Beyond the Horizon: Exploring Cross-Market Security Discrepancies in Parallel Android Apps

Shishuai Yang*†, Guangdong Bai‡(✉), Ruoyan Lin*†, Jialong Guo*†, and Wenrui Diao*†(✉)
*School of Cyber Science and Technology, Shandong University
{shishuai, 202337028, 202237083}@mail.sdu.edu.cn, diaowenrui@link.cuhk.edu.hk
†Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University
‡The University of Queensland, g.bai@uq.edu.au

***Abstract*—Multi-channel distribution of Android apps offers convenience to users, yet simultaneously introduces security concerns. Although apps published on Google Play and third-party markets share the same version code, differences in app content may still arise. Notably, a recent incident involving the third-party market version of Pinduoduo app containing malicious code highlights the intentionally-differentiated implementations of app functionalities by developers between Google Play and third-party markets. The case of Pinduoduo may be just the tip of the iceberg, underscoring the need for a comprehensive investigation of the disparities between Google Play and third-party market versions of apps.**

In this work, we systematically analyze the differences in security and privacy of cross-market apps that claim to share the same version code. Specifically, we propose three research questions that cover differences in app protection, security threats, and permission usage. To answer these questions, we constructed a dataset containing 17,218 app pairs (filtered from 236,731 apps) and permission mappings (27,046 SDK mappings, 1,656 ContentProvider mappings, and 309 Intent mappings) for API levels 16 - 33. This dataset enables us to perform a comprehensive differential analysis. Consequently, our investigation unveiled a series of captivating and insightful findings. Approximately 29.02% of apps show differences in one or all three aspects. For example, the third-party market versions of apps often request more permissions compared to their Google Play counterparts, particularly among apps in the game category. Our work can help developers and app store operators improve cross-market app consistency, enhancing the quality of the Android app ecosystem and user experience.

I. Introduction

The openness of Android's app distribution channels is a key feature that distinguishes it from other mobile OSes such as iOS. Android developers can release their apps not only through the official app store (Google Play), but also through various third-party markets. While the multi-channel distribution offers convenience to users across different regions, it also introduces security concerns. Apps distributed through Google Play undergo a rigorous review process by the Google Play team before being released to users. In contrast, the review requirements in third-party markets are often not as rigorous as those of Google Play. For purposes such as data collection and advertising fraud, developers may thus opt to release their apps on third-party markets to circumvent Google Play's review and restrictions. This practice increases the risk of downloading malicious apps from third-party markets.

The case of Pinduoduo app [14], a popular mobile e-commerce shopping platform with approximately 900 million users, has demonstrated a real-world example of the risks associated with third-party market distribution. A recent report [13] revealed that its third-party market version contains malicious code that can exploit the Android OS. This code can potentially prevent users from removing the app from their devices, and can even install malicious apps in the background, remove other legitimate apps, and spy on users. In contrast, there is no evidence suggesting that the Google Play version of Pinduoduo contains malicious code. The Pinduoduo case may only be the tip of the iceberg. It highlights that the differences in implementation between the Google Play version and third-party market version of apps could be considerably significant (we will refer to Google Play as "**GP**", third-party market as "**3rdPM**"). In our study, we focus on analyzing cross-market versions of apps that have the same version code, as such versions are expected to maintain consistent security behaviors across all distribution channels.

To the best of our knowledge, there is currently no systematic research on the differences between the GP and 3rdPM versions of Android apps. This may be due to two difficulties in constructing the dataset: (1) Developers may not publish apps simultaneously on both GP and 3rdPM or maintain the same version. (2) Google Play currently only supports downloading the latest version of apps, not a specific version. The most relevant work was carried out by Wang et al. [36]. Through a comprehensive comparative study of GP and 3rdPMs, they pointed out that 3rdPMs have a higher proportion of malware, frequent instances of fake and cloned apps, and apps that often request excessive permissions and lack timely updates. In contrast, we conducted a complementary study by exploring the differences in security between the GP and 3rdPM versions of apps, such as permissions and vulnerabilities, which have not been systematically studied.

Our Work. The Pinduoduo case highlights a potential increased risk of security issues in the 3rdPM versions of apps. Therefore, our research is driven by the need to understand the variations in security and privacy between the GP and 3rdPM versions of apps. Specifically, we explore these differences across three key dimensions: app protection, security threats,

and permission usage. More precisely, our objective is to answer the following three research questions:

⇒ **RQ1** (***app protection***): *Is there any difference in the deployment of app protection measures between the GP and 3rdPM versions of apps?*

⇒ **RQ2** (***security threats***): *Is there any difference in vulnerabilities and malicious behaviors between the GP and 3rdPM versions of apps?*

⇒ **RQ3** (***permission usage***): *Is there any difference in permission usage between the GP and 3rdPM versions of apps?*

The answer to RQ1 highlights differences in apps' basic protection capabilities. Furthermore, the answer to RQ2 reveals disparities in vulnerabilities and malicious behaviors. These security threats may affect the security of the entire Android OS. Finally, RQ3 explores variations in permission usage, offering valuable insights into privacy risks and permission management practices.

To answer these research questions, we first constructed a dataset of 236,731 3rdPM apps, with 44,283 apps also available on GP. From them, we filtered the apps of which GP version and 3rdPM version have the same version code, and 17,218 pairs of apps (i.e., GP and 3rdPM versions) were kept for analysis. In our analysis, we first adopted a series of heuristic methods to identify the differences in app protections, such as packer and signature verification. Then, we used state-of-the-art vulnerability detection tools (such as `MobSF` [10] and `AndroBugs` [1]) to identify differences in vulnerabilities among apps. In addition, we uploaded apps to `VirusTotal` [17] to compare differences in malicious behaviors. Finally, for the differences in permissions, we constructed permission mappings for API levels 16 - 33 to identify SDK APIs, Content Uris, and Intent Actions protected by permissions in apps, including 27,046 SDK mappings, 1,656 ContentProvider mappings, and 309 Intent mappings.

Key Findings. Although GP and 3rdPM versions of most apps remain consistent in terms of security, those with differences typically exhibit the following features.

- **Protection Measures Degradation.** Developers often minimize certain security features like packers and signature verification in GP versions of apps due to stringent security reviews. Conversely, they enhance protection in 3rdPM versions to boost app security.
- **App Repackage.** The differences in the certificate's Organization fields indicate that the 3rdPM version or GP version of apps may be generated through repackaging, suggesting the possibility of copyright violation.
- **Enhanced Permission Requests.** The 3rdPM version of apps tends to request more system permissions, averaging 10 extra system permissions for each app. Our further investigation found that excessive system permission requests are caused by third-party libraries (**TPLs** for short) rather than by developer code. Moreover, a significant portion of these permissions remain unused in practice.
- **Heightened Vulnerability and Malware Risks.** Due to excessive permission requests and the usage of TPLs, the 3rdPM version of apps is more susceptible to introducing vulnerabilities and being flagged as malware.

Contributions. The main contributions of this paper are:

- **Systematic Differential Study.** We conduct the first systematic study on the differences in security and privacy between the GP and 3rdPM versions of apps from app protection, security threats, and permission usage. In particular, we proposed and answered three significant research questions with sufficient supporting evidence.
- **Practical Results.** This work can enhance the overall security and transparency of the mobile app ecosystem. Furthermore, it provides valuable insights for app developers and store operators regarding app development and the regulation and governance of the mobile app market. This work can also enhance users' awareness of security.
- **Open Data Sharing.** The raw measurement data for each research question, as well as the permission mappings we generated for Android API levels 16 - 33, are available at https://doi.org/10.5281/zenodo.13232037. The permission mappings will contribute to the research community, facilitating studies on apps' privacy compliance checks and malicious behavior analysis.

II. Background

In this section, we provide the necessary background of the Android permission mechanism and app protection measures.

A. Android Permission Mechanism

The permission mechanism is an Android OS-level security mechanism designed to manage the access permissions of apps to system resources and sensitive data [12]. It is based on the Linux access control model and ensures that apps can only access specific functions and data with authorization by allocating and managing permissions. In Android, permissions are categorized into different groups, each representing a specific set of system functionalities or sensitive data. For instance, permissions can control whether an app can access the camera, location information, contacts, etc. In contrast to system permissions, custom permissions are used to protect the data generated by apps and are applied to the four major components of apps.

The permissions for Android are divided into three levels: normal, dangerous, and signature. Normal-level permissions typically involve the basic functionalities of apps, and users do not need to provide explicit authorization. Once apps request these permissions, they are automatically granted. Dangerous-level permissions involve sensitive data and system functionalities, requiring user authorization at runtime. Signature-level permissions require the requesting app to be signed with the same digital certificate as the app declaring the permission, ensuring that the permissions are used by legitimate apps.

B. App Protection Measures

Packer. App packer refers to hiding the original DEX file of the app so that the actual DEX file within the app is the packer code [38]. During app runtime, packer code often customizes the Application class to achieve the decryption and restoration of the original DEX files, as this class serves as the entry point of the app. Typically, developers will use security vendors' packer products rather than attempting to implement their own, as the packing process entails intricate technicalities and specialized expertise. Security vendors' products can deliver a heightened level of professional assurance. Therefore, detecting whether an app is packed mainly involves detecting which packer products the app has used.

APK Signature. Signatures are primarily used to verify the authenticity and integrity of content. APK typically includes two types of signatures [16]: (1) Signature of the developer certificate: The developer certificate primarily includes core information such as the certificate holder, issuer, and public key, all formatted according to the X.509 standard. If two apps are published by the same developer, then the certificate's signature or holder is usually the same. (2) Signature of APK content: Developers sign the APK using a private key and embed the corresponding public key within the developer certificate. When users install or update an app, the Android OS utilizes public key to verify the APK's signature, ensuring the app has not been tempered.

Anti-Analysis Techniques. Anti-debugging, anti-VM, and anti-root are used to enhance the security of apps [21]: (1) Anti-debugging is used to prevent malicious attackers from using debuggers to analyze the operation of apps, modify code, or obtain sensitive information. (2) Anti-VM aims to detect whether apps are running in a virtual environment to prevent attackers from executing malicious operations or analyzing the behavior of apps. (3) Anti-root aims to detect whether a device has been rooted to prevent attackers from using root privileges to modify, tamper with, or access sensitive data in apps.

III. Methodology and Dataset

This section illustrates our measurement approach and constructed datasets to answer the proposed research questions.

A. Methodology

As illustrated in Figure 1, on a high level, our measurement contains three main steps, as follows:

- App Collection. First, we constructed the app dataset for differential analysis used in this study, including the GP and 3rdPM versions of apps.
- Permission Mappings Generation. Next, we generated the permission mappings for subsequent permission differential analysis, including SDK mappings, Intent mappings, and ContentProvider mappings.
- Differential Analysis. Based on the constructed app dataset and permission mappings, we employ a combined dynamic and static differential analysis approach to answer the proposed research questions.

B. App Collection

Collection of 3rdPM Version of Apps. To gather as many apps as possible published on both GP and 3rdPM, we first built a large dataset from ten popular 3rdPMs, including F-Droid, 9apps, 360, 2265, Anzhi, LapTopPCAPK, Lenovo, Leyou, Mdpda, and Uptodown. For each app market, we launched multiple processes in parallel to crawl apps simultaneously and used the app name as an identifier to determine whether it had already been crawled. As developers may upload apps to multiple 3rdPMs, we calculated the MD5 value of each app to remove duplicate apps. Furthermore, to guarantee that each app in the 3rdPM dataset possesses a unique version, we removed apps that shared same package names but had differing version codes. Some corrupted apps were also excluded. As a result, we obtained a total of 236,731 apps.

Collection of GP Version of Apps. After constructing the 3rdPM dataset, we checked whether these apps were published on GP. The URL of the app's detail page on GP includes an id parameter, and the value of id parameter is the app's package name[1]. Therefore, we obtained the package names of all apps in the 3rdPM and constructed the corresponding URLs to send HTTPS requests to the GP server. If the status code of HTTPS response is 200, the app exists on GP. Among the 236,731 apps from the 3rdPM dataset, 44,283 apps are also published on GP. After obtaining the list of apps available on 3rdPM and GP, we also recorded the version code of each app.

Since Google Play no longer provides an API for bulk downloading apps, we used Raccoon [15] to download apps from GP. Since GP cannot be accessed anonymously, we need to use a Google Account to log in to Raccoon before downloading apps. Raccoon will mimic a high-end smartphone through the device profile, connect to the GP server, and download apps. All communications between Raccoon and GP adhere to Google's privacy statement [20], ensuring compliance with GP's guidelines and avoiding any breach of their policies. Raccoon also supports downloading historical versions of apps by specifying the package name and version code. However, Raccoon can only download one app at a time, so we must use a script to automate the apps' downloading process. Although 44,283 apps are available on both 3rdPM and GP, developers may release different versions on 3rdPM and GP. For example, developers may release version 001 of an app on 3rdPM and version 002 on GP, meaning not all 44,283 apps will have the same versions available on GP. Based on the recorded package names and version code of the 3rdPM version of apps, we use Raccoon to send download requests to GP for all 44,283 apps one by one. After completing the download of a certain number of apps, Raccoon may become unresponsive, which leads to us having to restart it periodically. Therefore, the downloading process is not continuous but has been intermittently ongoing for approximately ten days. We can only download 17,218 apps in ten days as the 3rdPM versions of many apps do not have matching versions available on GP.

[1]URL: https://play.google.com/store/apps/details?id="package name"

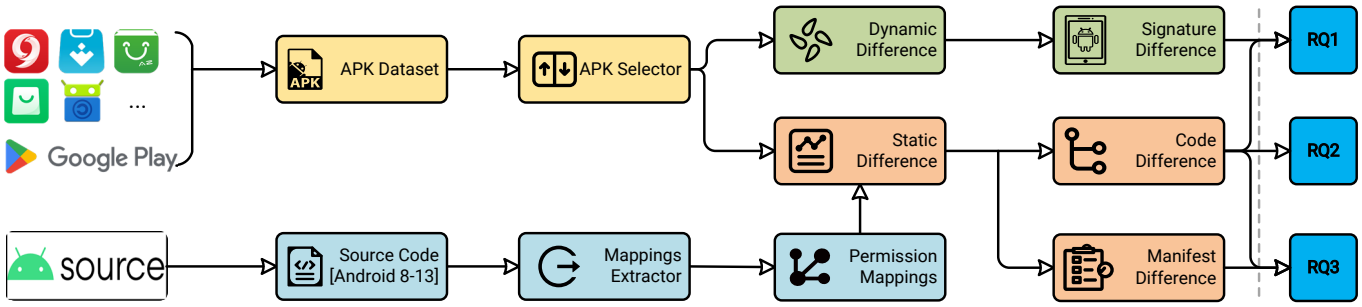

Fig. 1: Overview of measurement flow.

Apps Filter. We determine if two APK files are exactly the same by calculating their MD5 values. Apps with the same hash values are assured of having the same content. Since this work primarily focuses on apps with the same version code but exhibit differences in content, we must remove the GP and 3rdPM versions of apps with the same `MD5` value from the dataset. First, we calculated the `MD5` value of the GP and 3rdPM versions of `APK` files. Among 17,218 app pairs, the `MD5` values of 33.85% are the same. As `APK` files are essentially `ZIP` compressed packages, we calculated the `MD5` value of each file in `APK` for comparison. 37.13% of app pairs have different `MD5` values of `APK` files but the same `MD5` value for all their internal files. In this case, it may be because these two apps used different packaging methods, such as packaging tools, packaging options, or signing methods. After removing the app pairs with the same `MD5` value, our dataset is left with 4,997 app pairs for subsequent differential analysis.

C. Permission Mappings Generation

In Android, permissions are typically mapped to three entities: API Calls, Intent Actions, and Content Uris. During the permission differential analysis, we need to use permission mappings to determine whether the extra requested permissions are actually used. Although `Axplorer` [5] provides permission mappings for API levels 16 - 25, `Axplorer` is no longer maintained. As a result, we cannot obtain permission mappings for API levels 26 - 33. Therefore, we must construct the latest permission mappings.

Permission Annotation and Comments. Since Android 6.0 (API level 23), Google officially records permission mappings in two ways: (1) Using Java annotation `@RequiresPermission("PermissionName")` to associate API with permissions. (2) Using `@link android.Manifest.permission#"PermissionName"` to describe the permissions required to call this API in the comments.

SDK Mappings Generation. To determine whether the SDK APIs called within the apps are protected by permissions, we need to construct the API to permission mapping. Each API level corresponds to a version of the Android OS and provides a specific set of APIs for app development. Therefore, we downloaded the source code of Android 8 - 13, which includes eight Android versions: 8.0, 8.1, 9.0, 10.0, 11.0, 12.0, 12.1, and 13.0, corresponding to API levels 26 - 33, respectively. To facilitate the subsequent extraction of SDK mappings through annotations and comments (abbreviated as A&C), we extracted the SDK source from each of the above Android versions.

The permission mappings provided by `Axplorer` are incomplete, lacking some permission mappings, such as `CAMERA` and `INTERNET`. In addition, the SDK mappings extracted through A&C for API levels 26 - 33 may also be incomplete. Therefore, we generate complete SDK mappings for API levels 16 - 33 by merging the SDK mappings generated by A&C with those generated by `Axplorer`. We use the SDK mappings generation for API level 26 as an example to explain how we generate complete SDK mappings for API levels 16 - 33, as shown in Figure 2:

(1) SDK Mappings Generation by Axplorer. First, we generated partial SDK mappings for API level 26 based on the SDK mappings provided by `Axplorer` for API level 25. We need to check whether the APIs in the SDK mappings for API level 25 exist in the SDK source of API level 26 and whether they are deprecated. If an API is marked as deprecated, it may be removed in future Android versions but is still usable in the current version. Google usually offers alternative APIs in the deprecated API's comments. Invoking alternative APIs usually requires the same permissions as the deprecated APIs.

- If APIs in SDK mapping for API level 25 do not exist in the SDK source of API level 26, we will not perform any operation (path: ①→③).
- If APIs in SDK mappings for API level 25 exist in the SDK source of API level 26 and are not deprecated, we will add them to the SDK mappings for API level 26 (path: ①→②→④→⑥).

```
1 /** ... @deprecated Use {@link getImei}
      which returns IMEI for GSM or {@link
      getMeid} which returns MEID for CDMA.
      ... **/
2 @Deprecated
3 @RequiresPermission(android.Manifest.
      permission.READ_PHONE_STATE)
4 public String getDeviceId() { .... }
```

Listing 1: Example of deprecated getDeviceId().

- If APIs in SDK mappings for API level 25 exist in the SDK source of API level 26 but have been deprecated, we will add APIs and the alternative APIs to the SDK

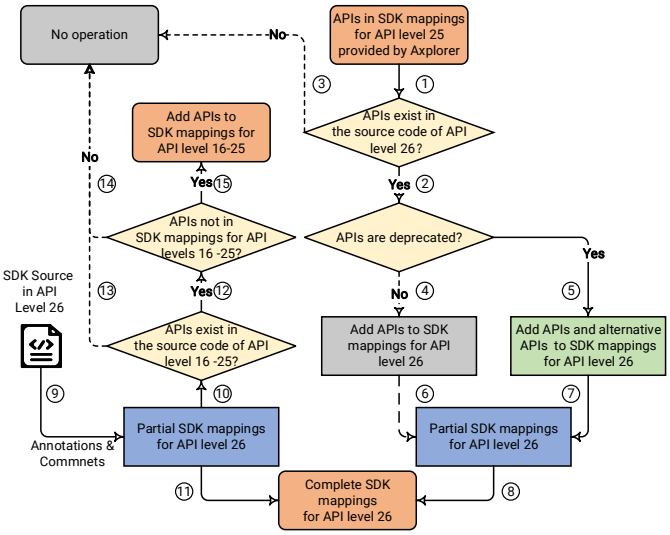

Fig. 2: SDK mappings supplement and generation.

mappings for API level 26 (path: ① → ② → ⑤ → ⑦). For example, as shown in Listing 1, the `getDeviceId()` was deprecated in API level 26. Google recommends using the `getImei()` or `getMeid()` as alternative for `getDeviceId()` (line 1). Calling `getImei()` or `getMeid()` requires the same permission as `getDeviceId()`. Therefore, we will add the `getDeviceId()`, `getImei()`, and `getMeid()` to the SDK mappings for API level 26.

(2) SDK Mappings Generation and Supplement through A&C. Next, we also extract partial SDK mappings for API level 26 from the SDK source of API level 26 through A&C. If APIs in SDK mappings for API level 26 exist in the SDK source of API levels 16-25. Also, these APIs do not exist in the SDK mappings for API levels 16 - 25 provided by `Axplorer`. We will add these APIs to the SDK mappings for API levels 16 - 25, thereby enriching the SDK mappings supplied by `Axplorer`. (path: ⑨ → ⑩ → ⑫ → ⑮). Otherwise, we will not perform any operation (path: ⑨ → ⑩ → ⑬ or ⑨ → ⑩ → ⑫ → ⑭).

(3) Complete SDK Mappings Generation. Finally, we will generate the complete SDK mappings for API level 26 by merging two partial SDK mappings: (a) SDK mappings generated by API level 25 (① → ② → ④ → ⑥ → ⑧ ∪ ① → ② → ⑤ → ⑦ → ⑧). (b) SDK mappings extracted from the SDK source of API level 26 through A&C (⑨ → ⑪). In addition, we will remove some duplicate SDK mappings from two partial SDK mappings.

We supplement and generate complete SDK mappings for API levels 16 - 33 by iterating the above steps. Notably, when using comments to extract SDK mappings, permissions may appear in the comments of APIs, but calling APIs may not require those permissions. We manually verified and excluded these APIs from the SDK mappings.

ContentProvider Mappings and Intent Mappings Generation. To determine whether the Content Uris and Intent Actions used in apps are protected by permissions, we need to construct ContentProvider mappings and Intent mappings. `Axplorer` provides ContentProvider mappings for API levels 16 - 25. To obtain ContentProvider mappings for API levels 26 - 33, we retrieved all system ContentProviders protected by permissions from the source code of Android versions 8 - 13. For these ContentProviders, we extracted Content Uris and their corresponding permissions. Furthermore, we also extract Intent mappings from the source code of `android.content.Intent` class through A&C. This class contains all Intent Actions along with required permissions.

D. Differential Analysis

After constructing the app dataset and permission mappings, the App Selector will perform differential analysis by selecting the GP and 3rdPM versions of apps. Below is an overview of our differential analysis steps, while a more detailed explanation can be found in Section IV.

- For RQ1, we adopted a combined dynamic and static differential analysis approach to analyze the differences in app protection measures.
- For RQ2, we employed state-of-the-art vulnerability scanning tools (`MobSF` and `AndroBugs`) and uploaded apps to `VirusTotal` to compare the differences in vulnerabilities and malware.
- For RQ3, we investigated the differences between the system and custom permissions used in the manifest files of apps. Furthermore, we conducted an in-depth analysis of the sources of these differing permissions and whether they are actually used.

IV. Findings

This section summarizes our empirical research results on the research questions proposed in Section I.

RQ1. Is there any difference in the deployment of app protection measures between the GP and 3rdPM versions of apps?

This question primarily explores the differences in apps' ability to resist reverse engineering and tampering, highlighting the variations in their basic protection capabilities.

Packers. `APKiD` [4] is based on Yara rules [18], which can identify packer products in apps efficiently, so we deployed `APKiD` to detect whether apps are packed. Furthermore, we also added some custom rules to increase the number of packers that can be detected. Among 4997 app pairs, 6.48% of 3rdPM apps are packed, compared to only 1.40% of GP apps being packed. Moreover, 5.12% of apps are exclusively packed in their 3rdPM versions, while a negligible 0.04% of apps are packed only in the GP version. The GP version of apps is more likely to reduce protection than the 3rdPM version, possibly attributable to GP's review policies. Google requires that apps must provide a unique user experience and cannot duplicate existing apps. Therefore, apps similar to existing apps on GP will be rejected from publication [8]. Packer will hide the original code of apps and only leave the packer code, which often has a high similarity across packer apps. Therefore, to publish apps on GP, developers may need

TABLE I: Distribution of selected packer products between the GP and 3rdPM versions of apps.

3rdPM \ GP	None	DexProtector	APKProtect	AppSealing	AppGuard	Jiagu	Bangcle	Ijiami	Tencent	SecNeo	Baidu	Alibaba	UUSafe	YiDun
None*	0	0	0	0	0	0	0	0	1	1	0	0	0	0
DexProtector	0	11	0	0	0	0	0	0	0	0	0	0	0	0
APKProtect	1	0	0	0	0	0	0	0	0	0	0	0	0	0
AppSealing	2	0	0	0	0	0	0	0	0	0	0	0	0	0
AppGuard	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Jiagu	131	0	0	0	0	31	0	0	1	0	0	0	0	0
Bangcle	38	0	0	0	0	0	0	0	0	0	0	0	0	0
Ijiami	2	0	0	0	0	0	0	1	0	0	0	0	0	0
Tencent	73	0	0	0	0	2	0	0	19	0	0	0	0	0
SecNeo	2	0	0	0	0	0	0	0	0	1	0	0	0	0
Baidu	4	0	0	0	0	0	0	0	0	0	0	0	0	0
Alibaba	1	0	0	0	0	0	0	0	0	0	0	0	0	0
UUSafe	1	0	0	0	0	0	0	0	0	0	0	0	0	0
YiDun	1	0	0	0	0	0	0	0	0	0	0	0	0	1

∗: "None" indicates that no packer product was used.

to add additional features to differentiate them from other packed apps. However, this also opens another avenue for attackers. When the 3rdPM version or the GP version of apps are packed, they can find an unpacked version from them to crack. Despite potential variations between the two versions, the core functionality of the app usually remains consistent.

The distribution of selected packer products for the GP and 3rdPM versions of apps is shown in Table I. The rows of the table represent the packer products used in 3rdPMs, while the columns represent the packer products used in GP. For example, "131" represents 131 apps that used Jiagu in the 3rdPM version, but were not packed in the GP version. Among these packer products, the most popular ones are Jiagu, Tencent, and DexProtector on GP and 3rdPM. Although many apps use Bangcle on 3rdPMs, no apps use Bangcle on GP. Google imposes clear transparency and compliance requirements for apps published on GP [19]. If GP's security policy deems that apps packed by certain packer products may contain malicious behavior or hidden features, these apps will be rejected from publication. This can lead to differences in the selection of packer products between the 3rdPM and GP versions of apps.

Certificate Signature. We extracted the SHA-1 signatures of the developer certificates from the GP and 3rdPM versions of apps and compared them. Among 4,997 app pairs, 63.3% of app pairs have the same SHA-1 signature, meaning that the GP and 3rdPM versions of these apps were developed by the same developers. 36.7% of the app pairs have different SHA-1 signatures of certificates. The Subject field is used to identify the certificate holder's information, which includes an Organization field to indicate the certificate holder's organizational name. Therefore, we thoroughly analyzed the Organization fields for these apps with different SHA-1 signatures. Among the 1,834 app pairs with different SHA-1 signatures, 1,205 apps have the Organization field of certificates set concurrently in both the GP and 3rdPM versions. For 99.67% of apps, the Organization field of certificates differs between the GP and 3rdPM versions. This could involve two potential scenarios: (1) Developers use different certificates in different markets. For example, the Organization field in the 3rdPM version of the Wireguard app[2] is fdroid.org, while the GP version is Google Inc. (2) Apps on 3rdPM or GP may be generated through repackaging. For example, Organization field in the 3rdPM version of the Supersolid app[3] is Unknown, while the GP version is Supersolid.

Signature Verification. Signature verification means that apps check their content for tampering by verifying signatures. If the content has been tampered with, it is typically designed to interrupt the operation of apps upon launch (pops up an error message box or crashes to the home screen). Furthermore, the code for app signature verification may exist within either the Java layer or the native layer, rendering it challenging to detect through static methods. Therefore, to analyze signature verification differences, performing dynamic testing by installing apps on the device is necessary. We excluded some apps that would crash upon launch, either the GP or 3rdPM version, and left 2,926 app pairs for subsequent analysis. For the detection of signature verification, we employed a script that automatically repackaged apps, applied signatures, and installed them on the device. The script will open an app and wait for 8 seconds, then use the adb (Android Debug Bridge) command to retrieve the current top element of the Activity stack. If it identifies an error prompt box or the home screen, we consider that the app has implemented signature verification.

Among 2,926 app pairs, 4.38% implement signature verification consistently across both GP and 3rdPM versions. Moreover, an exclusive adoption of signature verification in the 3rdPM versions is observed in 13.87% of apps. In contrast, a minimal fraction, accounting for only 0.58%, enforce signature verification solely within the GP version. It may be because GP includes code similarity checks, and repackaged apps are prohibited from publishing. Some 3rdPMs lack this check, making it essential for developers to add signature verification to protect the integrity of their apps. Furthermore, an alarming 81.17% of apps do not implement signature verification on both GP and 3rdPM versions, suggesting these apps are at significant risk of repackaging.

Anti-Analysis Techniques. We adopted `APKiD` to detect the deployment of anti-debugging and anti-VM in apps. Additionally, we used `MobSF` to evaluate anti-root deployment. Due

[2]Package: `com.wireguard.android`, Version Code: 491.

[3]Package: `com.supersolid.honestfood`, Version Code: 1000904904.

TABLE II: Vulnerability list for differential analysis.

Owasp-Mobile	Vulnerability Name	Description	Tools
M1: Improper Platform Usage	android_webview	Insecure WebView Implementation. Execution of user-controlled code in WebView is a critical Security Hole.	MobSF
	android_webview_debug	Remote WebView debugging is enabled.	MobSF
M2: Insecure Data Storage	android_temp_file	App creates temp file. Sensitive information should never be written in a temp file.	MobSF
	android_insecure_file_mode	Use insecure file mode (MODE_WORLD_READABLE or MODE_WORLD_WRITABLE).	AndroBugs
	android_keystore_protection	The Keystores are not protected by password or use "byte array" or "hard-coded cert info" to do SSL pinning.	AndroBugs
M3: Insecure Communication	android_insecure_ssl	Insecure Implementation of SSL. Trusting all certificates or accepting self-signed certificates is a critical Security Hole.	MobSF
	android_webview_ignore_ssl	Insecure WebView Implementation. WebView ignores SSL Certificate errors and accepts any SSL Certificate.	MobSF
	android_http	URLs that are NOT under SSL.	AndroBugs
M5: Insufficient Cryptography	android_insecure_random	The App uses an insecure Random Number Generator.	MobSF
	android_sha1	SHA-1 is a weak hash with hash collisions.	MobSF
	android_md5	MD5 is a weak hash known to have hash collisions.	MobSF
	cbc_padding_oracle	This configuration is vulnerable to padding oracle attacks.	MobSF
	android_aes_ecb	The App uses ECB mode in the encryption algorithm.	MobSF
	android_weak_ciphers	Weak Encryption algorithm used.	MobSF
M7: Client Code Quality	android_sql_raw_query	App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection.	MobSF
M9: Reverse Engineering	android_hardcoded	Files may contain sensitive hardcoded information such as usernames, passwords, and keys.	MobSF

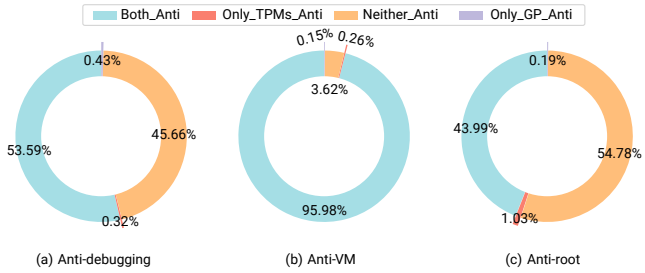

Fig. 3: Differences in the use of anti-analysis techniques.

to the need for static code analysis for anti-debugging, anti-VM, and anti-root detection, we exclude packed apps. The deployment differences of anti-debugging, anti-VM, and anti-root between the GP and 3rdPM versions of apps are shown in Figure 3. Most apps show consistency, with percentages of 99.25%, 99.60 %, and 98.77%, respectively. These apps either simultaneously deploy protection measures in both the GP and 3rdPM versions, or they are not deployed. Particularly in the case of the anti-VM, 95.98% of apps have adopted this protection on both the GP and 3rdPM versions. Running apps on a virtual machine is often the first step in performing specific app analysis, such as debugging apps, so this protection is commonly deployed in apps.

RQ2. Is there any difference in vulnerabilities and malicious behaviors between the GP and 3rdPM versions of apps?

This question primarily focuses on exploring the differences in vulnerabilities and malicious behaviors within apps, significantly impacting the security of the entire Android OS.

Vulnerabilities. Apps on 3rdPM may be more prone to contain vulnerable code implementations, as the review measures of 3rdPM might not be as rigorous as GP. However, it remains unknown whether there are differences in the vulnerabilities contained in the GP and 3rdPM versions of apps. The automated mobile app security evaluation framework, `MobSF` [10], is designed to conduct thorough security scans on apps. To investigate the differences in vulnerabilities, we primarily utilized `MobSF` for detecting vulnerabilities. It categorizes detected vulnerabilities according to severity levels such as High, Warning, and Info. We mainly focused on the vulnerabilities classified as High and Warning levels since they pose higher security risks. Similar to `MobSF`, `AndroBugs` [1] is also a security evaluation framework. We deployed `AndroBugs` to complement our detected vulnerabilities. By combining the detection results from `MobSF` and `AndroBugs`, we mapped the identified vulnerabilities to six categories of the OWASP Top 10 Mobile Risks [11], as shown in Table II. Notably, security risks in other categories are not caused by app code. This work primarily focuses on security risks caused by app code.

The "Vulnerability Differences" of Table III illustrates the differences in vulnerability contained between the GP and 3rdPM versions of apps. Most apps exhibit consistency for each vulnerability. Whether the 3rdPM or GP versions of apps, they either both contain vulnerabilities or neither contains vulnerabilities. We have discovered some common vulnerabilities, including android_http, android_insecure_random, and android_hardcoded. More than 80% of apps contained these vulnerabilities on the GP and 3rdPM versions. This is mainly due to some TPLs. For example, in the case of android_http, many apps use TPLs that include code implementations for cleartext traffic communication, such as com.google.android.gms and org.openxmlformats.schemas. Although GP employs a more rigorous security review process, most developers do not address the security vulnerabilities of the 3rdPM versions when publishing the GP versions of apps. Meanwhile, the occurrence rates of some vulnerabilities are relatively low, such as android_insecure_file_mode, android_insecure_ssl, android_webview_ignore_ssl, android_aes_ecb, and android_wea

k_ciphers. More than 80% of apps on both the GP and 3rdPM versions do not contain these vulnerabilities.

For each vulnerability, a small portion of apps exhibit differences, meaning that the vulnerabilities are present only in the GP or 3rdPM version. The proportion of differences in vulnerability contained between the GP and 3rdPM versions of apps ranges from 0.39% - 2.53%. The android_webview_debug vulnerability exhibits the smallest difference, while the android_md5 vulnerability exhibits the largest difference. For the M2, M3, M5, M7, and M9 in the OWASP Top 10 Mobile Risks, if there are differences in the vulnerabilities contained in the 3rdPM and GP versions of apps, the 3rdPM version is more likely to contain vulnerabilities. However, M1 shows differences, and the GP version is more prone to contain vulnerabilities than the 3rdPM version.

To understand the causes behind the differences in vulnerabilities, it's crucial to determine the source of the vulnerabilities—whether they stem from developers' coding practices or the integration of TPLs. We mainly used `Libd` [29] to identify TPLs. Additionally, we used the TPL feature files provided by `LibRadar` [31] and manually added some TPLs to the feature file to supplement the number of TPLs we can identify. This is outlined in the "Responsible Party" of Table III. The differences in vulnerabilities contained in the 3rdPM and GP versions of apps are primarily caused by TPLs, and developers rarely actively introduce vulnerabilities. In the case of android_aes_ecb, the two versions of apps exhibited the largest difference. In the 3rdPM version of apps, only 0.83% of vulnerabilities were introduced by developer code, whereas in the GP version, 22.22% of vulnerabilities were introduced by developer code. After conducting reverse engineering on apps, the primary reason is that apps added BLE (Bluetooth Low Energy) related functionalities in the GP version, such as device discovery and connection management. However, the AES algorithm with ECB mode was used for communication encryption.

Malicious Behavior. `VirusTotal` [17] is an online malware detection service incorporating dozens of antivirus engines, so we uploaded apps to `VirusTotal` to obtain the differences in malware. If any antivirus engine identifies apps as malicious, then we consider apps to be malicious. 17.19% of the 3rdPM version of apps are identified as malicious, while the GP version is benign. Only 0.94% of the GP versions of apps are identified as malicious, while the 3rdPM versions are benign. 6.27% (293) of the GP and 3rdPM versions of apps are both identified as malicious. Among these 293 apps, the 3rdPM version is typically identified as malicious by an average of six antivirus engines, whereas the GP version is identified as malicious by an average of two antivirus engines. Malware detection discrepancies between the GP and 3rdPM versions of apps are notably greater than vulnerability detection. 3rdPM versions are more frequently identified as malicious or flagged by more antivirus engines. Thus, integrating specific TPLs or requesting excessive permissions (described in RQ3) in the 3rdPM version of apps is more likely to be recognized as malware by antivirus engines.

RQ3. Is there any difference in permission usage between the GP and 3rdPM versions of apps?

This question primarily focuses on whether apps request extra permissions in the 3rdPM or GP versions and whether these permissions are actually used. It offers valuable insights into privacy risks and permission management practices.

System Permissions. Firstly, we extracted the manifest files of the GP and 3rdPM versions of apps for differential analysis. Among 4,671 app pairs, 13.17% of apps in the 3rdPM version request system permissions that are not present in the GP version, averaging 10 extra system permissions for each app. 0.64% of apps in the GP version request system permissions that are not present in the 3rdPM version, averaging 2 extra system permissions for each app. 0.56% of apps requested unique permissions in both the GP and 3rdPM versions. Many apps request more system permissions in the 3rdPM versions compared to their GP versions, likely due to Google's permission declaration form mechanism [6]. Developers must submit a permission declaration form to the GP team when publishing apps on GP. If any requested permissions are not listed on this form, the app cannot be published. The GP team reviews each app's core functionalities to ensure that all requested permissions are necessary for their intended usage, avoiding unnecessary permissions. On the other hand, 3rdPM may enforce more relaxed scrutiny on permissions.

To confirm whether the differences in permission requests are related to the category of apps, we used the google-play-scraper [7] to retrieve metadata for these apps that exhibit differing permission requests from GP. The category distribution for these apps is illustrated in Figure 4. Whether in 3rdPM or GP, apps categorized as game tend to request more permissions. Game apps may need to access various hardware resources on the device, such as camera, microphone, and storage. These permissions enable features such as augmented reality, voice chat, and game recording, enhancing the gaming experience. Retaining excessive permissions in benign apps can be exploited by malicious apps. Therefore, game apps should require more attention and scrutiny regarding security and privacy, especially permission control.

Sources Analysis. Since many 3rdPM versions of apps request more system permissions than the GP version, we primarily focus on two questions: (1) *Do the extra requested permissions in the 3rdPM version actually get used?* (2) *If the extra requested permissions are used, who uses them – developer code or TPLs?*

After extracting permission mappings (Section III-C), we performed a static analysis on the 3rdPM version of apps that request extra permissions using `AndroGuard` [2]. We retrieved the SDK APIs, Intent Actions, and Content Uris used in these apps and recorded their usage locations. Notably, apps typically use Content Uris in two ways: (1) Directly using the Content Uris string. (2) Utilizing pre-defined Content Uris constant provided by the Android OS. For the recognition of Content Uris constants, we need to pre-extract the mappings of Content Uris constant to Content Uris string from the

TABLE III: Differences in vulnerabilities and their responsible parties between GP and 3rdPM versions of apps.

Owasp-Mobile	Vulnerability Name	Vulnerability Differences				Responsible Party (3rdPM)		Responsible Party (GP)	
		Both*	3rdPM*	GP*	Neither*	TPLs	Developer Code	TPLs	Developer Code
M1	android_webview	54.74%	0.24%	0.36%	44.66%	92.86%	7.14%	96.88%	3.12%
	android_webview_debug	29.29%	0.15%	0.24%	70.33%	81.82%	18.18%	91.67%	8.33%
M2	android_temp_file	53.86%	1.18%	0.21%	44.74%	96.00%	4.00%	92.86%	7.14%
	android_insecure_file_mode	15.69%	0.82%	0.19%	83.30%	95.31%	4.69%	100%	0%
	android_keystore_protection	22.33%	1.22%	0.17%	76.28%	100%	0%	90.0%	10.0%
M3	android_insecure_ssl	18.37%	1.24%	0.11%	80.28%	100%	0%	100%	0%
	android_webview_ignore_ssl	8.57%	0.77%	0.04%	90.62%	100%	0%	100%	0%
	android_http	90.92%	0.75%	0.13%	8.20%	96.02%	3.98%	100%	0%
M5	android_insecure_random	80.48%	1.52%	0.21%	17.79%	90.51%	9.49%	85%	15%
	android_sha1	68.44%	0.75%	0.24%	30.57%	97%	3%	100%	0%
	android_md5	63.65%	2.27%	0.26%	33.82%	95.25%	4.75%	94.12%	5.88%
	cbc_padding_oracle	33.87%	2.16%	0.06%	63.9%	98.28%	1.72%	100.0%	0%
	android_aes_ecb	11.90%	1.84%	0.13%	86.13%	99.17%	0.83%	77.78%	22.22%
	android_weak_ciphers	9.21%	1.11%	0.04%	89.64%	90.16%	9.84%	100%	0%
M7	android_sql_raw_query	73.6%	0.75%	0.15%	25.5%	98.83%	1.17%	100%	0%
M9	android_hardcoded	80.28%	1.97%	0.09%	17.66%	98.94%	1.06%	91.23%	8.77%

∗: "Both" indicates that both the GP and 3rdPM versions contain vulnerabilities, "3rdPM" indicates that only the 3rdPM version has vulnerabilities, "GP" indicates that only the GP version has vulnerabilities, and "Neither" represents that neither version has vulnerabilities.

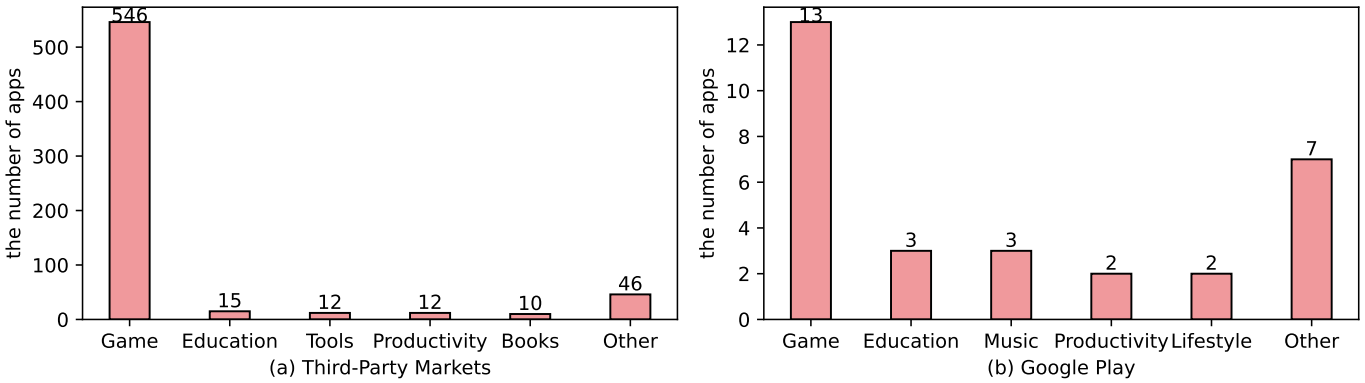

Fig. 4: Category distribution of apps that requested extra permissions in the GP or 3rdPM version.

SDK source. Then, using these mappings, we can convert Content Uris constant into the corresponding Content Uris string. Subsequently, to determine whether the SDK APIs, Intent Actions, and Content Uris used in apps are protected by permissions, we selected the corresponding API level permission mappings based on each app's targetSdkVersion for comparison.

To determine the caller (developer code or TPLs) of SDK APIs, Intent Actions, and Content Uris, we must identify the TPLs used in apps through `Libd` and `LibRadar`. For each 3rdPM version of the app, we extracted lists of extra requested permissions, permissions used by developer code, and permissions used by TPLs. By comparing these three lists, if an app's extra requested permissions are not present in either the permissions list requested by developer code or the permissions list requested by TPLs, we consider permissions to be over-claim, indicating that permissions are requested but not used. The percentage of extra requested permissions in 3rdPM versions of apps introduced by developer code, TPLs, and over-claim, as shown in Figure 5. This figure primarily explains the sources of the extra requested permissions in the 3rdPM versions, which can be attributed to developer code, TPLs, or over-claim. For example, "579" indicates that 579 apps added extra permissions in the 3rdPM version, with only 0 - 10% of these permissions introduced by developer code in each app. Therefore, among 579 apps, most of these extra requested permissions are introduced by TPLs or over-claim.

The CDF (cumulative distribution function) of the usage count of extra requested permissions in the 3rdPM version of apps among developer code, TPLs, and over-claim is illustrated in Figure 6. The CDF also reveals that in the 3rdPM version of apps, the extra requested permissions are seldom introduced by developer code. The distribution of permissions introduced by developer code ranges from 0 to 5. In 89.86% apps, developers introduce 0 permissions. The distribution of permissions introduced by TPLs ranges from 0 to 10. In 73.01% of apps, the number of permissions introduced by TPLs is less than or equal to 5. The distribution of permissions for over-claim from 0 to 21. In 92.67% of apps, the number of over-claim permissions is less than or equal to 11.

87.83% of apps have over-claim permissions. Also, the protectionLevel for most over-claim permissions fall under the normal and dangerous, accounting for approximately 71.42%. Furthermore, a portion of over-claimed permissions with pro-

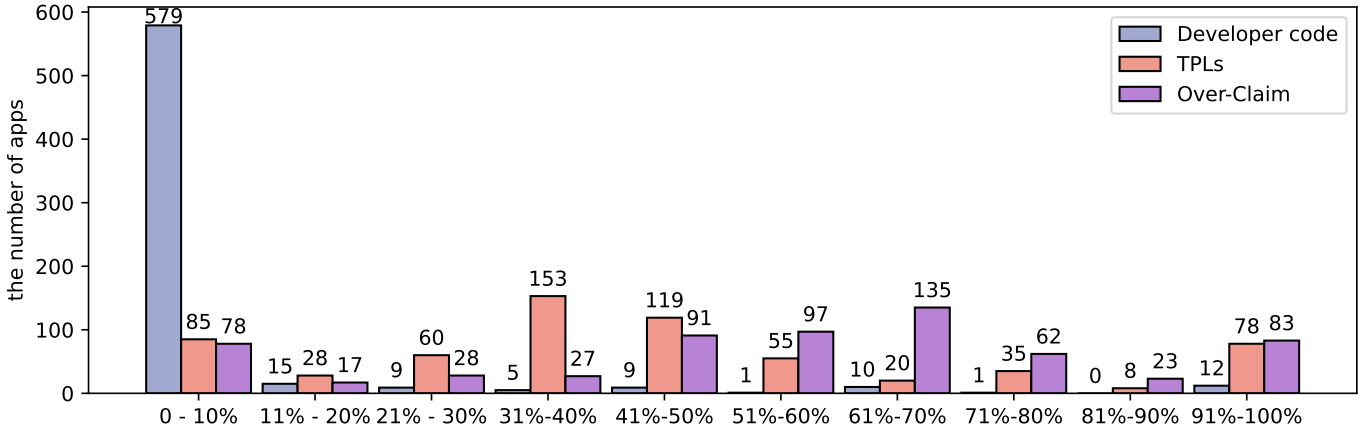

Fig. 5: Percentage of extra requested permissions in 3rdPM versions introduced by developer code, TPLs, and over-claim.

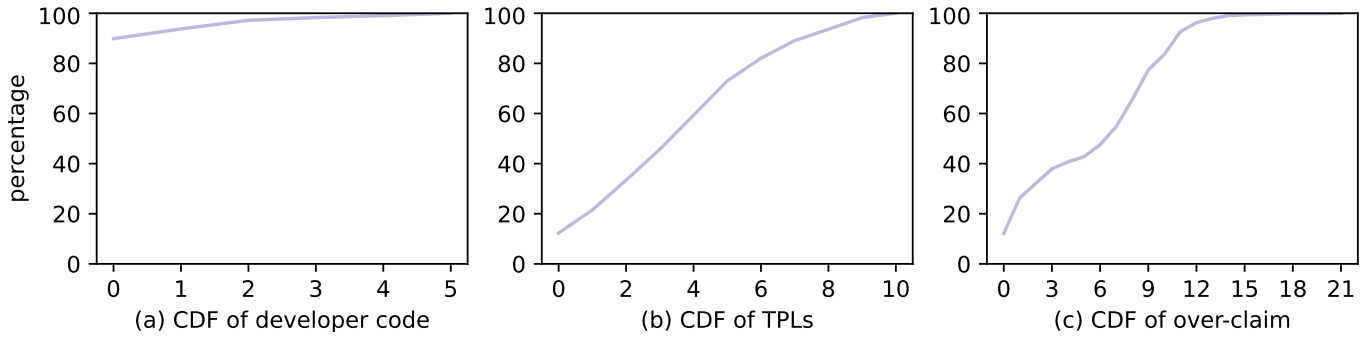

Fig. 6: CDF of the usage count of extra requested permissions in the 3rdPM version of apps.

tectionLevel is categorized as Signature or SignatureOrSystem, accounting for approximately 29.58%. Although many 3rdPM apps request these permissions, they usually cannot obtain such permissions.

Custom Permissions. Among 4,671 app pairs, 1.28% of apps declare custom permissions in the 3rdPM version that are not present in the GP version, averaging 2 extra custom permissions for each app. 0.15% of apps in the GP version declare custom permissions that are not present in the 3rdPM version, averaging 1 extra custom permissions for each app. 0.09% of apps declare unique custom permissions in both the GP and 3rdPM versions. As the prefix of custom permissions typically corresponds to the package name of apps, we have removed the package name prefix of custom permissions. In the 3rdPMs versions of apps, the names and counts of the top 5 extra defined custom permissions are: (`PUSH_PROVIDER`, 24), (`PUSH_WRITE_PROVIDER`, 24), (`PROCESS_PUSH_MSG`, 24), (`TT_PANGOLIN`, 19), and (`C2D_MESSAGE`, 7).

These extra defined custom permissions are required mainly by third-party push services. App developers must declare these custom permissions in the `manifest` file; otherwise, they cannot use these third-party push services. To attract more users and increase the app's exposure, the 3rdPM may encourage developers to use push services to regularly send users notifications, reminders, and promotional information, thereby increasing the app's user activity and retention rate. For example, the 3rdPM version of Italki app[4] uses two push services, `JPUSH` [9] and `C2DM` [3], while the GP version does not use any push service.

V. Discussions

A. Cross-Market App Consistency

Our differential analysis results indicate that the GP and 3rdPM versions of most apps maintain consistent security features, with differences primarily arising from varying review policies across markets. Due to the relaxed review policies of 3rdPMs, developers may integrate more push and advertising libraries into the 3rdPM versions of apps to obtain profit. These TPLs introduce extra permissions and vulnerabilities, making the 3rdPM versions more likely to be flagged as malware. However, since Google Play implements a stricter review process that prohibits the release of such apps, developers have to remove the aforementioned TPLs and permissions when publishing the GP versions. However, a stricter review process also often leads app developers to minimize the protection measures of the GP versions. For example, developers usually avoid packing the GP versions because Google Play might consider packed apps to contain hidden malicious behavior, which could lead to apps being rejected for release.

[4]Package: `com.italki.app`, Version Code: 21256.

B. Best Practices

End-users. The results of RQ2 and RQ3 indicate that the 3rdPM versions of apps are more prone to containing vulnerabilities and excessive permissions. When apps are temporarily unavailable on GP (as GP regularly removes apps that violate its policies), users should prioritize finding alternative apps on GP instead of downloading them from 3rdPMs.

Store Operators. Based on the findings of RQ1, operators of 3rdPMs and GP should refine their review processes to accurately identify whether packed apps contain malicious behavior. For GP operators, it's also crucial to ensure that benign packed apps can be successfully released on GP. Derived from the results of RQ2 and RQ3, 3rdPMs operators need to review whether apps are requesting extra permissions and whether the additionally introduced TPLs contain vulnerabilities, especially for game apps.

App Developers. Firstly, developers should not reduce the protection measures for the GP versions simply because GP has implemented stricter review processes (based on RQ1). Secondly, when developers release apps on 3rdPMs, they need to scrutinize the code they write and the TPLs to avoid introducing additional security vulnerabilities (based on RQ2). Lastly, developers need to maintain consistent security settings between the GP and 3rdPM versions, such as the principle of least privilege, to prevent privacy leaks (based on RQ3).

C. Threats to Validity

Internal Validity. The major threat to the "Sources Analysis" of RQ3 comes from the invocation of Non-SDK APIs within the apps [37]. Calling a small subset of Non-SDK APIs in developer code or TPLs may also require permissions. Ignoring the invocation of Non-SDK APIs could lead to biases in the Sources Analysis. Fortunately, Google has implemented restrictions on access to non-SDK APIs in Android 9.0, which may help reduce threats. In addition, during the differential analysis, various static analysis tools were used, such as `Libd` and `MobSF`. The code behaviors detected by static analysis tools may not necessarily be triggered at runtime, leading to false positives. This is a limitation of all approaches that rely on static analysis. However, when performing comparative analysis on numerous apps, using dynamic analysis presents issues with code coverage and lacks scalability.

External Validity. Given hundreds of Android 3rdPMs, collecting apps from all 3rdPMs is impossible. Performing a differential analysis on apps with the same version code across different 3rdPMs and GP might show minor differences. To mitigate potential biases, we collected apps from as many 3rdPMs as possible and selected ten popular 3rdPMs. Therefore, our findings can represent the overall differences between the 3rdPM and GP versions of apps.

VI. Related Work

Many works have focused on measuring the behavior of apps and developers in the app markets. The most relevant work was conducted by Wang et al. [36]. They compared 16 Chinese app markets and Google Play, revealing differences in developer behavior across these app markets, including aspects like code maintenance and the use of third-party services. Viennot et al. [33] used PlayDrone to download over 1.1 million Android apps from Google Play. They investigated four valuable questions, demonstrating that PlayDrone can help improve the quality of app content in Google Play. Chen et al. [24] proposed AR-Miner to mine and analyze user reviews from mobile app markets, which helps developers quickly identify valuable feedback from a large number of comments. Linares-Vásquez et al. [32] found that in 7,097 free apps from Google Play, heavy use of fault- and change-prone APIs could negatively impact app success. Hu et al. [27] developed CHAMP, a tool using text mining and NLP to detect policy-violating behaviors from user reviews in app markets. CHAMP efficiently uncovers non-compliance, offering violation scores and key comments for apps. Cai et al. [22] conducted a large-scale, longitudinal study on 62,894 benign apps developed over the past eight years to analyze the symptoms and causes of compatibility issues in the Android ecosystem. Zhou et al. [39] developed the DiehardDetector, finding that 21% of 80K apps from Google Play used diehard methods. Liu et al. [30] introduced DAPANDA to automatically detect aggressive push notifications in Android apps, identifying over 1,000 aggressive notifications across 20,000 apps from eight app markets.

The deployment of app protection and the detection of malicious behavior are also noticed by security researchers. Berlato et al. [21] analyzed 38,323 apps on Google Play and found that 59% of these apps neither implemented anti-debugging nor anti-tampering protections. Dong et al. [26] proposed the PackDiff, which through dynamic analysis exposed issues with commercial app packers, revealing that most introduced unnecessary sensitive data access and performance issues. Chen et al. [23] measured method similarities within apps using the centroid characteristics of dependency graphs, efficiently and precisely detecting clones in over 150,000 apps across five Android markets. Li et al. [28] developed IccTA to detect data leaks between components in Android apps, uncovering a significant number of privacy violations across 108 apps from MalGenome and 15,000 apps from Google Play. Wang et al. [35] conducted a large-scale systematic measurement study on four types of app signing issues. The results revealed that in the 25 markets studied, 7% to 45% of the apps contained at least one signing issue. Wang et al. [34] proposed WuKong, a two-phase app clone detection method that first identifies suspicious apps by comparing lightweight static semantic features. Then, it conducts a fine-grained comparison of more detailed features for apps identified in the first phase. Experiments on over 100,000 Android apps from five markets demonstrated the method's effectiveness and scalability. Dong et al. [25] created FraudDroid to detect mobile ad fraud. By analyzing 12,000 apps from eight app markets, FraudDroid confirmed and shared 335 cases of ad fraud associated with ad networks.

VII. Conclusion

Even if Google Play and third-party market versions of apps have the same version code, they might have differences in implementations. In this work, we systematically analyzed the differences in security and privacy between apps with the same version code from three points of view, including app protection, security threats, and permission usage. A series of captivating and valuable insights have been revealed, which can help developers and store operators provide improvement directions and guidance, enhancing the quality of the Android apps ecosystem and user experience.

Acknowledgements

This work was partially supported by Taishan Young Scholar Program of Shandong Province, China (Grant No. tsqn202211001), Shandong Provincial Natural Science Foundation (Grant No. ZR2023MF043), Xiaomi Young Talents Program, and Australian Research Council Discovery Projects (DP240103068).

References

[1] (2023) Androbugs. [Online]. Available: https://github.com/AndroBugs/AndroBugs_Framework
[2] (2023) Androguard. [Online]. Available: https://github.com/androguard/androguard
[3] (2023) Android cloud to device messaging (c2dm). [Online]. Available: https://www.vogella.com/tutorials/GoogleCloudMessaging/article.html
[4] (2023) Apkid. [Online]. Available: https://github.com/rednaga/APKiD
[5] (2023) Axplorer. [Online]. Available: https://github.com/reddr/axplorer
[6] (2023) Declare permissions for your app. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/9214102
[7] (2023) Google-play-scraper. [Online]. Available: https://github.com/JoMingyu/google-play-scraper
[8] (2023) Intellectual property. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/9888072
[9] (2023) Jpush. [Online]. Available: https://www.jiguang.cn/en/push
[10] (2023) Mobsf. [Online]. Available: https://github.com/MobSF/Mobile-Security-Framework-MobSF
[11] (2023) Owasp mobile top 10. [Online]. Available: https://owasp.org/www-project-mobile-top-10/
[12] (2023) Permissions on android. [Online]. Available: https://developer.android.com/guide/topics/permissions/overview
[13] (2023) Pinduoduo app report. [Online]. Available: https://www.lookout.com/documents/threat-reports/us/lookout-tg-pinduoduo.pdf
[14] (2023) Pinduoduo app website. [Online]. Available: https://en.pinduoduo.com/
[15] (2023) Raccoon. [Online]. Available: https://raccoon.onyxbits.de/
[16] (2023) Sign your app. [Online]. Available: https://developer.android.com/studio/publish/app-signing
[17] (2023) Virustotal. [Online]. Available: https://www.virustotal.com/gui/home/upload
[18] (2023) Yara rules. [Online]. Available: https://yara.readthedocs.io/en/stable/index.html
[19] (2024) Developer policy center. [Online]. Available: https://play.google/developer-content-policy/
[20] (2024) Privacy statement. [Online]. Available: https://raccoon.onyxbits.de/privacy/
[21] S. Berlato and M. Ceccato, "A Large-Scale Study on the Adoption of Anti-Debugging and Anti-Tampering Protections in Android Apps," *Journal of Information Security and Applications*, 2020.
[22] H. Cai, Z. Zhang, L. Li, and X. Fu, "A Large-Scale Study of Application Incompatibilities in Android," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Beijing, China, July 15-19, 2019*, 2019.
[23] K. Chen, P. Liu, and Y. Zhang, "Achieving Accuracy and Scalability Simultaneously in Detecting Application Clones on Android Markets," in *Proceedings of the 36th International Conference on Software Engineering (ICSE), Hyderabad, India, May 31 - June 07, 2014*, 2014.
[24] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, "AR-Miner: Mining Informative Reviews for Developer from Mobile App Marketplace," in *Proceedings of the 36th International Conference on Software Engineering (ICSE), Hyderabad, India, May 31 - June 07, 2014*, 2014.
[25] F. Dong, H. Wang, L. Li, Y. Guo, T. F. Bissyandé, T. Liu, G. Xu, and J. Klein, "FraudDroid: Automated Ad Fraud Detection for Android Apps," in *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Lake Buena Vista, FL, USA, November 04-09, 2018*, 2018.
[26] Z. Dong, H. Liu, L. Wang, X. Luo, Y. Guo, G. Xu, X. Xiao, and H. Wang, "What Did You Pack in My App? A Systematic Analysis of Commercial Android Packers," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Singapore, Singapore, November 14-18, 2022*, 2022.
[27] Y. Hu, H. Wang, T. Ji, X. Xiao, X. Luo, P. Gao, and Y. Guo, "CHAMP: Characterizing Undesired App Behaviors from User Comments based on Market Policies," in *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE), Madrid, Spain, 22-30 May 2021*, 2021.
[28] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. L. Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. D. McDaniel, "IccTA: Detecting Inter-Component Privacy Leaks in Android Apps," in *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE), Florence, Italy, May 16-24, 2015*, 2015.
[29] M. Li, W. Wang, P. Wang, S. Wang, D. Wu, J. Liu, R. Xue, and W. Huo, "LibD: Scalable and Precise Third-Party Library Detection in Android Markets," in *Proceedings of the 39th International Conference on Software Engineering (ICSE), Buenos Aires, Argentina, May 20-28, 2017*, 2017.
[30] T. Liu, H. Wang, L. Li, G. Bai, Y. Guo, and G. Xu, "DaPanda: Detecting Aggressive Push Notifications in Android Apps," in *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, November 11-15, 2019*, 2019.
[31] 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), Austin, TX, USA, May 14-22, 2016*, 2016.
[32] M. L. Vásquez, G. Bavota, C. Bernal-Cárdenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk, "API Change and Fault Proneness: A Threat to the Success of Android Apps," in *Proceedings of the 9th ACM joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), Saint Petersburg, Russian Federation, August 18-26, 2013*, 2013.
[33] N. Viennot, E. Garcia, and J. Nieh, "A Measurement Study of Google Play," in *Proceedings of the 34th International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Austin, TX, USA, June 16-20, 2014*, 2014.
[34] H. Wang, Y. Guo, Z. Ma, and X. Chen, "WuKong: A Scalable and Accurate Two-Phase Approach to Android App Clone Detection," in *Proceedings of the 24th International Symposium on Software Testing and Analysis (ISSTA), Baltimore, MD, USA, July 12-17, 2015*, 2015.
[35] H. Wang, H. Liu, X. Xiao, G. Meng, and Y. Guo, "Characterizing Android App Signing Issues," in *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, November 11-15, 2019*, 2019.
[36] H. Wang, Z. Liu, J. Liang, N. Vallina-Rodriguez, Y. Guo, L. Li, J. Tapiador, J. Cao, and G. Xu, "Beyond Google Play: A Large-Scale Comparative Study of Chinese Android App Markets," in *Proceedings of the 18th Internet Measurement Conference (IMC), Boston, MA, USA, October 31 - November 02, 2018*, 2018.
[37] S. Yang, R. Li, J. Chen, W. Diao, and S. Guo, "Demystifying Android Non-SDK APIs: Measurement and Understanding," in *Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering (ICSE), Pittsburgh, PA, USA, May 25-27, 2022*, 2022.
[38] Y. Zhang, X. Luo, and H. Yin, "DexHunter: Toward Extracting Hidden Code from Packed Android Applications," in *Proceedings of 20th European Symposium on Research in Computer Security (ESORICS), Vienna, Austria, September 21-25, 2015*, 2015.
[39] H. Zhou, H. Wang, Y. Zhou, X. Luo, Y. Tang, L. Xue, and T. Wang, "Demystifying Diehard Android Apps," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, September 21-25, 2020*, 2020.