Improve Dynamic Analysis Coverage in Android with DroidBot

Yuanchun Li, Hanno Lemoine and Hugo Gascon
EECS
Peking University

Abstract

As it is the case for malware targeting the desktop, dynamic analysis is also used for detection of Android malware. While dynamic analysis works by executing the target app in a real Android environment and monitoring the behaviors during runtime, its effectiveness relays on the amount of code it is able to execute, this is, its **coverage**. Because some malicious behaviors only appear at certain states, the more states covered, the more malicious behaviors detected. The goal of DroidBot is to help achieving a higher coverage in automated dynamic analysis. In particular, DroidBot works like a robot interacting with the target app and tries to trigger as many malicious behaviors as possible.

The Android official tool for this kind of analysis used to be Monkey, which behaves similarly by generating pseudo-random streams of user events. Instead, DroidBot understand the app by statically analyzing its code and dynamically analyzing its UI hierarchy. This knowledge is extremely helpful to generate relevant interactions with the app.

With the knowledge from static and dynamic analysis, DroidBot adopts a biased-random strategy to trigger more sensitive behaviors. Experiments had shown that DroidBot is better than Monkey in discovering sensitive behaviors of malwares.

Research Questions

DroidBot aims to address two research questions:

1. How to improve coverage in automated malware dynamic analysis? Currently, automated test input generation tools include Monkey[1], Dynodroid[4] and GUIRipper[5], etc. Monkey achieves the highest coverage according to an empirical study[1].
2. How to measure the coverage of malware dynamic analysis? For benign apps, the metric to measure the performance of dynamic analysis (or testing) is method coverage, line coverage and path coverage etc. However, for malware analysis, the traditional coverage does not reflect the effectiveness of testing tool on one hand, and is difficult to measure on the other hand.

Approach

Figure 1. Architecture of DroidBot

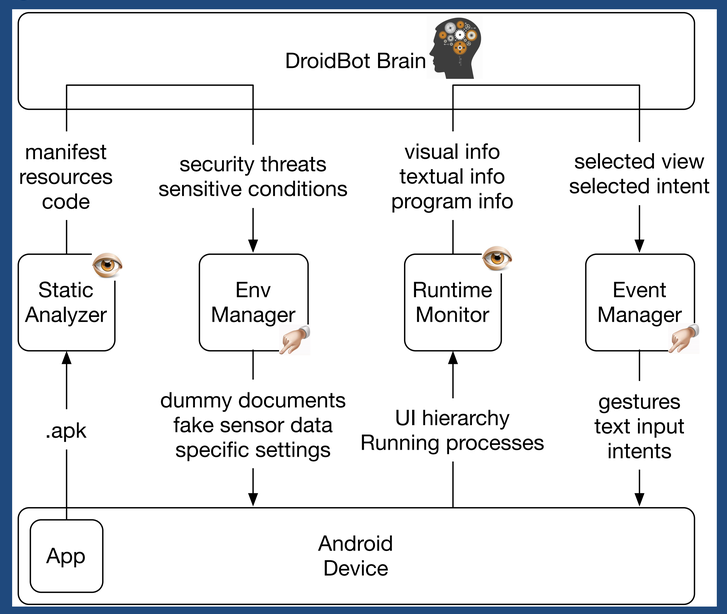
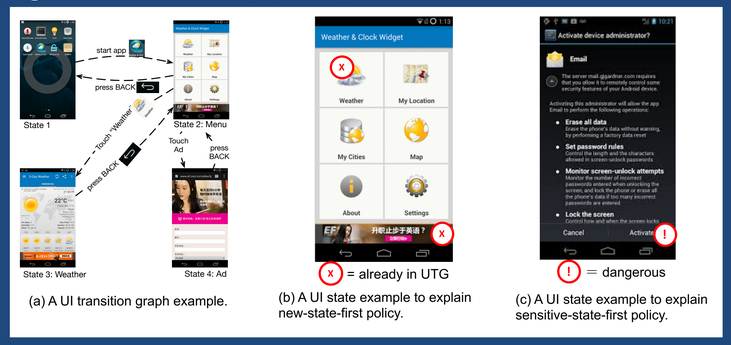


Figure 2. A UI transition graph example and UI state examples



(a) A UI transition graph example.

(b) A UI state example to explain new-state-first policy.

(c) A UI state example to explain sensitive-state-first policy.

The architecture of DroidBot is shown in Figure 1. DroidBot has five modules, Static Analyzer and Runtime Monitor extract static and dynamic information from app and device respectively. DroidBot Brain makes decisions based on the extracted information and performs operations before app installation and during app execution, to trigger more sensitive behaviors.

In EnvManager, we only consider permission information currently, and add dummy documents to device based on the permissions requested by app.

In EventManager, we use a biased-random exploration strategy when send events. We introduce a graph model of app which is called UI state transition graph (UTG). The graph is built at runtime. Figure 2(a) shows an example of UTG.

Based on the graph model, we implemented two biased-random strategies:

1. New state first. When selecting event to send, DroidBot prefers the events which will lead to a new UI state (aka. a state not in UTG). As shown in Figure 2(b), DroidBot will not consider to touch the Ad bar which is already in UTG.
2. Sensitive state first. As DroidBot is designed for malware analysis, it prefers sensitive states (states where sensitive behaviors happen) to safe states. For example in Figure 2(c), DroidBot will click the Activate button rather than the Cancel button.

Evaluation

We compare DroidBot with Monkey on effectiveness of triggering sensitive behaviors of Android malwares. We consider the amount of sensitive behaviors triggered and the speed of triggering sensitive behaviors.

Figure 3 demonstrates that DroidBot is able to trigger more sensitive behaviors than Monkey, and Figure 4 demonstrates that DroidBot can trigger sensitive behaviors faster.

Figure 3. Amount of sensitive behaviors triggered by DroidBot and Monkey.

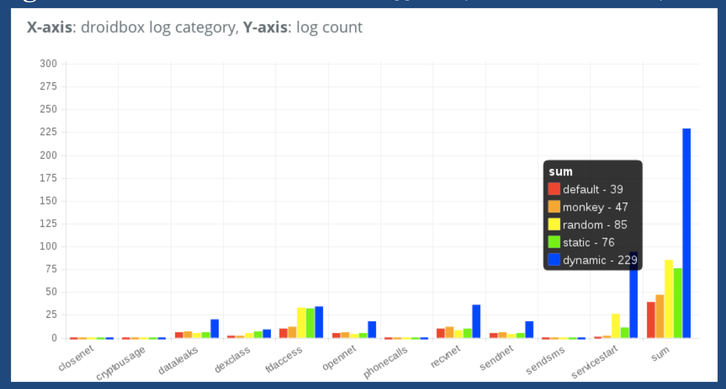
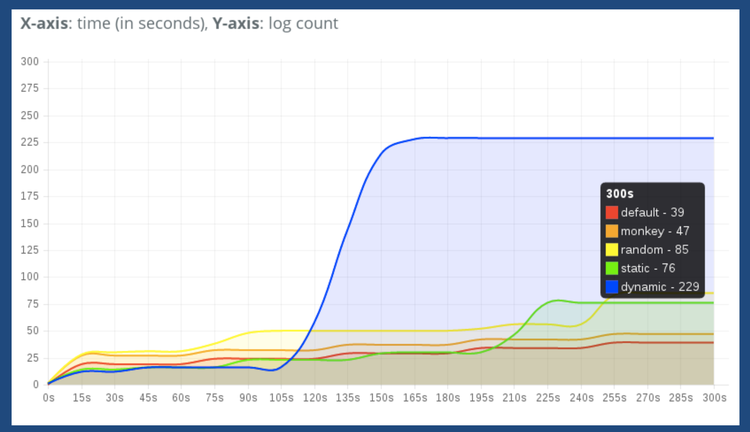


Figure 4. Speed of triggering sensitive behaviors of DroidBot and Monkey. "default" means starting the app and doing nothing, "monkey" means using ADB Monkey tool to interact with the app, and "random", "static", and "dynamic" are policies of DroidBot.



Conclusion

We propose DroidBot, an automated test input generation tool for malware dynamic analysis. DroidBot configure the test environment based on static information extracted from APK file in order to make the device more like a realistic phone, and uses biased random strategy based on a UI transition graph model to trigger more sensitive behaviors at runtime.

Through experiments with DroidBox, we find DroidBot is able to trigger sensitive behaviors in dynamic analysis, more and faster compared to Monkey, which is the state-of-the-art.

References

[1] Lantz, P., Desnos, A., & Yang, K. (2012). DroidBox: Android application sandbox.

[2] UI/Application Exerciser Monkey. http://developer.android.com/tools/help/monkey.html

[3] Choudhary, S. R., Gorla, A., & Orso, A. (2015, November). Automated Test Input Generation for Android: Are We There Yet?(E). In Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on (pp. 429-440). IEEE.

[4] Machiry, A., Tahiliani, R., & Naik, M. (2013, August). Dynodroid: An input generation system for Android apps. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (pp. 224-234). ACM.

[5] Hackner, D. R., & Memon, A. M. (2008, May). Test case generator for GUITAR. In Companion of the 30th international conference on Software engineering (pp. 959-960). ACM.

[6] Yang, W., Xiao, X., Andow, B., Li, S., Xie, T., & Enck, W. (2015, May). Appcontext: Differentiating malicious and benign mobile app behaviors using context. In Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on (Vol. 1, pp. 303-313). IEEE.

Acknowledgements

I would like to thank my GSoC mentors Hugo Gascon, Hanno Lemoine, and other cool guys from the Honeynet Project, I learned a lot from them.

Thank the open source projects AndroidViewClient and Androguard that DroidBot is based on.

Also I would like to thank Google for offering the chance to students to help them get involved in open source projects.