Technische Universität München

Comparing Stimulation Techniques for Detecting Android Piggybacked Malware

Master Thesis

Supervisors: Prof. Dr. Alexander Pretschner, Aleieldin 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

Android dominates the handheld devices market with more than 80% marketshare. Needless to say, such popularity encourages malware authors to write and distribute malicious applications (hereafter apps) that target Android devices. To smoothen the spread of their malicious apps, malware authors tend to design their instances to mimick the appearance and behavior of benign apps (e.g., games), effectively tricking users into voluntarily installing malicious apps on their own devices. This model resembles that of Trojan horses, which have been distributed and studied for decades on various platforms.

More recently, however, malware authors targeting Android have been adopting a more sophisticated technique to develop and distribute their malicious instances viz., repackaging [2]. In essence, malware authors leverage the ease of decompiling, modifying the source code, and repackaging legitimate Android apps, and repackage legtimate, trusted apps with malicious payloads [2][4][6]. This breed of malware is often refered to as either piggybacked malware [2][3] or repackaged malware [7][8].

The primary threat that repackaged malware poses is undermining user trust in legitimate apps, their developers, and the app distribution infrastructure. This can potentially have devastating effects on the entire Android ecosystem. Aware of the threat it poses, the research community has been studying Android repackaged malware in pursuit of methods to analyze and detect it. To elude dynamic-based analysis and detection methods, malware authors hide their malicious payloads by wrapping them with conditions that infrequently evaluate to true. Such conditions, also known as triggers, are manually designed by the malware author, and usually depend on system properties (e.g., date, time, GPS location, etc.), app/system intents and notifications (e.g., android.intent.action.BOOT_COMPLETED), custom values forwarded to the app via its authors (e.g., via SMS messages), or a combination of those [2][4]. Effectively, trigger conditions render the injected malicious payload dormant.

In order to be able to analyze and detect this breed of malware, there needs to be mechanisms that stimulate those dormant, malicious segments within piggybacked apps. Otherwise, the resulting runtime behavior of the malware instances under test will not depict the true nature of the app. Ultimately, detection mechanisms (e.g., using machine learning classifiers) will fail to recognize piggybacked apps as malicious.

Goal

There are already multiple efforts that attempt to force hidden, dormant segments within Android apps to run, such as ([4][5][3]). Unfortunately, they have not either been evaluated using small datasets, outdated datasets (such as [8]), or not been evaluated on malware at all. In this thesis, we attempt to compare two methods of stimulating dormant malicious behavior within Android apps (viz., *GroddDroid*[1] and *FuzzDroid* [5]). Moreover, we compare the detection results based on data generated from using those two methods against a random-based method that we developed at the chair. The two aforementioned

stimulation methods will be used to generate runtime behaviors (i.e., in the form of API

call traces) of Android benign and malicious apps, from which we plan to extract numerical features for classification. The detection module hinges on a variety of machine learning classifiers, and will adopt an active learning setting to provide the stimulation methods with feedback about their performance. Lastly, we will evaluate different tools using two datasets viz., MalGenome [8] and the more recent Piggybacked [2] datasets.

Work-plan

1. Setting up the tools *GroddDroid* and *FuzzDroid*.
    a. Reading the two papers.
    b. Downloading and setting up the tools.
2. Setting up mechanism to monitor app runtime behavior using both tools.
3. Designing the features to extract from the recorded runtime behavior.
4. Connecting the stimulation tools to machine learning classifiers.
5. Designing the feedback mechanism to suit each of the tools.
6. Connecting the classifiers back to the stimulation tools.
7. Evaluating the performance of both tools using the MalGenome and Piggybacked datasets.
8. 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.

References

[1] Adrien Abraham, Radoniaina Andriatsimandefitra, Adrien Brunelat, Jean-François Lalande, and Valérie Viet Triem Tong. GroddDroid: a Gorilla for Triggering Malicious Behaviors. In *10th International Conference on Malicious and Unwanted Software*, Fajardo, Puerto Rico, 2015. IEEE Computer Society.

[2] 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.

[3] Li Li, Daoyuan Li, Tegawendé François D Assise Bissyande, Jacques Klein, Haipeng Cai, David Lo, and Yves Le Traon. Automatically locating malicious packages in piggybacked android apps. In *4th IEEE/ACM International Conference on Mobile Software Engineering and Systems*, 2017.

[4] Xiaorui Pan, Xueqiang Wang, Yue Duan, XiaoFeng Wang, and Heng Yin. Dark hazard: Learning-based, large-scale discovery of hidden sensitive operations in android apps. 2017.

[5] Siegfried Rasthofer, Steven Arzt, Stefan Triller, and Michael Pradel. Making malory behave maliciously: Targeted fuzzing of android execution environments. In *Proceedings of the 39th International Conference on Software Engineering*, pages 300–311. IEEE Press, 2017.

[6] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Lorenzo Cavallaro. The evolution of android malware and android analysis techniques. *ACM Computing Surveys (CSUR)*, 49(4):76, 2017.

[7] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 317–326. ACM, 2012.

[8] 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.