Android Mobile Permissions

Loic Niragire

TIM-7010 Computer Networks & Mobile Computing

Dr. Phillip Davis

TABLE OF CONTENTS

1. Introduction

The authors in this paper analyze a recent publication on Android Mobile Permissions and discuss improvements in this area of research. Application developers on Android utilize a platform-provided API to access the phone's hardware, settings, and user data. Meanwhile, access to security-relevant parts of the API is managed at the installation time through what is referred to as a permission system. During the installation, the application states all required security access and only proceeds once granted by the user. It is recommended that developers follow the least-privilege model with their permission requests to not over-expose users' devices by requesting more access than necessary (Felt, Chin, Hanna, Song, & Wagner, 2011).

The latest permission model introduced in Android 6.0 Marshmallow allows runtime permission requests rather than the previously discussed model. Where all required permissions are requested at the installation time (Android Developers, 2019), in this model, request permissions are only issued when required. So rather than requesting location access at installation time and therefore tracking the device, the application would request location detailed only when needed for a given task. This is a significant departure from the previous model and highlights the issue of over-privilege in mobile application development.

2. Analyzed Work

I analyzed a research publication sponsored by Google on reducing permission requests in mobile apps where the authors explored the use of deep learning for detecting unnecessary permission requests (Peddinti, et al., 2019). Before launching an app, the model compares permission requests between the given app and a set of functionally similar apps. Therefore, giving developers a visibility into commonly acceptable permission requests from the user's perspective. Thus, the app alerts the developer if requested permissions deviate significantly from the established baseline.

2.1.1. Significance - User's perspective

By computing a baseline of requested and granted permissions for a set of apps, essentially, this model establishes a general sense of the user's perspective. This awareness is a significant insight for a developer before launching an app. More importantly, it allows developers to be proactive if requesting more than the standard set of permissions - perhaps by providing a detailed explanation for permissions in question.

2.1.2. Significance - Third-party library awareness

A common source of unnecessary permission requests results from poor usage of third-party libraries, most notably, configuration issues. Most novice developers simply opt for default configurations and therefore request all permissions requested by a library (Chitkara, Gothoskar, Harish, Hong, & Agarwal, 2017). By compiling a set of functionally related apps and their permissions, developers can fine tune their third-party library configurations to remove unnecessary requests.

3. The Process

The process utilized in this research work is comprised of three parts. The first part deals with identifying a set of functionally related apps for a given app. Meanwhile, the second part handles the comparison between app requested permissions and its related peers. Lastly, the third part is concerned with notification whereby developers are notified of any significant permission request deviation relative to their peers.

3.1.1. Identifying similar apps – distance function

The first challenge to overcome in this work was identifying similar apps for a given app. A naïve approach to this could be grouping by app categories, but this does not capture functionality. Different apps can belong to the same category while offering different functionalities. Another approach considered relies on user behavior while browsing in Google Play. In other words, exploring user clicks as they visit suggested apps in Google Play and linking them as similar. This iterative clustering method, known as "user behavior clustering (UBC)," is mainly used to find interesting suggestions. However, this approach is not optimized for functionality comparison. Also, it prefers apps in the same primary language and locale.

To identify app similarity, the authors developed a deep-learning algorithm that creates an embedding based on word2vec for each app by using the app's description, name, and category. Additionally, the embedding was also computed based on UBC data. The similarity between two apps was measured using the cosine distance of their app vectors from the word2vec algorithm. Therefore, establishing the peers of an app as its nearest neighbors based on this distance. For instance, using this model, the closest peer to Gmail was found to be another email app with a cosine distance of 0.41 while weakly related apps had distances such as 0.68, 0.80, and 0.85.

Finally, an arbitrarily peer similarity cutoff score was decided to produce between 150 to 200 high-quality peers per app.

3.1.2. Developer notification

With peer apps identified, an alert is sent, through Google Play Console, to developers when their apps ask for permissions requested by very few functionally similar apps. Future work suggested expanding these notifications into other development environment.

4. Proposed Improvement/future work

The proposed method in this paper makes an implicit assumption that requested permissions by identified peers are necessary. Therefore, subsequent apps could request unnecessary permissions if they are consistent with their peers. In such cases, no alert is captured. One proposed improvement to this work involves a dynamic analysis of peers' requested permissions. Additionally, alerts are currently supported only through Google Play Console. Future work will explore alert support in other developer tools such as Android Studio, Gradle, and more.

5. Related work

Previous research works on permission requests have explored static code analysis tools for detecting over-privileged scenarios (Felt, Chin, Hanna, Song, & Wagner, 2011). The produced tool from this work, namely Stowaway, can compare required permissions for invoking specified API calls versus required permissions. The study applied Stowaway to over 940 Android applications and found that over a third of them required unnecessary permissions, which can be attributed to developer confusion. Stowaway cannot handle complex reflective calls. Meanwhile, 61% Of the tested applications use Java reflection to make API calls.

Other related works have explored permission frameworks to mediate access to secure resources using plugins. The basic idea being that users can rely upon various plugins to manage

permission requests. Permission frameworks, therefore, are meant to provide an abstract layer between the app requesting permission and the plugin handling those requests. This approach provides an extra isolation layer between apps and plugins by defining a communication interface (Raval, Razeen, Machanavajjhala, Cox, & Warfield, 2019). The authors in this paper developed a permission framework for Android - named DALF. Thus, each permission request is communicated to DALF, passing the request to the plugin mediating that app-resource combination.

A recent study at the University of Toronto explored a novel approach to permission mapping by focusing on API source code semantics (Shim & Jung, 2020). Using natural language processing techniques, researchers extracted permission information from source code comments and java documents. Similarly, they extracted API signatures from source code and mapped required permissions to their corresponding APIs. Additionally, they categorized all permissions and APIs based on analyzed comments and generated a measure of potential risk level from misuse. Their approach attempts to address the lack of permission oversight in Android. With 165 permissions currently defined in Android, there is no mechanism to inspect all list of APIs with required permissions. Consequently, developers tend to request more permissions than necessary, given actual API calls made, and it is not trivial to detect these scenarios without an API-permission map. Their study obtained a list of API-permission map with 3,012 APIs from Android API version 10.

References

Android Developers. (2019). Retrieved from developer.android.com: https://developer.android.com/training/articles/user-data-permissions.html?hl=en

Atkins, A. S., Ali, A. H., & Shah, H. (2006). Extending E-Business Applications Using Mobile Technology. ACM - Mobility '06, 1-6.

Chitkara, S., Gothoskar, N., Harish, S., Hong, J. I., & Agarwal, Y. (2017). Does this App Really Need My Location?: Context-Aware Privacy Management for Smartphones. Wearable Ubiquitous Technol. . ACM.

Felt, P. A., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011). Android Permissions Demystified. ACM - CCS'11, 627-637.

Peddinti, S. T., Bilogrevic, I., Taft, N., Erlingsson, ú., Anthonysamy, P., & Hogben, G. (2019). Reducing Permissio Requests in Mobile Apps. Internet Measurement Conference. Amsterdam, Netherland: ACM.

Picco, G. P., Julien, C., Murphy, A. L., Musolesi, M., & Roman, G.-C. (2014). Software Engineering for Mobility: Reflecting on the Past, Peering into the Future. ACM - FOSE'14, 13-28.

Raval, N., Razeen, A., Machanavajjhala, A., Cox, L. P., & Warfield, A. (2019). Permissions Plugins as Android Apps. Mobile Systems, Applications, and Services - Mobisys (pp. 180-192). Seoul, Republic of Korea: ACM.

Shim, H., & Jung, S. (2020). Semantic-aware Comment Analysis Approach for API Permission Mapping on Android. International Conference on Natural Language Processing and Information Retrieval (pp. 61-69). Seoul, Republic of Korea: ACM.