Measuring and Characterizing (Mis)compliance of the Android Permission System

Anna Barzolevskaia, Enrico Branca, and Natalia Stakhanova

Abstract—**Within the Android mobile operating system, Android permissions act as a system of safeguards designed to restrict access to potentially sensitive data and privileged components. Multiple research studies indicate flaws and limitations of the Android permission system, prompting Google to implement a more regulated and fine-grained permission model. This newly-introduced complexity creates confusion for developers leading to incorrect permissions and a significant risk to users security and privacy. We present a systematic study of theoretical and practical misuse of permissions. For this analysis we derive the unified permissions and call mappings that represent theoretical requirements of permissions and calls. We develop PChecker, an approach that identifies the discrepancies between the official Android permissions documentation and permission implementation in the Android platform source code based on these mappings. We evaluate four versions of the Android Open Source Project code (major versions 10–13) and shed light on the prevalence of discrepancies between the official Android guidelines for permissions and their implementation in the Android platform source code. We further show that these discrepancies result in miscompliance in third-party Android apps.**

Index Terms—**Android, documentation, app, permissions, non-SDK restriction lists, security.**

I. Introduction

MOBILE phones have transformed how people communicate and share information. With ubiquitous flexibility, Android has become one of the most widely used mobile operating systems (OS) in the world. The convenient and extensive access to phone resources adopted by Android has quickly revealed inefficiencies of the existing protections. Among them, Android permissions is a fundamental system of controls designed to restrict an application's access to potentially sensitive data and privileged components.

Numerous studies pointed out limitations of the Android permission system [1], [2], [3], [4], [5], [6]. For example, earlier Android versions did not regulate the use of privileges, allowing any app to declare permissions and gain access to any data or functionality of the device. This negligence has resulted in privacy leaks [7], unrestricted access to advanced functionalities [8], [9], overprivileged apps [10] and exploitable weaknesses [11]. Allowing users to grant or reject a permission requested by an app has not rectified the security problems, as studies showed, users still had a vague understanding of what permissions should or should not be granted to applications (apps) [12].

Android permission system have since evolved significantly to a more regulated and fine-grained permission model. With Android 9, the restrictions on non-SDK interfaces were introduced, regulating access to parts of the Android platform for applications and services. The subsequent versions expanded these restrictions further covering official SDK interfaces and non-SDK interfaces, i.e., internal, unstable, test, and temporary interfaces.

Despite a more sophisticated permission system, documentation remained a low priority for Android developers. In fact, the official documentation for Android permissions, including their functionality, requirements, and use cases, has been historically lacking (e.g., [10]), with the permissions' difference between versions largely undefined or vaguely stated. This lack of information leads to ambiguity, which creates the potential for permission misuse and improper implementation. Indeed, recent studies show that even high risk permissions continue to be automatically granted to apps [13].

In this study, we analyze the state of Android permissions defined by the Android platform and their application in practice by answering the following research questions:

- RQ1: What is the overall state of Android permission documentation?
- RQ2: What type of inconsistencies exist in the official Android guidelines for permissions?
- RQ3: What type of inconsistencies exist in the protections provided by the Android permission system for guarded interfaces in Android platform code?
- RQ4: Do the discovered inconsistencies appear in practice in third-party Android apps?

To facilitate this analysis, we develop PChecker, an approach that identifies the discrepancies between the official Android guidelines for permissions and their implementation in the Android platform source code. To ensure consistent and comprehensive analysis, we construct the unified permission and guarded call mappings that identify theoretical requirements of permissions and calls. To create these mappings, we analyze the source code of four most recent Android versions, 10-13 (APIs 29, 30, 31, 33), extracting permissions and their properties as

Manuscript received 13 June 2023; revised 29 December 2023; accepted 26 January 2024. Date of publication 12 February 2024; date of current version 19 April 2024. Recommended for acceptance by L. Mariani. *(Corresponding author: Natalia Stakhanova.)*

The authors are with the Department of Computer Science, University of Saskatchewan, Saskatoon, S7N 5C9, Canada (e-mail: jhu168@mail.usask.ca; enrico.branca@usask.ca; natalia@cs.usask.ca).

Digital Object Identifier 10.1109/TSE.2024.3362921

0098-5589 © 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

noted by the Android developers, as well as methods guarded by permissions. We complement this information with the official Android documentation and restriction information for the Android platform. The resulting unified permission and guarded call mappings simplify analysis of protections provided by the Android permission system and reveal existing inconsistencies between permissions and calls.

To further understand the relevance and scope of the discovered inconsistencies in practice, we explore whether these discrepancies appear in the apps produced by the third-party Android developers. We employ PChecker to review the permissions requested by and granted to Android apps.

In summary, this work presents three contributions:

- *We develop PChecker, an approach for detection of the permissions-related discrepancies between the official Android guidelines and their implementation in the Android platform source code*. Our method offers a concrete guidance to domain experts to address current issues and assists third-party developers in creating fully compliant apps. PChecker can accurately identify the problem's location and provide a meaningful explanation for a discrepancy.
- *We derive a unified permission mapping that enables domain experts to evaluate the consistency of protections provided by the Android permission system.* We identify 13 types and 1,388 cases of contradictions between the official documentation and the Android platform source code, most are minor inconsistencies, but some concern policy clashes and severely outdated documentation. This is the first work to systematically outline the existing categorization and to reveal the presence, scope and types of discrepancies between the Android documentation and its permission system implementation across four different versions.
- *We derive a unified guarded call mapping* that, together with the unified permission mapping, reveals inconsistencies in protections declared by the Android platform. We identified 3,362 discrepancies between the SDK (and non-SDK) interface restrictions and the corresponding permission restrictions across the analyzed Android versions.
- *We use PChecker to evaluate permissions requested and granted to third-party Android apps*. We conduct an analysis of 3,681 apps, discovering 4,056 instances of discrepancies related to permissions requested by apps, and another 4,092 discrepancies with permissions automatically granted to apps. Overall, we found at least one of these issues in each of 584 apps.

Our findings highlight alarming patterns and shed light on the existing misleading and contradicting guidance given to Android app developers. The pervasiveness of permission non-compliance in Android third-party apps emphasizes the challenges for creating fully permission-compliant apps. PChecker and the unified permission and call mappings are publicly available[1]. We are in the process of reporting the discovered inconsistencies and conflicts to Android.

[1]https://cyberlab.usask.ca/pchecker.html

The paper is structured as follows. In Section II, we provide necessary background on permissions and how they are used in Android platform. In Section III, we describe our methodology for generating unified mappings, inconsistency lists and PChecker. Sections IV and VI discuss the results the unified permission and guarded call mapping results. In Section VII, we report the results produced by PChecker for our testing app. Section VII-C presents the findings of the third-party apps analysis. We offer a discussion of our findings in Section VIII, present related work in Section IX and finally summarize our study in Section XI.

II. Background

Android permissions is a system of safeguards implemented as an Access Control List that is designed to restrict access to potentially sensitive data and privileged components. If an application requires restricted device functionality to operate, the corresponding permissions should be declared in the app's manifest file for the host OS to attempt to grant these permissions to the app. Android 1–7 did not regulate the use of privileges: any app could declare any permission and gain access to any data or functionality of the device[2]. Android 8 introduced the allowlist [14], where privileged/system apps (mainly OEMs — original equipment manufacturers) had to declare permissions in the system configuration XML files. With API 29, the permission system was changed and restrictions on non-SDK interfaces were introduced, regulating access to parts of the Android platform for applications and services [15]. These restrictions took the form of restriction lists. With Android 10, this system was expanded, covering all official (SDK interfaces) and internal/unstable/test/temporary interfaces (non-SDK interfaces), including permissions, with restricted permissions detailed in both manifest and restriction-lists files.

Permission categorization: Historically, Android differentiated permissions with respect to the necessary protection level as `normal` permissions, i.e., permissions to resources with minimal risk, `dangerous` permissions that are associated with elevated risk (e.g., access to personal data), `signature` permissions which are granted to apps from the same developer, and `SignatureOrSystem` (previously `signature|privileged`) that regulate access to privileged resources and are allocated for apps installed in the Android system image [10].

The Android permission system has significantly evolved over the years. At first, permissions were only granted during app installation time. Starting from Android 6, dangerous permissions were granted at runtime of the application and starting with Android 9 (API level 28), Android has further restricted the use of permissions tied to elevated risks.

Currently, Android official documentation groups permissions as follows [16]:

- Install-time permissions that are granted to the application when it is installed, these include permissions for resources that do not expose sensitive data or critical functions.

[2]https://web.archive.org/web/20130822155513/https://developer.android.com/training/articles/security-tips.html

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

- Runtime permissions allow access to restricted data and functions, such as calls, notifications, GPS location, etc. These require user's direct approval to be granted. These permissions are requested at the runtime of the application.
- Special permissions allowed for use only by the Android platform and OEMs.

These Android-defined permissions facilitate access to system resources for developers, and thus referred to as system-defined permissions. Beyond these, Android allows developers to include application-defined permissions known as custom permissions. Custom permissions allow apps to share their functionality with other apps, including those signed with a trusted certificate, which are normally isolated from each other [17]. In this work, we focus specifically on system-defined permissions.

Naming convention: Android permissions must conform to a naming convention. A permission should be prefixed with an app's package name (or `android` for Android-defined permissions), with reverse-domain-style naming. The prefix should be followed by `.permission.`, and then a description of the capability that the permission represents, in `UPPER_SNAKE_CASE` [18]. For example, `android.permission.SYSTEM_CAMERA`. Third-party developers are expected to declare a list of required permissions in the AndroidManifest.xml file. The corresponding APIs that govern access are then invoked by the app if the necessary permissions are granted.

III. PCHECKER

In the core of the PChecker approach are the unified permissions and guarded calls mappings. The close inspection of the available Android documentation and Android API source code suggested that the official documentation describing Android permissions and interfaces available for the third-party developers is limited and often contradicting. We therefore construct the unified mappings and empirically determined the available Android system permissions and interfaces leveraging several sources: Android source code available for several API versions, official documentation, and Google restrictions lists. For construction of mappings, we selected the four latest(at the time of the work) Android Open Source Project (AOSP) snapshots for Android versions 10–13. We extracted the code of the main branch for each API, i.e., android10-s3-release (API 29), android11-s1-release (API 30), android12-s5-release (API 31), android13-s3-release (API 33). Note, that the version 12.1 (API 32) is treated similarly to version 12 (API 31) as Google provides restriction lists for major releases only, hence it appears that restrictions imposed for API 31 remain unchanged for API 32. The flow of this analysis is shown in Fig. 1.

A. Building the Unified Permission Mapping

Since third-party developers are expected to declare a list of required permissions in the AndroidManifest file, we parsed the Android platform source code and extracted permissions from the system's Android Manifest files. We identified and extracted string literals following the permission naming convention and

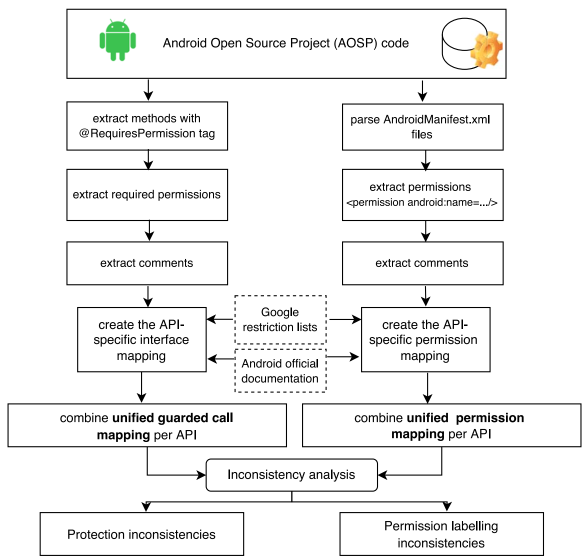

Fig. 1. Constructing the unified mappings.

comments present in the source code following the XML file structure shown in Fig. 4.

The collected permissions were then mapped to permission categories. As a starting point, we used the categorization present in the official documentation and in the Android non-SDK restriction lists published by Google, which describe the general availability of elements (an interface or a variable defined in the AOSP source code) in the Android OS [15]. We also compared these published lists with the restriction lists included in each AOSP release and generated automatically during code compilation. We confirmed that these generated lists represents only a subset of the officially posted non-SDK restriction lists. This mapping step produced a theoretical categorization of permissions from an official documentation perspective available as a guide to third-party developers.

The collected comments present in the Android platform source code and the public Android documentation were used to derive attributes and their categories further described in Section IV-A. For this categorization, we grouped all annotations, labels, and explanation present in code. We further manually supplemented these groups with information from the public documentation if it was available. In some cases, when the information was unavailable, we deduced it by manually examining relevant documentation or code. For example, when details about the initial introduction of a permission were not explicitly stated in the documentation, we determined the 'Introduced' attribute value by cross-referencing public documentation across adjacent Android versions. Once these derived attributes and their categories were finalized, we labelled permissions present in the mapped theoretical categorization using the derived attributes. This process was done automatically based on the permission names.

The derived categorization system together with the corresponding permissions present in each API formed the API permission mappings. We combined them to comprise the *unified permission mapping*. This unified mapping represents a

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

```
@RequiresPermission(android.Manifest.permission.NFC_TRANSACTION_EVENT)
    @SdkConstant(SdkConstantType.BROADCAST_INTENT_ACTION)
    public static final String ACTION_TRANSACTION_DETECTED =
            "DETECTED";
```

Fig. 2. An example of a variable protected by `@RequiresPermission` tag.

theoretical view of permissions and the protections they intend to provide across Android API versions. An example of a unified permission mapping entry is shown in Fig. 5.

B. Building the Unified Guarded Call Mapping

Android permissions guard access to the SDK and non-SDK interfaces available within the Android platform. To further understand how the theoretical permission protections are applied by the Android platform, we leveraged static analysis to inspect requirements for the SDK and non-SDK interfaces protected by permissions. In our analysis, we relied on the official documentation and the restriction lists featuring calls to the SDK and non-SDK interfaces with their corresponding assignment to the restriction categories.

Specifically, we extracted all elements present in the restriction lists. We then automatically examined four versions of the Android platform source code (API levels 29, 30, 31, 33) in a linear sweep fashion and retrieved elements that were accompanied by the `@RequiresPermission` tag [19]. According to documentation, this tag indicates that the following element requires permissions. An example of this tag usage is presented in Fig. 2. There are three possible scenarios: single permission is needed, multiple permissions are required (specified by the *allOf* field), or a list of permissions is necessary, where any of them can be used to invoke a method (indicated by the *anyOf* field). For each `@RequiresPermission` tag found, we collected the required permissions, the name of the element, the types of parameters (for methods), and the path to the file containing it.

Having the source code elements that require permissions, we had to find their corresponding interfaces in the restriction lists. In order to attribute an interface to an element, we sorted interfaces by their relative paths. Then, for each retrieved source code element, we pulled interfaces, the relative paths of which were present at the end of the element's path. We iterated through the interfaces and compared the values of parent class, element name and input parameter types. For example, an element section of the restriction list pseudo-line "someMethod(I[IZLjava/lang/String;)I" means that the method expects parameters *int*, *int array*, *boolean*, *String* in that order, and the method returns a single *int* value. Each **L** indicates a following non-primitive data reference also present in the restriction lists file, each of them ending with ";" and starting with one of the following: **Landroid**, **Ljavax**, **Ljava**, **Lcom**, **Lorg**, **Llibcore**, **Lsun**, **Ldalvik**, **Ljdk**. As for primitive data types, we inferred their representations as follows: **I** - *int*, **J** - *long*, **Z** - *boolean*, **F** - *float*, **B** - *byte*, **C** - *char*, **D** - *double*, **S** - *short*, **V** - *void*. A letter-code preceding by a **[**(square bracket) indicates that an array of the following data types is expected.

We matched the collected interfaces and methods extracted from the code by comparing the invocation path found in the restriction list with the path collected during code analysis, the values of the parent class, the element name, and the input parameter types. This allowed us to transitively connect permission requirements to interface calls.

The matched methods were further labelled with the corresponding restriction categories. During this parsing, we annotated the collected methods with their alternatives (where present) that developers introduced over the versions to gradually replace blocked functionality with safer public alternatives. These were obtained from the changes to restriction lists published by Google [20], [21], [22]. This final mapping formed the *unified guarded call mapping*. An example of a unified permission mapping entry is shown in Fig. 6.

C. Inconsistency Analysis

The inconsistency analysis is the final step that correlates the unified permission and call mappings to reveal discrepancies between the official Android guidelines for permissions and their implementation in the Android platform source code. Since permissions evolve over time, this analysis is conducted for each API. For each permission present in the unified mapping, we automatically match the assigned categories found in the documentation with the permission attributes found in the source code. The instances where this information does not align are then grouped and manually analyzed to verify discrepancy. The confirmed conflicts are manually labelled with the corresponding conflict label. These types of discrepancies represent situations where official documentation is conflicting with the source code. We refer to the list of identified discrepancies as *permission labelling inconsistencies*. These are theoretical contradictions. An example of a labelling inconsistency is shown in Fig. 5. Permission `OVERRIDE_WIFI_CONFIG` in Android 13 (API 33) was moved to the *public-api* category, however, in the source code it is accompanied by the plain-text comment "*not for use by third-party applications*".

We conduct a similar analysis for guarded call mappings to reveal inconsistencies in the protections expected to be provided by the Android platform. From Android 9 (API 28), Android started introducing restrictions to non-SDK interfaces, gradually removing the non-SDK APIs from the official documentation. Google further stated that only the SDK API packages listed in the official documentation [23] are open to third-party developers [24]. Our analysis aims to explore how consistency of protections is enforced for both non-SDK and SDK interfaces.

To analyze relationships between calls and permissions, we compared restriction categories of permissions and the elements they are guarding. Intuitively, we expected the level of restrictions placed on elements and the corresponding permissions to be equivalent, i.e., elements considered to be high-risk should be guarded by the corresponding permissions with a high level of risk. Similar to permissions, we automatically identify instances contradicting this assumption, i.e., where these restrictions do not align. These cases are grouped

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

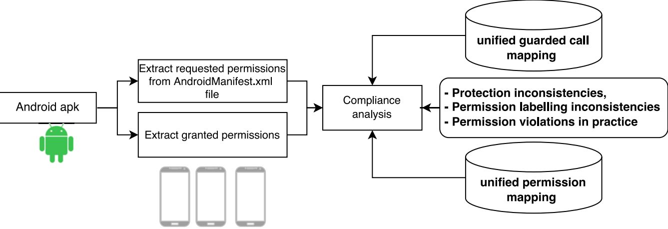

Fig. 3. The flow of app analysis.

and labelled manually. We refer to these as *protection inconsistencies*. An example of this inconsistency is given in Fig. 6. In Android 13 (API 33), a method "canAuthenticate" is *public*, i.e., can be assessed through public SDK by third-party developers. It requires however a *blacklist* permission `USE_BIOMETRIC_INTERNAL`.

Both permission labelling and protection inconsistencies are theoretical contradictions.

D. App Analysis

The consistency analysis of the unified mappings reveals theoretical labelling conflicts which primarily represent contradiction within the Android platform and its documentation. To understand whether these inconsistencies are present in practice, we analyzed third-party Android apps using the flow presented in Fig. 3.

The process starts by retrieving all app permissions declared in AndroidManifest file. For each permission, PChecker extracts permission information from the unified permission mappings according to the Android app target version designated by `targetSdkVersion`. The analysis is performed in two stages: (1) theoretical permission compliance that examines requested permissions and whether these permissions can be requested by third-party apps according to their corresponding restrictions indicated in unified mappings, and (2) a practical compliance check that examines whether the permissions requested by an app and those that are granted to it in practice comply with the unified mappings.

More specifically, given an APK file, PChecker records the supporting metadata using AAPT2 *dump badging* command, which extracts information from the APK's Manifest file [25]. For each of the requested permissions, the corresponding permission restrictions are retrieved from the unified permission mapping and compared against the list of permission labelling inconsistencies.

For the compliance check, the app is installed on an Android device using *Android Debug Bridge (adb)*, a command-line tool that enabled us to issue actions on devices [26]. The app itself can be either pointed to as a file on the host machine or downloaded directly from the market (e.g., Google Play) by specifying the package name. For example, the Google Play store, page is opened with *adb shell "am start market:details?id=<package name>"*, then a press of the "Install" button is simulated with *adb shell "input tap X Y"*, where X and Y are the coordinates provided for each device. Although Google Play market currently favours AABs (Android App Bundles) format, apps downloaded on the device are stored as APK files and third-party marketplaces still use the APK format. Hence, our analysis is focused on APK files.

The requested and granted permissions are obtained using the *adb shell "dumpsys package"*, once after installation and once after a simulated with the *adb monkey* command pseudo-run [27]. This pseudo-run is performed in order to ensure that the app can be launched. The parsed permissions are mapped to the unified permission mapping according to the app's `targetSdkVersion`, and compared against the information contained in the mapping. The instances where mapping indicates restrictions for third-party apps are automatically flagged and grouped based on the mapping information. These instances are manually analyzed and labelled as permission violations in practice. These issues should be viewed separately from the theoretical permission labelling conflicts, as these issues represent how permissions are actually handled on Android devices.

Since many APKs lack one or more `SdkVersion` boundaries, we only consider the `targetSdkVersion` value as the targeted API level. When the app's `targetSdkVersion` is not specified by developers in the app's Manifest, we use the `targetSdkVersion` detected by the OS and returned by the adb dump.

E. Experimental Setup

PChecker was implemented as a fully automated stand alone analyzer written in Python programming language. For our evaluation of permission violations in practice, we used the following devices:

- KingPad SA8 (Android 10, API level 29);
- Umidigi A9 Pro (Android 11, API level 30);
- Samsung Galaxy A22 5G (Android 11, API level 30);
- Samsung Galaxy S21 FE 5G (Android 12, API level 31);
- Samsung Galaxy A22 5G (Android 13, API level 33);
- Google Pixel 7 Pro (Android 13, API level 33).

IV. RQ1: The State of Android Permission Documentation

To characterize the state of Android permission documentation, we build the unified permission mapping for four analyzed APIs. This process produced 765 individual permissions, including 533 for API 29, 590 for API 30, 689 for API 31/32, and 765 for API 33. Out of 765 permissions found in API 33 source code, 206 of them were documented in the official Android documentation that covers the latest version [28].

A. Retrieved Permission Categories

In our analysis, we classify permissions by the following attributes: **Restriction**, **Tag**, **Introduced**, **Deprecated**, **Protection**, **Type**, **Status** and **Usage**. An example of the source of these attributes is shown in Fig. 4.

Restriction. Starting from Android 9 (API 28), Android began restricting which non-SDK API interfaces third-party

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

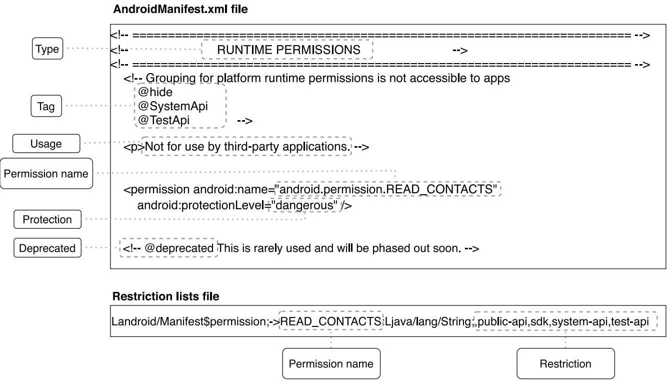

Fig. 4. An example of Android permission annotations in AndroidManifest.xml file and Android restrictions lists.

```
▼ android.permission.OVERRIDE_WIFI_CONFIG {4}
  ▼ seen_in_manifests [4]
      0 : 29
      1 : 30
      2 : 31
      3 : 33
  ▼ seen_in_lists [4]
      0 : 29
      1 : 30
      2 : 31
      3 : 33
  ▼ versions {4}
    ▶ 29  {10}
    ▶ 30  {10}
    ▶ 31  {10}
    ▼ 33  {10}
        tag : sdk
        restriction_list : public-api,sdk,system-api,test-api
        f_group : Permissions for accessing networks
        deprecated :☐ false
        status : relevant
        protection_level : signature|privileged|knownSigner
        usage : not_by_third_party_apps
        type : INSTALL PERMISSIONS
        restriction : public
        protection : signature
  introduced : 33
```

Fig. 5. An example of a unified mapping entry for a permission.

apps may use. The restrictions are enforced in the ART of the application processes. These non-SDK interfaces are neither documented in the Android framework nor stable. The restrictions lists have since been published for each major version of Android [15]. Each restriction list includes permissions with the corresponding interfaces and restriction categories. Officially, the restriction lists include the following categories [15]:

- *Blocklist (blacklist)* — interfaces inaccessible to third-party developers;
- *Conditionally blocked max-x (greylist-max-x)* — interfaces usable by apps targeting an API level up until level x, inaccessible for others above that level. For example, permissions under the category conditionally blocked max-r should only be used by apps targeting Android 11(R) or lower versions, as they will consequently be blocked for target version 12(S) or higher;

```
▼ Landroid/hardware/biometrics/BiometricManager;->canAuthenticate()I {3}
  ▼ seen_in_source [4]
      0 : 10
      1 : 11
      2 : 12
      3 : 13
  ▼ seen_in_lists [4]
      0 : 10
      1 : 11
      2 : 12
      3 : 13
  ▼ versions {4}
    ▶ 10  {9}
    ▶ 11  {9}
    ▶ 12  {9}
    ▼ 13  {9}
        version : 13
        invocation : /core/java/android/hardware/biometrics
                     /BiometricManager#canAuthenticate()#fb75af660023cf3757f45450c2f359a72803dd52
      ▼ perms {3}
        ▼ basic {3}
          ▼ perms [1]
              0 : USE_BIOMETRIC_INTERNAL
          ▼ params [0]
              (empty array)
          ▼ req [1]
              0 : null
        ▶ read {3}
        ▶ write {3}
        rl  : public-api,sdk,system-api,test-api
        restr : public
        restr_line : Landroid/hardware/biometrics/BiometricManager;->canAuthenticate()I,public-
                     api,sdk,system-api,test-api
        perms_changed :☐ false
        restrs_changed :☐ false
        alt : null
```

Fig. 6. An example of a unified guarded call entry.

- *Unsupported (greylist)* — unrestricted at the time of publishing, but are not included in the documentation, and therefore subject to changes without notice;
- *SDK (whitelist)* — supported and documented interfaces.

Our analysis revealed a few additional restriction lists' categories for APIs 29–33 including:

- *public-api* — officially supported SDK interfaces documented in the package index [28];
- *sdk* — SDK interfaces, the available documentation does not explain how these differ from other groups, replaced whitelist starting at API 31;
- *system-api* — an unclear label, appears only in conjunction with "public-api" and "sdk";
- *test-api* — appears to indicate a permission used in internal testing;
- *max-target-x* — appears to indicate Conditionally blocked max-x interfaces, replaced greylist-max-x starting at API 31;
- *blocked* — appears to indicate blacklisted interfaces, replaced blacklist starting at API 31;
- *lo-prio* - an unclear label that is always combined with max-target-x, appeared at API 31;
- *unsupported* — appears to indicate greylisted interfaces, replaced greylist starting at API 31;
- *removed* — appears to indicate to-be-removed permissions, it is always combined with unsupported, first introduced in API 31.

Given a variety of undocumented labels in the non-SDK restriction lists, we have grouped them into categories shown in Table I to facilitate the analysis.

- the *public* category is characterized by the "public-api" list and includes "public-api,system-api,test-api,whitelist" and "public-api,sdk,system-api,test-api". We assume that

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COUNTS OF RESTRICTION CATEGORIES FOR PERMISSIONS

Category	A10(Q)	A11(R)	A12(S)	A13(T)
public	158	166	183	206
sdk	205	241	303	357
blacklist	26	47	69	75
conditional block	139	131	133	127
max O (A8)	*139*	*131*	*130*	*124*
max P (A9)	*0*	*0*	*0*	*0*
max Q (A10)	*0*	*0*	*0 (A11)*	*0*
max R (A12)	*0*	*0*	*3*	*3*
unsupported	5	4	1	0
missing	0	1	0	0
Total	533	590	689	765

'public' take precedence over other categories, as the lest restricted.

- the *sdk* category is derived from "whitelist"/"sdk" lists and includes: "system-api,whitelist", "system-api,test-api,whitelist" and "sdk,system-api,test-api";
- the category *blacklist* is characterized by "blacklist"/ "blocked" lists and includes: "blacklist; blacklist,test-api", "blocked" and "blocked,test-api";
- the *conditional block* category is derived from "greylist"/ "max-target" lists and includes: "greylist-max", "lo-prio,max-target-x" and "lo-prio,max-target-x,test-api";
- the *unsupported* category is derived from "greylist"/ "unsupported" lists and includes: "greylist", "greylist,test-api" and "removed,unsupported";
- the *missing* category includes permissions present in the AndroidManifest, but absent from the restriction lists for that API level.

Tag. "Tags" are annotations that describe the intended usage of permissions by Android developers and are used to generate Java code documentation in HTML format from Android source code. Although some tags are referenced in the official documentation, there is no explanation of their functionality. Android source code contains the following tags:

- `@hide` — indicates permissions that are excluded from the public SDK API and consequently not intended for use by third-party developers;
- `@SystemApi` — indicates permissions available for internal system developers and, therefore, also not shown in public SDK APIs, as a result, each permission also has `@hide` annotation;
- `@TestApi` — indicates a permission used in testing;
- `@deprecated` — indicates a deprecated permission;
- `@removed` — indicates a permission removed from the API;

In our analysis, we found a few permissions in AOSP API 29 labelled as `#SystemApi` (note the use of #, not @). In the later versions, we observed that these permissions were relabelled with the `@SystemApi` tag. We assume that the initial labelling was a mistake.

Introduced. The "Introduced" attribute shows an API level, at which the permission was introduced, i.e., its functionality should not be supported in preceding Android versions. We could not find introduction API levels for many undocumented permissions. However, we could infer some introduction API levels during our manual inspection and comparison of permissions collected from the source code of different Android versions.

During this process, we also found inconsistencies in the available documentation concerning introduction levels. For example, the permission `READ_PRECISE_PHONE_STATE` is claimed to have been added in API level 30 (Android 11) in the official documentation, while it has been present in the source code since API level 29 (Android 10), and the permission `READ_NEARBY_STREAMING_POLICY`, claimed to have been added in API level 33 (Android 13) [29] is present in the source code of API level 31 (Android 12). An argument can be made that the more precise meaning of "Introduced" is "Exposed", as, in both of these cases, the permission was at first omitted from the public API generation with the tag `@hide`, which was later removed.

Deprecated. The "Deprecated" attribute indicates that the permission is outdated. A permission may be deprecated from a certain API level, meaning its use is accepted, if the `targetSdkVersion` of an app is lower than the deprecation API level, and there are no other restrictions for that particular case. Deprecated status may be noted in the official documentation, tagged, or mentioned in the comments in the source code. As we noticed, undocumented permissions are often labelled with the 'deprecated' status, although no corresponding API level information is provided. There is no documentation that clearly states what happens if these rules are not followed.

Protection. This attribute characterizes risk implied by a requested permission. In the increasing order of severity, permissions fall into one and only one of the following base groups [30]:

- `normal` — have the least risk associated with them;
- `signature` — permission is granted if an application requesting it is signed with the same signing key/certificate as the app defining it;
- `dangerous` — allows more substantial access to restricted data and functions, requires user acceptance at runtime of the application;
- `internal` — managed internally by the system and only granted according to additional protection flags;

These base protection levels may be complemented with additional protection flags [30]. As such, protection level can be represented by a base protection and zero or more flags. *Although the documentation mentions only four base protection levels, in our analysis of the source code, we found another base protection level 'system'.* It appeared in API 31–33 source code for one permission `android.permission.SYSTEM_CAMERA` that has a protection level: `system|signature|role`. However, `privileged` protection flag used to be named `system` that was deprecated in API 23. It was used for functionality accessible to applications built in to a system image. We suspect this is an indication of outdated source code.

Type. Each permission in AndroidManifest files from the source code is listed under one of three AndroidManifest file sections, which we refer to as "Type":

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

- Install permissions — supposed to be granted at application installation, their protection levels include normal and signature;
- Runtime permissions — might only be granted at application runtime by a user, as these permissions are considered to be high risk and should be labelled with dangerous protection level;
- Removed permissions — claimed to be present only for backward compatibility. Note that this type does not correlate to the `@removed` tag, as it appears for permissions of other types.

It seems that these sections are included for organization purpose, rather than being intended to enforce the functionality. For example, we found permissions listed under Install section with a `dangerous` protection level and permissions with protection levels `normal` and `signature` in Runtime section. Such inconsistencies do not necessarily lead to flawed security enforcement, but create ambiguity for both third-party and Android developers, complicating interpretation and development.

Status. "Status" is inferred from the Android API comments indicating the universal status of each permission. As such, we extracted three categories for this attribute:

- *relevant* — the permission is current and usable;
- *backward compatible* — the permission is needed for backward compatibility but is not expected to be used in recent applications, most often is also `@deprecated` with a *relevant* alternative;
- *removed* — unclear case, we were not able to identify why a 'removed' permission that is not backward compatible, was not removed from the source code.

Usage. "Usage" refers to various Android API comments indicating permission usage restrictions. Analyzing the Android source code comments, we inferred the following groups pertaining to the associated permissions:

- *general* — open for third-party developers;
- *not for use by third-party applications* — reserved for OS and OEMs;
- *restricted* — permissions reserved for specific internal services/processes.

This attribute is the most inconsistent, i.e., many permissions are restricted according to the other attributes but lack any indication of their usage restrictions with this attribute.

For some permissions, usage restrictions could be inferred from comments of the related permissions. For example, the permission `POWER_SAVER` does not have usage restrictions, but is described as "superseded by `DEVICE_POWER` permission", which in turn is described as "not for use by third-party apps". Another example is `INTERACT_ACROSS_USERS_FULL`, which has no restrictions mentioned, but is described as the "fuller form of `INTERACT_ACROSS_USERS`", which is also described as "not available to third party applications". In both cases, we conservatively infer that the "not for use by third-party apps" restriction applies to the earlier versions of these permissions as well.

We derived this attribute from comment sections in the source code of Android and used it as a control variable to detect discrepancies between the classification of permissions found in official documentation, and the classification of permissions found within the source code.

B. Permission Transition Across Android Versions

Our analysis revealed that permissions commonly and often silently transition between different categories of the non-SDK restriction lists as versions progress. Table I shows transitions of individual permissions found in these restriction lists.

There are a substantial number of permissions that remained assigned to the *public* and *sdk* categories across versions. Both categories are claimed to be intended for free use by developers. However, *public* permissions are documented, while documentation for *sdk* permissions is inconsistent. For example, all permissions listed in the *sdk* category in the restriction lists also have the tag `@SystemApi` in the source code, which implies that Android developers intended the permissions to be only granted to system apps. This intuitively contradicts the available documentation that associates the *sdk* category with accessible development tools (as noted in Section IV-A).

As Table I shows, the total number of permissions in *public*, *sdk* and *blacklist* categories increase over time. While a growing number of officially supported permissions is justifiable by the fast-evolving needs of the Android platform, a significant increase of blacklisted (inaccessible to developers) permissions (from 26 in Android 10 to 75 in Android 12) raises questions.

Intuitively, permissions are blacklisted when they are deemed to pose a security risk. However, we found that the majority of permissions associated with the *blacklist* category do not exist in the previous Android versions (Table IV). This is an indication that many new permissions are created with no intention of ever being available for third-party use, and if that were to happen, there might not be sufficient safeguards. We leave the investigation of such cases for future research. In a few of the cases, blacklisted permissions become officially supported (*sdk* or *public* categories). For example, the permission `android.permission.MANAGE_TEST_NETWORKS` had been initially introduced as blacklisted in Android 10 and then was moved to the *sdk* category in Android 12.

Similarly, `WIFI_UPDATE_USABILITY_STATS_SCORE` and `WIFI_SET_DEVICE_MOBILITY_STATE` permissions were blacklisted in Android 10 and then moved to the *sdk* category in Android 11. However, *the current official Android documentation incorrectly states that these permissions are unsupported in Android 10* [31], [32].

The number of *conditionally blocked* permissions did not decrease significantly over the versions (from 139 (Android 10) to 127 (Android 13)). The overwhelming majority of them were only allowed for APKs targeting Android 8 or lower, indicating that access will be blocked for versions Android 9 and higher. Interestingly, some of the permissions in this category transitioned to *sdk*, i.e., supported permissions, effectively implying that, although they were supposed to be blocked after Android 8, their presence is needed in later Android versions.

Across the four major Android versions covering restriction lists, we have identified one permission present in the AOSP source code, but missing from the restriction lists, we

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

TABLE II
COMBINATIONS OF PERMISSION ATTRIBUTES

Restriction		Tag		Protection	A10	A11	A12	A13
public	+	no tag	+	normal	45 (8.44%)	47 (7.97%)	54 (7.84%)	58 (7.58%)
public	+	no tag	+	signature	82 (15.38%)	88 (14.92%)	94 (13.64%)	103 (13.46%)
public	+	no tag	+	dangerous	30 (5.63%)	30 (5.08%)	34 (4.93%)	40 (5.23%)
public	+	no tag	+	internal	-	-	-	4 (0.52%)
public	+	*unknown*	+	*unknown*	1 (0.19%)	1 (0.17%)	1 (0.15%)	1 (0.13%)
sdk	+	hide	+	signature	-	-	-	1 (0.13%)
sdk	+	SystemApi	+	normal	1 (0.19%)	1 (0.17%)	1 (0.15%)	2 (0.26%)
sdk	+	SystemApi	+	signature	203 (38.09%)	238 (40.34%)	288 (41.80%)	333 (43.53%)
sdk	+	SystemApi	+	dangerous	1 (0.19%)	1 (0.17%)	1 (0.15%)	1 (0.13%)
sdk	+	SystemApi	+	internal	-	-	11 (1.60%)	18 (2.35%)
sdk	+	SystemApi	+	system	-	1 (0.17%)	1 (0.15%)	1 (0.13%)
sdk	+	*unknown*	+	*unknown*	-	-	1 (0.15%)	1 (0.13%)
cond. block (max O)	+	hide	+	normal	17 (3.19%)	15 (2.54%)	15 (2.18%)	14 (1.83%)
cond. block (max O)	+	hide	+	signature	120 (22.51%)	114 (19.32%)	112 (16.26%)	107 (13.99%)
cond. block (max O)	+	SystemApi	+	signature	-	-	1 (0.15%)	1 (0.13%)
cond. block (max O)	+	*unknown*	+	*unknown*	2 (0.38%)	2 (0.34%)	2 (0.29%)	2 (0.26%)
cond. block (max R)	+	hide	+	signature	-	-	3 (0.44%)	3 (0.39%)
unsupported	+	hide	+	signature	4 (0.75%)	3 (0.51%)	-	-
unsupported	+	SystemApi	+	signature	1 (0.19%)	1 (0.17%)	1 (0.15%)	-
blacklist	+	hide	+	signature	26 (4.88%)	47 (7.97%)	69 (10.01%)	73 (9.54%)
blacklist	+	SystemApi	+	signature	-	-	-	2 (0.26%)
missing	+	hide	+	signature	-	1 (0.17%)	-	-
Total					**533**	**590**	**689**	**765**

labelled it as *missing* in Table I. This permission is `android.permission.ADD_TRUSTED_DISPLAY` and it is missing from Android 11 restriction list. It was added in Android 12 as *blocked*.

> ***Answer to RQ1:*** *Our analysis revealed that the Android platform uses a more sophisticated permission classification than indicated by the official Android documentation and it is not fully conveyed to third-party developers. The official categorization is incomplete and, on many occasions, inconsistent with the treatment of individual permissions in the source code.*

V. RQ2: Permission Labelling Inconsistencies

While in our analysis we derived several permissions attributes, only three attributes present actionable mechanisms: Restriction, Protection, and Tag. Restriction lists and Protection levels are checked against the declared app permissions at the OS level, and Tags indicate developer's intention. Status, Usage and Type do not appear to be enforced in practice.

We further analyze derived permissions along three actionable attributes (Table II).

Unrestricted-access permissions. The vast majority of permissions (363–563, or 68–73% of all permissions depending on the version) have the restriction type *public* or *sdk*, indicating that they are open to third-party developers. Yet, among them, 4% (22 permissions in Android 13) are intended for `internal` use only, i.e., reserved for operating system functionality which directly contradicts their supposed availability for third-party developers, according to their protection.

We observe a similar discrepancy with 205–355 (38–46%) *sdk* permissions that have the `@SystemApi` tag, which indicates that these permissions are only available to system developers, despite the fact that "SDK" in Android refers to accessible development resources.

One permission has been assigned a protection level `@SystemApi` in Android 11. *This protection level does not exist in any official documentation.*

Limited-access permissions. We observe contradictory labelling for 165–202 (26–31%) permissions in Android 10–13 that have the restriction type *blacklist* or *conditionally blocked*.

133–121 *conditionally blocked (max O)* permissions are labelled with the `@hide` tag, implying that while these are not shown in public APIs, they are open to third-party apps targeting Android 8 (max O).

This contradicting labelling may be used as a preventative measure to caution third-party developers against using these permissions beyond the targeted version. This, for example, seems to be the case with three *conditionally blocked (max R)* permissions found in Android versions 12 and 13.

Undefined-access permissions 0–5 (0–0.9%) permissions have the restriction type *unsupported*. These are `signature` permissions. 4 of them are flagged with `@hide` tag and one as `@SystemApi`. All of them were relabelled over time:

- one transitioned to *blacklist* in API 30;
- three — to *conditional block R*in API 31;
- one — to *sdk* in API 33.

Unsupported permissions indicate that developers should not rely on them as they may be altered without notice.

Missing permissions. We identified one permission in the source code manifest files that is not present in the restriction lists (labelled as *missing*). The presence of the `@hide` tag indicates that Android developers intended to restrict access

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

to this permission for third-party developers. However, the absence of documentation for this permission and not other internal permissions suggests a larger issue of inconsistency in the documentation across the Android platform.

Vice versa, we found permissions given in the official restriction lists that are not present in any of the manifest files for the versions we parsed (their protection and tag attributes are labelled as *unknown*). These permissions are:

- `BIND_VISUAL_VOICEMAIL_SERVICE`
- `CLEAR_APP_GRANTED_URI_PERMISSIONS`
- `MANAGE_SCOPED_ACCESS_DIRECTORY_PERMISSIONS`
- `SEND_CATEGORY_CAR_NOTIFICATIONS`

The first three are present in all restriction lists iterations, and the last one only appeared in the *sdk* restriction lists starting Android 12 (API 31). There is also *no Android documentation mentioning these permissions*. We have confirmed that implementation for these methods is also absent. However, further analysis showed that these permissions (with the corresponding implementation) appear in some of the other Android releases.

The lack of uniform presence of permissions in publicly available and documented sources poses a significant risk of them being misused by developers.

A. Permission Labelling Inconsistencies

We identified the following thirteen types of labelling inconsistencies[3]:

- (**NR**) Runtime permissions that have protection level `normal` as opposed to expected `dangerous`;
- (**SR**) Runtime permissions that have protection level `signature`, while documented to have `dangerous`;
- (**DC**) Install permissions that have protection level `dangerous`, while documented to have protection level `normal` or `signature`;
- (**BG**) permissions with a restriction *blacklist* and thus are expected to be properly annotated as *not available for use in third-party apps*, yet lacking this annotation;
- (**ET**) permissions annotated as *not available for use in third-party apps*, yet not tagged with `@hide` to be excluded from the public API documentation;
- (**PT**) permissions annotated as *not available for use in third-party apps* and assigned to the *public* restriction list;
- (**EL**) permissions annotated as *restricted* and not tagged with `@hide` to be excluded from the public API documentation;
- (**PL**) permissions annotated as *restricted* and assigned to the *public* restriction list;
- (**EI**) permissions that have a protection level `internal` but are not tagged with `@hide` from the public API;
- (**PI**) permissions have protection level `internal`, but are assigned to the *public* restriction list;
- (**IG**) permissions with a protection level `internal` that are not properly annotated as *not available for use in the third-party apps*;
- (**HG**) permissions excluded from the public API documentation by the tag `@hide`, yet not properly annotated as *not available for use in third-party apps*;
- (**OG**) permissions excluded from the public API documentation by the tag `@SystemApi` that are not annotated as *not available for use in third-party apps*;

[3]The coding of conflicts uses the following scheme: tags (**E** — `no tag`, **H** — `@hide`, **O** — `@SystemApi`); restriction categories (**P** — *public*, **B** — *blacklist*); protection levels (**N** — `normal`, **S** — `signature`, **D** — `dangerous`, **I** — `internal`); usages (**G** — *general*, **T** — *not for 3rd-party*, **L** — *restricted*); types (**C** — Install, **R** — Runtime).

TABLE III
DISCOVERED PERMISSION LABELLING INCONSISTENCIES

Conflicts	Number of Conflicts				Total Changes	
	A10	A11	A12	A13	Added	Removed
NR	4	4	5	6	2	0
SR	4	9	10	11	7	0
DC	1	1	5	6	5	0
BG	7	15	30	34	33	6
ET	40	41	42	48	8	0
PT	40	41	42	48	8	0
EL	2	3	3	3	1	0
PL	2	3	3	3	1	0
EI	0	0	0	4	4	0
PI	0	0	0	4	4	0
IG	0	0	4	12	12	0
HG	88	92	105	107	32	13
OG	83	101	123	150	64	0
Total	271	310	372	435	181	19

TABLE IV
TRANSITION OF BLACKLIST PERMISSIONS ACROSS VERSIONS

Blacklist Changes	A10	A11	A12	A13
Newly added	26	24	28	14
Removed	-	2	3	0
Moved	-	3	3	8
blacklist → sdk	-	2	3	7
blacklist → public	-	0	0	1
unsupported → blacklist	-	1	0	0
Total	26	47	69	75

We discovered 1,388 cases of labelling contradictions between the official documentation and the Android platform source code of the analyzed Android versions. The summary of these conflicts is listed in Table III. As our analysis showed, *the vast majority of these contradictions tend to persist across different versions and are rarely rectified*. For example, the number of **NR**, **SR** and **DC** inconsistencies increased slightly over time, and all discrepancies present in older versions transitioned to subsequent releases.

The **BG** inconsistencies, i.e., permissions with a *blacklist* restriction, were partially corrected: two permissions were moved to the *sdk* category and one completely removed in Android 12, while one permission was moved to the *public* category and two to the *sdk* category in Android 13.

Yet, among the additions to the **BG** group, almost all were newly introduced, except for two. For example, the permission `TEST_MANAGE_ROLLBACKS` had been *unsupported*, then was moved to *blacklist* category in Android 11, however, this change was not reflected in the code. On

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

TABLE V
RESTRICTION CATEGORIES OF INTERFACES PRESENT IN SOURCE CODE

Restriction Category	A10(Q)		A11(R)		A12(S)		A13(T)	
	Present in Restriction List	Found in Source Code	Present in Restriction List	Found in Source Code	Present in Restriction List	Found in Source Code	Present in Restriction List	Found in Source Code
Total methods	389,084	1131	428,360	1721	495,713	2304	537,427	2147
public	94186	246 (.3%)	100448	332 (.3%)	108774	494 (.5%)	116717	362 (.3%)
sdk	9013	557 (6.2%)	11609	854 (7.4%)	14596	968 (6.6%)	17023	1099 (6.5%)
blacklist	152936	117 (.1%)	189612	284 (.1%)	248898	470 (.2%)	282514	515 (.2%)
cond. block	106493	96 (.1%)	102422	134 (.1%)	102298	264 (.3%)	100169	103 (.1%)
max O (A8)	*105695*	*88 (.1%)*	*100824*	*123 (.1%)*	*97607*	*220 (.2%)*	*95543*	*82 (.1%)*
max P (A9)	*798*	*8 (1.0%)*	*776*	*10 (1.3%)*	*762*	*6 (.8%)*	*756*	*6 (.8%)*
max Q (A10)	*0*	-	*822*	*1 (.1%)*	*823*	*0 (.0%)*	*823*	*4 (.5%)*
max R (A11)	*0*	-	*0*	-	*3106*	*38 (1.2%)*	*3043*	*11 (.4%)*
max S (A12)	*0*	-	*0*	-	*0*	-	*4*	*(.0%)*
unsupported	26456	115 (.4%)	24269	117 (.5%)	21147	108 (.5%)	21004	68 (.3%)
Absent methods	-	*7*	-	*15*	-	*34*	-	*49*
Total	**389,084**	**1138**	**428,360**	**1736**	**495,713**	**2338**	**537,427**	**2196**

the other hand, the permission ADD_TRUSTED_DISPLAY is present in the code but is missing from the restriction lists for Android 11.

Similarly, none of the labelling **ET**, **PT**, **IG**, **OG** contradictions were resolved, while some of the newly added permissions were introduced with contradictory labels.

We noticed a few exceptions to this pattern. For example, OVERRIDE_WIFI_CONFIG from the *sdk* restriction category had been tagged @SystemApi in Android 12. In Android 13, it was moved to the *public* category, accompanied by the removal of the @SystemApi tag. Yet, in the source code, its plain-text comment still contains "*not for use by third-party applications*", hence, the appearance of the **ET** inconsistency.

Similarly, one permission in **PT** group was moved from *sdk* to *public* restriction category, while its comment was not updated.

One **EL**, one **PL**, four **EI**, and four **PI** permissions were all introduced with their respective inconsistencies, and none were fixed or removed.

Most of the twelve permissions appearing with the **IG** group were newly introduced permissions, except for two cases (in Android 12 and 13) that both previously had protection signature, which was then changed to internal, while other restrictions stayed the same (tag @SystemApi, restriction *sdk*).

HG permissions were altered on several occasions: eight became **OG** inconsistencies, three were removed in the next major version, and one had its tag removed (which appears to be a fix). All 32 permissions across Android versions 11–13 were introduced with the inconsistency.

OG group had no permissions fixed or removed. Out of 67 permissions appearing with this inconsistency across Android versions 11–13, 57 were first-time introduced, while 10 already existed: eight were changed from the **HG** group, and two had their *not for third-party* usage comment removed.

The majority of these discrepancies pertain to non-actionable attributes that contradict actionable attributes and can be largely attributed to poor or outdated code documentation. However, one particular conflict, **PI**, involves two conflicting actionable attributes — the internal protection level and *public* restriction. This conflict was observed in the latest Android version 13, indicating that Android development practices still lack internal mechanisms for ensuring consistency in categorization.

> ***Answer to RQ2:*** *Our analysis revealed that some attribute combinations provide redundant or inconsistent labelling as a result of official documentation conflicting with the source code. Overall, we discovered 1,388 cases of permission labelling inconsistencies between the official documentation and the four analyzed versions of Android platform source code. Among them, we identified one permission present in the Android source code but not mentioned in the restriction lists. Similarly, we found four permissions given in the official restriction lists but missing from the manifest files of the four versions. In all four versions, the number of inconsistencies either increased or, in rare cases, remained the same. We have not observed any instances where the number of conflicting permissions decreased as it appears that discrepancies are rarely rectified in the subsequent versions.*

VI. RQ3: INCONSISTENCIES IN THE PROTECTIONS PROVIDED FOR GUARDED INTERFACES

Across Android versions 10–13, we derived 3,615 unique elements (e.g., methods, variables) protected by permissions. Similar to permissions, the elements in restriction lists are categorized into the following categories: *blacklist*, *conditionally blocked*, *unsupported*, *sdk* and *public*. The summary of the elements for each category are given in Table V.

As the results show, 70 methods were absent from the restriction lists and not found in the official documentation. It appeared that some of these elements were tied to the bug reports and internal testing, while most of the remaining were located under */services/core/java/com/android/server* path in the source code and included services such as BluetoothManagerService, ContentService, NotificationManagerService, etc.

It is interesting to note that the number of these absent methods increased over time. The nature of this absence is not clear, since the restriction lists appear to be automatically

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

TABLE VI
OVERVIEW OF PROTECTION INCONSISTENCIES

Call Restriction	Permission Requirement*	Permission Restriction	A10 (P)	A11 (Q)	A12 (R)	A13 (S)
blacklist	one	unknown	-	-	2	2
		cond. block O	16	10	29	35
		cond. block R	-	-	1	1
		public	16	53	86	66
		sdk	40	122	165	207
		unsupported	-	1	2	-
	all	unknown, public	-	-	-	1
		public, sdk	-	-	3	3
		public	3	1	18	4
		sdk	-	-	-	3
	any	unknown, sdk	-	3	5	9
		blacklist, sdk	-	1	-	-
		public, sdk	-	-	-	2
		public	-	-	2	2
		sdk	-	10	17	30
		Total calls:	**75 (64%)**	**201 (71%)**	**330 (70%)**	**365 (71%)**
conditional block O	one	unknown	1	1	-	-
		blacklist	1	1	2	2
		public	27	45	127	14
		sdk	12	39	32	32
	all	public, sdk	1	1	1	1
		public	1	1	32	2
		sdk	4	4	4	4
	any	cond. block O, sdk	-	-	-	-
		public	4	5	3	4
		sdk	2	8	5	8
		Total calls:	**53 (60%)**	**105 (85%)**	**206 (94%)**	**67 (82%)**
conditional block P	one	public	7	7	4	3
		sdk	1	1	1	1
	any	unknown, sdk	-	1	-	-
		public	-	1	1	2
		Total calls:	**8 (100%)**	**10 (100%)**	**6 (100%)**	**6 (100%)**
conditional block Q	one	sdk	-	1	-	-
	any	public	-	-	-	4
		Total calls:	**-**	**1 (100%)**	**0**	**4 (100%)**
conditional block R	one	cond. block O	-	-	1	1
		public	-	-	26	2
		sdk	-	-	4	4
	all	public, sdk	-	-	-	-
		public	-	-	4	-
	any	sdk	-	-	2	2
		Total calls:	**-**	**-**	**37 (97%)**	**9 (82%)**
public	one	blacklist	-	2	4	3
		cond. block O	1	-	-	-
		sdk	23	44	50	61
	all	sdk	3	3	3	3
	any	cond. block O	1	-	-	-
		public, sdk	-	12	15	21
		sdk	3	7	6	5
		Total calls:	31 (13%)	**68 (21%)**	**78 (16%)**	**106 (29%)**
sdk	one	unknown	5	6	2	2
		blacklist	7	11	10	49
		cond. block O	17	19	19	19
		public	150	215	224	217
		unsupported	1	1	1	-
	all	blacklist, public	-	1	1	1
		blacklist	-	2	2	2
		public	23	26	48	10
	any	unknown, cond. block O	1	-	-	-
		unknown, sdk	-	5	2	-
		blacklist	-	-	1	1
		cond. block O, sdk	13	-	1	-
		cond. block O	4	-	-	-
		public, sdk	3	6	13	15
		public	4	23	26	9
		Total calls:	**228 (41%)**	**315 (37%)**	**350 (36%)**	**325 (30%)**
unsupported	one	blacklist	6	1	2	2
		cond. block O	8	4	3	3
		public	44	55	49	17
		sdk	49	37	29	25
	all	public	2	2	6	2
		sdk	-	-	-	1
	any	unknown, sdk	-	1	-	-
		blacklist, public, sdk	1	1	1	-
		public, sdk	1	2	2	3
		public	1	2	2	2
		sdk	-	7	9	9
		Total calls:	**112 (97%)**	**112 (96%)**	**103 (95%)**	**64 (94%)**

* - 'one' indicates there is only one required permission, 'all' indicates several permissions are required, 'any' indicates that any of the listed permissions are sufficient to invoke a method.

generated as a sum for all AOSP builds and are meant to include all existing methods including functionality limited for third-party developers.

Among the elements protected by permissions, 32 methods appeared to transition from the *unsupported* to the *blocked* restriction category. Out of them, only one, *enableVerboseLogging(I)* had an alternative call, *setVerboseLoggingEnabled(Z)*, suggested on the restriction lists update page for Android 11.

We discovered numerous protection inconsistencies that are summarized in Table VI. As our review shows, the most consistent are the *public* and *sdk* elements: more than 50% match the restrictions of their permissions.

Interestingly, among the discovered inconsistencies, we observed two *public* elements protected by a single blacklisted permission that existed since Android 11, (the number of similar elements changed to five in Android 12 and four in Android 13, and four different permissions were identified as their requirements). For example, a method in the BiometricManager called "canAuthenticate" was protected by a *public* permission `USE_BIOMETRIC`, which was changed to a new *blacklist* permission `USE_BIOMETRIC_INTERNAL`, while the element restriction remained *public* in Android 10–13. One factor that could explain this behaviour is the optional `conditional` parameter, an optional boolean value for `@RequiresPermission`. It is set to `true` if an element may not require permissions under some circumstances, such as certain call parameters, certain platforms, etc. [33]. However, this is not the case for "canAuthenticate". There is seemingly no reason for the element to stay public, if it cannot be accessed inside the public SDK.

The majority of *blacklisted* elements can be accessed with *public* or *sdk* permissions. Although these elements are not available to the public, it is interesting to see that no additional safeguards are placed by Android developers.

The permissions that safeguard the *conditionally blocked* elements are predominantly restricted as *public* and *sdk*. Since these elements are primarily maintained for backward compatibility purposes, the existence of unrestricted permissions is not a cause for concern.

We discovered, however, a clear contradiction in one of the elements available Android 12 and 13 and restricted as *conditional block R*. It is safeguarded with one permission restricted as *conditional block O*. In other words, this method is blocked for applications that target Android 12 or higher and is protected by a permission that is blocked on devices with Android 9 and higher. Consequently, if an application is targeting Android 9, 10, or 11, the permission is blocked and consequently, the method is not accessible regardless of its seeming availability for these Android versions.

As for the *unsupported* interfaces, there are no guidelines to be applied due to their unrestricted nature. Our analysis confirms that their numbers are decreasing as Android promised [34], although the rate of decrease is slow (with 26,000 elements in Android 10 and 21,000 elements in Android 13).

Among other problems, we discovered 32 elements requiring permissions absent from our collected *unified permission mapping*:

- `MANAGE_ROLES_FROM_CONTROLLER`,
- `PERMISSION_MAINLINE_NETWORK_STACK`,
- `MANAGE_SIM_ACCOUNTS` and
- `ACCESS_LAST_KNOWN_CELL_ID`.

These permissions were not defined for any of the considered Android versions. Furthermore, we could not find any information about them, neither in the official documentation nor in any other open sources. These permissions are labelled as *unknown* in Table VI. They correspond to 22 cases of elements with a *blacklist* restriction, two cases of elements with a *conditional block O* restriction, one case of an element with a *conditional block P* restriction, 23 cases of elements with an *sdk* restriction, and one case of an element with an *unsupported* restriction across Android versions 10–13.

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

TABLE VII
PERMISSIONS GRANTED TO THE TESTING APP (BENCHMARK ANALYSIS)

Restriction	Tag	Type	Status	Protection	Usage	Issues, Conflicts	Permissions on Android 12	Granted on Devices			
								10	11	12	13
public	sdk	install	relevant	normal	general	-	44	36	38	44	44
public	sdk	install	backw. comp	normal	general	-	4	3	3	4	4
public	sdk	runtime	relevant	normal	general	NR	5	5	5	5	5
public	sdk	runtime	backw. comp	normal	general	NR	1	1	1	1	1
sdk	**system**	**install**	**relevant**	**normal**	**not for 3rd-p.**	**N3,SY**	**1**	**1**	**1**	**1**	**1**
cond. block (o)	**hide**	**install**	**relevant**	**normal**	**general**	**CB**	**0**	**2**	**0**	**0**	**0**
cond. block (o)	**hide**	**removed**	**backw. comp**	**normal**	**general**	**CB**	**15**	**15**	**15**	**15**	**15**
blacklist	**hide**	**install**	**relevant**	**signature**	**general**	**BL**	**30**	**0**	**0**	**0**	**0**

Note: Accented in **bold** are rows where a permission violation occurred, the rest of cases were handled according to documentation

Initially, permissions were established to control access to Android functionality, while later, restriction lists were introduced as a backup measure to prevent access to specific elements through reflection or JNI, even if the necessary permission was granted. However, the absence of a coherent link between these two systems has resulted in significant challenges in their maintenance. It appears that the changes are made to these systems independently. As our analysis shows, it becomes more and more difficult for Android developers to ensure their accuracy, for vendors to comply with them, and we suspect it becomes nearly impossible for third-party developers to follow them correctly and conscientiously.

> ***Answer to RQ3:*** *We identified 3,362 protection inconsistencies. Among them, we discovered 32 interfaces protected with permissions absent from the official Android documentation, the restricted lists and source code of four analyzed versions.*

VII. RQ4: PRESENCE OF DISCOVERED INCONSISTENCIES IN THIRD-PARTY ANDROID APPS

A. Permission Violations in Practice

In addition to the identified theoretical permission labelling conflicts, we discovered several scenarios in practice when (1) apps are requesting permissions that should not be accessible to third-party apps, and (2) apps are granted permissions that are not expected to be granted to third-party apps. PChecker identified the following permission violations:

- (**AH**) permissions tagged `@hide` are not meant to be available to third-party apps, and consequently should not be requested by third-party developers;
- (**AO**) permissions tagged `@SystemApi` should be only available to system apps, and not third-party apps;
- (**N3**) permissions marked as *not for third-party apps* should not be granted to third-party apps;
- (**RE**) permissions marked as *restricted* should not be granted to third-party apps;
- (**CB**) permissions restricted as *Conditionally blocked* should not be granted to apps that target an Android version over the set limit, e.g., *max-O* permissions should be blocked for apps installed on devices with Android version over 8 (O);
- (**BL**) permissions restricted as *blacklisted* should not be granted to apps;
- (**SY**) permissions tagged `@SystemApi` should not be granted to third-party apps.

B. Baseline Analysis

To establish a baseline for our analysis, we developed a testing app containing all permissions derived for Android versions 10–13. None of the permissions were required for the app's functionality. We then used PChecker to analyze this testing app. The results of this analysis are presented in Table VII. Out of 689 requested permissions, our testing app was granted 63–71 permissions. Among them, only 43–55 were open to third-party developers (*public* and *sdk* categories).

All existing permissions with the **NR** conflict were granted at installation according to their `normal` protection level. Even though they are granted automatically, their placement under the Runtime section in the source code determines their runtime evaluation, mishandling them could lead to protection blind spots.

The largest number of discovered conflicts (17) is related to a restriction attribute *Conditionally blocked (O)* (**CB** issue). Despite being expected to be blocked for apps targeting Android 9 and higher, these permissions were automatically granted during installation. While we intentionally left the target version of our app unspecified, all devices assumed it to have the `targetSdkVersion` of 32. This reinforces the fact that the Android platform (or the corresponding OEM implementation) does not respect restriction rules for outdated permissions.

Another conflicting case that PChecker has discovered is one blacklisted permission (**BL** issue) that was granted on Android 13. The `READ_NEARBY_STREAMING_POLICY` permission was *blacklisted* for API levels 31 and 32. This permission was moved to the *public* restriction list for API level 33. However, despite being identified as having `targetSdkVersion` 32 (Android 12) on all four devices.

Both CB and BL issues were granted to our app in direct contradiction with the official documentation that states that conditionally blocked permissions are restricted to app's target

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

TABLE VIII
THE SUMMARY OF ANDROID APPS

Source	# APKs	Installable	Runnable	Apps' Target API Level				
				29	30	31	32	33
androgalaxy (2019)	257	190	175 (68.09%)	175	0	0	0	0
androidapkfree.com (2020)	319	200	169 (52.98%)	160	9	0	0	0
apkgod (2020)	471	393	356 (75.58%)	356	0	0	0	0
APKMAZA (2020)	34	25	23 (67.65%)	23	0	0	0	0
APKPure (2021)	317	180	165 (52.05%)	155	10	0	0	0
appsapk.com (2020)	161	132	122 (75.78%)	122	0	0	0	0
CracksHash (2022)	1068	913	705 (66.01%)	38	470	179	18	0
CracksHash (2021)	2015	1476	969 (48.09%)	391	520	57	1	0
F-droid (2020)	1202	1186	997 (82.95%)	943	54	0	0	0
Google Play (2023)	200	184	184 (92%)	0	2	69	14(16*)	99(97*)
Total	**6044**	**4879**	**3865 (62.99%)**	**2363**	**1065**	**305**	**33(35*)**	**99(97*)**

* - the Samsung device with Android 12 downloaded two apps with API levels 32 which had API levels 33 on other devices

API level, and blocklisted permissions cannot be used regardless of app's target API level [15]. According to the official documentation, in all these cases, the OS was expected throw an error, yet the permissions were granted.

Among similar cases is the `READ_INSTALL_SESSIONS` permission that was granted to our testing app on installation in all four cases. The permission is labelled as "not a third-party API (intended for system apps)" and is a part of `@SystemApi` tag category (**N3**, **SY** issue). The permission grants the app visibility into details of installed on the device apps. If automatically granted, a malicious app may gain sensitive information. For example, the presence of a diabetes app on a phone implies that a user likely has diabetes [35].

C. App Analysis

To understand whether the discovered inconsistencies appear in third-party apps, we leverage PChecker to explore Android apps collected in the wild. For this analysis, we collected all publicly accessible sets of benign apps from six websites distributing Android applications: androidapkfree.com, appsapk.com, androgalaxy (no longer exists), CracksHash, apkgod, APKMAZA. We complemented this set with a dataset of open-source apps extracted form F-droid market and used in the study by Foroughipour et al. [36] and all accessible apps on APKPure.com market. To provide the comparison we downloaded the top 200 free apps from Google Play store. This resulted in a set of 35,653 APKs. We excluded from our analysis duplicate apps, invalid APKs that could not be decompressed, APKs without an AndroidManifest.xml file, APKs that had no API level present in the manifest file, had an invalid API level (more than 33), or whose signature was not verified successfully by apksigner [37], a signature verification tool. We further excluded apps that contained less than 2 files and contained less than 2 permissions in the manifest file. Apps with no permissions declared in the manifest file automatically receive at least 2 permissions from the compiler when the package is created, i.e., presence of only 2 permissions indicated that developers have not requested any permissions. Excluding invalid and irrelevant for analysis apps, left us with 17,598 APKs.

Our analysis was based on the available restriction lists introduced for Android 10, hence, we further selected APKs that had a `targetSdkVersion` $\geq$ 29 (Android 10), assuming they were developed in line with recent practices and limitations set by Google. This resulted in a subset containing 6,044 applications. Out of which, only 3,865 applications were successfully installed on all test devices and analyzed using PChecker. For comparative analysis, we present the results separately for apps derived from the unofficial markets (Table IX) and Google Play (Table X).

D. Requested Permissions

Conditionally blocked permissions. 913 (2.4%) requested permissions in our apps from the unofficial markets are *conditionally blocked O*. Since all selected apps in our set target Android 29 or higher, all these permissions were expected to be blocked. Yet, developers chose to use them and the devices granted them. This also holds true for the apps retrieved from Google Play where conditionally blocked permissions comprised 3% of all requested permissions.

Not for third-party developers. 2.4% of all requested permissions are tagged with `@hide`. These permissions are not part of the public API, but *were requested by apps nonetheless*. Furthermore, these permissions are either conditionally blocked or blacklisted/unsupported, emphasizing their restricted nature.

Similarly, the permissions with the `@SystemApi` tag are not expected to be used by third-party developers, yet 183–186 (0.5%) are requested. Some of these requested permissions (2–16%) were granted, which directly contradicts the official documentation. In comparison, Google Play apps requested less permissions with this attribute (0.2%), and they were granted in only two cases.

Our study shows a common trend of third-party developers routinely requesting permissions that should not be available to apps. While the number of these cases is gradually reducing, i.e., fewer are being granted on Android 13 than on Android 11, it is still significant in the latest versions of Android.

Blacklisted permissions. We found a small percentage of cases where apps targeting API 29 from the unofficial

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

TABLE IX
RESULTS OF ANDROID APPS ANALYSIS (UNOFFICIAL MARKETS)

Restriction	Tag	Type	Status	Protection	Usage	Android 10		Android 11		Android 12		Android 13	
						Requested	Granted	Requested	Granted	Requested	Granted	Requested	Granted
public	no tag	Install	relevant	normal	general	21865	21643 (99.0%)	21865	21836 (99.9%)	21854	21853 (100.0%)	21854	21853 (100.0%)
public	no tag	Runtime	relevant	normal	general	251	248 (98.8%)	251	248 (98.8%)	251	250 (99.6%)	251	250 (99.6%)
public	no tag	Install	backw. comp	normal	general	934	934 (100.0%)	934	934 (100.0%)	934	934 (100.0%)	934	934 (100.0%)
public	no tag	Runtime	backw. comp	normal	general	332	332 (100.0%)	332	332 (100.0%)	332	332 (100.0%)	332	332 (100.0%)
public	no tag	Install	relevant	signature	general	1886	0 (.0%)	1886	6 (.3%)	1886	6 (.3%)	1886	0 (.0%)
public	**no tag**	**Install**	**relevant**	**signature**	**not for 3rd-p.**	298	0 (.0%)	298	**10 (3.4%)**	298	**10 (3.4%)**	298	0 (.0%)
public	no tag	Install	relevant	signature	restricted	42	0 (.0%)	42	0 (.0%)	42	0 (.0%)	42	0 (.0%)
public	no tag	Runtime	relevant	signature	restricted	74	0 (.0%)	74	0 (.0%)	74	0 (.0%)	74	0 (.0%)
public	no tag	Install	backw. comp	signature	general	2	0 (.0%)	2	0 (.0%)	2	0 (.0%)	2	0 (.0%)
public	no tag	Install	relevant	dangerous	general	402	0 (.0%)	402	0 (.0%)	402	0 (.0%)	402	0 (.0%)
public	no tag	Runtime	relevant	dangerous	general	10364	0 (.0%)	10249	0 (.0%)	10238	0 (.0%)	10237	0 (.0%)
public	no tag	Runtime	backw. comp	dangerous	general	25	0 (.0%)	25	0 (.0%)	25	0 (.0%)	25	0 (.0%)
cond. block O	**hide**	**Install**	**relevant**	**normal**	**general**	8	**8 (100.0%)**	8	0 (.0%)	8	0 (.0%)	8	0 (.0%)
cond. block O	**hide**	**Removed**	**backw. comp**	**normal**	**general**	895	**895 (100.0%)**	895	**895 (100.0%)**	895	**895 (100.0%)**	895	**895 (100.0%)**
cond. block O	**hide**	**Install**	**relevant**	**signature**	**general**	5	0 (.0%)	5	**2 (40.0%)**	5	**2 (40.0%)**	5	0 (.0%)
cond. block O	**hide**	**Install**	**relevant**	**signature**	**not for 3rd-p.**	2	0 (.0%)	2	**2 (100.0%)**	2	**2 (100.0%)**	2	0 (.0%)
cond. block O	hide	Install	relevant	signature	restricted	3	0 (.0%)	3	0 (.0%)	3	0 (.0%)	3	0 (.0%)
unsupported	hide	Install	Removed	signature	not for 3rd-p.	3	0 (.0%)	3	0 (.0%)	3	0 (.0%)	3	0 (.0%)
blacklist	**hide**	**Install**	**relevant**	**signature**	**general**	2	0 (.0%)	2	**2 (100.0%)**	2	**2 (100.0%)**	2	0 (.0%)
sdk	**SystemApi**	**Install**	**relevant**	**normal**	**not for 3rd-p.**	4	**4 (100.0%)**	4	**4 (100.0%)**	4	**4 (100.0%)**	4	**4 (100.0%)**
sdk	**SystemApi**	**Install**	**relevant**	**signature**	**general**	71	0 (.0%)	69	**10 (14.5%)**	69	**10 (14.5%)**	69	0 (.0%)
sdk	**SystemApi**	**Install**	**relevant**	**signature**	**not for 3rd-p.**	97	0 (.0%)	97	**14 (14.4%)**	97	**14 (14.4%)**	97	0 (.0%)
sdk	**SystemApi**	**Install**	**relevant**	**signature**	**restricted**	2	0 (.0%)	2	**1 (50.0%)**	2	**1 (50.0%)**	2	0 (.0%)
sdk	**SystemApi**	**Install**	**backw. comp**	**signature**	**general**	8	0 (.0%)	7	0 (.0%)	7	0 (.0%)	7	0 (.0%)
sdk	SystemApi	Install	backw. comp	signature	not for 3rd-p.	3	0 (.0%)	3	0 (.0%)	3	0 (.0%)	3	0 (.0%)
sdk	SystemApi	Runtime	relevant	dangerous	general	1	0 (.0%)	1	0 (.0%)	1	0 (.0%)	1	0 (.0%)
Total						**37579**	**24064 (64.0%)**	**37461**	**24296 (64.9%)**	**37439**	**24315 (64.9%)**	**37438**	**24268 (64.8%)**

Note: Accented as **bold** are rows where an issue occurred for any app on any of the devices, the rest were handled according to documentation

TABLE X
RESULTS OF ANDROID APPS ANALYSIS (GOOGLE PLAY MARKET)

Restriction	Tag	Type	Status	Protection	Usage	Android 10		Android 11		Android 12		Android 13	
						Requested	Granted	Requested	Granted	Requested	Granted	Requested	Granted
public	no tag	Install	relevant	normal	general	1493	1449 (97.1%)	1553	1517 (97.7%)	1471	1470 (99.9%)	1449	1449 (100.0%)
public	no tag	Runtime	relevant	normal	general	56	46 (82.1%)	61	49 (80.3%)	57	55 (96.5%)	55	55 (100.0%)
public	no tag	Install	backw. comp	normal	general	35	35 (100.0%)	36	36 (100.0%)	35	35 (100.0%)	33	33 (100.0%)
public	no tag	Runtime	backw. comp	normal	general	43	43 (100.0%)	45	45 (100.0%)	42	42 (100.0%)	40	40 (100.0%)
public	no tag	Install	relevant	signature	general	52	2 (3.8%)	55	0 (.0%)	51	0 (.0%)	52	0 (.0%)
public	no tag	Install	relevant	signature	not for 3rd-p.	7	0 (.0%)	7	0 (.0%)	7	0 (.0%)	7	0 (.0%)
public	no tag	Runtime	relevant	signature	restricted	4	0 (.0%)	4	0 (.0%)	4	0 (.0%)	4	0 (.0%)
public	no tag	Install	relevant	dangerous	general	86	0 (.0%)	95	0 (.0%)	85	0 (.0%)	90	0 (.0%)
public	no tag	Runtime	relevant	dangerous	general	853	0 (.0%)	886	0 (.0%)	850	0 (.0%)	832	0 (.0%)
public	no tag	Runtime	backw. comp	dangerous	general	1	0 (.0%)	1	0 (.0%)	1	0 (.0%)	1	0 (.0%)
cond. block O	**hide**	**Removed**	**backw. comp**	**normal**	**general**	86	**86 (100.0%)**	92	**92 (100.0%)**	89	**89 (100.0%)**	87	**87 (100.0%)**
cond. block O	hide	Install	relevant	signature	not for 3rd-p.	1	0 (.0%)	1	0 (.0%)	1	0 (.0%)	1	0 (.0%)
cond. block R	hide	Install	Removed	signature	not for 3rd-p.	2	0 (.0%)	2	0 (.0%)	2	0 (.0%)	2	0 (.0%)
sdk	**SystemApi**	**Removed**	**backw. comp**	**normal**	**general**	0	0 (.0%)	0	0 (.0%)	1	**1 (100.0%)**	1	**1 (100.0%)**
sdk	SystemApi	Install	relevant	signature	not for 3rd-p.	6	(.0%)	6	0 (.0%)	5	0 (.0%)	5	0 (.0%)
Total						**2725**	**1661 (61.0%)**	**2844**	**1739 (61.1%)**	**2701**	**1692 (62.6%)**	**2659**	**1665 (62.6%)**

Note: Accented as **bold** are rows where an issue occurred for any app on any of the devices, the rest were handled according to documentation

TABLE XI
RESULTS OF ANDROID APPS ANALYSIS (GOOGLE PLAY MARKET, SAMSUNG DEVICES)

Restriction	Tag	Type	Status	Protection	Usage	Android 11		Android 12		Android 13	
						Requested	Granted	Requested	Granted	Requested	Granted
public	no tag	Install	relevant	normal	general	1136	1104 (97.2%)	1535	1534 (99.9%)	1590	1598 (100.5%)
public	no tag	Runtime	relevant	normal	general	44	34 (77.3%)	61	59 (96.7%)	67	67 (100.0%)
public	no tag	Install	backw. comp	normal	general	29	29 (100.0%)	36	36 (100.0%)	38	38 (100.0%)
public	no tag	Runtime	backw. comp	normal	general	30	30 (100.0%)	45	45 (100.0%)	49	49 (100.0%)
public	no tag	Install	relevant	signature	general	44	0 (.0%)	51	0 (.0%)	56	0 (.0%)
public	no tag	Install	relevant	signature	not for 3rd-p.	7	0 (.0%)	8	0 (.0%)	8	0 (.0%)
public	no tag	Runtime	relevant	signature	restricted	4	0 (.0%)	4	0 (.0%)	4	0 (.0%)
public	no tag	Install	relevant	dangerous	general	65	0 (.0%)	87	0 (.0%)	96	0 (.0%)
public	no tag	Runtime	relevant	dangerous	general	656	0 (.0%)	901	0 (.0%)	933	0 (.0%)
public	no tag	Runtime	backw. comp	dangerous	general	1	0 (.0%)	1	0 (.0%)	1	0 (.0%)
cond. block O	**hide**	**Removed**	**backw. comp**	**normal**	**general**	73	**73 (100.0%)**	95	**95 (100.0%)**	101	**101 (100.0%)**
cond. block O	hide	Install	relevant	signature	not for 3rd-p.	1	0 (.0%)	1	0 (.0%)	1	0 (.0%)
cond. block R	hide	Install	Removed	signature	not for 3rd-p.	1	0 (.0%)	2	0 (.0%)	2	0 (.0%)
sdk	**SystemApi**	**Removed**	**backw. comp**	**normal**	**general**	1	**1 (100.0%)**	1	**1 (100.0%)**	1	**1 (100.0%)**
sdk	SystemApi	Install	relevant	signature	not for 3rd-p.	5	0 (.0%)	5	0 (.0%)	5	0 (.0%)
Total						**2097**	**1271 (60.6%)**	**2833**	**1770 (62.5%)**	**2952**	**1854 (62.8%)**

Note: Accented as **bold** are rows where an issue occurred for any app on any of the devices, the rest were handled according to documentation

markets requested a *blacklisted* permission. These cases concern one unique permission START_ACTIVITIES_ FROM_BACKGROUND, which was at first (Android 10, 11) assigned to the *blacklist* category, but then moved to the *sdk* restriction list. This permission was granted on Android versions 11 where it was restricted as *blacklisted*.

Contradictory documentation. We observe in our analysis that permissions are requested and often granted *regardless of their annotations and context* in apps from both the unofficial and Google Play markets.

For example, 895 permissions that were requested by apps from the unofficial markets are in the 'Removed' section of the

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

TABLE XII
PERMISSION ISSUES IN PRACTICE

Issue Code	Unofficial Markets Apps						Google Play Apps						Google Play Apps (Samsung Devices)				
	# APK	A10	A11	A12	A13	Total	# APK	A10	A11	A12	A13	Total	# APK	A11	A12	A13	Total
"Requested" issues																	
AH	464	903	901	901	895	3600	43	89	95	92	90	366	46	75	98	104	277
AO	117	4	29	29	4	66	7	6	6	6	6	24	7	6	6	6	18
Total	**538**	**907**	**930**	**930**	**899**	**3666**	**46**	**95**	**101**	**98**	**96**	**390**	**49**	**81**	**104**	**110**	**295**
"Granted" issues																	
CB	462	903	899	899	895	3596	43	86	92	89	87	354	46	73	95	101	269
BL	2	4	30	30	4	68	0	0	0	0	0	0	0	0	0	0	0
N3	6	0	2	2	0	4	0	0	0	0	0	0	0	0	0	0	0
RE	1	0	1	1	0	2	0	0	0	0	0	0	0	0	0	0	0
SY	6	4	29	29	4	66	1	0	0	1	1	2	1	1	1	1	3
Total	**462**	**911**	**961**	**961**	**903**	**3736**	**43**	**86**	**92**	**90**	**88**	**356**	**46**	**74**	**96**	**102**	**272**

manifest file which lists permissions removed from the current API. Yet, these permissions were requested and all of them were granted to the analyzed apps. There were three cases of permissions, accompanied by the `@removed` tag in the source code, requested by applications, and none of them were granted. The *backward compatibility* status also seems to be largely ignored, as 98% of such requested permissions were granted to relatively up-to-date apps (unofficial markets).

According to usages, 407 (1%) requested permissions are specifically labelled as *not for third-party applications*, yet 4-30 (1-7%) of them were granted. Moreover, 121 (0.3%) *restricted* permissions were requested by apps (unofficial markets). One was granted in Android 11 and 12.

Overall, the permission annotations and indicated restrictions are largely ignored by third-party developers regardless of the source of apps. However, the apps from Google Play market requested a narrower spectrum of permissions, e.g., the apps requested less permissions from *sdk* group with `@SystemApi` tag.

E. Granted Permissions

Our findings show that the apps from the unofficial markets installed on devices were granted between 24,064 and 24,315 permissions in total, representing 64–65% of the total requested permissions (88-89% out of the requested 'Install' permissions). 30% of the requested permissions were present in 'Runtime' section, which were not expected to be granted on installation. The overall numbers are similar for apps extracted from Google Play market. Our analysis revealed several violations where unexpected permissions were requested by and/or granted to 584 apps in total on installation. The summary of the issues is shown in Tables XII and XIV.

CB issue. We found 3,736 instances of issues in 462 analyzed apps from unofficial markets, i.e., 98% of the requested CB permissions (356 instance in 43 Google Play apps) were granted on Android 10–13. This directly contradicts Android documentation and the Android-defined treatment of conditionally blocked permissions, i.e., handling these permissions is expected to be equivalent to the *blacklist* category for all analyzed apps. For example, `android.permission.USE_CREDENTIALS` permission, theoretically allowed for use in apps that target Android version 8 (API 27) or lower, was granted to 273 apps targeting Android 10 and above. This permission allows an app to use authentication tokens, i.e., login information, to identify a user and perform actions on his behalf (e.g., access Meta, X), without requesting having a user enter password.

BL issue. Another issue concerns **two** apps that were granted the `START_ACTIVITIES_FROM_BACKGROUND` permission *blacklisted* in API 29. The apps were targeting Android 10 and 11 and thus were expected neither to request this permission nor have it granted. This is a sensitive permission, it allows to automatically start activities while minimizing interruptions for the user (e.g., start video recording without asking a user). In Android 13, this permission was moved to the *sdk* category with a `signature|privileged|vendorPrivileged|oem|verifier, + |role` protection level which indicates a vendor-only permission. Yet, it was granted to two third-party apps on Android 11 and 12.

N3 issue. Among the analyzed applications from unofficial markets, we also discovered 68 issues, where permissions, recognized by Android developers as "not for third-party use", were granted to six third-party apps. For example, `INSTALL_PACKAGES`, a *not for third-party* permission that can be used to install other packages, was granted to two apps targeting API 29.

SY issue. This ties into cases where permissions marked as `@SystemApi` and, therefore, expected to be absent from public APIs, were requested and granted to 6 third-party apps (1 Google Play app) on **66** occasions, contradicting Android documentation. For example, `GRANT_RUNTIME_PERMISSIONS` was granted to one app on devices with Android 11 and 12. It belongs to the *sdk* restriction category and is recognized as system-only according to the `@SystemApi` tag. The plain-text comment describes its functionality as "Allows an application to grant specific permissions", which heavily implies that third-party applications are not intended to use it.

All the above-mentioned permissions were requested by applications in their Manifest files, and all were automatically granted by the system without any user consent.

Missing permissions. In our theoretical permission labelling analysis, we encountered 4 permissions listed in the official restriction lists, but not present in any of the four Android code versions we chose for our analysis (refer to Section III).

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

TABLE XIII
UNDEFINED PERMISSIONS IN APKS

App's Target API Level	# APKs	Requested in Manifest	Present in Unified Mapping	Interpreted by Devices							
				A10		A11		A12		A13	
				Requested	Granted	Requested	Granted	Requested	Granted	Requested	Granted
29	2363	21724	***Undefined***	**28**	**0**	**28**	**13**	**889**	**14**	**7875**	**12**
			- present in manifest	*28*	*0*	*28*	*13*	*28*	*14*	*28*	*12*
			- not present in manifest	*0*	*0*	*0*	*0*	*861*	*0*	*7847*	*0*
30	1063	13643	***Undefined***	**17**	**0**	**17**	**0**	**581**	**11**	**4037**	**11**
			- present in manifest	*17*	*0*	*17*	*0*	*17*	*11*	*17*	*11*
			- not present in manifest	*0*	*0*	*0*	*0*	*564*	*0*	*4020*	*0*
31	236	2657	***Undefined***	**0**	**0**	**0**	**0**	**0**	**0**	**632**	**0**
			- present in manifest	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
			- not present in manifest	*0*	*0*	*0*	*0*	*0*	*0*	*632*	*0*
32	19	213	***Undefined***	**0**	**0**	**0**	**0**	**0**	**0**	**28**	**0**
			- present in manifest	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
			- not present in manifest	*0*	*0*	*0*	*0*	*0*	*0*	*28*	*0*

TABLE XIV
GRANTED TO APKS PERMISSIONS WITH ISSUES

Permission	Security Issues	Granted on Devices				APKs Combined	APKs Target API Level			
		A10	A11	A12	A13		29	30	31	32
android.permission.ACCESS_WIMAX_STATE	CB	4	-	-	-	4	4	-	-	-
android.permission.AUTHENTICATE_ACCOUNTS	CB	176	176	176	176	176	98	74	4	-
android.permission.BACKUP	SY,N3	-	2	2	-	2	2	-	-	-
android.permission.CHANGE_COMPONENT_ENABLED_STATE	N3	-	2	2	-	2	2	-	-	-
android.permission.CHANGE_WIMAX_STATE	CB	4	-	-	-	4	4	-	-	-
android.permission.CONNECTIVITY_INTERNAL	SY,N3	-	2	2	-	2	2	-	-	-
android.permission.FLASHLIGHT	CB	113	113	113	113	113	48	61	4	-
android.permission.GRANT_RUNTIME_PERMISSIONS	SY	-	1	1	-	1	1	-	-	-
android.permission.INSTALL_PACKAGES	N3	-	2	2	-	2	2	-	-	-
android.permission.INTERACT_ACROSS_USERS	SY,N3	-	2	2	-	2	2	-	-	-
android.permission.LOCAL_MAC_ADDRESS	SY	-	2	2	-	2	2	-	-	-
android.permission.MANAGE_ACCOUNTS	CB	172	172	172	172	172	98	67	7	-
android.permission.MANAGE_APP_OPS_MODES	CB,N3	-	2	2	-	2	2	-	-	-
android.permission.MANAGE_NETWORK_POLICY	CB	-	2	2	-	2	2	-	-	-
android.permission.MANAGE_USB	SY	-	2	2	-	2	2	-	-	-
android.permission.OVERRIDE_WIFI_CONFIG	SY,N3	-	2	2	-	2	2	-	-	-
android.permission.PEERS_MAC_ADDRESS	SY	-	2	2	-	2	2	-	-	-
android.permission.READ_INSTALL_SESSIONS	SY,N3	4	4	4	4	4	4	-	-	-
android.permission.READ_LOGS	N3	-	2	2	-	2	2	-	-	-
android.permission.READ_PROFILE	CB	52	52	52	52	52	30	22	-	-
android.permission.READ_SOCIAL_STREAM	CB	4	4	4	4	4	4	-	-	-
android.permission.READ_USER_DICTIONARY	CB	8	8	8	8	8	6	2	-	-
android.permission.REBOOT	N3	-	2	2	-	2	2	-	-	-
android.permission.REVOKE_RUNTIME_PERMISSIONS	SY	-	1	1	-	1	1	-	-	-
android.permission.START_ACTIVITIES_FROM_BACKGROUND	BL	-	2	2	-	2	2	-	-	-
android.permission.SUBSCRIBED_FEEDS_READ	CB	6	6	6	6	6	6	-	-	-
android.permission.SUBSCRIBED_FEEDS_WRITE	CB	6	6	6	6	6	6	-	-	-
android.permission.SUBSTITUTE_NOTIFICATION_APP_NAME	SY,N3	-	2	2	-	2	2	-	-	-
android.permission.TETHER_PRIVILEGED	SY,N3	-	2	2	-	2	2	-	-	-
android.permission.UPDATE_APP_OPS_STATS	SY,N3	-	2	2	-	2	2	-	-	-
android.permission.USE_CREDENTIALS	CB	273	273	273	273	273	150	106	17	-
android.permission.WRITE_MEDIA_STORAGE	SY	-	2	2	-	2	2	-	-	-
android.permission.WRITE_PROFILE	CB	7	7	7	7	7	6	1	-	-
android.permission.WRITE_SECURE_SETTINGS	N3	-	2	2	-	2	2	-	-	-
android.permission.WRITE_SMS	CB,(SY*)	25	25	25	25	25	13	11	1	-
android.permission.WRITE_SOCIAL_STREAM	CB	4	4	4	4	4	4	-	-	-
android.permission.WRITE_USER_DICTIONARY	CB	15	15	15	15	15	8	7	-	-
com.android.browser.permission.READ_HISTORY_BOOKMARKS	CB	22	22	22	22	22	15	7	-	-
com.android.browser.permission.WRITE_HISTORY_BOOKMARKS	CB	12	12	12	12	12	7	5	-	-
Total cases		**907**	**939**	**939**	**899**	**947**	**551**	**363**	**33**	**0**

* - permission gained the `@SystemApi` tag at API level 33

In the analyzed apps, we did not see the use of any of these permissions.

F. Controlled Experiment

Some OEMs are known to customize AOSP. Since these customizations may introduce permission management differences, we performed controlled experiments on Samsung devices for apps derived from Google Play market. The results for these apps are given in Table X. We expected the number of the requested by our apps permissions to be similar across devices. We observed noticeable differences between permissions requested by Google Play apps on Samsung devices in our controlled experiment. It appears that the same

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

TABLE XV
GRANTED TO APKs FROM GOOGLE PLAY PERMISSIONS WITH ISSUES

Permission	Security Issues	Granted on Devices				APKs Combined	APKs Target API Level				
		A10	A11	A12	A13		29	30	31	32	33
android.permission.AUTHENTICATE_ACCOUNTS	CB	20	22	22	22	23	-	-	9	2	12
android.permission.FLASHLIGHT	CB	13	15	13	13	15	-	-	6	-	9
android.permission.MANAGE_ACCOUNTS	CB	21	21	21	21	22	-	-	9	1	12
android.permission.READ_PROFILE	CB	9	10	11	10	11	-	-	4	-	7
android.permission.USE_CREDENTIALS	CB	22	23	21	20	23	-	-	9	-	14
android.permission.WRITE_SMS	CB,(SY*)	-	-	1	1	1	-	-	-	-	1
com.android.browser.permission.READ_HISTORY_BOOKMARKS	CB	1	1	1	1	1	-	-	-	-	1
Total cases		**86**	**92**	**90**	**88**	**96**	**0**	**0**	**37**	**3**	**56**

* - permission gained the @SystemApi tag at API level 33

TABLE XVI
GRANTED TO APKs FROM GOOGLE PLAY PERMISSIONS WITH ISSUES (SAMSUNG DEVICES)

Permission	Security Issues	Granted on Devices			APKs Combined	APKs Target API Level				
		A11	A12	A13		29	30	31	32	33
android.permission.AUTHENTICATE_ACCOUNTS	CB	19	23	24	24	-	-	10	2	12
android.permission.FLASHLIGHT	CB	12	15	17	17	-	-	7	-	10
android.permission.MANAGE_ACCOUNTS	CB	16	21	22	22	-	-	9	1	12
android.permission.READ_PROFILE	CB	10	12	12	12	-	-	5	-	7
android.permission.USE_CREDENTIALS	CB	15	22	24	24	-	-	10	-	14
android.permission.WRITE_SMS	CB,(SY*)	2	2	2	2	-	-	1	-	1
com.android.browser.permission.READ_HISTORY_BOOKMARKS	CB	-	1	1	1	-	-	-	-	1
Total cases		**74**	**96**	**102**	**102**	**0**	**0**	**42**	**3**	**57**

* - permission gained the @SystemApi tag at API level 33

set of Google Play apps have been requesting less permissions across almost all groups due to OEM customization (Table X). This maybe an indication that inconsistent Android permission categorization forces OEMs to limit allowed permissions.

G. Undefined Permissions

In our analysis, PChecker encountered applications that requested permissions absent from the unified permission mapping for the API level corresponding to the app's targetSdkVersion. In other words, third-party developers requested permissions that did not exist for the version of the Android API the app was designed for. The summary of these cases is shown in Table XIII.

Among the apps targeting Android 10, 22 apps requested 28 permissions that were not available for API 29. For example, 16 of these apps requested MANAGE_EXTERNAL_STORAGE permission only added to SDK in API 30 (Android 11) [38]. 11 apps requested QUERY_ALL_PACKAGES added in API 30 [39], and one app requested SCHEDULE_EXACT_ALARM introduced two years later in API 31 [40]. Although none of them were granted on Android 10, this exemplifies the fact that third-party developers are oblivious to Android documentation requirements.

Also, 15 applications targeting API 30 requested 17 absent permissions, namely SCHEDULE_EXACT_ALARM, BLUETOOTH_SCAN, and BLUETOOTH_CONNECT. The latter two were introduced in API 31 [28]. 13 of these absent permissions were granted to the apps.

The largest number of permissions appear to be requested on Android 12 and 13 devices. Among these cases though, we identified several instances where the permissions are not present in the apps' Android Manifest files, but added automatically by the OS. For example, BLUETOOTH_SCAN, BLUETOOTH_CONNECT and BLUETOOTH_ADVERTISE were reported as "requested" by 407 apps targeting API 29 and 30 comprising 861 and 570 instances, accordingly. These permissions were introduced in API 31 [28].

Similarly, on the Android 13 device, *all* 3,681 applications (targeting API 29, 30, 31/32) "requested" POST_NOTIFICATIONS permission introduced in API 33. This permission appears to limit all app notifications on Android 13, so it is automatically "requested" by the system on behalf of apps [41]. In comparison, 29 out of 69 Google Play apps with targetSdk 31 and 9 out of 14 Google Play apps with targetSdk 32 requested this permission themselves, and 47 apps had it added automatically on the Android 13 device.

Permissions READ_MEDIA_AUDIO, READ_MEDIA_VIDEO and READ_MEDIA_IMAGES were "requested" by 2,471 out of 3,681 applications. These are finer-grained media permissions that were introduced in API 33 to replace READ_EXTERNAL_STORAGE [42]. While not all of these permissions may be needed, it appears that Android implicitly requests all three permissions on Android 13.

We observe similar behaviour with regards to the BODY_SENSORS_BACKGROUND permission, which was added automatically to 8 apps requesting BODY_SENSORS. This OS behaviour appears to be compatibility-related, as in Android 13, the former permission was introduced for more controlled access [42].

Although the amount of undefined permissions granted is small (less than 1%), these cases show that the OS version

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

of the device plays a major role in the way apps' permissions are treated, introducing even more uncertainty for third-party developers.

> ***Answer to RQ4:*** *Our analysis confirms that discovered permission inconsistencies appear in practice in third-party Android apps from both official and unofficial app distribution markets. Overall, 584 (15,9%) of the 3681 apps had at least one of the inconsistencies. Among them, 4,056 instances were related to permission labelling inconsistencies. This shows that third-party developers routinely request permissions that should not be available to apps, such as 'not for third-party use' or blacklisted permissions. We also discovered 4092 instances of permissions violations in practice where apps requested and were granted permissions which were not expected to be granted according to the documentation and defined restrictions.*

VIII. Discussion

A. Summary

- **Inadequate documentation**. Our analysis shows that the existing and available to developers official documentation is incomplete. Among different issues, we observed permissions and protection levels that are missing from the documentation but present in the code. Similarly, we observed that permission classification described by Android documentation is limited and mostly not conveyed to third-party developers. For example, examining officially published restrictions lists, we noted several undocumented categories that characterize the use of permissions. Our findings are consistent with earlier studies that showed that Android documentation is not always complete, relevant and accurate [10], [43], [44].
- **Conflicting documentation**. We discovered numerous cases of contradictory labelling of permissions, i.e., restrictions applied to permissions are contradictory to other annotations associated with these permissions and the corresponding comments provided by Android developers. Conflicts in permission categorization hinder their proper use by OEMs, Android developers and third-party developers. As our analysis showed, the vast majority of labelling contradictions tend to persist across different versions and are rarely corrected.
- **Disjointed security enforcement.** Access to restricted interfaces is also governed by Android permissions. Our analysis indicates that the expected protections of controlled interfaces are often not consistent with the required permissions. It appears that modifications to one aspect often are not consistently addressed in corresponding modifications to the other.
- **Contradictory use of permissions.** We compared our theoretical mappings to practical execution results on several Android devices and discovered numerous unexpected cases of permissions being granted, contrary to their theoretical attributes. Android put in place conditional restrictions to gradually phase out outdated permissions limiting access by third-party developers to these permissions. In spite of the efforts, these restrictions are not followed and restricted permissions are commonly granted by the Android operating system on devices beyond the target app version.
- **Disregard of restrictions by third-party developers.** Our analysis shows a prevalent trend among third-party developers to routinely request permissions not available to their apps, e.g., due target version of their app or permissions restrictions. We attribute many of these instances to the confusion of developers due to out-of-date documentation and code organization inconsistent with the official documentation. We believe that a significant number of these occurrences are a result of developers being confused by obsolete, incomplete, and contradicting documentation and inconsistent treatment of permissions and interfaces in AOSP that does not align with the official documentation. Similar results were reported by other studies pointing out the use of hidden and unavailable to third-party developers API methods by third-party apps [3], [45]. Our results potentially offer an explanation of the root cause of this phenomena which may be happening due to inconsistent and contradicting documentation rather than developer's intentional misuse.
- **Overprivileged applications.** The problem of overprivileged applications has been extensively researched over the past 10 years [3], [5], [10], [45]. Despite lacking any functionality that necessitates permissions, our testing app was given over 60 unnecessary permissions, demonstrating that this issue remains prevalent today.
- **Silent permissions added by Android OS** Our results revealed that in some cases Android OS requests permissions on behalf of the apps. In many cases, these permissions are more restrictive than the ones requested by apps. This is an interesting finding which may suggest that Android is well aware of problems developers face with permissions and therefore embedded a mechanism to take control of some access on behalf of developers.

B. Implications

For Android development team: *First,* clear and precise documentation is crucial for providing guidance to developers. Our analysis performed from the perspective of a third-party developer, showed that documentation is inconsistent and contradictory, it creates confusion and incorrect use of permissions. Many instances of these problems could be addressed with an improved API documentation. Given a fast evolution of Android APIs, their interfaces and corresponding permissions, we recommend to maintain listing of permission requirements for all APIs with the expected behavior in back compatibility cases. *Second,* the restriction lists were released publicly to provide more guidance to developers. Our study

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

showed that they contain contradicting information and are not providing consistent information even across different releases within one version. Creating one-stop source for permission information and restrictions is likely to reduce inconsistencies creating more transparent view of security enforcement to guide developers. *Third,* there are numerous cases where Android OS grants permissions in contradiction to Android documentation, e.g., in cases of conditionally blocked permissions which handling, as stated by Android, is expected to be equivalent to the blacklist category. This treatment to some extent incentivizes developers to request these permissions regardless of their restrictions. *Forth,* OEM customizations limit the permissions granted on different devices. The developers may be blind to the unexpected absence of some permissions consequently leading to uncertainty in app behavior across devices. Note, vendor-customization of Android images has been known to introduce ambiguity for third-party developers, e.g., resulting in overprivileged apps [46], privilege escalation vulnerabilities in pre-installed apps [47]. A more unified efforts to enforce universally expected treatment of permissions across OEMs can improve developers understanding of permissions. *Fifth,* Android-approved tools for permission verification in third-party apps before their release can help improve the use of permissions by developers and lead to more consistent and expected treatment of permissions.

For Android app developers: To assist the app developers avoid overprivilege and reduce confusion, we constructed unified permission mappings. These mappings can help developers understand how their permissions are evolving between different APIs and provide a comprehensive set of restriction attributes present in Android code and documentation. To automate the process for a given app, PChecker offers a convenient mechanism for developers to verify the existing permissions, their treatment and expected behavior.

For Android device manufacturers: In the Android ecosystem, OEMs customize the Android operating system with their own features. Our PChecker framework is capable of testing and evaluating the permissions issues of apps in response to OEMs customizations.Understanding the impact of variations in permission behaviours due to manufacturer-specific customizations deserves further research efforts.

IX. Related Work

Over the years, there have been numerous studies focusing on Android security in general and permission system in particular. In this section we introduce related work on Android permission system deficiencies and negative impact of Android API evolution.

A. Evolution of Android API

The negative impact of fast evolving Android API has been noted by many studies. Several works [48], [49], [50], [51], [52] explored compatibility issues resulting from rapid changes to the Android platform and its APIs. Syer et al. [53] showed that the dependence on platform APIs was proven to correlate with the emergence of defects in app source code. While Linares-Vasquez et al. [54] confirmed that changes in APIs negatively impact third-party applications (with issues from bugs to crashes after system updates) and cause frustration in users.

The constant evolution of the framework also produces deprecated APIs which present compatibility and maintenance challenges [43]. These changes make the task of keeping complete, relevant and accurate documentation extremely difficult [10], [43]. In 2016, Li et al. [3] found that 5.4% (1269) of third-party apps used hidden API methods, unavailable to third-party developers. Similarly, Li et al. [45] showed that deprecated Android API methods are present in 37,87% of randomly selected 10,000 third-party Android apps. Recently, Liu et al. [44] reported that most third-party apps use silently-evolved interfaces that lack updated documentation, i.e., 957 out of 1,000 analyzed real-world Android apps used at least one of these API methods. A broad analysis of the evolution of non-SDK restrictions in the Android platform was offered by Yang et al. [55]. We, on the other hand, focus specifically on the Android platform source code and documentation changes that affect the analysis of permissions.

B. Android Permission System

The early studies primarily aimed to understand the use of permissions by applications, focusing mostly on documented permissions. Enck et al. [8] proposed to detect malicious Android applications based on requested permissions. Jiao et al. [56] analyzed function calls using sensitive permissions to detect malware. Barrera et al. [9] visualized permissions of 1,100 applications to explore permission usage patterns of applications with similar characteristics. They noted that some permissions provide much broader functionality coverage than others, and proposed a hierarchical permission model for more-limited purposeful access. Similarly, Jeon et al. [57] proposed a more fine-grained permission model with sub-permissions and an approach to infer the proposed permissions for already existing apps.

C. Overprivileged Apps

Overprivileged applications have always posed risks to devices [45], [58]. Stevens et al. [59] pointed out that developers do not follow the principle of least privilege, although the misuse of permissions is less likely to occur when more popular permissions are properly documented. Krutz et al. [60] found that permissions are mostly added to apps in their earlier lifetimes, and, if removed later, done so by developers more experienced than the ones who added them. This means that community expertise mitigates misuse resulting from lacking documentation. Scoccia et al. [61] created a dataset of permission mistakes and fixes in open-source apps to enable further study of this aspect. Our study is complementary to these works. We investigate existing inconsistencies in the official documentation which may further lead to the permission mistakes and overprivileged apps.

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

D. Handling Permissions Concerns

Improper security policy enforcement has been explored at the Android framework level. Sellwood et al. [11] analyzed how permission architecture changes between OS versions and presented an app that activates malicious behaviour after an OS upgrade. AceDroid [62] studied systematic categorization of access control in the Android framework for versions 5–7. Calciati et al. [6] showed that apps may request new dangerous permissions between updates, which could be automatically granted by the OS if the app already possesses a granted permission in the same permission group.

Numerous studies offered solutions to limit overprivileged apps. To estimate the minimum required permissions, Aafer et al. [5] proposed Arcade that using path-sensitive analysis generates a list of permissions an app needs and compares it to the requested permissions. In a similar vein, Dynamo [63] leveraged dynamic instrumentation and gray-box fuzzing to extract permission mappings. Axplorer [2] improved these existing efforts by statically generating a permission map for the Android framework. A static analysis tool FicFinder [64] was designed to detect compatibility issues in Android applications resulting from API changes based on the associated context. Yang et al. [65] proposed a framework to detect instances in apps where they use resources modified between API updates. ACMiner [66] evaluated Android's access control enforcement rules by performing a consistency analysis of authorization checks. He et al. [67] leveraged static analysis to discover inconsistent security enforcement for non-SDK APIs. Kratos [4] used permission mapping to discover vulnerabilities and inconsistent security enforcement by comparing permission requirements through possible access paths.

E. Summary

The vast majority of the mentioned approaches focus on over/under-privileged applications and aim to estimate the need for requested permissions in the existing apps. These studies assume the correctness of the Android documentation, and therefore see overprivileged apps as developers' intentional misuse. We, on the other hand, show that this assumption may not be correct. We focus on the official state of permissions and reveal the inconsistency in functional and descriptive attributes of permissions as they are defined by Android and intended for use by Android developers.

X. Threats to Validity

Internal Validity *First*, our analysis may not be complete for all Android devices. Android offers its platform code publicly through its AOSP repository that includes numerous releases (e.g., mainline, quarterly, generic, developer preview, beta releases) and releases applicable to different types of devices (e.g., vehicle, wearable devices). Each type of release has different objectives and may have different set of permissions. Our analysis focused on the phones devices only. For completeness, we employed only the latest releases (at the time of analysis[4]) of four Android versions. We selected code releases labelled with 'release' which, as indicated by Android, should correspond to main releases for phone devices. As Android evolves and releases new restriction lists, our unified mapping approach can be applied to derive mappings for newer versions.

Second, Our derived permission categorization may not absolutely complete and correct. To ensure completeness, we leveraged official documentation, public restriction lists and Android source code. From the source code, we derived all available permission-related information including declaration of permissions and the corresponding comments left by Android development team. For correctness, we ensured that our categories are built on top of the categorization present in the official documentation and in the Android restriction lists. To gain more insights into specific labelling in the code, we carefully inspected source code comments. We cross-validated our derived categorization with each official category.

External validity Our analysis on the selected apps may not represent the state of all possible third-party apps. To counter this, we collected apps from multiple sources to ensure diversity of app categories and audiences. We collected apps from F-Droid, the largest database of open-source apps, from GooglePlay Store, a well known closed-source distribution platform for Android apps, and other well-known Android communities for sharing apps.

XI. Conclusion

In this study, we analyzed permissions, an essential component of the Android access control system. Our analysis sheds light on how various types of permissions are handled by Android. We created a unified permission mapping and categorized permissions based on both actionable and non-actionable attributes that are often overlooked in official Android documentation.

Our study is the first to provide quantitative characterization of inconsistencies between the official Android documentation and Android platform source code. We show that the categorization provided by Android documentation has been kept incomplete, outdated, and, on many occasions, inconsistent with the treatment of individual permissions in the source code and in OS instances for an extended period of time and across multiple API versions. We further developed PChecker to evaluate discrepancies among permissions requested and granted to applications. Our analysis of 3,681 Android apps showed the presence of over 14,000 issues. The severity of the discovered issues ranges from requesting restricted system permissions to having blacklisted permissions granted to third-party applications. Understanding the impact of these permissions and their patterns across a larger set of apps requires further study.

[4]At the time of analysis Android 14 has not been introduced.

References

[1] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android permission specification," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 217–228.

[2] M. Backes, S. Bugiel, E. Derr, P. McDaniel, D. Octeau, and S. Weisgerber, "On demystifying the Android application framework: Re-visiting Android permission specification analysis," in *Proc. 25th USENIX Conf. Secur. Symp.*, 2016, pp. 1101–1118.

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

[3] L. Li, T. F. Bissyandé, Y. Le Traon, and J. Klein, "Accessing inaccessible Android APIs: An empirical study," in *Proc. IEEE Int. Conf. Softw. Maintenance Evolution (ICSME)*, 2016, pp. 411–422.

[4] Y. Shao, Q. A. Chen, Z. M. Mao, J. Ott, and Z. Qian, "Kratos: Discovering inconsistent security policy enforcement in the Android framework," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016.

[5] Y. Aafer, G. Tao, J. Huang, X. Zhang, and N. Li, "Precise Android API protection mapping derivation and reasoning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 1151–1164.

[6] P. Calciati, K. Kuznetsov, A. Gorla, and A. Zeller, "Automatically granted permissions in Android apps: An empirical study on their prevalence and on the potential threats for privacy," in *Proc. 17th Int. Conf. Mining Softw. Repositories*, 2020, pp. 114–124.

[7] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending Android permission model and enforcement with user-defined runtime constraints," in *Proc. 5th ACM Symp. Inf., Comput. Commun. Secur.*, 2010, pp. 328–332.

[8] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 2009, pp. 235–245.

[9] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to Android," in *Proc. 17th ACM Conf. Comput. Commun. Secur.*, 2010, pp. 73–84.

[10] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 627–638.

[11] J. Sellwood and J. Crampton, "Sleeping Android: The danger of dormant permissions," in *Proc. 3rd ACM Workshop Secur. Privacy Smartphones Mobile Devices*, 2013, pp. 55–66.

[12] G. S. Tuncay, J. Qian, and C. A. Gunter, *See No Evil: Phishing for Permissions With False Transparency*. USENIX Association, 2020, pp. 415–432.

[13] M. Diamantaris, E. P. Papadopoulos, E. P. Markatos, S. Ioannidis, and J. Polakis, "REAPER: Real-time app analysis for augmenting the Android permission system," in *Proc. 9th ACM Conf. Data Appl. Secur. Privacy*, 2019, pp. 37–48.

[14] "Privileged permission allowlisting." Android Open Source Project. Accessed: Dec. 2022. [Online]. Available: https://source.android.com/docs/core/config/perms-allowlist

[15] "Restrictions on non-SDK interfaces." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/guide/app-compatibility/restrictions-non-sdk-interfaces

[16] "Permissions on Android." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/guide/topics/permissions/overview#system-components

[17] "Define a custom app permission." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/guide/topics/permissions/defining

[18] "Define a custom app permission." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/guide/topics/permissions/defining

[19] "Requires permission." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/reference/androidx/annotation/RequiresPermission

[20] "Updates to non-SDK interface restrictions in Android 11." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/about/versions/11/non-sdk-11

[21] "Updates to non-SDK interface restrictions in Android 12." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/about/versions/12/non-sdk-12

[22] "Updates to non-SDK interface restrictions in Android 13." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/about/versions/11/non-sdk-13

[23] "Package index." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/reference/packages

[24] "Improving stability by reducing usage of non-SDK interfaces." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://android-developers.googleblog.com/2018/02/improving-stability-by-reducing-usage.html

[25] "AAPT2 (Android Asset Packaging Tool)." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/studio/command-line/aapt2

[26] "Android debug bridge (ADB)." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/studio/command-line/adb

[27] "UI/Application Exerciser Monkey." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/studio/test/other-testing-tools/monkey

[28] "Manifest.permission." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/reference/android/Manifest.permission

[29] "Manifest.permission." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/reference/android/Manifest.permission#READ_NEARBY_STREAMING_POLICY

[30] "Android developers reference." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/reference/android/R.attr#protectionLevel

[31] "Updates to non-SDK interface restrictions in Android 10." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/about/versions/10/non-sdk-q

[32] "Updates to non-SDK interface restrictions in Android 11." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/about/versions/11/non-sdk-11

[33] "Requires permission: Public fields—Conditional." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/reference/androidx/annotation/RequiresPermission#conditional()

[34] "Restrictions on non-SDK interfaces." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/guide/app-compatibility/restrictions-non-sdk-interfaces#list-names

[35] A. Pham, I. Dacosta, E. Losiouk, J. Stephan, K. Huguenin, and J.-P. Hubaux, "HideMyApp: Hiding the presence of sensitive apps on Android," in *Proc. 28th USENIX Conf. Secur. Symp.*, 2019, pp. 711–728.

[36] A. Foroughipour, N. Stakhanova, F. Abazari, and B. Sistany, "AndroClonium: Bytecode-level code clone detection for obfuscated Android apps," in *Proc. ICT Syst. Secur. Privacy Protection*, W. Meng, S. Fischer-Hübner, and C. D. Jensen, Eds., Cham, Switzerland: Springer-Verlag, 2022, pp. 379–397.

[37] "apksigner." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/studio/command-line/apksigner

[38] "Manifest.permission." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/reference/android/Manifest.permission#MANAGE_EXTERNAL_STORAGE

[39] "Manifest.permission." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/reference/android/Manifest.permission#QUERY_ALL_PACKAGES

[40] "Manifest.permission." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/reference/android/Manifest.permission#SCHEDULE_EXACT_ALARM

[41] "Notification runtime permission." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/develop/ui/views/notifications/notification-permission#new-apps

[42] "Behavior changes: Apps targeting Android 13 or higher." Android Developers. Accessed: Dec. 2022. [Online]. Available: https://developer.android.com/about/versions/13/behavior-changes-13

[43] L. Li, J. Gao, T. F. Bissyandé, L. Ma, X. Xia, and J. Klein, "CDA: Characterising deprecated Android APIs," *Empirical Softw. Eng.*, vol. 25, pp. 2058–2098, 2020.

[44] P. Liu, L. Li, Y. Yan, M. Fazzini, and J. Grundy, "Identifying and characterizing silently-evolved methods in the Android API," in *Proc. 43rd Int. Conf. Softw. Eng. Softw. Eng. Pract.*, 2021, pp. 308–317.

[45] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Permission evolution in the Android ecosystem," in *Proc. 28th Annu. Comput. Secur. Appl. Conf.*, 2012, pp. 31–40.

[46] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang, "The impact of vendor customizations on Android security," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 623–634.

[47] M. Elsabagh, R. Johnson, A. Stavrou, C. Zuo, Q. Zhao, and Z. Lin, "FIRMSCOPE: Automatic uncovering of privilege-escalation vulnerabilities in pre-installed apps in Android firmware," in *Proc. 29th USENIX Conf. Secur. Symp.*, 2020, pp. 2379–2396.

[48] T. McDonnell, B. Ray, and M. Kim, "An empirical study of api stability and adoption in the Android ecosystem," in *Proc. IEEE Int. Conf. Softw. Maintenance*, 2013, pp. 70–79.

[49] L. Li, T. F. Bissyandé, H. Wang, and J. Klein, "CiD: Automating the detection of API-related compatibility issues in Android apps," in *Proc. 27th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2018, pp. 153–163.

[50] D. He, L. Li, L. Wang, H. Zheng, G. Li, and J. Xue, "Understanding and detecting evolution-induced compatibility issues in Android apps," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng.*, 2018, pp. 167–177.

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.

[51] H. Cai, Z. Zhang, L. Li, and X. Fu, "A large-scale study of application incompatibilities in Android," in *Proc. 28th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2019, pp. 216–227.
[52] H. Xia et al., "How Android developers handle evolution-induced API compatibility issues: A large-scale study," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, 2020, pp. 886–898.
[53] M. D. Syer, M. Nagappan, B. Adams, and A. E. Hassan, "Studying the relationship between source code quality and mobile platform dependence," *Softw. Qual. J.*, vol. 23, no. 3, pp. 485–508, 2015.
[54] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "API change and fault proneness: A threat to the success of Android apps," in *Proc. 9th Joint Meeting Found. Softw. Eng.*, 2013, pp. 477–487.
[55] S. Yang, R. Li, J. Chen, W. Diao, and S. Guo, "Demystifying Android non-SDK APIs: Measurement and understanding," in *Proc. 44th Int. Conf. Softw. Eng.*, 2022, pp. 647–658.
[56] H. Jiao, X. Li, L. Zhang, G. Xu, and Z. Feng, "Hybrid detection using permission analysis for Android malware," in *Proc. Int. Conf. Secur. Privacy Commun. Netw.*, 2015, pp. 541–545.
[57] J. Jeon et al., "Dr. Android and Mr. Hide: Fine-grained permissions in Android applications," in *Proc. 2nd ACM Workshop Secur. Privacy Smartphones Mobile Devices*, 2012, pp. 3–14.
[58] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin, "Permission re-delegation: Attacks and defenses," in *Proc. 20th USENIX Conf. Secur.*, 2011, p. 22.
[59] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen, "Asking for (and about) permissions used by Android apps," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, 2013, pp. 31–40.
[60] D. E. Krutz, N. Munaiah, A. Peruma, and M. W. Mkaouer, "Who added that permission to my app? An analysis of developer permission changes in open source Android apps," in *Proc. 4th Int. Conf. Mobile Softw. Eng. Syst.*, 2017, pp. 165–169.
[61] G. L. Scoccia, A. Peruma, V. Pujols, B. Christians, and D. E. Krutz, "An empirical history of permission requests and mistakes in open source Android apps," in *Proc. 16th Int. Conf. Mining Softw. Repositories*, 2019, pp. 597–601.
[62] Y. Aafer, J. Huang, Y. Sun, X. Zhang, N. Li, and C. Tian, "AceDroid: Normalizing diverse Android access control checks for inconsistency detection," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018.
[63] A. E. M. Dawoud and S. Bugiel, "Bringing balance to the force: Dynamic analysis of the Android application framework," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021.
[64] L. Wei, Y. Liu, and S.-C. Cheung, "Taming Android fragmentation: Characterizing and detecting compatibility issues for Android apps," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2016, pp. 226–237.
[65] G. Yang, J. Jones, A. Moninger, and M. Che, "How do Android operating system updates impact apps?" in *Proc. 5th Int. Conf. Mobile Softw. Eng. Syst.*, 2018, pp. 156–160.
[66] S. A. Gorski et al., "ACMiner: Extraction and analysis of authorization checks in Android's middleware," in *Proc. 9th ACM Conf. Data Appl. Secur. Privacy*, 2019, pp. 25–36.
[67] Y. He et al., "A systematic study of Android non-SDK (hidden) service API security," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 2, pp. 1609–1623, Mar./Apr. 2023.

Anna Barzolevskaia received the M.S. degree in computer science from the University of Saskatchewan, Canada.

Enrico Branca is a Researcher with the Department of Computer Science, University of Saskatchewan, Canada. He has been working in information security for over a decade with experience in software security, information security management, and cyber security R&D.

Natalia Stakhanova is the Canada Research Chair in Security and Privacy, and an Associate Professor with the University of Saskatchewan, Canada.

Authorized licensed use limited to: University of Saskatchewan. Downloaded on November 18,2024 at 21:30:38 UTC from IEEE Xplore. Restrictions apply.