Poster: Software Security for Mobile Devices

Steven Arzt[¶], Alexandre Bartel[¶], Richard Gay[†], Steffen Lortz[†], Enrico Lovat[‡], Heiko Mantel[†], Martin Mohr[**], Benedikt Nordhoff[§], Matthias Perner[†], Siegfried Rasthofer[¶], David Schneider[†], Gregor Snelting[**], Artem Starostin[†], Alexandra Weber[†]
[¶]TU Darmstadt, Email: ⟨firstname.lastname⟩@ec-spride.de
[†]TU Darmstadt, Email: ⟨lastname⟩@mais.informatik.tu-darmstadt.de
[‡]TU Munich, Email: enrico.lovat@in.tum.de
[§]WWU Münster, Email: b.n@wwu.de
[**]Karlsruhe Institute of Technology, Email: ⟨firstname.lastname⟩@kit.edu

I. Motivation

Android is the most commonly used mobile operating system, with a market share of 76% at the end of 2014.[1] At the same time, security violations by Android applications are often reported. These commonly take the form of the application revealing the user's sensitive information (as in the example of a flashlight app that reveals the user's device ID and location[2]) or behaving in a way that is unexpected and harmful to the user (e.g., by sending costly SMS without his knowledge[3]).

The Android ecosystem offers a number of security mechanisms, such as the sandboxing of applications and a permission system restricting access to critical resources. Moreover, applications available on Google Play are scanned to detect malicious behavior. However, as the prevalence of security problems on Android devices indicates, these mechanisms are not sufficient for enabling security-aware end users of Android devices to reliably enforce their personal security requirements.

II. A Certifying App Store

To address this problem, we are developing the *RS*^{3} *Certifying App Store*. It integrates techniques from different research areas into a one-stop security solution for end users of Android devices. Specifically, we employ *static information flow analysis* to ensure confidentiality of sensitive user data before installation and *runtime enforcement* to prohibit unwanted functionality of applications. Users can choose from a set of predefined policies or specify their own security requirements. Both the enforced security policies and the results of analyzing applications are presented to the user in an intuitive fashion. In this manner, the security guarantees obtained by applying the security techniques integrated in the app store are made explicit to the user.

The main novelty of our approach is that we focus on the end user and make state-of-the-art security technology available to him. A second contribution is the integration of technologies tackling orthogonal aspects of security into a single tool.

[1] http://www.idc.com/prodserv/smartphone-os-market-share.jsp
[2] http://www.digitaltrends.com/mobile/brightest-flashlight-ftc-punishment/
[3] https://blog.malwarebytes.org/mobile-2/2013/09/uncovering-an-android-botnet-involved-in-sms-fraud/

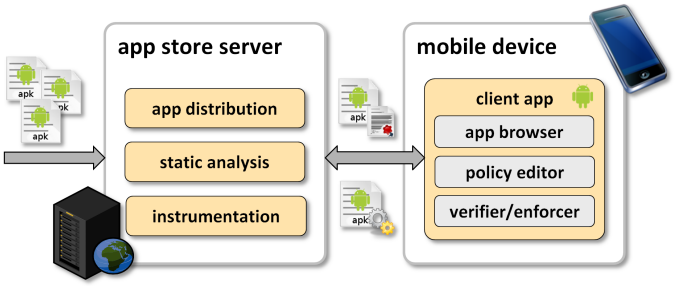

Fig. 1. Architecture of the RS^3 Certifying App Store

A. Architecture

The RS^3 Certifying App Store has a client-server architecture, which is depicted in Figure 1. Similar to the servers of existing app stores, the *server* of the RS^3 Certifying App Store stores and distributes Android apps uploaded as Android application package (`apk`) files. In addition, our server hosts tools for the static information flow analysis and the instrumentation of Android apps with mechanisms for runtime enforcement, and offers their functionality as a service to its clients.

The *client* of the RS^3 Certifying App Store is an app running on Android mobile devices. It provides an app browser for selecting apps to install, a policy editor for specifying security policies for the selected apps, and mechanisms for ensuring that installed apps satisfy the specified security policies.

To ensure the security of an app, the client requests the static analysis or the instrumentation of an app w.r.t. a security policy from the server. The client then receives the app together with the analysis result or with mechanisms for runtime enforcement instrumented into the app's bytecode. Finally, the client verifies the analysis results and controls the runtime enforcement of policies in instrumented apps.

B. Prototype

We have developed prototypes of both the client and the server of the RS^3 Certifying App Store.

Installing the client app requires no changes to the operating system. Users can define their information flow policies by using a policy editor allowing them to select categories of sensitive information that they do not want to be revealed, e.g., "Contact Data" or "Location Data". The client graphically displays the possible flows of sensitive information in an app (cf. Figure 2). For specifying policies on the behavior of applications, the client offers a policy editor that allows users

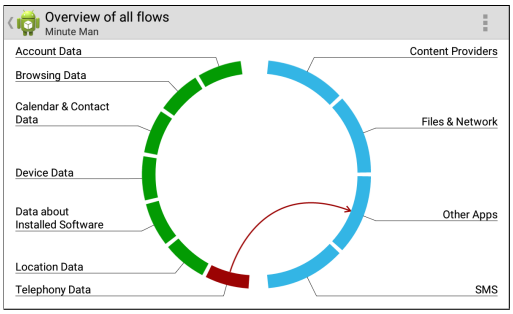

Fig. 2. Screenshot of the visualization of information flows in an app

to instantiate parametric policy templates to their particular security requirements. Together, the policy editors and the visualization of results make the security guarantees provided by the app store easily understandable to the user.

The server-side part of our app store can be set up on any conventional web server, since it is implemented using only commonly available technology (i.e., PHP, Java, and MySQL). The results of statically analyzing apps are cached in order to increase responsiveness to the end user. The instrumentation of apps is not yet performed on the server. Rather, apps are instrumented before being uploaded.

III. Security Modules

The RS^3 Certifying App Store currently integrates two security modules. One provides static analyses for checking whether apps adhere to information flow policies, the other provides a runtime enforcement mechanism for preventing unwanted behavior of apps.

A. Static Analysis

The RS^3 Certifying App Store employs the RSCP Analyzer [1], a type-based static analysis, to certify that apps comply with the security policies defined by users before installation. More specifically, the analysis certifies that the app does not leak information declared as confidential by the user to untrusted output channels, e.g., to the Internet. The certificate of the security of an app with respect to an information flow policy is computed by a security type inference.

Because the certification of apps can be resource-intensive and mobile devices are typically limited in computational power, the analysis employs the principle of proof-carrying code. This means that security certificates are computed on the server of the app store. The resulting certificates are then verified on the user's device by type-checking the app against the certificate using the rules of the security type system. This is much less complex than computing the certificate and, hence, allows most of the workload of the analysis to be shifted from the mobile device to the more powerful server. Due to the verification of certificates on the mobile device, the user does not have to trust the server. The security type system has been proven sound with respect to a security property based on a formal semantics of Dalvik bytecode. Hence, users can have a high degree of confidence in the result of the analysis.

We are currently also integrating Joana, another static information flow analysis tool, which was recently adapted for the analysis of Android applications [2]. Joana is based on program dependency graphs (PDGs) and utilizes sophisticated program analyses (e.g., alias analysis) as the basis for a flow-, context-, field-, and object-sensitive information flow analysis that aims to be highly precise.

By integrating two static analysis tools with complementary advantages, we enable users to choose the tool best suited to their requirements. The integration also enables us to compare the two approaches.

B. Dynamic Enforcement

The RS^3 Certifying App Store furthermore integrates DroidForce [3] to enforce user-defined policies on the behavior of apps. Such policies may depend on information available only at runtime, e.g., when limiting the number of SMS that may be sent per hour or when disabling Internet access at night. To enforce such policies, DroidForce instruments the bytecode of applications by injecting policy enforcement points (PEPs) for monitoring and controlling the occurence of security-critical events.

At runtime, the PEPs observe the occurence of such events and query the policy decision point (PDP), which is part of the client of our store, for whether the event shall be allowed to occur. The PDP decides this based on whether the execution of an event would violate the policy defined by the user. The decision of the PDP is then enforced by the PEPs. Because the PEPs in all instrumented applications communicate with the same instance of the PDP, system-wide security policies can be enforced.

IV. Conclusion and Future Work

With the RS^3 Certifying App Store, we contribute the first tool that allows end users of Android mobile devices to specify and reliably enforce their personal security requirements and to obtain explicit security guarantees. This is achieved by applying state-of-the-art security technologies from different areas of research to address two different kinds of security problems that are prevalent in Android applications.

Currently, only one of the security modules can be applied to each app in the store due to technical limitations. In the future, we also want to support the combined application of the tools to enforce different aspects of security for the same app. Furthermore, the scope of the RS^3 Certifying App Store could be extended to further security problem domains by integrating additional complementary tools.

Acknowledgment

This work was funded by the DFG under the projects COOR (MA 3326/3-1/2/3), IFC4MC (MU 1508/2, Sn 11/12-1), INTERFLOW (BO 2528/5-1), RSCP (MA 3326/4-1/2/3), and SADAN (PR 1266/1-3) in the priority program RS^3 (Reliably Secure Software Systems, SPP 1496).

References

[1] S. Lortz, H. Mantel, A. Starostin, T. Bähr, D. Schneider, and A. Weber, "Cassandra: Towards a Certifying App Store for Android," in *Proceedings of the 4th ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2014.

[2] M. Mohr, J. Graf, and M. Hecker, "JoDroid: Adding Android Support to a Static Information Flow Control Tool," in *Proceedings of the 8th Working Conference on Programming Languages*, 2015, to appear.

[3] S. Rasthofer, S. Arzt, E. Lovat, and E. Bodden, "DroidForce: Enforcing Complex, Data-Centric, System-Wide Policies in Android," in *Proceedings of the 9th International Conference on Availability, Reliability and Security*, 2014.