2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems

DroidMate: A Robust and Extensible Test Generator for Android

Konrad Jamrozik · Andreas Zeller
Center for IT-Security, Privacy and Accountability (CISPA), Saarland University
Saarland Informatics Campus, Saarbrücken, Germany
{jamrozik,zeller}@cs.uni-saarland.de

ABSTRACT

DROIDMATE is a fully automated GUI execution generator for Android apps. DROIDMATE *explores* an app, i.e. (a) repeatedly reads at runtime the device GUI and *monitored* calls to Android APIs methods and (b) makes a decision what next GUI action (click, long-click, text entry, etc.) to execute, based on that data and provided *exploration strategy*. The process continues until some *termination criterion* is met. DROIDMATE is (1) *fully automatic:* after it has been set up, the exploration itself does not require human presence; (2) *extensible:* without recompiling DROIDMATE, anybody can run it with their own *exploration strategy, termination criterion,* or a set of *monitored* methods; (3) *robust:* tested on 126 apps being in the top 5 in all Google Play categories except Games, it ran successfully on 123 of them; (4) *easy to set up:* it works on Android devices and emulators out-of-the-box, without root or OS modifications; and (5) *easy to modify:* documented sources, built and tested with continuous integration server, are publicly available.

1. INTRODUCTION

Dynamic analysis is a powerful program analysis technique with applications like defect discovery, stress testing, specification mining and malicious behavior detection. However, dynamic analysis requires *executions* that will thoroughly cover program behavior. Executions can be obtained from tests written by developers. Tests, unfortunately, are expensive to write and keep in sync with the evolving code base. Executions can also be gathered from users during production. Yet, such a collection is not always feasible: Besides potential performance and privacy issues, the program in question might need to be analyzed *before* production begins. One alternative to writing tests or collecting executions during production is therefore to *generate* executions—commonly known as *generating tests.*

MobileSoft'16 May 16-17 2016, Austin, TX, USA

ACM ISBN 978-1-4503-4178-3/16/05.

DOI: http://dx.doi.org/10.1145/2897073.2897716

2. TEST GENERATION FOR ANDROID

For the Android platform, recent years have seen a raise of powerful test generators exercising Android apps. MONKEY [9] is a simple fuzz tester, generating random streams of user events such as clicks, touches, or gestures; although typically used as robustness tester, it has been used to find GUI bugs [5] and security bugs [8]. While MONKEY generates pure random events, the DYNODROID tool [7] focuses on those events handled by an app, getting higher coverage while needing only 1/20 of the events. Given an app, all these tools run fully automatically; no model, app code, or annotation is required. Other recent Android test generators like PUMA [4] or ANDLANTIS [3] achieve high levels of scalability, while BRAHMASTRA [2] is good at covering 3rd party components.

All these testing tools however share one or more flaws that make them hard to use in research and practice. Many existing test generators for Android are not actively maintained anymore and support only legacy Android versions. DYNODROID for example, works only for Android 2.3.5 while version 6 is current. In addition, most tools are very hard to deploy, to modify, and their source code is not available. What is worse, claims of scalability hide the fact that the generators are unable to properly handle great deal of more complex apps. BRAHMASTRA, after discarding the unsupported apps, works on 1,010 apps from the original set of 12,571 apps (8%) [2]. PUMA is compatible with 9,644 apps from a set of 18,962 (51%) [4]. These low percentages result from the tools having to rely on complex Dalvik bytecode manipulation of the tested apps, which is hard for nontrivial apps.

3. DROIDMATE KEY STRENGTHS

DROIDMATE, an Android execution generator demonstrated in this paper, is designed with an explicit goal of being usable in research and practice, and of being able to handle even the most complex apps. DROIDMATE support for Android 6 is under active development, while currently it supports Android 4.4.2.

DROIDMATE is highly robust, working unattended on 123 out of 126 top apps [1] from the Google Play Store. Two of the remaining three were already pre-installed on the device and thus could not be modified as needed, and the last one had more than one main launchable activity, which is not supported, but exceedingly rare case. This level of robustness was achieved by putting significant effort into supporting graceful handling and recovery from various possible app failure scenarios as well as by usage of minimal and precise modification of the tested apps bytecode, using components from the commercial-grade APPGUARD [1] tool.

[1]The tested apps metadata is available at http://www.droidmate.org

DROIDMATE provides facilities to determine tested device GUI structure, to act on its GUI elements and to read the details of the accessed Android SDK APIs methods, including their signatures, parameter values and stack traces. All this information becomes immediately available during the execution generation, and thus can be used to adapt it online.

Furthermore, by design, the logic determining the execution generation process is replaceable without recompiling DROIDMATE itself, as well as logic for determining when generation has to end or which methods are to be monitored. To sum up: *DROIDMATE is designed to be usable by other researchers and to work reliably even on the most complex of apps.*

4. IMPLEMENTATION

DROIDMATE generates tests by *exploring* the *Application under Test* (*AuT*), that is, by interacting at runtime with its GUI elements (called *views* in Android) and reasoning about the *AuT* behavior to influence further GUI interaction.

DROIDMATE takes as input directory containing a set of .apk files to be explored, the *AuT*s. Each of these files is then *inlined*, that is, it undergoes a slight dalvik bytecode modification to enable monitoring of Android SDK APIs methods. Changing the set of monitored methods does not require re-inlining apps, thus any given app needs to be inlined only once.

Next, in sequence, DROIDMATE installs each app on an Android device and then launches the *AuT*'s main activity. DROIDMATE then operates in a loop, as seen on Figure 1, whose centerpiece is the *exploration strategy*.

The exploration strategy is called for the first time with a result of conducting a *reset exploration action*. In general, it takes as input *exploration action run result* which contains information on the displayed GUI views and called *monitored method*s after last *exploration action*. Given this information, it *decides* which next exploration action to conduct. Exploration action can be any of: *click, long-click, press home, press back, reset, terminate*. The actions are then executed on the device, handling and recovering from a plethora of possible failures, up to and including intermittent loss of connection to the device. Exploration continues until some *termination criterion* is met. The data output from the explorations conducted in full DROIDMATE run is persisted as serialized java objects. DROIDMATE provides facilities to extract from it textual reports and data points for plotting various charts. The data is very detailed and sufficient to replay the tests, either manually or automatically.

DROIDMATE comes with DYNODROID-inspired exploration strategy, as described in [6], time-based termination criterion and monitors a set of methods as defined in APPGUARD. Because exploration strategy can be easily exchanged, one could provide a strategy that leverages precomputed static knowledge about the application or manually provided information, like login credentials.

To read the GUI XML structure and to conduct GUI interactions on the device, DROIDMATE leverages *uiautomator*, the official Google GUI automation framework. DROIDMATE keeps alive an *uiautomator* JUnit test on the device, running a TCP server communicating with a TCP client located on a developer machine. This communication scheme is setup before the first explored app is installed.

DROIDMATE is implemented predominately in the Groovy language, with any further development happening in JetBrains' Kotlin. It is built with Gradle. All its documented source code, built and tested with a continuous integration server, as well as all associated documentation and papers can be accessed at

`http://www.droidmate.org/`

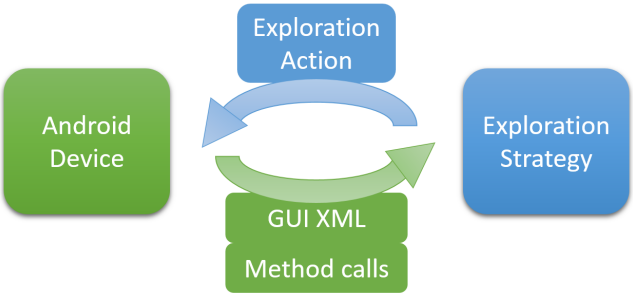

Figure 1: The exploration loop

5. REFERENCES

[1] BACKES, M., GERLING, S., HAMMER, C., MAFFEI, M., AND VON STYP-REKOWSKY, P. AppGuard–fine-grained policy enforcement for untrusted Android applications. In *Data Privacy Management and Autonomous Spontaneous Security*, J. Garcia-Alfaro, G. Lioudakis, N. Cuppens-Boulahia, S. Foley, and W. M. Fitzgerald, Eds., Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 213–231.

[2] BHORASKAR, R., HAN, S., JEON, J., AZIM, T., CHEN, S., JUNG, J., NATH, S., WANG, R., AND WETHERALL, D. Brahmastra: Driving apps to test the security of third-party components. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.* (2014), pp. 1021–1036.

[3] BIERMA, M., GUSTAFSON, E., ERICKSON, J., FRITZ, D., AND CHOE, Y. R. Andlantis: Large-scale Android dynamic analysis. *CoRR abs/1410.7751* (2014).

[4] HAO, S., LIU, B., NATH, S., HALFOND, W. G., AND GOVINDAN, R. PUMA: Programmable UI-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2014), MobiSys '14, ACM, pp. 204–217.

[5] HU, C., AND NEAMTIU, I. Automating GUI testing for Android applications. In *Proceedings of the 6th International Workshop on Automation of Software Test* (New York, NY, USA, 2011), AST '11, ACM, pp. 77–83.

[6] JAMROZIK, K., VON STYP-REKOWSKY, P., AND ZELLER, A. Mining sandboxes. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, TX, USA, 2016), ICSE 2016, ACM, p. to appear.

[7] MACHIRY, A., TAHILIANI, R., AND NAIK, M. Dynodroid: An input generation system for Android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (New York, NY, USA, 2013), ESEC/FSE 2013, ACM, pp. 224–234.

[8] MAHMOOD, R., ESFAHANI, N., KACEM, T., MIRZAEI, N., MALEK, S., AND STAVROU, A. A whitebox approach for automated security testing of Android applications on the cloud. In *Proceedings of the 7th International Workshop on Automation of Software Test* (Piscataway, NJ, USA, 2012), AST '12, IEEE Press, pp. 22–28.

[9] Monkey: UI/Application exerciser. `http://developer.android.com/tools/help/monkey.html`. Retrieved 2015-02-01.