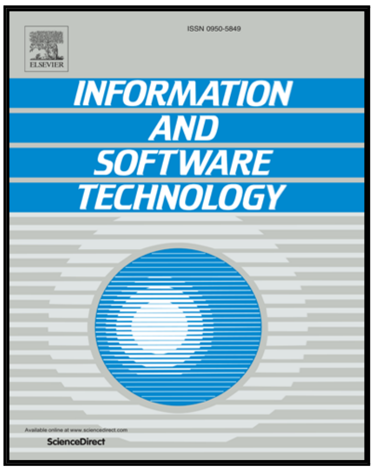

Static analysis of android apps: A systematic literature review

Li Li, Tegawendé F. Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, Yves Le Traon

PII:	S0950-5849(17)30298-7
DOI:	10.1016/j.infsof.2017.04.001
Reference:	INFSOF 5823

To appear in: *Information and Software Technology*

Received date:	2 February 2016
Revised date:	6 March 2017
Accepted date:	4 April 2017

Static Analysis of Android Apps: A Systematic Literature Review

Li Li[a,1], Tegawendé F. Bissyandé[a], Mike Papadakis[a], Siegfried Rasthofer[b], Alexandre Bartel[a,2], Damien Octeau[c], Jacques Klein[a], Yves Le Traon[a]

[a]*Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Luxembourg*
[b]*Fraunhofer SIT, Darmstadt, Germany*
[c]*University of Wisconsin and Pennsylvania State University*

Abstract

Context: Static analysis exploits techniques that parse program source code or bytecode, often traversing program paths to check some program properties. Static analysis approaches have been proposed for different tasks, including for assessing the security of Android apps, detecting app clones, automating test cases generation, or for uncovering non-functional issues related to performance or energy. The literature thus has proposed a large body of works, each of which attempts to tackle one or more of the several challenges that program analysers face when dealing with Android apps.

Objective: We aim to provide a clear view of the state-of-the-art works that statically analyse Android apps, from which we highlight the trends of static analysis approaches, pinpoint where the focus has been put, and enumerate the key aspects where future researches are still needed.

Method: We have performed a systematic literature review (SLR) which involves studying 124 research papers published in software engineering, programming languages and security venues in the last 5 years (January 2011 - December 2015). This review is performed mainly in five dimensions: problems targeted by the approach, fundamental techniques used by authors, static analysis sensitivities considered, android characteristics taken into account and the scale of evaluation performed.

Results: Our in-depth examination has led to several key findings: 1) Static analysis is largely performed to uncover security and privacy issues; 2) The Soot framework and the Jimple intermediate representation are the most adopted basic support tool and format, respectively; 3) Taint analysis remains the most applied technique in research approaches; 4) Most approaches support several analysis sensitivities, but very few approaches consider path-sensitivity; 5) There is no single work that has been proposed to tackle all challenges of static analysis that are related to Android programming; and 6) Only a small portion of state-of-the-art works have made their artefacts publicly available.

Conclusion: The research community is still facing a number of challenges for building approaches that are aware altogether of implicit-Flows, dynamic code loading features, reflective calls, native code and multi-threading, in order to implement sound and highly precise static analyzers.

1. Introduction

Since its first commercial release in September 2008, the Android mobile operating system has witnessed a steady adoption by the manufacturing industry, mobile users, and the software development community. Just a few years later, in 2015, there were over one billion monthly active Android users, meanwhile its official market (Google Play) listed more than 1.5 million apps. This adoption is further realised at the expense of other mobile systems, since Android accounts for 83.1% of the mobile device sales in the third quarter of 2014 [1], driving a momentum which has created a shift in the development community to place Android as a "priority" target platform [2].

Because Android apps now pervade all user activities, ill-designed and malicious apps have become big threats that can lead to damages of varying severity (e.g., app crashes, financial losses with malware sending premium-rate SMS, reputation issues with private data leaks, etc). Data from anti-virus vendors and security experts regularly report on the rise of malware in the Android ecosystem. For example, G DATA has reported that the 560,671 new Android malware samples collected in the second quarter of 2015 revealed a 27% increase, compared to the malware distributed in the first quarter of the same year [3].

To deal with the aforementioned threats, the research community has investigated various aspects of Android development, and proposed a wide range of program analyses to identify syntactical errors and semantic bugs [4, 5], to discover sensitive data leaks [6, 7], to uncover vulnerabilities [8, 9], etc. In most cases, these analyses are performed statically, i.e., without actually running the Android app code, in order not only to ensure scalability when targeting thousands of apps in stores, but also to guarantee a traversal of all possible execution paths. Unfortunately, static analysis of Android programs is not a trivial endeavour since one must account for several specific features of Android, to ensure both soundness and completeness of the analysis. Common barriers to the design and implemen-

Email address: `li.li@uni.lu` (Li Li)
[1]Corresponding author.
[2]the author was employed at the Technical University of Darmstadt, Germany, when he first worked on this paper