements.

• We designed and implemented a novel compliance analysis tool *Colaine* that can automatically assess the "configurable flow-to-guideline" compliance. *Colaine* is based on our new, formally defined consistency model for TPL's configurable data practices. We have released the source codes and datasets at [40].

• We explored new insights to enhance the design of privacy label disclosure guideline, and discussed the design of a unified, fine-grained privacy disclosure for TPLs `PBOM`, allowing efficient adoption and smooth interoperability.

# 2 BACKGROUND

## 2.1 Configurable TPL

Configurable TPLs provide app developers with the flexibility to tailor the library to their specific app functionality needs and/or privacy requirements. As shown in § 4.2, 94 out of 107 widely-used iOS TPLs are configurable TPLs.

The ability to configure a TPL typically involves the use of a configurable API (e.g., `setAnalyticsCollectionEnabled`) that allows developers to set specific parameters or values that dictate the behavior of the library. These configurations can range from simple on/off switches for specific features, to more complex rules and policies for handling user data.

**TPL privacy configuration**. In our study, we categorize TPL configuration settings into two types: *privacy configurations*, which relate to the collection and usage of user data, and *non-privacy configurations*, which affect other aspects of the TPL behaviors, such as performance optimization and UI customization.

Examples of privacy configurations include the settings to enable or disable functionalities affecting its data usage practices, such as analytics tracking, logging, or error reporting. Additionally, via privacy configuration settings of a configurable TPL, app developers will be able to assign user attributes, such as user IDs or session IDs, which can help the library keep track of specific users' behavior and enable personalization of the app's features. Another important privacy configuration related to TPLs is to enable different privacy compliance settings. Configuration settings related to privacy include compliance with different privacy laws (e.g., GDPR (General Data Protection Regulation), COPPA (Children's Online Privacy Protection Act), CPRA (California Privacy Rights Act), etc.), various restrictions on data access and usage (e.g., `[VunglePrivacySettings setPublishIDFV: NO]`, `[ALPrivacySettings setDoNotSell: NO]`). More examples of privacy configuration settings and their purposes can be found in [40]. In our study, the term "configuration" or "configuration settings" refers specifically to privacy configurations, as we focus on studying the configurations affecting data usage practices of TPL.

**Data practices in TPL privacy configuration**. In our study, data practices related to privacy configurations are represented as $(a, X)$, where $a$ and $X$ are the data operation and the associated configuration settings. Typical data operations $a$ under the context of privacy configuration include:

• Data is collected by default, while the app developer can disable the configuration, named "disable" configuration. For instance, the TPL Branch [1] automatically tracks user events (e.g., app_launch_time) and user attributes (e.g., UserID) when app developers initialize the TPL. However, such tracking data can be disabled by setting `Branch.setTrackingDisabled(true)`.

• Data is collected only when an app developer, that integrates a configurable TPL, enables the configuration, named "enable" configuration. For example, the TPL Appsflyer [6] allows the app to collect advertising data by configuring `AppsFlyerAdRevenue.start()`, with the intention of delivering personalized and targeted ads, as well as calculating advertising revenue.

## 2.2 TPL Privacy Disclosure Guideline

In recent years, many TPL vendors (e.g., Facebook, Google, Flurry Analytics) have released their TPL privacy disclosure guidelines to assist app developers in accurately and comprehensively specifying privacy disclosures related to TPL data practices. Unlike app privacy disclosures (i.e., iOS privacy Label [5], Google Play Data safety section [19], mobile app privacy policies) that are designed to provide transparency to *end-users*, TPL privacy disclosure guidelines are intended to guide *app developers* on how to disclose data use of TPLs to their app users. By following these guidelines, app developers can ensure that their users have a clear and transparent understanding of the data collection and usage practices of the libraries used within their apps. In our study, we focus on "iOS privacy label disclosure guideline" provided by TPL vendors, which aims to assist iOS app developers in generating iOS privacy label that accurately reflect the data collection and usage behavior of the TPL used within in their apps. Note that iOS app privacy compliance issues of iOS privacy label have been extensively studied in [76], which is outside the scope of this paper. More details about the differences between the "iOS privacy label disclosure guideline" and the "privacy label" can be found in [40].

**Privacy label disclosure guideline**. Privacy label disclosure guideline outlines a series of *privacy statements* (Figure 1) that specify data usage practices of a TPL that app developers must know when setting iOS privacy label for their apps. Here we denote a privacy statement as $(d, a, X)$, where $d$ is a privacy data item, $a$ and $X$ are the data operation and the associated configuration settings applied to $d$, see detail in § 3.4.1. In general, privacy statements in the privacy label disclosure guideline can be broadly classified into two types: those that are associated with configurable data usage practices and those that are not. The first type of privacy statement is typically included when a TPL collects, processes, or shares data via privacy configurations (see §2.1). Figure 1 shows an example of a privacy label disclosure guideline specifying that the data item "Advertising data" is collected only when the app developer configures the API "AppsFlyerAdRevenue.start()", while "Production Interaction" is collected by default if the app developer does not disable this practice.The second type of privacy statement will be included if the TPL does not collect data, or always collect data, regardless of different configurations. As examples in Figure 1, the privacy label disclosure guideline clearly notified app developers that the "User ID" data is always collected, while "sensitive info" is not collected at all.

Note that the taxonomy of the iOS privacy label defines 32 privacy data items [5]. Therefore, TPL vendors should align the data items in the library with those pre-defined in the iOS privacy label. This helps ensure that app developers can accurately disclose the

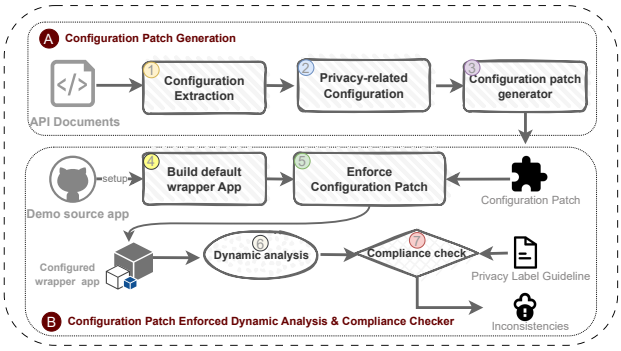


**Figure 2: Overview of *Colaine***

data collection and usage practices of the TPL to end-users in a clear and standardized way.

## 2.3 Consistency model

In the field of privacy compliance analysis [43, 46, 70, 77, 85], a consistency model refers to a formally-defined measure to assess the extent to which the data usage practices (e.g., data collection behaviors in TPL) adhere to the privacy statements (e.g., a privacy statement indicating which party collects what data) specified by the associated privacy disclosures (e.g., privacy label disclosure guideline). More specifically, a disclosure can correctly and completely indicate critical information in a data usage practice if the practice follows all privacy statements in the disclosure. Otherwise, an inappropriate disclosure occurs, indicating that certain privacy statements are violated by the data usage practice.

**Commonly-used inconsistency detection logic**. An inconsistency detection logic captures the difference between a data flow in a data usage practice and its associated privacy statements in privacy disclosure. A privacy statement or a data flow can typically be formalized as a triplet in the form of $(e, c, d)$, where $e$ is the subject, $c$ is the predicate and $d$ is the object. Here, the predicate can be along with positive or negative sentiments. For example, PoliCheck [43] 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 our study, we develop a novel consistency model tailored to the context of TPLs. Specifically, we focus on formalizing TPL's configurable data usage practices and the associated privacy statements in privacy label disclosure guidelines. We elaborate on the annotation and inconsistency analysis in § 3.4.1.

# 3 METHODOLOGY

In this section, we elaborate on the design, implementation, evaluation of *Colaine* – a tool to automatically check the compliance of privacy label disclosure guidelines with configurable TPLs.

## 3.1 Overview

**Architecture**. Figure 2 illustrates the architecture of *Colaine*, which comprises two primary components: Configuration Patch Generator (CPG) and Configuration Patch-Enforced Dynamic Analysis & Compliance Check (EDA). Specifically, in CPG, *Colaine* first collects API documentation of configurable TPLs to extract configuration descriptions and associated code snippets (❶). After that, *Colaine* uses NLP techniques to identify those privacy-related configurations that impact TPL data collection and usage behavior (❷). Given each privacy-related configuration, *Colaine* parses its semantics and generates various machine-readable configuration patches for different configuration settings (❸). In EDA, with the input of default wrapper apps integrating each TPL (❹), those configuration patches will be used to generate iOS apps, with each app enforcing a specific configuration setting (❺). *Colaine* then takes those apps as input for privacy compliance check: particularly, *Colaine* performs an end-to-end execution (fully automated app UI execution, dynamic instrumentation, and network monitoring) to investigate data usage practices, abstracted as tuples $(d, a, X)$ as defined in § 3.4.1 (❻). *Colaine* finally reports non-compliance by comparing those tuples with privacy statements in privacy label disclosure guideline of the corresponding TPL, following the consistency model defined in § 3.4.1 (❼).

**Dataset summary**. We summarize the dataset produced and consumed by each stage of our pipeline as below. Table 1 shows the datasets and corpora used in our study.

We began by utilizing a list of top TPLs, of which 99% of iOS apps in the Apple Store use at least one [22], to identify 305 leading TPL vendors across 14 categories. To collect their privacy label disclosure guidelines, we searched the websites of these vendors using a predefined list of privacy-related words (see details in [40]), and found that 107 (35%) of them provided privacy label disclosure guidelines. After that, we manually collected their API documentation. Next, *Colaine* extracts 1,338 privacy configurations' descriptions and code snippets from API documentation, and generated 1,605 configuration patches. In EDA, *Colaine* built up 107 default wrapper apps for 107 TPLs. After that, *Colaine* applied 1,605 configuration patches on the default wrapper apps and successfully generated 1,458 configured wrapper apps for dynamic analysis and compliance check. Overall, our study reported 47 non-compliant TPLs associated with 181 privacy statements in their privacy label disclosure guidelines.

## 3.2 Privacy Configuration Patch Generation

The goal of CPG is to generate configuration patches from a TPL's API document. It consists of three sub-components: (1) extracting configuration descriptions and code snippets from API documents; (2) identifying privacy configurations from all configuration descriptions; and (3) constructing machine-readable configuration patches for each privacy configuration.

*3.2.1 Configuration Information Extraction.* API documentation usually follows a standard format for presenting configuration settings, which includes a paragraph description accompanied by a block of code. However, the underlying DOM structure often differs significantly across different TPL vendors, making it difficult to use a single HTML parser for uniformly extracting relevant information. Instead, Optical Character Recognition (OCR) techniques, which recognize text and code within an image, provide a distinct advantage when capturing configuration descriptions and code snippets from diverse TPLs. In our study, *Colaine* uses Playwright library [62] to automate the process of capturing full-page screenshots of API

**Table 1: Summary of datasets and corpora**

| Name | Source | Size | Timestamp (yyyyMM) | Usage |
|---|---|---|---|---|
| $D_g$ | Cocopods | 107 TPL | 202303 | Detection |
| $C_g$ | 107 TPL privacy label disclosure guidelines | 2076 sentences | 202303 | Detection |
| $C_{api}$ | 107 TPL API documentations | 1338 privacy configurations | 202304 | Detection |
| $D_a$ | iOS Apps | 5k+ | 202206 | Measurement |

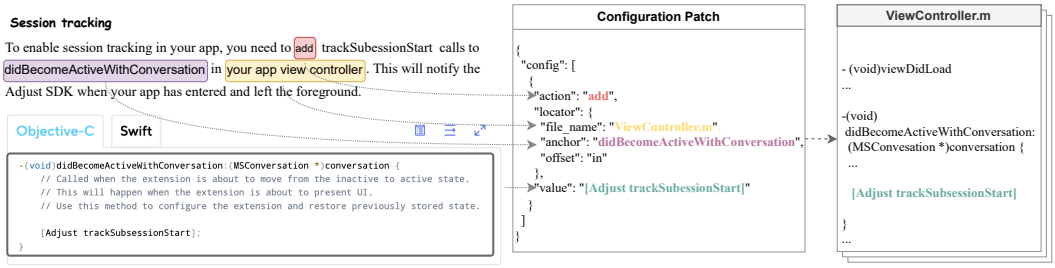


**Figure 3: Configuration patch generation and enforcement**

documentation for each TPL. These screenshots are then fed into an OCR model, which is used to extract configuration descriptions and code snippets. To create an OCR model for configuration setting recognition, we fine-tuned the base model from Butler [7] by utilizing manually annotated configuration descriptions and code snippets of 858 configurations across 10 TPLs.

**Evaluation**. We randomly sampled and manually checked 172 configurations from 97 TPLs (not including the 10 TPLs for model training). Our approach achieves 95.34% accuracy.

*3.2.2 Privacy Configuration Identification.* As mentioned in § 2.1, TPL typically provides a diverse range of configurations for various aspects, including privacy configuration (e.g., `[VunglePrivacySettings setPublishIDFV: NO]`) and non-privacy configuration (e.g., `Instabug.tintColor=UIColor.lightGrayColor;`). In our study, we focus on the configurations affecting data usage practices of TPL, namely *privacy configuration*. Below we elaborate on how we identify privacy configurations from configurations extracted in the previous step.

Specifically, we build up a binary classification model that identifies privacy configurations and non-privacy configurations. In our study, we compared the effectiveness of five models: BLSTM w. attention, BLSTM w/o attention, BiGRU w. attention, BiGRU w/o attention, Logistic Regression with TF-IDF, on this binary classification task, and BLSTM with attention outperforms other models (see Appendix § 8.2). BLSTM is able to memorize longer sequences of the input data and learn the context of sentences, while the attention mechanism focuses the model more on privacy configuration-related words (e.g., user privacy data (email address), privacy laws (GDPR), data usage (tracking/analytics), etc.). Note that to build and evaluate the models, we manually labeled 1043 configurations (157 privacy configurations and 886 non-privacy configurations) from 13 distinct TPLs, for training, testing, and evaluation purposes.

**Implementation.** After recognizing configuration descriptions and code snippets from screenshots of API documentations (See § 3.2.1), *Colaine* tokenizes them into words and concatenates each word's vector generated by BERT. The resulting word embeddings are fed into a two-layer LSTM, with hyperparameter tuning for optimal unit numbers. An attention layer is added to the LSTM to highlight important features, and the output is globally averaged and concatenated with the length input. The model uses a dense softmax output layer for classification and is trained using categorical cross-entropy loss and the Adam optimizer. Evaluation is performed using the F1 score.

**Evaluation**. Three annotators independently labeled privacy configurations based on an annotation guide [40]. In total, 157 privacy configurations and 886 non-privacy configurations from 13 TPLs are labeled. Note that the imbalance between privacy and non-privacy configurations reflects the real-world distribution of fewer privacy configurations in TPLs. Our aim was to train our model on a realistic dataset for effectiveness and practical applicability. For model training, we used 858 configurations (119 privacy and 739 non-privacy) from 10 TPLs as the training set, and 185 configurations (38 privacy and 147 non-privacy) from 3 TPLs as the testing set to evaluate our approach. The results show that *Colaine* achieves 92.5% precision and 92.11% recall in identifying privacy configurations. False positives primarily arise when privacy data mentioned in configuration descriptions originate from app developers rather than users (e.g., configuring secret API keys to grant entitlement access). False negatives occur due to OCR's inability to extract complete descriptions and code snippets, leading to missed privacy configurations.

*3.2.3 Configuration Patch Construction.* The `Configuration patch` specifies a set of instructions that dictate, given a privacy configuration, how the default wrapper app should be modified to achieve a desired data usage practice. Specifically, it specifies the semantics of a privacy configuration setting, including the operation to enforce the configuration, the value of the configuration, the path to add the configuration setting. In this subsection, we discuss how we design the format of configuration patches and how to generate machine-readable configuration patches from a configuration description in a TPL API documentation, as shown in Figure 3.

**Configuration Patch Format.** In our study, we adopt the format defined in JSON Patch [26], which outlines a method for describing changes to JSON documents as specified in IETF's RFC 6902 [4], to design the format of `Configuration patch`. This format aims

to facilitate partial updates for configuration options/settings in a standard and machine-readable manner.

More specifically, each `Configuration patch` is represented in a JSON format, expressing an array of patch objects to apply to a target wrapper app. Each patch object specifies a unique modification to the default wrapper app's configuration settings. Particularly, each patch object encapsulates three crucial elements: `(op, value, path)`. These elements represent which modification operation (*op*) performs on what configuration setting value (*value*) and enforces at what target location (*path*), as elaborated below. Note that we name these elements following the taxonomy of JSON Patch [26] specified in IETF's RFC 6902 [4].

• `op` field: The `op` value can be one of `add`, `remove` or `replace`; The `add` action adds a target to the specified locator. The `remove` action removes the target from the specified locator. The `replace` action replaces the target at the specified locator with a new target.

• `path` field: For `path` field, its value is a three-tuples `(file_name, anchor, offset)`. The `file_name` shows which file to modify (e.g., *AppDelegate.m*). The `anchor` and `offset` narrow down the modification scope to the exact location where the operation is performed. The anchor is a function call and the `offset` can be one of "`in`","`before`", "`after`" which indicates location restrictions. For example, in a configuration description *"The setUserId has to be called before measureSession call in AppDelegate.m"*, the `file_name` is *AppDelegate.m*, the `anchor` is the function *measureSession* and the `offset` is `before`.

• `value` field: The *value* field represents the value of a configuration setting that needs to be operated. For instance, the configuration patch enables UserID setting by calling the API `setUserId`, and the corresponding value is specified as *"value": "Tune.SetUserId ("userId");"*.

**Configuration Patch Generator.** To generate machine-readable configuration patches `(op, path, value)`, in our study, we implement a series of Natural Language Processing (NLP) techniques to retrieve semantic information of each privacy configuration in the TPL's API documentation

• *Retrieving* `op`. As mentioned earlier, the `op` field in the configuration patch defines the modification operation (`add`, `remove` or `replace`) to enable the privacy configuration. Meanwhile, the VERB used in a configuration description dictates how the configuration can be enabled. Therefore, in our study, we construct a mapping between the VERB in a configuration description and the value of the `op` field in the configuration patch. Specifically, we first identify the VERB in a configuration description via Part-of-Speech (POS) tagging techniques, which label a word in the text corpus as corresponding to a particular part of speech as well as its context (such as nouns and verbs). If a word is identified as a VERB (tagged as "VB"), *Colaine* then uses a VERB-`op` mapper to determine its corresponding operation. In our study, we manually construct a VERB-`op` mapper by analyzing 119 privacy configurations from 10 TPLs, comprising 88%, 7.8%, and 4.2% configurations for the `op` "add", "remove", and "replace", respectively. Table 6 shows examples of the mappings between VERB and `op`.

• *Recovering* `value`. TPLs typically provide ready-to-use code snippets for privacy configuration settings (80.4% in our study). These code snippets can be directly extracted from API documentations by the OCR techniques (See § 3.2.1).

However, there may be situations where the TPL provides multiple parameter values for app developer to specify. For instance, the configuration description *"the age-restricted user flag can be set to Yes or No"* indicated that the TPL offers two parameter options for the function `setIsAgeRestrictedUser`. However, the code snippet typically only shows one of these options. In such cases, two configurations should be spawned by setting different Boolean values. Things can get more complex when the parameter options are Enumeration Type with customized values. For example, the configuration description *"possible consent values are CHBGDPRConsentBehavioral and CHBGDPRConsentNonBehavioral"* indicates that two customized values can be specified. To retrieve the possible values in the enum_list, we first extract the parameter value specified in the code snippet using regex expression `\b(?:[A-Z][a-z0-9]*|[a-z]+[A-Z][a-z0-9]*)\b`. Then, we identify other parameter options that have a conjunction relationship with `CHBGDPRConsentBehavioral` in the configuration description's dependency tree.

• *Retrieving* `path`. As mentioned in § 3.2.3, the `path` in configuration patch consists of three-tuples, including `file_name`, `anchor`, and `offset`. Our system employs regular expressions to match `file_name` and `anchor`, and conducts dependency analysis to dereference `offset` by examining grammatical relationships. Specifically, `file_name` typically has a ".m" extension for implementation files (source code) and a ".h" extension for header files in Objective-C. Our system uses the regular expression `^[\w\s-]+(\.m|\.h)\$` to match file names mentioned in configuration descriptions. In cases where the API documentation does not mention the file extension (".m" or ".h"), we also use a keyword list of common iOS classes and objects [12] to recognize `file_name`. The `anchor` is typically represented as a function name or function call. Our system utilizes the regular expression `^[a-z][a-zA-Z0-9]*\$` to match function names and `^\\s*\\[[^\\[\\]]+\\s+[a-zA-Z0-9_-]+(?:\\s*:[^:\\]]+)*\\]\\s*\$` to match function calls. To identify the `offset`, we leverage a dependency parser SpaCy [37] to locate the adposition of the `anchor`. Specifically, if the dependency relationship between them is a prepositional modifier (e.g., before measureSession) and the adposition is in the keywords list (e.g., in, to, before, after), we regard it as the `offset`.

**Evaluation.** To evaluate CPG, we randomly sampled 130 privacy configurations reported by Privacy-related Configuration Identification (See § 3.2.2), which were manually confirmed, to inspect their configuration patches generated by CPG. *Colaine* achieve 87.6% precision and 91.5% recall to generate configuration patches. To compare the effectiveness with LLMs, we have crafted prompts to construct configuration patch, based on insights from related works [79, 80, 83]. Our experiments showed that LLMs could correctly construct 9 out of 11 configuration patches, close to our methodology's 90% effectiveness. However, LLMs require substantial amount of computing resources, with financial concerns especially for processing many APIs as in our research. Hence, our preference for custom, light-weight NLP tools over LLMs prioritizes efficiency and practicality, despite LLMs' potential for similar accuracy.

## 3.3 Configuration Patch-Enforced Dynamic Analysis & Compliance Checker

In EDA, *Colaine* first sets up a default wrapper app integrating the target TPL. It then enumerates each configuration patch to generate multiple apps with different configuration settings. These generated apps are then tested using a dynamic analysis pipeline to understand data usage practices and present it as a set of $(d, a, X)$. Finally, the compliance check module is used to detect any potential inconsistencies between the actual data practices and TPL privacy label disclosure guideline.

*3.3.1 Build Default Wrapper App.* The goal of this task is to construct a clean default wrapper app for each target TPL before introducing different configuration settings. However, the process of setting up the default wrapper apps may vary for different TPLs. Some TPLs provide ready-to-use wrapper apps, while others may require dependent libraries or registering for TPL developer accounts. A typical process is to create an empty app, integrate the TPL, install dependencies, register for TPL developer accounts, initialize TPL, sign the app with an Apple Developer account, and compile the app.

Specifically, we create an app wrapper using Xcode by integrating the TPL and specifying the version in the Podfile according to the version number mentioned in its privacy label disclosure guideline. Note that if the privacy label disclosure guideline does not specify a version number, we test the latest one. We then employ the CocoaPods dependency manager [3] to install all required dependencies, ensuring compatibility with iOS devices. In some cases (67 out of 107), we manually register for TPL accounts to embed unique API tokens for seamless integration and functionality. To sign the wrapper app, the signature is usually provided by the TPL vendor's developer team account. To run the app on our device, we replace the default signature with our Apple developer account. Finally, we utilize the xcodebuild command-line tool [54] to compile the demo app project into an executable binary.

*3.3.2 Configuration Patch Enforcement.* Here we deploy configuration patches (see § 3.2.3) to the default wrapper app, generating a wrapper app enforcing a specific configuration (named *configured wrapper apps*). In the patch enforcement process, configuration patches guide the generation of configured wrapper apps. Specifically, the `path` assists in navigating to the modification location, while `op` and `value` indicate the operation to be performed and the value of the configuration, respectively. In our study, we employed clang [10] to convert the Objective-C source code into an Abstract Syntax Tree (AST) representation, enabling us to accurately identify code locations that require revision. By navigating the AST structure, we can modify relevant code segments according to the configuration patches, seamlessly adding, replacing, or deleting code as necessary. For instance, in the configuration patch example shown in Figure 3, the "locator" field aids in pinpointing the modification site. The "file_name" indicates "ViewController.m" as the file for modification, while the "anchor: didBecomeActiveWithConversation" and "offset: in" identify the modification location within the "didBecomeActiveWithConversation" function. Our tool traverses the AST to locate the node for modification. Additionally, the "op: add" and "value: [Adjust trackSubessionStart]" instruct our tool to insert the sub-node "[Adjust trackSubessionStart]" within the "didBecomeActiveWithConversation" function in the AST. After applying the changes to the AST, we convert it back into Objective-C source code, resulting in an updated version incorporating the target configuration. The modified source code can then be compiled to build the iOS app.

**Evaluation.** To evaluate the performance of Configuration Patch Enforcement, we record the compilation status after applying each configuration. In our study, we applied a total of 1,605 configuration patches from 107 TPLs, with 90.84% successfully compiling into executable binaries. The failures are attributed to two causes. Firstly, OCR/NLP inaccuracies: 135 (8.41%) configuration patches were inaccurately generated due to OCR/NLP errors, as discussed in § 3.2, which affected correct compilation into wrapper-apps. Secondly, iPhone/Device requirements: 12 (0.75%) patches required the newer iOS-15.0.1, which was unavailable to us at the time of testing.

*3.3.3 Dynamic Analysis.* To investigate the data usage practices of TPLs, we deployed an open-sourced dynamic analysis pipeline Lalaine [76] on configured wrapper apps we generated in the previous step. Lalaine performs end-to-end execution (fully automated app UI execution, dynamic instrumentation, and network monitoring) on an app to infer (data, purpose) from the call stack for sensitive system APIs injected by Frida [18] and network traffic recorded by Fiddler [33]. However, Lalaine operates on a jailbroken iOS environment, as required by Frida, which may lead to crashes when the TPL employs jailbreak detection techniques. To address this limitation, our system embeds `FridaGadget.dylib` into the app during the Building phases. This allows Frida to instrument the app without the need for a jailbroken environment upon launch.

## 3.4 Consistency Model and Compliance Checker

*3.4.1 Consistency model.* In this paper, we propose a novel consistency model for configurable TPL's privacy label disclosure guideline, which includes an operation-level inconsistent disclosure based on whether the operation is under-claimed by the disclosure guideline and three types of configuration-level inconsistent disclosure based on whether the configurations applied to a certain data object are *omitted*, *contrary*, or *inadequate* in the privacy label disclosure guideline. Different from previous works [43, 46, 76] which defined the operation in the consistency model as a binary variable [43, 46] or a constant [76], our consistency model involves 4 types of operations along with the configurations applied to the data objects. More details about these distinctions can be found in [40]. Below we introduce the formal representation of the privacy label disclosure guideline, data flows, and the semantic relationship between data objects.

**Problem Definition**. In our definition, we let $\mathcal{D}$ represent the data items (e.g, Device ID) defined in the privacy label disclosure guideline; $\mathcal{A} = \{Y, D, E, N\}$ denotes the operation with 4 values, where $Y$ is short for `Yes` representing a compulsive collection, i.e., collecting at any time, $D$ is short for `Disablable` indicating that the TPL collects the data by default, while the collection can be disabled by certain configurations, $E$ is short for `Enablable` indicating the TPL does not collect the data by default, while the collection can be

enabled by certain configurations, and $N$ is short for `No` representing a compulsive no-collection, i.e., not collecting at any time. We define the *strict total order* over $Y, D, E, N$ as $Y > D > E > N$ according to the privacy risk of each operation; $C_X$ represents the configurations provided by privacy label disclosure guideline and $C'_X$ represents the configurations from the actual data flows.

**Definition 1** (Disclosure Guideline Representation: $S$)**.** A TPL's privacy label disclosure guideline is modeled as a set of tuples $S = \{s|s : (d_s, a_s, X_s), d_s \in \mathcal{D}, a_s \in \mathcal{A}, X_s \subseteq C_X\}$. Each tuple $s$ represents a privacy statement for a certain privacy-sensitive data item $d_s$, which discloses one operation $a_s$ and a set of configurations $X_s$ applied to $d_s$.

**Definition 2** (Data Flow Representation: $f$)**.** An individual data flow $f$ in a TPL is represented as a tuple $f = (d_f, a_f, X_f)$, where $d_f \in \mathcal{D}$ is a privacy-sensitive data item, $a_f \in \mathcal{A}$ and $X_f \subseteq C'_X$ are the operation and the configurations applied to $d_f$ in flow $f$.

**Definition 3** (Semantic Relationship)**.** Similar to prior work [43, 46, 76], 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 $v \sqsubset_o u$ or $u \sqsupset_o v$. 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 Disclosure: $\mathbb{S}_f$)**.** The privacy label disclosure guideline $s = (d_s, a_s, X_s)$ for data item $d_s$ is relevant to the flow $f = (d_f, a_f, X_f)$ (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 disclosure in the guideline. Let $C_f^S = \bigcup X_{s, s \in \mathbb{S}_f}$ denote all configurations in flow-$f$-relevant privacy disclosure guideline, which is also called the configuration portfolio of the privacy disclosure guideline relevant to $f$.

Given a flow $f$, by comparing its operation $a_f$, its configurations $X_f$, and the configuration portfolio $C_f^S$ of its relevant privacy label disclosure guideline, we reveal the operation-level flow-to-label inconsistency as well as three types of configuration-level inconsistencies.

**Privacy compliance risk 1** ($[\nvDash_{\mathcal{A}}]$ Operation-Level Incorrect Disclosure)**.** The privacy label disclosure guideline $S$ is an *operation-level incorrect disclosure* with regard to a flow $f$ if there exists a statement $s$ in flow-$f$-relevant disclosure $\mathbb{S}_f$

$$\exists s \in \mathbb{S}_f, a_f > a_s \implies S \nvDash_{\mathcal{A}} f.$$

**Privacy compliance risk 2** ($[\nvDash_C]$ Configuration-Level Omit Disclosure)**.** The privacy label disclosure guideline $S$ is an *configuration-level omit disclosure* with regard to a flow $f$ if the operation in the flow and the flow-$f$-relevent disclosure are both `Disablable` while the privacy label disclosure guideline fails to provide any configuration that can disable the collection operation.

$$\exists s \in \mathbb{S}_f, a_f = a_s = D \wedge C_f^S = \emptyset \implies S \nvDash_C f.$$

**Privacy compliance risk 3** ($[\nvDash_C]$ Configuration-Level Contrary Disclosure)**.** The privacy label disclosure guideline $S$ is an *configuration-level incorrect disclosure* with regard to a flow $f$ if the operation in the flow and the flow-$f$-relevent disclosure are both `Disablable` while there exists disablable configuration claimed by privacy label disclosure guideline which fails to disable the collection operation in the data flow.

$$\exists s \in \mathbb{S}_f, a_f = a_s = D \wedge \exists x \in C_f^S, x \notin X_f \implies S \nvDash_C f.$$

**Privacy compliance risk 4** ($[\nvDash_C]$ Configuration-Level Inadequate Disclosure)**.** The privacy label disclosure guideline $S$ is an *configuration-level inadequate disclosure* with regard to a flow $f$ if the operation in the flow and the flow-$f$-relevant disclosure are both `Disablable` while the privacy label disclosure guideline fails to provide all configurations that can disable the collection operation.

$$\exists s \in \mathbb{S}_f, a_f = a_s = D \wedge C_f^S \neq \emptyset \wedge C_f^S \subsetneqq X_f \implies S \nvDash_C f.$$

**Discussion**. Our consistency model is designed to identify and flag data collection and configuration-level inconsistencies between data usage practices and the stated data collection practices in privacy label disclosure guidelines. However, it does not cover purpose-level inconsistencies. This is because the majority of TPLs (87.85%, 93 out of 107) did not mention data purposes in their privacy label disclosure guidelines, making it infeasible for the compliance check in purpose-level.

*3.4.2 Compliance checker.* To conduct the compliance check, we first formalize dynamic analysis results into a data flow representation, and then transform privacy label disclosure guideline into a disclosure representation. Specifically, we use a semi-automatic method to formalize the privacy label disclosure guideline, employing an HTML parser for table-formatted disclosures and NLP techniques for sentence-based disclosures [42, 43, 73], manually validating the results. Subsequently, we adhere to the consistency model to assess the presence of any inconsistencies in the compliance analysis.

## 3.5 Evaluation of the *Colaine* System

**Experiment settings**. We ran *Colaine* on three iPhones (iOS versions 12.4.1, 13.7, 14.8.1), one Mac mini with an Apple M1 chip with 8-core CPUs, and two MacBook Pros with Apple M1 chip and Intel CPU.

**Evaluation results**. To evaluate the overall effectiveness of *Colaine*, we first craft the groundtruth set. We (three annotators, experienced in privacy and iOS development) randomly selected 11 TPLs (10% of total TPLs), and read through API documentation to identify privacy configurations. For each privacy configuration, we manually patched it to the demo app and examined data collection behavior by manually testing each patched app instance. In total, 118 privacy configurations are identified, and 136 patched app instances are created and fully tested. Further, we leverage domain experts to manually inspect the identified privacy configurations, generated network traffic, and stack traces to summarize the data and their corresponding operations. As a result, 18 non-compliant cases are identified, denoted as $(d, a, X)$, which are associated with 5 TPLs. This process yields the Fleiss' Kappa of 97.88%.

On the ground-truth dataset, *Colaine* identified 125 privacy configurations, generated 147 configuration patches, and successfully patched 132 app instances. *Colaine* reported 14 non-compliant $(data, operation)$ pairs (4 TPLs), showing a precision of 92.85% and a recall of 72.22% (for non-compliant data/purposes pairs), or a precision of 100% and a recall of 60% (for non-compliant TPLs). On average, it took 330 seconds (300 seconds for executing an app and 30 seconds for inconsistency analysis) to investigate one app.

• *Falsely detected inconsistency.* The one false positive, i.e., *Colaine* reported $(d, a, X)$ pairs, stems from the incorrect identification of privacy configurations in Start.io [38]. Such failure can cause *Colaine* to believe the identifier data is collected forcefully by the code, which in turn causes *Colaine* to report a false positive inconsistency. Specifically, we observed that Start.io's API documentation lacks appropriate formatting, as it utilizes a horizontal scrollbar to display program code that exceeds the available width. This formatting issue leads the OCR module to misidentify the code, resulting in the exclusion of specific privacy configurations during the patching process. Start.io's privacy label disclosure guideline indicates that identifiers data is collected by default. However, due to the OCR's failure to extract the correct configuration that can disable such data collection, *Colaine* detects that the identifier data is collected compulsorily by the code, which in turn causes *Colaine* to report a false positive inconsistency.

• *False negatives.* We identified two reasons for missed cases: (1) the inherent limitations of dynamic analysis, which cannot ensure complete code execution, leading to missed data collection detection; (2) inactive functionalities in the demo app, where a subscription is required to activate certain features. More specifically, *Colaine* missed detecting five non-compliant (data, operation) pairs. Among these cases, four were related to enabled data collection, while one involved data collected by default. Notably, the operation of "always-collected" data did not produce any false positives. This can be attributed to the fact that such data is typically collected immediately upon launching the TPL, ensuring that the corresponding code is always reachable. Regarding four missing operations of "enable-able collected" data, they occurred for operations where the dynamic analysis did not trigger the necessary user-specific actions or UI interactions. These operations were more hidden or required specific user inputs to be activated. As a result, the dynamic analysis was unable to reach and capture these cases due to limitations in the UI auto-execution tool. In the case of "collected by default", the dynamic analysis did reach the corresponding code. However, due to customized encryption implemented before data transfer, *Colaine* was unable to recognize the data in the network traffic, leading to a false negative result.

# 4 MEASUREMENT

## 4.1 Landscape

**Scope and magnitude**. Our study reveals that non-compliant TPLs are prevalent among popular TPLs with iOS privacy label disclosure guidelines. Running *Colaine* on 107 popular TPLs and their privacy label disclosure guidelines, *Colaine* reported 47 non-compliant TPLs: 37 incorrectly disclose data collection operations (O1-O6), 8 have missing configuration information (C1), 3 present invalid configuration settings (C2), and 2 did not disclose comprehensive configuration information (C3) as shown in Table 4. We detail the discussion of each type of non-compliance in § 4.3. Those non-compliant TPLs covering 8 categories, including Advertising (29.78%), Analytics (25.53%) and Engagement (17.02%).

**Table 2: Top-10 Non-compliant libraries (integrated in the most apps)**

| Non-compliant library | # of apps | Inconsistency type | Non-compliant data |
|---|---|---|---|
| CleverTap | 71.7k | O2, C2 | Email, Sensitive Info, UserID |
| RevenueCat | 53.8k | O6 | UserID |
| Chartboost | 49.3k | O1, O3, C2 | Other Diagnostic Data, Coarse Location, UserID |
| Flurry Analytics | 35.8k | C1, O3 | Precise Location, Coarse location |
| AdColony | 31.4k | O1,O4 | Precise Location, Health, Other Diagnostic Data |
| IronSource | 22.4k | O4, O3, O1 | Sensitive Info, Coarse Location, Precise Location, UserID |
| LogRocket | 13.5k | O2, C2 | Coarse Location |
| Mixpanel | 8.9k | C1, O5 | Coarse Location, UserID, User Content |
| Instabug | 4.5k | O4, O3, C1 | Audio Data, Customer Support, Emails & Text Messages, Name, Email Address |
| mParticle | 4.2k | C1 | User Content |

Table 2 illustrates the top 10 non-compliant TPLs based on the number of iOS apps integrating them. 70% of them contain more than one type of non-compliance. We identified several non-compliance issues with privacy label disclosure guideline of CleverTap, a TPL integrated by 71.7k iOS apps. These issues include presenting invalid configuration (C2) and inaccurately stating that data is only collected under specific configurations but in fact it is always collected (O2). Specifically, CleverTap's privacy label disclosure guideline [2] informs app developers that they can disable user data collection (i.e., email, gender) from CleverTap by setting `[[CleverTap sharedInstance] setOptOut: YES]`. However, our analysis found that this configuration does not stop user data collection. Also, CleverTap's privacy label disclosure guideline inaccurately states that the *UserID* is only collected if the developer explicitly configures `NSDictionary` to send this data, while in reality, CleverTap automatically generates and collects *UserID* for each user upon initialization.

**Impact of non-compliant TPLs**. 47 non-compliant TPLs, reported by *Colaine*, have been integrated into 82.26% iOS apps, covering 25 categories of the Apple App Store. Some apps with non-compliant TPLs are of high ratings: Solitaire (882.8k ratings, #29 in Casino), Slotomania (455.1k ratings, #37 in Casino), Pull the Pin (247.5k ratings, #5 in Board). Also, we observed that 1,549 apps integrated more than one non-compliant TPLs. For instance, the app *Drop The Number: Merge Puzzle*, ranked 34 in the Game category with a review score of 4.7 based on 71.6k ratings, integrated four non-compliant TPLs (IronSource, Appsflyer, RevenueCat, and Applovin).

Moreover, in our study, we ask a research question: how many iOS apps with a non-compliant TPL, which, per privacy label disclosure guideline, set the TPL to disable a by-default data collection (but in fact, data is still collected), fail to disclose such behavior in

**Table 3: Impact of non-compliant TPLs**

| TPL | Privacy Data | # Apps with "Disable" configuration | # Apps with non-compliant privacy labels |
|---|---|---|---|
| CleverTap | Email, UserID | 9 | 8 |
| LogRocket | Coarse Location | 6 | 5 |
| Chartboost | UserID | 8 | 5 |

their privacy labels? To answer this question, we collected 90 iOS apps integrated non-compliant TPLs CleverTap, LogRocket, and Chartboost, and reverse-engineered their binary codes using IDA Pro to find those "disable" configurations. After that, we fetched their privacy label in Apple App Store for a compliance check. To this end, we found 9 privacy-conscious apps that disable data collection of CleverTap, 6 for LogRocket, and 8 for Chartboost.

As shown in Table 3, we discovered that 8 out of 9 apps using CleverTap have non-compliant privacy labels as they fail to disclose the collection of user data (i.e., Email, UserID, and Sensitive info). These eight apps have high ratings with an average score of 4.3 (out of 5) and demonstrate privacy-consciousness by configuring `setOptOut` to `YES` in an attempt to stop CleverTap from collecting user data. However, they are misled by the invalid configuration (C2), which ultimately results in a violation of the Apple App Store's privacy requirements. Our study shows that non-compliant privacy configurations in privacy label disclosure guideline may easily lead to non-compliant privacy labels for iOS apps. This finding supplements the recent study [76] on 18 TPLs discussing that iOS apps with non-compliant privacy labels have been observed to integrate TPLs with incorrect privacy label disclosure guidelines. Note that [76] did not study privacy configurations in compliance analysis.

## 4.2 Privacy Label Disclosure Guideline

**Configurable vs non-conf. data practices**. Our study reveals that among 107 real-world privacy label disclosure guidelines of TPLs, 94 (87.85%) of them provide privacy configurations and allow optional data usage in their API Documentation. However, only 50 (46.73%) of them TPLs disclose configurable data usage practices. This lack of transparency makes it challenging for app developers to understand the potential privacy implications of using these privacy configurations of TPLs in their apps. The most common TPL categories that provide configurable data disclosure are *User Engagement*, followed by *Analytics* and *Ads & Monetization*. This may be because TPLs in these categories typically involve collecting and processing user data for monetization purposes.

As mentioned in § 2.1, the privacy statements in the privacy label disclosure guideline of a TPL can be broadly classified into two types: those that are related to configurable data usage practices and those that are not, while those configurable can be further associated with "enable" configuration (`E`) and "disable" configuration (`D`). Among 2076 privacy statements (19.4 per privacy label disclosure guideline on average) extracted from privacy label disclosure guidelines, 322 are associated with the configurable data usage practices, and 1,754 are not. Among all 322 configurable practices, 273 are associated with "enable" configuration (`E`) and 49 "disable" configuration (`D`). Out of the 1,754 data, 491 are always collected, while 1,263 are never collected.

**Table 4: Statistics of Non-compliant TPLs**

| Inconsistency Type | Categories | Data | # of Non-compliant TPLs | # of privacy statements |
|---|---|---|---|---|
| Inconsistent Operation Disclosure | O1 | Sensitive PII | 2 | 4 |
| | | PII | 4 | 6 |
| | | Non-PII | 7 | 13 |
| | O2 | Sensitive PII | 1 | 2 |
| | | PII | 6 | 11 |
| | | Non-PII | 5 | 10 |
| | O3 | Sensitive PII | 3 | 8 |
| | | PII | 8 | 22 |
| | | Non-PII | 4 | 9 |
| | O4 | Sensitive PII | 3 | 7 |
| | | PII | 6 | 16 |
| | | Non-PII | 4 | 9 |
| | O5 | Sensitive PII | 0 | 0 |
| | | PII | 4 | 7 |
| | | Non-PII | 4 | 8 |
| | O6 | Sensitive PII | 1 | 1 |
| | | PII | 3 | 6 |
| | | Non-PII | 2 | 5 |
| Invalid Configuration Setting Disclosure | C1 | Sensitive PII | 1 | 2 |
| | | PII | 6 | 15 |
| | | Non-PII | 4 | 11 |
| | C2 | Sensitive PII | 1 | 3 |
| | | PII | 1 | 2 |
| | | Non-PII | 1 | 1 |
| | C3 | Sensitive PII | 1 | 1 |
| | | PII | 1 | 2 |
| | | Non-PII | 0 | 0 |

**Updateability**. The ability to update privacy label disclosure guideline is crucial for maintaining policy/regulation compliance and transparency, especially given the rapid evolution of TPLs. Outdated guidelines can increase the risk of data breaches and non-compliance with laws and regulations. To investigate the updateability of privacy label disclosure guideline, we obtain its last modified date using various methods, such as checking the guideline itself, examining the HTTP header, or consulting the Wayback Machine [39] (see details in § 8.4). Our findings reveal that a significant portion of privacy label disclosure guidelines (37.37%) have not been updated for more than one year and 13.08% do not update for more than six months. However, some privacy-conscious TPLs (25.23%) frequently update their privacy label disclosure guidelines. We also examined the frequency of TPL updates since the last modified date by manually counting the number of TPL releases in the changelog after the last modified date. On average, TPLs release 18.88 versions after the last modified date, indicating that TPL versions are actively updated, while privacy disclosures lag behind in updates.

## 4.3 Non-compliant TPLs

*4.3.1 Incorrect Operation Disclosure.* As formally defined in § 3.4.1, in the context of configurable TPL, we consider the following four data collection operations {`Y, D, E, N`}, where "`Y`" represents that data is collected compulsively without being affected by any configurations. "`D`" indicates that data is collected by default but can be disabled by specific configurations. "`E`" means that data is only collected when certain configurations are enabled. "`N`" depicts that data is not collected under any configurations.

An incorrect operation-level disclosure arises when the data collection behavior (denoted as $a_f$) is actually manifested as being collected, but in the privacy label disclosure guideline (denoted as $a_s$) implying that data collection is not taking place at all or

under specific configuration. These inconsistencies can typically be attributed to the following six categories:

**O1: $a_f = Y, a_s = N$**. *Colaine* reports 8 TPLs in this category, where data is mandatory collected at the code level, while the guideline erroneously states that data is not collected under any configurations. For example, IronSource [23] mandatorily collects *Precise Location* data and sends it to its own tracking service at k.isprog.com. However, IronSource's privacy label disclosure guideline [24] incorrectly states that *Precise Location* data is not collected. Such inconsistency can result in considerable privacy risks. When app developers integrate the TPL, data is automatically collected beyond their control, while the guideline misleadingly states that it is not being collected.

**O2: $a_f = Y, a_s = E$**. In this category, data is mandatory collected by TPL, while the guideline inaccurately states that data is only collected under specific configurations. *Colaine* reports 6 TPLs with such violation, associated with 17 different privacy configurations such as `setUserEmail`, `trackPreciseLocation`, `logUserEvent`. For instance, CleverTap's privacy label disclosure guideline [2] incorrectly states that *UserID* will only be collected if the developer configures `NSDictionary *profile = @{@"UserID ": @"123456"}` to send this data. However, CleverTap [11] automatically generates and collects *UserID* for each user upon initialization. This inconsistency may lead app developers to mistakenly believe that their user data is safe because they did not enforce configurations that collect data. However, data is indeed collected compulsively, without being under the control of the app developers.

**O3: $a_f = D, a_s = N$**. *Colaine* reports 9 TPLs in this category, where data is collected by default but can be disabled, while the guideline inaccurately states that data is not collected at all. For example, Chartboost [8] collects the user's *Coarse Location* derived from the device IP address by default upon initialization. This data collection can be disabled using `Chartboost.addDataUseConsent(.CCPA(.optOutSale))`. However, Chartboost privacy label disclosure guideline [9] erroneously states that *Coarse Location* is not collected. This inconsistency leads app developers to believe that data collection will not occur. Consequently, they miss the opportunity to take measures to control data collection by disabling certain configurations. However, data is indeed collected by default.

**O4: $a_f = E, a_s = N$**. Here data is collected under specific configurations at the code level, contrary to the guideline's claim that no data is collected under any configurations. For example, Instabug [20] allows app developers to enable bug reporting functionality using `IBGBugReporting.enabledAttachmentTypes = ScreenRecording`. When this feature is enabled, audio is recorded and collected. However, Instabug's privacy label disclosure guideline [21] incorrectly states that *Audio Data* is not collected. This inconsistency may cause app developers to inadvertently enable certain configurations that trigger data collection, as they trust the guidelines' assurance of no data collection under those configurations.

**O5: $a_f = Y, a_s = D$**. In this category, the data is collected compulsively at the code level, while the guideline discloses that the data collection can be disabled by specific configurations. For example, Mixpanel [29] mandatory collects user ID and event ID, which are alphanumerically generated in its code and used to track user events and build user profiles. This data collection cannot be disabled through any privacy configuration patch. However, Mixpanel's privacy label disclosure guideline [30] incorrectly states that this data is only collected by default. This inconsistency may lead to app developers mistakenly believing data collection can be disabled under proper configurations, while the data collection at the code level would not be affected by any configuration.

**O6: $a_f = D, a_s = E$**. In this category, the data is collected by default at the code level, while the guideline discloses that the data is only collected under specific configurations (not collected by default). For example, RevenueCat [35] by default collects IDFV as User ID, retrieved from the system API using `- identifierForVendor`. However, RevenueCat's privacy label disclosure guideline [36] states that User ID is not collected by default, except when the app developer configures it using `Purchases.configure(withAPIKey: <my_api_key>, appUserID: <my_app_user_id>)`. In such a scenario, app developers, who did not enable certain configurations, will believe data collection would not be triggered, while the data is collected by default.

*4.3.2 Invalid Configuration Setting Disclosure.* Notably, even the data collection operations `{Y, D, E, N}` have been accurately disclosed in privacy label disclosure guideline, the associated configuration settings (denoted as $X_s$) could be omitted (C1), incorrect (C2) or inadequate (C3). Based on § 3.4.1, we consider the following three configuration setting inconsistencies.

**C1: $a_f = a_s = D, X_s = \emptyset$**. This category pertains to situations where the privacy label disclosure guideline of a configurable TPL lacks configuration settings, especially those that could disable data collection behavior. This omission can pose challenges for app developers who aim to safeguard their user data from being collected by third parties, as they may not know how to enforce the appropriate data protection measures. For example, mParticle [31] automatically logs custom events to collect user event data. However, the privacy label disclosure guideline [32] of mParticle does not mention that this data collection can be disabled by setting `event.shouldUploadEvent = NO`.

**C2: $a_f = a_s = D, \exists x \in X_s, x \notin X_f$**. This category highlights cases that the configuration setting ($x$) specified in the privacy label disclosure guideline is supposed to disable certain data collection, but does not actually work. For example, LogRocket [27] mandatory collects Coarse Location data calculated from the device's IP address. It retrieves the device's IP address by calling the iOS system API `getifaddrs` at a high frequency (averaging 94 times per configured app instance). However, LogRocket's privacy label disclosure guideline [28] incorrectly informs app developers that the location data is collected by default and can be disabled. However, we have examined all the related privacy configurations (e.g., `networkCaptureEnabled = false`, `viewScanningEnabled: Bool = false`) and found that none of them can actually disable this data collection. This inconsistency can potentially mislead app developers into believing that they have already disabled data collection by setting the configurations mentioned in the guideline. However, these configurations are invalid and do not prevent data collection.

**C3: $a_f = a_s = D, X_s \subset X_f$**. Inadequate Configuration Setting Disclosure means the TPL vendor has already declared one or multiple configuration settings but fails to disclose all of them. For example,

Dynatrace's privacy label disclosure guideline [17] discloses the configuration `DTXInstrumentGPSLocation = false` to disable precise location collection. However, we discovered an additional privacy configuration from Dynatrace [16], i.e., `privacyConfig.dataCollection dataCollectionLevel = .off`, that can also disable precise location collection. Such inadequate disclosure can lead app developers to miss alternative options for managing data collection, potentially resulting in suboptimal privacy settings or non-compliance with privacy regulations.

*4.3.3 Severity of non-compliances.* Table 4 show the breakdown of all reported inconsistencies. To better characterize the severity of each, we further categorize data type into three groups: Sensitive PII, PII, and non-PII (based on the definitions in Handbook for Safeguarding Sensitive PII [15] and ). In our study, we found that the violations of non-compliant TPLs associated with configurations are much more prevalent (158 out of 322 privacy statements) than those not associated with configurations (23 out of 1,754 privacy statements). This indicates that privacy violations are more likely to occur when it comes to configurable data usage practices. Furthermore, we observed that the configuration-related violations are associated with more sensitive data types (24 vs 4 in sensitive PII, 81 vs 6 in PII), which could potentially lead to more severe privacy risks. Therefore, it is crucial for configurable TPLs to provide clear and accurate privacy guideline in their API documentation to help app developers understand the data usage practices and make informed decisions when configuring the TPLs.

# 5 DISCUSSION

## 5.1 Generalizability and Scalability

**Adaptability to Android TPLs**. We test our prototype on Android TPLs and their privacy label disclosure guidelines to further assess its adaptability to Android platform.

In the implementation phase, to generate privacy configuration patch, we utilize a small subset of Android API documentation to fine-tune the OCR model (as detailed in Section 3.2.1), the Privacy Configuration Classification Model (Section 3.2.2), and the Configuration Patch Generator (Section 3.2.3). This approach was effective due to the similarity in format and description between Android and iOS API documentation. Additionally, the enforcement process on Android parallels that on iOS, necessitating only the substitution of Android-compatible tools. More specifically, we utilize Android Studio to create a default app wrapper. This tool automatically constructs `.apk` files by merging `.aar` SDK files with a basic Android application template. To enforce the privacy configuration patch to this default app wrapper, we utilize javaparser [25]) to convert the Java source code into an Abstract Syntax Tree (AST) representation. Subsequently, we traverse the AST structure, modifying code segments in accordance with the configuration patches. Finally, the AST is reconverted into Java source code, yielding a fully configured wrapper application. Regarding the Dynamic Analysis & Compliance Checker, their settings are platform-independent and adhere to the design outlined in Section 3.3. Note that the toolsets utilized by *Colaine*, such as Nosmoke (a cross-platform UI automation tool) and Frida (an instrumentation tool), are inherently cross-platform and can be adjusted to support Android functionalities. We will leave the large-scale compliance analysis of Android TPLs' disclosure guidelines as our future work.

**Scalability.** Our compliance analysis system operates automatically, though preparing inputs such as API documentation, demo source apps, and privacy guidelines does require manual effort. Scalability presents minimal challenges for SDK vendors and app developers. SDK vendors, possessing all necessary inputs, can utilize our tool for comprehensive compliance checks across their SDK versions. App developers, typically dealing with a manageable number of SDKs, find assembling these inputs reasonable.

## 5.2 Recommendation

**PBOM: privacy disclosure language for TPLs**. As mentioned earlier, most of privacy label disclosure guidelines fail to ensure privacy transparency for downstream developers. Particularly, we observe that the divergent specifications of privacy label disclosure guidelines across different TPLs, which include non-uniform formats, disparate locations, and ambiguous wording, may lead to app developers misinterpreting data collection practices. Note that while Apple recently (Dec. 2023) release "privacy manifest" requirement [34] for TPL vendors to specify their data practices, this privacy manifest design fails to account for the nuances of configurations, offering less granularity and not adequately reflecting the diverse data practices that can arise from different TPL configurations. In addition, existing privacy label disclosure guidelines have struggled to be adopted or lack a practical and scalable implementation, which hinders their effectiveness in ensuring privacy accountability throughout the mobile software supply chain. To this end, we believe the design and deployment of a unified and fine-grained privacy-accountable disclosure for TPLs is essential. Its potential benefits are multi-faceted: (1) TPL vendors can integrate it into their CI/CD pipelines, ensuring the continuous generation and release of such privacy disclosure for every TPL version, (2) downstream customers, especially app developers, can utilize such privacy disclosure to ensure better compliance, and (3) Apple can audit such privacy disclosure to further achieve its privacy and accountability objectives.

To fully realize the benefits of such privacy disclosure, a unified and compatible schema is essential. We propose this privacy disclosure PBOM to be built upon the emerging software bill of materials (SBOM) standard, such as CycloneDX [13], while introducing new schema elements and attributes specific to privacy disclosures while maintaining backward compatibility with existing SBOM formats. In particular, the CycloneDX standard is highly extensible, allowing for complex data to be represented in the BOM [14]. It supports Properties, a name-value store used to describe additional data about components, services, or the BOM that is not native to the core specification. The current design for data collection falls under the *service:data:flow* category. However, it only includes data classifications (e.g., PII, PIFI, public)[1]

[1]Data classification involves tagging data according to its type, sensitivity, and value if altered, stolen, or destroyed.

**Listing 1: Example of privacy disclosure PBOM for Flurry**

```
{
  "bomFormat": "CycloneDX", "specVersion": "1.4",
  "services": [{
    "provider": {"name": "Flurry Org", "url": ["https://www.flurry.com/"]},
    "name": "Flurry iOS SDK", "version": "12.2.0",
    "data": [{
      "classification": "Privacy Disclosure",
      "flow": [{
        "properties": [
          {"name": "Direction", "value": "Inbound"},
          {"name": "Data", "value": "Precise Location"},
          {"name": "Purpose", "value": "Analytics"},
          {"name": "Action_property", "value": "by default but disable-able"},
          {"name": "Configuration", "value": "[Flurry trackPreciseLocation:NO];"
      }]}
    ]} ]} ]}
}
```

and the flow direction (e.g., "inbound", "outbound")[2]. These elements alone are insufficient to fully disclose data collection practices. We propose adding four properties to thoroughly express data flow. These include (1) `{name: "data", value: <data_items>}`, which describes the data items transmitted in the flow; (2) `{name: "Purpose", value: <Purpose_list>}`, which outlines data usage; (3) `{name: "Action_property", value: [compulsively, by default but disable-able, enable-able, void]}`, indicating whether the data collection is configurable; and (4) `{name: "Action_config", value: <configuration>}`, describing the configuration that can enable or disable data collection behaviors. This representation of data flow can be easily converted into tuple-based representation (e.g., *(data, (Action_property, Action_config), purpose))*, which can be directly input into consistency models for compliance checks. Additionally, the name and version fields are required to facilitate fine-grained privacy disclosure.

**Example**. Listing 1 presents an example of privacy disclosure PBOM for Flurry, which extends the SBOM format specified by *CycloneDX 1.4*. The "`name`" and "`version`" fields provide a fine-grained privacy disclosure for the *Flurry iOS SDK 12.2.0*. The four newly added properties indicate that the *Precise Location* data is collected by default for the purpose of *Analytics*. However, this data collection can be disabled by configuring the setting `[Flurry trackPreciseLocation:NO]`.

## 6 RELATED WORKS

**Security and privacy analysis on TPLs.** Literature extensively studies security and privacy risks originating from TPLs in mobile apps. To measure the integration/dependencies of TPLs in mobile apps, various TPLs detection techniques [44, 48, 50, 56, 57, 60, 72, 82], and various isolation techniques [59, 68, 69] are proposed. To measure the privacy leakage caused by the widespread integration of TPLs, prior research has utilized both static [64] and dynamic [49, 65, 66] program analysis, revealing that sensitive user data collection is prevalent. Further, to resonate possible causes of such privacy risk, researchers [50] have identified that outdated library versions often contribute to privacy risks, and several auto-update techniques of TPLs have been proposed to address this issue [51, 53, 55]. In addition to issues with the out-of-date nature of TPLs themselves, researchers [41, 61, 67, 71, 81] have also discovered that misconfigurations of TPLs by app developers can result in the leakage of sensitive personal information. Unlike previous studies, we systematically investigate privacy compliance issues of configurable TPL's privacy label disclosure guideline. Specifically, we formally defined and examined to which extent privacy configurations that control data practices are missing (**C1**), invalid (**C2**), or inadequate (**C3**), and the degree to which operations dependent on these configurations are incorrectly disclosed (**O2-O6**).

**Privacy compliance analysis on mobile apps.** Privacy compliance checks [42, 43, 45, 47, 63, 64, 70, 71, 75, 78, 84, 85] have evolved from coarse-grained analysis to more complex, fine-grained analysis, transitioning from data-level consistency [70, 85] to entity-sensitive consistency [43], and purpose-sensitive consistency [46, 76]. Unlike previous studies, our consistency model and compliance analysis are tailored to configurable TPL's privacy label disclosure guideline, where data collection operation is configurable (i.e., `Y, D, E, N`) through the enforcement of privacy configurations in TPLs. To address this, *Colaine* extend the consistency model in literature to a configuration-aware consistency model, enabling new formal definitions of inconsistencies and non-compliance issues. More details about these distinctions can be found in [40].

Specifically, *Colaine* is designed for TPLs, unlike Lalaine [76], PurPliance [46], and PoliCheck [43], which targeted app-level analysis. *Colaine* introduces a configuration-aware consistency model, filling a gap overlooked by prior studies. Our model identified six operation-level (O1-O6) and three configuration-level (C1-C3) inconsistencies in TPL configurations, beyond the scope of prior models [43, 46, 76] that discussed only O1. For 47 non-compliant TPLs identified by Colaine, using prior models would only identify 8 (O1).

## 7 CONCLUSION

Our study proposes a new consistency model, and designs and implements *Colaine* to automatically check the compliance of privacy label disclosure guidelines, taking into account the configurable data practices in TPLs. Running on 107 TPLs and their privacy label disclosure guidelines, *Colaine* found 47 non-compliant TPLs associated with 181 different privacy statements. Our research further provided detailed measurement results for non-compliance issues of privacy label disclosure guideline that complement recent understandings of privacy compliance issues. Our study brings new insights into proper privacy label disclosure guideline design and implementation, essential to improve privacy compliance assurance.

## AVAILABILITY

The artifact for this work is available at https://zenodo.org/records/10976599.

## ACKNOWLEDGMENT

We thank the shepherd and anonymous reviewers for their valuable and constructive feedback. We also thank the undergraduate students Willem Armentrout, Samuel Beardmore and Aidan Kim at Indiana University Bloomington for their efforts in data collection and procession and cases studies. This work is supported in part by NSF CNS-2339537, 2330265.

[2]Direction is relative to the service. Inbound flow indicates that data enters the service. Outbound flow signifies that data leaves the service. Bi-directional implies that data flows both ways, and unknown suggests that the direction is not known

# REFERENCES


[1] https://help.branch.io/developers-hub/docs/attribution-api, Title = Branch API.
[2] Clevertap SDK guideline. https://docs.clevertap.com/docs/att.
[3] cocoapods., howpublished = https://cocoapods.org/.
[4] JavaScript Object Notation (JSON) Patch. https://www.rfc-editor.org/rfc/pdfrfc/rfc6902.txt.pdf, 2013.
[5] App privacy details on the App Store, 2021. https://developer.apple.com/app-store/app-privacy-details/.
[6] Appsflyer. https://dev.appsflyer.com/hc/docs/ios-sdk, 2023.
[7] Butler Labs. https://login.butlerlabs.ai/, 2023.
[8] Chartboost. https://answers.chartboost.com/en-us/articles/207657373, 2023.
[9] Chartboost privacy label guideline. https://answers.chartboost.com/en-us/articles/115001490451, 2023.
[10] clang. https://libclang.readthedocs.io/en/latest/_modules/clang/cindex.html, 2023.
[11] Clevertap SDK. https://developer.clevertap.com/docs/getting-started, 2023.
[12] Common iOS classes and object. https://developer.apple.com/documentation/technologies, 2023.
[13] CycloneDX. https://cyclonedx.org/docs/1.4/json/#services_items_data_items_classification, 2023.
[14] CycloneDX Extension. https://cyclonedx.org/use-cases/#properties--name-value-store, 2023.
[15] DHS Definition of data. https://www.dhs.gov/privacy-training/what-personally-identifiable-information, 2023.
[16] Dynatrace. https://www.dynatrace.com/support/help/platform-modules/digital-experience/mobile-applications/instrument-ios-app/customization/oneagent-sdk-for-ios, 2023.
[17] Dynatrace privacy label guideline. https://www.dynatrace.com/support/help/manage/data-privacy-and-security/data-privacy/user-privacy-for-ios, 2023.
[18] Frida API. https://frida.re/docs/javascript-api/#script, 2023.
[19] Google Play Data Safety. https://support.google.com/googleplay/android-developer/answer/10787469?hl=en, 2023.
[20] Instabug. https://docs.instabug.com/docs/ios-integration, 2023.
[21] Instabug privacy label guideline. https://docs.instabug.com/docs/app-privacy-details, 2023.
[22] iOS SDK Ranking. https://appfigures.com/top-sdks/all/all, 2023.
[23] Ironsource. https://developers.is.com/ironsource-mobile/ios/levelplay-starter-kit/, 2023.
[24] Ironsource privacy label guideline. https://developers.is.com/ironsource-mobile/ios/apples-privacy-questionnaire-answers-ironsource/, 2023.
[25] Javaparser. https://github.com/javaparser/javaparser, 2023.
[26] JSON Patch. https://jsonpatch.com/, 2023.
[27] LogRocket. https://docs.logrocket.com/reference/ios, 2023.
[28] LogRocket privacy label guideline. https://docs.logrocket.com/reference/apple-privacy-questionnaire, 2023.
[29] Mixpanel. https://developer.mixpanel.com/docs/ios-objective-c-quickstart, 2023.
[30] Mixpanel privacy label guideline. https://mixpanel.com/legal/app-store-privacy-details, 2023.
[31] mParticle. https://docs.mparticle.com/developers/sdk/ios/initialization/, 2023.
[32] mParticle privacy label guideline. https://docs.mparticle.com/developers/sdk/ios/ios14/, 2023.
[33] Network traffic monitor. https://www.telerik.com/fiddler, 2023.
[34] Privacy Manifest. https://developer.apple.com/documentation/bundleresources/privacy_manifest_files?language=objc, 2023.
[35] RevenueCat. https://www.revenuecat.com/docs/ios, 2023.
[36] RevenueCat privacy label guideline. https://www.revenuecat.com/docs/apple-app-privacy, 2023.
[37] SpaCy. https://spacy.io/, 2023.
[38] Start.io. https://support.start.io/hc/en-us/articles/360006012653-iOS-SDK-Integration-Standard-#step1ios, 2023.
[39] Wayback Machine - Internet Archive, 2023. https://archive.org/web/.
[40] Project Website. https://sites.google.com/view/colainewebsites, 2024.
[41] Noura Alomar and Serge Egelman. Developers say the darnedest things: Privacy compliance processes followed by developers of child-directed apps. *Proceedings on Privacy Enhancing Technologies*, 4(2022):24, 2022.
[42] 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.
[43] 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.
[44] Michael Backes, Sven Bugiel, and Erik Derr. Reliable third-party library detection in android and its security applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 356–367, 2016.
[45] 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.
[46] 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.
[47] 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.
[48] Yue Chen, Yulong Zhang, Zhi Wang, Liangzhao Xia, Chenfu Bao, and Tao Wei. Adaptive android kernel live patching. In *USENIX Security Symposium*, pages 1253–1270, 2017.
[49] Jonathan Crussell, Ryan Stevens, et al. Madfraud: Investigating ad fraud in android applications. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 123–134, 2014.
[50] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on android. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2187–2200, 2017.
[51] Yue Duan, Lian Gao, Jie Hu, and Heng Yin. Automatic generation of non-intrusive updates for third-party libraries in android applications. In *RAID*, pages 277–292, 2019.
[52] 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.
[53] Jie Huang, Nataniel Borges, Sven Bugiel, and Michael Backes. Up-to-crash: Evaluating third-party library updatability on android. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 15–30. IEEE, 2019.
[54] Apple Inc. xcodebuild. https://developer.apple.com/library/archive/technotes/tn2339/_index.html, 2023.
[55] Bodong Li, Yuanyuan Zhang, Juanru Li, Runhan Feng, and Dawu Gu. Appcommune: Automated third-party libraries de-duplicating and updating for android apps. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 344–354. IEEE, 2019.
[56] Li Li, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. An investigation into the use of common libraries in android apps. In *2016 IEEE 23Rd international conference on software analysis, evolution, and reengineering (SANER)*, volume 1, pages 403–414. IEEE, 2016.
[57] Menghao Li, Wei Wang, Pei Wang, Shuai Wang, Dinghao Wu, Jian Liu, Rui Xue, and Wei Huo. Libd: Scalable and precise third-party library detection in android markets. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 335–346. IEEE, 2017.
[58] 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.
[59] 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.
[60] Ziang Ma, Haoyu Wang, Yao Guo, and Xiangqun Chen. Libradar: Fast and accurate detection of third-party libraries in android apps. In *Proceedings of the 38th international conference on software engineering companion*, pages 653–656, 2016.
[61] Abraham H Mhaidli, Yixin Zou, and Florian Schaub. " we can't live without {Them!}" app developers' adoption of ad networks and their considerations of consumer risks. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 225–244, 2019.
[62] Microsoft. Playwright. https://playwright.dev/, 2023.
[63] 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.
[64] 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.
[65] 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.
[66] 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.
[67] 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.
[68] Jaebaek Seo, Daehyeok Kim, Donghyun Cho, Insik Shin, and Taesoo Kim. Flexdroid: Enforcing in-app privilege separation in android. In *NDSS*, 2016.
[69] Shashi Shekhar, Michael Dietz, and Dan S Wallach. Adsplit: Separating smartphone advertising from applications. In *USENIX security symposium*, volume 2012, 2012.
[70] 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.
[71] Mohammad Tahaei, Kopo M Ramokapane, Tianshi Li, Jason I Hong, and Awais Rashid. Charting app developers' journey through privacy regulation features in ad networks. *Proceedings on Privacy Enhancing Technologies*, 1:24, 2022.
[72] Haoyu Wang and Yao Guo. Understanding third-party libraries in mobile app analysis. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 515–516. IEEE, 2017.
[73] 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.
[74] Jice Wang, Yue Xiao, Xueqiang Wang, Yuhong Nan, Luyi Xing, Xiaojing Liao, JinWei Dong, Nicolas Serrano, Haoran Lu, XiaoFeng Wang, and Yuqing Zhang. Understanding malicious cross-library data harvesting on android. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 4133–4150. USENIX Association, August 2021.
[75] 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.
[76] Yue Xiao, Zhengyi Li, Yue Qin, Xiaolong Bai, Jiale Guan, Xiaojing Liao, and Luyi Xing. Lalaine: Measuring and characterizing non-compliance of apple privacy labels. 2023.
[77] Yue Xiao, Zhengyi Li, Yue Qin, Xiaolong Bai, Jiale Guan, Xiaojing Liao, and Luyi Xing. Lalaine: Measuring and characterizing {Non-Compliance} of apple privacy labels. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1091–1108, 2023.
[78] 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.
[79] Daoguang Zan, Bei Chen, Yongshun Gong, Junzhi Cao, Fengji Zhang, Bingchao Wu, Bei Guan, Yilong Yin, and Yongji Wang. Private-library-oriented code generation with large language models. *arXiv preprint arXiv:2307.15370*, 2023.
[80] Daoguang Zan, Bei Chen, Zeqi Lin, Bei Guan, Yongji Wang, and Jian-Guang Lou. When language model meets private library. *arXiv preprint arXiv:2210.17236*, 2022.
[81] Xueling Zhang, Xiaoyin Wang, Rocky Slavin, Travis Breaux, and Jianwei Niu. How does misconfiguration of analytic services compromise mobile privacy? In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1572–1583, 2020.
[82] Yuan Zhang, Jiarun Dai, Xiaohan Zhang, Sirong Huang, Zhemin Yang, Min Yang, and Hao Chen. Detecting third-party libraries in android applications with high precision and recall. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 141–152. IEEE, 2018.
[83] Shuyan Zhou, Uri Alon, Frank F Xu, Zhiruo Wang, Zhengbao Jiang, and Graham Neubig. Docprompting: Generating code by retrieving the docs. *arXiv preprint arXiv:2207.05987*, 2022.
[84] 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.
[85] 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.

# 8 APPENDIX

## 8.1 Finding privacy page URI keywords list

['privacy', 'privacy-labels', 'data-disclosure', 'app-store-connect-requirements', 'data-collection', 'data-types', 'data-safety', 'privacy-questionnaire', 'privacy-section','ios14', 'prepare-for-ios14'].

## 8.2 Binary classifiers comparison

**Table 5: Model Performance Comparison**

| Model | Precision | Recall | F1-score |
|---|---|---|---|
| (BLSTM) w. attention | **0.925** | **0.921** | **0.923** |
| (BLSTM) w/o attention | 0.878 | 0.895 | 0.886 |
| BiGRU based encoder w. attention | 0.905 | 0.868 | 0.886 |
| BiGRU based encoder w/o attention | 0.8889 | 0.842 | 0.865 |
| Logistic Regression with TF-IDF | 0.860 | 0.763 | 0.809 |

To determine the most effective model for identifying privacy configurations, we conducted a comparative experiment using five widely-used models. We trained the models on a dataset of 858 configurations and tested them on a separate set of 183 configurations. The comparison results are presented in Table 5.

## 8.3 Verb list associated with each op

**Table 6: Verb list associated with each op.**

| Operation | Verb list Example |
|---|---|
| Add | add, set, pass, copy, specify, use, call, enable, run |
| Remove | remove, dismiss, delete, drop, disable, comment out |
| Replace | replace, reset, update, change, override, modify |

Please see Table 6.

## 8.4 Retrieve last modified date

To investigate the updateability of privacy label disclosure guideline, we obtain its last modified date by using one of the following methods, depending on availability: (1) 65.42% of the dates are explicitly mentioned on the webpage; (2) 15.89% of the dates are determined by examining the HTTP header "Last modified date"; (3) 11.21% of the dates are retrieved from the last snapshot taken by the Internet Archive Wayback Machine [39]. The remaining 7.48% of dates are unknown and cannot be retrieved.

## 8.5 Other Materials

For additional content, including (1) Differences between iOS privacy labels and third-party libraries, (2) Comparison with prior consistency models, and (3) Examples of privacy-related configurations, please visit our website [40].