Privacy Settings of Third-Party Libraries in Android Apps: A Study of Facebook SDKs

David Rodriguez
ETSI Telecomunicación, Universidad Politécnica de Madrid
Madrid, Spain
david.rtorrado@upm.es

Joseph A. Calandrino
Washington, D.C., USA
jcalandr@alumni.princeton.edu

Jose M. Del Alamo
ETSI Telecomunicación, Universidad Politécnica de Madrid
Madrid, Spain
jm.delalamo@upm.es

Norman Sadeh
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
sadeh@cs.cmu.edu

Abstract

Previous studies have demonstrated that privacy issues in mobile apps often stem from the integration of third-party libraries (TPLs). To shed light on factors that contribute to these issues, we investigate the privacy-related configuration choices available to and made by Android app developers who incorporate the Facebook Android SDK and Facebook Audience Network SDK in their apps. We compile these Facebook SDKs' privacy-related settings and their defaults. Employing a multi-method approach that integrates static and dynamic analysis, we analyze more than 6,000 popular apps to determine whether the apps incorporate Facebook SDKs and, if so, whether and how developers modify settings. Finally, we assess how these settings align with the privacy practices that developers disclose in the apps' privacy labels and policies.

We observe widespread inconsistencies between practices and disclosures in popular apps. These inconsistencies often stem from privacy settings, including a substantial number of cases in which apps retain default settings over alternatives that offer greater privacy. We observe fewer possible compliance issues in potentially child-directed apps, but issues persist even in these apps. We discuss remediation strategies that SDK and TPL providers could employ to help developers, particularly developers with fewer resources who rely heavily on SDKs. Our recommendations include aligning default privacy settings with data minimization principles and other conservative practices and making privacy-related SDK information both easier to find and harder to miss.

Keywords

Third-party libraries, software development kits, privacy settings, Facebook SDK, Android applications, dynamic analysis, default settings, compliance analysis, privacy labels, privacy policies

1 Introduction

Mobile applications are integral to modern life, from how we communicate with others and entertain ourselves to how we manage our health and finances. Today, mobile app developers rely heavily on software development kits (SDKs). SDKs comprise a collection of software tools and programs, and they routinely incorporate third-party libraries (TPLs) into apps. SDKs help developers produce sophisticated apps rapidly and efficiently. They can do anything from assisting developers in building, deploying, and managing apps to facilitating targeted advertising, social media logins, and much more. Despite the benefits of SDKs, their use in mobile apps has raised concerns regarding privacy.

SDK providers often offer developers free or subsidized services in exchange for collecting and utilizing user data across all apps that use their SDKs [63]. Given their market share and business practices, the data available to the parties behind some widely used SDKs may be significant [63]. The integration of TPLs into Android apps—whether via SDKs or not—introduces a well-documented array of privacy concerns [1, 37, 38, 59, 66, 68, 82]. These libraries can access user data ranging from location to personal communications, potentially without explicit user consent [2] or even developer awareness [11]. Beyond jeopardizing user privacy, this access places app developers at risk of failing to comply with their privacy promises and laws like GDPR or CCPA, which may create obligations from data minimization to consumer controls on personal information [31, 51].

While privacy concerns regarding TPLs may be well known, a gap exists in understanding the impact of TPL privacy settings and their defaults on the privacy of apps. Even if a TPL offers developers extensive privacy-related settings options, prior work suggests that defaults tend to favor functionality over privacy, encouraging practices that may be unnecessary or opaque [14]. Furthermore, developers may be reluctant to change default settings for TPLs [59]. If developers fail to choose appropriate privacy-related TPL settings, apps may not adhere to the developers' privacy promises and obligations, and users may lack knowledge of and control over apps' privacy practices. Addressing these issues requires a deeper investigation into how developers interact with privacy-related TPL settings, ideally contextualized by developers' privacy commitments and obligations.

We consider the privacy-related settings and defaults provided by Facebook (Meta) SDKs for Android, specifically the Facebook Android SDK [19] and Facebook Audience Network SDK [18]. These SDKs offer valuable case studies due to their widespread use and configurable privacy settings, which can significantly influence app

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–15

https://doi.org/XXXXXXX.XXXXXXX

Table 1: Analyzed Facebook SDKs and their modules.

SDK name	Module (SDK) name	Maven artifact ID[a]	Description
Facebook Android SDK	Facebook Core SDK	`facebook-core`	Provides analytics and functionality for other SDK modules.
	Facebook Login SDK	`facebook-login`	Allows users to authenticate using Facebook credentials.
	Facebook Share SDK	`facebook-share`	Enables sharing from an app on Facebook.
	Facebook Messenger SDK	`facebook-messenger`	Integrates Facebook Messenger functionality.
	Facebook App Links SDK	`facebook-applinks`	Supports links into other apps (Android deep links).
	Facebook Marketing SDK	`facebook-marketing`	Facilitates integration of Facebook marketing capabilities.
Audience Network SDK	-	`audience-network-sdk`	Enables advertising and audience monetization.

[a] Maven is a project management system that works with a public repository to integrate Android SDKs. We omit groupId prefix (`com.facebook.android`) from artifact IDs.

privacy practices. Our study focuses on whether and how developers modify these settings. Understanding how developers interact with privacy settings, particularly defaults, is crucial to addressing privacy concerns that TLPs and SDKs raise. Our research employs a combination of static and dynamic analysis, with both approaches providing complementary insights and robust validation of developers' privacy settings choices. We also examine how developers' choices align with apps' stated privacy practices, which can not only uncover discrepancies but also suggest failures of developers to appreciate available choices and implications.

Our analysis reveals that a large proportion of apps retain default SDK privacy-related settings over privacy-enhancing alternative settings. We also identified potential compliance issues in privacy labels and policies when compared with actual app behavior, and many of the concerns are associated with privacy-related SDK settings and their defaults. These findings offer valuable insights into privacy-relevant choices that app developers make when utilizing SDKs, with potential implications for developers, providers of TPLs/SDKs, and policymakers.

Contributions. This study examines the privacy settings available to and made by app developers integrating the Facebook Android SDK and Facebook Audience Network SDK. We employ static and dynamic analysis on more than 6,000 Android apps. Key contributions include:

(1) *Compilation and Detailed Examination of SDK Privacy Settings*: We document and explain the privacy-related settings available in both Facebook SDKs along with their defaults, highlighting their privacy impact.
(2) *Static Analysis*: This analysis assesses Facebook SDK integration in apps, and it compiles and offers insights into certain developer choices regarding privacy-related settings.
(3) *Dynamic Analysis*: We develop new methods that validate and expand on our static analysis. These methods utilize runtime details to confirm SDK integration, determine SDK version, and analyze or confirm privacy-related settings choices. Our findings suggest that static analysis alone may not offer a complete picture.
(4) *Compliance Analysis*: We examine apps' privacy labels and policies, identifying potential discrepancies between declared and apparent privacy practices stemming from SDK use. This analysis highlights possible compliance issues and offers hints of underlying causes.

The remainder of this paper is organized as follows. Section 2 introduces the two Facebook SDKs and their privacy-related settings. Section 3 reviews related work. Section 4 describes our research approach. Section 5 presents findings from our analysis of more than 6,000 apps, discussing SDK integration, privacy-related settings modifications, and compliance issues observed. Section 6 contextualizes our findings with existing developer studies and explores mitigation strategies, and Section 7 addresses the study's limitations. Section 8 concludes and outlines future research directions.

2 Facebook SDKs

We focus on two Facebook SDKs for Android, the Facebook Android SDK and the Audience Network SDK, that are among the most popular social and ad network SDKs in the Android ecosystem [7]. Both SDKs bundle TPLs with apps and provide configurable privacy-related settings that developers can modify through an app's Android Manifest file, app code, or the Meta Developers Platform [57], a centralized developer hub that includes features for managing apps and configuring Facebook SDKs. As Section 4 discusses, our analysis can infer modifications that developers make to these settings via these three methods.

2.1 Facebook Android SDK

The Facebook Android SDK (or "Facebook SDK for Android") offers an extensive range of functionality, including user authentication via Facebook login and content sharing on the platform. This SDK employs a modular architecture that enables developers to integrate specific features independently [28]. All modules rely on essential functionality from the required Facebook Core SDK (or simply Core SDK), discussed below, but developers can otherwise incorporate each module into apps as desired [28, 32]. Table 1 provides an overview of the SDK and its modules.

The Facebook Core SDK module manages privacy-related settings for all modules, providing an array of such settings. These settings have been available in their current form since version 4.34.0 (June 2018) [64]. We compiled privacy-related settings and confirmed the default value for each via both Meta's official documentation [24] and manual examination of the Facebook Core SDK's source code [32]. We provide details on each setting below (and in Table 2). Although Meta's documentation mentions possibilities like delaying data collection "*to obtain user consent or fulfill legal obligations*" [20], the default for each privacy-related setting

Table 2: Privacy-related settings available in the analyzed Facebook SDKs.

SDK	Setting	Definition	Default	Available Since Version (Date)
Facebook Android SDK	AutoLogAppEvents	Enables collection of events and other data for user interaction and engagement tracking.	Enabled	4.34.0 (June 2018)
	AutoInit	Controls automatic initialization of the Facebook Android SDK upon app launch.	Enabled	4.34.0 (June 2018)
	AdvertiserIDCollection	Allows collection of Advertising Identifier (AdID) for personalized advertising.	Enabled	4.34.0 (June 2018)
	LimitEventAndDataUsage	Restricts logged events from being used for purposes other than analytics or conversions.	Disabled	4.34.0 (June 2018)
Audience Network SDK	DataProcessingOptions	Allows constraints on Facebook's use and sharing of user data.	Disabled	5.5.0 (August 2019)
	MixedAudience	Adjusts data collection and use to assist compliance with children's privacy laws.	Disabled	5.6.0 (October 2019)

notably is the option that immediately facilitates or minimizes restrictions on data collection, use, and sharing.

AutoLogAppEvents. Determines whether the SDK collects and logs user interactions, events, and other data, such as app downloads, in-app purchases, ad interactions, email address, name, phone number, physical address details (city, state/province, zip/postal code, and country), gender, and date of birth. Collection and logging is enabled by default. Developers can change this setting via the app's Manifest, code instructions, or the Meta Developers Platform [57].[1]

AutoInit. Controls the automatic initialization of the SDK upon app launch. This setting is enabled by default, which allows the SDK to start functioning immediately and collect data for analytics without additional initialization code. According to Meta's documentation, developers can disable this setting via the app's Manifest. Although the SDK can be re-enabled through code instructions, it is unclear whether this method also allows for disabling the SDK.

AdvertiserIDCollection. Governs the collection of Android's Advertising Identifier (also known as AAID, GAID, or AdID; we use AdID alone for consistency). This identifier is generally used by SDKs to track user activity across apps and deliver personalized advertising [74]. The collection of AdID is enabled by default but can be disabled through the Manifest or via code instructions.

LimitEventAndDataUsage. Can restrict whether logged user interactions and events sent to Facebook are used for purposes beyond analytics and conversions. These restrictions are disabled by default, but when enabled, the data collected will not be used for targeted advertising or detailed marketing profiling [26]. This setting is stored on the device and persists across app launches. Unlike other settings, this one is not detailed in Meta's official documentation, but we identified it in the SDK code. This setting can be changed via code instructions.

[1]The automatic logging of events can also be managed through the Events Manager [58], a tool within Meta's Business Suite [56] that allows for monitoring, analyzing, and managing events tracked by the SDK.

2.2 Facebook Audience Network SDK

Facebook's Audience Network SDK [21] is for advertising and monetization, providing tools for integrating Facebook ads into apps. We compiled privacy-related settings for this SDK from the official documentation's section on best practices [22, 23].

DataProcessingOptions. Allows developers to specify how Facebook should handle user data. This setting is also known as Limited Data Use (LDU). By default, data usage is not limited, but developers can change this setting to restrict data processing based on user location or Meta's geolocation. These restrictions support compliance with U.S. state privacy regulations [22] by limiting data use for personalized ads, sharing with third parties, and data retention duration. This setting can be modified only through code instructions.

MixedAudience. Helps developers manage apps used by both children and adults, facilitating compliance with the Children's Online Privacy Protection Act (COPPA) in the U.S. [23]. This setting is disabled by default, but when enabled, the SDK adjusts data collection practices to limit personal data collection from children and restrict data use for targeted advertising. The setting can be modified only via code instructions or the Meta Developers Platform.

3 Related Work

The incorporation of SDKs and TPLs into mobile apps can create substantial privacy and compliance risks. Previous research has identified risk stemming from the inheritance framework of Android permissions: all libraries in an app can use the permissions granted to the app [68, 70, 77]. SDKs that enable app monetization through advertising frequently collect personal data to deliver ads based on targeted audiences. Such data collection may lack transparency [27, 84] and could violate privacy regulations such as GDPR, CPPA, and COPPA, especially in apps targeting children [1].

Additionally, developers face challenges understanding the implications of SDK integration in their apps, which can lead to unintentional data leaks and further exacerbate risk [1, 84, 85]. The widespread adoption of SDKs and TPLs creates risks that many apps transmit personal data without proper user consent [13, 38].

Malicious Android libraries may even target SDKs and TPLs from other vendors in the same app to extract user data, since third-party components of apps are not isolated from each other [75].

To assess the real-world privacy impact of TPLs and SDKs in the mobile ecosystem, detecting their presence in apps and analyzing their behavior can be useful. TPL detection techniques and tools can be categorized based on their operational principles. Detection techniques based on package structure analyze the organizational hierarchy of code packages, utilizing predefined patterns to identify the presence of TPLs [55]. Class-dependency analysis evaluates the interdependencies among classes to detect modular components indicative of TPLs [53]. Control flow graph (CFG) and opcode analysis techniques trace control flow and low-level code structure, looking for library-specific patterns that facilitate identification of TPLs [78]. Signature-based detection utilizes a database of known TPL signatures, matching code segments to signatures to detect libraries [79]. Heuristic and machine learning tools leverage algorithms to recognize code patterns and adapt based on data [16].

Zhan et al. [80] reviewed tools available for identifying TPLs in Android apps. Their empirical study compared these tools based on various characteristics, including effectiveness, accuracy of version identification, resilience to code obfuscation, and ease of use. In this study, LibScout [10] outperformed other tools in terms of effectiveness and accurate TPL detection, albeit with higher processing times. Consequently, we leverage LibScout in our study to help identify TPLs.

Various prior approaches evaluate the behavior of Android apps. These approaches can generally be categorized as static and dynamic analysis techniques. Static analysis involves examining app code and other material without executing apps. FlowDroid [8], IccTA [52], and Amandroid [76] are widely used static analysis tools. Dynamic taint analysis and network traffic analysis are common dynamic analysis approaches. Dynamic taint analysis involves tracking the flow of data through an app during execution to identify app behavior and data uses [44, 69]. Network traffic analysis typically employs traffic interception tools such as Mitmproxy to obtain HTTP(S) communication, decrypt as necessary, and identify (potentially concerning) transmitted data [36, 47, 50, 66, 83].

Some prior work has sought to determine whether TPLs are responsible for particular app behavior and data practices. Hao et al. [37] dynamically instrumented APIs and inspected their call stack to assess whether a TPL is causing data leaks. Other related work relies on the same conceptual approach and uses different tools like Frida to inspect stack traces for API calls of interest [39, 66, 68]. One component of our analysis applies a similar approach to analyze the integration of SDKs in apps, including details of the SDKs' privacy-related settings.

Existing literature has linked SDK configurations with privacy leaks [27] and urged developer caution when using SDKs [81]. Surveys and interviews with developers reveal a reluctance to modify default settings for advertising SDKs [59]. Default settings can facilitate extensive data collection, potentially undermining user consent and putting privacy compliance at risk [12].

Closer to our work, Kollnig et al. [50] explored changes to TPLs' default privacy settings by Android and iOS developers. Their static analysis focused only on the inspection of the apps' Manifest files. They conclude that modifications are infrequent, risking violation of the GDPR's data minimization principle. Our research extends this inquiry with a deep examination of two prominent SDKs' settings, defaults, and options alongside developer choices and app privacy disclosures. We complement static analysis with dynamic analysis. This approach allows us to monitor for changes made outside of an app's Manifest file, including changes at runtime.

We also consider the possibility that SDK settings result in mismatches between app behavior and an app's privacy policy or label. Prior studies have documented potential discrepancies (and associated risks) between these privacy disclosures and app practices, considering diverse issues including repercussions of SDK integration [12, 30, 50, 66, 84]. These discrepancies may undermine transparency-oriented marketplace policies, user trust, and compliance with formal privacy policies [46].

4 Research Method

We combined state-of-the-art static, dynamic, and compliance analysis techniques into an analysis platform that sheds light on developer choices regarding the Facebook Android SDK and Facebook Audience Network SDK's privacy-related settings. Our static and dynamic analyses are complementary, with dynamic analysis validating and expanding on observations from static analysis. We analyzed app code, execution behavior, communications, and metadata. Our compliance analysis compares findings from the static and dynamic analyses with developers' representations in privacy labels and privacy policies. To facilitate analysis of a large volume of apps, we segmented the analysis tasks (as well as the downloading and storing of apps) into Docker modules and used RabbitMQ asynchronous queries to coordinate the modules (see Figure 1).

This analysis provides a detailed view of developer practices surrounding the Facebook SDKs' privacy-related settings, identifies risks, and yields insights into how developers approach the settings. The following sections describe our analysis platform.

4.1 Download and Storage

Our download module uses an unofficial Google Play Store API [33] to fetch apps (APKs) and metadata, such as download statistics and privacy policy URLs. Multiple "workers" operate simultaneously and independently. Each simulates a real device connection through an individual Google account. This multi-worker approach allows us to parallelize the downloading of terabytes of app data, achieving a peak download speed of six apps per minute (approximately 360MB per minute) using three workers. The module also uses Selenium to download privacy policies and extracts privacy labels from apps.

Our storage module acts as a centralized API server, storing and efficiently serving APKs, privacy policies, and privacy labels to other components of the analysis platform. This centralized storage works well, meeting the high-throughput demands of our platform.

4.2 Static Analysis

Our static analysis approach is designed to determine the presence of the Facebook SDKs in an Android app and collect evidence of how developers configure certain privacy-related settings. Illustrated in Figure 1, the static analysis pipeline is structured into two principal phases: SDK presence identification and settings analysis. This

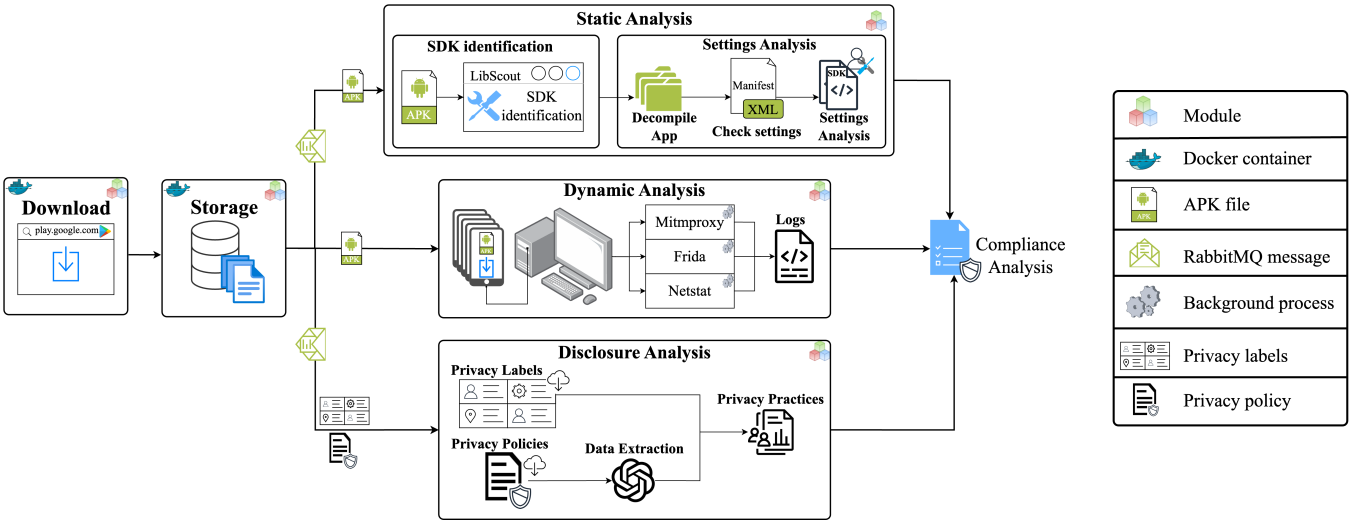


Figure 1: Architecture of our analysis platform.

component's modular design facilitates parallel execution across multiple machines.

SDK Identification. To identify SDKs in apps, our static analysis pipeline relies on LibScout [10]. LibScout uses TPL profiles to recognize SDKs in Android apps and is robust to code obfuscation like identifier renaming and API hiding [80]. LibScout constructs detailed profiles based on the class hierarchy and method signatures extracted from an SDK's compiled .jar or .aar files.

LibScout comes with default profile data for the general Facebook Android SDK, but developers may choose to include or exclude modules of this SDK (see Section 2.1). Therefore, we seek to identify the required Facebook Core SDK submodule, and we also must identify the Audience Network SDK. To address this, we developed scripts to crawl relevant SDKs available in the Maven repository and automate LibScout profile generation.[2]

To handle the raw unstructured logs that LibScout outputs, we wrote a parsing script that uses predefined text-processing rules. Our module logs each relevant SDK identified.

Settings Analysis. If the prior phase suggests an app contains a relevant SDK, the analysis focuses on evidence of privacy-related SDK settings in the app's Manifest file. We use Apktool [6] to extract and reconstruct the Manifest file from an app's compiled APK without data loss. Given the Manifest file, we analyze the XML elements and attributes to locate any assignment of relevant settings that developers can modify via this file: Facebook Android SDK's `AutoLogAppEvents`, `AutoInit`, and `AdvertiserIDCollection` settings. This analysis reveals modifications as well as instances where default settings are explicitly or implicitly unchanged.

[2]While we considered a similar approach to identify versions of SDKs, we were concerned about reliability given the sometimes-small code differences between versions. The scripts to crawl TPLs and generate LibScout profiles are available at https://github.com/DavidRodriguezTorrado/PrivacySDKSettingsAnalyzer

4.3 Dynamic Analysis

Our dynamic analysis validates, extends, and occasionally offers a different perspective from static analysis. We examine apps' runtime behavior and monitor traffic. The analysis allows us to verify the integration of Facebook SDKs and learn versions. It also reveals adjustments to privacy-related Facebook SDK settings via methods beyond an app's Manifest file alone, including code and Meta Developers Platform. In addition, we capture app communication and assess whether transmissions stem from a Facebook SDK. The combination of static and dynamic analysis provides a nuanced view of developer configuration choices and actual SDK usage.

Our dynamic analysis module utilizes five Redmi 10 devices running Android API 30 (Android 11) physically located and running in Spain. This allows parallelization and mitigates potential bottlenecks. Devices are equipped with active Frida servers for app instrumentation.

Installation and Execution. After the download module acquires an app, a RabbitMQ message initiates dynamic analysis. The app is installed on a Redmi 10 device, and background applications are halted. The app undergoes a 120-second idle phase with no user interaction followed by a 180-second interactive phase with pseudo-random events triggered via Android Monkey. Following analysis of an app, we restore the device configuration to its initial state.

SDK and Version Identification. To identify Facebook SDK integration and version, we manually inspected the code of both Facebook SDKs and identified methods (primarily getters) that reveal the SDK version and configuration values. Frida, a dynamic instrumentation toolkit, allows real-time interaction with and manipulation of processes running in user space. Frida enables monitoring and interception of all methods and calls during execution. Through injected JavaScript, we dynamically triggered the SDK-identifying methods, allowing us to capture the actual SDK version. This complements LibScout's SDK identification—ensuring the reliability of both identification methods—and reveals SDK version.

Settings Analysis. We use Frida to monitor privacy-related Facebook SDK settings.[3] Our use of Frida to infer SDK behavior and configuration changes at runtime extends our static analysis of Manifest files. All settings in Table 2 have setter methods, allowing developers to modify the settings at runtime. All of the Facebook Android SDK settings also provide getter methods for retrieving current values.

After confirming the presence of Facebook SDKs, we continuously check for SDK initialization every second to avoid triggering it prematurely by accessing the privacy settings. Once initialization is detected, we query the getters every five seconds to capture initial values and track any subsequent changes. During Frida execution, we also intercept the setters, recording both previous and new values to track configuration changes.

Network Traffic Monitoring. During both the idle and interactive execution phases, we monitor network traffic using Mitmproxy and Frida. Mitmproxy intercepts HTTP traffic from the app and decrypts encrypted (HTTPS) traffic. We use Netstat to identify open ports on the mobile device, ensuring that connections are exclusively made by the app under analysis. Our Frida scripts manipulate certificate validation to bypass certificate pinning and ensure Mitmproxy can decrypt HTTPS traffic. Frida also helps us trace the source of network communications. Our scripts intercept calls to networking methods (e.g., sockets) and log contextual information, such as the specific port used and the stack trace. This allows us to identify the code triggering the communication, including whether the code is in a third-party library. Leveraging work by Rodriguez et al. [66], we cross-reference this data with Mitmproxy logs to associate communication content with the source app and libraries. This approach enables us to determine what data is being transmitted by the applications and which SDKs are responsible for transmission.

4.4 Disclosure and Compliance Analysis

We compare evidence of apps' practices from our static and dynamic analysis against developers' declared practices in apps' privacy labels and privacy policies. This comparison can expose potential mismatches stemming from privacy-relevant Facebook SDK settings and can hint at underlying causes.

Privacy Labels. In 2022, Google implemented privacy labels [48, 49] as an accessible format for users to learn of apps' privacy practices [40]. Before the adoption of privacy labels by major app stores, privacy policies were the primary method for informing users about these practices.

Meta offers guidance regarding practices that developers should disclose on privacy labels for apps that integrate the Facebook SDKs [29]. That guidance depends on the specific SDK and the custom events configured within the SDKs. For example, developers can modify automatic event logging (via `AutoLogAppEvents`) to limit collection of user data (see Section 2.1). Both the Facebook Android SDK and the Audience Network SDK collect device identifiers; however, it is not explicitly stated how adjusting SDK settings (e.g., `AdvertisingIDCollection`) should impact privacy label disclosures.

We checked if AdID collection is declared in privacy labels and if AdID is collected and transmitted by apps. If an app integrates the Facebook Core SDK and the `AdvertisingIDCollection` setting is either unmodified or enabled, the privacy label should declare the collection of AdID. Additionally, we analyzed network traffic to verify whether the AdID is transmitted by the Facebook SDKs and cross-referenced these findings with the corresponding privacy labels to assess the alignment between actual practices and disclosures.

Privacy Policies. Privacy policies have traditionally been the primary means of informing users of Android apps' privacy practices. Google mandates that all apps have a privacy policy, which must be accessible from both the Play Store listing and the app itself [42]. We seek to assess whether app behavior aligns with practices declared in privacy policies and to identify discrepancies stemming from integrated Facebook SDKs. We also note apparent inconsistencies between privacy labels and privacy policies.

Because privacy policies are written in natural language, extraction of relevant information is challenging. To address this, we leverage an existing LLM-based privacy policy analysis tool [67]. This tool leverages ChatGPT with a carefully designed prompt that integrates advanced prompting techniques, iterative refinements, and context retention strategies to detect privacy practices. The method demonstrates high accuracy in identifying statements related to the collection of the AdID identifier, achieving an F1-score between 0.984 and 1.0 across two datasets of privacy policies annotated by legal experts. This allows for an automated approach that facilitates analysis of privacy policy disclosures at the large scale of our study and enables focused manual verification of more critical findings.

5 Results

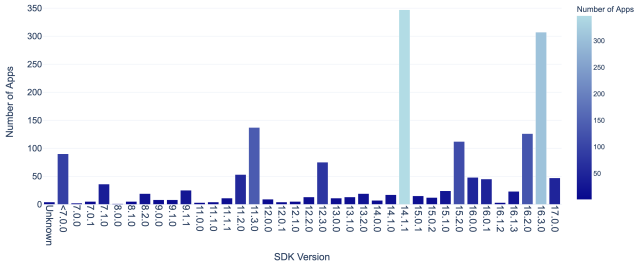
To evaluate whether and how developers modify privacy-related settings of Facebook SDKs in Android apps, we compiled a set of popular apps from AndroZoo [3], a regularly updated repository containing metadata for over four-million apps. We focused on apps with metadata collected in 2023 or later to ensure the timeliness of our analysis and the availability of apps in the Google Play Store.

We analyzed apps in order of popularity, ultimately downloading 8,848 apps to send through our analysis pipeline (see Figure 1). All analyzed apps had more than one-million downloads. During static or dynamic analysis, some apps experienced issues, ranging from LibScout processing errors to app installation and communication problems. Installation issues may stem from factors like the rooted device environment used for dynamic analysis, and execution errors when using tools like Mitmproxy or Frida also affected app analysis. We ultimately successfully processed 6,203 apps. To ensure a snapshot of practices at roughly a single point of time, we downloaded and analyzed all apps in April and May 2024.

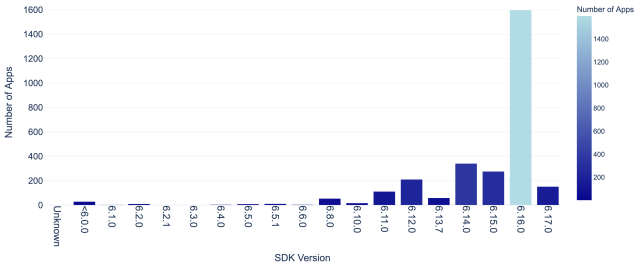
5.1 Facebook SDKs integration

We used both static and dynamic analysis to identify Facebook SDKs in apps. This choice allowed for cross-verification. LibScout and our Frida-based method agreed on the presence or absence of

[3]The Frida script used to detect Facebook SDK integration and monitor privacy-related settings is available at https://github.com/DavidRodriguezTorrado/PrivacySDKSettingsAnalyzer.



(a) Distribution of Facebook Core SDK versions. Versions prior to 7.0.0 (May 2020) are aggregated.



(b) Distribution of Audience Network SDK versions. Versions prior to 6.0.0 (September 2020) are aggregated.

Figure 2: Distribution of Facebook SDK versions.

the Facebook Core SDK and Audience Network SDK in 97.45% of the apps analyzed. As a precaution, we excluded 158 apps where the methods disagreed, leaving 6,045 apps.

The Facebook Core SDK, which manages privacy settings for the Facebook Android SDK and its components, was integrated into 1,693 apps (28.00%). The Audience Network SDK was found in 2,897 apps (47.92%). Additionally, 1,345 apps (22.25%) integrate both the Facebook Core SDK and the Audience Network SDK, while 3,245 apps (53.68%) integrate at least one of these SDKs.

We relied on our Frida-based version identification technique for version identification. While this method faced challenges due to potential changes in the location or accessibility of version-indicating methods, those challenges affected only a small number of apps (four for the Facebook Core SDK and one for the Audience Network SDK).

Figure 2 illustrates the distribution of versions of the Facebook Core SDK and the Audience Network SDK within our sample. The Facebook Core SDK exhibits a broader version distribution, with many developers retaining earlier versions. We did not observe any clear relationship between the Facebook Core SDK version and app popularity in our data. Conversely, the Audience Network SDK trends towards more recent versions being widely adopted. Our most observed version (6.16.0) matches the most deployed version according to the Google Play SDK Index [61].

Based on Google Play Store app categories, Facebook SDKs are particularly popular in gaming apps in our dataset. For instance, the Facebook Audience Network SDK is integrated into 85.7% of 'Arcade' apps, 85.5% of 'Word' apps, and 82.7% of 'Puzzle' apps. Similarly, the Facebook Core SDK is present in 83.3% of 'Casino' apps, 67.7% of 'Strategy' apps, and 66.7% of 'Role Playing' apps. Beyond gaming, categories like 'Music' (38.3% Core SDK, 72.3% Audience Network SDK) and 'Shopping' (35.5% Core SDK) also have significant integration rates. Conversely, Facebook SDK integration rates are lower for 'Medical' apps (15.4% Core SDK, 7.7% Audience Network SDK) and 'Educational' apps (7.4% for both SDKs).

5.2 Privacy-Related Settings Configuration

Our analysis indicates that some developers actively modify Facebook SDK privacy-related settings, which may suggest some level of awareness of these options. However, our observations also reveal instances where settings are configured in ways that do not result in meaningful changes, which could reflect complexities or challenges in navigating the available options and understanding their implications.

AutoLogAppEvents (Facebook Android SDK). A substantial portion of the apps (1,409, 83.23%) that integrate the Facebook Core SDK explicitly set a value for the automatic event logging option in the Manifest file. However, nearly two-thirds of these apps (939) explicitly assigned the default value (enabled) to the setting, and only one-third of the apps explicitly disabled this feature. The remaining 284 apps (16.77%) did not set any value, thus retaining the default (enabled).

In turn, our dynamic analysis revealed that 1,226 apps (72.42%) had automatic event logging enabled at runtime. In five apps, discrepancies between static and dynamic analyses were observed, likely due to adjustments made via the Meta Developers Platform, which allows for SDK configuration.[4]

Furthermore, we detected scarce runtime changes to this configuration attribute via code instructions (i.e., setter methods). We noted 32 apps enabling the setting when it was already enabled, three apps disabling it when it was already disabled, and one app disabling it when it was previously enabled. The repeated overriding of values without effecting changes could imply that developers face difficulties or ambiguities in understanding the impact of these configurations at runtime, as noted in prior studies on SDK documentation and configuration challenges [71, 72]. Section 6 further explores these issues.

AutoInit (Facebook Android SDK). The documented method to prevent the automatic initialization of the Facebook Android SDK is to disable it in the Manifest file [5], but this setting can also be altered in the app code. Our analysis revealed that 186 apps (10.99%) employed the Manifest approach to disable auto-initialization. Although this setting is enabled by default, 66 apps explicitly enabled it via the Manifest.

During dynamic analysis, we used Frida to check the SDK initialization status at one-second intervals. We found that in 194 apps, the SDK did not initialize at app startup. This suggests that eight

[4]Note that a direct comparison between the numbers obtained from static and dynamic analyses is not feasible due to differences in the coverage of each method. While static analysis includes all analyzed apps, dynamic analysis is limited to apps that can be executed successfully during the analysis process. Consequently, some apps included in the static analysis could not be dynamically analyzed, resulting in a mismatch in the datasets.

Table 3: Summary of analyzed Facebook SDKs' privacy settings. The third column is the percentage of apps that did not explicitly assign a value to the setting (default or otherwise) via the Manifest or in code. The fourth column is the percentage of apps that enabled a privacy-enhanced configuration, determined by cross-referencing changes observed in the Manifest, getters, and setters. For reasons we discuss in Section 5.2, we exclude `MixedAudience`.

SDK	Setting	No value explicitly assigned (%)	Privacy-enhanced configuration (%)
Facebook Core	AutoLogAppEvents	24.75%	17.90%
Facebook Core	AutoInit	84.64%	11.46%
Facebook Core	AdvertiserIDCollection	31.25%	6.79%
Facebook Core	LimitEventAndDataUsage	100%	0%
Audience Network	DataProcessingOptions (LDU)	99.76%	0.14%

apps employed alternative methods to delay initialization beyond the Manifest settings. Additionally, we observed three instances where the `AutoInit` configuration was changed from `False` to `True` at runtime, thereby initiating the SDK. According to the official documentation, this setting is intended to allow developers to obtain user consent before SDK initialization. However, we did not assess whether these apps implemented such consent mechanisms. Conversely, one app stopped the SDK by changing its configuration from `True` to `False` at runtime, and 38 set but did not change the previous (`True`) value.

AdvertiserIDCollection (Facebook Android SDK). A large number of apps (1,082, 63.91%) explicitly chose a value for this setting, but most (958) enabled it, which is the default configuration. Conversely, only 124 apps disabled this setting explicitly through the Manifest file.

Our dynamic analysis revealed that 1,406 apps (83.05%) had this setting enabled at runtime, while only 115 apps (6.79%) had it disabled. The remaining apps could not be executed due to errors encountered during the dynamic analysis execution. Notably, five apps appeared to modify this setting at runtime but did not alter the actual state; three remained enabled, and two remained disabled. Only two apps effectively changed the state of this setting: one enabling and one disabling it.

The discrepancy in the number of apps with this setting disabled between the static and dynamic analyses is primarily due to the fact that some apps could not be successfully executed during the dynamic analysis. However, in two specific apps, the values observed in the static and dynamic analyses did not align. This difference is likely because Frida may not have captured configuration changes that occurred very early in the app's execution. However, our multi-method approach, combining Manifest inspection with real-time monitoring via getters, mitigates this issue. By retrieving the actual runtime value of settings through getters at regular intervals, we ensure that even early programmatic changes are eventually captured, providing a comprehensive view of the app's true configuration.

LimitEventAndDataUsage (Facebook Android SDK). This setting can be modified only via code using a provided setter method. This option is disabled by default (thus not limiting usage of collected data). We did not observe any apps modifying this setting, potentially because Meta's official documentation for developers does not discuss it. Consequently, when retrieving the value of this attribute using its corresponding getter, we observed that all apps had it disabled (set to `False`).

DataProcessingOptions (Audience Network SDK). As detailed in Section 2.2, this setting allows developers to modify data processing to comply with U.S. state privacy regulations by adjusting the Limited Data Usage (LDU) option.

We discovered seven apps that explicitly disabled LDU mode (maintaining the default setting), and only four apps enabled it. Given that 2,897 apps integrate this SDK, the configuration rate for this privacy-preserving option is notably low at 0.14%. It is important to note, however, that this setting is designed specifically for compliance with U.S. regulations, and our experiment was conducted on apps available in Spain. The versions of these apps in other regions, particularly the U.S., could differ in this respect, potentially affecting the observed results.

MixedAudience (Audience Network SDK). We did not observe any apps setting this option. We did observe changes in previous tests we ran when developing our infrastructure. This option can be configured through the Meta Developer Portal and has no getter for us to monitor. Therefore, the lack of evidence does not establish that developers are not assigning a value, but only that they are not doing so via the Manifest or in code.

Summary. Our analysis shows varying levels of developer modification of Facebook SDK privacy settings. While some settings, such as `AutoLogAppEvents`, had their default values overridden in 17.90% of apps, `LimitEventAndDataUsage` was not modified in any app. As shown in Table 3, many apps retained the default configurations, with 88.54% leaving `AutoInit` unchanged. The percentage of apps opting for privacy-enhanced configurations remains low across most settings, such as `AdvertiserIDCollection`, where only 6.79% of apps disabled the default data collection setting.

5.3 Data Transfers

General Traffic Analysis. During the dynamic analysis phase, we successfully intercepted 80,449 unique connections from 4,959 apps, with 3,589 of these apps transmitting a range of user data. Our traffic inspection revealed several key trends in data transmission practices. Device model and AdID were the most frequently transferred types of personal data, with 29,784 and 17,332 transfers, respectively, suggesting a focus on advertising and device-specific

optimizations. Conversely, other data such as email addresses had significantly lower transfer frequencies.

Location data transfer, though less frequent, was notable in apps requiring location-based services. Specifically, coarse device location data appeared in 372 transfers, while precise location data was present in 264 transfers. WiFi-related data, including router identifiers such as BSSID and MAC address, was also documented, indicating that some apps collect detailed network connection information, potentially for geolocation services and network optimization.

Our analysis revealed large disparities in the percentage of apps transmitting user data across different categories. For instance, only 16.66% (54) of the apps in the Educational category transmitted user data. In contrast, 90.32% (93) of the apps in the Shopping category sent user data.

Utilizing the IPInfo service [15, 45], our dynamic analysis geolocated the IP addresses of the servers to which data was transmitted. Most of the traffic, which originated in Spain, was directed to servers located within the same country. The United States, Russia, and Singapore emerged as the second, fourth, and fifth most frequent destinations, respectively, underscoring significant cross-border data flows, particularly to non-EU countries. Data was transmitted to servers in a total of 40 countries, including geographically distant nations such as Oman, Malaysia, South Africa, Taiwan, Japan, and South Korea.

Facebook SDK Traffic Analysis. Cross-referencing Mitmproxy and Frida logs enabled us to extract stack traces from 86.46% of the connections that contained known user data, helping us to pinpoint the responsible libraries. Among these, Facebook SDKs were identified as one of the top sources of off-device personal data transmission, with 917 connections containing personal data across 518 apps, ranking second after Google's libraries and surpassing Unity3d.

Among all data types transferred by Facebook's SDKs, AdID was the most prevalent (54.03%), followed by device model (45.89%) and the WiFi router's BSSID (0.08%), as illustrated in Figure 3. Both the Facebook Core SDK and Audience Network SDK transmitted AdID and device model data, but BSSID transmission was observed exclusively in apps integrating the Facebook Core SDK. While we cannot determine exactly how Facebook uses this data, it could be employed for purposes such as profiling users, delivering personalized ads, and measuring ad performance.

Facebook's SDKs appear to transmit fewer types of data compared to other top SDKs. This difference could be due to our limitations in detecting certain data types within network connections (e.g., navigation and shopping history, which the Audience Network SDK's official documentation suggests are collected). Despite these limitations, the data that is transmitted is consistent with a business model that heavily relies on user data for advertising and analytics.

Geographic analysis of data transfers shows concentration of data flows within the EU, potentially due to GDPR regulations. The majority of the data sent by the Facebook SDKs is directed to servers located in Spain (99.58%)—the location of our test devices—and a smaller fraction goes to Portugal (0.42%). The fact that most connections remained within the country of origin suggests a localized approach to data handling.

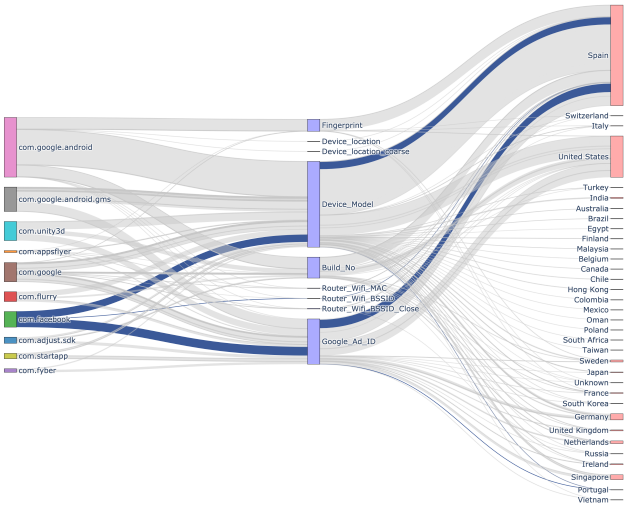


Figure 3: Personal data flows from the top ten third-party libraries to various countries. Facebook SDKs' transfers are highlighted in blue.

5.4 Disclosure Analysis

This section focuses on analyzing compliance by examining the handling of the AdID in apps that integrate the Facebook Core SDK, which regulates the automatic collection of this data through the `AdvertiserIDCollection` privacy setting. Our approach correlates three elements: the `AdvertiserIDCollection` setting in the SDK, the actual transmission of the AdID over the network observed through dynamic analysis, and the disclosures related to device identifiers found in privacy labels and policies. This analysis allows us to identify potential discrepancies and compliance issues specifically related to the collection and transmission of the AdID.

Privacy Label Analysis. We assess the congruence between the declared behaviors in privacy labels and apps' actual data collection practices regarding the AdID. The Google Play Store mandates that apps' privacy labels declare off-device data transmission, including by SDKs: "*This includes user data transmitted off device from your app by libraries and/or SDKs used in your app, irrespective of whether data is transmitted to you or a third-party server*" [40]. Facebook instructs developers to disclose the collection of "*Device or other IDs*" when integrating their SDKs [29].

For apps integrating the Facebook Core SDK, we collected developer choices for the `AdvertiserIDCollection` setting in the app's Manifest file. We also examined evidence of the setting's value from our dynamic analysis. Given those details, we scrutinized privacy labels to identify discrepancies regarding AdID collection practices and to draw insights.

For analytical clarity, we define three distinct sets within our study: L, S, and D, corresponding to labels in Google Play Store, Static analysis, and Dynamic analysis, respectively:

- L comprises privacy labels l_a for each app a, where $l_a = 1$ signifies a label indicating AdID collection, and $l_a = 0$ otherwise. Thus, $L(a)$ represents the label of app a.

- S is the set of apps assessed via static analysis, with a function $M : S \rightarrow \{-1, 0, 1\}$ mapping each app s based on the Manifest's AdID setting: enabled (1), unchanged (0), or disabled (-1). Hence, $M(a)$ represents the AdID setting value of app a in the Manifest.
- D represents apps evaluated through dynamic analysis, with a function $C : D \rightarrow \{0, 1\}$ determining the operational status of AdID collection: enabled (1) or disabled (0). Therefore, $C(a)$ represents the runtime value of AdID collection setting for app a.

Each app a belongs to the set $A = L \cap S \cap D$, as comprising apps that successfully underwent static and dynamic analysis and where the status of the `AdvertiserIDCollection` setting for the Facebook Core SDK was ascertainable.

We define $f_{\text{UCI}}(L(a), M(a), C(a))$ and $f_{\text{ACI}}(L(a), M(a), C(a))$, related to the compliance status of each app a, where $f = 0$ denotes potential non-compliance. For brevity, we omit the explicit reference to each app a in subsequent expressions, using the simplified notation $f_{\text{UCI}}(L, M, C)$ and $f_{\text{ACI}}(L, M, C)$:

(1) **Function $f_{\text{UCI}}(L, M, C)$**: This function flags instances of potentially unaware non-compliance ($f_{\text{UCI}}(L, M, C) = 0$), where a developer leaves the default AdID collection setting, which allows collection, despite not declaring this collection in the privacy label. It is defined as:

$$f_{\text{UCI}}(L, M, C) = \begin{cases} 0 & \text{if } L = 0 \wedge M = 0 \wedge C = 1 \\ 1 & \text{otherwise} \end{cases}$$

This condition may indicate a developer's lack of awareness of the Facebook Android SDK's practices and settings.

(2) **Function $f_{\text{ACI}}(L, M, C)$**: This function identifies cases of non-compliance ($f_{\text{ACI}}(L, M, C) = 0$) where we have evidence of a developer choice to enable AdID collection—explicitly choosing a setting option in the Manifest file—without disclosure in the privacy label. It is defined as:

$$f_{\text{ACI}}(L, M, C) = \begin{cases} 0 & \text{if } L = 0 \wedge M = 1 \wedge C = 1 \\ 1 & \text{otherwise} \end{cases}$$

The proportions of apps exhibiting potentially unaware and aware non-compliance are calculated as:

(1) **Portion of Potentially Unaware Compliance Issues (P_{UCI})**: The fraction of apps where the developer did not explicitly change the default `AdvertiserIDCollection` setting in the Manifest that present a potential compliance issue. $N_{\text{AdID default}}$ represents the total number of apps where we observed no evidence that the setting was changed in the Manifest file or at runtime.

$$P_{\text{UCI}} = \frac{\sum_{a \in A} f_{\text{UCI}}(L, M, C)}{N_{\text{AdID default}}} \tag{1}$$

Among apps that did not alter the default data collection configuration (518 in total), 155 failed to report that AdID was collected. This accounts for 29.25% of the apps in this group (P_{UCI}), suggesting that developers are potentially not compliant, and possibly unaware of it.

(2) **Portion of Potentially Aware Compliance Issues (P_{ACI})**: The fraction of apps where developers explicitly assigned `AdvertiserIDCollection` to `True` in the Manifest that present a potential compliance issue. $N_{\text{AdID explicitly enabled}}$ refers to the total number of apps where the developers explicitly enabled collection.

$$P_{\text{ACI}} = \frac{\sum_{a \in A} f_{\text{ACI}}(L, M, C)}{N_{\text{AdID explicitly enabled}}} \tag{2}$$

For 865 apps, we observed that developers enabled collection of AdID explicitly by setting `AdvertiserIDCollection` to `True` in the app's Manifest, and this matches the setting we observed in our dynamic analysis of the app. Of these apps, 240 did not disclose this collection in their privacy labels. This equates to $P_{\text{ACI}} = 27.75\%$.

These figures highlight a substantial potential compliance issue: 399 out of 1,388 apps (28.75%) have `AdvertiserIDCollection` enabled—whether explicitly set or left as the default—but fail to disclose this in their privacy labels. When considering all 1,693 apps that integrate the Facebook Core SDK, this non-disclosure rate accounts for 23.57% of all apps. Furthermore, 9.16% of all Core SDK-integrating apps and 38.85% of potentially non-compliant apps have compliance issues that may stem from default settings.

Privacy Policy Analysis. We extended our compliance verification to privacy policies, employing the method described in Section 4.4. This method evaluates whether privacy policies disclose the collection of both device IDs and IP addresses together, as these two data types were annotated together in the ground truth dataset used for validation. In contrast, privacy labels treat device IDs and IP addresses as distinct categories, making a direct comparison between privacy policies and labels challenging. Despite these differences in how data types are categorized, our methodology effectively identifies discrepancies in compliance. Specifically, our analysis examined instances where privacy policies did not declare the collection of identifiers or IP addresses while app settings explicitly enabled AdID collection via the Manifest. Our findings reveal that 30 of the 865 apps (3.47%) that enabled the `AdvertiserIDCollection` in the Manifest did not declare this practice in their privacy policies. Moreover, through our network traffic analysis, we observed that half of these apps (15) transmitted the AdID over the network.

Additionally, we investigated the apps where developers did not attempt to configure the `AdvertiserIDCollection` setting. Out of 518 such apps, we successfully analyzed the privacy policies of 374 apps, finding that 28 apps (7.49%) failed to declare AdID collection. Across all 1,388 apps with AdID collection enabled—either by default or explicitly—we analyzed 1,037 privacy policies and identified 58 apps (5.59%) that did not disclose AdID collection. The remaining privacy policies could not be retrieved or were inaccessible via the URLs provided in the Google Play Store.

Labels and Policies Comparison. Due to the differing data types covered under privacy labels and policies, direct comparison is challenging. However, inconsistencies are clear when privacy labels declare the collection of device and other IDs which are absent from the corresponding privacy policies. Our examination identified 445 privacy policies that explicitly stated no collection of device IDs or IP addresses, yet 182 corresponding privacy labels indicated otherwise. Out of the 445 apps whose privacy policies stated no

collection of such data, we observed 120 (26.97%) apps sending the AdID over the network. These discrepancies affirm the findings of previous studies, underscoring persistent misalignments between declared privacy labels, policies [46], and app behavior.

5.5 Children's Apps Analysis

Apps that may be used by children represent a particularly sensitive subset of the app ecosystem. Frameworks such as the Pan European Game Information (PEGI) [60] system in Europe and the Entertainment Software Rating Board (ESRB) [9] in the United States provide widely adopted age-appropriateness ratings for games and apps, which are used by major platforms like the Google Play Store [35].

For apps to be listed in Google Play's family category, the "Google Play Families Policy" [41] sets comprehensive guidelines on app content and data practices, including restrictions on transmitting sensitive personal information. The policy also mandates only advertising SDKs certified under the Families Self-Certified Ads SDK Program [34]. The Audience Network SDK is not certified at the time of writing.

We consider apps that meet all of the following three criteria: 1) have a "PEGI 3" rating (least stringent), 2) are "Teacher Approved" on Google Play, and 3) have committed to the "Play Families Policy." Our evaluation of 73 apps that met these criteria did not uncover issues related to the integration of Facebook SDKs: we did not observe any evaluated apps that integrated the Audience Network SDK, transmitted the AdID or device locations, or had observed inaccuracies in privacy labels. However, we identified four apps that integrated the Facebook Core SDK. Of these, two had the settings `AutoLogAppEventsEnabled` and `AdvertiserIDCollection` enabled, one had these settings disabled, and for one app, this information could not be retrieved through dynamic analysis. While the integration of the Facebook Core SDK itself may not violate the Play Families Policy, enabling these settings could conflict with the data minimization principles mandated by COPPA and GDPR.

We conducted a broader analysis on 779 apps that declared adherence to the Play Families Policy, without considering additional criteria such as PEGI ratings or the Teacher Approved badge. This analysis revealed several potential compliance issues: three apps were found to transmit the AdID, possibly violating policy restrictions on transmitting sensitive data for children or users of unknown age. Of these, one app did not disclose this data collection in its privacy label, and the other two failed to declare it in their privacy policies. Additionally, six apps integrated the Audience Network SDK, which may not be permitted under the Play Families policy. Seventeen apps integrated the Facebook Core SDK, with 11 of these having the `AutoLogAppEventsEnabled` setting enabled and 10 having the `AdvertiserIDCollection` setting enabled.

Previous research has consistently highlighted significant COPPA compliance concerns for potentially child-directed apps. Studies found that over half of the analyzed Android apps targeting children potentially violated COPPA due to data collection practices and the lack of consent mechanisms [50, 65]. Compliance challenges were also observed within Google's family categories, including apps in the "Designed for Families" program, a precursor to the current Play Families Policy where similar privacy violations were identified [84]. A contributor to these violations was the integration of third-party SDKs, some of which were explicitly prohibited in child-directed apps due to their data handling practices [65]. Furthermore, non-compliance was often linked to developers' insufficient knowledge and misconfiguration of privacy settings within SDKs [4, 50], issues that align with the findings of our analysis.

Our findings indicate reduced integration of Facebook SDKs in both the more restrictive subset of 73 apps and the broader set of 779 apps. This suggests potentially improved privacy practices for apps matching the criteria for these groups, particularly regarding the use of third-party SDKs. However, it remains speculative whether this reduction can be attributed solely to policies such as the Play Families Policy. Other factors, including heightened scrutiny, regulatory pressures, and evolving industry standards, likely contribute to these results. Furthermore, significant possible gaps persist in compliance with COPPA and GDPR, particularly in the configuration of privacy settings.

6 Discussion

This section draws on qualitative studies of developers to explore their perspectives on third-party SDK and library integration, highlighting developer motivations, configuration practices, awareness of privacy implications, and challenges faced in managing privacy compliance. Our analysis primarily focuses on the privacy practices of mobile apps and integrated third-party libraries. An understanding of developers' perspectives and challenges provides crucial context for interpreting our findings. We discuss these perspectives and challenges before turning to mitigation approaches.

6.1 Developers' Perspectives and Challenges

SDK Selection. In an ecosystem dominated by free applications, developers often view advertising providers as a "necessary evil" essential for sustaining their business [25]. Research suggests that developers' choice of ad networks, typically integrated into apps via SDKs, is influenced by recommendations from colleagues, information obtained from forums, and trust in large organizations such as Google's AdMob [59]. Additionally, some developers explicitly report relying on major organizations under the assumption that such entities inherently comply with legal standards, further reducing their concerns about privacy risks [4]. These findings align with our observations, as nearly half (47.92%) of the top-downloaded apps integrate Facebook's Audience Network SDK, reflecting a strong preference for established ad networks.

Challenges in SDK Integration. Despite their reliance on these SDKs, developers frequently encounter significant challenges during the integration process. A key frustration stems from the fragmented and inconsistent nature of privacy-related documentation provided by SDK vendors. This documentation is often written in dense legal language, scattered across multiple sources, or presented with inconsistent terminology and formatting, making it difficult for developers to find and correctly implement essential privacy configurations [43, 72]. Our examination of Meta's documentation highlighted this challenge: privacy-related guidance was often scattered and not centralized. Critical information, such as privacy label disclosures for Facebook SDKs in Android apps, was frequently buried in sources like blog posts [29]. This decentralization creates

difficulties for developers in understanding and correctly configuring SDKs, particularly for managing privacy settings and accurately disclosing privacy practices. The *AutoLogAppEvents* setting, which affects whether apps automatically send data to Meta, illustrates this issue: we had difficulty determining the exact types of data transmitted, and uncertainty further complicates developers' efforts to provide clear and accurate privacy disclosures.

Privacy Settings in SDKs. Many developers are unaware of the privacy settings provided by third-party SDKs, leading to incorrect configurations and potential compliance issues [4]. Compounding these challenges, ad networks frequently configure their SDKs with privacy-unfriendly default settings that maximize data collection and targeted advertising [59], as supported by our examination of Facebook's SDKs. These defaults, often specified in documentation and sample code, may implicitly encourage developers, particularly those lacking in-depth privacy knowledge, to adopt configurations that expand data collection, such as personalized ads or broad data-sharing permissions [72]. Developers have expressed a reluctance to modify default settings, which could contribute to practices that may conflict with users' privacy expectations [59], especially given issues surrounding documentation clarity and accessibility.

Privacy Compliance Challenges. The complexity and opacity of third-party SDKs and libraries pose further challenges in terms of privacy compliance. Developers frequently report difficulties in understanding the full extent of data collection practices by these libraries, as their behavior is often unpredictable or insufficiently documented [11, 25, 71]. This lack of transparency complicates developers' efforts to manage privacy settings effectively and comply with legal frameworks such as GDPR, COPPA, and CCPA [4]. The knowledge gap regarding the operation of these SDKs also impacts developers' ability to disclose app behaviors accurately in privacy policies, potentially leading to non-compliance. Smaller development teams or independent developers, in particular, often lack the technical expertise or resources to manage privacy compliance effectively, relying instead on external services, legal templates, or app store guidance [4, 62]. Additionally, the delegation of privacy responsibilities to legal or specialized teams, rather than integrating privacy considerations throughout the development process, further exacerbates compliance challenges [43].

The challenges developers face in complying with privacy requirements when integrating third-party SDKs and libraries are widespread and multifaceted. Studies by Li et al. [54] and Tahaei et al. [73] show that developers perceive privacy compliance as burdensome, offering minimal personal benefit. This leads many developers to adopt a reactive approach, responding primarily to external pressures, such as operating system updates or app store policies, rather than proactively integrating privacy considerations from the outset [54]. This reactive mindset is further complicated by the need to balance functionality with stringent privacy requirements, a struggle that frequently arises when developers draft or update privacy policies [73].

Developers' Approaches to Privacy Management. A significant contributor to this reactive stance is the lack of robust tools and reliable support systems for implementing privacy-preserving measures. Developers often rely on fragmented and informal resources, which exacerbates inconsistencies in compliance efforts [43, 72]. Horstmann et al. [43] emphasize the absence of standardized procedures for verifying privacy implementations, drawing parallels to the more structured guidance found in the security domain. The call for practical, accessible guidelines is echoed by Ekambaranathan et al. [25] and Balebako et al. [11], who argue that current data protection frameworks and SDK documentation are insufficiently clear, leaving developers without the necessary support to make informed decisions.

Moreover, developers' limited engagement in technical testing of SDKs further complicates privacy compliance. Alomar et al. [4] highlight that only a small fraction of developers actively test data collection practices to ensure they align with legal standards, revealing a critical gap in proactive privacy management. This gap reflects a fragmented accountability structure, with developers potentially caught between legal obligations and inadequate tools or guidance. Collectively, these findings underscore a fragmented accountability structure and a significant need for enhanced documentation, comprehensive tools, and better-integrated support systems to empower developers.

6.2 Mitigation Approaches

The prior section discusses research that suggests developer reluctance to change settings, challenges in doing so, and broader compliance difficulties. In combination with that research, our results offer additional evidence that SDK privacy-related settings and their defaults may be contributing to real-world privacy issues. While developers are responsible for their apps, interventions by third-party SDK providers and others may reduce the likelihood of these issues.

One straightforward mitigation strategy is for SDK providers to take a privacy-by-design approach and choose more cautious default privacy-related settings. The trade-offs of this choice depend on the potential privacy harms, benefits of the different settings options to various parties, and developer appreciation of and willingness to change privacy-related settings. While the appropriate choice may depend on the circumstances, we note that arguments that developers can easily switch to more conservative privacy settings might also suggest that developers could easily switch from more conservative defaults to alternatives.

Given the numerous challenges identified in integrating and managing third-party SDKs, addressing gaps in documentation and support mechanisms could also offer meaningful benefits. SDK providers should strive not only to make information related to the privacy and data protection aspects of SDK integration easy to find but also to make critical information hard to miss. Evaluation of the efficacy of SDK provider documentation, guidance, and other support mechanisms with respect to privacy-related settings could suggest further areas for improvement.

Marketplaces also could assist. Beyond implementing privacy compliance mechanisms and checks that address SDK settings risks, marketplaces could mandate that providers consolidate SDK privacy information. The Google Play SDK Index [61] aggregates information about popular SDKs, including versions and required permissions. The index offers a link to each SDK's privacy details, but these links are sometimes missing or outdated. With privacy

manifests [17] in the App Store, Apple offers a more direct solution by mandating that SDKs and apps include detailed files outlining their privacy practices. Unlike more fragile links, Apple proposes integrating privacy manifests directly into the SDK's metadata. This approach provides a centralized and structured format, making it easier for developers to understand what data is collected and shared. The approach also increases the potential for automated compliance assessment.

While Apple's manifests in particular could potentially make privacy information more readily accessible, they do not guarantee full transparency or accuracy. These manifests rely on the SDK providers to self-report data practices. Additionally, neither Apple nor Google's approach fully prevents developers from overlooking critical privacy information, highlighting a need for enhanced warnings or guidance to ensure developers recognize and disclose necessary privacy details in their labels and policies. Future work could empirically evaluate the effectiveness of Apple and Google's approaches in fostering compliance.

7 Limitations

Construct Validity. Our analysis leverages the AndroZoo dataset, which aggregates a comprehensive collection of apps from various sources and is consistently updated over time. AndroZoo collects APK files primarily from the Google Play Store and other third-party marketplaces using automated crawlers. These APKs are selected based on their availability at the time of crawling, without specific criteria for functionality or popularity. This non-selective approach ensures a broad range of apps are archived, but it may not perfectly reflect the current distribution in the Google Play Store. To ensure relevance, we focused on apps with high download numbers, enabling us to analyze privacy practices in apps with significant user bases. While this approach might not capture the full diversity of the Play Store, it allows us to derive meaningful insights from widely used apps.

Internal Validity. Our study confronts challenges to internal validity mainly due to potential obfuscation in the apps analyzed, which could obscure SDK behaviors and impact result accuracy. To mitigate this, we utilized LibScout, known for its resilience to obfuscation. Additionally, our methods based on Frida have been validated against LibScout, demonstrating comparable performance in detecting Facebook SDKs. This consistency increases confidence in our analytical approach and the reliability of our findings.

Our Frida-based methods aim to retrieve values representing privacy-related settings, which can be altered by various means. We conducted a rigorous manual review of the Facebook Core SDK source code to ensure that the variable's value that we retrieve reflects any alterations via the Manifest, code, or the Meta Developers Platform. Although we cannot explicitly inspect changes through the Meta Developers Platform, we can infer them from misalignment between static and dynamic analyses. This gives us greater confidence in these results over simply observing settings in the Manifest file.

Code injection with Frida during dynamic analysis is initiated in spawn mode, ensuring that the app is launched with Frida attached, which helps minimize the possibility of unobserved behavior. While rare instances exist where Frida may not capture a setter at runtime, we mitigate this by cross-referencing multiple sources of information: the initial Manifest configuration, the attribute's value at startup retrieved via getters as available, and continuous retrieval of these values every five seconds during the app's execution. This multi-faceted approach allows us to maintain a comprehensive understanding of the app's privacy settings and minimizes the likelihood of undetected configuration changes.

External Validity. Our study focuses on two popular Facebook SDKs in relatively popular Android apps. Our findings may not generalize to other SDKs, apps, and mobile platforms. Nevertheless, the study yields insights into developer practices when integrating two of the most popular SDKs into apps with many downloads.

8 Conclusions

We conducted a detailed examination of privacy-related settings in Android apps that use two popular Facebook SDKs. Our dynamic analysis, which evaluates SDKs' actual runtime configuration values, exposed discrepancies between settings choices declared statically in a Manifest file and settings values in practice. Relying solely on Manifest analysis would lead to inaccurate estimates of apps modifying privacy-related settings, as key configurations like `AutoLogAppEvents` and `AdvertiserIDCollection` can also be changed through code. Furthermore, the Audience Network SDK's settings and the `LimitEventAndDataUsage` setting cannot be modified via the Manifest, meaning they would go entirely undetected without additional analysis. By accessing getters, we determined actual runtime values, and by capturing setters, we identified changes made during app execution—insights that Manifest analysis alone would miss.

Our findings indicate that developers often fail to accurately reflect SDK-related practices in privacy disclosures—potentially driven in part by default settings—leading to discrepancies between declared and actual practices. One option to address this is for SDK providers to choose more conservative default SDK settings choices. SDK providers should also ensure that documentation, guidance, and tools effectively help developers configure privacy-related settings appropriately. App marketplaces can aggregate SDK privacy details and guidelines, ensuring easier access to accurate information. Additionally, marketplaces could implement privacy compliance mechanisms and checks that address SDK settings risks.

Future work could extend this study by including a broader selection of Android apps and exploring additional popular SDKs. The current study focuses on two Facebook SDKs, and analyzing other SDKs would provide insights not only into the privacy practices and behaviors of the SDKs but also into how developers configure and manage the privacy settings of these SDKs. Expanding to a more diverse set of apps would allow us to explore how different types of apps integrate and utilize SDKs, offering a more comprehensive view of app developer privacy management practices.

Acknowledgments

The work of Jose M. Del Alamo was partially supported by the CEDAR project, funded by the Horizon Europe research program (2021-2027) under grant agreement no. 101135577, and the work of David Rodriguez was partially supported by the PRESECREL project, funded by the Plan Estatal de Investigación Científica y

Técnica y de Innovación 2017-2020 (Ministerio de Ciencia e Investigación (Spain) - MCIN/AEI/10.13039/501100011033) under Grant agreement PID2021124502OB-C43. This research has also been partially supported by the National Science Foundation under its Secure and Trustworthy Computing (SaTC) Program (grant CNS-1914486). The authors would like to thank U.S. Federal Trade Commission staff for feedback regarding this research.

References

[1] A. Akash, S. Chithra, P. Vasuki, T. Shanmughapriya, and N. M. MG. 2022. Towards Privacy for Android Mobile Applications. In *2022 International Conference on Futuristic Technologies (INCOFT)*. IEEE, 1–8.

[2] Ar Akash, S Chithra, P Vasuki, T Shanmughapriya, and Nivas Muthu MG. 2022. Towards Privacy for Android Mobile Applications. In *2022 International Conference on Futuristic Technologies (INCOFT)*. IEEE, 1–8.

[3] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 468–471. https://doi.org/10.1145/2901739.2903508

[4] Noura Alomar and Serge Egelman. 2022. Developers Say the Darnedest Things: Privacy Compliance Processes Followed by Developers of Child-Directed Apps. , 250–273 pages. https://doi.org/10.56553/popets-2022-0108

[5] Android Developers. 2024. App manifest overview. https://developer.android.com/guide/topics/manifest/manifest-intro. Accessed: 03 October 2024.

[6] Apktool. n.d.. Apktool Official Website. Retrieved May 31, 2024 from https://apktool.org/

[7] AppBrain. 2024. The list of top Ad networks for Android. https://www.appbrain.com/stats/libraries/ad-networks

[8] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, and P. McDaniel. 2014. Flowdroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps. *ACM SIGPLAN Notices* 49, 6 (2014), 259–269.

[9] Entertainment Software Association. n.d.. Entertainment Software Rating Board. Retrieved May 31, 2024 from https://www.esrb.org/

[10] M. Backes, S. Bugiel, and E. Derr. 2016. Reliable Third-Party Library Detection in Android and Its Security Applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 356–367.

[11] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason Hong, and Lorrie Faith Cranor. 2014. The Privacy and Security Behaviors of Smartphone App Developers. In *Workshop on Usable Security*. Citeseer, Internet Society, 1–10. https://doi.org/10.14722/usec.2014.23006

[12] Y. Chen, M. Zha, N. Zhang, D. Xu, Q. Zhao, X. Feng, K. Yuan, F. Suya, Y. Tian, K. Chen, X. Wang, and W. Zou. 2019. Demystifying Hidden Privacy Settings in Mobile Apps. In *2019 IEEE Symposium on Security and Privacy (SP)*. 570–586. https://doi.org/10.1109/SP.2019.00054

[13] H. Cheng, G. Hu, J. Liu, Z. Kang, C. Pan, and Z. Zhang. 2022. Detecting Third-Party Libraries for Privacy Leakage in Packed Android Applications. In *2022 China Automation Congress (CAC)*. IEEE, 5053–5058.

[14] Hichang Cho, Sungjong Roh, and Byungho Park. 2019. Of promoting networking and protecting privacy: effects of defaults and regulatory focus on social media users' preference settings. *Computers in Human Behavior* 101 (2019), 1–13.

[15] Miguel Cozar, David Rodriguez, Jose M. Del Alamo, and Danny Guaman. 2022. Reliability of IP Geolocation Services for Assessing the Compliance of International Data Transfers. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 181–185. https://doi.org/10.1109/EuroSPW55150.2022.00024

[16] H. Cui, G. Meng, Y. Li, Y. Li, Y. Zhang, J. Sun, D. Zhu, and W. Wang. 2022. LibHunter: An Unsupervised Approach for Third-Party Library Detection without Prior Knowledge. In *2022 IEEE Symposium on Computers and Communications (ISCC)*. 1–7.

[17] Apple Developer Documentation. n.d.. Adding Privacy Manifests. Retrieved May 31, 2024 from https://developer.apple.com/documentation/bundleresources/privacy_manifest_files/adding_a_privacy_manifest_to_your_app_or_third-party_sdk

[18] Meta Developer Documentation. n.d.. Audience Network SDK for Android. Retrieved May 30, 2024 from https://developers.facebook.com/docs/audience-network/setting-up/platform-setup/android/add-sdk

[19] Meta Developer Documentation. n.d.. Facebook SDK for Android. Retrieved May 30, 2024 from https://developers.facebook.com/docs/android/

[20] Meta Developer Documentation. n.d.. Meta App Events. Retrieved Retrieved May 30, 2024 from https://developers.facebook.com/docs/app-events/getting-started-app-events-android/

[21] Meta Developer Documentation. n.d.. Meta Audience Network for Android. Retrieved Retrieved May 30, 2024 from https://developers.facebook.com/docs/audience-network/setting-up/platform-setup/android/add-sdk

[22] Meta Developer Documentation. n.d.. Meta Data Processing Options for US Users. Retrieved Retrieved May 30, 2024 from https://developers.facebook.com/docs/audience-network/optimization/best-practices/data-processing-options

[23] Meta Developer Documentation. n.d.. Mixed Audience & COPPA. Retrieved Retrieved May 30, 2024 from https://developers.facebook.com/docs/audience-network/optimization/best-practices/coppa

[24] Meta Developer Documentation. n.d.. Official Documentation. Retrieved Retrieved May 30, 2024 from https://developers.facebook.com/docs/

[25] Anirudh Ekambaranathan, Jun Zhao, and Max Van Kleek. 2021. "Money makes the world go around": Identifying Barriers to Better Privacy in Children's Apps From Developers' Perspectives. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 46, 15 pages. https://doi.org/10.1145/3411764.3445599

[26] Inc. Facebook. 2024. Facebook SDK for Android - Class FacebookSdk. https://developers.facebook.com/docs/reference/android/current/class/FacebookSdk/ Accessed: 2024-09-22.

[27] Á. Feal, J. Gamba, J. Tapiador, P. Wijesekera, J. Reardon, S. Egelman, and N. Vallina-Rodriguez. 2021. Don't Accept Candy from Strangers: An Analysis of Third-Party Mobile SDKs. In *Data Protection and Privacy: Data Protection and Artificial Intelligence*. Vol. 13. 1.

[28] Meta for Developers Blog. 2017. Optimizing and Improving the Android SDK. Retrieved Retrieved May 30, 2024 from https://developers.facebook.com/blog/post/2017/09/26/android-sdk-optimization/

[29] Meta for Developers Blog. n.d.. Resources for Completing App Store Data Practice Questionnaires for Apps That Include the Facebook or Audience Network SDK. Retrieved May 31, 2024 from https://developers.facebook.com/blog/post/2022/07/18/resources-for-completing-app-store-data-practice-questionnaires-apps-facebook-or-audience-network-sdk/

[30] Jack Gardner, Yuanyuan Feng, Kayla Reiman, Zhi Lin, Akshath Jain, and Norman Sadeh. 2022. Helping Mobile Application Developers Create Accurate Privacy Labels. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 212–230. https://doi.org/10.1109/EuroSPW55150.2022.00028

[31] GDPR 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation). https://eur-lex.europa.eu/eli/reg/2016/679/oj Accessed: 2024-09-22.

[32] GitHub. n.d.. Facebook Android SDK. Retrieved Retrieved May 30, 2024 from https://github.com/facebook/facebook-android-sdk

[33] GitHub. n.d.. Google Play Unofficial Python API. Retrieved May 31, 2024 from https://github.com/marty0678/googleplay-api/

[34] Google Play Console Help. 2024. Participate in the Families Self-Certified Ads SDK Program. https://support.google.com/googleplay/android-developer/answer/12955712 Accessed: 2024-09-22.

[35] Google Play Help. 2024. Apps and Games Content Ratings on Google Play. https://support.google.com/googleplay/answer/6209544 Accessed: 2024-09-22.

[36] D. S. Guamán, D. Rodriguez, J. M. del Alamo, and J. Such. 2023. Automated GDPR Compliance Assessment for Cross-Border Personal Data Transfers in Android Applications. *Computers & Security* 130 (2023), 103262.

[37] X. Hao, D. Ma, and H. Liang. 2022. Detection and Privacy Leakage Analysis of Third-Party Libraries in Android Apps. In *International Conference on Security and Privacy in Communication Systems*. Cham: Springer Nature Switzerland, 569–587.

[38] Y. He, X. Yang, B. Hu, and W. Wang. 2019. Dynamic Privacy Leakage Analysis of Android Third-Party Libraries. *Journal of Information Security and Applications* 46 (2019), 259–270. https://doi.org/10.1016/j.jisa.2019.03.014

[39] Y. He, X. Yang, B. Hu, and W. Wang. 2019. Dynamic Privacy Leakage Analysis of Android Third-Party Libraries. *Journal of Information Security and Applications* 46 (2019), 259–270. https://doi.org/10.1016/j.jisa.2019.03.014

[40] Google Play Console Help. n.d.. Data Safety Section. Retrieved May 31, 2024 from https://support.google.com/googleplay/android-developer/answer/10787469?hl=en

[41] Google Play Console Help. n.d.. Google Play Families Policies. Retrieved May 31, 2024 from https://support.google.com/googleplay/android-developer/answer/9893335?hl=en

[42] Google Play Console Help. n.d.. User Data. Retrieved May 31, 2024 from https://support.google.com/googleplay/android-developer/answer/10144311?visit_id=638525050145726100-2464963387&rd=1

[43] Stefan Albert Horstmann, Samuel Domiks, Marco Gutfleisch, Mindy Tran, Yasemin Acar, Veelasha Moonsamy, and Alena Naiakshina. 2024. "Those things are written by lawyers, and programmers are reading that." Mapping the Communication Gap Between Software Developers and Privacy Experts. , 151–170 pages. https://doi.org/10.56553/popets-2024-0010

[44] H. Inayoshi, S. Kakei, and S. Saito. 2022. Plug and Analyze: Usable Dynamic Taint Tracker for Android Apps. In *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 24–34.

[45] IPinfo. n.d.. IPinfo Official Website. Retrieved May 31, 2024 from https://ipinfo.io/

[46] Akshath Jain, David Rodriguez, Jose M. Del Alamo, and Norman Sadeh. 2023. ATLAS: Automatically Detecting Discrepancies Between Privacy Policies and Privacy Labels. In *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 94–107. https://doi.org/10.1109/EuroSPW59978.2023.00016

[47] Q. Jia, L. Zhou, H. Li, R. Yang, S. Du, and H. Zhu. 2019. Who Leaks My Privacy: Towards Automatic and Association Detection with GDPR Compliance. In *Wireless Algorithms, Systems, and Applications: 14th International Conference, WASA 2019, Honolulu, HI, USA, June 24–26, 2019, Proceedings*, Vol. 14. Springer International Publishing, 137–148.

[48] Patrick Gage Kelley, Joanna Bresee, Lorrie Faith Cranor, and Robert W. Reeder. 2009. A "nutrition label" for privacy. In *Proceedings of the 5th Symposium on Usable Privacy and Security* (Mountain View, California, USA) *(SOUPS '09)*. Association for Computing Machinery, New York, NY, USA, Article 4, 12 pages. https://doi.org/10.1145/1572532.1572538

[49] Patrick Gage Kelley, Lorrie Faith Cranor, and Norman Sadeh. 2013. Privacy as part of the app decision-making process. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 3393–3402. https://doi.org/10.1145/2470654.2466466

[50] Konrad Kollnig, Anastasia Shuba, Reuben Binns, Max Van Kleek, and Nigel Shadbolt. 2022. Are iPhones Really Better for Privacy? A Comparative Study of iOS and Android Apps. *Proceedings on Privacy Enhancing Technologies* 2022, 2 (2022), 6–24. https://doi.org/10.2478/POPETS-2022-0033

[51] California State Legislature. 2018. California Consumer Privacy Act of 2018. https://leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?lawCode=CIV&division=3.&title=1.81.5.&part=4.&chapter=55.&article= Accessed: 2024-09-22.

[52] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, and P. McDaniel. 2015. Iccta: Detecting Inter-Component Privacy Leaks in Android Apps. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 280–291.

[53] M. Li, P. Wang, W. Wang, S. Wang, D. Wu, J. Liu, R. Xue, and W. Huo. 2020. Large-Scale Third-Party Library Detection in Android Markets. *IEEE Transactions on Software Engineering* 46, 9 (2020), 981–1003.

[54] Tianshi Li, Elizabeth Louie, Laura Dabbish, and Jason I. Hong. 2021. How Developers Talk About Personal Data and What It Means for User Privacy: A Case Study of a Developer Forum on Reddit. *Proc. ACM Hum.-Comput. Interact.* 4, CSCW3, Article 220 (jan 2021), 28 pages. https://doi.org/10.1145/3432919

[55] Z. Ma, H. Wang, Y. Guo, and X. Chen. 2016. LibRadar: Fast and Accurate Detection of Third-Party Libraries in Android Apps. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. 653–656.

[56] Inc. Meta Platforms. n.d.. Meta Business Suite. Retrieved Retrieved Oct 2, 2024 from https://business.facebook.com/

[57] Inc. Meta Platforms. n.d.. Meta Developers Platform. Retrieved Retrieved May 30, 2024 from https://developers.facebook.com/apps

[58] Inc. Meta Platforms. n.d.. Meta Events Manager. Retrieved Retrieved Oct 2, 2024 from https://www.facebook.com/events_manager2/

[59] Abraham H. Mhaidli, Yixin Zou, and Florian Schaub. 2019. "We Can't Live Without Them!" App Developers' Adoption of Ad Networks and Their Considerations of Consumer Risks. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*. USENIX Association, Santa Clara, CA, 225–244. https://www.usenix.org/conference/soups2019/presentation/mhaidli

[60] PEGI. n.d.. Pan European Game Information. Retrieved May 31, 2024 from https://pegi.info/

[61] Google Play. n.d.. Google Play SDK Index. Retrieved May 31, 2024 from https://play.google.com/sdks

[62] Maxwell Prybylo, Sara Haghighi, Sai Teja Peddinti, and Sepideh Ghanavati. 2024. Evaluating Privacy Perceptions, Experience, and Behavior of Software Development Teams. In *Twentieth Symposium on Usable Privacy and Security (SOUPS 2024)*. USENIX Association, Philadelphia, PA, 101–120. https://www.usenix.org/conference/soups2024/presentation/prybylo

[63] Jennifer Pybus and Mark Coté. 2024. Super SDKs: Tracking personal data and platform monopolies in the mobile. *Big Data & Society* 11, 1 (2024), 20539517241231270.

[64] Maven Repository. n.d.. Facebook Core SDK. Retrieved Retrieved May 30, 2024 from https://mvnrepository.com/artifact/com.facebook.android/facebook-core

[65] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, and Serge Egelman. 2018. "Won't somebody think of the children?" Examining COPPA compliance at scale. *Proceedings on Privacy Enhancing Technologies* 2018, 3 (April 2018), 63–83. https://doi.org/10.1515/popets-2018-0021

[66] D. Rodriguez, J. M. Del Alamo, C. Fernández-Aller, and N. Sadeh. 2024. Sharing is Not Always Caring: Delving Into Personal Data Transfer Compliance in Android Apps. *IEEE Access* 12 (2024), 5256–5269. https://doi.org/10.1109/ACCESS.2024.3349425

[67] David Rodriguez, Ian Yang, Jose M. Del Alamo, and Norman Sadeh. 2024. Large language models: a new approach for privacy policy analysis at scale. *Computing* 106 (Aug 2024), 3879–3903. https://doi.org/10.1007/s00607-024-01331-9

[68] C. Schindler, M. Atas, T. Strametz, J. Feiner, and R. Hofer. 2022. Privacy Leak Identification in Third-Party Android Libraries. In *2022 Seventh International Conference on Mobile and Secure Services (MobiSecServ)*. IEEE, 1–6. https://doi.org/10.1109/MobiSecServ50855.2022.9727217

[69] J. Schütte, A. Kuechler, and D. Titze. 2017. Practical Application-Level Dynamic Taint Analysis of Android Apps. In *2017 IEEE Trustcom/BigDataSE/ICESS*. 17–24.

[70] J. Seo, D. Kim, D. Cho, I. Shin, and T. Kim. 2016. FLEXDROID: Enforcing In-App Privilege Separation in Android. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, San Diego, CA, USA, 1–15. https://doi.org/10.14722/ndss.2016.23485

[71] Mohammad Tahaei, Ruba Abu-Salma, and Awais Rashid. 2023. Stuck in the Permissions With You: Developer & End-User Perspectives on App Permissions & Their Privacy Ramifications. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 168, 24 pages. https://doi.org/10.1145/3544548.3581060

[72] Mohammad Tahaei and Kami Vaniea. 2021. "Developers Are Responsible": What Ad Networks Tell Developers About Privacy. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI EA '21)*. Association for Computing Machinery, New York, NY, USA, Article 253, 11 pages. https://doi.org/10.1145/3411763.3451805

[73] Mohammad Tahaei, Kami Vaniea, and Naomi Saphra. 2020. Understanding Privacy-Related Questions on Stack Overflow. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3313831.3376768

[74] Eeva Terkki, Ashwin Rao, and Sasu Tarkoma. 2016. Investigating User Profiling and Privacy Leaks in Mobile Ad Networks. *Tiny Trans. Comput. Sci.* 4 (2016). https://api.semanticscholar.org/CorpusID:41901100

[75] J. Wang, Y. Xiao, X. Wang, Y. Nan, L. Xing, X. Liao, and Y. Zhang. 2021. Understanding Malicious Cross-Library Data Harvesting on Android. In *30th USENIX Security Symposium (USENIX Security 21)*. 4133–4150.

[76] F. Wei, S. Roy, X. Ou, and Robby. 2018. Amandroid: A Precise and General Inter-Component Data Flow Analysis Framework for Security Vetting of Android Apps. *ACM Transactions on Privacy and Security (TOPS)* 21, 3 (2018), 1–32.

[77] J. Zhan, Q. Zhou, X. Gu, Y. Wang, and Y. Niu. 2017. Splitting Third-Party Libraries' Privileges from Android Apps. In *Information Security and Privacy: 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3–5, 2017, Proceedings, Part II*. Springer International Publishing, 80–94.

[78] X. Zhan, L. Fan, S. Chen, F. Wu, T. Liu, X. Luo, and Y. Liu. 2021. ATVHunter: Reliable Version Detection of Third-Party Libraries for Vulnerability Identification in Android Applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 1695–1707.

[79] X. Zhan, L. Fan, T. Liu, S. Chen, L. Li, H. Wang, Y. Xu, X. Luo, and Y. Liu. 2020. Automated Third-Party Library Detection for Android Applications: Are We There Yet?. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 919–930.

[80] X. Zhan, T. Liu, Y. Liu, Y. Liu, L. Li, H. Wang, and X. Luo. 2021. A Systematic Assessment on Android Third-Party Library Detection Tools. *IEEE Transactions on Software Engineering* 48, 11 (2021), 4249–4273.

[81] X. Zhang, X. Wang, R. Slavin, T. Breaux, and J. Niu. 2020. How Does Misconfiguration of Analytic Services Compromise Mobile Privacy?. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1572–1583.

[82] Kaifa Zhao, Xian Zhan, Le Yu, Shiyao Zhou, Hao Zhou, Xiapu Luo, Haoyu Wang, and Yepang Liu. 2023. Demystifying Privacy Policy of Third-Party Libraries in Mobile Apps. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 1583–1595. https://doi.org/10.1109/ICSE48619.2023.00137

[83] Y. Zhou. 2021. An Automated Pipeline for Privacy Leak Analysis of Android Applications. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1048–1050. https://doi.org/10.1109/ASE51524.2021.9678875

[84] Sebastian Zimmeck, Peter Story, Daniel Smullen, Abhilasha Ravichander, Ziqi Wang, Joel Reidenberg, N. Cameron Russell, and Norman Sadeh. 2019. MAPS: Scaling Privacy Compliance Analysis to a Million Apps. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019), 66–86. https://doi.org/10.2478/popets-2019-0037

[85] S. Zimmeck, S. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, S. Wilson, N. Sadeh, S. Bellovin, and J. Reidenberg. 2017. Automated Analysis of Privacy Requirements for Mobile Apps. In *24th Network & Distributed System Security Symposium (NDSS 2017)*. 286–296. https://doi.org/10.14722/ndss.2017.23034