Poster: On Demystifying the Android Application Framework: Re-Visiting Android Permission Specification Analysis

Michael Backes*, Sven Bugiel†,
Erik Derr†, Sebastian Weisgerber†
*CISPA, Saarland University, MPI-SWS
†CISPA, Saarland University
{backes,bugiel,derr,weisgerber}@cs.uni-saarland.de

Patrick McDaniel‡, Damien Octeau§
‡Pennsylvania State University
mcdaniel@cse.psu.edu
§Pennsylvania State University & University of Wisconsin
octeau@cs.wisc.edu

***Abstract*—In contrast to the Android application layer, Android's application framework's internals and their influence on the platform security and user privacy are still largely a black box for us. In this paper, we establish a static runtime model of the application framework in order to study its internals and provide the first high-level classification of the framework's protected resources. We thereby uncover design patterns that differ highly from the runtime model at the application layer. We demonstrate the benefits of our insights for security-focused analysis of the framework by re-visiting the important use-case of mapping Android permissions to framework/SDK API methods. We, in particular, present a novel mapping based on our findings that significantly improves on prior results in this area that were established based on insufficient knowledge about the framework's internals. Moreover, we introduce the concept of permission locality to show that although framework services follow the principle of separation of duty, the accompanying permission checks to guard sensitive operations violate it.**

I. Introduction

Android's application framework—i.e., the middleware code that implements the bulk of the Android SDK on top of which Android apps are developed—is responsible for the enforcement of Android's permission-based privilege model and as such is also a popular subject of recent research on security extensions to the Android OS. These extensions provide various security enhancements to Android's security, ranging from improving protection of the user's privacy [1], [2], to establishing domain isolation [3], [4], to enabling extensible access control [5], [6].

Android's permission model and its security extensions are currently designed and implemented as best-effort approaches. As such they have raised questions about the efficacy, consistency, or completeness [7] of the policy enforcement. Past research has shown that even the best-efforts of experienced researchers and developers working in this environment introduce potentially exploitable errors [8], [9], [10], [11]. In light of the framework size (i.e., millions of lines of code) and based on past experience [8], [9], [12], [11], [13], static analysis promises to be a suitable and effective approach to (help to) answer those questions and hence to demystify the application framework from a security perspective. Unfortunately, on Android, the technical peculiarities of the framework impinging on the analysis of the same have not been investigated enough. As a consequence, past attempts on analyzing the framework had to resort to simple static analysis techniques [14]—which we will show in this paper as being insufficient for precise results—or resort to heuristics [11].

In order to improve on this situation and to raise efficiency of static analysis of the Android application framework, one is confronted with open questions on *how to enable more precise static analysis of the framework's codebase*: where to start the analysis (i.e., what is the publicly exposed functionality)? Where to end the analysis (i.e., what are the data and control flow sinks)? Are there particular design patterns of the framework runtime model that impede or prevent a static analysis? For the Android application layer, those questions have been addressed in a large body of literature. Thanks to those works, the community has a solid understanding of the sinks and sources of security- and privacy-critical flows within apps (e.g., well-known Android SDK methods) and a dedicated line of work further addressed various challenges that the Android *application* runtime model poses for precise analysis (e.g., inter-component communication [15], [16], [17], [18] or modelling the Android app life-cycle[19], [20]). Together those results form a strong foundation on which effective security- and privacy-oriented analysis is built upon. In contrast to the app layer, for the application framework we have an intuitive understanding of what constitutes its entry points, but no in-depth technical knowledge has been established on the runtime model, and almost no insights exist on what forms the security and privacy relevant targets of those flows (i.e., what technically forms the sinks or "protected resources").

A. Our Contributions.

This paper contributes to the demystification of the application framework from a security perspective by addressing technical questions of the underlying problem on *how* to statically analyze the framework's code base. Similar to the development of application layer analyses, we envision that our results contribute some of the first results to a growing knowledge base that helps future analyses to gain a deeper understanding of the application framework and its security challenges.

How to statically analyze the application framework. We present a systematic top-down approach, starting at the framework's entry points, that establishes knowledge and solutions about analyzing the control and data flows within the framework and that makes a first technical classification of the security and privacy relevant targets (or resources) of those flows. The task of establishing a precise static runtime model of the framework was impeded by the absence of any prior knowledge about framework internals beyond black-box observations at the framework's documented API and

manual analysis of code fragments. Hence we generate this model from scratch by leveraging existing results on statically analyzing Android's application layer at the framework layer. The major conceptual problem was that the design patterns of the framework strongly differ from the patterns that had been previously encountered and studied at the application layer. Consequently we devised a static analysis approach that systematically encompasses all framework peculiarities while maintaining a reasonable runtime. As result of this overall process, we have established a dedicated knowledge base that subsequent analyses involving the application framework can be soundly based upon.

AXPLORER *tool and evaluation.* Unifying the lessons learned above, we have built an Android application framework analysis tool, called AXPLORER. We evaluate AXPLORER on four different Android versions—v4.1.1 (API level 16), v4.2.2 (17), v4.4.4 (19), and v5.1 (22)—validate our new insights and demonstrate how specialized framework analyses, such as message-based IPC analysis and framework component interconnection analysis, can be used to speed up subsequent analysis runs (e.g. security analyses) by 75% without having to sacrifice precision. As additional benefit the resulting output can be used by independent work *as is* to create a precise static runtime model of the framework without the need to re-implement the complex IPC analysis.

Android permission analysis. Finally, to demonstrate the benefits of our insights for security analysis of the framework, we conduct an Android permission analysis. In particular, we re-visit the challenge of creating a permission map for the framework/SDK API. In the past, this problem has been tackled [21], [14] without our new insights in the peculiarities of the framework runtime model, and our re-evaluation of the framework permission map reveals discrepancies that call the validity of prior results into question. Using AXPLORER, we create a new permission map that improves upon related work in terms of precision. Moreover, we introduce a new aspect of permission analysis, permission locality, by investigating which framework components enforce a particular permission. We found permissions that are checked in up to 10 distinct and not necessarily closely related components. This indicates a violation of the separation of duty principle and can impede a comprehensive understanding of the permission enforcement.

REFERENCES

[1] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending Android permission model and enforcement with user-defined runtime constraints," in *Proc. 5th ACM Symposium on Information, Computer and Communication Security (ASIACCS '10)*. ACM, 2010.

[2] Y. Zhou, X. Zhang, X. Jiang, and V. Freeh, "Taming information-stealing smartphone applications (on Android)," in *Proc. 4th International Conference on Trust and Trustworthy Computing (TRUST '11)*. Springer-Verlag, 2011.

[3] M. Ongtang, S. E. McLaughlin, W. Enck, and P. McDaniel, "Semantically rich application-centric security in Android," in *Proc. 25th Annual Computer Security Applications Conference (ACSAC '09)*. ACM, 2009.

[4] S. Bugiel, S. Heuser, and A.-R. Sadeghi, "Flexible and fine-grained mandatory access control on Android for diverse security and privacy policies," in *Proc. 22nd USENIX Security Symposium (SEC '13)*. USENIX Association, 2013.

[5] S. Heuser, A. Nadkarni, W. Enck, and A.-R. Sadeghi, "Asm: A programmable interface for extending android security," in *Proc. 23rd USENIX Security Symposium (SEC '14)*. USENIX, 2014.

[6] M. Backes, S. Bugiel, S. Gerling, and P. von Styp-Rekowsky, "Android Security Framework: Extensible multi-layered access control on Android," in *Proc. 30th Annual Computer Security Applications Conference (ACSAC '14)*. ACM, 2014.

[7] J. P. Anderson, "Computer security technology planning study, volume ii," Deputy for Command and Management Systems, HQ Electronics Systems Division (AFSC), L. G. Hanscom Field, Tech. Rep. ESD-TR-73-51, Oct. 1972.

[8] A. Edwards, T. Jaeger, and X. Zhang, "Runtime verification of authorization hook placement for the Linux security modules framework," in *Proc. 9th ACM Conference on Computer and Communication Security (CCS '02)*. ACM, 2002.

[9] X. Zhang, A. Edwards, and T. Jaeger, "Using cqual for static analysis of authorization hook placement," in *Proc. 11th USENIX Security Symposium (SEC' 02)*. USENIX, 2002.

[10] D. Song, J. Zhao, M. G. Burke, D. Sbirlea, D. Wallach, and V. Sarkar, "Finding tizen security bugs through whole-system static analysis," *CoRR*, vol. abs/1504.05967, 2015. [Online]. Available: http://arxiv.org/abs/1504.05967

[11] Y. Shao, J. Ott, Q. A. Chen, Z. Qian, and Z. M. Mao, "Kratos: Discovering inconsistent security policy enforcement in the android framework," in *Proc. 23rd Annual Network and Distributed System Security Symposium (NDSS '16)*. ISOC, 2016.

[12] V. Ganapathy, T. Jaeger, and S. Jha, "Automatic placement of authorization hooks in the Linux Security Modules framework," in *Proc. 12th ACM Conference on Computer and Communication Security (CCS '05)*. ACM, 2005.

[13] L. Tan, X. Zhang, X. Ma, W. Xiong, and Y. Zhou, "Autoises: Automatically inferring security specifications and detecting violations," in *Proc. 17th USENIX Security Symposium (SEC '08)*. USENIX, 2008.

[14] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: Analyzing the android permission specification," in *Proc. 19th ACM Conference on Computer and Communication Security (CCS '12)*. ACM, 2012.

[15] 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.

[16] 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)*. ACM, 2014.

[17] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. Mcdaniel, "IccTA: Detecting Inter-Component Privacy Leaks in Android Apps," in *Proc. 37th International Conference on Software Engineering (ICSE '15)*, 2015.

[18] D. Octeau, D. Luchaup, M. Dering, S. Jha, and P. McDaniel, "Composite Constant Propagation: Application to Android Inter-Component Communication Analysis," in *Proc. 37th International Conference on Software Engineering (ICSE '15)*, 2015.

[19] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities," in *Proc. 19th ACM Conference on Computer and Communication Security (CCS '12)*. ACM, 2012.

[20] 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.

[21] A. Porter Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proc. 18th ACM Conference on Computer and Communication Security (CCS '11)*. ACM, 2011.