안드로이드 기반 IoT 애플리케이션의 Permission Smell 탐지 방법

오지강 ^O^, 천신, 이욱진
한양대학교 컴퓨터공학과
wzq0515@hanyang.ac.kr, xxtx0122@hanyang.ac.kr, scottlee@hanyang.ac.kr

Permission Smells Detection for IoT Applications on Android Platform

Zhiqiang Wu^O^, Xin Chen, Scott Uk-Jin Lee
Department of Computer Science & Engineering, Hanyang University

Abstract

With the rapid development of the Internet of Things (IoT), the end-users may remotely control their IoT devices via their Android devices. The developers may request some unnecessary permissions and request sensitive permission access every time without checking the granted status, which leads to the risk of over privilege and privacy leakage. In this paper, we propose a methodology used to detect two kinds of permission smells on Android devices by similarity coefficient and depth-first search algorithm to assist the developers in detecting the Android-specific smells during development.

1. INTRODUCTION

Internet of Things (IoT) devices extensively used in the smart home, industry automation, agricultural management, and other smart environments have become an essential part of modern society. Generally, the vendors of IoT devices made Android applications used by their end-users to control the corresponding IoT devices via network or Bluetooth remotely. However, in this very competitive market, developers pushed to release new versions of their applications to retain users regularly. With such pressure, developers may unintentionally employ bad design or implementation, which leads to introduce the code smells [1]. Android-specific code smells are different from the traditional Object-Oriented Program since they often refer to a misuse of the Android framework [2].

Nevertheless, IoT applications request lots of permissions to support their functionalities, including normal level, dangerous level, and protected level permissions. The developers usually misuse permissions in Manifest.xml and do not check the granted status of dangerous permission with the bad practice. These bad practices under developing can be caused by either carelessness by the developers or intentionally by an attacker so that the applications exist the risk of over privilege, privacy leakage, and reliability. Therefore, assisting developers in building secure and reliable IoT applications with deadline pressure is a considerable challenge. Nevertheless, the security testing costs extra time by the security team after development.

In this paper, we propose a methodology to detect the relevant permission smells on IoT applications such as unnecessary permissions [2] and missing checkSelfPermission [3]. The two kinds of permission smells detected by similarity coefficient and depth-first search algorithm, respectively. As a result, the developers may run the algorithm to dig out the permissions smells just-in-time, which might improve the security and quality during development.

The remainder of this paper is organized as follows. Section 2 presents the related work on the permission smells of Android applications. Section 3 elaborates on the methodology for detecting the permission smells. Finally, we discuss the conclusion of this paper in Section 4.

2. RELATED WORK

In this section, we discuss related work on permission smells. Android application is the event-driven program that has multiple entry points and its typical mechanism like permission and Inter-Component Communication (ICC) [4], which is quite different from the traditional Java program. Therefore, Android-specific code smells [2] are defined by Ghafari et. al. There is a total of 28 Android-specific smells. Nevertheless, the permission-related smells are easily omitted by developers, which also make the security and quality issue.

2.1 Unnecessary Permissions

In general, the developers request more unnecessary permissions in the Manifest.xml file to satisfy the functionalities. However, the unnecessary permissions may cause the over privileges and the risk of privacy leakage by attackers. Currently, several previous studies [5-6] analyzed the descriptions of applications from official markets to verify the unnecessary permissions from the description. The description may give a fantastic introduction of its functionalities, which may attract more users to download. However, the secondary functions tend to unmentioned in its description, which cannot expose all used permissions.

2.2 Missing checkSelfPermission

Since Android 6.0, the dangerous permissions must be granted by users when they required at the first time. Therefore, the methods that use the relevant APIs should check whether the permission was granted by users firstly. The developers may request permission access every time without using checkSelfPermission() to check the granted status of permission, which will cause a high chance of the application crashing or not performing its intended functionality. [3].

Overall, there are lots of research works on Android security with permissions. However, these researches focus on the cacoethic usage of permissions from the view of security analysts after development, which may not provide the best practice of developing reliable applications during development.

3. METHODOLOGY

In this paper, we present a static analyses approach to detect the misuse of permissions on the Android source code under development. Figure 1 shows the overall structure of permission smell detection.

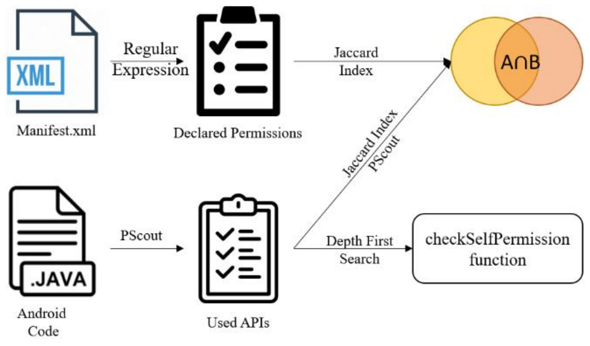

Figure 1 Structure of Permission Smell Detection

3.1 Permissions and APIs Extraction

In this section, we descript two methods to extract the declared and used permissions from Manifes.xml and Java source code, respectively.

The declared permissions have two types of permission in the Manifest.xml file. One is single permission that has multiple callable APIs. The other one is a permission group that contains a set of dangerous permissions such as SMS and PHONE. A permission group is granted by users, which means its attached permissions are also granted. To cover all declared permissions, we wrote Regular Expression to filter the permissions and permission groups. Moreover, the permission groups should be mapped to a set of permissions since an application may not use all permissions from the group.

To analyze the actual usage of permissions in the source code, we made use of the API call for permission mappings extracted by PScout [7]. PScout can generate the Android APIs to permission mappings, but the various APIs may map to the same permission and called multiple times. In this case, the duplicated called APIs removed as the first step after extraction. Then, the APIs mapped into a permission set that stores the result of actually used permissions by PScout. If the mapped permission from PScout does not exist in the result, the permission appended at the end of the result.

3.2 Unnecessary Permissions Detection

In the previous section, we extracted the declared permissions from the Manifest.xml file and the used permissions in the source code. Here, we employed the Jaccard Index [8] to calculate the similarity coefficient between the declared and used permissions, which is a statistic used for assessing the similarity and diversity of sample sets. The Jaccard Index equation is shown as follows.

$$J(p_d, p_u) = \frac{|p_d \cap p_u|}{|p_d| + |p_u| - |p_d \cap p_u|}$$

where p_d denotes a set of permissions which declared in the Manifest.xml, p_u denotes a set of permissions that are used in the source code. In this case, p_d contains each permission of p_u for an executable application. In the ideal situation, p_d same as p_u. If so, there is not existing unnecessary permission smell in that application and $J(p_d, p_u)$ equals 1. Otherwise, the values of the Jaccard Index should be [0,1), which indicates that there are unnecessary permissions declared in the Manifest.xml file.

In this paper, we determine there is not existing unnecessary permission smell as the following two conditions. First, the value of the Jaccard index is 1, which the declared and used permission are the same without any redundancy. Second, the redundant permissions are normal level permissions when the value of the Jaccard Index less than 1. The unnecessary permissions in the declared permissions are dangerous level or protected level permissions that may cause the risks of potential security utilized by attackers such as over privilege and privacy leakage.

3.3 CheckSelfPermission Function Search

As section 2.2 mentioned above, the developers did not use checkSelfPermission() method to check whether users granted the required permission. However, the application repeatedly requests the granted permissions, which has a high chance of the app crashing or not performing its intended functionality. In this paper, we employed the backward searching algorithm to check whether the developers use checkSelfPermission() to check the granted status of dangerous permission before using the corresponding APIs to improve its internal quality.

```
Input: api, method
1: function checkStatus(api, method)
2:     code ← method.split()
3:     permission ← gerPermission(api)
4:     while code is not Null do
5:       statement ← code.pop()
6:       if 'checkSelfPermission' in statement then
7:         if permission in statement then
8:             return True
9:         end if
10:      end if
11:      if other method in statement then
12:        if checkStatus(api,method) then
13:            return True
14:        end if
15:      end if
16:    end while
17:    return False
18: end function
Output: checkSelfPermission is used or not
```

List 1 Algorithm that checks 'checkSelfPermission'

List 1 presents a pseudo-code of the algorithm we designed for function call analysis. It checks whether the checkSelfPermission() method is called before the use of sensitive APIs. The algorithm uses the depth-first search to search the function call. Two conditions determine the granted status has checked before its usage. First, one statement has checked the granted status of permission by checkSelfPermission() function (Line 6-10). Second, the other method called to check the granted status with the target function (Line 11-15).

If this algorithm returns true as a result, the developers have utilized the checkSelfPermission() method to check whether the required permission granted. Otherwise, the application requests the corresponding permission access every time, which indicates a permission smell.

4. CONCLUSION

In this paper, we proposed a methodology that can detect two kinds of permission smells on IoT applications, named unnecessary permissions and missing checkSelfPermission. The two permission smells detected by similarity coefficient, and a static analysis algorithm, which may help developers detect the Android-specific smells just-in-time. Moreover, this methodology provides a novel insight for developing secure software during development.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2019R1H1A1101270).

REFERENCE

[1] Z. B. Celik, G. Tan and P. McDaniel, "IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT", in Proc. of Network and Distributed Systems Security Symposium, 2019

[2] P. Gadient, M. Ghafari, P. Frischknecht and O. Nierstrasz, "Security Code Smells in Android ICC", Empirical Software Engineering, vol. 24, 2019, pp. 3046-3076

[3] C. Dennis, D. E. Krutz and M. W. Mkaouer, "P-Lint: A Permission Smell Detector for Android Applications", in proc. of International Conference on Mobile Software Engineering and Systems, 2017, pp. 219-220

[4] P. Kong, L. Li, J. Gao, K. Liu, T. Bissyande and J. Klein, "Automated Testing of Android Apps: A Systematic Literature Review", IEEE Trans. on Reliability, 2018, pp.1-23

[5] R. Pandita, X. Xiao, W. Yang, W. Enck and T. Xie, "WHYPER: Towards Automating Risk Assessment of Mobile Applications", in proc. of USENIX Security Symposium, 2013, pp. 527-542

[6] Z. Wu, X. Chen and S. U. J. Lee, "Identifying Latent Android Malware from Application's Description using LSTM", in proc. of International Conference on Information, System and Convergence Applications, 2019, pp. 40-42

[7] K. W. Y. Au, Y. Zhou, Z. Huang and D. Lie, "PScout: analyzing the Android permission specification", in proc. of ACM Conference on Computer and Communications Security, 2012, pp. 217-228

[8] https://en.wikipedia.org/wiki/Jaccard_index