Unveiling the Ownership: An Empirical Study of Android App Associations via SDK IDs

Qinsheng Hou[1,2], Wenrui Diao[2,3(✉)], Chaoshun Zuo[4], Qingchuan Zhao[5],
Lingyun Ying[6], Yacong Gu[7,8], Libo Chen[1(✉)], Shanqing Guo[2,3(✉)], Haixin Duan[7,9] and Zhi Xue[1]
[1]*School of Computer Science, Shanghai Jiao Tong University*
[2]*State Key Laboratory of Cryptography and Digital Economy Security, Shandong University*
[3]*School of Cyber Science and Technology, Shandong University*, [4]*University of Electronic Science and Technology of China*
[5]*City University of Hong Kong*, [6]*QI-ANXIN Technology Research Institute*
[7]*Tsinghua University*, [8]*Tsinghua University-QI-ANXIN Group JCN*, [9]*Quancheng Lab*,
[(✉)]Corresponding authors
{houqinsheng, bob777, zxue}@sjtu.edu.cn, diaowenrui@link.cuhk.edu.hk, chaoshun.zuo@uestc.edu.cn, qizhao@cityu.edu.hk,
yinglingyun@qianxin.com, guyacong@tsinghua.edu.cn, guoshanqing@sdu.edu.cn, duanhx@tsinghua.edu.cn

***Abstract*—In the Android app ecosystem, apps are linked through various associations, such as the same ownership, corporate ecosystems, or app repackaging. Revealing these associations offers critical insights into app owners' behaviors and potential violations. However, no work systematically measures these stakeholder relationships at an ecosystem scale. Moreover, previous works are fundamentally limited by their reliance on fragile, identity-unbound metadata or easily obfuscated code structures, leading to unreliable features and restricted application scenarios.**

In this work, we conduct a large-scale empirical measurement of potential violations and app owner behaviors in the wild. To bridge the capability gap, we introduce the third-party SDK ID, a novel, developer identity-bound feature obtained directly from app files, and design ANDROLINKER, a pipeline combining static and dynamic analysis to extract features to uncover explicit and implicit app associations. By deploying ANDROLINKER across our constructed datasets, we answer three research questions: (1) ANDROLINKER robustly extracts SDK IDs with 90.32% precision and achieves 98.9% accuracy in authorship attribution, significantly outperforming existing tools. (2) At the ecosystem scale, we detected 6,452 market apps linked to known potentially unwanted apps (PUAs), driving the permanent removal of 2,515 PUAs. (3) Through an in-depth analysis of app associations, we exposed cross-market PUA groups employing dual-profit and diverse evasion strategies, and uncovered the corporate ecosystems of technology companies. These results demonstrate the effectiveness of SDK IDs and ANDROLINKER in demystifying complex associations and securing the Android ecosystem.

***Index Terms*—SDK IDs, App Associations, Authorship Attribution, Third-party SDKs, Android Ecosystem**

I. INTRODUCTION

The rise of the mobile Internet has made smartphones indispensable, with Android now dominating 75.18% of the smartphone market share [1]. However, this vast app ecosystem is not merely a collection of isolated apps, but a complex web of stakeholders. In this ecosystem, apps can be linked through various associations, like the same developer, different developers in the same group, or app repackaging. Understanding these app associations is valuable for app clone detection, malware family classification, and forensic analysis. For example, McAfee discovered a group of 38 Minecraft copycat games (ever downloaded by 35 million users) on Google Play with the Android adware "HiddenAds" [2]. All these potentially unwanted apps (PUAs) belong to the same app owner[1]. Revealing these app associations offers critical insights into app owners' behaviors and potential violations in the ecosystem.

Recent studies have explored app associations for various analyses [3]–[9], [9], [10], like app clone detection, malware analysis, and forensic analysis. Their main approaches heavily rely on app similarity comparisons or signature certificate matching, which are increasingly failing to build app associations. First, features such as code structures and resources are susceptible to various code factors, including different development models and code obfuscation. Second, the chain of trust established by certificates is severely broken, as many certificates are signed at will, and many app owners publish apps with multiple certificates. In Google Play, around 45% of app owners and 66% of apps suffer from this issue [11]. Therefore, there is an urgent need for a new reliable feature for associating apps. In addition, despite proposing various methods for analyzing app associations, no work has measured and analyzed **Potential Violations** and **App Owner Behaviors** in the Android app ecosystem using app associations.

Our work. To address this gap, we introduce a novel perspective: leveraging the third-party **SDK ID** as a core feature to uncover app associations. Third-party SDKs[2] are widely used in Android apps to simplify development tasks by providing functions like advertising, payments, and location. We noticed that app owners must embed their identification strings provided by SDK service providers into the app to use a third-party SDK, which is defined as the SDK ID in this paper. Over 90% of SDKs used by millions of Android apps are from

[1]In this paper, app owners refer to independent app developers or app development groups that include multiple developers.

[2]In this paper, the third-party SDKs are third-party commercial SDKs.

third-party providers [12], and 78.8% of apps integrate over 10 third-party SDKs [13], making SDK IDs widespread across the ecosystem. Unlike easily forgeable metadata, an SDK ID is identity-bound. To access essential third-party services, app owners need to register their SDK server platform accounts via their phone numbers or e-mails. Because SDK providers manage services and billing via SDK IDs, app owners cannot modify them at will and are unlikely to share them in the real world. Besides, SDK IDs are usually hard-coded in apps, making them extractable from apps. Since an app can embed multiple SDK IDs and the same SDK ID can appear across apps, SDK IDs can serve as a transitive feature. Thus, in addition to the **Explicit Association** built based on features, we can also explore the **Implicit Association** between apps via the SDK ID transitivity.

Driven by this insight, we conducted a large-scale measurement of potential violations and app owner behaviors in the Android app ecosystem using app associations. To achieve this and empirically demonstrate the effectiveness of SDK IDs, we developed a hybrid analysis pipeline, named ANDROLINKER, to automatically extract features and uncover explicit and implicit associations. Besides, to empirically validate SDK IDs and conduct an ecosystem-scale measurement, we constructed two distinct datasets. First, we built a rigorously verified **Ground Truth** dataset comprising 542 apps, 12,239 SDK IDs, and established app authorship relationships, serving as a reliable baseline for evaluating authorship identification capabilities. Second, to scale our analysis to the real world, we compiled a massive **Real-world Dataset** comprising 121,949 *Market Apps* from 10 diverse app markets and 31,328 *PUAs*. This large-scale dataset enables us to measure the distribution of PUAs across app markets and conduct in-depth stakeholder analysis across the ecosystem.

Through this comprehensive measurement, we aim to answer the following research questions:

- *RQ1* **Empirical Validity**: How effective is our SDK ID-based method in authorship identification compared to existing state-of-the-art tools?
- *RQ2* **Potential Violations**: What is the prevalence and distribution of PUAs hidden within the legitimate Android app ecosystem?
- *RQ3* **App Owner Behaviors**: What is the relationship of interest between apps, and how do seemingly unrelated apps belong to the same app owner?

Contributions. Our empirical study yields several valuable observations and concrete contributions to the community:

- **Novel Perspective**: To the best of our knowledge, this is the first study to define the SDK ID as a developer-bound feature and apply it to uncover app associations in a large-scale measurement analysis.
- **Empirical Validity**: By applying our pipeline to the rigorously established ground truth, our empirical results show that using SDK IDs improves authorship identification performance, achieving higher accuracy (19.6%), precision (8.5%), and recall (13.2%) than existing SOTA tools.
- **Real-world Measurement**: By applying our SDK ID-based association method to real-world app markets, we detected 6,452 market apps linked to known PUAs, with 2,515 already removed, demonstrating that SDK IDs greatly enhance PUA detection. Through in-depth analysis of app associations, we identified cross-market PUA groups employing dual-profit and diverse evasion strategies, as well as the corporate ecosystems of technology companies.

Data Availability. The Ground Truth dataset and the experiment details are available on GitHub [14].

II. BACKGROUND AND MOTIVATION

This section introduces the mechanics of third-party SDK IDs and the motivation case.

A. Third-party SDK ID

To conduct a large-scale measurement of app associations, we require a feature deeply embedded in the app supply chain and strictly bound to the developer's real-world identity. Third-party SDK IDs uniquely meet these criteria.

Currently, most Android apps integrate third-party commercial SDKs to implement essential functions such as payment, user login, push notifications, and advertising. As shown in Figure 1 (a), to utilize these services, an app owner must complete a strict registration workflow: (1) register an account on the SDK provider's platform and provide app details to obtain a unique identification string, namely the SDK ID, (2) hard-code this SDK ID into the app's code or configuration files, and (3) upload the SDK ID to the SDK server for verification during runtime initialization.

This workflow endows the SDK ID with three unique properties crucial for tracing app ownership:

- **Identity-bound**: SDK platforms require developers to provide real-world credentials for account registration. Major SDK providers (e.g., WeChat, Amap, Umeng) mandate the developer's mobile phone number, email, or linked social media accounts. In many regions (like China), these credentials are tied to real-name registration, strongly binding the SDK ID to the developer's physical identity.
- **Non-repudiable**: SDK providers manage critical services, API quotas, and financial billing exclusively through SDK IDs, and app owners cannot modify them at will. Furthermore, owners are economically disincentivized from sharing their SDK IDs with unrelated parties.
- **Widespread and Extractable**: Apps widely use third-party SDKs, making SDK IDs widespread across apps. Besides, SDK IDs are typically hard-coded in plain text within configuration files (e.g., AndroidManifest.xml) or dex files, making them reliably extractable via automated analysis.

B. Motivation Case

To demonstrate the real-world capability of SDK IDs to uncover app associations, we present the motivating case shown in Figure 1 (b). There are three apps collected from different sources: com.gongke.fanli (from VirusTotal), com.jj.maozhua (from OPPO), and com.lifan.live (from Wandoujia). These

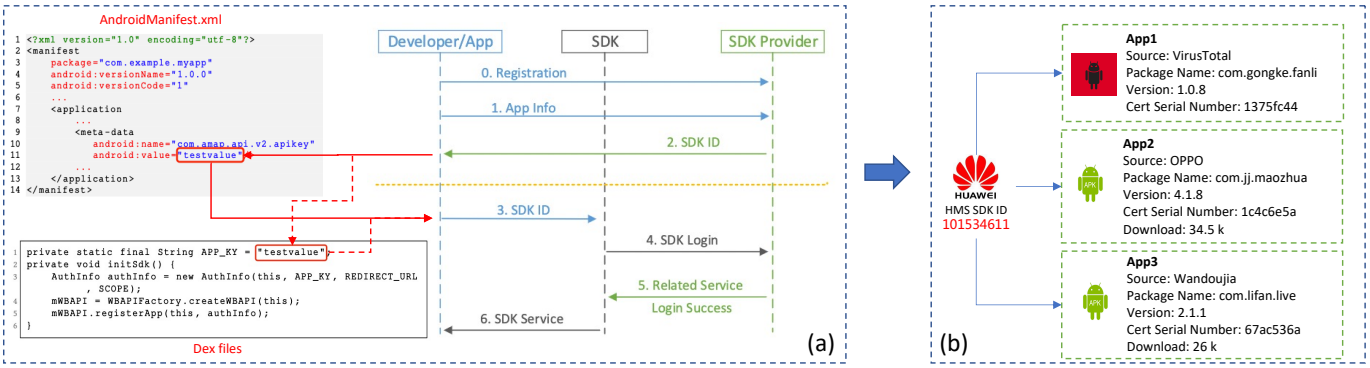

Fig. 1: **SDK ID Workflow & Motivation Case**.

apps exhibit completely different package names, app versions, and certificate serial numbers (1375fc44, 1c4c6e5a, and 67ac536a, respectively), making existing tools fail to establish any link between them.

However, deep inspection revealed that all three apps utilize the Huawei Mobile Services SDK [15] and communicate with the SDK server using the same SDK ID: 101534611. Given the non-repudiable nature of SDK IDs, this shared fingerprint strongly indicates common ownership or a shared operational group. Subsequent behavioral analysis confirmed that these apps provided identical content and shared the same user management system. We reported this finding to the respective app markets, which validated our SDK ID-based association and removed the related apps.

Observation.. The motivating case reveals the limitations of existing methods and highlights the value of using SDK IDs to associate apps. Here are our observations.

- **The inadequacy of existing methods**: Traditional association methods, which rely heavily on metadata such as package names and certificates, are increasingly ineffective at establishing reliable app associations in the current app ecosystem.
- **Evasion by PUA owners**: PUA owners can easily bypass existing association methods by routinely mutating, repackaging, or forging these features across different apps, rendering traditional methods ineffective.
- **The necessity of third-party SDKs**: Despite their evasion strategies, PUA owners, much like regular developers, heavily depend on commercial third-party SDKs for essential services and profit. To minimize operational costs and centralize management, they chose to reuse the same SDK accounts across their apps, thereby embedding the same SDK IDs.
- **SDK IDs as a robust core feature**: SDK IDs can serve as a non-repudiable app owner's fingerprint. Therefore, we can use the SDK ID as an effective new core feature to significantly enhance app associations and uncover implicit app groups.

III. Empirical Study Design

Two sequential objectives drive our empirical study. First, empirically validate the effectiveness and robustness of SDK IDs as an attribution feature compared to traditional features (RQ1). Second, leverage this validated feature to uncover potential violations and stakeholder relationships across the Android ecosystem (RQ2 & RQ3). To achieve these goals, we designed an automated methodology. This section details our dataset construction and the automated hybrid analysis pipeline (AndroLinker), which is developed to systematically extract features and uncover explicit and implicit app associations.

A. Dataset Construction

A credible empirical study relies heavily on the quality, scale, and specific purpose of its data. To fulfill our dual objectives, empirically proving the validity of SDK IDs and measuring the real-world ecosystem, we constructed two distinct datasets.

1) Ground Truth Dataset (For Empirical Validation): To establish a reliable baseline for evaluating authorship identification accuracy and demonstrating the superiority of the SDK ID-based method over existing tools (RQ1), we built a rigorously verified ground truth dataset comprising 542 apps and 12,239 SDK IDs. To minimize omissions and false positives, we adopted a cross-verification strategy. Initially, we utilized four automated tools [16]–[19] to identify embedded third-party SDKs, and manually confirmed their results to take the union. Crucially, we then conducted a rigorous manual validation process. For each identified SDK, we analyzed its official developer documentation to pinpoint the exact configuration fields or API calls used for initialization. We then reverse-engineered the related apps to extract and confirm the hard-coded SDK IDs. The authorship relationships among these apps were meticulously inferred using market data, historical certificates, and network information, and were ultimately verified through background investigations (e.g., corporate registry databases). To further ensure the validity and rigorousness of our manual verifications, we employed an independent cross-checking mechanism. Two authors independently cross-checked the extracted SDK IDs against the official

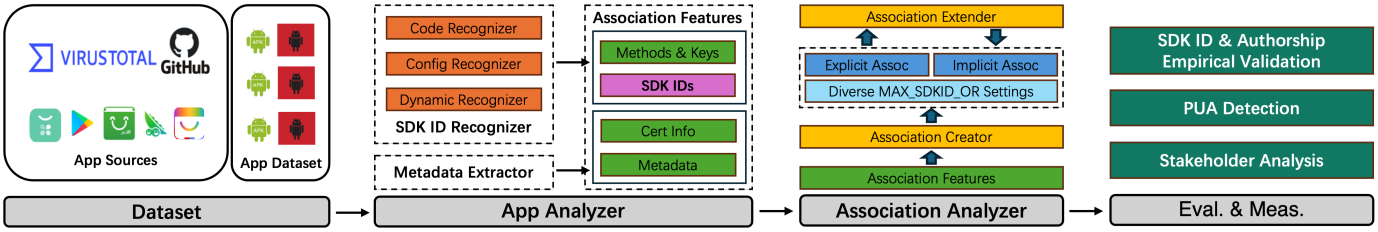

Fig. 2: **Workflow of** ANDROLINKER.

SDK developer documentation. Any discrepancies were resolved through joint discussion to reach a final consensus. An extracted SDK ID was labeled as True only if its hard-coded position, data format, and API call usage strictly matched the integration examples provided in the official SDK developer documentation. This rigorous manual verification ensures the validity of our ground truth dataset.

2) Real-world Ecosystem Dataset (For Ecosystem Measurement): To conduct a breadth analysis of the app ecosystem to uncover potential violations and do stakeholder analysis at scale (RQ2 & RQ3), we collected a massive dataset encompassing both benign market apps and PUAs.

Market Apps. To ensure our empirical findings broadly represent the Android ecosystem, we automatically collected 121,949 apps across 10 diverse app markets. This selection covers both the official Google Play and 9 third-party markets, which inherently exhibit varying levels of review strictness. Specifically, for 9 third-party markets, we downloaded apps based on their top-ranking charts: OPPO [20], Anzhi [21], Wandoujia [22], Baidu [23], Duote [24], Coolapk [25], Sogou [26], Tencent [27], and Mumayi [28]. For Google Play, to bypass its rate-limiting mechanism, we collected metadata for popular apps (e.g., package names, version numbers) from Google Play. We then downloaded related apps from AndroZoo (a public Android app dataset).

PUAs. For analyzing potentially unwanted apps (PUAs), we specifically targeted gambling and porn apps. These categories inherently rely on third-party SDKs to implement essential features, such as push notifications, user authentication, and payment processing [9], [10]. Although they are not the only PUA categories, they provide the most representative and high-risk environment for evaluating the effectiveness of our SDK ID-based associations. Therefore, this selection does not introduce a negative bias into our methodology. In contrast, it highlights the real-world capability of our approach against evasive threats. To systematically collect these apps, we first downloaded initial seed samples from known gambling and porn websites and submitted them to VirusTotal to extract their detection labels. Through manual analysis, we identified and filtered the specific labels accurately representing gambling and pornography. Using these target labels, we then downloaded a large-scale dataset of related apps directly from VirusTotal. To broaden the PUA analysis, we also collected fraudulent dating apps [29] previously identified in related work [30]. In total, we collected 31,328 target PUAs.

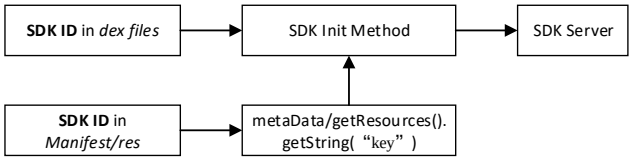

Fig. 3: **Call flow of the SDK ID in the app**.

B. ANDROLINKER

To empirically demonstrate the effectiveness of SDK IDs and process our datasets, we developed a hybrid analysis pipeline, named ANDROLINKER. As shown in Figure 2, ANDROLINKER consists of *App Analyzer* and *Association Analyzer*, which takes the apps in the dataset as input and outputs the association results.

TABLE I: Details of identity features.

Feature Name	Source
SDK ID	AndroidManifest.xml/Resource/Dex
SDK Init Method	Dex
SDK Config Key	AndroidManifest.xml/Resource
Package Name	AndroidManifest.xml
Version Number	AndroidManifest.xml
Version Code	AndroidManifest.xml
App Name	AndroidManifest.xml
Main Activity	AndroidManifest.xml
Activities	AndroidManifest.xml
Services	AndroidManifest.xml
Content Providers	AndroidManifest.xml
Broadcast Receivers	AndroidManifest.xml
Cert Serial Number	App Cert
Cert Subject	App Cert
Cert Issuer	App Cert
Cert SHA1	App Cert

1) App Analyzer: Accurate feature identification is crucial for association analysis. *App Analyzer* is designed to achieve this goal, including two submodules: **SDK-ID Recognizer** and **Metadata Extractor**. SDK-ID Recognizer aims to accurately identify SDK IDs in apps. To build complete app associations, we need to reduce false positives and false negatives of app associations. Although traditional features (e.g., package name and certificate) may miss some app associations, they are still useful as extra features to uncover as many app associations as possible. We implement Metadata Extractor to extract extra features from apps. Table I summarizes the extracted features, which work together synergistically. Among the core features, SDK IDs establish the fundamental identity-bound

```
<manifest
    ...
    <application ...>
        <meta-data
            android:name="com.amap.api.v2.apikey"
            android:value="testvalue"/>
            ...
</manifest>
```

Listing 1: Example of the AndroidManifest.xml file.

app associations, while Init Methods and Config Keys prevent rare string collisions (e.g., identical pure numeric IDs from different SDKs). Meanwhile, supplementary metadata (e.g., Certificates, App Names, and Components) are leveraged to mitigate false positives and false negatives. Specifically, they help filter out false associations by cross-verifying metadata similarities alongside certificates. Moreover, they are used to uncover app associations without third-party SDKs.

SDK-ID Recognizer. Diverse SDK ID formats and configuration locations across third-party SDKs make common rule-based identification ineffective. Besides, analyzing SDK development documents and collecting SDK ID integration methods case by case is costly and limits SDK IDs' identification range. Moreover, with thousands of hard-coded strings in an app, accurate SDK ID extraction is like finding a needle in a haystack. To address this challenge, our core idea is to identify SDK IDs based on their usage within the app.

Whether hard-coded in the app's code or embedded within configuration files (e.g., `AndroidManifest.xml`), an SDK ID must ultimately be loaded into memory and passed as a parameter to the SDK's initialization API before being transmitted to the provider's server (Figure 3). Based on this fundamental mechanism, we implemented SDK-ID Recognizer with three components: Code Recognizer, Config Recognizer, and Dynamic Recognizer.

Code Recognizer: To extract hard-coded SDK IDs, we first identify SDK initialization methods. We define *developer methods* as those that match the app's package name and the main activity prefix. Using Androguard [31], we extract candidate methods meeting five heuristics: (1) accepts at least one `String` parameter, (2) is a non-developer (third-party SDK) method, (3) is directly invoked by a developer method, (4) is unobfuscated (deferring these methods to dynamic analysis), and (5) appears in $\geq$10 apps [13], filtering out rare SDKs insufficient for uncovering app associations. Methods appearing in a few apps (e.g., 1-5) are typically private internal functions rather than shared commercial SDKs. Including them introduces massive noise (false positives) and computational overhead, lacking the cross-app transitivity essential for our analysis. Thus, a 10-app threshold achieves an optimal balance: it filters out non-transitive private methods while preserving as many third-party SDKs capable of building app associations.

To eliminate noisy strings and identify actual SDK initialization methods, we pipeline data flow analysis [32] with an LSTM model. For each candidate method, we extract its string parameters from 10 randomly sampled apps. Among these string parameters, a subset (1,400 for training, 400 for testing) is manually labeled (50% SDK IDs, 50% non-SDK IDs) using official SDK documentation to establish a training set. After training the LSTM (`dropout`=0.5, early stopping, `batch_size`=10, `epochs`=100, `test accuracy`=98.5%), we classify the remaining strings to confirm the actual SDK initialization methods. Finally, using these verified methods, we perform the targeted data flow analysis and LSTM-based classifier across the entire dataset to systematically extract all SDK IDs. The LSTM-based classifier performs a binary text classification task, which is formalized as follows:

- **Input**: Raw strings extracted from the parameters of SDK initialization methods.
- **Preprocessing**: Lightweight data cleaning on the extracted strings (e.g., filtering out non-ASCII garbled text).
- **Output**: Predicting whether the string is an SDK ID (Label#1 indicates an SDK ID, while Label#0 indicates a non-SDK ID).

Config Recognizer: To systematically extract SDK IDs embedded within configuration files (e.g., `AndroidManifest.xml` and resource files), we trace the execution paths of the verified SDK initialization methods. Using call graph analysis, we first locate standard Android string extraction APIs (e.g., `metaData.getString` and `getResources().getString`) invoked within these paths. We then apply data flow analysis to extract the hard-coded key parameters passed to these APIs. By mapping these localized keys, we retrieve the corresponding value strings directly from the configurations. Finally, to eliminate useless configuration data, we pass these retrieved values through our trained LSTM model, which semantically validates and identifies SDK IDs.

Dynamic Recognizer: To ensure measurement completeness against packers and obfuscation, which fundamentally degrade static analysis, we implemented the dynamic recognizer leveraging redroid [33] and DroidBot [34]. We first instrument the Android source code by injecting hooks into critical functions (e.g., network transmission and data storage) to comprehensively capture runtime behaviors, generated files, and cryptographic master keys. To maximize validity and app compatibility, this customized system is compiled into a dual-architecture environment: redroid9 (x86 equipped with libhoudini [35]) and redroid12 (ARM). Apps are initially deployed on redroid9, with an automated fallback to redroid12 upon execution failure. Within this sandbox, DroidBot autonomously stimulates the app's UI using a breadth-first exploration strategy for 5 minutes to trigger SDK initializations. Guided by the SDK initialization methods and configuration keys already identified during our static analysis, we extract candidate strings from the captured network traffic and runtime artifacts. Finally, our trained LSTM model semantically validates these extracted strings, successfully identifying SDK IDs and effectively overcoming static evasion techniques.

Metadata Extractor. While SDK IDs serve as our core associated feature, maximizing the discovery of app asso-

ciations requires supplementary metadata to reduce false negatives. Utilizing Androguard, we extract three dimensions of traditional app metadata. First, we parse the `AndroidManifest.xml` to retrieve foundational identifiers, including the package name, version, app name, and main activity. Second, we map the app's architectural topology by extracting all declared components (activities, services, providers, and receivers), as high structural homology strongly indicates code cloning or shared ownership. Finally, we extract certificate details, including the serial number, issuer, subject, and SHA1 fingerprint, to robustly track developer identities and signature rotations across the dataset.

2) Association Analyzer: We design *Association Analyzer* to fully uncover app associations, including **Association Creator** and **Association Extender**. Association Creator builds explicit associations using SDK IDs and extra features from App Analyzer. Association Extender uses the transitivity of SDK IDs to uncover implicit associations from explicit ones.

Association Creator. To rigorously quantify the strength of the association between any two apps, we mathematically define the Maximum SDK ID Overlap Rate (MAX_SDKID_OR). For SDK IDs extracted from code, we ensure that their initialization methods are identical. For those extracted from configurations, we strictly match the related key-value pairs. The MAX_SDKID_OR is then calculated as the number of validated same SDK IDs divided by the minimum number of SDK IDs across both apps.

Next, we conducted a rigorous pilot study to derive optimal configurations for different empirical analysis scenarios. We manually analyzed three curated ground-truth datasets (20 representative app pairs (40 apps) per dataset, including confirmed same-owner, PUA, and pirated instances) to observe the distributions of MAX_SDKID_OR values and metadata similarities. Details are available on [14]. Though bounded by manual verification, these thresholds serve as primary filters, supplemented by multi-dimensional metadata for high accuracy. Based on these empirical observations, we established tailored heuristic rulesets for target analysis scenarios.

(1) Authorship & Stakeholder Analysis: App owners frequently maintain a group of functionally diverse apps targeting different user groups, which naturally leads to varying SDK integrations. Our pilot study revealed that apps belonging to the same owner exhibit MAX_SDKID_OR values ranging from 40% to 100%. Therefore, to reliably attribute apps to the same developer or perform deep stakeholder analysis, we mandate a baseline MAX_SDKID_OR threshold of 40%. Furthermore, to reduce false negatives (i.e., missing related apps with lower SDK overlaps), we incorporate 100% certificate consistency as a supplementary heuristic, since apps with the same signature inherently belong to the same owner. Besides, we filter out piracy cases to reduce false positives.

(2) App Piracy: App piracy relies heavily on wholesale code repackaging, leading to exceptionally high MAX_SDKID_OR values (80% to 100%). However, because attackers cannot reuse the victim's signature, pirated pairs demonstrate complete certificate inconsistency. To identify app piracy, both to measure its prevalence and to serve as a vital exclusion filter for other analyses, we established a dedicated heuristic: an 80% MAX_SDKID_OR threshold, an 80% minimum similarity across metadata (app names, package names, components, and main activities), and certificate inconsistency (0%).

(3) PUA Detection: PUA owners typically batch-produce PUAs (e.g., app clones for gambling or pornography) by reusing core functionalities, resulting in a high degree of SDK ID overlap (ranging from 60% to 100%). To evade market reviews, however, PUA owners frequently change their certificates. Therefore, for PUA detection, we supplement the link with certificate consistency and enforce a stricter MAX_SDKID_OR threshold of 60%. Besides, we require a concurrent 60% similarity across essential metadata features (components, certificates, package name prefixes, app names, and main activity prefixes). We also exclude false positives caused by app piracy.

Association Extender. A unique advantage of using SDK IDs to associate apps is their transitivity. Because app-SDK ID associations can be one-to-many, SDK IDs can bridge apps that lack direct metadata links (e.g., App A and App C form an implicit association if both share an SDK ID with App B). However, due to the complexity of real-world app associations, where one app can link to dozens or hundreds of other apps, association extensions risk infinite loops and excessive resource consumption.

We implement Association Extender, a BFS-based method, to extend app associations. Taking explicit app associations as input, the method iteratively processes each app as a distinct starting node. For a given starting app, it uses a queue to track pending traversals and a visited set to strictly prevent cycles. During traversal, whenever an explicit association to a neighboring app is identified, the method records the directed link and enqueues the neighbor if it has not been visited before. Upon exhaustion of the queue, the method extracts the complete subgraph of explicit and implicit associations anchored to that specific app. The final output aggregates these complete traversal sets for every app across the dataset.

IV. Empirical Results

In this section, our empirical study addresses three core research questions: validating the effectiveness of SDK IDs (RQ1), demystifying hidden PUAs in the wild (RQ2), and uncovering stakeholder relationships (RQ3).

Environment Setup. AndroLinker runs on four servers: one Ubuntu 18.04 (256 GB RAM, 48-core Intel Xeon), one Ubuntu 20.04 (128 GB RAM, 48-core Huawei Kunpeng 920), and two CentOS 7.2 (128 GB RAM, 40-core Intel Xeon). Particularly, we set a 5-minute timeout for each app's data flow analysis, as we found that few apps can obtain normal results beyond this limit.

A. RQ1: Empirical Validity

SDK ID Extraction. Accurate feature extraction is the prerequisite for reliable ecosystem measurement. As shown in Ta-

ble II, ANDROLINKER extracted 13,240 candidate SDK IDs from the 542 ground-truth apps. Cross-referencing these with manual verifications yielded 11,959 True Positives, achieving a robust extraction precision of 90.32%. To ensure empirical transparency, we investigated extraction errors: False Positives (9.68%, 1,281 IDs) primarily stem from resource identifiers (e.g., video ad IDs like `b5e02cf71e3bcb` in the TopOn SDK [36]) that have similar structural syntax but lack developer identity binding. Besides, False Negatives (2.29%, 280 IDs) are mostly low-entropy strings (e.g., short strings like `110557` in the Meizu SDK [37]) that are conservatively filtered out by our semantic validation pipeline to suppress noise rigorously.

Authorship Attribution. Based on the accurately extracted SDK IDs, we perform app authorship identification. Considering the availability of existing works [3], [6]–[8], [38], we chose AppAuth [3] and AAA [38], the only two open-source and reproducible tools.We re-run the open-source implementations of the baseline tools on our Ground Truth dataset to ensure a strict and fair comparison.

As detailed in Table III, ANDROLINKER significantly outperforms existing tools, achieving an overall Accuracy of 98.9%, a Precision of 99.6%, and a Recall of 99.2%. To strictly isolate the efficacy of our new feature, we also evaluated SDK IDs independently (without supplementary metadata). Remarkably, relying solely on SDK IDs achieved a 100% Precision, 97.2% Recall, and 97.4% Accuracy. This independent feature alone outperforms the best existing tool (AAA, 79.3% accuracy). Furthermore, as AppAuth and AAA rely on training with explicitly associated apps and related code structural features, they completely fail to handle unassociated apps (0 True Negatives). In contrast, ANDROLINKER correctly identifies 95.2% (40/42) of True Negatives and yields 20 times fewer False Positives than AAA.

The Superiority of SDK IDs. The empirical results validate that identity-bound SDK IDs successfully bridge the massive capability gaps left by traditional metadata. While existing tools (e.g., AppAuth and AAA) are good at attributing highly homologous codebases, they fail when faced with enterprise-level multi-architecture development paradigms or aggressive obfuscation. For instance, ANDROLINKER successfully associated `com.orangewealth.orangeclient` (a financial app) and `com.chinatelecom.bestpayclient` (a payment app). Despite possessing different code structures and certificates, both apps shared identical SDK IDs (e.g., Vivo, Xiaomi, and Meizu SDKs), confirming their common ownership under China Telecom. By penetrating code obfuscation and structural diversity, ANDROLINKER establishes highly reliable empirical links that traditional attribution tools fundamentally miss.

Answer: ANDROLINKER's app associations effectively identify app authorship, outperforming AppAuth and AAA with higher accuracy (19.6%), precision (8.5%), and recall (13.2%). SDK ID is the key to this improvement.

TABLE II: Statistics of SDK ID extraction evaluation.

Name	Number	Percentage
Ground Truth	12,239	-
ANDROLINKER	13,240	-
True Positive	11,959	90.32%
False Positive	1,281	9.68%
False Negative	280	2.29%

TABLE III: Statistics of authorship identification comparison.

Tool Name	TP	TN	FP	FN	Precision	Recall	Accuracy
ANDROLINKER	496	40	2	4	99.6%	99.2%	98.9%
SDK ID	486	42	0	14	100%	97.2%	97.4%
AppAuth	413	0	42	87	90.8%	82.6%	76.2%
AAA	430	0	42	70	91.1%	86.0%	79.3%

B. RQ2: Potential Violations

To answer RQ2, we deployed ANDROLINKER across our real-world dataset to systematically measure the prevalence and distribution of potentially unwanted apps (PUAs) hidden within legitimate app markets. PUA owners typically batch-produce PUAs (e.g., illegal gambling and porn apps) using shared templates, deliberately rotating app metadata (like package names and certificates) to evade market review mechanisms. However, because they rely on third-party SDKs for core functionalities (e.g., payment, authentication, and push notifications) [9], their identity-bound SDK IDs can be used to build associations.

PUA Discovery and Ecosystem Distribution. By executing our association method (Section III-B2), ANDROLINKER successfully linked 6,452 seemingly benign market apps to 2,732 known PUAs, shown in Table IV. This indicates that 5.29% of the analyzed market ecosystem is connected to underground PUA groups. Crucially, our analysis reveals the absolute dominance of our proposed feature: the number of PUAs detected exclusively via identity-bound SDK IDs (4,071 apps, accounting for 63.10% of the associated set) was nearly twice that identified solely by traditional supplementary features (2,381 apps, 36.90%). This stark contrast empirically highlights the systemic failure of existing market vetting mechanisms that over-rely on easily forgeable metadata (e.g., package names or certificates), effectively validating the superiority of SDK IDs in real-world PUA detection.

Furthermore, our ecosystem-wide measurement exposes significant differences in market governance and reviewing stringency. Among the 9 third-party markets, Duote exhibited the most severe PUA infiltration with an association rate of 9.56%, suggesting highly permissive pre-release reviewing. Conversely, Tencent maintained the most secure ecosystem, exhibiting a minimal association rate of 1.14%. Interestingly, international platforms like Google Play present a divergent regulatory paradigm. Although ANDROLINKER successfully linked 1,851 apps to known PUA groups on Google Play, none were removed. This aligns with their specific policy boundaries, which conditionally permit certain gambling [39] or adult content [40], illustrating how PUA groups meticulously

TABLE IV: Statistics of PUA detection.

TN: the total app number, **TAN**: the total associated app number, **TAP**: TAN/TN, **SAN**: the SDK ID associated app number, **SAP**: SAN/TAN, **EAN**: the extra features associated app number, **EAP**: EAN/TAN, **RA**: the removed app number.

Markets	TN	TAN	TAP(%)	SAN	SAP(%)	EAN	EAP(%)	RA
OPPO	36,592	2,573	7.03%	1,682	65.37%	891	34.63%	2,201
Google Play	30,095	1,851	6.15%	1,283	69.31%	568	30.69%	*
Anzhi	13,950	312	2.24%	121	38.78%	191	61.22%	47
Wandoujia	12,623	298	2.36%	91	30.54%	207	69.46%	43
Baidu	9,271	289	3.12%	139	48.10%	150	51.90%	72
Duote	9,263	886	9.56%	637	71.90%	249	28.10%	91
Coolapk	4,645	161	3.47%	83	51.55%	78	48.45%	25
Sogou	2,800	39	1.39%	21	53.85%	18	46.15%	8
Tencent	2,378	27	1.14%	7	25.93%	20	74.07%	22
Mumayi	332	16	4.82%	7	43.75%	9	56.25%	6
MarketTotal	121,949	6,452	5.29%	4,071	63.10%	2,381	36.90%	2,515
PUAs	31,328	2,732	8.72%	1,784	65.30%	948	34.70%	-
Total	153,277	9,184	5.99%	5,855	63.75%	3,329	36.25%	2,515

Remarks: ∗ Google Play allows gambling apps and conditionally allows porn apps.

TABLE V: Example of PUA detection.

PackageName	Owner	Source	Downloads	State
com.shatang.fjyl	shatang	VirusTotal	-	-
com.hzsj.dsjy	youyuan.com	OPPO	9M	Removed
com.meigui.mgxq	unisoftwork.com	OPPO	98K	Removed
com.youyuan.yhb	youyuan.com	OPPO	2M	Monitoring
com.tongde.qla	rfydong.com	OPPO	718K	Monitoring
com.hzsj.dsjy	rfydong.com	Tencent	17M	Removed
com.youyuan.yhb	youyuan.com	Tencent	4M	Removed
com.cinnabar.fjlxjy	cinnabar.cn	Wandoujia	3K	Investigating

exploit regional policy nuances to maximize their distribution networks.

PUA Evasion: A Cross-Market Case Study. To illustrate how PUA groups manipulate developer identities to evade market review, we present the case of *com.shatang.fjyl*, an illegal porn app identified via VirusTotal. ANDROLINKER successfully linked this seed PUA to 7 market apps in multiple app markets, shown in Table V. Obviously, these market apps were released with different certificates and claimed by four distinct owner entities (e.g., *youyuan.com*, *unisoftwork.com*, *rfydong.com*, and *cinnabar.cn*), causing traditional authorship tools to fail to associate them. However, guided by the shared SDK IDs, our subsequent behavioral analysis confirmed that all these apps provided identical illicit content and shared a unified user management system (Figure 4).

To decisively validate these PUA links, we collected comprehensive forensic evidence, including shared SDK IDs and behavioral screenshots, and submitted responsible disclosures to the affected markets (OPPO, Tencent, and Wandoujia). The real-world impact was substantial. Four highly popular apps were permanently removed: *com.hzsj.dsjy* (9 million downloads) and *com.meigui.mgxq* (98,000 downloads) from OPPO, alongside a variant of *com.hzsj.dsjy* (17 million downloads) and *com.youyuan.yhb* (4 million downloads) from Tencent. Furthermore, OPPO officially placed *com.youyuan.yhb* and *com.tongde.qla* under strict surveillance, confirming they will "remove the apps from the app market if regulation violations are found". Meanwhile, *com.cinnabar.fjlxjy* is currently under active investigation by Wandoujia. The definitive punitive actions and verifiable feedback from these major markets

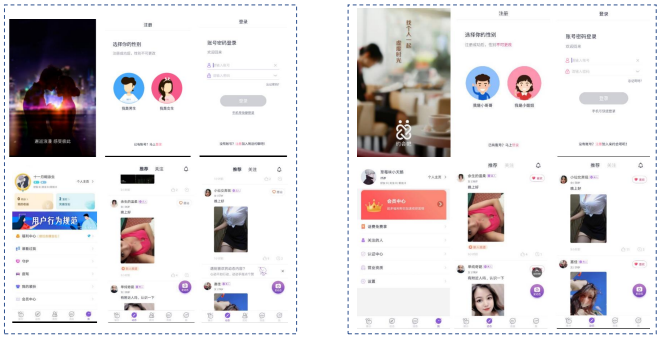

Fig. 4: **Linked app content**. Left: com.shatang.fjyl (VirusTotal). Right: com.youyuan.yhb (app markets).

validate ANDROLINKER's effectiveness in detecting cross-market camouflaged PUAs.

Real-world Impact. Beyond academic measurement, we actively translated our empirical findings into ecosystem remediation. We reported our comprehensive forensic evidence (e.g., shared SDK IDs and app behavior screenshots) to the official reporting channels of the affected markets. Following their investigations, the markets validated our SDK ID-based associations and permanently removed 2,515 violating apps.

This remediation impact was particularly pronounced within OPPO, which hosts the largest share of our dataset and had previously faced intense public media criticism regarding the persistence of camouflaged PUAs (e.g., fraudulent payment behaviors and porn content) [41]. Upon receiving our reports, OPPO initiated a massive takedown of 2,201 violating apps. More importantly, this disclosure catalyzed systemic policy improvements at the vendor level: the OPPO Security Response Center (OSRC) officially updated its *Security Intelligence Scoring Standards* [42] to incorporate such app-link vulnerabilities and awarded us a $1,000 bug bounty. This industrial endorsement unequivocally underscores ANDROLINKER's capability to uncover hidden PUAs in app markets and fundamentally augment the security posture of the Android ecosystem.

False Positive Mitigation. During our large-scale PUA detection, we initially identified 911 false positive associations. Our root cause analysis revealed that App Development Frameworks caused these associations. These frameworks routinely embed preset, boilerplate SDK ID examples into their generated app templates. Because developers frequently fail to remove these unused SDKs, numerous unrelated apps inadvertently share identical SDK IDs, leading to incorrect associations. Fortunately, these framework-generated apps exhibit highly rigid and predictable architectural signatures, such as generic package naming conventions and default main activities. By establishing explicit exclusion rules to match and filter these structural features, ANDROLINKER successfully eliminated this systemic noise, ensuring the high fidelity of our final PUA association results.

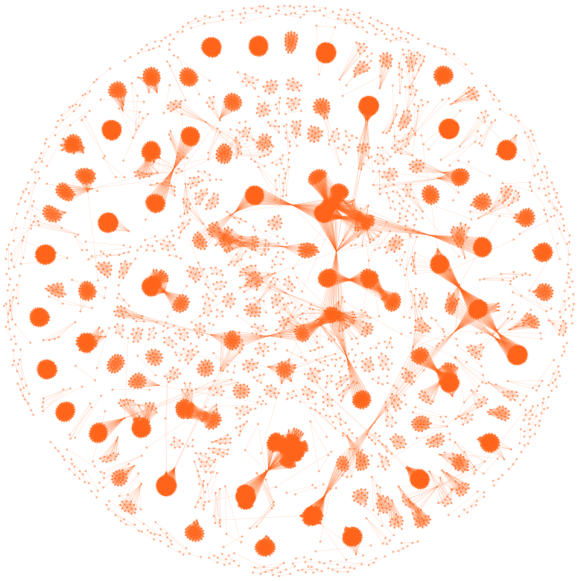

Fig. 5: **Visualization of association groups**. The nodes represent associated apps. The edges represent the explicit associations between apps.

Uncovering App Piracy Groups. Beyond traditional PUA detection, our association pipeline exposed large-scale app piracy and repackaging. Because pirates wholesale clone legitimate apps to mislead users, these pairs inherently exhibit an exceptionally high SDK ID overlap alongside complete certificate inconsistency. Through AndroLinker, we detected targeted intellectual property theft against major apps, including a repackaged variant of *Kugou*, two fake *Baidu* apps, and 17 parasitic clones of *WiFi Master*. More alarmingly, our ecosystem-scale measurement uncovered specialized underground groups dedicated solely to copyright infringement. Specifically, we identified "Team xiaokang," which mass-produced 47 piracy apps affecting 20 distinct victim developers, and "Team np," responsible for 28 piracy apps impacting 21 victims. These findings strongly demonstrate AndroLinker's capability to systematically expose broader copyright violations.

Answer: Our measurement shows that AndroLinker effectively detects PUAs in app markets, identifying 6,452 market apps linked to known PUAs, with 2,515 removed. The SDK ID significantly enhances PUA detection.

C. RQ3: App Owner Behaviors

To systematically answer RQ3, we shift our focus to the broader ecosystem, aiming to uncover the interest relationships and stakeholder networks among real-world apps. In the real-world ecosystem, app owners frequently manage portfolios of functionally distinct apps targeting diverse user groups. Developers often employ varying certificates, code structures, and market accounts naturally due to functional divergence, brand segmentation, and corporate asset management strategies. However, beneath this fragmented surface metadata, app owners inherently reuse identical SDK IDs to access third-party services across their diverse apps. This practice fundamentally streamlines their operational infrastructure, unifies data asset tracking, and significantly reduces maintenance costs. By systematically tracking these identity-bound features, AndroLinker successfully groups these seemingly isolated apps to reveal their underlying stakeholder relationships.

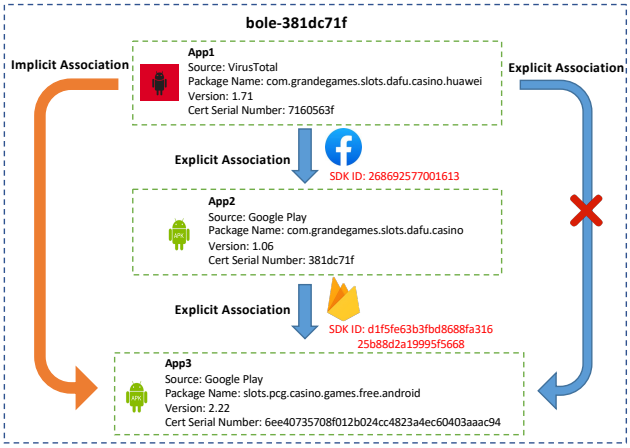

Fig. 6: **Example of app implicit association**.

Global Ecosystem View and Association Transitivity. Processing explicit associations with AndroLinker's *Association Extender* yields isolated association sets, where all apps in a set belong to a unified owner group. We label each group using the most frequently occurring app certificate Subject and Serial Number within that group. Figure 5 provides a visual overview of these macro-level association groups. Within this topology, nodes represent individual apps, and edges denote explicit associations. Crucially, the graph reveals that some expansive groups comprise multiple dense app clusters bridged by specific intermediate nodes. The presence of these connecting nodes empirically demonstrates the transitivity of SDK ID-based associations, enabling the discovery of implicit app associations that lack directly associated features.

Unveiling Implicit Links: A Case Study. To illustrate this transitivity mechanism, Figure 6 details a specific implicit group labeled `bole-381dc71f`. Within this group, a PUA identified via VirusTotal (App1) shares an identical Facebook SDK ID with a benign Google Play app (App2). While App1 has no direct explicit link to another Google Play app (App3), App2 and App3 are connected via a shared Firebase SDK ID. Leveraging this chain of explicit associations, AndroLinker establishes an implicit association between App1 and App3, effectively revealing a common underlying stakeholder.

In-depth Discovery I: The PUA Dual-Profit Model. By analyzing these comprehensive association graphs, we uncovered sophisticated operational strategies, most notably within 555 hybrid groups containing both known PUAs and market apps. A prime example is the `psd-64c4bbee` group, including 8 dating apps (one illegal porn PUA and 7 market apps), shown in

TABLE VI: Example of PUA group

Assoc SDK ID refers to the number of SDK IDs that can create app associations.

PackageName	Version	Source	Downloads	Assoc SDK ID
com.psd	6.6.5	VirusTotal	-	-
com.psd	6.1.1	OPPO	2M	6
com.psd.live	6.6.0	OPPO	1M	3
com.duixiangqin	1.1.7	OPPO	211K	3
com.psd	6.7.5	Duote	*	6
com.psd	6.8.3	Duote	*	5
com.psd.live	6.7.1	Duote	*	1
com.psd.panliao	6.4.0	Duote	*	4

Remarks: $*$ Duote does not show the number of app downloads.

TABLE VII: Example of implicit group

Owner	App Num	Cert Num
Douyin	98	2
Banciyuan	8	1
Duoshan	3	1
TikTok	2	1

Table VI. Deep inspection of the OPPO market revealed three connected apps (3.2 million downloads in total): `com.psd`, `com.psd.live`, and `com.duixiangqin`. While the first two share a certificate and market account (with `com.psd.live` serving as an alternate version), `com.duixiangqin` employs a different certificate and developer account to evade market review. Besides, the apps from Duote expose a further evasion strategy: two `com.psd` variants share identical package names and certificates but were uploaded via different market accounts. The above strategies reveal a deliberate "dual-profit" model: PUA owners profit from illegal underground channels while deploying camouflaged, legally structured variants to launder their presence and secure legitimate revenue streams in mainstream app markets.

In-depth Discovery II: Implicit Corporate Matrices. Beyond PUA groups, our measurement also penetrates the complex corporate shells of legitimate technology companies. ANDROLINKER uncovered an implicit group belonging to ByteDance [43], `ByteDance-12a97df`, comprising 111 apps signed by 5 different certificates and registered under 4 seemingly independent developer entities, shown in Table VII. Specifically, `Douyin [44]` (managing 98 apps) and `TikTok [45]` (2 apps) are managed under separate certificates. Similarly, `Duoshan [46]` was incubated internally as a distinct social app in 2019, whereas the ACG community app `Banciyuan [47]` was acquired externally in 2017. Despite their disparate development origins and different market positioning, ANDROLINKER successfully clustered all 111 apps into a single unified group. This case study demonstrates that SDK IDs can serve as digital fingerprints, capable of mapping the implicit corporate matrices of technology companies.

Answer: ANDROLINKER uncovers app associations to mine stakeholder relationships. Our in-depth analysis revealed the PUA dual-profit model and implicit corporate matrices.

V. DISCUSSION

In this section, we discuss threats to validity, ethical issues, and responsible disclosures.

A. Threats to Validity

SDK ID Evasion. A theoretical limitation of our approach is that app owners could deliberately register distinct SDK IDs for every published app to break associations. However, the economic and operational costs make this largely difficult in practice. Major SDK providers mandate strict registration workflows, often requiring real-name verification, linked phone numbers, or social media accounts (especially under strict data regulations in regions like China). Consequently, acquiring massive batches of legitimate SDK IDs is extremely expensive for small groups without resorting to illegal data trading. For larger groups, managing massive isolated SDK IDs dramatically increases maintenance overhead. Thus, large-scale SDK ID reuse remains an economically unavoidable and empirically prevalent situation for app owners.

Static Thresholds. A potential threat is our reliance on static thresholds (e.g., 40%, 60%, and 80% MAX_SDKID_OR) derived from our pilot study. Rigid thresholds might misclassify borderline cases. To mitigate this, ANDROLINKER systematically supplements SDK ID overlaps with multi-dimensional metadata features (e.g., certificate consistency and configuration similarity) to minimize false negatives. Future work could explore machine-learning models for dynamic thresholding to enhance adaptability.

Technical Evasion. Adversaries might attempt to deploy dynamic payloads or encrypt SDK IDs in the code. While such strategies degrade static analysis, they introduce significant development complexity and runtime instability (e.g., app crashes). Crucially, regardless of how heavily an SDK ID is obfuscated against static analysis, it must eventually be decrypted in memory and transmitted to the provider's server during runtime initialization. ANDROLINKER analyzes packed or obfuscated apps using the dynamic recognizer, which successfully captures their SDK IDs in network traffic or runtime artifacts. Although attackers can theoretically bypass any feature-based detection [48], the cost of bypassing SDK IDs would be enormous, making our approach effective for broad ecosystem measurement.

Analysis Boundaries. For the minority of apps without SDKs, ANDROLINKER falls back on traditional metadata (e.g., certificates) to establish explicit links, though this sacrifices the capability to uncover implicit associations. Furthermore, to ensure the scalability of our measurement across 150,000+ apps, we empirically enforced a 5-minute timeout for data flow analysis. Consequently, less than 1% of apps failed to extract code-based SDK IDs within this time limit. We mitigate this by extracting SDK IDs from these apps' configurations.

B. Ethics and Responsible Disclosure

Ethics. All our datasets were legally collected from public sources. PUA data came from VirusTotal and public databases.

Google Play app information was collected from Google Play, with files from Androzoo. For other market apps, we obtained the app files from their public channels.

Responsible Disclosure. To translate our empirical findings into real security improvements, we practiced responsible disclosure. We actively shared our findings, comprising app associations and behavioral evidence, with the official reporting channels of the affected market. All contacted markets formally acknowledged and confirmed our findings, leading to the large-scale removal of the reported PUAs.

VI. Related Work

We discuss our empirical study within two primary research areas: app authorship analysis and threat measurement.

App Authorship Analysis. Many studies have explored app authorship attribution and clone detection. Previous methodologies heavily rely on code features and structural homology. Wang et al. [6] proposed A^{3}ident, an approach that attributes an app's core code to its developer via authorship decoupling and identification. Xu et al. [3] introduced AppAuth, a learning-based approach to identify authorship in Android app clones. Aydogan et al. [38] developed AAA to leverage source code-based features to attribute both benign and malicious apps. Besides, other approaches utilize data structures and opcode usage [7], hard-coded string analysis [8], dependency graph centroids [4], code similarity analysis [49], fuzzy hashing [5], and resource file comparisons [50].

These existing methods are fundamentally limited because they rely on **identity-unbound features**. Code structures and style features are highly susceptible to changes in development paradigms, third-party SDK integration, and code obfuscation. More importantly, they completely fail when a single developer manages functionally diverse apps with entirely different codebases. Gray et al. [51] emphasize that traditional malware authorship attribution is inadequate in complex threat environments because its reliance on static features makes it highly vulnerable to adversarial obfuscation, and its single-author assumption fails to account for modern, multi-developer agile collaboration. In contrast, we introduce the SDK ID as a novel, unforgeable **identity-bound feature**. By binding app associations to the developer's real-world service registration, AndroLinker overcomes code structural diversity and obfuscation, effectively expanding the application scenario from clone detection to authorship identification, PUA detection, and stakeholder analysis.

Threat Measurement. Recent studies have increasingly used app associations to investigate underground app ecosystems. Gao et al. [9] built expansion methods to identify suspicious services in gambling networks, while Hong et al. [10] and Hu et al. [30] conducted empirical studies on mobile gambling scams and fraudulent dating apps. Furthermore, Guo et al. [52] explored underground app distribution via Telegram channels. To track app developer identities across markets, Sebastián et al. [53] proposed Retriever to extract indicators of compromise (IOCs) from app market web pages.

A critical bottleneck in these works is their reliance on either manual association efforts, flawed existing tools, or unstable data collection mechanisms. For example, Retriever's generality is severely limited by diverse market page structures and evolving anti-crawling defenses, leading to high maintenance costs. When relying solely on app files, it falls back on certificate SHA1 hashes, which have been proven highly unreliable for author attribution due to certificate abuse [11], [54]. Unlike these approaches, AndroLinker enables automated, reliable ecosystem-scale associations by statically and dynamically extracting identity-bound SDK IDs directly from app files, completely bypassing web scraping limitations and broken certificate chains of trust.

VII. Conclusion

We conducted a large-scale empirical measurement to demystify complex Android app associations. Recognizing the fragility of traditional metadata against evasion strategies, we introduced the SDK ID as a novel, developer identity-bound associated feature. We designed AndroLinker, a hybrid analysis pipeline that extracts SDK IDs as the core feature to construct explicit and implicit app associations. Our evaluation demonstrates the effectiveness of SDK IDs, and AndroLinker significantly outperforms existing authorship identification tools. Deploying AndroLinker on a real-world dataset, we successfully unmasked cross-market PUA groups employing dual-profit and diverse evasion strategies. Also, we revealed the implicit commercial matrices of technology companies. Besides, our findings motivated the major app markets to permanently remove 2,515 PUAs, demonstrating AndroLinker's real-world impact on ecosystem governance and app review.

Acknowledgment

We thank our anonymous reviewers for their insightful comments. This work was supported by State Key Laboratory of Cryptography and Digital Economy Security, Shandong University (No. KFYB2501), Key R&D Program of Shandong Province, China (No. 2024CXGC010114, 2025CXPT085), National Natural Science Foundation of China under Grant (No. 62372268), Natural Science Foundation of Shandong Province (Grant No. ZR2025MS991), National Key Research and Development Program of China under Grant 2024YFB3108500, National Radio and Television Administration Laboratory Program (FYY20230001ZSB011), Young Scientists Fund of the National Natural Science Foundation of China (Grant No. 62402277), Postdoctoral Fellowship Program of CPSF grants GZC20231361, and Startup Fund for Young Faculty at SJTU (SFYF at SJTU).

References

[1] StatCounter, "Mobile Operating System Market Share Worldwide," https://gs.statcounter.com/os-market-share/mobile/worldwide, 2025.

[2] Bill Toulas, "Android Minecraft clones with 35M downloads infect users with adware," https://www.bleepingcomputer.com/news/security/android-minecraft-clones-with-35m-downloads-infect-users-with-adware/, accessed: 2026-03-01.

[3] G. Xu, C. Zhang, B. Sun, X. Yang, Y. Guo, C. Li, and H. Wang, "Appauth: Authorship attribution for android app clones," *IEEE Access*, 2019.

[4] K. Chen, P. Liu, and Y. Zhang, "Achieving accuracy and scalability simultaneously in detecting application clones on android markets," in *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, 2014.

[5] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in *Second ACM Conference on Data and Application Security and Privacy, CODASPY 2012, San Antonio, TX, USA, February 7-9, 2012*, 2012.

[6] W. Wang, G. Meng, H. Wang, K. Chen, W. Ge, and X. Li, "A^{3}ident: A two-phased approach to identify the leading authors of android apps," in *IEEE International Conference on Software Maintenance and Evolution, ICSME 2020, Adelaide, Australia, September 28 - October 2, 2020*, 2020.

[7] H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Authorship attribution of android apps," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY 2018, Tempe, AZ, USA, March 19-21, 2018*, 2018.

[8] V. Kalgutkar, N. Stakhanova, P. Cook, and A. Matyukhina, "Android authorship attribution through string analysis," in *Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018, Hamburg, Germany, August 27-30, 2018*, 2018.

[9] Y. Gao, H. Wang, L. Li, X. Luo, G. Xu, and X. Liu, "Demystifying illegal mobile gambling apps," in *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, 2021.

[10] G. Hong, Z. Yang, S. Yang, X. Liao, X. Du, M. Yang, and H. Duan, "Analyzing ground-truth data of mobile gambling scams," in *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, 2022.

[11] K. Hageman, Á. Feal, J. Gamba, A. Girish, J. Bleier, M. Lindorfer, J. Tapiador, and N. Vallina-Rodriguez, "Mixed signals: Analyzing software attribution challenges in the android ecosystem," *IEEE Trans. Software Eng.*, 2023.

[12] Jin-Hyuk Kim, Yidan Sun, and Liad Wagman, "The Value of Technology Releases in the Mobile App Ecosystem," https://datacatalyst.org/wp-content/uploads/2021/01/The-Value-of-Technology-Releases-in-the-Mobile-App-Ecosystem-012921.pdf, 2021.

[13] J. Xu and Q. Yuan, "LibRoad: Rapid, Online, and Accurate Detection of TPLs on Android," *IEEE Trans. Mob. Comput.*, 2022.

[14] "Ground Truth Dataset," https://github.com/chicharitomu14/AndroLinker_Ground_Truth_Data, accessed: 2026-03-04.

[15] "HUAWEI Mobile Services," https://developer.huawei.com/consumer/en/hms/, accessed: 2024-06-28.

[16] Z. Ma, H. Wang, Y. Guo, and X. Chen, "Libradar: fast and accurate detection of third-party libraries in android apps," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, 2016.

[17] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in android and its security applications," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, 2016.

[18] J. Zhang, A. R. Beresford, and S. A. Kollmann, "Libid: reliable identification of obfuscated third-party android libraries," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*, 2019.

[19] Y. Wu, C. Sun, D. Zeng, G. Tan, S. Ma, and P. Wang, "Libscan: Towards more precise third-party library identification for android applications," in *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, 2023.

[20] OPPO, "OPPO App Market," https://developers.oppomobile.com/newservice/capability?pagename=app_store, accessed: 2026-03-01.

[21] "Anzhi Market," http://www.anzhi.com/, accessed: 2024-06-28.

[22] "Wandoujia," https://www.wandoujia.com/, accessed: 2026-03-01.

[23] "Baidu App Store," https://shouji.baidu.com/, accessed: 2026-03-01.

[24] "Duote," https://www.duote.com/android/, accessed: 2026-03-01.

[25] "CoolAPK," https://www.coolapk.com/, accessed: 2026-03-01.

[26] "Sogou Download," http://as.sogou.com/, accessed: 2026-03-01.

[27] "Tencent Appstore," https://appstore.tencent.com/, accessed: 2026-03-01.

[28] "Mumayi," http://www.mumayi.cn/, accessed: 2026-03-01.

[29] fakedatingapp, "fakedatingapp," https://github.com/fakedatingapp/fakedatingapp, accessed: 2026-03-01.

[30] Y. Hu, H. Wang, Y. Zhou, Y. Guo, L. Li, B. Luo, and F. Xu, "Dating with scambots: Understanding the ecosystem of fraudulent dating applications," *IEEE Trans. Dependable Secur. Comput.*, 2021.

[31] Anthony Desnos, "Androguard," https://github.com/androguard/androguard, accessed: 2024-06-28.

[32] C. Zuo, Z. Lin, and Y. Zhang, "Why does your data leak? uncovering the data leakage in cloud from mobile apps," in *Proceedings of the 2019 IEEE Symposium on Security and Privacy*, San Francisco, CA, May 2019.

[33] remote android, "redroid-doc," https://github.com/remote-android/redroid-doc, accessed: 2025-09-10.

[34] Y. Li, Z. Yang, Y. Guo, and X. Chen, "Droidbot: a lightweight ui-guided test input generator for android," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, 2017.

[35] Intel, "Intel Bridge Technology," https://www.intel.com/content/www/us/en/developer/topic-technology/bridge-technology.html, accessed: 2025-09-10.

[36] "TopOn SDK," https://www.toponad.com/en, accessed: 2026-03-01.

[37] "Meizu SDK," https://open.flyme.cn/, accessed: 2026-03-01.

[38] E. Aydogan and S. Sen, "Android authorship attribution using source code-based features," *IEEE Access*, 2024.

[39] Google, "Top free casino games," https://play.google.com/store/apps/category/GAME_CASINO, accessed: 2026-03-01.

[40] ——, "Inappropriate Content," https://support.google.com/googleplay/android-developer/answer/9878810?hl=en, accessed: 2026-03-01.

[41] Southern Metropolis Daily, "The manually tested "safe" apps still contain pornographic content and fraudulent payment behaviors. How did OPPO conduct the checking? [In Chinese]," https://m.mp.oeeee.com/a/BAAFRD000020210812541413.html, 2021.

[42] OPPO SRC, "Security Intelligence Scoring Standards V2.0," https://security.oppo.com/en/noticeDetail?notice_only_key=NOTICE-1555015013067137024, accessed: 2026-03-01.

[43] ByteDance, "ByteDance," https://www.bytedance.com/, accessed: 2026-03-01.

[44] ByteDance, "Douyin," https://www.douyin.com/, accessed: 2026-03-01.

[45] TikTok, "TikTok," https://www.tiktok.com/, accessed: 2026-03-01.

[46] ByteDance, "Duoshan," https://www.duoshan.com/, accessed: 2026-03-01.

[47] ByteDance, "Banciyuan," https://www.bcy.net/, accessed: 2024-06-28.

[48] M. Y. Wong, M. Landen, M. Antonakakis, D. M. Blough, E. M. Redmiles, and M. Ahamad, "An inside look into the practice of malware analysis," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, 2021.

[49] S. Hanna, L. Huang, E. X. Wu, S. Li, C. Chen, and D. Song, "Juxtapp: A scalable system for detecting code reuse among android applications," in *Detection of Intrusions and Malware, and Vulnerability Assessment - 9th International Conference, DIMVA 2012, Heraklion, Crete, Greece, July 26-27, 2012, Revised Selected Papers*, 2012.

[50] O. Gadyatskaya, A. Lezza, and Y. Zhauniarovich, "Evaluation of resource-based app repackaging detection in android," in *Secure IT Systems - 21st Nordic Conference, NordSec 2016, Oulu, Finland, November 2-4, 2016, Proceedings*, 2016.

[51] J. Gray, D. Sgandurra, L. Cavallaro, and J. B. Alís, "Identifying authorship in malicious binaries: Features, challenges & datasets," *ACM Comput. Surv.*, 2024.

[52] Y. Guo, D. Wang, L. Wang, Y. Fang, C. Wang, M. Yang, T. Liu, and H. Wang, "Beyond app markets: Demystifying underground mobile app distribution via telegram," in *Abstracts of the 2025 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2025, Stony Brook, NY, USA, June 9-13, 2025*, 2025.

[53] S. Sebastián and J. Caballero, "Towards attribution in mobile markets: Identifying developer account polymorphism," in *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, 2020.

[54] H. Wang, H. Liu, X. Xiao, G. Meng, and Y. Guo, "Characterizing android app signing issues," in *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, 2019.