Navigating the Privacy Compliance Maze: Understanding Risks with Privacy-Configurable Mobile SDKs

Yifan Zhang[1*], Zhaojie Hu[2*], Xueqiang Wang[2†], Yuhui Hong[1],
Yuhong Nan[3], XiaoFeng Wang[1], Jiatao Cheng[3], and Luyi Xing[1†]

[1]Indiana University Bloomington, {yz113, yuhhong, xw7, luyixing}@indiana.edu
[2]University of Central Florida, {zhaojie.hu, xueqiang.wang}@ucf.edu
[3]Sun Yat-sen University, nanyh@mail.sysu.edu.cn, chengjt6@mail2.sysu.edu.cn

Abstract

The rise of privacy laws like GDPR and CCPA has made privacy compliance a requirement for mobile apps. Yet, achieving it is difficult due to the apps' use of third-party SDKs with opaque data practices. Recently, to assist apps in complying with privacy laws, many leading third-party SDKs have started providing privacy APIs for configuring the SDK's data practices. Nevertheless, the extent to which such a paradigm, referred to as *privacy-configurable SDKs* (or PICO *SDK*s), truly enhances app privacy compliance remains unclear to the community.

This question can only be answered through a systematic measurement study, which is nontrivial and requires in-depth analysis of the implementation of privacy APIs in PICO *SDK*s, as well as the way they are utilized, sometimes through a "wrapper" SDK that encapsulates other SDKs. To address this challenge, we developed PICOSCAN, a privacy risk analysis framework targeting Android, one of the most common mobile platforms. PICOSCAN automatically analyzes the code of both apps and SDKs to detect practices that potentially invade user privacy. Applying PICOSCAN to 65 most popular PICO *SDK*s and over 48,000 Google Play apps, we uncovered significant privacy risks in today's Android ecosystem. A large number of them fail to correctly utilize privacy APIs as prescribed, and even when these APIs are used, they often do not align with user privacy preferences. Moreover, our study reveals that many wrapper SDKs do not accurately convey privacy configurations to the SDKs they encapsulate, resulting in compliance risks. Our findings expose systematic failures in the design, implementation, and usage of PICO *SDK*s, highlighting the urgent need for more effective solutions to enhance the privacy assurance of Android apps. We will open-source the framework and make the data produced by this study publicly available.

1 Introduction

With the emergence of privacy laws and regulations such as GDPR and CCPA, concerns have been shifted to their compliance, especially for today's mobile apps. A key challenge in mobile privacy compliance is that mobile SDKs' data practices are often unclear to apps, yet privacy laws hold app developers accountable [36, 68, 80, 83]. Recently, to help apps comply with privacy laws and regulations, many SDKs provide mechanisms, particularly privacy APIs, for apps to configure the SDK's data practices (e.g., on *data collection*, *sharing*, or *selling*) to comply with privacy laws. For instance, for compliance with the Children's Online Privacy Protection Rule (COPPA), AppLovin's SDK [22], which is an ad network, offers the privacy API `setIsAgeRestrictedUser`. Once this API is invoked by an app, the SDK will refrain from collecting and processing any data from the current (child) user [15]. While different SDKs support various privacy APIs for apps to configure their compliance with specific laws, such a paradigm, dubbed *privacy-configurable SDKs* (or PICO *SDK*s), has been implemented by a large portion of top SDKs and is widely used in apps on the Android platform.

Emerging privacy risks with PICO SDKs. The PICO *SDK*s are supposed to improve privacy assurance for Android apps. However, our preliminary study reveals subtle privacy compliance risks in (1) how apps use the privacy configuration APIs (or capabilities), and (2) how PICO *SDK*s implement and fulfill the expected compliance. Essentially, the PICO *SDK*s and their adoption bring a variety of general and novel privacy compliance risks, collectively called *privacy-configuration compliance risks* or PICO *risk*s in our research. While observations of noncompliance related to SDKs have been reported before [51, 71], to the best of our knowledge, there has been no systematic, large-scale study on the PICO *risk*s to understand the fundamental causes of the problem.

Particularly, we found that configuring privacy compliance through PICO *SDK* can be more complicated than expected. An app often uses multiple PICO *SDK*s (e.g., for serving ads, analytics, and social networks) and is supposed to properly configure relevant privacy APIs of all these PICO *SDK*s. A PICO *SDK*, such as the ads network SDK AppLovin, internally wraps and uses ads from other ads network SDKs, such as *Google AdMob* and *Unity Ads*, which are also PICO *SDK*s

*The two lead authors contributed equally to this work.
†Corresponding authors.

with privacy APIs. These PICO *SDK*s wrapped by AppLovin are transitive dependencies of the app and we call AppLovin a PICO *wrapper*. The wrappers and the SDKs they wrap form complicated delegation relations in privacy configurations: once the app configures specific compliance requirements for the wrapper (e.g., age restriction), the wrapper is supposed to faithfully propagate the privacy configurations to all PICO *SDK*s it encapsulates. This can easily go wrong, leading to privacy risks. Specifically, in the absence of standards, not all (wrapped) PICO *SDK*s support the same sets of regulations, and for the same regulations, they may not even come with the same levels of support. Privacy violations have actually occurred, for example, when an app configures AppLovin for GDPR compliance, while such a configuration cannot be propagated to the wrapped SDK Google AdMob, due to the latter's partial support for GDPR. In the paper, we present detailed case studies and measurements of PICO *risk*s (§ 6), together with nine different types of PICO *risk*s (§ 4).

Detecting privacy risks with PICO *SDK*s. We developed PICOSCAN, an automatic privacy risk analysis framework, to detect PICO *risk*s in real apps on Android, one of the most common mobile platforms. This is based on a set of PICO privacy principles we summarized for different roles in the PICO ecosystem, i.e., the host app, PICO *SDK* and PICO *wrapper* (§ 3.3). PICOSCAN leverages a PICO *usage inspector* to analyze the use of PICO *SDK*s in the apps to determine whether the use violates the app-related principles. It also utilizes a PICO *SDK inspector* to check whether individual and wrapper PICO *SDK* implementations uphold applicable privacy principles. These two components were developed to mainly perform static analysis, with supplemental dynamic analysis to support PICO *risk* detection for specific PICO privacy principles (§ 4).

Measurement of PICO *risk*s in the wild. To understand the compliance risks associated with PICO *SDK*s in the wild, we built a PICO *SDK* metadata database named PICO METADB. This database comprises 65 of the most popular PICO *SDK*s on the Android platform, covering three major categories: 43 ad SDKs, 20 analytics SDKs, and 2 social network SDKs. Running PICOSCAN on 48,305 Google Play apps (downloaded in October, 2023), our measurement results indicate that despite the enhanced privacy configurability, PICO *SDK*s and their use in the wild are far from being compliant, and privacy risks can occur at every step of SDK integration and be introduced by every role in the PICO *SDK* ecosystem. Overall, among 24,844 apps in our dataset that integrate PICO *SDK*s, 7,880 apps (31.7%) are found to contain at least one PICO *risk*. Specifically, 31.5% apps fail to invoke privacy APIs in the first place, leaving privacy configurations of PICO *SDK*s unattended. The risk is substantial considering that over 43.1% PICO *SDK*s include privacy-invading default configurations that allow the collection and processing of user personal data. Even for apps that invoke privacy APIs, they are likely to configure the APIs in a way inconsistent with users' preferences, e.g., 259 apps either hard-code privacy APIs with data-enabling values directly, or set privacy APIs in conflict with user privacy preferences. Further, we found that there are numerous privacy risks even for the apps that perfectly configure privacy APIs, due to the erroneous implementations in PICO *SDK*s and PICO *wrapper*s. For example, 16 PICO *SDK*s that cover 14,869 apps have ambiguous consent configurations such that they still collect user personal data even when privacy APIs are invoked to decline the consent. Things become even more complicated for PICO *wrapper*: 12 PICO *wrapper*s (54.5% out of 22) are found to not relay privacy configurations to the PICO *SDK*s they encapsulate.

These findings shed new light on PICO *SDK*s, leading to better understanding about their privacy assurance. First, with the modularization of app development using closed-source components from diverse parties, developers require enhanced tooling, including privacy-configurable SDKs, to maintain consistent privacy promises across all app components. However, this study demonstrates that this goal may not be easily achieved solely relying on configurability. *Inconsistent privacy configurations* may arise from various development flaws (e.g., missing or erroneous configurations), and unfulfilled responsibilities by PICO *wrapper*s (e.g., broken relay). Second, the design and use pattern of PICO *SDK*s tend to expose app users to privacy risks. Third, misimplementation and misinterpretation are common in PICO *SDK*s, possibly due to the absence of systematic review and proper standards. Based on our findings, we provide suggestions to key stakeholders, such as developers of apps and PICO *SDK*s, as well as policy makers and law enforcement agencies, with the hope of helping end-users obtain genuine privacy assurance when using Android apps (§ 7).

Contributions. We summarize the contributions as follows:

- We present a systematic study of privacy risks for the use and implementation of privacy-configurable SDKs (PICO *SDK*) on Android, and identified a range of general, novel privacy compliance risks, collectively called PICO *risk*s.
- We developed PICOSCAN, a compliance risk analysis framework to automatically identify PICO *risk*s in Android apps and PICO *SDK*s. We will fully release the source code.
- We performed a large-scale study of PICO *risk* with over 48 thousand Google Play apps and 65 popular PICO *SDK*s. This reveals that PICO *SDK*s, though created for privacy compliance, often fail to provide expected privacy assurance.

2 Background

Privacy configuration capabilities supported by third-party SDKs. The mobile SDK ecosystem has evolved significantly, with many SDKs now offering dedicated "*advanced privacy settings*" documents (such as [29]) to assist app developers in achieving privacy compliance. These documents often contain three types of information: legal and privacy disclosure, app-level privacy configurations, and user-level privacy configurations. Legal and privacy disclosure outlines privacy policies describing how an SDK processes and shares infor-

mation, along with methods for end users to exercise privacy rights (e.g., through data subject access request). App-level privacy configurations enable app developers to customize SDK usage and functionality mostly from the server side (e.g., via SDK dashboard), such as specifying data points for an analytics SDK. These configurations apply to all end-users of an app. User-level privacy configurations enable app developers to customize privacy exposure for each end-user. This study focuses on user-level privacy configurations.

Privacy Configurable (PICO) SDKs. To support user-level configurations, third-party SDKs may offer a set of predefined privacy APIs. We refer to such SDKs as privacy-configurable SDKs (PICO *SDK*s). The privacy APIs of PICO *SDK*s may control various data practices, such as data collection, sharing, and selling, in compliance with privacy laws (e.g., GDPR). When using a PICO *SDK*, an app often invokes its initialization API to configure global SDK options. For example, an ads SDK initializes the ads unit ID before requesting ads from backend ad networks. PICO *SDK* privacy APIs are typically invoked alongside the SDK initialization APIs to ensure that privacy configurations are picked up early on by SDKs.

3 Overview of PICO *SDK*s

3.1 Status Quo of PICO *SDK*s

Landscape of PICO *SDK*s. Privacy APIs of PICO *SDK*s aim to help apps fulfill requirements of a range of privacy laws, regulations, and industry standards, such as GDPR [26], COPPA [25], US states' consumer privacy acts [27, 46, 75, 76], China's Personal Information Protection Law (PIPL) [30], and CARU Advertising Guidelines [24]. We conducted a survey to understand the privacy laws supported by popular Android SDKs. Specifically, we gathered a list of 203 most popular SDKs (ranked by AppBrain [4], a service providing Android market statistics) including 137 Ads SDKs, 41 Analytics SDKs, and 25 Social Network SDKs. We focus on the three SDK categories since they were known to involve privacy practices [65, 67, 86]. We asked two security researchers to find API documents for the popular SDKs, and then cross check and combine their results. In total, we found API documents for 91 of the 203 SDKs. From the documents, we searched privacy APIs based on keywords including "`privacy`", "`compliance`", "`regulations`" and names of specific laws. The most commonly supported laws are GDPR (61 SDKs), CCPA (39 SDKs), and COPPA (34 SDKs), spanning a total of 65 PICO *SDK*s. We provide a detailed list of privacy laws and regulations along with the number of SDKs that offer privacy APIs to ensure compliance online [17].

Scope of research. Based on the survey, our study focuses on the support of three laws (GDPR, CCPA, COPPA) by the 65 PICO *SDK*s (see PICO METADB in Section 4.4). We study how effectively their privacy APIs support privacy compliance and whether apps have used them properly to achieve expected compliance goals. Notably, although each privacy law defines a range of privacy-related requirements, we specifically focus on the following requirements for each law, which are most commonly supported by the privacy APIs. We documented all 191 privacy APIs in a database (§ 4.4) and have released them at [17].

• *GDPR compliance with consent requirements.* Based on GDPR, PICO *SDK*s (as data controllers) should establish a legal basis for processing personal data, in most cases by obtaining users' consent. Hence, many PICO *SDK*s have provided privacy APIs with parameters that allow app developers to specify users' consent status, thereby enabling the PICO *SDK*s to administer private data processing according to the consent status. An example is Vungle [79], an ad network SDK which offers an API `setGDPRStatus`. By passing in a `false` parameter value, an app that integrates Vungle can instruct Vungle to avoid collecting a range of personal data restricted by GDPR, such as Android ID and advertising ID.

• *CCPA compliance with the right to opt-out.* Under CCPA, PICO *SDK*s may not have to obtain explicit user consent for data collection (although businesses must provide clear and conspicuous privacy notice). Rather, CCPA requires that end users who are California residents be able to opt-out of data sharing and selling from businesses. To fulfill such a privacy right, PICO *SDK*s commonly provide a privacy API for apps to specify end users' choices of "`opt-in`" or "`opt-out`", based on which the PICO *SDK*s restrict the sharing and trading of user data. For example, the Vungle SDK has an API `setCCPAStatus` to capture users' "`opt-out`" status.

• *COPPA compliance and protection for children users.* COPPA limits the collection and use of children's personal data, except with explicit and verifiable consent from parents. Other laws such as GDPR and CCPA also have corresponding restrictions for processing children's personal data. For compliance, a most common practice we observed from PICO *SDK*s is to provide a privacy API that allows apps specify whether the current user is under age restriction (being a child). The PICO *SDK*s will then refrain from collecting data from the child users. For example, the Vungle SDK provides a privacy API (`setCOPPAStatus`) that takes a boolean parameter to indicate whether the current app user is a child or not.

3.2 Motivating Example

We show real examples of PICO *risk*s in Figure 1. `CallApp` [60][1] is a popular app that has over 100 million downloads on Google Play. It offers attractive security features for users, such as detecting caller identity, and blocking spams. The app, for maximized monetization, implemented in-app ads bidding that selectively displays ads from multiple ads networks using their SDKs, such as `Google AdMob`, `AppLovin`, and `Facebook Audience Network`. Offerings of these ads SDKs include showing

[1] The app is subject to all three privacy laws. It is available to users in the EU and CA. It also collects users' date of birth and permits children to use it.

personalized ads based on users' personal data (e.g., user and device identifiers, and demographic information, etc.). Most of the ads SDKs are PICO *SDK*s, providing privacy APIs for apps to configure them in accordance with applicable privacy laws, such that the apps as a whole can be compliant. Upon analyzing the CallApp app, we found that both the in-app use and internal implementations of PICO *SDK*s are complicated, resulting in several types of subtle privacy compliance risks in Android apps that were unknown, or at least less known before.

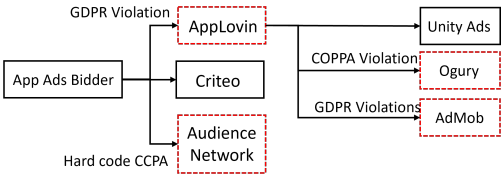

Figure 1: PICO SDKs in the CallApp

First, the app intends to be GDPR-compliant by calling AppLovin SDK's privacy API `setHasUserConsent` with an argument of either `true` or `false`, depending on the user choice. AppLovin itself will not collect or use personal data for delivering personalized ads without user consent (GDPR compliance). However, the AppLovin SDK not only delivers ads from its own ads network, but also internally wraps and bids ads from other ads SDKs, such as *IronSource*, *Vungle*, *Google AdMob*, and *Unity Ads*, etc. These SDKs wrapped by AppLovin are transitive dependencies of the app and we call AppLovin a wrapper SDK or PICO *wrapper*. The expectation is that, once the app urges a PICO *wrapper* like AppLovin for GDPR-compliance, it should propagate the compliance requirement to all ads SDKs it internally uses (by calling their privacy APIs for GDPR). We find that the expectation fails to be fulfilled for key challenges: (1) Not all wrapped ads SDKs support compliance configurations evenly. Google AdMob, for example, partially supports GDPR configuration: after a PICO *wrapper* passes along "not consent" to AdMob's privacy API, AdMob will not deliver personalized ads but will always collect user data including device/user identifiers [1,2]. Failing to account for the different levels of support for the same law among PICO *SDK*s results in compliance risks for the app. (2) Not all PICO *SDK*s used by a PICO *wrapper* support the same laws. For example, the Yandex SDK and Kidoz SDK have not supported privacy configurations for CCPA, and both of them can be used by the wrapper AppLovin.

Second, with many PICO *SDK*s used in an app, their host app or PICO *wrapper* might not configure compliance APIs of all PICO *SDK*s, or configure them in a correct manner. Ogury, for example, is an ads SDK supporting COPPA configuration. However, AppLovin does not call Ogury's COPPA API `applyChildPrivacy`, although AppLovin did call COPPA configuration APIs of all other ads SDKs it wrapped. Also, for CCPA compliance, when the user is a California resident, the app is supposed to invoke CCPA APIs in PICO *SDK*s to communicate the user's opt-out status for data sharing/sale. Although the app uses the Facebook Audience Network (SDK) to deliver ads, it uses a hard-coded "`opt-in`" option when configuring Facebook SDK's CCPA API `setDataProcessingOptions`, posing a violation to users who choose "`opt-out`".

Further, although many PICO *SDK*s feature privacy APIs that support multiple laws, their default configurations (without privacy API calls by the app) do not comply with privacy laws. For example, the InMobi SDK [28] by default does not comply with COPPA unless the app calls the `setIsAgeRestricted` API with an argument value `true`.

3.3 PICO Privacy Principles

We cannot evaluate privacy compliance risks without specific privacy principles tailored for PICO *SDK*s. This section, based on the motivating example (§ 3.2), summarizes a set of such principles for different roles in the PICO ecosystem, i.e., the host app, PICO *SDK* and PICO *wrapper*.

Privacy principles for apps using PICO *SDK*s. Privacy APIs in PICO *SDK*s are designed to meet the key privacy objective of returning privacy control to end users (in alignment with the expectations reported in prior studies [42,51,64,73]). Hence, it is crucial that apps integrating PICO *SDK*s configure the privacy APIs correctly and effectively based on the actual privacy preferences of end users.

> *Principle #1 (Complete, User-controlled Privacy Configuration)*: Apps should ensure complete, correct privacy configurations of all PICO *SDK*s in accordance with user preferences.

Privacy principles for PICO *SDK*s. Similar to the principle of "*secure by default*" [49], which requires automatic enabling of security measures without user intervention, the principle of "*privacy by default*" has become common sense among privacy practitioners [37,52,58] and is also part of privacy laws [44,48].

> *Principle #2 (Privacy by Default)*: PICO *SDK*s should have default configurations that protect user privacy.

Individual PICO *SDK*s offer privacy APIs for enabling apps using them to manage PICO *SDK* privacy practices. Hence, it is natural to mandate that privacy APIs to effectively fulfill all promised privacy compliance without ambiguity, akin to the "*complete mediation*" security principle by Saltzer and Schroeder [72].

> *Principle #3 (Nonambiguous, Fulfilled Privacy Compliance)*: PICO *SDK*s should fulfill all promised privacy compliance once relevant privacy APIs are configured; the promise (for the compliance effect of privacy API configurations) should be nonambiguous.

Privacy principles for PICO *wrapper*s. As demonstrated in § 3.2, when using PICO *wrapper*s (one that encapsulates other PICO *SDK*s), the underlying PICO *SDK*s are not exposed to

apps directly. Instead, their privacy APIs are configured by the PiCo *wrapper*s, similar to the delegation design pattern [56]. Therefore, it is the responsibility of the PiCo *wrapper*s to properly propagate (or forward) privacy configurations of the app to all internal PiCo *SDK*s.

Notably, a PiCo *wrapper* itself is usually a PiCo *SDK* by providing privacy compliance APIs (e.g., AppLovin in Figure 1) and thus should follow principles of PiCo *SDK*s.

> *Principle #4 (Complete Privacy Delegation)*: PiCo *wrapper*s should fully propagate app's compliance configurations to all internal PiCo *SDK*s in use.

Discussion. The principles are not direct adoption of prior general principles, and they are contextualized to PiCo *SDK*s. Consequently, the principles will guide the identification of PiCo *risk*s in real apps and SDKs (§ 4), although they may not be exhaustive. For example, given that data practices extend into PiCo *SDK* backends, it becomes necessary to establish privacy principles that apply to these backends. We leave further exploration of those principles for future research.

4 Analyzing PiCo Risks

PiCo *risk*s occur when apps, PiCo *SDK*s, or PiCo *wrapper*s violate the above privacy principles (§ 3.3). In this section, we introduce PiCoScan, an automatic analysis framework to detect PiCo *risk*s.

Overview. Figure 2 shows the overview of PiCoScan, including the techniques and tools used in it. PiCoScan takes Android apps as input and identifies PiCo *risk*s in two aspects: (1) whether and how the apps use PiCo *SDK*s, and (2) how PiCo *SDK*s and PiCo *wrapper*s implement their privacy APIs to meet the expected privacy compliance, with two main components, i.e., PiCo *usage inspector* (*PUI*) and PiCo *SDK inspector* (*PSI*), respectively.

PUI first identifies the usage of PiCo *SDK*s in the apps through static analysis, which involves searching for package names of PiCo *SDK*s and invocations of their privacy APIs. This is enabled by a PiCo SDK metadata database (PiCo MetaDB) that we built from the 203 most popular mobile SDKs listed on AppBrain [4]. The database contains various useful information for the analysis (as we will detail in § 4.4), such as the package names of PiCo *SDK*s, API signatures, etc. For each privacy API invocation found in the apps, *PUI* further identifies whether the invocation is made by the app code or by a PiCo *wrapper*. In the case of the former, *PUI* applies app-specific principles (Principle #1) to detect privacy risks (§ 3.3) related to app code invoking the privacy API. For the latter, PiCoScan leverages *PSI* to detect privacy risks based on wrapper-specific principles (Principle #4). *PSI* further inspects the implementation behind the privacy APIs of PiCo *SDK*s and identifies privacy risks based on PiCo *SDK* related principles (Principle #2 and #3). Note that *PSI* analyzes the PiCo *SDK*s and PiCo *wrapper*s that are already compiled into the apps as an alternative to analyzing separate SDKs, since some SDKs are not publicly available, e.g., Amazon mobile ads SDK [21] is invitation-only.

Directly applying high-level principles in code-level detection is difficult. Thus, we developed a set of more specific, fine-grained technical patterns of PiCo *risk*s, called PiCo *Violation Patterns* (*PVP*s), corresponding to each principle (Table 1). *PUI* and *PSI* scan for the occurrences of the PVPs in the app code, the code of PiCo *SDK*s and PiCo *wrapper* hosted in the apps, and report them as PiCo *risk*s. PiCoScan is mainly built on static analysis and is supplemented by dynamic analysis for the scan of certain *PVP*s. For example, under Principle #2, to identify the default compliance status of PiCo *SDK*s (e.g., `true` or `false` for age-restricted data collection), we leverage static analysis to identify specific variables that store the compliance value and then use dynamic analysis to find the exact default value (detailed in § 4.2.2). Figure 2 outlines the static and dynamic analysis techniques and tools adopted for different *PVP*s. We will elaborate on the choice of these techniques in the latter part of this section.

Table 1: PiCo Violation Patterns (PVPs)

Role	Privacy Principle	PVP
App	#1: Complete, user-controlled privacy configuration	#1: Missing configuration #2: Configuration disregarding or inconsistent with user privacy preferences #3: Erroneous configuration
PiCo SDK	#2: Privacy by default	#4: Privacy-invading by default
	#3: Nonambiguous, fulfilled privacy compliance	#5: Ambiguous consent configuration #6: Ineffective configuration
PiCo Wrapper	#4: Complete privacy delegation	#7: Uneven privacy support #8: Broken privacy delegation #9: Conditional propagation

4.1 PiCo Usage Inspector (PUI)

4.1.1 Identifying PiCo Usage from Apps

To determine whether an app uses PiCo *SDK*s, including whether privacy APIs are invoked and whether they are invoked by first-party code (app code) or third-party code (PiCo *wrapper*s), *PUI* builds a global call graph for the app and traverses all the edges to identify reachable methods that match PiCo *SDK* initialization APIs, as defined in the PiCo MetaDB. It then determines the parties using the PiCo *SDK*s by comparing the caller of the initialization APIs to the package names of the main app or any other SDKs. The identification of PiCo *wrapper*s is slightly difficult, SDKs often lack clear disclosure about whether they are wrappers for other SDKs. In this study, we identify PiCo *wrapper*s by uncovering inter-SDK connections on an app's call graph. Specifically, for each discovered PiCo *SDK* initialization API on the call graph, we explore the subgraph rooted in this API to locate the initialization APIs of another SDK. This relationship, where the initialization API of one SDK leads to that of another, is a clear indicator that the former SDK functions as a wrapper for the latter SDK.

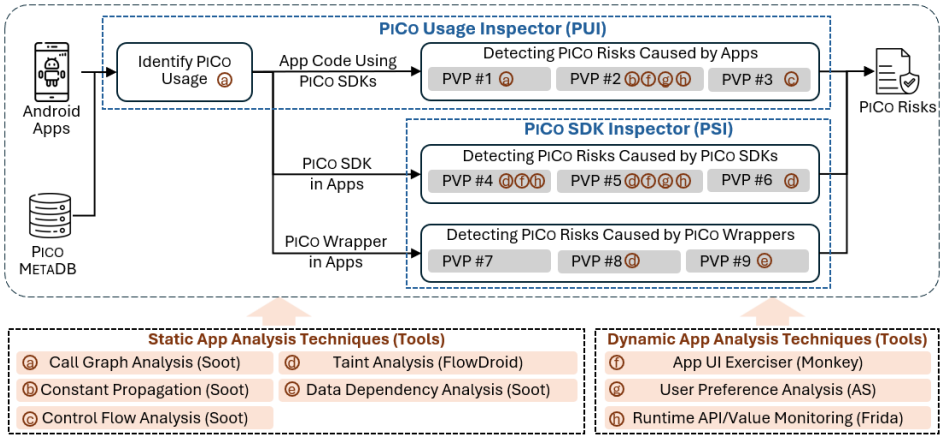

Figure 2: Overview of PiCoScan

4.1.2 Patterns of PiCo Risks Caused by Apps

Under the privacy principles governing apps (Principle #1), we focus on three *PVP*s whose presence in the first-party code indicates risks associated with the app's use of privacy APIs (or configuration capabilities) of PiCo *SDK*s. In general, apps using PiCo *SDK*s are expected to configure the privacy APIs of PiCo *SDK*s correctly, adhering to (1) the API definition and programming requirements, and (2) users' privacy preferences typically collected through app UIs, such as whether they consent to data collection (see Figure 7 in the Appendix for an example). We elaborate on the *PVP*s as follows.

Missing configurations (*PVP* #1). Apps using a PiCo *SDK* but failing to configure the privacy compliance of the SDK by invoking its privacy APIs indicate a PiCo *risk*. An app often uses multiple PiCo *SDK*s but may not configure all of them. An example is the "Copy My Data" (`com.mediamushroom.copymydata`) app [59], which leverages the IronSource SDK to show ads but does not invoke any of the IronSource privacy APIs.

Configurations disregarding or inconsistent with user privacy preferences (*PVP* #2). Apps are expected to capture user privacy preferences and communicate them to PiCo *SDK*s by properly invoking the PiCo *SDK*s' privacy APIs. However, they might fail to capture user privacy preferences or, if they do, they may fail to use the proper parameter values when invoking the privacy APIs. A special situation within this pattern is when an app uses hard-coded parameter values (e.g., `consent: true`) to configure privacy APIs – an issue happened in many apps. Concerning COPPA compliance, for example, the "SuperLozzi" (`com.ltlk.z7`) app, a play-and-earn app with over one million downloads on Google Play, collects the user's date of birth. However, the app calls AppLovin's COPPA API `setIsAgeRestrictedUser` with a hard-coded parameter value of "`false`", which disables AppLovin's COPPA compliance even when the user is a child (see measurements in § 6.2).

Erroneous configurations (*PVP* #3). Even if apps configure the necessary privacy APIs with the correct parameters that reflect user preferences, their configurations may not effectively achieve the desired compliance. This happens when the use of privacy APIs fails to meet the programming requirements specified for using these APIs. For example, the "Solitaire King" (`com.solitaire.king.card.creative.wuliu`) app incorporates the PiCo *SDK* AppLovin to display ads and configures multiple SDK privacy APIs related to CCPA, GDPR, and COPPA. However, AppLovin requires its initialization function, `initializeSdk`, to be invoked after configuring its privacy APIs. Unfortunately, this requirement was not followed by many apps, rendering their privacy configurations ineffective.

4.1.3 Detecting PiCo Risks Caused by Apps

To detect missing configurations (*PVP* #1), *PUI* traverses the apps' call graphs to find out whether the apps initialize and use a PiCo *SDK* but do not call any of their privacy APIs.

For *PVP* #2, we detect the inconsistencies between privacy configurations and the user's privacy preferences using dynamic analysis. Specifically, *PUI* launches the apps under analysis, exercises their UIs, and utilizes accessibility service (AS) APIs [20] to monitor the rendered app UIs. *PUI* identifies privacy consent dialogs that request user consent for data collection and usage (Figure 7 in the Appendix). This is done by matching a set of common privacy-related keywords in the texts of consent dialogs, i.e., via regex matching "`^.*(?:privacy policy|collect your|i agree|your consent).*$`". *PUI* leverages AS APIs to automatically click the button on consent dialogs to decline the consent request, by matching keywords such as "`no`", "`decline`", "`cancel`", "`not now`" or "`disagree`". Meanwhile, *PUI* instruments and monitors runtime parameter values of PiCo *SDK* privacy APIs, in particular those indicating "`true`" or "`false`" for user consent, and "`opt-in`" or "`opt-out`" for data sharing and sale. Since the consent request has been declined, runtime values indicating a `true`" for user consent or `opt-in`" represent privacy risks related to *PVP* #2. *PUI* uses Frida [8] for monitoring runtime values. In case that privacy consent dialogs are not observed, *PUI* performs

constant propagation on the interprocedural control-flow graph (ICFG) of the apps to determine whether the parameters of privacy APIs are constants (i.e., hard-coded).

Under *PVP* #3, *PUI* checks whether invoking individual privacy APIs must occur before or after invoking other APIs of the PiCo *SDK*s. In our study, the ordering requirements were extracted from SDK documents and recorded in the PiCo MetaDB (14 privacy APIs from eight PiCo *SDK*s require specific orders, see Appendix 9). Using the PiCo MetaDB, *PUI* checks the app code and API calls along the control flows to identify misordered API invocations.

4.2 PiCo SDK Inspector (PSI) for PiCo *SDK*s

PSI analyzes individual PiCo *SDK*s to check violations to related principles (principles #2 and #3 in § 3.3), focusing specifically on three PVPs as elaborated below.

4.2.1 Patterns of PiCo Risks Caused by PiCo *SDK*s

Privacy-invading by default (*PVP* #4). The PiCo Principle #2 requires that individual PiCo *SDK*s come with a default configuration in compliance with related privacy regulations. However, this is not true for many PiCo *SDK*s. Take, for example, the SDK of InMobi [28], a mobile marketing platform specializing in personalized ads. The InMobi SDK will stop collecting certain personal information (such as *advertising ID*) and notify its server to comply with COPPA for the current user only after the host app calls the `setIsAgeRestricted` API with a `true` parameter value. If the above API is not called, InMobi defaults to assuming that all app users are adults, to whom COPPA does not apply, and thus allows for the collection of personal data for user profiling and serving personalized ads. Privacy-invasive default configurations like these are found in many popular PiCo *SDK*s (see measurements in § 6.3).

Ambiguous consent configuration (*PVP* #5). At the core of many privacy regulations is a requirement for user consent before processing personal data. For example, GDPR requires that data controllers (apps or SDK providers) must obtain user consent for the collection and use of personal data, including the use for serving personalized ads [26]. Under this requirement, the `requestConsentInfoUpdate` API of AdMob is used to help apps collect user consent "*to use your personal data for personalized ads*" (by showing a consent dialog developed by the SDK, Figure 6), and apply the user consent status for the SDK's operation. While not all PiCo *SDK*s come with built-in consent dialogs, they commonly provide APIs for apps to specify the user's consent (`true` or `false`), such as for personalized ads [39]. However, the intended compliance effects of the consent status can be ambiguous: for configurations of `consent: false`, it is unclear whether the SDKs will not collect user personal data, or they will still collect user data but just do not use the data for specific purposes such as serving personalized ads. Our study of real PiCo *SDK*s showed that such an ambiguity is indeed underlined by different compliance effects among PiCo *SDK*s, resulting in privacy risks: for example, the AdMob SDK always collects user identifiers such as *adid* and *ip address* regardless of the consent status configured by the API; in contrast, Kochava will disable the data collection being more compliant.

Ineffective configuration (*PVP* #6). When apps correctly configure privacy APIs, the PiCo *SDK*s are expected to achieve the compliance goals. Problems arise when the PiCo *SDK*s have implementation mistakes that render privacy APIs ineffective. For example, the MyTarget SDK offers a GDPR API `setUserConsent` to support the configuration of user consent for data collection. Behind the API, the user consent status is passed along and stored in an SDK internal variable `isConsent`. In the SDK implementation, every collection of device identifiers, such as Google and Huawei ad IDs, occurs after inspecting the variable. Such a variable (referred to as IPSVs in PiCoScan's detection, § 4.2.2), essentially records the configured consent status, which is important information governing the SDK's privacy practices. However, in the MyTarget SDK implementation, the `isConsent` variable can be modified by other unexpected program paths, specifically from the SDK's CCPA API `setCcpaUserConsent`. Any consent status configured for CCPA will overwrite that of GDPR, silently rendering the GDPR configuration ineffective. This overwriting causes problems in cases where the apps integrating the SDKs need to ensure compliance with both GDPR and CCPA, e.g., when an app must protect users who are California residents while the app itself operates as an EU-based business.

4.2.2 Detecting PiCo Risks Caused by PiCo *SDK*s

Detecting default privacy-invading configurations (*PVP* #4). To assess whether the privacy configurations of a PiCo *SDK* are privacy-invading by default, we developed a method in *PSI* to model the implementation of the PiCo *SDK*'s configurations. Specifically, our observation is that once a privacy API is invoked with certain parameters (called privacy parameters), a PiCo *SDK* usually stores the values of the parameters (e.g., whether the user is a child) as internal states in local variables, such as class or instance fields. We refer to such variables storing privacy parameter values as *internal privacy state variables* or IPSVs. Further privacy practices of the PiCo *SDK*, such as invocations of system APIs to collect personal data, will be governed by checking the IPSV values. In the InMobi SDK, for example, the data flow behind its COPPA API `setIsAgeRestricted` shows that the privacy parameter `isAgeRestricted` is eventually stored in an IPSV, i.e., `<com.inmobi.media.ea: Boolean c>`. In multiple SDK functions that attempt to collect user personal data, it first checks the IPSV to determine whether the collection should proceed. Notably, while the privacy parameter can be stored in various locations, including in shared preferences [31] for persistence, it is often loaded into the IPSV for the SDK to easily access it in different functions.

Based on the above observation, *PSI* first performs static taint analysis on the apps to identify the IPSVs corresponding to the SDK's privacy parameters. This analysis is then complemented by dynamic analysis to identify the runtime values of these IPSVs using Frida [8]. If the runtime values of the IPSVs match the values enabling data collection, sharing or selling as recorded in the PICO METADB, *PSI* reports a PICO *risk* corresponding to PVP #4. More specifically, to identify IPSVs, we use privacy APIs' parameters as sources and find program variables to which the parameter values are propagated (within the SDK code). Considering that the values may be propagated to some intermediate variables, *PSI* filters out those with different types from the privacy parameters and those local variables internal to a method; *PSI* specifically retains variables used in a condition check that acts as a guard for user personal data collection.

Identifying ambiguous consents (*PVP* #5). To detect *PVP* #5, *PSI* identifies the collection of personal data in the SDK code even after configuring `consent: false` in the invocation of privacy APIs. The first step uses static analysis: *PSI* identifies code locations that access personal data. This is done by checking invocations of a list of system APIs (compiled from prior work [70, 87]) that access device and user data (e.g., device identifiers, advertising identifiers, and location). Similar to prior work [40], the static analysis also finds out the program paths in the SDK that transmit the user data to the Internet. Next, *PSI* runs the app and uses `consent: false` to configure the privacy APIs of PICO *SDK*s. This is done using dynamic instrumentation that manipulates related privacy parameters upon the app's invocation of privacy APIs. *PSI* reports privacy risks if it finds that the program paths that collect and transmit user data are executed at runtime (by inspecting call stack traces), even under `consent: false` configurations of the privacy APIs.

Identifying ineffective configurations (*PVP* #6). The presence of multiple program paths manipulating common IPSVs related to privacy APIs is an indicator of potentially ineffective configurations. Therefore, we detect them by checking whether other public APIs of a PICO *SDK* can cause changes to the IPSVs of any of its privacy APIs. For this purpose, we extract the set of IPSVs, S_{IPSV}, for each privacy API of a PICO *SDK*. Next, we perform static taint analysis to collect all internal variables affected by other public APIs of the same PICO *SDK*, i.e., S_v. We cross-compare the S_{IPSV} and S_v to identify whether the IPSVs of the privacy API are manipulated by other program paths. Using this method, we can detect ineffective configurations in other 8 PICO *SDK*s besides MyTarget, such as Appodeal, and BidMachine (see measurements in §6.3).

4.3 PICO SDK Inspector for PICO *wrapper*

The PICO *wrapper* SDKs are special PICO *SDK*s that provide privacy APIs to apps and are expected to fully and correctly propagate apps' privacy configurations to their wrapped PICO *SDK*s. This requirement is instrumental considering that the wrapped PICO *SDK*s are transitive dependencies of the apps, which are often unknown and inaccessible for app developers to configure. Under the Principle #4, PICOSCAN focuses on three patterns of violations related to PICO *wrapper*s.

4.3.1 Patterns of PICO Risks Caused by PICO *wrapper*s

Uneven privacy support (*PVP* #7). In the absence of standards for PICO *SDK*s, individual PICO *SDK*s may support different sets of laws and regulations, and for the same regulations, they may not even come with the same levels of support. For example, PICO *wrapper*s Appodeal [38] and AppLovin [47] provided privacy APIs for GDPR compliance. Meanwhile, they both wrapped Meta Audience Network [7], SDK of another ads network, to help serve ads. Notably, Meta Audience Network has not provided privacy APIs to configure GDPR compliance. This can directly lead to privacy risks with GDPR in real apps. For example, when the app `com.solutions.class11dkg` configures Appodeal with "`consent: false`" for data collection based on the user's privacy preference, Appodeal cannot propagate the configuration to Meta Audience Network, which actually collects user data and serve ads in the app regardless of user's consent, posing a risk.

Broken privacy delegation (*PVP* #8). Even when the PICO *wrapper* uses an individual PICO *SDK* that has aligned privacy law support, it is possible that the PICO *wrapper* fails to fully propagate the app's privacy configurations to the corresponding APIs of the individual PICO *SDK*. For example, AppLovin (PICO *wrapper*) internally uses the Facebook Audience Network (PICO *SDK*) for serving ads. The AppLovin SDK has a CCPA API `setDoNotSell`, that enables app users to opt out of data sharing/sale using a `true` parameter. However, the AppLovin SDK does not use such a parameter or related information to invoke the Facebook CCPA API `AdSettings.setDataProcessingOptions`, leading to a privacy risk.

Conditional propagation (*PVP* #9). A variant of the broken delegation is the presence of extra conditions imposed by the wrapper that blocks propagation of privacy configurations. An example is the PICO *wrapper* Admost and its wrapped SDK Unity Ads. Both SDKs have a GDPR related API to configure user consent for data collection, i.e., `AdMostConfiguration.setUserConsent`, and `MetaData.set` respectively. Based on user choices, when the host app configures Admost with the user's consent status for collecting personal data, Admost will only conditionally propagate the consent status to Unity Ads. When Admost considers the user's IP address is not within EU countries (thus may not be subject to GDPR), it will configure Unity Ads with a "`consent: true`". This happens even if (1) the user does not consent to data collection or (2) the app intends to configure the PICO *wrapper* not to collect user data. Such a design pattern is problematic: (1) it can violate the app's compliance expectation and privacy goal; (2) using IP addresses to determine whether the user is subject to GDPR is not reliable [6, 12].

Further, such a practice violates Unity Ads' privacy goals for users: Unity Ads noted that "*Consent extends beyond GDPR and should be applied in any region*" [77].

4.3.2 Detecting PiCo Risks Caused by PiCo *wrapper*s

PUI detects *PVP* #7 by comparing privacy APIs and different privacy laws supported by the wrapper and its internal PiCo *SDK*s. Such information has been recorded in the PiCo MetaDB (see PiCo MetaDB in § 4.4). For *PVP* #8, *PSI* adopted static taint analysis to check whether the privacy parameters in the wrapper's privacy APIs are propagated through program paths to parameters of internal PiCo *SDK*s' privacy APIs. *PUI* considers that such program paths may come with condition checks before invocations of the internal PiCo *SDK*'s privacy APIs. To identify improper conditions that restrict the privacy propagation (*PVP* #9), we leverage an approach based on the data dependency of the conditions being checked, inspired by prior studies that detect hidden operations [84] or logic bombs [88]. Specifically, *PSI* performs backward data dependency analysis for a condition variable. If its value is independent of a privacy parameter (those in the wrapper's privacy APIs), this indicates that such a condition can be implicit and unknown to the apps that configure the wrapper's APIs. We consider such a condition to be a violation under *PVP* #9. For example, in identifying the above example involving Admost which imposes an improper condition by checking IP addresses, *PSI* finds that IP addresses is not propagated from any parameters of Admost SDK's APIs.

4.4 PiCo MetaDB

PiCoScan comes with a PiCo MetaDB that stores metadata for 65 PiCo *SDK*s on Android whose privacy APIs support GDPR, CCPA, or COPPA. The 65 PiCo *SDK*s are filtered from 203 popular SDKs (see § 3.1), and cover three major SDK categories: 43 ads SDKs, 20 analytics SDKs, and 2 social network SDKs. Table 2 provides a more detailed breakdown of SDK types and the supported privacy laws.

From the API documents of the PiCo *SDK*s found in § 3.1, we asked the two security researchers to independently identify two types of APIs: privacy configuration APIs and the SDKs' initialization APIs, by searching for keywords, i.e., "`initialize`", "`privacy`", and relevant laws including "`GDPR`", "`COPPA`", "`CCPA`", and "`US State laws`", in the documents. They examined the API descriptions to find indicators (such as "`before`", "`after`", and "`order`") for the invocation order and recorded these invocation order requirements. Afterwards, they reviewed the API arguments to identify which arguments are related to "`consent`", "`opt-out`", and "`children`" (corresponding to three requirements noted in the scope of research in § 3.1). They were then asked to validate each other's findings, resolve any disagreements through discussions, and combine them.

In total, PiCo MetaDB includes 191 privacy APIs and 146 initialization APIs of the 65 PiCo *SDK*s. The privacy APIs include 106 APIs for GDPR compliance provided by 61 SDKs, 47 APIs for CCPA compliance provided by 39 SDKs, and 38 APIs for COPPA compliance provided by 34 SDKs. The PiCo MetaDB recorded metadata of the APIs useful for our analysis, including API signatures (including class names, method names, parameters types), corresponding privacy laws, parameters with privacy semantics and values of compliance (e.g., a `true` parameter indicating child users in a COPPA API `setIsAgeRestricted`), and constraints for the correct use of the APIs (e.g., order of API invocations), etc. We show an example of the metadata for Bytedance APIs in Appendix (Figure 5). For reproducibility, the API documents and the resulting PiCo MetaDB have been released at [17].

Table 2: Distribution of SDKs in PiCo MetaDB

SDK Type	# PiCo *SDK*s Supporting			Subtotal
	GDPR	CCPA	COPPA	
Ads	41	33	29	43
Analytics	20	5	4	20
Social Network	0	1	1	2
Subtotal	61	39	34	65

5 Implementation and Evaluation

5.1 Implementation

The static analysis techniques are mostly implemented with Soot [32], one of the most commonly used frameworks for Java and Android analysis. To achieve better coverage, we built app call graphs using an extensive list of app entries, including all lifecycle functions of the registered app components, as well as app methods that match any of the 879 subsignatures of the system callback methods (as reported in EdgeMiner [45]). We built the Interprocedural Control-Flow Graph (ICFG) in the Soot framework using Heros [16] . On top of the call graph and the ICFG, we implemented techniques for constant propagation, and data dependency analysis. In total, we implemented the static analysis with 4,587 lines of code.

The detection of certain PVPs, i.e., #4, #5, #6, and #8, relies on taint analysis, for which we leveraged FlowDroid [40], a state-of-the-art tool for static taint analysis. Specifically, for PVPs #4 and #6, we used FlowDroid to trace the propagation from privacy API parameters to SDK internal variables (IPSVs) storing the parameter values. A key challenge in directly applying FlowDroid is that it only accepts APIs as taint sources and sinks, rather than specific privacy API parameters. We resolved this issue by adapting FlowDroid's source-sink manager, `AccessPathBasedSourceSinkManager`, to allow any program statement and variables, including API parameters, to serve as a source in taint tracking. Based on that, we executed FlowDroid using the privacy API parameters recorded in PiCo MetaDB as sources and 138 networking and file access APIs collected from prior studies [66, 70, 87] as sinks. Then, we identified the IPSVs by analyzing the propagation of privacy API parameters within PiCo *SDK*s through a

`TaintPropagationHandler`. For PVP #5, we track the exposure of user personal data to the Internet by running FlowDroid with a compiled list of 26 sources and 138 sinks based on prior studies [66, 70, 87]. For PVP #8 (broken privacy delegation), we used the privacy parameters in the PICO *wrapper*'s privacy APIs as sources and the privacy APIs of the encapsulated PICO *SDK*s as sinks. We report a PVP #8 risk if an expected taint path (or information flow) was not observed.

To perform runtime user preference analysis, we monitor the content on app user interfaces by intercepting all window content change events using the accessibility service (AS) event `TYPE_WINDOW_CONTENT_CHANGED`. Upon each event, we enumerate all the UI elements by traversing the tree of `AccessibilityNodeInfo` objects, and tell whether the elements are related to privacy by matching crafted regex (as noted above). We exercise the privacy-related UI elements by issuing automated clicks by `AccessibilityNodeInfo.performAction`. In addition, the runtime value analysis is built upon Frida [8] – one of the most popular app dynamic instrumentation tools for Android. The dynamic analysis is implemented with 804 lines of Java code and 1,094 lines of Python code.

5.2 Evaluation

Groundtruth datasets. To assess the effectiveness of PICOSCAN in detecting PICO *risk*s, we constructed two groundtruth datasets: the app evaluation dataset and the PICO METADB evaluation dataset.

The app evaluation dataset contains 40 randomly selected apps along with their PICO *risk*s related to app-related *PVP*s (#1-#3). We installed these apps on a Pixel 6 device, manually exercised the apps, and used Frida to record the stack traces and parameters of any PICO *SDK* initialization and privacy APIs invoked by the apps [8]. To confirm the PICO *risk*s of these apps, we manually checked the stack traces and parameters based on the definition of the app-related *PVP*s. For example, we flagged *PVP* #1 if a PICO *SDK* initialization API was invoked by the app's first-party code without its privacy APIs. The appearance of any *PVP* in the apps is considered an instance of PICO *risk*, and in total, we identified 14 instances of PICO *risk* across 13 apps.

The PICO *SDK* evaluation dataset includes 10 randomly sampled PICO *SDK*s from the total of 65, with three of them being PICO *wrapper*s. To execute the SDKs, we selected the app with the highest number of downloads among all apps in APPSET that utilize each SDK. We installed, ran, and analyzed these apps containing the PICO *SDK*s using similar techniques as described above. We tailored the manual confirmation process to verify the presence of PICO *risk*s related to PICO *SDK* and PICO *wrapper*. For instance, to detect *PVP* #4, we manually identified methods in the apps that return privacy configurations through reverse-engineering, and then hooked into these methods to retrieve the default configuration." This process, particularly the manual reverse engineering and *PVP* confirmation, was labor-intensive and required two security researchers nine days to complete. In total, we identified 16 instances of PICO *risk* in these apps. Table 3 provides the breakdown of all the *PVP*s in the two groundtruth datasets.

Table 3: Groundtruth datasets

Dataset	*PVP*								
	#1	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**
App Groundtruth	11	2	1	-	-	-	-	-	-
PICO *SDK* Groundtruth	-	-	-	6	1	1	3	3	2

Evaluation results. We ran PICOSCAN on the apps in the two groundtruth datasets on a computer equipped with two AMD EPYC 7742 processors and 256 GB of memory. On average, each app took 425.7 seconds to finish the analysis. Then, we compared the results with the groundtruth datasets. Overall, PICOSCAN achieves 87.5% precision and 77.8% recall for detecting PICO *risk*s caused by apps, and 85.7% precision and 90.0% recall for detecting PICO *risk*s in PICO *SDK*s and PICO *wrapper*s.

• *False positives*. PICOSCAN reported five false positive PICO *risk*s: two related to PVP #1, two related to PVP #6, and one related to PVP #8. The primary cause of these false positives is the inaccurate and incomplete call graphs generated for the apps. For instance, two apps employ dynamic method dispatching or callbacks when invoking privacy APIs of PICO *SDK*s. PICOSCAN erroneously flagged these two apps with PVP #1 (missing configuration) because static analysis failed to identify the privacy APIs on the call graph, even though they were indeed invoked. Another app was flagged with PVP #8 (broken privacy delegation) because a wrapper SDK failed to call privacy APIs of its underlying PICO *SDK*s. However, upon investigation, we discovered that the reported PICO *wrapper*, Vungle, was not actually a wrapper SDK. PICOSCAN mistakenly identified it as a wrapper due to overestimation in call graph construction, which connected another SDK's initialization API to that of the Vungle SDK. Other false positives were also attributed to the inaccuracies of static analysis.

• *False negatives*. PICOSCAN reported four false negatives, with two related to PVP #1 and two related to PVP #4. The causes are twofold. First, for PVP #1, PICOSCAN failed to identify missing privacy API invocations in Unity3D. Specifically, Unity3D employs the same API for three different laws, with the first API parameter specifying which law. Due to the limitation that PICOSCAN does not differentiate APIs based on parameters, it mistakenly considers privacy APIs for CCPA and COPPA to be invoked, even though the API is intended solely for configuring GDPR. Second, for PVP #4, PICOSCAN detects the default value of privacy APIs by monitoring the values of IPSVs identified through static analysis. However, certain SDKs, such as Ogury, do not utilize IPSVs for storing privacy configurations; rather, they store the configuration directly into persistent storage (i.e., "`IS_CHILD_UNDER_COPPA`" in shared preferences). Consequently, PICOSCAN was unable to determine the default value for the absence of IPSVs.

6 Measurement

For measuring PICO *risk*s in the wild, we collect a set of 48,305 mobile apps (called APPSET) by crawling Google Play [57] using a Go-based crawler [33] in October 2023. Specifically, we collected the package names of all 7,753,757 apps that have ever appeared on Google Play (as recorded by AndroZoo [34]), and randomly shuffled the package names before crawling. We followed the order of the shuffled package names and crawled an app if it was relatively new (updated after January 1, 2020) and available in both the United States (including California) and the EU[2]. Among the apps in the APPSET, at least 827 were expected to comply with COPPA since they are committed to follow the Play Families Policy [9]. We report GDPR and CCPA risks over the entire APPSET, and COPPA risks over the 827 apps.

6.1 Landscape

Usage of PICO *SDK*s. Running *PUI* on the APPSET, we found that 58 PICO *SDK*s in the PICO METADB are actively used by apps in the APPSET, i.e., SDK initialization APIs are reachable from app entry points. These SDKs appear in a total of 24,844 (51.4%) apps, with AdMob appearing in the highest number (14,497) of apps and AppNexus in the lowest number (3) of apps. It is worth noting that not all apps use the most up-to-date SDK version. In particular, 12,480 (50.2%) of the 24,844 apps invoke the legacy version of at least one PICO *SDK* that does not implement the corresponding privacy APIs. Also, we identified a total of 22 PICO *wrapper*s from the apps using any PICO *SDK*, and these PICO *wrapper*s are used by 1,769 apps in the APPSET. Each PICO *wrapper* encapsulates an average of 3.08 SDKs, with *AppLovin* having the maximum number of 13 SDKs in a single app. An app can also use multiple PICO *wrapper*s. For example, `com.zloong.eu.dr.gp` app contains nine PICO *wrapper*s, i.e., *AdMob*, *Ironsource* and *AppLovin*, etc.

Overall PICO *risk*s. Table 4 shows the PICO *risk*s that violate privacy principles for three roles, i.e., apps, PICO *SDK*s, and PICO *wrapper*s. Specifically, at least 7,880 apps (31.7% of 24,844) fail to adhere to the privacy principles for apps and are affected by at least one of PVP #1 to PVP #3. The majority of the apps are affected by PVP #1, missing privacy configuration. Among the PICO *SDK*s in PICO METADB, 36 violate the privacy principles for individual PICO *SDK*s and are affected by at least one of PVP #4 to PVP #6. Most (28) of the 36 PICO *SDK*s use a privacy-invading default value for at least one of their privacy APIs (i.e., PVP #4). Additionally, out of the 22 PICO *wrapper*s, 16 are affected by at least one of PVP #7 to PVP #9, thus violating the complete privacy delegation principle.

6.2 PICO Risks Caused by Apps

Missing privacy configuration (PVP #1). Among the 24,844 apps using any PICO *SDK*, 7,818 (31.5%) missed privacy configurations, i.e., they use PICO *SDK*s but fail to set up their privacy APIs. The top 40 apps with the most missed privacy APIs lack, on average, 15 privacy APIs across at least eight PICO *SDK*s. These apps include very popular ones installed by over 10 million users, such as `com.thinkyeah.smartlockfree`, which lacks 24 privacy API calls in 10 PICO *SDK*s. We also found for 16 PICO *SDK*s, over 50% of the apps using each of them are affected by missing privacy configurations. Notable examples include Criteo (99.2%), Amplitude (94.3%), Bytedance (79.0%), and AppLovin (50.2%).

By examining the distribution of the affected apps and the developer documentations of the 39 affected PICO *SDK*s involving PVP #1, we were able to infer a few potential reasons. First, we found that the way in which PICO *SDK*s present privacy configuration options or APIs to app developers is a potential key factor for missing configurations. A large portion (25 out of 39, 64.1%) of PICO *SDK*s document privacy configuration APIs as advanced and optional features (in contrast to regular steps for SDK integration, see below), up to individual app developers to identify and adopt such advanced features. In contrast, documentations of the other 14 PICO *SDK*s (35.9%) suggest adoption of their privacy APIs as part of the regular steps for integrating the SDKs. Although adoption of their privacy APIs are still not mandatory, they may better encourage apps developers to identify and use privacy APIs. The latter group of PICO *SDK*s is significantly less affected by missing configurations compared to the former group, with a Q3 of 64.3% for the former group (i.e., a quarter of the SDKs have a missing configuration rate of over 64.3%) and a Q3 of 32.9% for the latter group, and average missing configuration rates of 43.2% and 23.8%, respectively. We released the list of the affected PICO *SDK*s in the above two groups, their documents and distribution of the percentage of apps missing privacy configurations for PICO SDKs at the time of our study online [17]. Second, in addition to documents, PICO *SDK*s also provide different levels of tooling for app developers to correctly integrate them. Take the Criteo and AppLovin SDK as an example. AppLovin provides a mediation debugger that help app developers to visually debug problematic integration of the SDKs, including configurations by privacy APIs, while Criteo does not provide similar tooling. This could be one potential reason why apps using AppLovin are much less affected than those using Criteo. Third, from the apps' perspective, it might be the case that more popular apps may have greater resources for privacy related engineering and testing. We noticed that only a small fraction (4.7%) of apps using Criteo had more than 1,000,000 installations, while a larger proportion (27.9%) of apps using AppLovin achieved such a milestone. Overall, although many PICO *SDK*s provide privacy configurability, a substantial number of Android apps have yet to adopt them. Hence, the Android ecosystem could greatly benefit from more effective tools, such as those provided by SDK providers, to ensure developers' adoption of these privacy APIs when integrating PICO *SDK*s.

[2] App availability in the EU is determined by a Spain IP.

Table 4: Overall PICO *risk*s

PICO Risks Caused by Apps			PICO Risks Caused by PICO SDKs			PICO Risks Caused by PICO Wrappers		
PVP	**# Apps**	**# Involved PICO SDKs**	**PVP**	**# PICO SDKs**	**# Affected Apps**	**PVP**	**# PICO Wrappers**	**# Affected Apps**
PVP #1	7,818	39	PVP #4	28	20,201	PVP #7	11	322
PVP #2	259	6	PVP #5	16	14,869	PVP #8	12	232
PVP #3	56	3	PVP #6	8	2,097	PVP #9	3	282
Subtotal	7,880	40	Subtotal	36	20,302	Subtotal	16	591

Notably, we don't have ground truth for what the app developers experienced and what development tools they actually used when integrating PICO SDKs. Hence, we may not judge all reasons why app developers came with PVP #1. However, based on our study, the reasons we discussed above could at least partially cause PVP #1 or make it difficult to avoid in the wild. The similar is true for our causal analysis of other PVPs.

Configuration disregarding or inconsistent with user privacy preference (PVP #2). We found that 259 apps, while invoking privacy APIs, fail to configure the APIs correctly according to user preferences. For example, 248 apps choose to hard-code the parameters of privacy APIs (in six PICO *SDK*s) which allows unexpected data exposure regardless of their preferences. We suspect that this choice is driven by factors such as the revenues and privacy engineering cost. For example, many apps often serve as ad publishers, which rely on data collection and personalized ads features to maximize revenues. Hard-coding privacy APIs allows them to quickly enable such features with minimal privacy engineering efforts.

In addition, around 8.8% apps using PICO *SDK*s are found to collect user privacy preferences through privacy dialogs. We found that 11 of these apps still invoke privacy APIs with parameters that enable data exposure, even when the privacy dialogs are declined. A closer inspection of these apps reveals that such discrepancies may stem from app developers' limited knowledge of privacy compliance. For example, the `socksrevive.plugin.dragon` app displays a privacy consent dialog stating that "*By agreeing you are confirming that you are over the age of 16 and would like to personalize your ad experience*" and asks users to consent by clicking a "*YES, I AGREE*" button. The presence of the dialog indicates that the app developer is at least aware of the importance of requesting privacy consent from users. However, the app only uses the dialog as a gating mechanism for new user registrations but fails to propagate the consent status to any other app components, including mobile ads, such as the BidMachine SDK. This problem potentially suggests that the app developer may assume that their app is privacy-compliant as long as a dialog is provided, without fully understanding the necessity of supporting compliance by configuring each individual component.

Erroneous configuration (PVP #3). We observed that eight PICO *SDK*s, such as AppLovin [23] and Ironsource [29], in the PICO METADB require that privacy APIs and initialization APIs be called in a specific order, which if violated leads to erroneous and invalid privacy configurations. Although the percentage is not high, there are still 56 apps that violate this requirement on three SDKs: AdMob, Ironsource, and Unity Ads. In total, these apps were installed over 20 million times. Given that most apps using these three SDKs meet the requirement, we believe that the issue with the 56 apps is primarily caused by implementation flaws of their developers, considering that all eight of these PICO *SDK*s explicitly noted the order constraints in their documents. We released the list of the eight SDKs and their documents online [17].

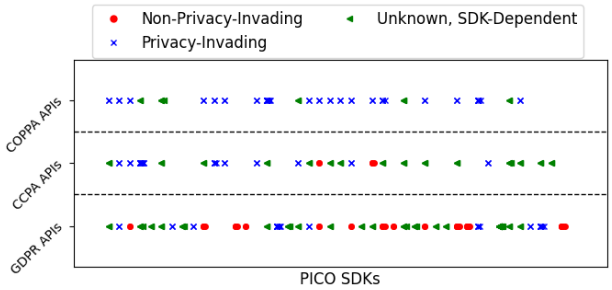

Figure 3: The distribution of default privacy configurations

6.3 PICO Risks Caused by PICO *SDK*s

Privacy-invading by default (PVP #4). Among the 65 PICO *SDK*s, 28 (43.1%) are found to have at least one privacy API with a privacy-invading default configuration, affecting 20,201 apps in APPSET. Figure 3 shows the distribution of the default values of PICO *SDK* privacy APIs.

The privacy-invading default values are not evenly distributed among the privacy APIs for different laws, i.e., the GDPR APIs of eight SDKs, the CCPA APIs of nine SDKs, and the COPPA APIs of 19 SDKs. A potential reason for this disparity could be the varied responsibility models adopted by the PICO *SDK*s, caused by the differing scopes of applicability across different laws. For instance, GDPR, with its broader reach, generally applies to any apps serving individuals within the EU and those based in the EU, while COPPA specifically targets apps serving children under the age of 13 in the US. As a result, many PICO *SDK*s have largely considered GDPR compliance as a baseline requirement, without privacy-invading default values. However, regarding COPPA, SDKs have generally adopted a model where it is the responsibility of app developers to inform SDKs of COPPA's applicability (while, without being informed, the SDK assumes that users are not children and collects their data by default, as described by Unity Ads [19]).

Another observation is that PICO *SDK* developers often do not disclose the default configuration of the privacy

APIs, which goes against the best practices of software configuration [74]. Even more concerning, some SDKs are potentially concealing their default configurations from public scrutiny. An example is the Meta Audience Network SDK. The SDK has a "*limited data use*" (LDU) feature for achieving privacy compliance with US state laws including CCPA. By default, the SDK uses `new String[0]` parameter on its `setDataProcessingOptions` to completely disable LDU. The way that this default parameter is specified is suspicious. Upon initialization, the SDK sets the parameter by reading the `DATA_PROCESSING_OPTIONS_KEY` key from the shared preference. Since LDU has never been configured before, the shared preference will return a *null*, which is then replaced by the SDK with a `new String[0]` value. Interestingly, the above key to access shared preference is obfuscated with a byte-wise XOR encoding by Facebook (in `com.facebook.ads.redexgen.X`), while all the other keys are left intact. This appears uncommon since the only benefit of obfuscating this key, as far as we can imagine, is to make external reviews more challenging to identify the default LDU enablement status.

We want to note that the discovered privacy-invading default values may not fully represent all PICO *SDK*s. In Figure 3, we label many APIs as "*Unknown, SDK-Dependent*" (◀) because the extracted API parameter does not match any value recorded in the PICO METADB. For example, Vungle's COPPA API has a default `COPPA_NOTSET` parameter, which does not match either of the recorded values, i.e., `COPPA_ENABLED` and `COPPA_DISABLED`. While such default values could also invade privacy, we did not report them as PICO *risk*s related to PVP #4.

Ambiguous consent configuration (PVP #5). Sixteen PICO *SDK*s have an ambiguous consent configuration in their privacy APIs, which, even if well configured, fails to stop the collection of user personal data. These PICO *SDK*s lead to unexpected and bypassing privacy exposure to 14,869 apps. One prominent piece of personal data being collected is the advertising ID, a unique ID for advertising. We believe this issue is mainly caused by the lack of standards in implementing privacy APIs. Even though PICO *SDK*s provide seemingly similar privacy APIs (e.g., for collecting user consent), they don't agree on the same privacy assurance.

Ineffective configuration (PVP #6). We identified that eight PICO *SDK*s have at least one privacy API that can become ineffective due to the presence of other program paths that affect its execution. These PICO *SDK*s are integrated by 2,097 apps in APPSET. Interestingly, all the eight PICO *SDK*s, such as MobileFuse and Appodeal, share IPSVs between their GDPR and CCPA APIs. As a result, the GDPR configuration of these PICO *SDK* is affected, e.g., overridden, by their CCPA configurations. The reason for this lies in the fact that, similar to MyTarget, these PICO *SDK*s have CCPA APIs intended to offer essentially the same privacy guarantees as their GDPR APIs, i.e., obtaining user consent for data collection. We believe this is a result of a common misinterpretation of CCPA. In reality, CCPA relies on implied user consent for data collection while requiring options that allow users to opt-out of selling and sharing personal data. Hence, obtaining user consent may not lead to full CCPA compliance, as it does not fulfill the fundamental CCPA requirement regarding opting out of selling and sharing with third parties directly.

6.4 Privacy Risks Caused by PICO *wrapper*s

Uneven privacy support (PVP #7). The support of privacy laws across PICO *SDK*s varies. For example, only 27 (41.5%) of the 65 PICO *SDK*s provide privacy APIs for all three privacy laws. Particularly, 14 out of the 22 PICO *wrapper*s offer APIs for all these laws. However, most of them (11) encapsulate at least one PICO *SDK* that does not provide APIs for all these laws, resulting in uneven privacy support.

Broken privacy delegation (PVP #8). Even with privacy APIs to the same privacy law, 12 wrappers (54.5% out of 22) fail to propagate the privacy configurations into at least one encapsulated PICO *SDK*, resulting in inconsistent privacy configurations in 232 apps. Exploring the documents and design of PICO *SDK*s allows us to make several reasonable guesses. First, the privacy APIs of PICO *SDK*s often appear in the optional "advanced settings" documents. PICO *wrapper*s are not required to configure privacy exposure of PICO *SDK* with these APIs before they can access the main functionality of PICO *SDK*s. As a result, many PICO *wrapper*s may refrain from calling the APIs unless it becomes mandatory. Second, even in the case that PICO *wrapper*s prioritize privacy propagation, their ability to accurately propagate configurations is often challenged by the opacity of other SDKs, and the mismatching semantics of privacy APIs. For instance, Facebook's CCPA API, `AdSettings.setDataProcessingOptions`, implements the "*Limited Data Use*" mode to ensure compliance with state laws such as CCPA, for individuals in California, Colorado, or Connecticut. Unfortunately, this may not be immediately evident to PICO *wrapper*s, which commonly interpret CCPA as "*do not sell/share*" or "*opt-out of selling/sharing*". Therefore, as we suggest in Section 7, adopting standard privacy APIs (e.g., the IAB compliance framework [11]) may help minimize the confusion caused by the custom privacy API implementations of various SDKs.

Conditional propagation (PVP #9). Furthermore, there can be an issue even if there is a path to propagate privacy configurations. For example, three PICO *wrapper*s, i.e., Appodeal, BidMachine, and Admost, have additional conditions such as checking IP geolocation when propagating privacy configurations to wrapped PICO *SDK*s. These conditions affect the propagation of privacy configurations to at least 18 PICO *SDK*s.

In Figure 4, we illustrate the PICO *risk*s that occur between a series of PICO *wrapper*s (left) and PICO *SDK*s (right).

7 Discussion

Disclosure. We reported all the PICO *SDK*s to related vendors including SDK and app developers, which were acknowledged

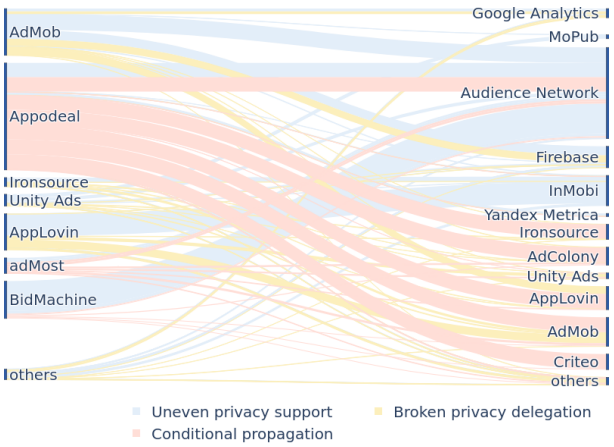

Figure 4: PICO *risk* by PICO *wrapper*s

by some vendors recently. For example, Vungle confirmed and fixed the issues in its latest version 7.3.1; five other SDK vendors appreciated the issues we reported and are investigating them. We will update the vendors' acknowledgement at [17].

Suggestions to stakeholders. Achieving privacy compliance requires collective efforts from multiple involved stakeholders.

• *SDK developers*. Many PICO *risk*s, particularly those associated with PICO *wrapper*s, result from ad hoc, non-standard, and varying implementations of privacy APIs across various SDKs. Therefore, SDK developers may consider adopting standard APIs, such as the IAB framework [11], to offer uniform and interoperable privacy API implementations. Fully integrating the standard APIs is challenging and can take a long time because it requires significant changes to both the client-side and server-side code of PICO *SDK*s, e.g., to fully support IAB signals. As a short-term and interim solution, without overhauling the SDKs' existing code, SDK developers may consider developing adapters on the client-side to parse, propagate, and map standard APIs to the current offerings of privacy APIs. Furthermore, missing configuration is the most common risk in the use of PICO *SDK*s. To mitigate this risk, it may be reasonable for SDK developers to implement a gating mechanism or debugger (as done by AppLovin) that verifies the presence of privacy APIs for each PICO *SDK* integration.

• *App developers*. Apps often incorporate many SDKs, and the privacy configurations of these SDKs can be erroneous and inconsistent. However, in the regular app development lifecycle, there are no review or testing procedures specifically designed to address such errors and inconsistencies. Therefore, app developers may consider implementing dedicated reviews or test cases before delivering their apps to the general public.

• *Privacy law enforcement agencies*. The current PICO *SDK* ecosystem has generally adopted a *confusing responsibility model*, where PICO *SDK*s act as data controllers benefiting from the processing of user-sensitive data, while only the apps (rather than PICO *SDK*s) are exposed to pursuits by law enforcement agencies [53–55]. Hence, PICO *SDK* providers have little motivation to support materialized compliance, such as incorporating privacy-compliant defaults for COPPA. To address this issue, a more effective approach for law enforcement agencies could be to directly perform compliance checks on the implementation of popular PICO *SDK*s, which can also reduce compliance risks for any apps using these SDKs.

Limitation Discussion. We developed common-sense principles and PVPs to model expected practices for using and implementing PICO *SDK*s. Though they cover various types of privacy risks, they are not exhaustive. Other relevant principles and PVPs may exist, which we will explore in future studies. Our PICOSCAN framework primarily relies on static app/SDK analysis, supplemented with dynamic analysis for some PVPs. Due to inherent limitations in processing dynamically loaded or triggered features, the framework's results may not fully reflect app/SDK behavior. This study primarily examines client-side privacy configurations and their implications for privacy. However, apps and SDKs may also use server-side privacy configurations, potentially conflicting with or overriding client-side settings. Analysis of server-side configurations and their interactions with client-side configurations is left for future research.

In addition, this study reported privacy risks that result in technical violations of several privacy principles. However, it is possible that these risks do not actually violate privacy laws. For example, when assessing risks for the CCPA, we assume that all apps available in California are subject to its provisions. In fact, the applicability of the CCPA is contingent upon threshold requirements such as gross annual revenue or the number of California residents whose data are processed. It is likely that the apps we reported on do not meet these requirements, and therefore their privacy risks regarding the CCPA may not constitute actual violations. A similar situation may arise with privacy risks related to the GDPR, especially in cases where the risks manifest in apps that do not use consent as their legal basis. Analyzing these non-technical factors and determining actual violations to privacy laws require the involvement of legal experts and additional non-technical information, which we consider outside the scope of this study. Nevertheless, we firmly believe that identifying privacy risks is highly valuable, as it helps narrow down the analysis of actual violations. Our PICOSCAN implementation identifies PICO *SDK*s and their privacy APIs by matching the package names and API signatures, respectively, as recorded in the PICO METADB. As a result, the implementation is not obfuscation-resistant and cannot cover apps that obfuscate the PICO *SDK* code, and hence the measurement results in § 6 represent only a lower bound of the actual PICO *risk*s.

8 Related Work

Over the past years, a series of prior studies have evaluated the compliance of the privacy policies of mobile apps, and reported incomplete, vague, incorrect, or misleading privacy policies regarding data collection, sharing, and data usage purposes [35, 36, 43, 66, 81, 82, 85]. Similar work also studied

the privacy policies of third-party SDKs [87]. Some other studies analyzed the issues in app consent notices, and reported the data collection and third-party tracking that either without prior consent notices [42, 50, 61–63, 68, 69], or with notices that do not meet the *freely given*, *specific*, *informed* and *unambiguous* requirements of GDPR [41, 61, 62, 64, 69, 78].

Two prior studies [51, 71] are related to our study since they also discussed the privacy configuration issues of third-party SDKs. Specifically, Reyes et al. [71] evaluated the COPPA options by analyzing the traffic generated by child-directed apps and reported that many of the apps fail to set up these options although their users are children. This study, relying on traffic analysis, was unable to attribute the traffic to the app code and identify the actual causes in the code that violate privacy principles, as has been done in our study. Du et al. [51] evaluated inconsistencies between user withdrawal choices and third-party library data collection. This privacy issue can stem from apps failing to configure SDKs for users' data withdrawal (part of PVP #2) or from SDKs failing to fulfill their data withdrawal promises by stopping data collection (part of PVP #5). However, PVP #2 and #5 in our study are more general, covering more privacy rights beyond withdrawal (or opt-out), such as user consent and child privacy rights. Particularly, compared to both related studies, we comprehensively studied the PiCo *SDK* ecosystem and developed several privacy principles that enable the analysis of a broad range of privacy risks. These risks are represented by nine distinct violation patterns (PVP #1-#9) and cover different roles in the ecosystem, including apps, PiCo *SDK*s, and PiCo *wrapper*s. Many of these risks arise from the interactions between these components and have not been systematically studied before.

9 Conclusion

In this paper, we conduct the first comprehensive measurement study on the privacy compliance risks of privacy-configurable SDKs (PiCo *SDK*s) and the apps using them. We built a database of 65 popular PiCo *SDK*s and analyzed over 48,000 Google Play apps to assess their adherence to privacy principles, covering both the app developer's code and the SDKs' code. This analysis is facilitated by our development of a compliance risk analysis framework, PiCoScan. Our findings unveil significant privacy risks associated with both app developers and third-party SDKs, highlighting systemic issues that underscore the need for improved privacy practices and configurations in the PiCo *SDK*s' ecosystem.

Acknowledgments

We thank our shepherd and the anonymous reviewers for their valuable comments and suggestions. This work is supported in part by NSF CNS-2330265, CNS-1801432 and CNS-2154199. This research was supported in part by Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute.

References

[1] Admob: Automatically collected user properties. https://support.google.com/admob/answer/9755590?hl=en.

[2] Admob: Data collected and shared automatically. https://developers.google.com/admob/android/privacy/play-data-disclosure#data_collected_and_shared_automatically.

[3] Admob's gdpr api document. https://developers.google.com/admob/android/privacy#load-and-show-form.

[4] Android library statistics. https://www.appbrain.com/stats/libraries.

[5] Appodeal's coppa api document. https://docs.appodeal.com/android/data-protection/coppa.

[6] Do i need to present a gdpr banner to ip addresses outside of gdpr regions? https://law.stackexchange.com/questions/84820/do-i-need-to-present-a-gdpr-banner-to-ip-addresses-outside-of-gdpr-regions.

[7] Facebook sdk best practices for gdpr compliance. https://developers.facebook.com/docs/app-events/gdpr-compliance/.

[8] Frida - dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers. https://frida.re/.

[9] Google play families policies. https://support.google.com/googleplay/android-developer/answer/9893335?hl=en.

[10] Hypermx's coppa api document. https://documentation.hyprmx.com/android-sdk/getting-started/privacy#age-restricted-user-flag.

[11] Iab ccpa compliance framework for publishers technology companies. https://iabtechlab.com/standards/ccpa/.

[12] Ip addresses and the gdpr. https://www.dbswebsite.com/blog/ip-addresses-gdpr/.

[13] Ironsource's privacy api document. https://developers.is.com/ironsource-mobile/android/regulation-advanced-settings/#step-1.

[14] Ogury's coppa api document. https://ogury-ltd.gitbook.io/android/.

[15] Privacy—consent, age-related flags, and data apis. https://dash.applovin.com/documentation/mediation/unity/getting-started/privacy.

[16] soot-oss/heros. https://github.com/soot-oss/heros.

[17] Supplementary materials website. https://sites.google.com/view/picoscan/home.

[18] Tappx's coppa api document. https://developers.tappx.com/en/android/COPPA/.

[19] Unity ads' coppa api document. https://docs.unity.com/ads/en-us/manual/COPPACompliance#UserLevelCOPPA.

[20] Accessibilityservice | android developers. https://developer.android.com/reference/android/accessibilityservice/AccessibilityService, 2023.

[21] Amazon mobile ads. https://developer.amazon.com/apps-and-games/mobile-ads, 2023.

[22] Applovin - a comprehensive mobile gaming solution. https://www.applovin.com/, 2023.

[23] Applovin's coppa api document. https://support.applovin.com/hc/en-us/articles/13891460597261-Privacy, 2023.

[24] Children's advertising review unit. https://bbbprograms.org/programs/all-programs/childrens-advertising-review-unit, 2023.

[25] Children's online privacy protection rule ("coppa"). https://www.ftc.gov/legal-library/browse/rules/childrens-online-privacy-protection-rule-coppa, 2023.

[26] Complete guide to gdpr compliance. https://gdpr.eu, 2023.

[27] Consumer data protection act. https://law.lis.virginia.gov/vacodefull/title59.1/chapter53/, 2023.

[28] Inmobi - mobile sdk for app monetization. https://www.inmobi.com/sdk, 2023.

[29] Ironsource's coppa api document. https://developers.is.com/ironsource-mobile/android/regulation-advanced-settings/#step-1, 2023.

[30] Personal information protection law of the people's republic of china - pipl. https://personalinformationprotectionlaw.com/, 2023.

[31] Sharedpreferences | android developers. https://developer.android.com/reference/android/content/SharedPreferences, 2023.

[32] Soot - a framework for analyzing and transforming java and android applications. https://soot-oss.github.io/soot/, 2023.

[33] 89z. Google play crawler. https://github.com/89z/googleplay.

[34] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In MSR'16.

[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 USENIX security 2019, pages 585–602, 2019.

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

[37] Inc. Anjuna Security. Privacy by default with anjuna and the nist framework. https://www.anjuna.io/resources/privacy-by-default-with-anjuna-and-the-nist-framework, 2023.

[38] Appodeal. Ccpa privacy policy - appodeal. https://appodeal.com/ccpa-privacy-policy/, 2023.

[39] Appodeal. Gdpr and ccpa. https://docs.appodeal.com/android/data-protection/gdpr-and-ccpa, 2023.

[40] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. Acm Sigplan Notices, 49(6):259–269, 2014.

[41] Dino Bollinger, Karel Kubicek, Carlos Cotrini, and David Basin. Automating cookie consent and {GDPR} violation detection. In USENIX Security 2022, pages 2893–2910, 2022.

[42] Duc Bui, Brian Tang, and Kang G Shin. Do opt-outs really opt me out? In CCS 2022, pages 425–439, 2022.

[43] Duc Bui, Yuan Yao, Kang G Shin, Jong-Min Choi, and Junbum Shin. Consistency analysis of data-usage purposes in mobile apps. In CCS 2021, pages 2824–2843, 2021.

[44] Lee A Bygrave. Data protection by design and by default: deciphering the eu's legislative requirements. Oslo Law Review, 4(2):105–120, 2017.

[45] Yinzhi Cao, Yanick Fratantonio, Antonio Bianchi, Manuel Egele, Christopher Kruegel, Giovanni Vigna, and Yan Chen. Edgeminer: Automatically detecting implicit control flow transitions through the android framework. In NDSS, 2015.

[46] Colorado Attorney General. Colorado privacy act (cpa). https://coag.gov/resources/colorado-privacy-act/, 2023.

[47] AppLovin Corporation. California privacy notice - applovin. https://www.applovin.com/ca-privacy-notice/, 2023.

[48] Cybersecurity and Infrastructure Security Agency. Art. 25 gdpr - data protection by design and by default. https://gdpr-info.eu/art-25-gdpr/, 2018.

[49] Cybersecurity and Infrastructure Security Agency. Shifting the balance of cybersecurity risk: Principles and approaches for security-by-design and -default. https://www.cisa.gov/sites/default/files/2023-04/principles_approaches_for_security-by-design-default_508_0.pdf, 2023.

[50] Martin Degeling, Christine Utz, Christopher Lentzsch, Henry Hosseini, Florian Schaub, and Thorsten Holz. We value your privacy... now take some cookies: Measuring the gdpr's impact on web privacy. arXiv preprint arXiv:1808.05096, 2018.

[51] Xiaolin Du, Zhemin Yang, Jiapeng Lin, Yinzhi Cao, and Min Yang. Withdrawing is believing? detecting inconsistencies between withdrawal choices and third-party data collections in mobile apps. In IEEE S&P 2024, pages 14–14. IEEE Computer Society, 2023.

[52] FTC. Ftc issues final commission report on protecting consumer privacy. https://www.ftc.gov/news-events/news/press-releases/2012/03/ftc-issues-final-commission-report-protecting-consumer-privacy, 2012.

[53] FTC. Android flashlight app developer settles ftc charges it deceived consumers. https://www.ftc.gov/news-events/news/press-releases/2013/12/android-flashlight-app-developer-settles-ftc-charges-it-deceived-consumers, 2013.

[54] FTC. Google and youtube will pay record $170 million for alleged violations of children's privacy law. https://www.ftc.gov/news-events/news/press-releases/2019/09/google-youtube-will-pay-record-170-million-alleged-violations-childrens-privacy-law, 2019.

[55] FTC. Ovulation tracking app premom will be barred from sharing health data for advertising under proposed ftc order. https://www.ftc.gov/news-events/news/press-releases/2023/05/ovulation-tracking-app-premom-will-be-barred-sharing-health-data-advertising-under-proposed-ftc, 2023.

[56] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns: elements of reusable object-oriented software. Pearson Deutschland GmbH, 1995.

[57] Google. Android apps on google play. https://play.google.com/store/apps.

[58] Google. Building products that are private by design for everyone. https://safety.google/intl/en_us/principles/, 2023.

[59] Google Play. Copy my data: Transfer content - apps on google play. https://play.google.com/store/apps/details?id=com.mediamushroom.copymydata&hl=en, 2023.

[60] GooglePlay. Callapp: Caller id block. https://play.google.com/store/apps/details?id=com.callapp.contacts, 2023.

[61] Xuehui Hu and Nishanth Sastry. Characterising third party cookie usage in the eu after gdpr. In WebSci 2019, pages 137–141, 2019.

[62] Simon Koch, Benjamin Altpeter, and Martin Johns. The {OK} is not enough: A large scale study of consent dialogs in smartphone applications. In USENIX Security 2023, pages 5467–5484, 2023.

[63] Konrad Kollnig, Pierre Dewitte, Max Van Kleek, Ge Wang, Daniel Omeiza, Helena Webb, and Nigel Shadbolt. A fait accompli? an empirical study into the absence of consent to {Third-Party} tracking in android apps. In SOUPS 2021, pages 181–196, 2021.

[64] Célestin Matte, Nataliia Bielova, and Cristiana Santos. Do cookie banners respect my choice?: Measuring legal compliance of banners from iab europe's transparency and consent framework. In 2020 IEEE Symposium on Security and Privacy (SP), pages 791–809. IEEE, 2020.

[65] Wei Meng, Ren Ding, Simon P Chung, Steven Han, and Wenke Lee. The price of free: Privacy leakage in personalized mobile in-apps ads. In NDSS, pages 1–15, 2016.

[66] Yuhong Nan, Xueqiang Wang, Luyi Xing, Xiaojing Liao, Ruoyu Wu, Jianliang Wu, Yifan Zhang, and XiaoFeng Wang. Are you spying on me?{Large-Scale} analysis on {IoT} data exposure through companion apps. In USENIX Security 2023, pages 6665–6682, 2023.

[67] 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.

[68] Trung Tin Nguyen, Michael Backes, Ninja Marnau, and Ben Stock. Share first, ask later (or never?) studying violations of {GDPR's} explicit consent in android apps. In USENIX Security 2021, pages 3667–3684, 2021.

[69] Trung Tin Nguyen, Michael Backes, and Ben Stock. Freely given consent? studying consent notice of third-party tracking and its violations of gdpr in android apps. In CCS 2022, pages 2369–2383, 2022.

[70] Xiang Pan, Yinzhi Cao, Xuechao Du, Boyuan He, Gan Fang, Rui Shao, and Yan Chen. {FlowCog}: Context-aware semantics extraction and analysis of information flow leaks in android apps. In USENIX Security 2018, pages 1669–1685, 2018.

[71] 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 PETS 2018, 2018.

[72] Jerome H Saltzer and Michael D Schroeder. The protection of information in computer systems. Proceedings of the IEEE, 63(9):1278–1308, 1975.

[73] Iskander Sanchez-Rola, Matteo Dell'Amico, Platon Kotzias, Davide Balzarotti, Leyla Bilge, Pierre-Antoine Vervier, and Igor Santos. Can i opt out yet? gdpr and the global illusion of cookie control. In Asia CCS 2019, pages 340–351, 2019.

[74] MadCap Software. The definitive guide to creating api documentation. https://assets.madcapsoftware.com/white-papers/White_Paper-The_Definitive_Guide_to_Creating_API_Documentation.pdf.

[75] State of California - Department of Justice - Office of the Attorney General. California consumer privacy act (ccpa). https://oag.ca.gov/privacy/ccpa, 2023.

[76] State of Connecticut - Office of the Attorney General. The connecticut data privacy act. https://portal.ct.gov/AG/Sections/Privacy/The-Connecticut-Data-Privacy-Act, 2023.

[77] Unity Technologies. Gdpr legal documentation for unity ads. https://docs.unity3d.com/Packages/com.unity.ads@3.3/manual/LegalGdpr.html, 2023.

[78] Christine Utz, Martin Degeling, Sascha Fahl, Florian Schaub, and Thorsten Holz. (un) informed consent: Studying gdpr consent notices in the field. In CCS 2019, pages 973–990, 2019.

[79] Vungle. Vungle sdk. https://support.vungle.com/hc/en-us/categories/200269670-Vungle-SDK, 2023.

[80] Jice Wang, Yue Xiao, et al. Understanding malicious cross-library data harvesting on android. In USENIX Security, 2021.

[81] 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 ICSE 2018, pages 37–47, 2018.

[82] Anhao Xiang, Weiping Pei, and Chuan Yue. Policychecker: Analyzing the gdpr completeness of mobile apps' privacy policies. In CCS 2023, pages 3373–3387, 2023.

[83] 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 at scale. In USENIX Security 2023.

[84] Pan Xiaorui, Wang Xueqiang, Duan Yue, Wang XiaoFeng, and Yin Heng. Dark hazard: Learning-based, large-scale discovery of hidden sensitive operations in android apps. In NDSS 2017, 2017.

[85] Le Yu, Xiapu Luo, Jiachi Chen, Hao Zhou, Tao Zhang, Henry Chang, and Hareton KN Leung. Ppchecker: Towards accessing the trustworthiness of android apps' privacy policies. TSE, 47(2):221–242, 2018.

[86] Xueling Zhang, Xiaoyin Wang, Rocky Slavin, Travis Breaux, and Jianwei Niu. How does misconfiguration of analytic services compromise mobile privacy? In ICSE 2020, pages 1572–1583, 2020.

[87] Kaifa Zhao, Xian Zhan, Le Yu, Shiyao Zhou, Hao Zhou, Xiapu Luo, Haoyu Wang, and Yepang Liu. Demystifying privacy policy of third-party libraries in mobile apps. In ICSE 2023, pages 1583–1595. IEEE, 2023.

[88] Qingchuan Zhao, Chaoshun Zuo, Brendan Dolan-Gavitt, Giancarlo Pellegrino, and Zhiqiang Lin. Automatic uncovering of hidden behaviors from input validation in mobile apps. In IEEE S&P 2020, pages 1106–1120. IEEE, 2020.

Appendix A

Privacy API Metadata in PiCo MetaDB. In Figure 5, we use the Bytedance SDK's initialization and COPPA-related privacy API to show what privacy API metadata looks like. Essentially, in a JSON object, we record the API signatures, associated privacy laws ("`apiType`"), and API parameters and their privacy semantics ("`apiSemantic`"), as well as constraints developers need to follow for the correct use of the APIs ("`apiConstraints`"), etc.

```
{
    "Bytedance": {
      "apis": [
        {
          "apiSignature": "<com.bytedance.sdk.openadsdk.
              api.init.PAGSdk: void init(android.content
              .Context,com.bytedance.sdk.openadsdk.api.
              init.PAGConfig,com.bytedance.sdk.openadsdk
              .api.init.PAGSdk$PAGInitCallback)>",
          "apiType": "Init",
          "apiSemantic": {}
        },
        {
          "apiSignature": "<com.bytedance.sdk.openadsdk.
              api.init.PAGConfig: void setChildDirected(
              int)>",
          "apiType": "COPPA",
          "apiSemantic": {
            "parameterIndex": 0,
            "dataDisablingValues": [1],
            "dataEnablingValues": [0]
          }
        }
      ],
      "apiConstraints": [
        {
          "order": ["COPPA", "Init"]
        }
      ]
    }
}
```

Figure 5: Metadata for a privacy API in PiCo MetaDB

Privacy laws and regulations that current PiCo *SDK*s support. We released the results of our comprehensive review of SDK documents online [17]. We report the privacy laws, regulations, or guidelines supported by the most popular PiCo *SDK*s listed on AppBrain, along with the number of SDKs that claim to support compliance with them using PiCo *SDK* privacy APIs. In total, the PiCo *SDK*s collectively support at least 20 privacy laws, regulations, or guidelines, with GDPR, CCPA, and COPPA being the three most supported laws.

Table 5: Constraints on Privacy API Invocations

PiCo *SDK*	Law	# of APIs	Order Constraint
AppLovin	COPPA	1	[COPPA, Init] [23]
Appodeal	COPPA	1	[COPPA, Init] [5]
Ironsource	COPPA	2	[COPPA, Init] [13]
Ironsource	CCPA	2	[CCPA, Init] [13]
Tappx	COPPA	1	[COPPA, Init] [18]
Unity Ads	COPPA	5	[COPPA, Init] [19]
HyperMX	COPPA	1	[COPPA, Init] [10]
Ogury	COPPA	1	[COPPA, Init] [14]
AdMob	GDPR	1	[GDPR, Init] [3]
AdMob	COPPA	1	[COPPA, Init] [3]

Constraints on privacy API invocations. In Table 5, we present the constraints that app developers need to follow to correctly invoke PiCo *SDK* privacy APIs. In total, eight PiCo *SDK*s require privacy APIs and initialization APIs to be called in a fixed order. Particularly, all of them instruct developers to call COPPA APIs before initialization APIs so that the SDKs can prepare child-directed content, such as ads, during the SDK initialization. These constraints cover a total of 14 unique privacy APIs.

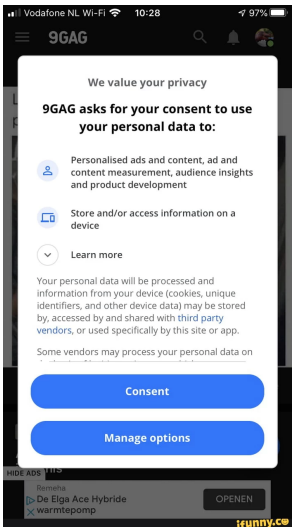

Figure 6: Privacy dialog of the AdMob SDK.

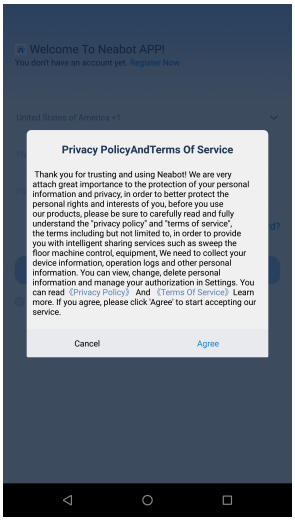

Figure 7: Privacy dialog of `com.jhkj.neabot` app.

Privacy dialogs of real apps and PiCo *SDK*s. In Figure 6, we show the privacy dialog posted by the AdMob SDK in apps that integrate it. This dialog asks users consent to "use your personal data to" offer personalized ads, etc. Figure 7 displays the privacy dialog posted by the first-party code of app `com.jhkj.neabot`, which requests users consent for data collection, etc.