Poster: R-DROID: In-depth Application Vetting for Android with Path-sensitive Value Analysis

Michael Backes*, Sven Bugiel†, Erik Derr†, Sebastian Gerling† and Christian Hammer†
*CISPA, Saarland University, MPI-SWS
backes@cs.uni-saarland.de
†CISPA, Saarland University
{bugiel,derr,gerling,hammer}@cs.uni-saarland.de

***Abstract*—This paper presents a novel approach for value-sensitive security analysis and in-depth vetting of Android applications. Our approach employs a slicing-based analysis to generate data-dependent statements for arbitrary points of interest in an application. While existing app vetting approaches already assess the results at this point, our analysis proceeds by further optimizing the slice statically. To this end, we introduce a *novel path-sensitive value analysis* that reduces the slice's size and the false-positives rate by leveraging optimization techniques from partial and symbolic evaluation, domain knowledge and copy propagation. The resulting optimized slices are finally input into three security modules: a data-leak detection module, a user-input propagation analysis, and a slice-rendering module, to structure groups of instructions within the slice. We consolidate all aforementioned features into a tool called R-DROID that we make publicly available. Our evaluation confirms that R-DROID is more precise and less prone to false positives than related tools. In a large-scale data leak analysis on a set of 22,700 Android apps from Google Play, R-DROID managed to identify a significantly larger set of potential privacy-violating information flows than previous work, including 2,157 sensitive flows of password-flagged UI widgets in 256 distinct apps.**

I. INTRODUCTION

Modern smartphone apps offer an abundance of features that request users to grant access to their highly sensitive, personal data. The wide proliferation of these apps has made them a prime target for malware developers, and the variety of reported privacy incidents has fueled the legitimate privacy concerns of end users that their sensitive data is stealthily collected, monetized, and disseminated [1], [2], [3], [4]. Centralized app markets have responded by trying to identify malicious and overly curious applications even before these apps are deployed on a user's smartphone. To this end, they strive for comprehensive *application vetting* to understand app internals and to thereby identify abnormal app behaviours.

Static analysis of apps is widely accepted as a well-suited, automated concept for application security vetting on a large scale. In the context of Android, prior work has already successfully identified particular security and privacy problems like (user-intended) privacy leak detection [5], [6], [7], [8], [9], component hijacking vulnerability detection [10] and misuses of (framework) features such as the crypto API [11] or dynamic code loading [12].

All these approaches share the goal to precisely capture which data flows into sensitive method calls. However, all these approaches are (fully or partially) agnostic to the concrete data values that arise while executing the app, e.g., they fail to resolve the concrete value for a receiver number in a text-message app (and hence cannot tell apart a legitimate app from premium SMS messages). As a consequence, the vast amount of prior work that disregards concrete data values is not capable of coping with more elaborate cases in which the assessment depends on the exact runtime values (strings or primitive values). The few existing approaches that do not consider this problem as out-of-scope offer coarse-grained approximations for runtime strings [13], [5], [14], [15] that result in highly conservative approximations (*"could be any string"*, or *"any combination of these strings"*) in more evolved cases. As a result, they face many *false positives* (i.e. false alarms) and still fail to assess certain cases completely.

Moreover, a widely disregarded aspect of prior work is that real-life application vetting always requires a significant amount of manual investigation to cope with unclear cases. However, manually investigating the outputs of existing approaches even for small cases (typically a huge list of data-dependent statements and an involved kind of formal security assessments) typically constitutes an intricate task, since these tools either directly work with the app's bytecode [5], [12] or transform it into an intermediate representation [16], [9], [10], [6] that is even less amenable to manual review than the original source code. As a consequence, the real-life vetting process is significantly impeded, and the lack of assistance to facilitate the subsequent manual reviewing task becomes evident.

Our contributions. To address the aforementioned challenges, we present a novel approach for value-sensitive security analysis and in-depth vetting of Android applications. Our approach relies on a slicing-based analysis to generate data-dependent statements for arbitrary points of interest in an application. While existing app vetting approaches already assess the results at this point, our analysis proceeds by further optimizing the slice statically. To this end, we leverage optimization techniques from partial and symbolic evaluation, domain knowledge and copy propagation to introduce a *novel path-sensitive value analysis* that reduces the slice's size and the false-positives rate while at the same time string parameters and primitive values are precisely retargeted. The resulting optimized slices can be processed further by distinguished *security modules* that provide further insights into the internal structure of the application and that facilitate the task of manual app reviewing. Technically, we make the following four contributions:

1. Novel path-sensitive value analysis. We start by leveraging a system dependency graph (SDG) that distinguishes different

objects of the same type (object sensitivity), fields of the same object (field sensitivity), calling contexts of invoked methods (context sensitivity), and definitions of the same local variable on different paths through a method (flow sensitivity.) On top of this slicing-based dependency analysis, we propose a novel path-sensitive value analysis that accurately captures information flow on each path of the program, and that tracks for each such path which data can flow to a sink; in particular, it can cope with strings and primitive values that are passed as parameters to security-relevant functions. By carefully yet soundly abstracting from the exact number of loop iterations an app performs, the analysis is only required to investigate a finite number of these paths. Our value analysis constitutes a multi-step optimization algorithm that leverages techniques from partial and symbolic evaluation, domain knowledge, and copy propagation. This optimization gives rise to three main advantages: First, it reduces the number of false positives by eliminating non-relevant statements that the data-dependency analysis failed to resolve. Second, more evolved security problems such as premium number assessment can be evaluated automatically. Third, it facilitates to understand the app's information flow in a subsequent manual reviewing, since it splits the result into slices per execution path and thereby efficiently reduces the overall output size (26% less instructions on average for our test set; up to 76% less in extreme cases).

2. Extended Android lifecycle modeling. We extend the control-flow of the SDG that underlies our approach's reasoning with a comprehensive Android lifecycle modeling approach that comprises previously disregarded or inaccurately captured features such as Fragments and asynchronous communication via AsyncTasks. To this end, we leverage specifications of the Android lifecycle in order to identify entry/callback methods, and thereby prepare the ground for an accurate subsequent static analysis. Our evaluation on Google Play, see below, shows that out of 22,700 apps we analyzed, 7,497 (33%) make use of at least one Fragment and 14,244 apps (62.7%) include at least one AsyncTask.

3. Complementary Analysis via Security Modules. Our approach supports the integration of further security modules to extend and amplify the analysis of app security and privacy. Security modules directly benefit from the underlying value analysis via access to the optimized path-sensitive slices. Modules can define their own sources/sinks and gain access to our approach's intermediate analysis results. We define three such security modules – data leakage detection, user input propagation, and slice rendering for manual code review – that we used for our large-scale evaluation on Google Play, see below.

4. R-DROID *and Large-scale Evaluation on Google Play.* We consolidate all aforementioned features into a tool called R-DROID that we make publicly available. The evaluation of R-DROID on the widely accepted, open-source testsuite DroidBench excels over FlowDroid [16], AmanDroid [9], and Fortify SCA with nearly optimal results: 97% precision (one false alarm) and no missed violation (with previous results achieving at most 87% precision). In a large-scale evaluation of 22,700 apps from Google Play, R-DROID managed to identify a significantly larger set of potential privacy-violating information flows than previous work, including 2,157 sensitive flows of password-flagged UI widgets in 256 distinct apps.

REFERENCES

[1] M. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *Proc. 5th ACM conference on Security and Privacy in Wireless and Mobile Networks (WISEC '12)*. ACM, 2012.

[2] "WhatsApp took all my contacts and sent to their servers without asking me - BlackBerry Forums at CrackBerry.com." http://forums.crackberry.com/blackberry-apps-f35/whatsapp-took-all-my-contacts-sent-their-servers-without-asking-me-649363/.

[3] F-Secure Labs, "Mobile Threat Report Q1 2014," http://www.f-secure.com/static/doc/labs_global/Research/Mobile_Threat_Report_Q1_2014.pdf, 2014.

[4] McAfee Labs, "McAfee mobile security report: Who's is watching you?" http://www.mcafee.com/us/resources/reports/rp-mobile-security-consumer-trends.pdf, Februray 2014.

[5] J. Hoffmann, M. Ussath, T. Holz, and M. Spreitzenbarth, "Slicing droids: Program slicing for smali code," in *Proc. of the 28th Annual ACM Symposium on Applied Computing (SAC '13)*. ACM, 2013.

[6] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale," in *Proc. 5th international conference on Trust and Trustworthy Computing (TRUST '12)*. Springer-Verlag, 2012.

[7] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "Appintent: analyzing sensitive data transmission in Android for privacy leakage detection," in *Proc. 20th ACM Conference on Computer and Communication Security (CCS '13)*. ACM, 2013.

[8] Z. Yang and M. Yang, "Leakminer: Detect information leakage on Android with static taint analysis," in *Proc. 2012 Third World Congress on Software Engineering (WCSE '12)*. IEEE Computer Society, 2012.

[9] F. Wei, S. Roy, X. Ou, and Robby, "Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps," in *Proc. 21th ACM Conference on Computer and Communication Security (CCS '14)*, 2014.

[10] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: Statically vetting android apps for component hijacking vulnerabilities," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*. New York, NY, USA: ACM, 2012, pp. 229–240.

[11] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 73–84. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516693

[12] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, and G. Vigna, "Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications," in *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, 2014.

[13] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. Le Traon, "Effective inter-component communication mapping in Android with Epicc: An essential step towards holistic security analysis," in *Proc. 22Nd USENIX Conference on Security (SEC '13)*. USENIX Association, 2013.

[14] A. Chaudhuri, A. Fuchs, and J. Foster, "SCanDroid: Automated security certification of Android applications," University of Maryland, Tech. Rep. CS-TR-4991, 2009. [Online]. Available: http://www.cs.umd.edu/~avik/papers/scandroidascaa.pdf

[15] E. Chin, A. Porter Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in Android," in *Proc. 9th International Conference on Mobile Systems, Applications, and Services (MobiSys '11)*. ACM, 2011.

[16] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," in *Proc. ACM SIGPLAN 2014 Conference on Programming Language Design and Implementation (PLDI 2014)*, 2014.