Cassandra: Towards a Certifying App Store

Technical Report TUD-CS-2014-0117

May 2014

Steffen Lortz,
Heiko Mantel,
Artem Starostin,
Timo Bähr,
David Schneider

TECHNISCHE
UNIVERSITÄT
DARMSTADT

MAIS
Modeling and Analysis
of Information Systems

Cassandra: Towards a Certifying App Store

Steffen Lortz, Heiko Mantel, Artem Starostin, Timo Bähr, and David Schneider

Modeling and Analysis of Information Systems (MAIS)
Computer Science Department, TU Darmstadt, Germany
lastname@mais.informatik.tu-darmstadt.de

Abstract Modern mobile devices store an abundance of information. However, users do not yet obtain satisfactory support for controlling what applications do with their personal data. In this article, we propose Cassandra, a tool that allows users to verify Android apps against their security needs before installation. Cassandra implements the core functionality of a conventional app store, augmented with the possibility to verify user-defined security policies. Cassandra's security analysis is fully automatically performed on a server. The analysis results are communicated to a mobile device using the proof-carrying code paradigm.

Keywords: software security, formal methods, information-flow control, mobility, program analysis, proof-carrying code, security engineering

1 Introduction and Related Work

Our tool, Cassandra, enables a user to check Android apps against her personal security needs before installation. The primary goal of Cassandra is to ensure that no private data and no other secrets are leaked by running an app. The silent leakage of a user's personal data is not just a technical possibility, but a serious threat in reality. For instance, [TK10, EOMC11] show that many apps forward private data to the Internet. Cassandra aims at countering this threat.

The overall functionality of Cassandra resembles the core functionality of app stores like, e.g., F-Droid [Lim10]. In particular, a user can browse the available apps on a server and can select apps for installation on her mobile device. The additional functionality offered by Cassandra is that a user can specify security policies and can verify apps against these policies before installation.

More specifically, Cassandra checks a given security policy against all flows of information that running a given app can possibly cause. The information flow analysis implemented in Cassandra was proven sound wrt. a formally defined, noninterference-like security property [LMSW14]. Thus, if an app passes Cassandra's security analysis, then it is guaranteed to comply with the user's security policy in the sense that the app's output to public sinks is independent from secrets. If the analysis fails then Cassandra provides the user with detailed information about which violations of her security policy are possible such that the user can make an informed decision whether to install the app or not.

Android already provides security mechanisms. Before installing an app, a user needs to authorize the accesses declared in the app's manifest. Android's permission system ensures that apps can only access protected resources if this is declared in their manifest. The permission system provides access control, but it does not control how information is propagated after an access. Android uses cryptographic signatures to ensure the authenticity and integrity of apps. These signatures are also used to constrain sharing of resources between apps, i.e., across their sandboxes. Android's built-in malware detection service verifies newly installed apps and warns users about potential malware [Pro12]. All these mechanisms are complementary to Cassandra's information flow analysis.

Starting with ScanDroid [FCF09], a number of security analysis tools has been proposed for Android in recent years. AndroidLeaks [GCEC12], CHEX [LLW+12], LeakMiner [YY12], Scandal [KYYS12], TrustDroid [ZCO12], Epicc [OMJ+13], and FlowDroid [FAR+13], support a static security analysis of Android apps at the level of Dalvik bytecode. All of these tools perform a data flow analysis and do not yet take implicit information flows properly into account. TaintDroid [EGC+10] also supports a data flow analysis, but requires modifications to the run-time environment. In contrast, Cassandra does not require any changes to the run-time environment or to apps. To our knowledge, Cassandra is the first information flow analysis tool for Android apps with a soundness result.

In this article, we focus on illustrating the use of Cassandra and on describing its architecture and implementation. The theory underlying Cassandra's information flow analysis is described in a companion paper [LMSW14].

The architecture of Cassandra builds on both the client-server paradigm and the proof-carrying code principle [Nec97]. The client of Cassandra is an Android app itself that runs on off-the-shelf Android devices. Using the client, a user can browse available apps, specify security policies, initiate an information flow analysis, and examine the analysis results. The server of Cassandra is implemented in Java and PHP. It runs on a web server. Cassandra's server incorporates a database of available apps that can be browsed by a client. The server also performs the information flow analysis for a given user-defined security policy. The results of a successful security analysis are communicated to the client in a security certificate that carries enough information to replay the security analysis on the client. This is where proof-carrying code (PCC) is used.

We will make Cassandra's source code available under the MIT License.

2 Using Cassandra

The user of a mobile device can employ Cassandra to ensure that she only installs apps on the device that respect her security policies. Installing apps using Cassandra is similar to installing apps from regular app stores like, e.g., F-Droid [Lim10] but involves additional steps to analyze the security of apps.

We illustrate this at the example of installing the app Minute Man, an app for optimizing call costs. This app automatically interrupts phone calls after 59 seconds to avoid the charge for a second minute. We implemented Minute Man ourselves inspired by similar existing Android apps like, e.g., [MX].

To install the Minute Man app, the user invokes Cassandra's client app and then selects Minute Man in the *app browser*. The left hand side of Figure 1 shows a screen-shot of the app browser's interface to the user (where black and white are inverted). Afterwards, the user specifies which data is private and which sinks are untrusted by marking information sources and sinks in the *policy editor*. In our example, the user wants an overview of all information flows caused by running Minute Man and, hence, she marks all displayed information sources and sinks as private and public, respectively (see center of Figure 1). The user's choice of private sources and public sinks constitutes a *security policy*.

To initiate the information flow analysis, the user presses a button. The analysis itself is not performed by the client but on a server (see Section 3). After the security analysis is complete, the user can inspect the analysis result using the *reporter*. The reporter either confirms that the security policy is satisfied by the given app or provides information about all violations. In our example, the reporter warns about one violation, namely that telephony data might be leaked to other apps (see right hand side of Figure 1). Based on the result of the security analysis, the user can make an informed decision whether to install the app or not. For installing apps, Cassandra uses the standard installer of Android.

A video of the Minute Man analysis with Cassandra is available at [Sta].

3 Architecture

The architecture of Cassandra builds on the client-server paradigm, where both the client and the server, again, have a modular architecture (see Figure 2).

Cassandra's client and server perform three interactions during the installation of an app. Firstly, the client retrieves the list of available apps together with information about their sources and sinks from the server (1). Secondly, the client informs the server which app has been chosen by the user and sends the user-defined security policy to the server (2). Thirdly, the server transmits the app and the security analysis result (4). If an app violates the user's security policy, then detailed information about each violation is sent to the client. If an

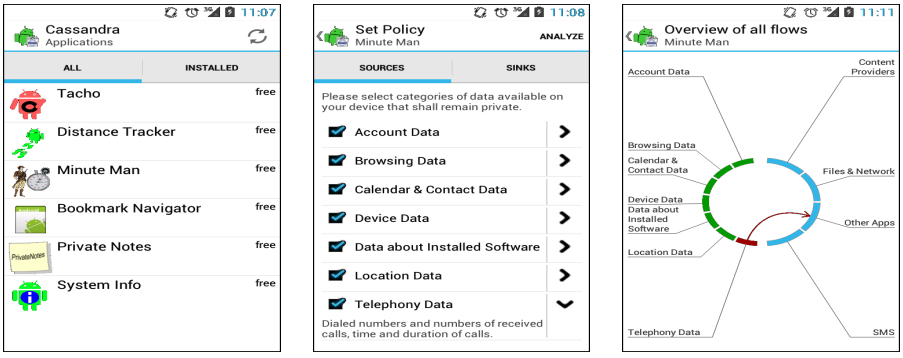

Figure 1. User interface of Cassandra's client (black and white inverted)

app satisfies the policy then, a security certificate is sent that is based on the proof-carrying-code principle [Nec97]. That is, the certificate contains enough information such that the client can efficiently replay the analysis.

Cassandra's client consists of four components (see Figure 2). The app browser (A), the policy editor (B), and the reporter (D) were explained already in Section 2. The *verifier* (C) receives the security certificate generated by the server and replays the security analysis. By this replay, Cassandra can ensure that it confirms the satisfaction of the user's security policy only if this policy is, indeed, respected. Cassandra is reliable, even if the server's security analysis is buggy, the server has been compromised, or the communication with the server has been corrupted. The security guarantees given by Cassandra to a user are as reliable as if the analysis were performed on the client, but without having to perform a computationally expensive security analysis on the mobile device.

Cassandra's server consists of three components. The *manager* (F) accepts requests for a security analysis. For each request by some client, the manager retrieves the chosen app from the *database* (G) and then calls the *certifier* (E). The certifier performs an information flow analysis for the given app and security policy. The certifier generates a security certificate if the app passed the analysis and compiles a report of all security violations, otherwise. The manager (F) forwards the analysis results returned by the certifier to the Cassandra's client.

The current version of Cassandra incorporates a certifier that performs a type-based information flow analysis (see Section 4). In the future, we plan to integrate other information flow analyses into Cassandra in addition. This was one motivation for the chosen modular architecture. Our current implementation supports the caching of analysis results to increase efficiency. By checking whether a suitable security certificate already exists before calling the certifier, the manager can avoid that an expensive security analysis is repeated.

4 Implementation

The client app of Cassandra is implemented in Java. It runs on unmodified Android devices, i.e., it requires no changes to the OS or Dalvik VM. To communicate with the server and to download apps, the client app needs the permission

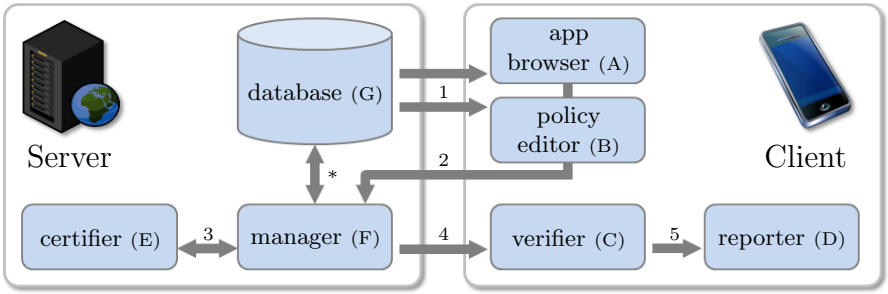

Figure 2. The components of Cassandra and their interaction

to access the Internet and the external memory of the mobile device. The server of Cassandra is implemented in Java and PHP. It is hosted on a web-server with a MySQL database and a Java runtime environment.

The security type system implemented in Cassandra tracks direct and indirect information flow. We show the typing rule for `if-testz` instructions in Dalvik bytecode as an example to give an impression of our security type system:

$$\text{tIfTestz}\frac{m[pp] = \texttt{if-testz}\ v_a, n \qquad \forall j \in region_m(pp).\mathsf{rda}(v_a) \sqsubseteq se(j)}{m, region_m, \mathsf{mda}, \mathsf{fda}, \mathsf{ada}, ret, se \vdash pp : \mathsf{rda} \rightarrow \mathsf{rda}}$$

This rule is applicable if the instruction at program point pp of method m is `if-testz` v_a, n. The instruction `if-testz` v_a, n branches to program point $pp+n$ or to $pp + 1$ depending on the value of register v_a. The typing rule requires the security environment $se(j)$ of all program points j which are executed depending on the branching at program point pp (i.e., $j \in region_m(pp)$) to be at least as private as register v_a, written $\mathsf{rda}(v_a) \sqsubseteq se(j)$. In private security environments, no public sinks must be written to prevent leaks from branching on private data.

For the complete security type system underlying Cassandra and for the soundness result, we refer the interested reader to [LMSW14].

5 Conclusion

To our knowledge, Cassandra offers the first information flow analysis for Android apps within a prototypical app store. Moreover, we are not aware of other information flow analyses for Dalvik bytecode with a soundness result. Another special feature is Cassandra's support of user-defined security policies.

In this article, we used the Minute Man app to illustrate the use of Cassandra. Cassandra's database contains further apps whose security we have successfully analyzed. Examples are a *Distance Tracker* app that measures a user's travel distance using the device's GPS location and a *Private Notes* app that allows a user to manage personal notes. The functionality of these self-implemented apps is similar to existing apps (see, e.g., [Ltd, Sol]). In the future, the collection of apps that are available in Cassandra's database shall grow further.

Acknowledgments. We thank Jan Erik Keller and Alexandra Weber for contributing to an early version of Cassandra. This work was supported by the DFG under the project RSCP (MA 3326/4-2) in the Priority Program RS^3, and by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE.

References

[EGC+10] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *USENIX Conference on Operating Systems Design and Implementation*, pages 393–407, 2010.

[EOMC11] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A Study of Android Application Security. In *USENIX Security Symposium*, 2011.

[FAR+13] C. Fritz, S. Arzt, S. Rasthofer, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Highly Precise Taint Analysis for Android Applications. Technical report, TU Darmstadt, Germany, 2013.

[FCF09] A. P. Fuchs, A. Chaudhuri, and J. S. Foster. SCanDroid: Automated Security Certification of Android Applications. Technical report, University of Maryland, USA, 2009.

[GCEC12] C. Gibler, J. Crussell, J. Erickson, and H. Chen. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. In *International Conference on Trust and Trustworthy Computing*, pages 291–307, 2012.

[KYYS12] J. Kim, Y. Yoon, K. Yi, and J. Shin. ScanDal: Static Analyzer for Detecting Privacy Leaks in Android Applications. In *Mobile Security Technologies*, 2012.

[Lim10] F-Droid Limited. F-Droid. `https://f-droid.org/`, 2010.

[LLW+12] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang. CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities. In *ACM Conference on Computer and Communications Security*, pages 229–240, 2012.

[LMSW14] S. Lortz, H. Mantel, A. Starostin, and A. Weber. A Sound Information-Flow Analysis for Cassandra. Technical report, TU Darmstadt, Germany, 2014.

[Ltd] Sports Tracking Technologies Ltd. Sports Tracker. `https://play.google.com/store/apps/details?id=com.stt.android`. Accessed in March 2014.

[MX] TouchSpot MX. Call Timer. `https://play.google.com/store/apps/details?id=com.gary.NoTePases`. Accessed in March 2014.

[Nec97] G. C. Necula. Proof-carrying Code. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 106–119, 1997.

[OMJ+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 *USENIX Conference on Security*, pages 543–558, 2013.

[Pro12] Android Open Source Project. Security Enhancements in Android 4.2. `http://source.android.com/devices/tech/security/enhancements42.html`, 2012.

[Sol] Crazelle Solutions. My Notes. `https://play.google.com/store/apps/details?id=com.crazelle.app.notepad`. Accessed in March 2014.

[Sta] A. Starostin. Cassandra Demo Video: Minute Man. `http://youtu.be/lp68uGtcjXI`. Accessed in March 2014.

[TK10] S. Thurm and Y. Iwatani Kane. Your Apps Are Watching You. Homepage of Wall Street Journal, `http://online.wsj.com/news/articles/SB10001424052748704368004576027751867039730`, 2010.

[YY12] Z. Yang and M. Yang. LeakMiner: Detect Information Leakage on Android with Static Taint Analysis. In *Third World Congress on Software Engineering*, pages 101–104, 2012.

[ZCO12] Z. Zhao and F. C. Colón Osorio. "TrustDroid": Preventing the Use of SmartPhones for Information Leaking in Corporate Networks through the Used of Static Analysis Taint Tracking. In *International Conference On Malicious And Unwanted Software*, pages 135–143, 2012.