Technische Universität München

Automatic Repackaging of Android Apps

Bachelor Thesis

Supervisors: Prof. Dr. Alexander Pretschner, Alei Salem
Email: pretschn, salem @ in.tum.de
Phone: +49 89 289 – 17, 314
Starting date: Immediately

Fakultät für Informatik
Lehrstuhl 4
Software and Systems Engineering
Prof. Dr. Alexander Pretschner

Boltzmannstraße 3 85748
Garching bei München

Tel: +49 89 289 17885,
+49 89 289 17314

Web: http://www22.in.tum.de

Context

The Android app acquisition ecosystem continues to thrive courtesy of the willingness of users to acquire new apps. Such willingness hinges on the trust Android users have in the apps they download, their developers, and the marketplaces on which they reside. Some popular apps, such as Angry Birds for instance, have been installed 100 to 500 million times [3]. This trust in genuine apps has recently been leveraged by malware authors as a novel method to write and distribute their malicious instances, known as *repackaging*.

Repackaging, also known as *piggybacking*, is made possible due to the ease of decompiling/disassembling and reverse engineering Android apps. Typically, a malware author would download a renowned app (e.g., Angry Birds), disassemble it, inject a malicious payload within the original code, repackage it, and upload it to an app marketplace, optionally with a different name or theme. Unable to distinguish between repackaged and original apps, Android users are tricked into voluntarily downloading and installing malicious apps onto their devices. Furthermore, in order to increase the stealth of their instances, authors of repackaged malware design their instances such that the malicious payload triggers upon the realization of a pre-set condition (e.g., location-, time-, or logic-based).

Needless to say, the spread of repackaged malware (especially on third-party app marketplaces) undermines the trust Android users have in legitimate apps, effectively jeopardizing a fundamental cornerstone on which the Android platform stands. Consequently, various efforts have been attempting to study and detect this breed of Android malware (e.g., [1][4][2]). Nevertheless, there is a lack of malware instances that adopt the aforementioned technique. This forces researchers to either evaluate their analysis/detection approaches on outdated datasets such as [5], or to manually traverse app marketplaces–in a substantially time-consuming process–in pursuit of repackaged malware. Thus, there is a desparate need for a (customizable) mechanism to generate malicious, repackaged versions of legitimate apps on demand, so as to facilitate the process of analyzing and detecting Android piggybacked malware.

Goal

In this thesis, we aspire to build a tool that enables researchers to generate piggybacked versions of benign Android apps on demand. The tool should support different types of triggers and malicious payloads, effectively enabling its users to control how their piggybacked malware behaves. Furthermore, the tool should be extensible to allow the users to seamlessly add new payloads/triggers mimicking the ones adopted by the real world malware instances. The typical process of repackaging can be seen in figure 1.

There are two dimensions to evaluating the implemented tool. Firstly, the tool must not destabilize the original functionality of the repackaged app after injecting the triggers/payloads specified by the user. Secondly, the code injected by the tool needs to abide by the design of the malware instance dictated by the user. In other words, the payloads must only be triggered upon the realization of the pre-specified trigger, and carry out its intended functionality. We consider those two dimensions in evaluating our tool.

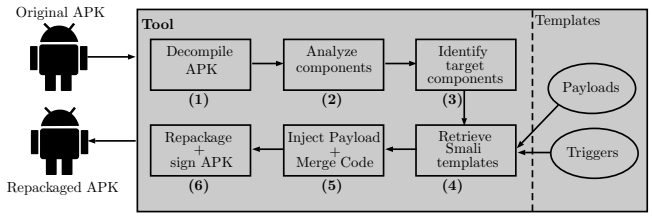

Figure 1: The components and process of the repackaging tool.

Given a set of legitimate apps that we download from the Google Play store (e.g., the top 50 free apps), we plan to generate multiple versions of those apps using the developed tool and random combinations of triggers and payloads. To evaluate the first dimension, we will compare the performance and stability of the apps before and after repackaging using fuzzing tools, such as Droidutan. Evaluating the second dimension (i.e., the injected code's functionality) can be carried out in a semi-automatic manner, in which we either use scripts to generate repackaged apps with triggers meeting the current system conditions (e.g., the current date) to guarantee triggering the malicious payloads, or alter the system properties.

Work-plan

1. Design the repackaging process.
    a. Enumerating possible triggers and payloads.
    b. Research deployment and injection methods.
    c. Implement triggers and payloads.
2. Design the tool.
    a. Design triggers/payloads storage and retrieval process.
    b. Design the targeting module.
    c. Design the code merging process.
3. Implement tool.
4. Evaluate the implemented tool
    a. Acquire the benign apps from Google Play.
    b. Design evaluation of injected code functionality.
    c. Run stability and performance results before repackaging.
    d. Write script to randomize triggers/payloads.
    e. Generate repackaged versions on benign apps.
    f. Re-run stability and performance tests.
    g. Run functionality tests.
5. Writing of the final thesis containing:
    a. Description of the problem and motivation.
    b. State of the art survey of repackaged malware detection in Android.
    c. Rationale for using the selected techniques.
    d. Implementation description.
    e. Evaluation.
    f. Conclusion and further work.

Deliverables

- A virtual machine containing the implemented framework.
- The thesis document in accordance with the TUM guidelines.

References

[1] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. Understanding android app piggybacking: A systematic study of malicious code grafting. *IEEE Transactions on Information Forensics and Security*, 12(6):1269–1284, 2017.

[2] Yuru Shao, Xiapu Luo, Chenxiong Qian, Pengfei Zhu, and Lei Zhang. Towards a scalable resource-driven approach for detecting repackaged android applications. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 56–65. ACM, 2014.

[3] Google Play Store. Angry birds, 2017.

[4] Haoyu Wang, Yao Guo, Ziang Ma, and Xiangqun Chen. Wukong: a scalable and accurate two-phase approach to android app clone detection. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pages 71–82. ACM, 2015.

[5] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95–109. IEEE, 2012.