Enhancing Transparency and Accountability of TPLs with PBOM: A Privacy Bill of Materials

Yue Xiao
Indiana University Bloomington
& IBM Research
Yorktown Heights, New York, USA
xiaoyue@ibm.com

Adwait Nadkarni
William & Mary
Williamsburg, Virginia, USA
apnadkarni@wm.edu

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

Abstract

Third-party libraries (TPLs) are extensively integrated into mobile apps for functionalities such as analytics, advertising, app monetization, and single-sign-on. While these libraries enhance app capabilities, they also introduce privacy risks and compliance issues. Existing privacy disclosures for TPLs, including privacy policies, privacy label guidelines, and privacy manifests, often lack uniformity, fine granularity, and timeliness, and fail to comprehensively disclose TPL data practices. We propose the Privacy Bill of Materials (PBOM), inspired by the Software Bill of Materials (SBOM), to enhance transparency, traceability, and accountability of TPLs. Our contributions include designing PBOM specifications, creating an automated PBOM generation pipeline, and conducting case studies to demonstrate PBOM's effectiveness in improving TPL transparency and accountability.

CCS Concepts

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

Keywords

Short-form Privacy Disclosures; Third-party Libraries; Privacy Compliance Check; User Privacy; Supply Chain Security and Privacy

ACM Reference Format:
Yue Xiao, Adwait Nadkarni, and Xiaojing Liao. 2024. Enhancing Transparency and Accountability of TPLs with PBOM: A Privacy Bill of Materials . In *Proceedings of the 2024 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3689944.3696159

1 Introduction

Privacy violations and compliance issues in mobile apps are major concerns for users, developers, and regulators. These concerns are further exacerbated by the extensive integration of third-party libraries (TPLs)—such as those for analytics, advertising, app monetization, or single-sign-on functionalities—into the mobile supply chain. While these libraries enhance app functionalities, they also introduce significant privacy risks and compliance issues. 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 [35, 37, 41, 51, 67, 76, 89, 95–97, 100].

To alleviate these privacy risks, some privacy-conscious third-party vendors release privacy disclosure guidelines [16, 20, 21] to help app developers accurately and comprehensively specify privacy disclosures related to TPL data practices. These guidelines can be formatted as a set of privacy statements, represented as $S = \{s|s : (d, a, X)\}$, where d denotes a data item, a represents the data operation, and X indicates the associated configuration settings applied to d, such as whether the data operation can be disabled or enabled by specific configurations. By following these guidelines, app developers gain a better understanding of and control over the data collection and usage practices of the libraries used within their apps. However, recent studies [99] highlight that privacy statements in these guidelines are often inconsistent with actual data practices at the data, operation, and configuration levels. For example, some guidelines erroneously state that data is not collected under any configurations, while in reality, eight TPLs were found in [99] to collect data mandatorily at the code level. This inconsistency undermines the accountability of privacy label disclosure guidelines for downstream developers.

Moreover, the privacy disclosures of TPLs often vary significantly in their format and location, making it challenging for downstream app developers to navigate and accurately interpret these documents. Privacy label guidelines can appear in various formats, such as tables, images, standard text, and bullet points, and are often scattered across third-party vendor websites. These disclosures may be buried within extensive API documentation, hidden in support or help sections, and sometimes even placed in blogs or FAQs, where privacy-related content is not typically expected. The lack of uniformity can further undermine their transparency and limit their practical usefulness. To standardize these guidelines, industry leaders like Apple and Google have made significant strides to enhance the transparency of data collection practices from TPLs. Apple released the "privacy manifest" specification [23] for TPL vendors to disclose their data practices in December 2023. Subsequently, Google announced the "Data Safety form" [26], urging TPL vendors to publish privacy guidance for their users. However, those privacy format's design fails to account for configuration nuances, offering less granularity and not adequately reflecting the diverse data practices arising from different TPL configurations. Additionally, existing privacy documents have struggled to be adopted or lack practical and scalable implementation, which hinders their

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

ACM ISBN 979-8-4007-1240-1/24/10
https://doi.org/10.1145/3689944.3696159

effectiveness in ensuring privacy accountability throughout the mobile software supply chain.

To address this gap, we propose the design of a unified and fine-grained privacy-accountable disclosure for TPLs, called PBOM (Privacy Bills of Materials). Inspired by Software Bills of Materials (SBOM), a nested inventory listing the ingredients of software components, PBOM aims to enhance the transparency and accountability of TPLs in the software supply chain.

Contributions. We summarize our contributions as follows:

- We design unified, fine-grained, and compatible specifications for PBOM, enabling a comprehensive description of data practices performed by TPLs.
- We propose a generator pipeline that can automatically produce PBOMs for TPLs, facilitating efficient adoption and smooth interoperability.
- We conduct case studies of two TPLs to demonstrate how PBOM can enhance the transparency and accountability of TPLs.

2 Background

2.1 Bills of Materials

A Bill of Materials (BOM) is a detailed inventory of components, materials, and parts needed to build a product. In software, a Software Bill of Materials (SBOM) [24] lists all software components, libraries, and dependencies, enhancing transparency, security, and compliance. The SBOM concept gained prominence with Executive Order 14028 [15], issued by President Biden on May 12, 2021, which mandates developers to provide SBOMs to improve software supply chain security.

Inspired by SBOM, we propose the Privacy Bill of Materials (PBOM) to address privacy risks and compliance issues associated with third-party libraries (TPLs) in the mobile supply chain. While SBOM focuses on software components, PBOM provides a comprehensive and fine-grained disclosure of data practices by TPLs. PBOM details data items collected, data sources, purposes, configurations, actions, destinations, and procedures. By adopting and extending BOM principles to privacy practices, PBOM enhances transparency, traceability, and accountability in the mobile software supply chain. It helps app developers, end users, and regulators understand and manage the privacy implications of TPLs, thereby improving compliance and reducing legal risks.

2.2 Mobile TPL Data Practices

Configurable third-party libraries (TPLs) offer app developers the flexibility to tailor libraries to meet specific app functionality needs and privacy requirements. The configuration process typically involves using configurable APIs to set parameters that control how the TPL handles user data. Configurations that affect data usage practices are referred to as privacy configurations. For example, the privacy configuration API `setAnalyticsCollectionEnabled` allows developers to enable or disable data collection features. The ability to configure a TPL introduces diverse data usage practices, such as disabling default data collection or modifying data handling behaviors, which should be accurately reflected in the PBOM. Our PBOM generator (§ 4) analyzes these configurable data usage practices to provide a comprehensive and fine-grained disclosure of TPL data practices.

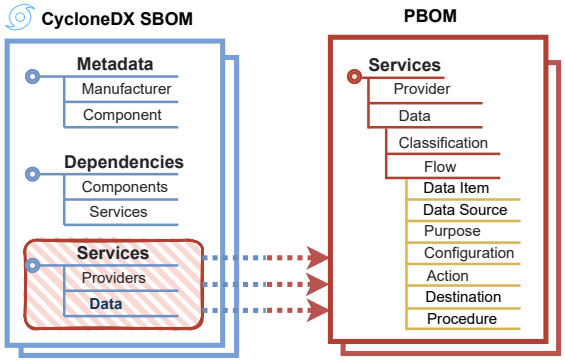

Figure 1: Specification Overview of PBOM.

3 PBOM Design

PBOM is a form of privacy disclosure designed to enumerate all data collection practices within software, aiming to ensure transparency and enhance data protection throughout the supply chain. In this section, we will illustrate the PBOM design goals in § 3.1, detail the PBOM specifications in § 3.2 and discuss its use cases in ensuring privacy assurance in § 3.3.

3.1 Design Goal

The design of PBOM should achieve high compatibility, fine granularity, and seamless transformability, with the aim of simplifying integration into existing SBOM tools, enhancing the transparency of privacy disclosures, and facilitating compliance checks.

- *Compatibility and Interoperability.* The PBOM format should seamlessly integrate with existing SBOM tools and workflows, allowing for efficient adoption and smooth interoperability.
- *Fine Granularity.* The privacy disclosure provided by PBOM should take into account the specific TPL product, version, and configuration, enabling developers visible to detailed factors that control data collection practices.
- *Transformability.* The field design of PBOM should be easily convertible into inputs for consistency models, thereby streamlining compliance checks and verification processes.

3.2 PBOM Design

To fully realize the benefits of such privacy disclosure, a unified and compatible schema is essential. We propose this privacy disclosure to be built upon the emerging software bill of materials (SBOM) standard, such as CycloneDX [17], 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 [18]. 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. However, rather than integrating the data collection inventory into the original SBOM, we design PBOM as an single

BOM file by extending the `services` field from the SBOM specification, as shown in Figure 1. The rationale behind this decoupling is that the inventory described in an SBOM typically remains static until the inventory changes. In contrast, data collection practices are much more dynamic and subject to change due to functionality updates, regulatory enforcement, and customer requests. This decoupling allows for the independent updating of data collection practices without the need to track and manage the original SBOMs.

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] and the flow direction (e.g., "inbound", "outbound")[2]. These elements alone are insufficient to fully disclose data collection practices. We propose adding seven properties to thoroughly express data flow, highlighted in the yellow block in Figure 1.

(❶) Data Item: This field specifies which data item is being collected. We follow the privacy data taxonomy defined by Google Play's Data Safety section [27], which categorizes 38 data types, such as Approximate location, Precise location, Email address, and Health info.

(❷) Data Source: This field explains the origin of the data. Data can come from Android system APIs, user inputs, or sensitive data from other TPLs.

(❸) Purpose: The purpose field is to help the customers understand how each data type is used by the software. The data collection purpose is required to be properly disclosed, according to Article 5 of the GDPR: *"the data that's collected must be for a specific and legitimate purpose and shouldn't be used in any way beyond that intention"*. Additionally, industry standards, such as Apple and Google, also require purpose disclosure in privacy label. Here, we focus on TPLs in Android, following the purpose taxonomy defined by Google privacy labels [27], which include: *"App functionality", "Analytics,Developer communications", "Advertising or marketing", "Fraud prevention, security, and compliance", "Personalization" and "Account management"*.

(❹) Configuration: TPLs usually provide app developers with configuration APIs to control data collection practices. Some dataflows can be triggered or disabled only when specific configurations are enforced. For example, location data by the Flurry library is only collected when `[Flurry trackPreciseLocation:Yes];` is configured. It is essential to incorporate such information into the disclosure of dataflow, as it makes data collection practices transparent to downstream app developers, giving them more leverage to control data collection practices when integrating such libraries and better protecting end user privacy.

(❺) Action: With configuration enforcement, data collection actions become configurable, rather than being simply collected or not collected. The taxonomy of Action includes four operation values $\{Y, D, E, N\}$, where Y (Yes) represents compulsive collection, i.e., collecting at any time; D (Disablable) indicates that the TPL collects data by default, but the collection can be disabled by certain configurations; E (Enablable) signifies that the TPL does not collect data by default, but the collection can be enabled by certain configurations; and N (No) represents compulsive no-collection, i.e., not collecting at any time.

(❻) Destination: The Destination field in the PBOM is essential for providing transparency about where user data ultimately resides, whether it is transmitted to external servers or stored locally on the device. There are two types of destinations: (1) user data transmitted out of the device to an external URL, and (2) user data preserved on the device as files [25], in key-value pairs [31], in a local database [30], or in external storage (such as `/sdcard`) [28]. If we detect data being accessed but neither type of destination is identified, we assign `None` to this field, indicating the data is being used in memory and not stored or transmitted anywhere. Such information enables more precise and fine-grained privacy compliance analysis, as different regulations and standards have varying requirements regarding where data ultimately ends up, thereby enabling better control and protection of user data.

(❼) Procedure: The Procedure field describes whether the data is being protected (e.g., de-identification or encryption) before storage or transmission. Article 4(5) of the GDPR requires companies and organizations to implement techniques or procedures to lower the risk of potential data breaches and safeguard personal data. Therefore, it is important to know whether dataflows utilize these techniques, as such information provides an additional layer of data protection.

3.3 PBOM Use Cases

The potential use cases of PBOM 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 reduce privacy risk and ensure better compliance, and (3) Regulatory body can fit PBOM into existing consistency models [37, 39, 89, 98, 99, 105] to audit privacy compliance to further achieve its privacy and accountability objectives.

❶ Ensure Continuous Compliance for TPL vendors. Traditional privacy disclosures often lack timeliness and are overly generalized. For example, privacy policies are typically updated every few months [92] or even years and generally cover all privacy practices for all products from a vendor. While privacy label guidance [3, 5, 12] is more specific, governing all versions of a particular product, it still falls short as different versions may have varying data collection practices [82]. The PBOM addresses these issues by providing more granular and timely updates. TPL vendors can integrate PBOM into their CI/CD pipelines, ensuring the continuous generation and release of up-to-date privacy disclosures for every TPL version. This integration ensures that privacy disclosures are both timely and precise, reflecting the most current data collection practices for each version of the product.

❷ Reduce Privacy Risk for Downstream Customers. By reviewing TPLs' PBOMs before integration, downstream customers can proactively identify, assess, and mitigate privacy risks associated with third-party libraries. This preemptive review allows organizations to understand data collection practices and make informed integration decisions. PBOMs facilitate the implementation of data

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

[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

minimization strategies by enabling customers to disable unnecessary data collection through configuration options. This reduces the amount of personal data collected, stored, and processed, enhancing privacy protection and compliance with regulatory requirements

❸ **Facilitate Privacy Auditing for Regulatory Body.** The PBOM can be seamlessly integrated into tools that automate privacy compliance checks to detect privacy violations and assess the potential privacy risks of software components. The representation of data flow in PBOM can be easily converted into a tuple-based format (i.e., *(data, purpose, action, configuration)*), which can be directly input into consistency models for compliance checks. Additionally, it covers all the necessary fields to perform comprehensive privacy compliance checks. For example, PBOM can be directly integrated with data-level compliance check tools [89, 105], which check whether certain data is disclosed in the policy. It is also applicable to entity-sensitive consistency models (e.g., PoliCheck [37]), which take into account the entities (third-party vs. first-party) involved in personal data collection. Furthermore, PBOM is adaptable to more complex consistency models by considering the *"purpose"* attribute, such as PurPliance [39] and Lalaine [98], which consider the purpose of data collection. Finally, PBOM includes configuration information that controls data flow, allowing for more fine-grained consistency models like Colaine [99], which considers configuration when measuring the compliance of privacy label guidance disclosures provided by third-party SDKs.

4 PBOM Engine: A PBOM Generation Plugin

In this section, we detail the design and conceptual pipeline of *PoGen*, a proposed tool for automatically generating PBOMs for TPLs. Figure 2 illustrates the architecture of *PoGen*, which comprises three phases: (❶) Preparation Phase: Configured Wrapper App Creation, (❷) Analysis Phase: UI, Static, and Dynamic Analysis, and (❸) Inference Phase: PBOM Generation.

In the preparation phase (❶), *PoGen* first collects API documentation of configurable TPLs to extract configuration descriptions and associated code snippets. Next, *PoGen* employs NLP techniques to identify privacy-related configurations, parsing their semantics to generate machine-readable configuration patches. 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. Each configuration patch is then applied to create configured wrapper apps that integrate the TPL under specific configuration settings. In the analysis phase (❷), *PoGen* uses these configured wrapper apps to perform UI, static, and dynamic analysis to thoroughly investigate the data collection practices of the TPL under each specific configuration. UI analysis identifies sensitive user inputs by analyzing UI-related resource files. Static analysis uses source and sink APIs to capture call traces of data flow. Dynamic analysis records network traffic during app execution to capture dynamic features. In the inference phase (❸), the analysis inputs (configuration patches, source APIs, sink APIs) and outputs (sensitive user inputs, call traces, network traffic) are used to infer the seven fields defined in the PBOM (e.g., data item, data source, purpose, configuration, action, destination, procedure).

4.1 Configured Wrapper App Creation

To fully analyze the TPL, we first need to create a wrapper app that integrates this TPL under specific configurations. This process comprises two primary components: (1) Privacy Configuration Patch Generation and (2) Configuration Patch Enforcement.

4.1.1 Privacy Configuration Patch Generation. We begin by collecting API documentation of configurable TPLs to generate configuration patches using the same JSON format as in [99]. This format specifies the semantics of a privacy configuration setting, including the `operation` to enforce the configuration, the `value` of the configuration, and the `path` to add the configuration setting. Using the Playwright library [73], we automate full-page screenshots of the API documentation for each TPL. These screenshots are fed into an OCR model to extract configuration descriptions and code snippets. We then fine-tune a privacy configuration classifier in [99] with Android TPL API documents to identify privacy-related configurations impacting TPL data collection and usage behavior. Utilizing NLP techniques, we retrieve the semantic information of each privacy configuration and generate machine-readable configuration patches for different settings.

4.1.2 Configuration Patch Enforcement. We begin by setting up a default wrapper app that integrates the target TPL without any configuration. This process involves integrating the TPL, installing dependencies, registering for TPL developer accounts, initializing the TPL, signing the app with an Android Developer account, and compiling the app. Once the default wrapper app is prepared, we enumerate each configuration patch to generate multiple apps with different configuration settings. The configuration patch is applied to the default wrapper app, resulting in a configured wrapper app that enforces the specific configuration. These configured wrapper apps are then used as inputs for UI and program analysis, enabling a thorough investigation of how different configurations impact data collection and handling practices.

4.2 UI & Static & Dynamic Analysis

In the analysis phase, *PoGen* thoroughly investigates the data collection practices of TPLs using UI, static, and dynamic analysis.

4.2.1 UI Analysis. To perform UI analysis for identifying sensitive user input in the configured wrapper app, we follow a structured approach in literature [34, 54, 75, 77]. First, we decode the configured wrapper app using a tool like "apktool" [2] to extract UI-related resource files, including layout files (`res/layout`) and text resources (`res/values/strings.xml`). Next, we preprocess these texts by splitting words, removing redundancies, and applying stemming. We gathered a set of sensitive UI elements by utilizing an existing ontology [36, 37] that provides subsumptive relations between low-level technical terminology and high-level privacy terms. Specifically, if privacy-label data items subsume the data value returned by the text of UI element, then this UI element is regarded as a candidate which handle sensitive data. Finally, those identified UI elements that carry user sensitive data serve as taint sources, facilitating static code analysis to trace data flows.

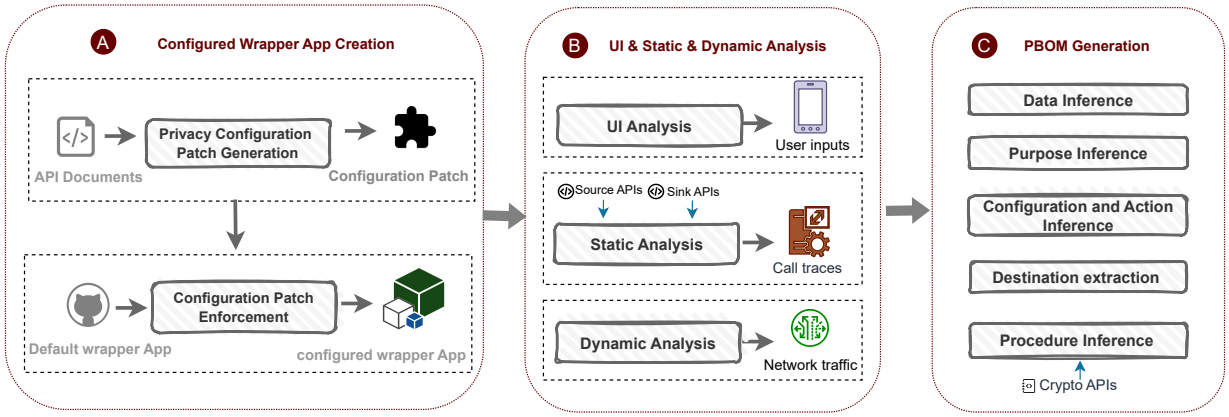

Figure 2: Overview of PBOM Implementation.

4.2.2 Static Analysis. We perform static analysis to capture call traces of data collection practices. To achieve this, we define the taint sources and sinks as follows:

• *Data Source.* The tainted sources include three components: (1) sensitive UI elements handling user inputs, identified through UI analysis; (2) sensitive system APIs that return user data. To compile a set of sensitive system APIs, we utilize an existing ontology [36, 37, 70, 98] to gather candidate APIs. Specifically, if privacy data items subsume the data value returned by an API, this API is considered a candidate; (3) third-party APIs that carry sensitive user data (e.g., `getCurrentAccessToken()` from the Facebook sign-on SDK). We reuse a Meta-DB from [94], which records 1,094 sensitive APIs from the top 40 third-party libraries, covering 91% of Google Play apps.

• *Data Sink.* Collected user data can either be stored locally on the device or transmitted externally. We categorize sink APIs into two types: storage-related and network-related. For storage APIs, we focus on classes and methods for reading from and writing to files, databases, and shared preferences, such as `File`, `FileOutputStream`, `SQLiteDatabase`, and `SharedPreferences`. For network APIs, we identify classes and methods responsible for network communications, examining built-in classes like `HttpURLConnection` and popular third-party libraries like OkHttp [8].

Finally, with the source and sink APIs collected, we perform static analysis to obtain call traces data. Using taint analysis [64], we trace data flow from sources to sinks, examining control and data flow within the application.

4.2.3 Dynamic Analysis. Our pipeline automatically installs each app using the *Android Debug Bridge (adb)* command [1] and schedules it to run on a set of rooted Android devices. To facilitate dynamic execution, we utilize an open-source UI execution tool called *nosmoke* [14], which generates actions and automatically triggers the app's UI interactions, such as clicks or swipes. This approach is consistent with common practices in Android app analysis [62, 69, 78, 82, 84]. For network monitoring, we employ *Fiddler* [22], a popular network monitoring tool capable of TLS decryption and handling common decoding schemes. Similar to previous studies [56, 79, 82–84], Fiddler allows us to decrypt and inspect app traffic, providing detailed insights into data transmission practices. By analyzing the captured network traffic, we can identify any de-identification techniques applied by the TPLs and the endpoints to which the data is transmitted. Besides, this network traffic analysis also aids in inferring the purpose of the data collection.

4.3 PBOM Generation

In this section, we detail the inference of the seven fields in the PBOM using the inputs and outputs from the analysis phase.

4.3.1 Data Inference. User privacy data can originate from three sources: ❶ Sensitive User Inputs, referred to as *User-Input Privacy Data*, ❷ Sensitive System APIs, referred to as *System-Centric Privacy Data*, and ❸ Third-party APIs, referred to as *Cross-TPLs Privacy Data*.

• *User-Input Privacy Data.* The content entered by users into a mobile app through its user interface (UI), such as credit card information, usernames, and passwords, can be highly sensitive. We obtained such User-Input Privacy Data through the app's UI analysis in § 4.2.1. In these cases, the `data_item` is the text from the identified sensitive UI element, and the `data_source` is the identified UI element.

• *System-Centric Privacy Data.* The operating system (OS) can return sensitive user data, such as GPS locations, through system APIs like `getLastKnownLocation()`. We infer such System-Centric Privacy Data by retrieving the return values of a set of sensitive system APIs from § 4.2.2. In these cases, the `data_item` is the API return value, and the `data_source` is the sensitive system API.

• *Cross-TPLs Privacy Data.* Third-party libraries can harvest data from other TPLs that handle sensitive user data (e.g., Facebook SDK). To cover such cases, we reuse a Meta-DB from [94], which records 1,094 API specifications and metadata of the top 40 third-party libraries. For each API, Meta-DB records the data it returns (e.g., session token, page likes, user ID, profiles, groups followed). In these cases, the `data_item` is the data returned by the API, and the `data_source` is the other TPL and the accessed API.

4.3.2 Purpose Inference. In our study, we analyze a set of high-profile TPLs by comprehensively reviewing their API documentation and investigating static and dynamic call traces, as well as

network traffic data, to identify features relevant for purpose prediction. We specifically select features tailored to the purpose categories and definitions provided by Google. These features are considered robust, meaning that missing these features might prevent the classifier from correctly differentiating between different purposes. To this end, we reused four features from MobiPurpose [56] and Lalaine [98], and introduced four new features. The features are categorized into three groups: *internal SDK features*, *call traces features*, and *traffic features*, as shown in Table 1. Further, we apply PyCaret [11] with Python 3.6 [91], an open-source machine learning package deployed with 18 algorithms, to select the best model as well as the hyper-parameters for purpose identification.

Table 1: Feature Selection

Group	Feature	Source
Internal SDK Features	Configuration description	SDK API Documentation
	Configuration API	
Call Traces Features	Function in Call Traces	Static Analysis
	Sensitive System API	
	Privacy-related UI Element	
Traffic Features	Domain Name	Dynamic Analysis
	URL Paths	
	KV Pairs	

4.3.3 Configuration and Action Inference. The configuration API can be directly obtained through the `value` field in the configuration patch generated in § 4.1.1. To infer the action of data practices, we follow this logic: If a dataflow occurs only when a specific configuration is enabled (and does not happen by default), then the action is `E` (Enablable by developer). If the configuration is disabled by default but can be enabled by the developer, the dataflow does not occur initially, and the action is `D` (Disablable by developer). If the dataflow always occurs regardless of the configuration, the action is `Y` (Always collected).

4.3.4 Destination extraction. Based on data end-up states, data can be categorized into three types: *Data in Use*, *Data at Rest*, and *Data in Transit*. *Data in Use* refers to data accessed by an SDK in memory without being saved or transmitted. In such cases, the destination will be assigned as `None`. *Data at Rest* refers to information stored on the device. In these cases, we resolve the location of data storage, such as databases, files, or external storage. *Data in Transit* means data is transmitted off the device and sent to a server. The network endpoint would be the destination.

• *Data in Use:* If a data flow from the entry point of the SDK to sensitive system/third-party APIs or user input elements is detected, but no flow to sink APIs is found, then the data is only in use and does not flow to the device or network, and the destination is `None`.

• *Data at Rest:* If the sink APIs in the data flow are storage APIs, we further resolve the specific files, local databases, or SharedPreferences where the data is stored. We use pattern matching to identify classes and methods related to file storage, such as `File`, `FileOutputStream`, and `FileWriter`. By tracing the initialization of File objects, we determine the file paths and names used for storing data. For a local database, we parse SQL strings to identify where data is stored. For SharedPreferences, we detect its usage by tracking methods such as `getSharedPreferences`, `putString`, and `putInt` to identify the preference file names and keys used for storing user data.

• *Data in Transit:* To resolve the specific network endpoints where user data is sent when a network API is identified as a sink, we utilize state-of-the-art tools [106] capable of statically resolving string values in Android apps (using a value set analysis approach, with backward slicing and string-related operation analysis). For endpoints whose values cannot be resolved statically due to runtime contexts (e.g., downloading additional code from the cloud/server [53], dynamically loading endpoints [94]), we perform dynamic analysis to monitor network traffic and capture HTTP traffic that transmits the data. Dynamic analysis can suffer from code coverage problems. Hence, for better coverage, if the endpoints cannot be resolved, we will put "TBD" in this field and rely on a manual process to validate the results.

4.3.5 Procedure inference. We mainly focus on two types of data processing (Data encryption and Data de-identification) before storage or transmission.

• *Encryption:* Data encryption is one of the most important ways to protect data used, stored, or transmitted in a mobile app. It can prevent unauthorized parties from reading private, confidential or sensitive data. To detect whether data is encrypted, we first summarized commonly used cryptographic libraries and APIs, as discussed in Literature [40, 47, 74, 88, 103], such as `javax.crypto` and `java.security`, focusing on classes and methods related to encryption, including `Cipher`, `SecretKeySpec`, and `MessageDigest`. Next, we perform taint analysis to trace the flow of data from sources (user inputs, sensitive data) to sinks (storage and network APIs). We use pattern matching to detect encryption and decryption operations, such as calls to `Cipher.getInstance()`, `cipher.init()`, and `cipher.doFinal()`. By building a call graph and systematically analyzing method calls, we verify whether encryption is applied before data is used, stored, or transmitted. If cryptographic operations are detected, we will assign "Encryption" to *Protection* field.

• *De-identification:* We determine whether de-identification techniques are applied by inspecting the network traffic. Specifically, we look for evidence of de-identification techniques such as hashing, pseudonymization, and data masking. Hashing involves transforming sensitive data, like personal identifiers, into fixed-length hash values using cryptographic functions (e.g., SHA-256), which cannot be easily reversed. Pseudonymization replaces original identifiers (e.g., names, Social Security numbers) with unique pseudonyms or tokens, allowing data linkage without revealing personal information. Data masking obscures parts of sensitive data, such as displaying only the last four digits of a credit card number or replacing characters in an email address with asterisks, ensuring that intercepted data cannot be used to identify individuals.

5 Case Study

Following the proposed pipeline in § 4 , we present use cases that demonstrate how data transparency and accountability requirements can be expressed by PBOM. We first introduce a use case that employs PBOM to express configurable data usage practices

for TPLs in mobile apps. We then show how to use PBOM to express cross-TPLs data processing, aiming to address compliance requirements for third-party data sharing.

5.1 Expressing Mobile App with Configurable TPL

In this section, we present a case study on the *Radar SDK*, a full-stack location infrastructure for various products and services. Listing 1 provides an example of PBOM for Radar SDK with version 3.8.0, which extends the SBOM format specified by *CycloneDX 1.4*.

- *Data Item:* The Radar library collects `Precise Location`, which refers to a physical location with an accuracy of at least 3 square kilometers.
- *Data Source:* The `Precise Location` is obtained from the Android system API: `getLastKnownLocation()`.
- *Purpose:* The Radar SDK provides location services to app developers to enhance their app functionality. This includes adding features such as geofencing, location tracking, trip tracking, geocoding, and search to the apps.
- *Configuration:* The Radar library allows downstream developers (i.e., app developers) to configure the tracking of users' locations in the background. By enabling the configuration `Radar.startTracking(RadarTrackingOptions.RESPONSIVE);` [29], the Radar library collects precise location data by detecting whether the device is stationary or moving. When moving, the library sends location updates to the server every 2-3 minutes.
- *Action:* Precise location tracking is configurable by the downstream developer. Therefore, the collection action here is `E`, meaning this data collection is not enabled by default and must be activated by the downstream developer. The developer can decide whether enabling this feature satisfies the app's functionality requirements.
- *Destination:* The precise location data is transmitted out of the device and sent to the SDK server at `https://api.radar.io/`.
- *Procedure:*. The raw precise location data is collected without any processing before transmission. Through the call trace between `getLastLocation()` [19] and network API `UrlRequest ()` [32].

In our preliminary study, we observed that 10.67% of apps integrating the Radar library had compliance issues with their privacy label disclosures. Using Lalaine [98], an automatic privacy label compliance check pipeline, we investigated 150 apps that integrated the Radar library. Among these, 16 apps had configured location tracking in Radar but failed to disclose the collection of precise location data. The release of a PBOM alongside the Radar library can significantly enhance transparency and accountability in data collection practices. PBOM clearly outlines the specific configurations and data collection methods employed by the Radar library, providing detailed information on data types, sources, purposes, and actions. By including such comprehensive documentation, app developers gain greater leverage over data collection settings, enabling them to disable tracking features if needed to protect their end-users' privacy.

5.2 Expressing Cross-TPLs Data Processing Operations

In this section, we present a case study on *Mobiburn* [6], a marketing library that provides betting strategies for mobile marketers.

Listing 1: PBOM Example for Radar library

```
{
  "bomFormat": "CycloneDX", "specVersion": "1.4",
  "services": [{
    "provider": {"name": "Radar", "url": ["https://radar.com/"]},
    "name": "Radar Android SDK", "version": "3.8.0",
    "data": [{
      "classification": "PII",
      "flow": [{
      "properties": [
        {"name": "Data_Item","value": "Precise Location"},
        {"name": "Data_Source", "value": "System API: getLastKnownLocation()"}
        {"name": "Purpose", "value": "App functionality"},
        {"name": "Configuration", "value": "Radar.startTracking(RadarTrackingOptions.RESPONSIVE);"},
        {"name": "Action", "value": "E (enabled by configuration)"},
        {"name": "Destination", "value": "https://api.radar.io/"},
        {"name": "Procedure", "value": "None"}]} ]}
    ]}
}
```

Mobiburn performs cross-library data harvesting from other SDKs (i.e., Facebook SDK) [94]. Listing 2 provides an example of PBOM for the Mobiburn library, version `v1.5.3`, which extends the SBOM format specified by *CycloneDX 1.4*.

- *Data Item:* Mobiburn strategically harvests user data from the Facebook SDK[3], a social media library that is extensively used by apps for single sign-on and carries user personal information. Specifically, Mobiburn harvests personal info from Facebook end users, including *id, first_name, gender, last_name, link, locale, timezone, updated_time, verified_email*.
- *Data Source:* The data is sourced from the Facebook SDK. Specifically, the Mobiburn library fetches a user's Facebook personal information (ID, name, gender, email, locale, link, etc.) by calling `getFbProfile()` after retrieving the user's Facebook access token. This is done by invoking the function `com.facebook.AccessToken.getToken()` through reflection techniques.
- *Purpose:* Mobiburn, as a marketing company, collects personal information from Facebook end-users to provide marketing services to other data suppliers and marketers. Therefore, the purpose of collecting personal info is `Advertising or marketing`.
- *Configuration:* The configuration field is `None`. The Mobiburn automatically initiates the collection of personal data from Facebook without any configurations.
- *Action:* The collection action is "`Y: Always collected`", as there is no configuration for app developers to control this data-harvesting behavior.
- *Destination:* The personal information is transmitted off the device and sent to the Mobiburn server at `https://api.mobiburn.com/`.

[3]The terms "SDK" and "TPL" can be used interchangeably.

Listing 2: PBOM Example for Radar library

```
{
  "bomFormat": "CycloneDX", "specVersion": "1.4",
  "services": [{
    "provider": {"name": "Mobiburn", "url": ["https://www.
     mobiburn.com/#/"]},
    "name": "Mobiburn Android SDK", "version": "v1.5.3",
    "data": [{
      "classification": "PII",
      "flow": [{
      "properties": [
        {"name": "Data_Item", "value": "Personal Info"},
        {"name": "Data_Source", "value": "Facebook:
     getFbProfile(String token); com.facebook.AccessToken.
     getToken()"}
        {"name": "Purpose", "value": "Advertising or
     marketing"},
        {"name": "Configuration", "value": "None"},
        {"name": "Action", "value": "Y (always collected)"},
        {"name": "Destination", "value": " https://api.
     mobiburn.com/"},
        {"name": "Procedure", "value": "RSA Encryption"}]}
    ]}
}
```

• *Procedure:* Mobiburn employs encryption before sending data over the Internet, specifically using RSA encryption implemented by `com.mobiburn`.

The data harvesting behavior from one TPL to the other third-party SDK can violate the terms and conditions of the data provider. In such case, Mobiburn's data harvesting from the Facebook SDK strongly violated Facebook's data sharing policy in their Terms of Service (ToS) [4], leading to legal action where Facebook banned and sued Mobiburn in August 2020 [13]. If Mobiburn had released a PBOM along with its library, the data collection practices would have been clearly documented, potentially preventing such violations. PBOM can enhance transparency by detailing data collection, sources, purposes, configurations, actions, destinations, and procedures. This comprehensive documentation allows stakeholders to verify compliance with data sharing policies and legal requirements. Moreover, by providing a clear and accountable disclosure of data practices, PBOM helps companies mitigate legal risks, ensuring that they adhere to regulatory standards and avoid potential lawsuits.

6 Related Works

Security and Privacy Analysis of TPLs. The literature extensively examines security and privacy risks associated with third-party libraries (TPLs) in mobile apps. Various detection techniques [38, 42, 45, 65, 66, 71, 93, 104] and isolation techniques [68, 86, 87] have been proposed to measure the integration and dependencies of TPLs. To assess privacy leakage caused by TPLs, researchers have utilized both static [76] and dynamic [44, 80, 82] program analysis, revealing the widespread collection of sensitive user data. Additionally, researchers have identified outdated library versions as a significant privacy risk factor and proposed auto-update techniques to mitigate this issue [46, 55, 63]. Misconfigurations by app developers have also been shown to result in the leakage of sensitive personal information [33, 72, 84, 90, 102]. Unlike previous studies, our work proposes PBOM, a novel solution designed to enhance the transparency, traceability, and accountability of TPLs in the mobile ecosystem. PBOM aims to mitigate privacy risks by providing a unified and fine-grained disclosure of TPL data practices, thereby serving as a proactive defense mechanism against privacy violations.

Privacy Labels and Other Short-Form Privacy Disclosures. The Platform for Privacy Preferences Project (P3P) [10], as studied in works such as [43, 48, 52, 81], served as a predecessor to privacy labels by allowing websites to express their privacy practices in a standard machine-readable format [9]. Privacy nutrition labels have been proposed and studied extensively in the literature [7, 49, 50, 57–59], aiming to provide clear and concise information about privacy practices. Previous research has conducted user studies to understand challenges from both developers' perspectives [67] and end-users' perspectives [101]. Recent studies [60, 61, 85, 98], have also examined the privacy labels of apps, questioning their accuracy and comprehensiveness. Those study highlights that creating accurate privacy labels is challenging for developers due to the complexity of interpreting data flows, which often involve undocumented data types, purposes, and configurations. The lack of standardized terminology and formats across TPLs further complicates the process, making it difficult to produce consistent and clear disclosures. Additionally, frequent updates to TPLs require ongoing revisions to privacy labels, leading to inaccuracies that undermine their effectiveness in communicating data practices. In our study, we address these shortcomings by proposing PBOM, a comprehensive and fine-grained privacy disclosure mechanism that extends beyond traditional privacy labels. PBOM aims to enhance transparency and accountability by providing detailed information about data sources, data items, configurations, actions, destinations, and procedures, thereby improving the accuracy and effectiveness of privacy disclosures in the mobile ecosystem.

7 Conclusion

This paper introduces the Privacy Bill of Materials (PBOM) to address privacy and compliance issues in mobile apps using third-party libraries (TPLs). PBOM provides detailed and structured privacy disclosures, enhancing TPL transparency and accountability. Our case studies on Radar SDK and Mobiburn illustrate how PBOM documents data collection practices, sources, purposes, configurations, actions, destinations, and procedures. PBOM empowers app developers to make informed decisions, protecting user privacy and reducing legal risks, thus improving privacy accountability in the mobile software supply chain.

8 Acknowledgment

We would like to express our gratitude to the anonymous reviewers for their valuable and constructive feedback. This work is supported in part by the National Science Foundation (CNS-1850725, 2343618) and Luddy Faculty Fellowship.

References

[1] Android Debug Bridge (adb). https://developer.android.com/tools/adb.
[2] apktool. https://apktool.org/.
[3] Branch SDK guidance. https://help.branch.io/using-branch/docs/answering-the-app-store-connect-privacy-questions.
[4] Facebook Terms of Use. https://www.facebook.com/help/581066165581870.
[5] Mob SDK Guidance. https://www.mob.com/wiki/detailed?wiki=288&id=172.
[6] Mobi Burn: Enabling Mobile Marketing Betting. https://www.mobiburn.com/#/.
[7] Mobile-App Privacy Nutrition Labels Missing Key Ingredients for Success. https://cacm.acm.org/magazines/2022/11/265814-mobile-app-privacy-nutrition-labels-missing-key-ingredients-for-success/fulltext.
[8] okhttp. https://square.github.io/okhttp/.
[9] P3p. https://www.w3.org/TR/P3P/.
[10] Platform for Privacy Preferences Project. https://www.w3.org/P3P/.
[11] PyCaret. https://www.kaggle.com/code/caesarmario/credit-card-approval-prediction-w-pycaret?scriptVersionId=95713769.
[12] Sentry SDK guidance. https://docs.sentry.io/product/security/mobile-privacy/.
[13] Taking Legal Action Against Those Who Abuse Our Platform. https://about.fb.com/news/2020/08/taking-legal-action-against-those-who-abuse-our-platform/.
[14] UI automation tool. https://github.com/macacajs/NoSmoke.
[15] Improving the nation's cybersecurity: Nist's responsibilities under the may 2021 executive order. https://www.nist.gov/itl/executive-order-14028-improving-nations-cybersecurity, 2021.
[16] Chartboost privacy label guideline. https://answers.chartboost.com/en-us/articles/115001490451, 2023.
[17] CycloneDX. https://cyclonedx.org/docs/1.4/json/#services_items_data_items_classification, 2023.
[18] CycloneDX Extension. https://cyclonedx.org/use-cases/#properties--name-value-store, 2023.
[19] Location Request. https://developer.android.com/develop/sensors-and-location/location/request-updates, 2023.
[20] LogRocket privacy label guideline. https://docs.logrocket.com/reference/apple-privacy-questionnaire, 2023.
[21] mParticle privacy label guideline. https://docs.mparticle.com/developers/sdk/ios/ios14/, 2023.
[22] Network traffic monitor. https://www.telerik.com/fiddler, 2023.
[23] Privacy Manifest. https://developer.apple.com/documentation/bundleresources/privacy_manifest_files?language=objc, 2023.
[24] Software Bill of Materials (SBOM). https://www.cisa.gov/sbom, 2023.
[25] Data and file storage overview. https://developer.android.com/training/data-storage, 2024.
[26] Google Play's Data safety section. https://support.google.com/googleplay/android-developer/answer/10787469?hl=en#zippy=%2Coptional-format-for-sdks, 2024.
[27] Google Play's Data safety section. https://support.google.com/googleplay/android-developer/answer/10787469?hl=en#zippy=%2Cpurposes, 2024.
[28] Manage all files on a storage device. https://developer.android.com/training/data-storage/manage-all-files, 2024.
[29] Radar SDK Documentation. https://radar.com/documentation/sdk/android, 2024.
[30] Save data in a local database using Room. https://developer.android.com/training/data-storage/room, 2024.
[31] Save simple data with SharedPreferences. https://developer.android.com/training/data-storage/shared-preferences, 2024.
[32] UrlRequest. https://developer.android.com/develop/connectivity/cronet/reference/org/chromium/net/UrlRequest, 2024.
[33] 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.
[34] Benjamin Andow, Akhil Acharya, Dengfeng Li, William Enck, Kapil Singh, and Tao Xie. Uiref: analysis of sensitive user inputs in android applications. In *Proceedings of the 10th acm conference on security and privacy in wireless and mobile networks*, pages 23–34, 2017.
[35] 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.
[36] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. PolicyLint: Investigating internal privacy policy contradictions on google play. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 585–602, Santa Clara, CA, August 2019. USENIX Association.
[37] 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.
[38] 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.
[39] 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.
[40] Alexia Chatzikonstantinou, Christoforos Ntantogian, Georgios Karopoulos, and Christos Xenakis. Evaluation of cryptography usage in android applications. *EAI Endorsed Transactions on Security and Safety*, 3(9):83–90, 2016.
[41] 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.
[42] 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.
[43] Lorrie Faith Cranor, Serge Egelman, Steve Sheng, Aleecia M McDonald, and Abdur Chowdhury. P3p deployment on websites. *Electronic Commerce Research and Applications*, 7(3):274–293, 2008.
[44] 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.
[45] 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.
[46] 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.
[47] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 73–84, 2013.
[48] Serge Egelman, Lorrie Faith Cranor, and Abdur Chowdhury. An analysis of p3p-enabled web sites among top-20 search results. In *Proceedings of the 8th international conference on Electronic commerce: The new e-commerce: innovations for conquering current barriers, obstacles and limitations to conducting successful business on the internet*, pages 197–207, 2006.
[49] Pardis Emami-Naeini, Yuvraj Agarwal, Lorrie Faith Cranor, and Hanan Hibshi. Ask the experts: What should be on an iot privacy and security label? In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 447–464. IEEE, 2020.
[50] Pardis Emami-Naeini, Henry Dixon, Yuvraj Agarwal, and Lorrie Faith Cranor. Exploring how privacy and security factor into iot device purchase behavior. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.
[51] 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.
[52] Julia Gideon, Lorrie Cranor, Serge Egelman, and Alessandro Acquisti. Power strips, prophylactics, and privacy, oh my! In *Proceedings of the Second Symposium on Usable privacy and security*, pages 133–144, 2006.
[53] Michael C Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pages 101–112, 2012.
[54] Jianjun Huang, Zhichun Li, Xusheng Xiao, Zhenyu Wu, Kangjie Lu, Xiangyu Zhang, and Guofei Jiang. {SUPOR}: Precise and scalable sensitive user input detection for android apps. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 977–992, 2015.
[55] 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.
[56] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I Hong. Why are they collecting my data? inferring the purposes of network traffic in mobile apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):1–27, 2018.
[57] Patrick Gage Kelley, Joanna Bresee, Lorrie Faith Cranor, and Robert W Reeder. A" nutrition label" for privacy. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, pages 1–12, 2009.
[58] Patrick Gage Kelley, Lucian Cesca, Joanna Bresee, and Lorrie Faith Cranor. Standardizing privacy notices: an online study of the nutrition label approach. In *Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 1573–1582, 2010.
[59] Patrick Gage Kelley, Lorrie Faith Cranor, and Norman Sadeh. Privacy as part of the app decision-making process. In *Proceedings of the SIGCHI conference on*

human factors in computing systems, pages 3393–3402, 2013.

[60] Rishabh Khandelwal, Asmit Nayak, Paul Chung, and Kassem Fawaz. Unpacking privacy labels: A measurement and developer perspective on google's data safety section. *arXiv preprint arXiv:2306.08111*, 2023.

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

[62] Kyungmin Lee, Jason Flinn, Thomas J Giuli, Brian Noble, and Christopher Peplin. Amc: Verifying user interface properties for vehicular applications. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 1–12, 2013.

[63] 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.

[64] Li Li, Tegawendé F Bissyandé, et al. Droidra: Taming reflection to support whole-program analysis of android apps. 2016.

[65] 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.

[66] 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.

[67] 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.

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

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

[70] Dexin Liu, Yue Xiao, chaoqi Zhang, Kaitao Xie, Xiaolong Bai, Shikun Zhang, and Luyi Xing. ihunter: Hunting privacy violations at scale in the software supply chain on ios. 2024.

[71] 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.

[72] 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.

[73] Microsoft. Playwright. https://playwright.dev/, 2023.

[74] Ildar Muslukhov, Yazan Boshmaf, and Konstantin Beznosov. Source attribution of cryptographic api misuse in android applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 133–146, 2018.

[75] 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.

[76] 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.

[77] Yuhong Nan, Zhemin Yang, Min Yang, Shunfan Zhou, Yuan Zhang, Guofei Gu, Xiaofeng Wang, and Limin Sun. Identifying user-input privacy in mobile applications at a large scale. *IEEE Transactions on Information Forensics and Security*, 12(3):647–661, 2016.

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

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

[80] 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.

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

[82] 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.

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

[84] 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.

[85] David Rodriguez, Akshath Jain, Jose M Del Alamo, and Norman Sadeh. Comparing privacy label disclosures of apps published in both the app store and google play stores. In *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 150–157. IEEE, 2023.

[86] Jaebaek Seo, Daehyeok Kim, Donghyun Cho, Insik Shin, and Taesoo Kim. Flexdroid: Enforcing in-app privilege separation in android. In *NDSS*, 2016.

[87] Shashi Shekhar, Michael Dietz, and Dan S Wallach. Adsplit: Separating smartphone advertising from applications. In *USENIX security symposium*, volume 2012, 2012.

[88] Shao Shuai, Dong Guowei, Guo Tao, Yang Tianchang, and Shi Chenjie. Modelling analysis and auto-detection of cryptographic misuse in android applications. In *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pages 75–80. IEEE, 2014.

[89] 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.

[90] 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.

[91] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[92] Isabel Wagner. Privacy policies across the ages: content of privacy policies 1996–2021. *ACM Transactions on Privacy and Security*, 26(3):1–32, 2023.

[93] 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.

[94] 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.

[95] 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.

[96] 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.

[97] 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.

[98] 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.

[99] Yue Xiao, chaoqi Zhang, Yue Qin, Fares Fahad S Alharbi, Luyi Xing, and Xiaojing Liao. Measuring compliance implications of third-party libraries' privacy label disclosure guidelines. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, 2024.

[100] 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.

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

[102] 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.

[103] Ying Zhang, Md Mahir Asef Kabir, Ya Xiao, Danfeng Yao, and Na Meng. Automatic detection of java cryptographic api misuses: Are we there yet? *IEEE Transactions on Software Engineering*, 49(1):288–303, 2022.

[104] 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.

[105] 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.

[106] Chaoshun Zuo, Zhiqiang Lin, and Yinqian Zhang. Why does your data leak? uncovering the data leakage in cloud from mobile apps. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1296–1310. IEEE, 2019.