Technische Universität München

Analysis and Detection of Malicious Behaviors in Repackaged Android Applications

Master Thesis

Supervisors: Prof. Dr. Alexander Pretschner, Alei Salem
Email: pretschn, salem @ in.tum.de
Phone: +49 89 289 – 17, 340
Starting date: Immediately

Fakultät für Informatik
Lehrstuhl 22
Software 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

Over the past decades malware authorship grew to become a profession. Nowadays, there exists a plethora of malware instances that span various targets, structures, motivations, and objectives [1]. Despite the different objectives they pursue, the majority of malware authors strive to prolong the period of time their instances remain operational within the infected system. In this context, malware instances are designed to evade the detection mechanisms employed by antiviral software. It is, therefore, not uncommon for modern malware to obfuscate its internal structure, alter its runtime behavior, trigger its malicious intents upon the realization of preset conditions or mimic the behavior of a benign application to evade detection [4].

Repackaging is one of the techniques recently adopted by malware authors to evade detection. In essence, malware authors download legitimate applications, implant their malicious segments within such applications, and distribute them as different versions of the legitimate applications. This practice is particularly common on Android platforms courtesy of their widespread and the trust users have in Android app distribution systems e.g. *Google Play Store* [3].

Embedding malicious segments within legitimate apps yields misleading representations of program behavior, in which malicious behaviors are scattered across normal program behavior. For instance, if a trace of API-calls is used as a representation of an application's behavior, the trace of a repackaged malware instance might comprise blocks of API-calls that depict the malicious behavior which will be scattered across the entire trace. This phenomenon, needless to say, confuses common detection mechanisms, which are usually machine learning-based. Thus, there needs to be a technique to identify and isolate such aforementioned malicious segments, so that the employed detection models could effectively recognize malicious behaviors within repackaged malware.

Goal

In this thesis, we attempt to devise a technique to detect repackaged malware via isolating the malicious segments that reside within their behavior representations. We plan to do so by adopting a semantic perspective to the program behavior. In other words, what a program is attempting to accomplish e.g. by issuing some API-calls. We believe that focusing on program semantics embosses the original intention of a program, even if such intention is hidden within the overall program behavior.

The proposed approach is going to be implemented in two phases. In the first phase, we will study non-repackaged malware instances and extract their behaviors. In order to remove any noisy behavior and focus on the semantics of the malicious behavior of such instances, we will implement and compare two approaches. The first approach attempts to remove segments that do not manipulate sensitive system resources e.g. the *Windows Registry*, effectively isolating how a malware instance typically manipulates such resources. The second approach attempts to identify and isolate the behaviors that statistically contribute the most to a program behavior being malicious/benign. This can be accomplished using associative-rules learning.

In the second phase, we will use the behaviors extracted from the first phase and attempt to match them to their counterparts extracted from repackaged malware. We hypothesize that the behaviors extracted during phase one will match segments within the repackaged traces, effectively recognizing repackaged malware behaviors. In order to evaluate the proposed approach, we use two types of datasets and measure the percentage of malicious behaviors our approach managed to recognize. Firstly, we extract malicious behaviors from the *Drebin* dataset of Android malware, and match them against behaviors from the *Genome* dataset comprising 1200 repackaged Android malware. We refer to this dataset as the "online" dataset, especially since it is generated by executing, stimulating, and monitoring such instances in order to extract their behaviors. Nevertheless, comprehensive automatic stimulation of Android apps is a challenging issue, given their complexity [2]. This might result into inaccurate representations of malicious behaviors.

In this context, we utilize the "offline" dataset, in which we manually and rigorously stimulate a small number of benign and malicious Android apps, extract their behavior, and use them to build a hidden Markov model. We will use the built model to further generate offline traces that probabilistically resemble the behaviors of Android apps.

Work-plan

1. Generate the datasets:
    a. Execute and store monitored behaviors of Android APKs.
    b. Manually run and stimulate randomly-selected malicious/benign Android APKs.
2. Implement the trace analysis modules (Phase I):
    a. Identify the API calls that manipulate system resrouces e.g. user contacts.
    b. Implement the noise-removal module.
    c. Research associative rules learning.
    d. Implement the malicious behavior isolation module.
3. Evaluate the proposed approach (Phase II)
4. Write the thesis document:
    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 and the extracted behaviors.
- The thesis document in accordance with the TUM guidelines.

Technische Universität München

References

[1] David Bell. *Cyberculture: The key concepts.* Psychology Press, 2004.

[2] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. Automated test input generation for android: Are we there yet? *arXiv preprint arXiv:1503.07217*, 2015.

[3] Symphony Luo and Peter Yan. Fake apps: Feigning legitimacy, 2014.

[4] Philip O'Kane, Sakir Sezer, and Keiran McLaughlin. Obfuscation: The hidden malware. *Security & Privacy, IEEE*, 9(5):41–47, 2011.

Fakultät für Informatik
Lehrstuhl 22
Software 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