Preventing Permissions Security Issues in Android: a Developer's Perspective

Mohammed El Amin TEBIB *, Pascal Andre†, 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*—Permissions related attacks are a widespread security issue in Android environment. Due to the misuse of the privileges, attackers steal the user rights and perform malicious actions. Most existing defence solutions are specified from end-users perspectives e.g. building anti-malwares and changing system configurations. In this paper we take the developers perspective because security should be a software design issue. We review existing approaches, we underline their limits and present our main contributions. The goal is to assist developers to prevent permissions related security flaws and to set permissions more effectively and accurately.**

I. Introduction

According to the Veracode team security report, 83% of Android apps have at least one security flaw[1]. As security is often a secundary concern for developers, most of the Android apps security problems are introduced during the development process[1]. In this study we focus on permission security issues identified at the development stage. Permission is a main security concept to ensure the privacy protection. It restricts the action performed by the ressource and controls the access to the operating system. A recent experimentation study [2] realised on 574 GitHub repositories of open-source Android apps, showed that permissions-related issues still are a frequent phenomenon in Android apps.

In Android apps, permissions are declared on the *manifest.xml* configuration file, and required at different stages: (1) system APIs interactions, (2) database access, (3) message passing system via intents, (4) invocation of specific protected methods in public APIs and (5) content provider data access. Android provides an official document to explain how to properly use each of them[2]. However, due to the continuous changes, this documentation becomes hardly readable to developers, leading to different security issues and drawbacks such as: 1) Wrong permission usage: due to some permissions similarity, developers may intentionally use wrong permissions. .e.g.: the use of *ACCESS_COARSE_LOCATION* instead of *ACCESS_FINE_LOCATION* 2) Permissions over-privilege: a widespread phoneme that occurs when the application declares more permissions than those used. The unused permissions could be exploited by hackers to perform malicious actions, especially ransomware. 3) Permissions under-privileged: occurs when the application uses more permissions than those declared, which does not conform with the transparency principle that each developers should respect. 4) Unprotected API occurs when developers forget to add an exception handler to some API methods, which may throw exceptions. Solving this non-exhaustive list of issues must be achieved at development time and not later. Our aim is to provide such an assistance tool, ensuring an effective analysis of the permissions used by the application, and and highlighting the security flaws of the apps under development.

In the rest of the paper, we review existing approaches proposed in the literature to assist developers avoid these issues, then we underline limits and presents our main contributions.

II. Background

Let briefly review existing methods focused on the detection and analysis of permissions related security issues in Android environment from developers perspectives.

PermCheck tool [4] is a static analysis tool that helps the developer to build apps with the least required permissions. PermCheck is based on a Permission-API database in which each permission is mapped to one or more methods, or one or more classes. During the static analysis of the Android application source files, each API reference is checked against the API database. The main limitation of this tool is that the permission-function pairs used by the tool is generated for only Android API level 2.2. also they consider only permission required for API in the main Java sources.

PermitMe [5] is an Eclipse plugin that improves the API-permission mapping proposed by, PermCheckTool [4] by analysing the Java source code. but again, only it could works for Android apps with April level 2.2.

PerHelper [6] handles *permissions under-privileged*. However it does not specify the required permissions present in (1) native code where a several API references that require permissions exist. and also (2) intents and broadcast Receivers, where privileged intent could be used.

Research	Year	Approach	Supported Api Level
PermCheck [4]	2011	static	2.2
PermitMe [5]	2014	static	5.0
PerHelper [6]	2018	static	26

TABLE I
PERMISSIONS CHECK ASSISTANCE TOOLS

[1]https://www.veracode.com/state-of-software-security-report
[2]http://developer.android.com/sdk/api_diff/30/changes

We underline the following limits in the existing works: 1) Many existing solutions would be completely/partially outdated due to the evolution of permissions (number and specification) and APIs. If the permission map is incomplete and inaccurate, it may lead to the false negatives when detecting used permission. 2) All the existing analysis approaches are static, while the most used IDEs for developing Android apps support emulator to run locally and testing their development. Adding a dynamic analysis during the development stage will provide clearer security guidelines for developers. For example, in case of the used APIs, dynamic analyses deliver information on whether those APIs misuse the required permissions or not. 3) There is a significant lack on native code analysis approaches (Java Native Interface). 4) For evaluating the effectiveness of proposed tools, no open source database exist that focus on the above mentioned permission issues. 5) All the approaches are based on the Abstract Syntax Tree (AST provided by the tool) which make the analysis process difficult and may lead to an accurate calculations.

III. Permissions Security Analysis and Detection

Security is not a user duty except some main privacy rules. Security is neither a developer's concern because his goal is mainly to develop an efficient and ergonomic running application. Developers are usually not security specialists. Consequently our goal is to provide assistance at development time and we target the following main research contributions:

- A framework based on Model Driven Reverse Engineering called PermDroid. Instead of analysing the source code from its AST, we propose to use a model based analysis. We started using the Modisco tool[3] which helps to understand and analyse the Android apps by providing a graphical representation of the program AST structure. Having models enables to use many powerful model transformation abilities.
- We investigate hybrid analysis techniques (static+dynamic) at developpement time. Static analysis will serve to: 1) determine the requested permissions declared on the manifest configuration file, 2) generate permissions used (PUs) by the application through inspecting the permission related APIs, 3) inspect methods involving sendind and receiving intents, (4) and methods involving the management of content provider. Wherease dynamic analysis could assist in 1) handling the dynamic loading of classes from embedded `.jar` and `.apk` files, 2) handling Java refection which is used by more than 57% (2013) of Android apps [8] which provides a program the ability to inspect classes, methods, interfaces and fields at runtime without knowing the names of the classes and methods in prior.
- To increase the detection accuracy of our solution in determining overprivilidges and underprivilidges permissions. Build a permission map that identifies what permissions are needed for each API call for all API level versions is needed. As a starting point we aim to use the permissions dataset proposed recently by Almomani et .al [7] that lists all the permissions starting from API level 1 to API level 30
- Formal Analysis of Security Policies. Formal methods have proven their potential and high accuracy either in Android malware detection via model checking [10], or in the modelling and analysis of the permission framework at system level [11]. We aim to combine to our tool a lightweight formal methods to automatically infer security-relevant properties related to intra-app granting permissions. Having the application model of Modisco will help to generate a formal model specifying 1) the permission level, 2) the permission related events and 3) component invocation (this list could be extended). Formal models enable the developers to rigorously check the defined security policies, and detect potential permission violations or misuse during development.

[3]https://www.eclipse.org/MoDisco/

IV. Conclusion & Perspectives

In this paper we presented the limits in existing works related to permissions attempting to assist developers take privacy as a first class element during the implementation of Android apps. In the following, we plan to continue the implementation of the proposed approach with static and dynamic verification modules and to integrate it as an Android Studio/IntelliJ based tool called "PermDroid".

References

[1] Guo, Wei. "Management system for secure mobile application development." Proceedings of the ACM Turing Celebration Conference-China. 2019.

[2] Scoccia, G. L., Peruma, A., Pujols, V., Malavolta, I., & Krutz, D. E. (2019, September). Permission issues in open-source Android apps: An exploratory study. In 2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM) (pp. 238-249). IEEE.

[3] AU, Kathy Wain Yee, ZHOU, Yi Fan, HUANG, Zhen, et al. Pscout: analyzing the Android permission specification. In : Proceedings of the 2012 ACM conference on Computer and communications security. 2012. p. 217-228.

[4] Vidas, T., Christin, N., & Cranor, L. (2011, May). Curbing Android permission creep. In Proceedings of the Web (Vol. 2, pp. 91-96).

[5] Bello-Ogunu, E., & Shehab, M. (2014, October). Permitme: integrating Android permissioning support in the ide. In Proceedings of the 2014 Workshop on Eclipse Technology eXchange (pp. 15-20).

[6] Xu, G., Xu, S., Gao, C., Wang, B., & Xu, G. (2019). PerHelper: Helping Developers Make Better Decisions on Permission Uses in Android Apps. Applied Sciences, 9(18), 3699.

[7] Almomani, I. M., & Al Khayer, A. (2020). A Comprehensive Analysis of the Android Permissions System. IEEE Access, 8, 216671-216688.

[8] Hoffmann, J., Ussath, M., Holz, T., & Spreitzenbarth, M. (2013, March). Slicing droids: program slicing for smali code. In Proceedings of the 28th Annual ACM Symposium on Applied Computing (pp. 1844-1851).

[9] NIRUMAND, Atefeh, ZAMANI, Bahman, et TORK LADANI, Behrouz. VAnDroid: A framework for vulnerability analysis of Android apps using a model-driven reverse engineering technique. Software: Practice and Experience, 2019, vol. 49, no 1, p. 70-99.

[10] Iadarola, G., Martinelli, F., Mercaldo, F., & Santone, A. (2019, October). Formal methods for Android banking malware analysis and detection. In 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS) (pp. 331-336). IEEE.

[11] He, X. (2017, July). Modeling and Analyzing the Android Permission Framework Using High Level Petri Nets. In 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) (pp. 232-239). IEEE.