Measuring and Characterizing (mis)compliance of the Android permission system

A thesis submitted to the
College of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Anna Barzolevskaia

Permission to Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Disclaimer

Reference in this thesis to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement, recommendation, or favoring by the University of Saskatchewan. The views and opinions of the author expressed herein do not state or reflect those of the University of Saskatchewan, and shall not be used for advertising or product endorsement purposes.

Requests for permission to copy or to make other uses of materials in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building, 110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan S7N 5C9 Canada

OR

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9 Canada

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. In spite of its newly-introduced complexity, misgranted permissions continue to present a significant risk to users.

We present research on theoretical and practical misuse of permissions using our methodology that leverages unified permissions and call mappings. To guide the automated evaluation of permission use and compliance in Android apps, we develop PChecker, a tool that reports permissions requested by and granted to Android devices.

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 use PChecker to analyze the permission use of 3,681 Android apps showing the common prevalence and occasional severity of non-compliance in real-world scenarios.

Acknowledgements

I'm extremely grateful to my supervisor, Dr. Natalia Stakhanova, for giving me the opportunity to study at the University of Saskatchewan, while also guiding and supporting me all the way throughout.

I am also grateful to the amazing people at the Student Wellness Centre for helping me manage my mental and physical health during my most difficult times.

Contents

List of Tables

List of Figures

1 Introduction

Mobile phones 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 has quickly revealed shortcomings 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 [2, 5, 6, 14, 63, 80]. For example, early Android versions did not regulate the use of privileged resources, allowing any application (app) to declare permissions and gain access to any data or functionality on the device. This resulted in privacy leaks [75], unrestricted access to advanced functionalities [8, 46], overprivileged apps that possess unnecessary access to resources [47] and exploitable weaknesses [79]. Allowing users to grant or reject permissions requested by an app did not solve the security problems. As studies showed, users continue to have a vague understanding of what permissions should or should not be granted to apps [84].

The Android permission system has evolved significantly to a more regulated and fine-grained permission model. With Android 9, restrictions on non-SDK (Software Development Kit) 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, testing, and temporary interfaces.

Despite a more sophisticated access system, the official documentation for Android permissions, including their functionality, requirements, and use cases, has been historically lacking (e.g., [47]), with the permissions' difference between versions largely undefined or vaguely stated. This lack of information leads to ambiguity, which creates the potential for misuse and improper implementation, ultimately resulting in security risks.

In this research, we analyze permissions in Android versions 10–13 from theoretical and practical perspectives. We investigate the differences between the official Android guidelines for permissions versus their implementation in the Android platform source code. We further explore how these discrepancies appear in practice in Android apps created by third-party developers. To understand the current practical state of permissions, we analyze the source code of four Android versions, 10–13 (APIs 29, 30, 31, 33), extracting permissions and their properties as noted by the Android developers, as well as some elements guarded by permissions. We further complement this information with the official Android documentation and restriction information for the Android platform to create unified permission and guarded call mappings that identify theoretical requirements of permissions and calls. These mappings simplify the analysis of protec-

tions provided by the Android permission system and reveal existing inconsistencies between permissions and calls.

To further understand the presence and scope of inconsistencies in practice, we designed and developed a tool, PChecker, that leverages our mappings to review the permissions requested by and granted to Android apps in comparison to the utilized SDK and host devices.

In summary, this work presents three contributions:

- *We derive a unified permission mapping that enables domain experts to analyze the consistency of protections provided by the Android permission system.* We show a mostly hidden internal categorization of permissions adopted by Android developers. Our analysis illustrates that the internal permission categorization is more complex than what is conveyed by the official Android documentation, which is incomplete, inconsistent, and sometimes contradictory with the actual implementation. For example, we discover permissions present in the Android platform source code but absent from any documentation. We identify thirteen types and 1,388 cases of contradictions between the official documentation and the Android platform source code. This is the first work to systematically outline the existing categorization and uncover security inconsistencies between the Android documentation and its permission system implementation that persist across 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 instances of discrepancies between the SDK (and non-SDK) interface restrictions and the corresponding permission restrictions across the analyzed Android versions.

- *We develop PChecker to evaluate permissions requested and granted to third-party Android apps.* We conduct an analysis of 3,681 apps, discovering 3,666 instances of discrepancies related to permissions requested by apps, and another 3,736 discrepancies with permissions automatically granted to apps. Overall, we found at least one of these issues in each of 538 apps.

Our findings highlight alarming patterns and shed light on the existing misleading and contradicting guidance given to Android app developers. This status quo emphasizes the challenges in reality for creating fully permission-compliant apps as demonstrated by the pervasiveness of permission non-compliance in Android apps. **Complete and relevant documentation falls behind the rapid development of the Android platform, introducing implementation ambiguity, disjointed security enforcement, compatibility issues and practical oversights that hinder Android development, third-party application development and relevant research.**

All the code used to create the unified mappings and the PChecker tool are available on the Github page: https://github.com/thecyberlab/pchecker.

2 Background and related work

Before delving into inconsistent and contradictory permissions structure in Android, we take a moment to explore Android, its structure, risks, and other works in studying permissions. Knowing this illuminates the novelty and value of our work.

2.1 Background

Android is an operating system specifically designed for mobile devices such as tablets and smartphones. It is based on a modified version of the Linux kernel and is mainly developed by Google under the name Android Open Source Project (AOSP) [51].

Android applications (apps) are distributed in the .apk file format. Each .apk file, when decompressed, includes one or multiple .dex files that contain the logic of an app in the form of Dalvik bytecode executed using an Android-Runtime environment (ART). The managed (virtualized) Dalvik bytecode execution environment is the primary engine for Android applications. The Android application programming interface (API) is the set of Android platform interfaces exposed to applications running in the managed runtime environment. The metadata of the app used by the Android OS is contained in the AndroidManifest.xml file found in the .apk.

The Android OS is iterated through major versions expressed as real numbers at first (for example, Android 4.2). Each major change in the Android OS is referred to by its index number called API (Application Programming Interface) level, or SDK version, starting at 1 and, at the time of writing, ending at 33 (Android 13.0). Each major version also has a code name (and code letters) that follows an alphabetic pattern (Android 8 is Oreo, or O). All these denominations are used by Android in the documentation and source code, and, consequently, are used throughout this work. The naming reference for the relevant versions is showcased in Table 2.1.

As OS architecture, logic, and available functionality change between Android versions, so do the requirements for applications suited to install and run on Android devices. It would be impractical for every app to be optimized for each of the 33 API levels. The different version denominations are shown in Table 2.1 As such, apps are expected to be built for a specific range of API levels. This range is expected to be displayed in the Manifest file via `minSdkVersion`, `targetSdkVersion` (or `compileSdkVersion` when used during app compilation) and `maxSdkVersion` fields [32]. These fields determine the expected configuration aspects of the APK, such as permissions, features, etc. In practice, however, the boundaries `minSdkVersion`

Table 2.1: Denominations of Android versions

Android Version	API Level	Letter Code	Release Year
6	23	M	2015
7	24,25	N	2016
8	26,27	O	2017
9	28	P	2018
10	29	Q	2019
11	30	R	2020
12	31,32	S	2021,2022
13	33	T	2022

and `maxSdkVersion` may be omitted when building the application, as testing the application on different versions requires resources and these values may limit the advertisement of the app on excluded OS versions. In many cases, the Manifest file contains only the `targetSdkVersion`.

Android permissions is a system of safeguards implemented as an Access Control List (ACL) [8] 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. Android 8 introduced the allowlist [78], 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 [35]. 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 `signature|privileged` (formerly `SignatureOrSystem`) that regulate access to privileged resources and are allocated for apps installed in the Android system image.

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 [31]:

- 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.
- 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.
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 [52]. In this work, we focus specifically on Android-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` [28]. For example, `android.permission.SYSTEM_CAMERA`.

Third-party developers are expected to declare a list of required permissions in the AndroidManifest.xml file, otherwise their app will not gain access to the resources of the device. The corresponding APIs that govern access are then invoked by the app if the necessary permissions are granted.

With this understanding of Android permissions, we can now examine previous work on recognizing the limitations of the system.

2.2 Related Work

Since the creation of Android in 2008, there have been numerous studies focusing on Android security in general and vulnerabilities of the existing access controls in particular. We report on seven general approaches that researchers pursued regarding Android security in general and permissions in particular:

- Earlier works reviewed the advantages and drawbacks of the permission model implemented by Android.
- Attention was also given to how end-users perceive permissions and react to permission requests.
- Numerous studies pointed out that developers struggle to keep up with the rapid API changes, resulting in compatibility issues.
- Applications tend to request more permissions than they actually need, expanding the attack surface.
- This problem escalates on vendor-customized Android images.

- The complex, unstable and flexible nature of Android is constantly exploited by malicious apps, which employ diverse techniques to force users to reveal sensitive data.
- The Android platform code itself has been extensively analyzed for security loopholes.

The early studies primarily aimed to understand the use of permissions by applications, focusing mostly on documented permissions. Enck et al. [46] proposed to detect malicious Android applications based on requested permissions. Jiao et al. [59] analyzed function calls using sensitive permissions to detect malware. Barrera et al. [8] 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. [58] proposed a more fine-grained permission model with sub-permissions and a tool to infer the proposed permissions for already existing apps. Book et al. [12] discovered that the use of dangerous permissions in libraries and apps increases over time. Additionally, advertisement libraries could abuse the permissions of the host apps to gather sensitive data, such as information about the device, network, location, etc. [50].

Many mobile users were found to be unaware of privacy implications concerning app permissions [61]. Wijesekera et al. [88] examined user attitudes towards unexpected permission requests made by apps and found that users would block one-third of all permission requests if given the ability to do so. Another study by Bonné et al. [11] explored user behaviour when presented with runtime permission requests. They found that, overall, 84% of requests were granted, and that users denied permissions they thought the app did not need to achieve the desired functionality and granted permissions based on their perception of the app's functionality. Moreover, out of all permissions granted, 10% were described as grudgingly accepted. Bianchi et al. [10] highlight that Android lacks GUI confirmation of the origin of an app, which allows malicious applications to mimic legitimate apps and confuse users with phishing (social engineering) and click-jacking (user interface manipulation) attacks into giving away sensitive information. They also propose market-level and on-device defence tools to mitigate such attacks.

In an attempt to provide user-controlled limitations on applications, Nauman et al. [75] developed Apex, an extension to the Android permission framework that allowed the user to define application runtime constraints. Lin et al. [65] proposed user privacy controls by determining the purpose of requested permissions in third-party apps and enabling the user to make more informed security decisions. Eling et al. [44] modified runtime permission requests to be more informative about the app's intentions and proved that informed users are more likely to deny permission requests. Later, Liu et al. [69] conducted a field study where Android users interacted with a Personalized Privacy Assistant which offered privacy-related suggestions on application permissions. Peddinti et al. [76] proposed a comparative mechanism for Google Play, which would evaluate permissions requested by an app against functionally similar apps and identify permissions that those apps rarely request. By presenting unnecessary, according to the competition, permissions, they nudged developers to remove such permissions.

Improper security policy enforcement has been explored at the Android framework level. Sellwood et al. [79] analyzed how permission architecture changes between OS versions and presented an app that activates malicious behaviour after an OS upgrade. AceDroid [1] studies systematic categorization of access control in the Android framework for versions 5–7. Several papers [13, 54, 62, 74, 90] explore compatibility issues resulting from rapid changes to the Android platform and its APIs. The extent of the dependence on platform APIs was proven to correlate to the likelihood of the emergence of defects in application source code [81]. The time spread of these publications (2013–2020) also indicates that these issues persist over Android's entire existence. The constant evolution of the framework also produces deprecated APIs which present compatibility and maintenance challenges [64]. These changes also make the task of keeping complete, relevant and accurate extremely difficult [47, 64]. Besides, changes in APIs often negatively impact third-party applications, which cause frustration in users from bugs and crashes after system updates [66]. Liu et al. [70] recently reported that most third-party applications use silently-evolved interfaces that lack updated documentation. With Android documentation severely lacking and obsolete, developers turn to forums and message boards for answers to questions about API changes [67]. Solutions were also introduced for automated compatibility-issue discovery.

Overprivileged applications have always posed risks to devices. Felt et al. [48] demonstrated that a permission granted to one application may be re-delegated to another application, which allowed the first application to perform privileged actions for the second app, so a single overprivileged application on a device may be exploited by malicious apps, expanding the attack surface. The dangers of overprivileged applications were studied by Wei et al. [87]. They analyzed how Android permissions are evolving over versions and concluded that the permissions system expands to accommodate access to newly-implemented functionality while keeping older permissions with broad access unchanged. They also noted that developers do not follow the principle of least privilege. A concerning example was presented by Fratantonio et al. [49]. An app with two permissions, `SYSTEM_ALERT_WINDOW` and `BIND_ACCESSIBILITY_SERVICE` could take control of the UI feedback loop and execute major attacks without the user's knowledge, such as stealing passwords and PINs, enabling permissions, intercepting 2FA tokens, accessing the Internet, etc. This vulnerability was a design issue and could not be quickly addressed within the AOSP, as it exploited the design flaws of core Android components.

The goal of breaking down executable code can be achieved via static and dynamic analyses. The static analysis does not perform code execution and focuses on the structure and logic of the written code and explores all possible routes the program could take. It finds issue occurrences to identify problems. In contrast, the dynamic analysis explores all the outcomes the program achieves during execution on runtime. It identifies problems to interpret issue occurrences. Felt et al [47] introduced the Stowaway tool to detect overprivileged applications targeting Android 2.2. Stowaway tracks the API calls made by an application to determine the full set of required permissions. Due to the dynamic nature of the analysis, Stowaway produces an incomplete set of permissions. PScout [5], on the other hand, used pure static analysis to

identify all actions that require permissions and therefore generated a more complete permissions map, i.e., correspondence between API calls to specific permissions in Android versions 2.2–4. Similarly, using path-sensitive analysis, Arcade [2] attempts to generate a list of permissions an app needs to estimate the minimum required permissions, then compares the list to developer-requested permissions. Axplorer [6] improves these existing efforts by statically generating a permission map for the Android framework. A static analysis tool FicFinder [86] was designed to detect compatibility issues in Android applications resulting from API changes based on the associated context. Yang et al. [91] proposed a framework to detect instances in apps where they use resources modified between API updates. Dynamo [16] leveraged dynamic instrumentation and gray-box fuzzing to extract permission mappings through path-sensitive analysis.

Moreover, many Android images are vendor-customized, which introduces even more ambiguity to the security state of devices. Wu et al. [89] analyzed applications on devices customized by multiple vendors. They reported that the majority of pre-installed applications were found to be overprivileged as a direct result of vendor customization, which is also a major reason for newly-introduced vulnerabilities for most vendors.

A static analysis system FIRMSCOPE proposed later by Elsabagh et al. [45] was also used to uncover privilege escalation vulnerabilities in pre-installed apps. BigMAC [57] uses static firmware and Android domain knowledge to model the security policy state of a running Android image to develop attack queries and identify vulnerable interfaces. A separate problem of custom permissions being treated the same way as system permissions was highlighted by Tuncay et al. [82]. They proposed Cusper, a modular permission system, which prevents the abuse of custom permissions to gain access to protected resources.

In the following works, a "permission map" implies call-to-permission relationships, that is, which permissions are required to access a particular interface. This concept corresponds to our *guarded call mapping*.

Third-party application marketplaces have been historically proven to host repackaged legitimate apps with added backdoors and malicious payloads [94]. Nevertheless, users assume that marketplaces analyze and reject malicious applications [61]. Although this assumption became more or less true in recent years, with Google Play restricting applications that target lower API levels and can abuse eased security requirements introduced for compatibility purposes [42], malicious applications that exploit more recent vulnerabilities may still be uploaded to the official markets [49], and third-party marketplaces are subjected to many fewer restrictions and scrutiny.

There have been many works that attempted to detect vulnerable and malicious third-party applications. CHEX [71] performs static analysis of apps with modelling asynchronous execution to detect hijack-enabling flows that introduce hijacking vulnerabilities where improper access control implementations expose sensitive recourses. FlowDroid [4] performs static context-sensitive taint analysis that ensures proper user input filtering in Android applications. Boxify [7] is a sandboxing environment created for app virtualization and testing, which does not require any modification of the tested app. Wang et al. [85] created LibCage to mitigate permission abuse by third-party libraries. Karbab et al. [60] used patterns in API method calls to

flag malicious Android applications. Pham et al. [77], contrariwise, attempt to isolate sensitive applications from outside access by presenting a container app that prevents information gathering employed by other packages. Similarly, MockDroid [9] was an early attempt to artificially limit application access to sensitive resources. Calciati et al. [14] report 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.

One of the challenges in detecting malware in applications is determining whether a security-sensitive action is benign and corresponds to core app functionality, or malicious in nature and is masked as benign. To solve this problem, a static analysis approach implemented in AppContext [93] evaluates the context of security-sensitive behaviours, such as events and conditions that trigger them. For example, an activity that only triggers at night has the potential to be malicious, as this could be an attempt to miss the user's view.

Over time, security research shifted to focus on the Android platform itself. Centaur [72] was a system designed for symbolic execution of the Android framework to discover vulnerabilities. After the update that added the runtime permissions, critiques of the Android permission model were published by Alepis et al. and Tuncay [3,83]. Chang et al. [15] discovered that implicit service invocations forbidden by Android due to the risk of hijacking attacks are still present in popular apps 30 months later. ACMiner [53] evaluates Android's access control enforcement rules by performing a consistency analysis of authorization checks by comparing them between possible service entry points. JGREAnalyzer [56] was created to detect, via call graph analysis, APIs vulnerable to JGR (Java Native Interface Global Reference) exhaustion DoS attacks after Android attempted to limit interface access with non-SDK restriction lists, expected to limit JNI reflections (indirect access to restricted resources trough intermediate services). FANS [68] employed automated generation-based fuzzing to detect vulnerabilities in Android system services. He et al. [55] leveraged static analysis to discover inconsistent security enforcement for non-SDK APIs. Kratos [80] leveraged permission mapping to discover vulnerabilities and inconsistent security enforcement by comparing permission requirements through possible access paths. Our work does not perform execution analysis but studies the state of available Android documentation and its interactions with practical implementations of security enforcement. We compare the treatment of code elements according to permissions and restriction lists: we find *a subset* of interfaces directly intended to be accessed with particular permissions and evaluate the consistency between restriction lists assigned and permissions required.

The vast majority of the mentioned approaches aim to estimate a permission map allowing one to determine over/under-privileged applications. The Android security model was formally outlined by Google developers [73], however, the section focusing on permissions themselves is rather brief and, therefore, incomplete. Permissions in these cases are analyzed as individual requirements for API access. We, on the other hand, focus on the inconsistency in functional and descriptive attributes of permissions as they are defined by Android and allowed for use by third-party developers. The evolution of non-SDK restrictions in the Android platform was previously studied [63,92]. We, therefore, limit our focus to the AOSP source code

and documentation changes that affect the analysis of permissions.

3 Android mappings framework

This chapter outlines the theoretical and practical frameworks constructed with the goal of characterizing Android permissions, characterizing interfaces that are guarded by permissions and analyzing which permissions are requested by and granted to third-party applications recently collected in the wild.

Close inspection of the available Android documentation and Android API source code suggests that the official documentation describing Android permissions and interfaces available for third-party developers is limited and often contradictory [70]. We therefore empirically determine the available Android system permissions and interfaces. We leverage three sources: Android source code for four API versions; official documentation; and Google restrictions lists. We parsed the source code of four Android versions 10–13:

- android10-s3-release (API 29);
- android11-s1-release (API 30);
- android12-s5-release (API 31);
- android13-s3-release (API 33).

We extracted permissions from the system's Android Manifest file [29], within Android Open Source Project (AOSP). Note, that version 12.1 (API 32) is treated similarly to API 31 as we observed no major changes compared to API 31 and no restriction lists were offered by Google. The flow of the analysis is shown in Figure 3.1, and is explained below.

3.1 Building the unified permission mapping

Since third-party developers are expected to declare a list of required permissions in the AndroidManifest file, we identified and extracted string literals following the permission naming convention.

We processed AndroidManifest files retrieved from the four latest AOSP snapshots for each Android version 10–13. For that, we wrote our own Python code that extracted the permissions and comments present in the source code by following the XML file structure demonstrated in Figure 4.1.

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 in the Android OS [35]. In this context, an element is either an interface or a variable defined in the AOSP source code. These lists seem to be generated

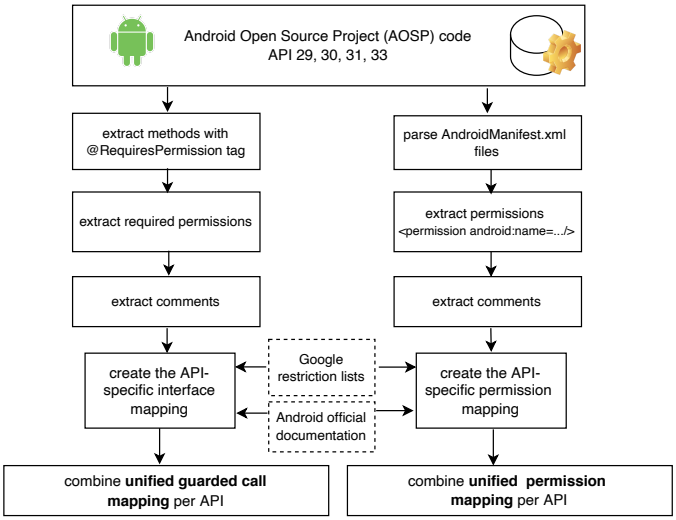

Figure 3.1: The flow of the analysis

automatically during code compilation and each individual file represents only a subset of the officially posted non-SDK restriction lists. This mapping produced a theoretical documented categorization of permissions from an official perspective available as a guide to third-party developers.

We further complemented this mapping by deriving attributes that are actually present and/or described within the API source code. The attributes and their categories are described in Section 4.1.

We manually verified and corrected when needed the assignment of permissions to the categories (Section 4.1) produced by our analysis. This step took us about 18 hours to complete: correctly identifying and recording the usage attribute (to whom a permission is expected to be available) from the plain-text comments proved to be the most time-consuming task. We further manually supplemented the collected permissions with information from the public documentation if it was available.

This process produced 775 individual permissions for all four analyzed APIs, 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 [29].

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*. An example of an entry is displayed in Appendix A. This unified mapping represents a theoretical (based on established ground truths) view of permissions and the protections they intend to provide across Android API versions.

3.2 Building the unified guarded call mapping

Android permissions guard access to the SDK and non-SDK elements available within the Android platform. To further understand how the theoretical permission protections are applied by the Android platform, we

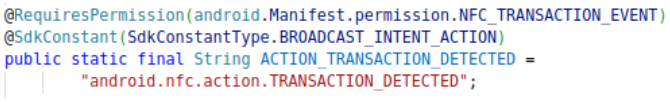

Figure 3.2: An example of a variable protected by `@RequiresPermission` tag

leveraged static analysis, as explained below, to inspect requirements for the SDK and non-SDK elements protected by permissions. The results were applied to the restriction lists featuring calls to the SDK and non-SDK elements with their corresponding assignment to the restriction categories.

Specifically, we extracted all elements present in the restriction lists. We then parsed four versions of the Android platform source code (API levels 29, 30, 31, 33) and retrieved all elements that were accompanied by the `@RequiresPermission` tag [33]. According to documentation, this tag indicates that the following element requires permissions if its declaration is preceded by this tag. An example of this tag usage is presented in Figure 3.2. 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 it expects to receive if it is a method, and the path to the file it was found in, starting from the base of the build directory. We comprised this list retroactively while attempting to connect the elements to their appearances in the restriction lists, the process for which is described further. We wrote our own Python code for this task to extract the relevant strings by leveraging Java syntax rules.

Having the source code elements that require permissions, we had to find their corresponding interfaces in the restriction lists. 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. The following interpretation was inferred by manually comparing some restriction lists file lines to their source code counterparts. Each line of a restriction list file is of the two following structures for methods and variables:
relative_path_to_file $ *(dollar sign)* **optional_parent_class** ;–> *(semi-colon, dash, more-than sign)*
element_name *OR* <**init**> *(less-than sign, "init", more-than sign) if class initialization*
(input_parameter_types) *(types in brackets) if method OR* **:** *(colon) if variable* **output_value_types** *if method OR* **value_type** *if variable* **,** *(comma)* **restriction lists** *separated by (comma)* (without any of the spaces and comments in *italic* added for reading convenience).
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. Following this logic, we attributed each of the retrieved source code elements to a single restriction lists file line, which allowed us to transitively connect permission requirements to interface calls.

The matched methods were further labelled with the corresponding restriction categories. We also collected alternative calls that developers introduced over the versions to gradually replace blocked functionality with safer public alternatives. These were obtained from changes to restriction lists published by Google [38–40], where an alternative to the blocked element was mentioned in plain-text comments. The collected methods were annotated with alternative calls if present and the corresponding permissions required for their invocations. This final mapping comprised the *unified guarded call mapping*. An example of an entry is displayed in Appendix B.

In Chapter 4, we further analyze these unified permission and call mappings to reveal inconsistencies and conflicts in the protections expected to be provided by the Android platform.

3.3 An approach for automated app analysis: PChecker

To facilitate automated analysis of apps using our derived mappings, we built the PChecker tool to identify apps with unexpected use of permissions for a given APK. PChecker retrieves all app permissions and translates them to our permission mappings according to the Android app target version designated by `targetSdkVersion`. The analysis is performed in two stages: (1) permission labelling compliance and (2) a practical consistency check that examines the permissions requested by an app and those that are granted to it in practice.

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 [24].

For the practical consistency 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 [25]. 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 [37]. 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`. Since many APKs lack one or more `SdkVersion` boundaries, we only consider the `targetSdkVersion` value as the targeted API level. In cases 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. For our evaluation, we used the following devices:

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

In its analyses, PChecker evaluates app permissions and how they are handled on different Android devices according to the theoretical permission conflicts we discovered through analysis of the unified permission and call mappings.

4 Findings of theoretical analysis

Here, we describe, in detail, our methodology for mapping (and later evaluating) Android permissions, individual and structural changes that affected permissions across Android versions, and give examples of outdated information we found in official Android documentation, inconsistencies in categorization and access limitations between permissions and the elements protected by them.

4.1 Retrieved permission categories

Our analysis revealed that the Android platform contains a more sophisticated permission classification than indicated by 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.

In our analysis, we could classify permissions by the following attributes: **Restriction**, **Tag**, **Introduced**, **Deprecated**, **Protection**, **Type**, **Status** and **Usage**. We will go through these one-by-one in this Chapter. An example of the source of these attributes is shown in Figure 4.1

4.1.1 Restriction

Starting from Android 9 (API 28), Android began restricting which non-SDK API interfaces third-party 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 [35]. Each restriction list includes permissions with the corresponding interfaces and restriction categories. Officially, the restriction lists include the following categories:

- 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;
- 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.

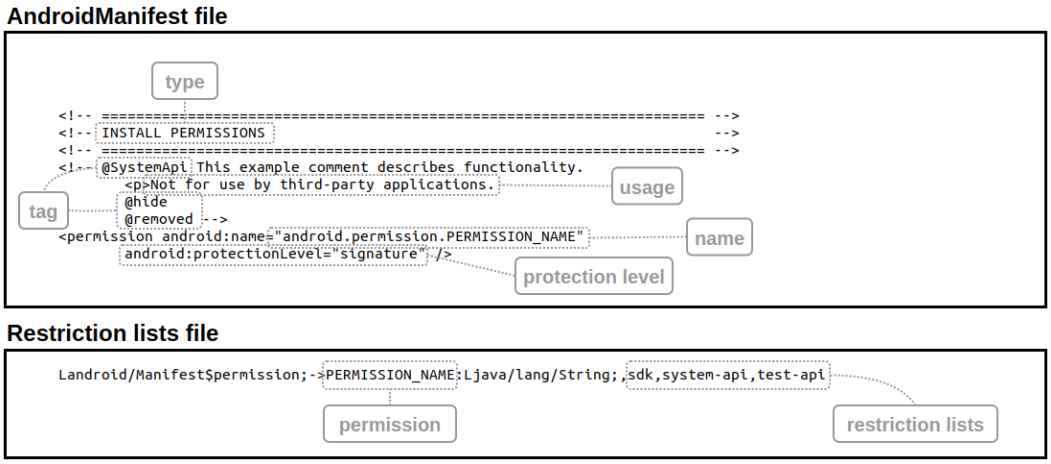

Figure 4.1: Android permission annotations in AndroidManifest.xml files and restrictions lists.

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 [29];
- sdk — SDK interfaces, the available documentation does not explain how these differ from other groups, replaced whitelist starting at API 31;
- system-api — non-SDK interfaces, used by system apps and services;
- test-api — interfaces used for 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 — an unclear label that 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 4.1 to facilitate the analysis. Each category represents certain list combinations pertaining to permissions.

- 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";
- 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";

Table 4.1: 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, API 26,27)	*139*	*131*	*130*	*124*
max P (A9, API 28)	*0*	*0*	*0*	*0*
max Q (A10, API 29)	*0*	*0*	*0*	*0*
max R (A11, API 30)	*0*	*0*	*3*	*3*
unsupported	5	4	1	0
missing	0	1	0	0
Total	533	590	689	765

- 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.

4.1.2 Tag

"Tags" are annotations that describe the intended usage of permissions and are primarily used for automatic API parsing of the source code. Although some tags are referenced in the official documentation, there is no official documentation explaining the functionality of these tags. Android source code contains the following tags (accompanied by unofficial explanations):

- `@hide` — indicates permissions that are excluded from the public SDK API and consequently should not be used 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;

- `no tag` — the absence of a tag indicates that a permission is a part of the public SDK and is accessible by all developers.

Permissions may be annotated with these tags in the source code, although this seems to be optional.

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.

4.1.3 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, i.e. "from time immemorial". However, we could infer some introduction API levels during our manual inspection and comparison of permissions collected from the source code of different versions of Android. For example, comparing adjacent versions gave us an understanding of which permissions were likely introduced in the newer 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 to the OS 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) [30] 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.

4.1.4 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 plain text 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.

4.1.5 Protection

"Protection" 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 [26]:

- `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 [26]. 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'.*

- `system` — an undefined system protection level.

There is no information about its functionality, and it appeared in API 31–33 source code. The permission is `android.permission.SYSTEM_CAMERA` and has a protection level: `system`|`signature`|`role`.

4.1.6 Type

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

- 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.

In practice, however, we found Install permissions with a `dangerous` protection level and Runtime permissions with protection levels `normal` and `signature`. Such inconsistencies do not necessarily lead to flawed security enforcement, but create ambiguity for both third-party and Android developers, complicating development.

4.1.7 Status

"Status" is inferred from the Android API comments indicating the universal status of each permission. Generally, if the status is not specified, a permission is considered available for use by developers. As such, we identified 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.

4.1.8 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.

4.2 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 4.1 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

Table 4.2: Individual permission transitions between restriction list categories

Android 10		Android 11		Android 12		Android 13	# Cases
public	→	public	→	public	→	public	158
sdk	→	public	→	public	→	public	1
sdk	→	sdk	→	sdk	→	public	1
sdk	→	sdk	→	sdk	→	sdk	203
blacklist	→	sdk	→	sdk	→	sdk	2
blacklist	→	blacklist	→	sdk	→	sdk	2
blacklist	→	blacklist	→	blacklist	→	sdk	3
blacklist	→	blacklist	→	blacklist	→	blacklist	15
blacklist	→	blacklist	→	none	→	none	2
blacklist	→	none	→	none	→	none	2
conditional block o	→	public	→	public	→	public	1
conditional block o	→	sdk	→	sdk	→	sdk	5
conditional block o	→	conditional block o	→	sdk	→	sdk	1
conditional block o	→	conditional block o	→	conditional block o	→	sdk	6
conditional block o	→	conditional block o	→	conditional block o	→	conditional block o	123
conditional block o	→	none	→	none	→	none	2
unsupported	→	blacklist	→	blacklist	→	blacklist	1
unsupported	→	unsupported	→	conditional block r	→	conditional block r	3
unsupported	→	unsupported	→	unsupported	→	sdk	1
none	→	public	→	public	→	public	6
none	→	sdk	→	sdk	→	sdk	30
none	→	blacklist	→	sdk	→	sdk	1
none	→	blacklist	→	blacklist	→	sdk	2
none	→	blacklist	→	blacklist	→	blacklist	20
none	→	blacklist	→	none	→	none	1
none	→	none	→	public	→	public	17
none	→	none	→	sdk	→	sdk	58
none	→	none	→	blacklist	→	public	1
none	→	none	→	blacklist	→	sdk	2
none	→	none	→	blacklist	→	blacklist	25
none	→	none	→	none	→	public	21
none	→	none	→	none	→	sdk	41
none	→	none	→	none	→	blacklist	14

code, which implies that they can only be granted to system apps. This intuitively contradicts the available documentation that associates the *sdk* category with accessible development tools (as noted in Section 4.1.1).

As Table 4.1 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).

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 4.5). 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 security 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, permissions `WIFI_UPDATE_USABILITY_STATS_SCORE` and `WIFI_SET_DEVICE_MOBILITY_STATE` 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 [22, 23].

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 higher Android versions.

Across the four major Android versions incorporating restriction lists, we have identified one permission present in the AOSP source code, but missing from the restriction lists, we labelled it as *missing* in Table 4.1. The permission missing from restriction lists of Android 11 is `android.permission.ADD_TRUSTED_DISPLAY`. It was added to the restriction list in Android 12 as *blocked*.

The overall analysis of how the assignment of individual permissions to the corresponding categories of restriction lists has evolved across Android versions is shown in Table 4.2. The value *none* means that a permission is not present in the restriction list of the corresponding Android version. We identified 173 such permissions. As the results show, some categories have been restricted over time (e.g., *unsupported*→*conditionally blocked*) and some were loosened (e.g., *blacklist*→*sdk*). These have the potential to open security and privacy loopholes leading to uncertain protection of app resources (we leave this investigation for future work). For example, when an app targets a version of Android where the permission is blacklisted but is installed on a device with an Android version supporting that permission, then the app may consequently grant access to the resource.

Table 4.3: Combinations of permission attributes

Combination by 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**

4.3 Permission-labelling analysis

While permissions are labelled with all of the attributes we derived, only three attributes present actionable mechanisms: **Restriction**, **Protection**, and **Tag**. Restriction lists and Protection levels are checked against declared app permissions at the OS level, and Tags are used to generate API documentation. **Status**, **Usage** and **Type** do not appear to be enforced in practice.

We derived combinations of permissions along three attributes, **Restriction**, **Protection**, and **Tag**, present in Android 10–13 source code (Table 4.3).

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 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.

As mentioned in Section 4.1.5, 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 also 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*. They 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, while missing permissions suggest inadequate documentation.

Non-existent permissions. We identified one permission in the source code manifest files that is not present in the restriction lists (its restriction category is labelled as *missing*).

Vice versa, we found permissions given in the official restriction lists that are not present in any of the manifest files 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*. The lack of uniform presence of permissions in publicly available and documented sources poses a significant risk of them being misused by developers.

4.4 Permission-labelling conflicts

Our analysis revealed that some attribute combinations provide redundant or inconsistent labelling as a result of official documentation conflicting with the source code. We identified the following thirteen types of conflicts[1]:

- (**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;

[1]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 4.4: Discovered permission labelling conflicts

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

- (**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*;

We discovered 1,388 cases of labelling contradictions in total between the official documentation and the Android platform source code. The summary of these conflicts for each of the analyzed Android versions is listed in Table 4.4.

As our analysis showed, *the vast majority of these contradictions tend to persist across different versions and are rarely corrected*. For example, the number of **NR**, **SR** and **DC** conflicts increased slightly over time, with all discrepancies present in older versions continuing to subsequent releases.

The **BG** conflicts, 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** conflict, 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 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 contradictions with **ET**, **PT**, **IG**, **OG** conflicts 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** conflict.

Similarly, one permission in **PT** conflicts 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 conflicts, and none were fixed or removed.

Most of the twelve permissions appearing with the **IG** conflict 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 conflict permissions were altered on several occasions: eight were changed to **OG** conflict, 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 conflict.

OG conflict had no permissions fixed or removed. Out of 67 permissions appearing with this conflict across Android versions 11–13, 57 were first-time introduced, while 10 already existed: eight were changed from the **HG** conflict, 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.

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.

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 [18] are open to third-party developers [17]. In this work, we

Table 4.5: 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

Table 4.6: Restriction categories of elements 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**

explore permissions use in both non-SDK and SDK interfaces, accessible for system and third-party apps.

4.5 Element-Permission inconsistencies

Across Android versions 10–13, we derived 3,615 unique elements (e.g., methods, variables) protected by permissions. Among them, 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 mishap is not clear, since the restriction lists are generated for AOSP, which, among other things, is meant to indicate 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.

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 4.6.

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. We discovered numerous inconsistencies that are summarized in Table 4.7.

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, an 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 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. [34]. However, this is not the case for "canAuthenticate". There is seemingly no reason for the element to stay public, as 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.

As for the *unsupported* elements, there are no guidelines to be applied due to their unrestricted nature. Our analysis confirms that their numbers are decreasing as Android promised [36], 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 4.7. 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.

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. Changes are made to these two systems independently. As our analysis shows, it becomes increasingly difficult for Android developers to ensure their consistency, for vendors to comply with them, and nearly impossible for third-party developers to follow them correctly and conscientiously.

Table 4.7: Overview of discrepancies between element and permission restrictions.

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.

5 Results of practical analysis

This chapter contains the practical results of this thesis. It presents the dataset of the applications collected and explains the process for and the results of the automated analysis of permissions in Android applications that leverages our theoretical findings.

5.1 Experimental analysis with PChecker

To further understand permission misuse in practice, we performed an analysis of third-party Android applications using our derived mappings as a basis for classification by the PChecker tool.

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 in practice:

- (**AH**) permissions tagged `@hide` should not be available to third-party apps, and consequently should not be requested by third-party developers;
- (**AO**) permissions tagged `@SystemApi` should not be available to 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.

These issues should be viewed separately from the permission labelling conflicts, which represent contradictory combinations of attributes. Here, a particular attribute property of a permission contradicts the way the permission is handled on Android devices.

5.2 Benchmark app analysis

Table 5.1: 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	no tag	Install	relevant	normal	general	-	44	36	38	44	44
public	no tag	Install	backw. comp	normal	general	-	4	3	3	4	4
public	no tag	Runtime	relevant	normal	general	NR	5	5	5	5	5
public	no tag	Runtime	backw. comp	normal	general	NR	1	1	1	1	1
sdk	**SystemApi**	**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 as **bold** are rows where an issue occurred for our testing app, the rest were handled according to documentation

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 5.1.

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, this permission was granted to our app in direct contradiction with the official documentation.

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

Table 5.2: The summary of Android apps

Source	# APKs	Installable	Runnable	Apps' target API level			
				29	**30**	**31**	**32**
androgalaxy (2019)	257	190	175 (68.09%)	175	0	0	0
androidapkfree.com (2020)	319	200	169 (52.98%)	160	9	0	0
apkgod (2020)	471	393	356 (75.58%)	356	0	0	0
APKMAZA (2020)	34	25	23 (67.65%)	23	0	0	0
APKPure (2021)	317	180	165 (52.05%)	155	10	0	0
appsapk.com (2020)	161	132	122 (75.78%)	122	0	0	0
CracksHash (2022)	1068	913	705 (66.01%)	38	470	179	18
CracksHash (2021)	2015	1476	969 (48.09%)	391	520	57	1
F-droid (2020)	1202	1186	997 (82.95%)	943	54	0	0
Total	**5844**	**4695**	**3681 (62.99%)**	**2363**	**1063**	**236**	**19**

information, e.g., concerning mobile health app installed on a device. For example, the presence of a diabetes app on a phone implies that a user likely has diabetes [77].

5.3 Automated analysis

We leverage PChecker to explore the usage of permissions in Android apps collected in the wild.

For this analysis, we collected 35,453 APKs from two Android markets: F-droid, APKPure.com; and six websites distributing Android applications: androidapkfree.com, appsapk.com, androgalaxy (no longer exists), CracksHash, apkgod, APKMAZA. 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 or had an invalid API level (more than 33), contained less than two files and contained less than two permissions in the manifest file. We also removed APKs whose signature was not verified successfully by apksigner [27], a signature verification tool. This left us with 17,398 APKs.

Among these, we selected APKs that had a `targetSdkVersion` ≥ 29 (Android 10), assuming they were developed in line with recent practices and limitations set by Google.This resulted in a subset containing 5,844 applications. Out of 5,844, only 3,681 applications were successfully installed on all four test devices. The summary of these applications is shown in Table 5.2.

We then analyzed these 3,681 apps using PChecker. The results of this assessment are given in Table 5.3.

Table 5.3: Results of the automated analysis of Android apps

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 (max O)	**hide**	**Install**	**relevant**	**normal**	**general**	**8**	**8 (100.0%)**	8	0 (.0%)	8	0 (.0%)	8	0 (.0%)
cond. block (max 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 (max O)	**hide**	**Install**	**relevant**	**signature**	**general**	5	0 (.0%)	**5**	**2 (40.0%)**	**5**	**2 (40.0%)**	5	0 (.0%)
cond. block (max 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 (max 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

5.3.1 Requested permissions

We expected the number of the requested by our apps permissions to be similar across devices. We observed noticeable differences between permissions requested in manifest files and permissions detected as requested by the Android OS.

Conditionally blocked permissions. 913 (2.4%) requested permissions in our apps 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.

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 available to the third-party developers, yet 183–186 (less than 1%) are requested. Some of these requested permissions (2–16%) were granted, which directly contradicts the official documentation.

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

Blacklisted permissions. We found a small percentage of cases where apps targeting API 29 requested a *blacklisted* permission. These cases concern one unique permission `START_ACTIVITIES_FROM_BACKGROUND`, which was at first (Android 10,11) assigned to restriction lists from the *blacklist* category, but then moved to the

Table 5.4: Permission violations in practice

Issue code	# APKs	Instances				
		A10	**A11**	**A12**	**A13**	**Total**
"Requested" issues						
AH	464	903	901	901	895	3600
AO	117	4	29	29	4	66
Total	**538**	**907**	**930**	**930**	**899**	**3666**
"Granted" issues						
CB	462	903	899	899	895	3596
BL	2	4	30	30	4	68
N3	6	0	2	2	0	4
RE	1	0	1	1	0	2
SY	6	4	29	29	4	66
Total	**462**	**911**	**961**	**961**	**903**	**3736**

sdk restriction lists. This permission was granted on Android versions 11 and 12, where the former perceives the permission as *blacklisted*, while the latter perceives the permission as *sdk*. Nevertheless, according to the app's `targetSdkVersion`, this permission should have been blocked.

Contradictory documentation. We observe in our analysis that permissions are requested and granted often *regardless of their annotations and context.*

For example, 895 permissions that were requested by apps are in the Removed AndroidManifest section which lists permissions removed from the current API. Yet, these permissions were requested and all of them were granted to the analyzed apps. Similarly, there are 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.

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. One was granted in Android 11 and 12.

This indicates that the permission annotations, and therefore, indicated restrictions are ignored by developers and are not followed.

5.3.2 Granted permissions

Our findings show that the apps installed on devices were granted between 24,064 and 24,315 permissions in total, representing 64–65% of the requested permissions.

Our analysis revealed several security violations where unexpected permissions were requested by and/or granted to 538 apps in total on installation. The summary of the issues is shown in Table 5.4, more details on granted permissions with the corresponding conflicts are provided in Table 5.5.

CB issue. We found **3,596** instances of issues in 462 analyzed apps, i.e., 98% of the requested CB permissions 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 request authentication tokens, which may lead to the exploitation of authenticated OS resources without user notification.

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.

In Android 13, this permission was moved to tthe *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, we also discovered **68** issues, where permissions, recognized by Android developers as "not for third-party use", were granted to six third-party applications. 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 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.

Non-existent permissions. In our analysis of the system's Android Manifest files (Section 4.3) we encountered 4 permissions listed in the official restriction lists, but not present in any of the four versions of Android code (10–13). In the analyzed apps, we did not encounter the use of any of these permissions:

- `BIND_VISUAL_VOICEMAIL_SERVICE`,
- `CLEAR_APP_GRANTED_URI_PERMISSIONS`,
- `MANAGE_SCOPED_ACCESS_DIRECTORY_PERMISSIONS` and
- `SEND_CATEGORY_CAR_NOTIFICATIONS`.

Table 5.5: Granted to APKs permissions with conflicting combinations

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	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		**907**	**939**	**939**	**899**	**947**	**551**	**363**	**33**	**0**

Table 5.6: 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	Total	21288	13374	21279	13426	22140	13427	29126	13378
			Defined	21260	13374	21251	13413	21251	13413	21251	13366
			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	Total	13530	8800	13443	8955	14002	8964	17457	8964
			Defined	13513	8800	13426	8955	13421	8953	13420	8953
			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	Total	2598	1762	2576	1794	2559	1811	3191	1811
			Defined	2598	1762	2576	1794	2559	1811	2559	1811
			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	Total	208	128	208	134	208	138	236	138
			Defined	208	128	208	134	208	138	208	138
			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*

5.3.3 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 5.6.

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) [19]. 11 apps requested `QUERY_ALL_PACKAGES` added in API 30 [20], and one app requested `SCHEDULE_EXACT_ALARM` introduced two years later in API 31 [21]. 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 [29]. 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 [29].

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 [43].

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` [41]. 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 [41].

Although the amount of undefined permissions granted is small (less than 1%), these cases show that the OS version of the device plays a major role in the way apps' permissions are treated, introducing even more uncertainty for third-party developers.

6 Conclusion and discussion

We now summarize the findings and results of this work, enumerate discovered problematic patterns and outline potential vectors for future research.

6.1 Summary

6.1.1 Summary of contributions

In this thesis, we analyzed permissions, an essential component of the Android access control system. Our analysis sheds light on how various types of permissions are handled in real-world Android implementations. 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.

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.

6.1.2 Discussion

Clear and precise documentation is crucial for providing guidance to developers. However, when documentation is inconsistent and contradictory, it can create confusion leading to security and privacy risks. Our analysis highlights the prevalence of non-compliance issues in Android applications, underscoring the impact of inconsistent documentation.

Specifically, our findings emphasize several important issues:

Documentation in theory:

- **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.

- **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. For example, public permissions open to third-party developers are combined with the presence of `internal` protection level and the `@SystemApi` labelling.

  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.

Documentation in practice:

- **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.** 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.

Noncompliance in practice:

- **Disregard of restrictions.**

  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.

- **Overprivileged applications.** The problem of overprivileged applications has been extensively researched over the past 10 years [2, 47, 63, 87]. Despite lacking any functionality that necessitates permissions, our testing app was given over 60 unnecessary permissions, demonstrating that this issue remains prevalent today.

6.2 Future work

We laid the groundwork for more fine-grained permission analysis and highlighted problems with current Android documentation, in the source code and in developer-written articles. This research can be expanded to leverage static path-sensitive analysis to further analyze the extent of inconsistent permission and call categorization. Our findings can also be implemented in practice: static and dynamic analysis of third-party applications could show which inconsistently-protected interfaces developers actually use and how these calls are handled by the Android OS in practice.

Given that the state of the Android permission model and overall security is imperfect within the AOSP alone, we advocate for more scrutinized control of vendor-customized images, such as the requirement for the generation of custom restriction lists per vendor, more research into security implications of the closed-source OS customization and requirements for vendors to comply and keep up with Android-defined security practices.

References

[1] Yousra Aafer, Jianjun Huang, Ninghui Li Yi Sun X. Zhang, and Chen Tian. Acedroid: Normalizing diverse android access control checks for inconsistency detection. In *Network and Distributed System Security Symposium*, 2018.

[2] Yousra Aafer, Guanhong Tao, Jianjun Huang, Xiangyu Zhang, and Ninghui Li. Precise android api protection mapping derivation and reasoning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, page 1151–1164, 2018.

[3] Efthymios Alepis and Constantinos Patsakis. Unravelling security issues of runtime permissions in android. *Journal of Hardware and Systems Security*, 3:45–63, 2019.

[4] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, page 259–269, 2014.

[5] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: Analyzing the android permission specification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, page 217–228, 2012.

[6] Michael Backes, Sven Bugiel, Erik Derr, Patrick McDaniel, Damien Octeau, and Sebastian Weisgerber. On demystifying the android application framework: Re-visiting android permission specification analysis. In *Proceedings of the 25th USENIX Conference on Security Symposium*, page 1101–1118, 2016.

[7] Michael Backes, Sven Bugiel, Christian Hammer, Oliver Schranz, and Philipp Von Styp-Rekowsky. Boxify: Full-fledged app sandboxing for stock android. In *Proceedings of the 24th USENIX Conference on Security Symposium*, page 691–706, 2015.

[8] David Barrera, H. Güneş Kayacik, Paul C. van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, page 73–84, 2010.

[9] Alastair R. Beresford, Andrew Rice, Nicholas Skehin, and Ripduman Sohan. Mockdroid: Trading privacy for application functionality on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, page 49–54, 2011.

[10] Antonio Bianchi, Jacopo Corbetta, Luca Invernizzi, Yanick Fratantonio, Christopher Kruegel, and Giovanni Vigna. What the app is that? deception and countermeasures in the android user interface. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, page 931–948, 2015.

[11] Bram Bonné, Sai Teja Peddinti, Igor Bilogrevic, and Nina Taft. Exploring decision making with android's runtime permission dialogs using in-context surveys. In *Proceedings of the Thirteenth USENIX Conference on Usable Privacy and Security*, page 195–210, 2017.

[12] Theodore Book, Adam Pridgen, and Dan Wallach. Longitudinal analysis of android ad library permissions. *Computing Research Repository*, 2013.

[13] Haipeng Cai, Ziyi Zhang, Li Li, and Xiaoqin Fu. A large-scale study of application incompatibilities in android. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, page 216–227, 2019.

[14] Paolo Calciati, Konstantin Kuznetsov, Alessandra Gorla, and Andreas Zeller. Automatically granted permissions in android apps: An empirical study on their prevalence and on the potential threats for privacy. In *Proceedings of the 17th International Conference on Mining Software Repositories*, page 114–124, 2020.

[15] Huan Chang, Lingguang Lei, Kun Sun, Yuewu Wang, Jiwu Jing, Yi He, and Pingjian Wang. Vulnerable service invocation and countermeasures. *IEEE Transactions on Dependable and Secure Computing*, 18:1733–1750, 2021.

[16] Abd Elhamed M. Dawoud and Sven Bugiel. Bringing balance to the force: Dynamic analysis of the android application framework. In *Network and Distributed System Security Symposium*, 2021.

[17] Android Developers. Improving stability by reducing usage of non-sdk interfaces, 2018. `https://android-developers.googleblog.com/2018/02/improving-stability-by-reducing-usage.html`.

[18] Android Developers. Package index, 2018. `https://developer.android.com/reference/packages`.

[19] Android Developers. Manifest.permission, 2019. `https://developer.android.com/reference/android/Manifest.permission#MANAGE_EXTERNAL_STORAGE`.

[20] Android Developers. Manifest.permission, 2019. `https://developer.android.com/reference/android/Manifest.permission#QUERY_ALL_PACKAGES`.

[21] Android Developers. Manifest.permission, 2019. `https://developer.android.com/reference/android/Manifest.permission#SCHEDULE_EXACT_ALARM`.

[22] Android Developers. Updates to non-sdk interface restrictions in android 10, 2021. `https://developer.android.com/about/versions/10/non-sdk-q`.

[23] Android Developers. Updates to non-sdk interface restrictions in android 11, 2021. `https://developer.android.com/about/versions/11/non-sdk-11`.

[24] Android Developers. Aapt2 (android asset packaging tool), 2022. `https://developer.android.com/studio/command-line/aapt2`.

[25] Android Developers. Android debug bridge (adb), 2022. `https://developer.android.com/studio/command-line/adb`.

[26] Android Developers. Android developers reference, 2022. `https://developer.android.com/reference/android/R.attr#protectionLevel`.

[27] Android Developers. apksigner, 2022. `https://developer.android.com/studio/command-line/apksigner`.

[28] Android Developers. Define a custom app permission, 2022. `https://developer.android.com/guide/topics/permissions/defining`.

[29] Android Developers. Manifest.permission, 2022. `https://developer.android.com/reference/android/Manifest.permission`.

[30] Android Developers. Manifest.permission, 2022. `https://developer.android.com/reference/android/Manifest.permission#READ_NEARBY_STREAMING_POLICY`.

[31] Android Developers. Permissions on android, 2022. `https://developer.android.com/guide/topics/permissions/overview#system-components`.

[32] Android Developers. Platform version requirements, 2022. `https://developer.android.com/guide/topics/manifest/uses-sdk-element`.

[33] Android Developers. Requires permission, 2022. `https://developer.android.com/reference/androidx/annotation/RequiresPermission`.

[34] Android Developers. Requires permission: Public fields - conditional, 2022. https://developer.android.com/reference/androidx/annotation/RequiresPermission#conditional().

[35] Android Developers. Restrictions on non-sdk interfaces, 2022. https://developer.android.com/guide/app-compatibility/restrictions-non-sdk-interfaces.

[36] Android Developers. Restrictions on non-sdk interfaces, 2022. https://developer.android.com/guide/app-compatibility/restrictions-non-sdk-interfaces#list-names.

[37] Android Developers. Ui/application exerciser monkey, 2022. https://developer.android.com/studio/test/other-testing-tools/monkey.

[38] Android Developers. Updates to non-sdk interface restrictions in android 11, 2022. https://developer.android.com/about/versions/11/non-sdk-11.

[39] Android Developers. Updates to non-sdk interface restrictions in android 12, 2022. https://developer.android.com/about/versions/12/non-sdk-12.

[40] Android Developers. Updates to non-sdk interface restrictions in android 13, 2022. https://developer.android.com/about/versions/11/non-sdk-13.

[41] Android Developers. Behavior changes: Apps targeting android 13 or higher, 2023. https://developer.android.com/about/versions/13/behavior-changes-13.

[42] Android Developers. Meet google play's target api level requirement, 2023. https://developer.android.com/google/play/requirements/target-sdk#why-target.

[43] Android Developers. Notification runtime permission, 2023. https://developer.android.com/develop/ui/views/notifications/notification-permission#new-apps.

[44] Nicole Eling, Siegfried Rasthofer, Max Kolhagen, Eric Bodden, and Peter Buxmann. Investigating users' reaction to fine-grained data requests: A market experiment. In *Proceedings of the 2016 49th Hawaii International Conference on System Sciences (HICSS)*, page 3666–3675, 2016.

[45] Mohamed Elsabagh, Ryan Johnson, Angelos Stavrou, Chaoshun Zuo, Qingchuan Zhao, and Zhiqiang Lin. Firmscope: Automatic uncovering of privilege-escalation vulnerabilities in pre-installed apps in android firmware. In *Proceedings of the 29th USENIX Conference on Security Symposium*, page 2379–2396, 2020.

[46] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, page 235–245, 2009.

[47] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, page 627–638, 2011.

[48] Adrienne Porter Felt, Helen J. Wang, Alexander Moshchuk, Steven Hanna, and Erika Chin. Permission re-delegation: Attacks and defenses. In *Proceedings of the 20th USENIX Conference on Security*, page 22, 2011.

[49] Yanick Fratantonio, Chenxiong Qian, Simon P. Chung, and Wenke Lee. Cloak and dagger: From two permissions to complete control of the ui feedback loop. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1041–1057, 2017.

[50] Xing Gao, Dachuan Liu, Haining Wang, and Kun Sun. Pmdroid: Permission supervision for android advertising. In *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*, pages 120–129, 2015.

[51] Google. Android open source project, 2023. https://android.googlesource.com/.

[52] Google. Define a custom app permission, 2023. `https://developer.android.com/guide/topics/permissions/defining`.

[53] Sigmund Albert Gorski, Benjamin Andow, Adwait Nadkarni, Sunil Manandhar, William Enck, Eric Bodden, and Alexandre Bartel. Acminer: Extraction and analysis of authorization checks in android's middleware. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, page 25–36, 2019.

[54] Dongjie He, Lian Li, Lei Wang, Hengjie Zheng, Guangwei Li, and Jingling Xue. Understanding and detecting evolution-induced compatibility issues in android apps. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, page 167–177, 2018.

[55] Yi He, Yacong Gu, Purui Su, Kun Sun, Yajin Zhou, Zhi Wang, and Qi Li. A systematic study of android non-sdk (hidden) service api security. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2022.

[56] Yi He, Yuan Zhou, Yajin Zhou, Qi Li, Kun Sun, Yacong Gu, and Yong Jiang. Jni global references are still vulnerable: Attacks and defenses. *IEEE Transactions on Dependable and Secure Computing*, 19:607–619, 2022.

[57] Grant Hernandez, Dave Jing Tian, Anurag Swarnim Yadav, Byron J. Williams, and Kevin R. B. Butler. Bigmac: Fine-grained policy analysis of android firmware. In *Proceedings of the 29th USENIX Conference on Security Symposium*, page 271–287, 2020.

[58] Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S. Foster, and Todd Millstein. Dr. android and mr. hide: Fine-grained permissions in android applications. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, page 3–14, 2012.

[59] Haofeng Jiao, Xiaohong Li, Lei Zhang, Guangquan Xu, and Zhiyong Feng. Hybrid detection using permission analysis for android malware. In *International Conference on Security and Privacy in Communication Networks*, pages 541–545, 11 2015.

[60] ElMouatez Billah Karbab, Mourad Debbabi, Abdelouahid Derhab, and Djedjiga Mouheb. Maldozer: Automatic framework for android malware detection using deep learning. *Digital Investigation*, 24:S48–S59, 2018.

[61] Patrick Gage Kelley, Sunny Consolvo, Lorrie Faith Cranor, Jaeyeon Jung, Norman Sadeh, and David Wetherall. A conundrum of permissions: Installing applications on an android smartphone. In *Financial Cryptography and Data Security*, page 68–79, 2012.

[62] Li Li, Tegawendé F. Bissyandé, Haoyu Wang, and Jacques Klein. Cid: Automating the detection of api-related compatibility issues in android apps. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, page 153–163, 2018.

[63] Li Li, Tegawendé F. Bissyandé, Yves Le Traon, and Jacques Klein. Accessing inaccessible android apis: An empirical study. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 411–422, 2016.

[64] Li Li, Jun Gao, Tegawendé F. Bissyandé, Lei Ma, Xin Xia, and Jacques Klein. Cda: Characterising deprecated android apis. *Empirical Software Engineering*, 25:2058–2098, 2020.

[65] Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I. Hong. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. In *Proceedings of the Tenth USENIX Conference on Usable Privacy and Security*, page 199–212, 2014.

[66] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. Api change and fault proneness: A threat to the success of android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, page 477–487, 2013.

[67] Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. How do api changes trigger stack overflow discussions? a study on the android sdk. In *Proceedings of the 22nd International Conference on Program Comprehension*, page 83–94, 2014.

[68] Baozheng Liu, Chao Zhang, Guang Gong, Yishun Zeng, Haifeng Ruan, and Jianwei Zhuge. Fans: Fuzzing android native system services via automated interface analysis. In *Proceedings of the 29th USENIX Conference on Security Symposium*, page 307–323, 2020.

[69] Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhimedi, Shikun Zhang, Norman Sadeh, Alessandro Acquisti, and Yuvraj Agarwal. Follow my recommendations: A personalized privacy assistant for mobile app permissions. In *Proceedings of the Twelfth USENIX Conference on Usable Privacy and Security*, page 27–41, 2016.

[70] Pei Liu, Li Li, Yichun Yan, Mattia Fazzini, and John Grundy. Identifying and characterizing silently-evolved methods in the android api. In *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice*, page 308–317, 2021.

[71] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. Chex: Statically vetting android apps for component hijacking vulnerabilities. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, page 229–240, 2012.

[72] Lannan Luo, Qiang Zeng, Chen Cao, Kai Chen, Jian Liu, Limin Liu, Neng Gao, Min Yang, Xinyu Xing, and Peng Liu. System service call-oriented symbolic execution of android framework with applications to vulnerability discovery and exploit generation. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, page 225–238, 2017.

[73] René Mayrhofer, Jeffrey Vander Stoep, Chad Brubaker, and Nick Kralevich. The android platform security model. *ACM Transactions on Privacy and Security*, 24:1–35, apr 2021.

[74] Tyler McDonnell, Baishakhi Ray, and Miryung Kim. An empirical study of api stability and adoption in the android ecosystem. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, page 70–79, 2013.

[75] Mohammad Nauman, Sohail Khan, and Xinwen Zhang. Apex: Extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, page 328–332, 2010.

[76] Sai Teja Peddinti, Igor Bilogrevic, Nina Taft, Martin Pelikan, Úlfar Erlingsson, Pauline Anthonysamy, and Giles Hogben. Reducing permission requests in mobile apps. In *Proceedings of the Internet Measurement Conference*, page 259–266, 2019.

[77] Anh Pham, Italo Dacosta, Eleonora Losiouk, John Stephan, Kévin Huguenin, and Jean-Pierre Hubaux. Hidemyapp: Hiding the presence of sensitive apps on android. In *Proceedings of the 28th USENIX Conference on Security Symposium*, page 711–728, 2019.

[78] Android Open Source Project. Privileged permission allowlisting, 2022. `https://source.android.com/docs/core/config/perms-allowlist`.

[79] James Sellwood and Jason Crampton. Sleeping android: The danger of dormant permissions. In *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, page 55–66, 2013.

[80] Yuru Shao, Qi Alfred Chen, Z. Morley Mao, Jason Ott, and Zhiyun Qian. Kratos: Discovering inconsistent security policy enforcement in the android framework. In *Network and Distributed System Security Symposium*, 2016.

[81] Mark D. Syer, Meiyappan Nagappan, Bram Adams, and Ahmed E. Hassan. Studying the relationship between source code quality and mobile platform dependence. *Software Quality Journal*, 23:485–508, 2015.

[82] Güliz Seray Tuncay, Soteris Demetriou, Karan Ganju, and Carl A. Gunter. Resolving the predicament of android custom permissions. In *Network and Distributed System Security Symposium*, 2018.

[83] Guliz Seray Tuncay, Carl A. Gunter, Tao Xie, Adam Bates, and Suman Jana. *Practical least privilege for cross-origin interactions on mobile operating systems*. PhD thesis, University of Illinois at Urbana-Champaign, 2019.

[84] Güliz Seray Tuncay, Jingyu Qian, and Carl A. Gunter. *See No Evil: Phishing for Permissions with False Transparency*, page 415–432. USENIX Association, 2020.

[85] Fabo Wang, Yuqing Zhang, Kai Wang, Peng Liu, and Wenjie Wang. Stay in your cage! a sound sandbox for third-party libraries on android. In *European Symposium on Research in Computer Security*, page 458–476, 2016.

[86] Lili Wei, Yepang Liu, and Shing-Chi Cheung. Taming android fragmentation: Characterizing and detecting compatibility issues for android apps. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, page 226–237, 2016.

[87] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, page 31–40, 2012.

[88] Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. Android permissions remystified: A field study on contextual integrity. In *Proceedings of the 24th USENIX Conference on Security Symposium*, page 499–514, 2015.

[89] Lei Wu, Michael Grace, Yajin Zhou, Chiachih Wu, and Xuxian Jiang. The impact of vendor customizations on android security. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, page 623–634, 2013.

[90] Hao Xia, Yuan Zhang, Yingtian Zhou, Xiaoting Chen, Yang Wang, Xiangyu Zhang, Shuaishuai Cui, Geng Hong, Xiaohan Zhang, Min Yang, and Zhemin Yang. How android developers handle evolution-induced api compatibility issues: A large-scale study. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, page 886–898, 2020.

[91] Guowei Yang, Jeffrey Jones, Austin Moninger, and Meiru Che. How do android operating system updates impact apps? In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, page 156–160, 2018.

[92] Shishuai Yang, Rui Li, Jiongyi Chen, Wenrui Diao, and Shanqing Guo. Demystifying android non-sdk apis: Measurement and understanding. In *Proceedings of the 44th International Conference on Software Engineering*, page 647–658, 2022.

[93] Wei Yang, Xusheng Xiao, Benjamin Andow, Li Sihan, Tao Xie, and William Enck. Appcontext: Differentiating malicious and benign mobile app behaviors using context. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, page 303–313, 2015.

[94] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, page 317–326, 2012.

Appendix A

Unified permission mapping example


```
"android.permission.READ_CONTACTS": {
  "seen_in_manifests": [29,30,31,33],
  "seen_in_lists": [29,30,31,33],
  "introduced": "1",
  "versions": {
    "29": {
      "tag": "sdk",
      "restriction_list": "public-api,system-api,test-api,whitelist",
      "f_group": "Permissions for accessing user's contacts including personal profile",
      "deprecated": false,
      "status": "relevant",
      "protection_level": "dangerous",
      "usage": "general",
      "type": "RUNTIME PERMISSIONS",
      "restriction": "public",
      "protection": "dangerous"
    },
    "30": {
      "tag": "sdk",
      "restriction_list": "public-api,system-api,test-api,whitelist",
      "f_group": "Permissions for accessing user's contacts including personal profile",
      "deprecated": false,
      "status": "relevant",
      "protection_level": "dangerous",
      "usage": "general",
      "type": "RUNTIME PERMISSIONS",
      "restriction": "public",
      "protection": "dangerous"
    },
    "31": {
      "tag": "sdk",
      "restriction_list": "public-api,sdk,system-api,test-api",
      "f_group": "Permissions for accessing user's contacts including personal profile",
      "deprecated": false,
      "status": "relevant",
      "protection_level": "dangerous",
      "usage": "general",
      "type": "RUNTIME PERMISSIONS",
      "restriction": "public",
      "protection": "dangerous"
    },
    "33": {
      "tag": "sdk",
      "restriction_list": "public-api,sdk,system-api,test-api",
      "f_group": "Permissions for accessing user's contacts including personal profile",
      "deprecated": false,
      "status": "relevant",
      "protection_level": "dangerous",
      "usage": "general",
      "type": "RUNTIME PERMISSIONS",
      "restriction": "public",
      "protection": "dangerous"
    }
  }
}
```


Figure A.1: An entry of the Unified permission mapping

Appendix B

Unified guarded call mapping example


```
"Landroid/accessibilityservice/AccessibilityService;->getFingerprintGestureController()Landroid/accessibilityservice/FingerprintGestureController;": {
  "seen_in_source": [10,11,12,13],
  "seen_in_lists": [10,11,12,13],
  "versions": {
    "10": {
      "version": "10",
      "invocation": "/core/java/android/accessibilityservice/AccessibilityService#getFingerprintGestureController()#442db5b0540d3fa7ab1cb29c3fe5bc308c48fba7",
      "perms": {
        "basic": {
          "perms": ["android.Manifest.permission.USE_FINGERPRINT"],
          "params": [],
          "req": [null]
        }
      },
      "rl": "public-api,system-api,test-api,whitelist",
      "restr": "public",
      "restr_line": "Landroid/accessibilityservice/AccessibilityService;->getFingerprintGestureController()Landroid/accessibilityservice/FingerprintGestureController;,public-api,system-api,test-api,whitelist",
      "perms_changed": null,
      "restrs_changed": null,
      "alt": null
    },
    "11": {
      "version": "11",
      "invocation": "/core/java/android/accessibilityservice/AccessibilityService#getFingerprintGestureController()#442db5b0540d3fa7ab1cb29c3fe5bc308c48fba7",
      "perms": {
        "basic": {
          "perms": ["android.Manifest.permission.USE_FINGERPRINT"],
          "params": [],
          "req": [null]
        }
      },
      "rl": "public-api,system-apt,test-api,whitelist",
      "restr": "public",
      "restr_line": "Landroid/accessibilityservice/AccessibilityService;->getFingerprintGestureController()Landroid/accessibilityservice/FingerprintGestureController;,public-api,system-api,test-api,whitelist",
      "perms_changed": false,
      "restrs_changed": false,
      "alt": null
    },
    "12": {
      "version": "12",
      "invocation": "/core/java/android/accessibilityservice/AccessibilityService#getFingerprintGestureController()#442db5b0540d3fa7ab1cb29c3fe5bc308c48fba7",
      "perms": {
        "basic": {
          "perms": ["android.Manifest.permission.USE_FINGERPRINT"],
          "params": [],
          "req": [null]
        }
      },
      "rl": "public-api,sdk,system-api,test-api",
      "restr": "public",
      "restr_line": "Landroid/accessibilityservice/AccessibilityService;->getFingerprintGestureController()Landroid/accessibilityservice/FingerprintGestureController;,public-api,sdk,system-api,test-api",
      "perms_changed": false,
      "restrs_changed": false,
      "alt": null
    },
    "13": {
      "version": "13",
      "invocation": "/core/java/android/accessibilityservice/AccessibilityService#getFingerprintGestureController()#442db5b0540d3fa7ab1cb29c3fe5bc308c48fba7",
      "perms": {
        "basic": {
          "perms": ["android.Manifest.permission.USE_FINGERPRINT"],
          "params": [],
          "req": [null]
        }
      },
      "rl": "public-api,sdk,system-api,test-api",
      "restr": "public",
      "restr_line": "Landroid/accessibilityservice/AccessibilityService;->getFingerprintGestureController()Landroid/accessibilityservice/FingerprintGestureController;,public-api,sdk,system-api,test-api",
      "perms_changed": false,
      "restrs_changed": false,
      "alt": null
    }
  }
}
```


Figure B.1: An entry of the Unified guarded call mapping

Appendix C

Conflicts in the Android 13 Manifest

Here, we report conflicts found in Android 13 for the convenience of Android developers to review and fix, as it is the most recent OS version.

`android.permission.ACCESS_IMS_CALL_SERVICE` : protection level - `signature`, permission in the Runtime section

`android.permission.MANAGE_OWN_CALLS` : protection level - `normal`, permission in the Runtime section

`android.permission.ACCESS_UCE_PRESENCE_SERVICE` : protection level - `signature`, permission in the Runtime section

`android.permission.ACCESS_UCE_OPTIONS_SERVICE` : protection level - `signature`, permission in the Runtime section

`android.permission.USE_FINGERPRINT` : protection level - `normal`, permission in the Runtime section

`android.permission.USE_BIOMETRIC` : protection level - `normal`, permission in the Runtime section

`android.permission.SEND_RESPOND_VIA_MESSAGE` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.INSTALL_LOCATION_PROVIDER` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.LOCATION_HARDWARE` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.OVERRIDE_WIFI_CONFIG` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.BLUETOOTH_PRIVILEGED` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.GET_ACCOUNTS` : protection level - `dangerous`, permission in the Install section

`android.permission.ACCOUNT_MANAGER` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.MODIFY_PHONE_STATE` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.MANAGE_DOCUMENTS` : described as *restricted*, yet not excluded from the public API; described as *restricted*, yet restriction - *public*

`android.permission.SET_TIME` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.SET_TIME_ZONE` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.WRITE_GSERVICES` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.SET_ANIMATION_SCALE` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.MOUNT_UNMOUNT_FILESYSTEMS` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.MOUNT_FORMAT_FILESYSTEMS` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.WRITE_APN_SETTINGS` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.WRITE_SECURE_SETTINGS` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.DUMP` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.READ_LOGS` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.SET_DEBUG_APP` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.SET_PROCESS_LIMIT` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.SET_ALWAYS_FINISH` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.SIGNAL_PERSISTENT_PROCESSES` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.DIAGNOSTIC` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.STATUS_BAR` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.UPDATE_DEVICE_STATS` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.READ_INPUT_STATE` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.INSTALL_PACKAGES` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.DELETE_PACKAGES` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.CHANGE_COMPONENT_ENABLED_STATE` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.CAPTURE_AUDIO_OUTPUT` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.MEDIA_CONTENT_CONTROL` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.REBOOT` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.FACTORY_TEST` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.BROADCAST_PACKAGE_REMOVED` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.BROADCAST_SMS` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.BROADCAST_WAP_PUSH` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.MASTER_CLEAR` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.CALL_PRIVILEGED` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.CONTROL_LOCATION_UPDATES` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.ACCESS_CHECKIN_PROPERTIES` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.BIND_APPWIDGET` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.GLOBAL_SEARCH` : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

`android.permission.BIND_CARRIER_SERVICES` : described as *restricted*, yet not excluded from the public API; described as *restricted*, yet restriction - *public*

`android.permission.WRITE_OBB` : protection level - `signature`, permission in the Runtime section

`android.permission.CALL_COMPANION_APP` : protection level - `normal`, permission in the Runtime section

android.permission.ENABLE_TEST_HARNESS_MODE : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.MANAGE_DYNAMIC_SYSTEM : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.START_ACTIVITY_AS_CALLER : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.REQUEST_INCIDENT_REPORT_APPROVAL : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.TEST_MANAGE_ROLLBACKS : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.MONITOR_DEFAULT_SMS_PACKAGE : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.MANAGE_SENSOR_PRIVACY : protection level - `internal`, yet not annotated as *not for third-party apps*

android.permission.MONITOR_INPUT : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.ACCESS_MESSAGES_ON_ICC : protection level - `signature`, permission in the Runtime section

android.permission.BIND_CELL_BROADCAST_SERVICE : protection level - `signature`, permission in the Runtime section

android.permission.MANAGE_EXTERNAL_STORAGE : protection level - `signature`, permission in the Runtime section; described as *restricted*, yet not excluded from the public API; described as *restricted*, yet restriction - *public*

android.permission.EXEMPT_FROM_AUDIO_RECORD_RESTRICTIONS : protection level - `signature`, permission in the Runtime section

android.permission.CAMERA_OPEN_CLOSE_LISTENER : protection level - `signature`, permission in the Runtime section

android.permission.VIBRATE_ALWAYS_ON : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.BIND_CONTROLS : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

android.permission.ACT_AS_PACKAGE_FOR_ACCESSIBILITY : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.MANAGE_COMPANION_DEVICES : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.PEEK_DROPBOX_DATA : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.MANAGE_MEDIA : protection level - `signature`, permission in the Runtime section

android.permission.HIGH_SAMPLING_RATE_SENSORS : protection level - `normal`, permission in the Runtime section

android.permission.BLUETOOTH_SCAN : protection level - `dangerous`, permission in the Install section

android.permission.BLUETOOTH_CONNECT : protection level - `dangerous`, permission in the Install section

android.permission.BLUETOOTH_ADVERTISE : protection level - `dangerous`, permission in the Install section

android.permission.UWB_RANGING : protection level - `dangerous`, permission in the Install section

android.permission.VIRTUAL_INPUT_DEVICE : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.CAMERA_INJECT_EXTERNAL_CAMERA : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.CLEAR_FREEZE_PERIOD : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.FORCE_DEVICE_POLICY_MANAGER_LOGS : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.START_FOREGROUND_SERVICES_FROM_BACKGROUND : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

android.permission.TOGGLE_AUTOMOTIVE_PROJECTION : protection level - internal, yet not annotated as *not for third-party apps*

android.permission.MANAGE_CREDENTIAL_MANAGEMENT_APP : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.KEEP_UNINSTALLED_PACKAGES : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.ASSOCIATE_COMPANION_DEVICES : protection level - internal, yet not annotated as *not for third-party apps*

android.permission.OVERRIDE_DISPLAY_MODE_REQUESTS : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.MODIFY_REFRESH_RATE_SWITCHING_TYPE : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.QUERY_AUDIO_STATE : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.READ_DREAM_SUPPRESSION : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.SET_AND_VERIFY_LOCKSCREEN_CREDENTIALS : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.OBSERVE_SENSOR_PRIVACY : protection level - internal, yet not annotated as *not for third-party apps*

android.permission.ASSOCIATE_INPUT_DEVICE_TO_DISPLAY : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.RESET_APP_ERRORS : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.INPUT_CONSUMER : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.CONTROL_DEVICE_STATE : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.GET_PEOPLE_TILE_PREVIEW : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.READ_GLOBAL_APP_SEARCH_DATA : protection level - internal, yet not annotated as *not for third-party apps*

android.permission.LOCATION_BYPASS : protection level - signature, permission in the Runtime section

android.permission.READ_BASIC_PHONE_STATE : protection level - normal, permission in the Runtime section

android.permission.MANAGE_WIFI_NETWORK_SELECTION : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

android.permission.MANAGE_WIFI_INTERFACES : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

android.permission.NEARBY_WIFI_DEVICES : protection level - dangerous, permission in the Install section

android.permission.REQUEST_COMPANION_PROFILE_APP_STREAMING : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

android.permission.REQUEST_COMPANION_PROFILE_AUTOMOTIVE_PROJECTION : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*; protection level - internal, yet not excluded from the public API; restriction - *public*, yet protection level - internal

android.permission.REQUEST_COMPANION_PROFILE_COMPUTER : annotated as *not for third-party apps*, yet not excluded from the public API; restriction - *public*, yet annotated as *not for third-party apps*

android.permission.USE_ATTESTATION_VERIFICATION_SERVICE : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.VERIFY_ATTESTATION : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.BIND_ATTESTATION_VERIFICATION_SERVICE : restriction - *blacklist*, yet not annotated as *not for third-party apps*

android.permission.REQUEST_UNIQUE_ID_ATTESTATION : restriction - *blacklist*, yet not annotated as *not for third-party apps*

`android.permission.BIND_GAME_SERVICE` : restriction - *blacklist*, yet not annotated as *not for third-party apps*

`android.permission.SUBSCRIBE_TO_KEYGUARD_LOCKED_STATE` : protection level - `internal`, yet not excluded from the public API; restriction - *public*, yet protection level - `internal`; protection level - `internal`, yet not annotated as *not for third-party apps*

`android.permission.MODIFY_TOUCH_MODE_STATE` : restriction - *blacklist*, yet not annotated as *not for third-party apps*

`android.permission.MODIFY_USER_PREFERRED_DISPLAY_MODE` : restriction - *blacklist*, yet not annotated as *not for third-party apps*

`android.permission.READ_ASSISTANT_APP_SEARCH_DATA` : protection level - `internal`, yet not excluded from the public API; restriction - *public*, yet protection level - `internal`; protection level - `internal`, yet not annotated as *not for third-party apps*

`android.permission.READ_HOME_APP_SEARCH_DATA` : protection level - `internal`, yet not excluded from the public API; restriction - *public*, yet protection level - `internal`; protection level - `internal`, yet not annotated as *not for third-party apps*

`android.permission.CREATE_VIRTUAL_DEVICE` : protection level - `internal`, yet not annotated as *not for third-party apps*

`android.permission.SEND_SAFETY_CENTER_UPDATE` : protection level - `internal`, yet not annotated as *not for third-party apps*

`android.permission.MANAGE_SAFETY_CENTER` : protection level - `internal`, yet not annotated as *not for third-party apps*

`android.permission.ACCESS_AMBIENT_CONTEXT_EVENT` : protection level - `internal`, yet not annotated as *not for third-party apps*