Handling over-privileged Android applications based on the minimum permission set identification

Mohammed El Amin TEBIB*, Pascal André†, Mariem Graa‡, Oum-El-Kheir Aktouf*
* Univ. Grenoble Alpes, Grenoble INP, LCIS - Email: mohammed-el-amin.tebib, oum-el-kheir.aktouf@univ-grenoble-alpes.fr
†Univ. of Nantes, LS2N - Email: pascal.andre@ls2n.fr
‡CNAM - Nantes - Email: mariem.graa@gmail.com

***Abstract*—Android applications (apps) access to smartphone resources that manipulate users sensitive data such as position, contacts and images. This access is controlled through *permissions* to restrict the actions performed by apps on system resources. To safely manage *permissions*, developers are highly recommended to assign the smallest set of permissions, those required to run the app. Unfortunately, most of them let apps opened to data privacy risks generated by other vulnerabilities. Previous works proposed assisting tools for developers to define the optimal set of permissions for their apps. However, these works suffer from two significant limits: *outdatedness* and *incompleteness*. In this paper, we present PermDroid, a tool implementing a new development approach for preventing over-privileged apps. PermDroid combines static and dynamic app analysis to produce more complete results. To obtain a good code coverage analysis with automatic updated results, the tool enables a collaborative mode. Developers will apply the analysis process from different nodes on the various Android versions. Combining their analysis results will help to improve the robustness against the continuous evolution of Android API versions. We implement PermDroid as an open source plugin of the IntelliJ IDE.**

***Index Terms*—Android, Development, Permissions, Over-privilege, Static analysis, Dynamic analysis, Collaborative construction**

I. Introduction

According to CVE details [1] (the official MITRE datasource for Android vulnerabilities), recent years witnessed the most significant increase of Android security threats, which can lead to serious security attacks enabling the manipulation of users sensitive data. Experimental research studies [1], [2] show that most of these vulnerabilities are introduced unconsciously at the development stage and 60% among them are related to privileges and private data manipulation. Privileges are access authorizations manipulated through the concept of *Permissions* in Android applications. These permissions are declared by the developer to define which system resources can be used by the app. The least privileges principle [3] tells developers to assign the smallest set of permission as possible to be able to run the app. But most developers declare useless permissions leading to over-privileged applications that can be used by malicious apps to perform insecure tasks such as invoking unprotected API [4]. Google provides an official documentation [5] to explain how to properly use each permission. However, due to the continuous changes of permissions rules, this documentation is hardly readable for developers; it is incomplete and contains inaccurate informations [6], [7]. Previous works proposed tools to assist developers to define an optimal set for a safe app execution context [8], [9], [10], [11]. This set is optimal in filling the gap between the set of *declared permissions* and the *used permissions*. The first one is defined statically while the second one depends on the execution. The key element to obtain an optimal permissions set is to build a *permission-mapping* (**PM**) indicating for each API call in the application which permissions have to be used [7]. Existing solutions suffer from *Outdatedness*, the tools are not up-to-date with Google specifications, and *Incompleteness*, meaning that these works are not accurate enough to detect a precise permissions set.

In this paper, we target these limitations by providing the following contributions:

1) We propose to combine static and dynamic analysis to target the *completeness* limitations. PermDroid performs a hybrid analysis at two levels: (1) The application level in order to extract a complete list of API calls performed by the app; and (2) the framework level to construct a more complete PM enabling developers to find all the required permissions for their corresponding API calls.
2) Instead of launching the analysis process on one local node, PermDroid builds the PM based on a collaborative development approach. A set of developers can simultaneously run the analysis process, share the analysis results, and iterate to ensure continuous updates with recent API versions (not yet covered in a previous analysis iteration).

We implement PermDroid as an IntellIJ/Android Studio IDE plug-in that can be used alongside the Android-specific development environment provided in the SDK.

The rest of the paper is organized as follows. Section 2 reviews the existing approaches proposed in the literature. Section 3 underlines limits and presents our main contributions. Finally, we resume the on-going work and we present the future steps.

II. Background & Related Works

In this section we recall the main concepts related to permissions and how to detect over-privileged application. We overview current research and on-going challenges.

[1] https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224

TABLE I: Overpriviledge Detection: Integrated IDE Solutions

ToolName	Year of publication	Support IDE	U.Permission Mapping	Approach	Api Level	Availabity
PermDroid	2021	IntellIJ, Android Studio	pscout, dynamo, arcade, self-constructed	Hybrid	9..30	Y
PerHelper	2018	IntellIJ	pscout	Static	12	N
PermiMe	2014	Eclipse	pscout	Static	12	N
Curbing	2011	Eclipse	Manual	Static	9	N

Permissions in Android applications are a basic element to define authorizations. They represent access rights granted to applications to use system resources such as: position, camera, voice, contacts, etc. The app requests to access a specific resource through the concept of API. For each *API call*, specific permission(s) is (are) required to be declared in the app. If permission is missing, the API can not be used (so the system resource can not be accessed). On the other hand, the application is considered *over-privileged* if a declared permission is not required by any corresponding API call. This useless declared permission could be exploited to create critical security effects at runtime. To understand how to automatically detect over-privileged applications, we formalized the algorithm 1. Starting from the set of *api-Calls* performed by the app, we investigate the set of the declared permissions *declaredPerms* to perform these calls. Each declared permission *perm* that does not correspond to any *apiCall* will be considered as unused *(perm.used=false)* and the app will be notified as over-privileged.

Algorithm 1 Detecting Over-privileged Applications

$declaredPerms \leftarrow getDeclaredPerms(manifest)$;
$apiCalls \leftarrow getApiCall(sourceCodeFiles)$;
Initialize perm.used = false $forallperm \in declaredPerms$
for each $apiCall \in apiCalls$ **do**
    $perms \leftarrow getPermissionsOfApiCall(apiCall)$;
    for each $p \in perms$ **do**
        if $p \in declaredPerms$ **then** perm.used=true;
        end if
    end for
end for

Research works over the last ten years proposed app analysis tools to inspect the application source code and raise up the list of unused permissions to developers in order to be removed from the app. In Table I we summarize the tools we found in the literature such as PerHelper[10], PermitMe[9], Curbing[8]. There are also general-purpose quality tools such as Sonarqube[12], FindBugs[13], Checkmarx [14] and Fortify[15]. We studied their abilities in detecting over-privileged applications.

We experimented industrial tools during a student cyber-security project by analyzing a malicious application presenting an over-privileged case [4] by exposing a permission that will not be used by the app. These tools are known as code smells analysis tools. They analyse statically an Android application and raise up the different code smells related to different metrics such as readiness, performance and security. We focused on security related metrics by installing and testing each tool in the above presented malicious app. None of them detected the extra used permission presented in the app. This project demonstrated that those tools that are well known by their performance in tracing code smells are not adapted to this kind of specific analysis in Android applications.

Since the mentioned academic tools are not available, we carefully studied the published paper related to each one. A comparative view is presented in Table I. These works provide a PM indicating for each API call to the application framework which permission(s) is (are) required. Except the oldest work Curbing [8] that builds the PM starting from a manual analysis of the official Android documentation, the remaining works are based on the PM of pscout[16] that statically analysed the application framework having the SDK version 2.2. After analysis of these works, we underline the following limits:

- Outdatedness. All the solutions are completely/partially outdated due to the evolution of permissions (number and specification) and APIs. All of them are based on pscout PM[16] that recognizes only permissions used for API level 12. If the PM is incomplete and inaccurate, it may lead to false negatives when detecting the used permissions.
- Incompleteness. All works are based on static analysis approaches. On the application side, static analysis serves to detect the existing API calls, and the list of declared permissions. However, dynamic development techniques such as Reflective calls to API are commonly used, and not addressed by these approaches leading to unsound results. On the application framework side, reusing pscout PM[16] does not provide full coverage for all API levels, and cannot handle reflective calls, which is an open problem for static analysis tools.
- Availability. We were surprised that none of these tools is available to be used in real development projects. As we found during the experimentation of the industrial tools that they are not able to detect over-privileged applications, academic tools are also not publicly available nor ready to be used in the context of industrial projects.

Next section presents our contribution to reduce these limitations.

III. Over-privileged Detection with Permdroid

We aim to improve the analysis results by performing static and dynamic analysis in a combined app and framework levels. The goal is to provide a better PM enabling developers to detect if their apps are over-privileged or not (see Figure 1).

Through static analysis, we first investigate the Program Structure Interface (PSI) provided by the IDE to determine the list of declared permissions, and the API calls performed by

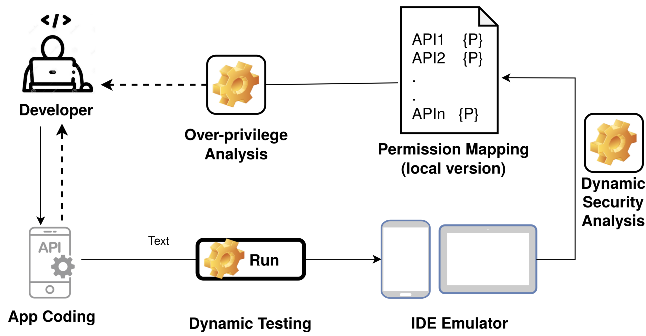

Fig. 1: Analysis process of Android apps

the app. We combine our analysis by monitoring the invoked APIs of system services during the dynamic testing of the app on the IDE emulator. This enables to collect all the API references performed during the app execution (including reflective calls).

Then we build a permission-mapping starting from the monitoring of access control checks performed by the Android framework. The goal is to collect permissions checked by the framework for each API call. To reach this goal, our tool PermDroid implements a dynamic testing service that generates input API calls and invokes the APIs of related system services. The framework runs the required security checks and lists the permissions required for each API call.Collecting the set of tuples `<api, permissions>` enables PermDroid to built a stand-alone PM set that will be used as a reference to investigate if the declared permissions by the app represent the optimal (minimum) set that could be used regarding the API calls that are performed.

While performing dynamic analysis enables to obtain complete and sound permission-mapping, there are some external factors that can explicitly affect the quality of the constructed PM such as (1) the analysis time & speed [The app execution time will not be sufficient to invoke all API system services and collect all the required permission corresponding to them] and (2) the rapid evolution of Android specification, which could make the solution not suitable to new framework versions. To deal with these challenges, we propose PermDroid to perform the analysis process in a collaborative mode as described in Figure 2, based on a crowd-sourcing principle.

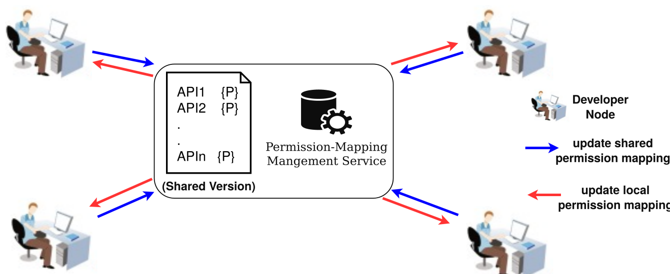

Fig. 2: Our Dynamic Collaborative Approach

Having several devices running in parallel reduces the analysis time and provide a wider coverage. On the other side it enables to automatically reproduce scenarios for new Android versions. The results of each analysis node will be published to enrich the permission-mapping directory to effectively detect over-privileged applications.

IV. Conclusion & Perspectives

Android developers face the security challenge of avoiding to program over-privileged applications, that may lead to vulnerabilities. After discussing the limits of existing works, we presented here a method to assist developers to avoid this issue. We propose a dynamic and collaborative approach to ensure a complete analysis coverage with accurate results, which is not dedicated to a specific Android API. The PermDroid plugin is currently experimented on at least 100 open source application. Future works will focus on extracting API calls for an application, install the crowd architecture and integrate the tool to real industrial development projects.

References

[1] J. Mitra, V.-P. Ranganath, T. Amtoft, and M. Higgins, "Sema: Extending and analyzing storyboards to develop secure android apps," *arXiv preprint arXiv:2001.10052*, 2020.
[2] W. Guo, "Management system for secure mobile application development," in *Proceedings of the ACM Turing Celebration Conference-China*, 2019, pp. 1–4.
[3] G. L. Scoccia, A. Peruma, V. Pujols, I. Malavolta, and D. E. Krutz, "Permission issues in open-source android apps: An exploratory study," in *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2019, pp. 238–249.
[4] "Ghera," Access 2021-12-25, https://bitbucket.org/secure-it-i/android-app-vulnerability-benchmarks/src/master/Permission/UnnecesaryPerms-PrivEscalation-Lean/.
[5] "Developer documentation," no date, accessed on 15/01/2022. [Online]. Available: https://developer.android.com/training/permissions/requesting
[6] Y. Aafer, G. Tao, J. Huang, X. Zhang, and N. Li, "Precise android api protection mapping derivation and reasoning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1151–1164.
[7] A. Dawoud and S. Bugiel, "Bringing balance to the force: Dynamic analysis of the android application framework," *Bringing Balance to the Force: Dynamic Analysis of the Android Application Framework*, 2021.
[8] T. Vidas, N. Christin, and L. Cranor, "Curbing android permission creep," in *Proceedings of the Web*, vol. 2, 2011, pp. 91–96.
[9] E. Bello-Ogunu and M. Shehab, "Permitme: integrating android permissioning support in the ide," in *Proceedings of the 2014 Workshop on Eclipse Technology eXchange*, 2014, pp. 15–20.
[10] G. Xu, S. Xu, C. Gao, B. Wang, and G. Xu, "Perhelper: Helping developers make better decisions on permission uses in android apps," *Applied Sciences*, vol. 9, no. 18, p. 3699, 2019.
[11] M. E. A. Tebib, P. André, O.-E.-K. Aktouf, and M. Graa, "Assisting developers in preventing permissions related security issues in android applications," in *European Dependable Computing Conference*. Springer, 2021, pp. 132–143.
[12] "Sonarqube," no date, accessed on 15/01/2022. [Online]. Available: https://docs.sonarqube.org/latest/
[13] "Findbugs," Access 15/01/2022. [Online]. Available: http://findbugs.sourceforge.net/
[14] "Checkmarx. the world runs on code. we secure it." no date, accessed on 15/01/2022. [Online]. Available: https://checkmarx.com/
[15] "Fortify," no date, accessed on 15/01/2022. [Online]. Available: https://www.microfocus.com/fr-fr/products/static-code-analysis-sast/overview
[16] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: analyzing the android permission specification," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 217–228.