Artifacts for Dynamic Analysis of Android Apps

Haipeng Cai
Washington State University, Pullman, USA
hcai@eecs.wsu.edu

Barbara G. Ryder
Virginia Tech, Blacksburg, USA
ryder@cs.vt.edu

I. Introduction

As Android continues to gain momentum in mobile computing, increasing research has been invested in analyzing Android apps. In particular, due to inherent limitations of static analysis (e.g., in dealing with dynamic language constructs and code obfuscation), dynamic analysis has been recognized as an alternative or complement [1]. However, in contrast to reusable utilities available for static analysis of Android apps (e.g., [2]), such utilities for dynamic analysis of apps are rare. To facilitate research based on dynamic app analysis, we share a set of reusable and extensible artifacts that we have applied for our recent dynamic characterization study of Android apps [3]. We show how these artifacts can be used to reproduce the study. Moreover, we briefly discuss how they can be reused and extended for more applications. The artifact package along with details on setup and usage instructions is available here. The package consists of the analysis code and dataset used in the study, and a VM for demonstration and replication.

II. Analysis Code

While the executables are sufficient for reproducing our study results, we share the source code also so that (1) the toolkit can be *customized* by interested readers to conduct similar characterizations but with varied metrics, and (2) development of different dynamic analysis tools can be facilitated by reusing some of the components in the toolkit. The source directory (*code*) includes six components:

- `dynCG`: a dynamic call graph construction and search tool, which profiles all method calls of an app including those via reflection and in exception-handling constructs.
- `eventTracker`: a profiler of system and user-interface events occurred during an app execution.
- `intentTracker`: a tracer of Intent objects carried by all exercised inter-component communications at runtime.
- `covTracker`: a statement coverage tracker working on the APK (without relying on the source code) of an app.
- `utils`: various utilities for bytecode instrumentation and manipulation, including a bytecode transformer that adds exception-handling constructs to specified methods.
- `reporters`: a set of statistics calculators for computing characterization metrics from an app execution trace.

All these components are based on Soot [2] using its Jimple IR. The first four can work as standalone tools and are extensible, which *demonstrate* how to readily write a dynamic analysis tool using Soot and can be used as *code templates* for that purpose. In particular, previous works that measured statement coverage for Android apps relied on source code of the apps. The *utils* component can be immediately reused for building various Soot-based tools. A build file and all libraries required are included in the artifact package. Also included are scripts for running these tools, scripts for experimental data analysis, and various other helper scripts (used to download, install/uninstall, query, and launch APKs, etc.).

III. Dataset

Study results. The data directory (*data*) includes the raw data of our study along with the results presented in the research paper. The data on metrics in three dimensions (*General/Structure*, *ICC*, and *Security*) is placed in the respective folder. R scripts for producing final results from the raw data in each folder are included accordingly. Each raw data file is explained here, and the purpose of each R script is explained here. A convenience script *produceall.sh* is included under *data* for processing all the raw data at once.

Benchmark suites. Our study used two benchmark suites: a suite of 125 individual apps and a suite of 62 app pairs that actually communicate at runtime as quickly triggered by random inputs from Monkey [4]. The first suite can be readily downloaded from Google Play using our helper scripts. The list of these apps is included (in *data/benchmarks/used-benig-apps-droidfax.txt*).

The second suite is even more worthy of sharing because finding a set of apps with dynamically communicating peers is not trivial. This suite is particularly useful for evaluating an inter-app dynamic analysis for Android. We have not only provided the pairs but also the statistics on the ICCs that linked them at runtime in our study (as detailed in *data/benchmarks/app-pair-statistics.html*).

Characterization metrics. We defined a set of 122 metrics in the three dimensions mentioned above. These metrics have been used for discovering new insights into the behavioral traits of Android apps in our study. Furthermore, they have been utilized for developing advanced malware classifiers as well (based on the behavioral profile, defined by these metrics, of benign apps versus malware) [5]. These metrics (detailed here) can be used by others for understanding app behaviors and reused for future studies and techniques.

References

[1] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, “The evolution of Android malware and Android analysis techniques,” *ACM Computing Surveys*, vol. 49, no. 4, p. 76, 2017.

[2] P. Lam, E. Bodden, O. Lhoták, and L. Hendren, “Soot - a Java bytecode optimization framework,” in *Cetus Users and Compiler Infrastructure Workshop*, 2011, pp. 1–11.

[3] H. Cai and B. Ryder, “Understanding Android application programming and security: A dynamic study,” in *Proceedings of International Conference on Software Maintenance and Evolution*, 2017.

[4] Google, “Android Monkey,” http://developer.android.com/tools/help/monkey.html, 2015.

[5] H. Cai, N. Meng, B. Ryder, and D. Yao, “Droidcat: Unified dynamic detection of Android malware,” Tech. Rep. TR-17-01, January 2017, http://hdl.handle.net/10919/77523.