A Case for Enforcing App-Specific Constraints to Mobile Devices by Using Trust Leases

Nuno Santos, Nuno O. Duarte, Miguel B. Costa, Paulo Ferreira
INESC-ID / Instituto Superior Técnico, Universidade de Lisboa

Abstract

In certain usage scenarios, mobile devices are required to operate in some constrained manner. For example, when movies are being screened in movie theaters, all devices in the room must be muted. However, typical mobile devices operate in unrestricted mode, allowing users to control their configurations. As a result, it is hard to guarantee that mobile devices operate under certain restrictions. In this paper, we present a security architecture that enables mobile applications to temporarily restrict the functionality of devices. To this end, we introduce a novel abstraction for mobile operating systems (MOS) called *trust lease*, which enables devices to safely switch between modes. We discuss the design implications that need to be addressed to implement this primitive on modern MOSes.

1 Introduction

Today's personal mobile devices are designed to be highly extensible and configurable by their users. A typical user has privileges to: install mobile applications on his device, grant access permissions to applications (e.g., network permissions), and configure system settings (e.g., turn off the sound, or enable flight mode). For some usage scenarios, however, such flexibility poses difficult challenges.

To illustrate why, consider the following example. In movie theaters, mobile devices must be put in silent mode during movie screenings. Because this operation must be performed on each device by its respective user, the correct enforcement of this restriction is prone to human error. In fact, it is fairly easy for someone to forget to mute their phone allowing for unfortunate calls to potentially disrupt the environment and embarrass the receiver of the call. As it turns out, ensuring that all attendees' devices are muted is hard using technical means.

Usage scenarios such as this one are characterized by the fact that mobile devices' functionality must be constrained. Android currently provides a set of APIs and system capabilities for installing a global security policy that cannot be overridden by the user [1]. Although such policies can prevent the local user from performing certain operations on the device, these mechanisms are more suitable for enterprise environments where a trusted system administrator is responsible for managing the security policies of the employees. For personal devices, alternative mechanisms are required in order to enforce such restrictions.

In this work, we propose a security architecture for mobile OSes that allows applications to temporarily restrict the operations that can be performed on a device. Such restrictions include, e.g., denying access to certain resources (e.g., network), preventing specific applications from being executed, or disallowing configuration changes (e.g., mute the device). For example, in the cinema use case just described, a mobile ticketing app can be implemented in order to automatically put the attendees' devices in silent mode during movie screenings. Before entering the screening room, the app requests the OS to mute the device until the movie finishes or the attendee leaves the room. The user must explicitly authorize the application to enforce such restrictions immediately before they are applied to the user's device.

To reason about system restrictions, we introduce an OS abstraction named *trust lease*. A trust lease represents an execution mode within the OS in which application-specific constraints apply. It must be issued by the OS upon requested by an application. Every lease must specify termination conditions (e.g., time out, location based) that force the lease to expire. When the lease expires, the user regains full control of the device. Both the termination conditions and the requested constraints must be agreed upon by both the calling application and the user. A third party can remotely attest that trust leases are enforced on a mobile device. Remote attestation is rooted in trusted hardware, such as ARM TrustZone [4].

In the rest of this paper, we further motivate our security architecture by discussing new applications that can be enabled by trust leases (Section 2). Then, we present the trust lease security architecture (Section 3), describe one use case (Section 4), and discuss some implications for MOS design (Section 5). Finally, we summarize the related work (Section 6) and conclude (Section 7).

2 Motivation

We describe three potential mobile apps that are unsupported by today's commodity mobile devices. All these

apps require the enforcement of app-specific constraints to the mobile device, feature unsupported in present MOSes. Because such apps restrict certain mobile device operations, we name them *strapps*.

Strapp mTicket. Digital ticketing apps are very popular. In general, a ticket gives its holder the right to enter a place, travel by public transport, or participate in an event. In many cases, a ticket also implies the duty to disable certain features of the holder's mobile device. For example, in airplanes, mobile devices must be shut down or turned into flight mode; in airports' customs areas, mobile devices cannot be used to take photos or make phone calls; in museums, devices cannot be used to take pictures; in movie theaters, devices must be put in silent mode, etc. However, such restrictions are oftentimes neglected by ticket holders.

The goal of strapp mTicket is to provide a digital ticketing service that restricts mobile device operations so as to abide by the terms of service that the ticket issuer established. The mTicket app running on the holders' devices maintains the digital tickets locally. When validating the ticket at the checkpoint of the restricted zone or service, mTicket will check the authenticity of the ticket and apply the necessary restriction measures to the mobile device, e.g., disable incoming calls in theaters, block the camera in museums, etc. After leaving the restricted zone or service, the restrictions previously applied will stop being enforced.

Strapp mMeet. Mobile devices are indispensable tools for professionals. Through these platforms, they have access to e-mail, calendar, web, documents, etc. However, the presence of malicious apps potentially running on their devices can be the source of information leakage. For example, malware can record private conversations using the device microphone, or keep track of the person's location; it can then stealthily transmit the resulting data to a remote site or temporarily buffer the data on local storage for deferred transmission. Because of such risks, in privacy sensitive activities (e.g., business meetings) users are currently left with no alternative but to switch off their devices. Indeed, this approach improves security, but renders devices useless.

mMeet aims at preventing information leakage from private meetings by mitigating user-space malware from the participants' mobile devices. To this end, mMeet will implement a *private mode*, which blocks the execution of every app except for a set of pre-defined trusted applications (e.g., e-mail client, contacts list). Thus, by suspending every untrusted app in the system, mMeet will disable user-space malware potentially running in the system. Before a meeting starts, each participant only needs to run mMeet and enter private mode. For improved reliability, each participant may also verify that its peers' devices have entered private mode too. To ensure that the mobile devices are trustworthy, they will remain in this mode until the meeting ends.

Strapp mExam. In academic institutions, examination procedures are fundamental to evaluate students. Small schools have shifted towards using dedicated computers as platforms for realizing exams. However, for universities with a large number of students, using dedicated computers entails significant investment. An alternative is to leverage students' own mobile devices as examination platforms. The challenge is then to prevent students from cheating. In fact, during an exam, an examinee can look up the answers for the exam by connecting to the Internet and searching the web, joining chat rooms with other colleagues, reading course material stored on the mobile device, etc.

mExam aims at turning the students' mobile devices into trustworthy examination platforms. To prevent cheating, mExam must restrict the mobile devices so as to deny potentially compromising operations, namely the execution of additional apps (e.g., browser) or access to system resources (e.g., network, SMS service). Before an exam starts, the students sitting in an examination room launch mExam on their devices. An examiner in the room authenticates students' identities and controls the exam duration. To start the process, the examiner instructs all devices to be restricted. Then, mExam displays the questions and takes students' answers. When the exam finishes, devices are unrestricted.

Discussion. These example applications help us to characterize the nature of constrains required by a strapp. We highlight three characteristics:

1. The scope of constraints is global. This is because strapps require granting or denying access permissions to objects across the entire system. One kind of constraints (R-type) regulates access to system resources (e.g., prevent access to the network) or services (e.g., prevent enabling the flight mode). Another (A-type) controls the execution of user apps. Apps may impose constraints of R-type only (mTicket), A-type only (mMeet), or both (mExam).

2. Constraints cannot be enforced indefinitely or arbitrarily. Because the user is the legitimate owner of the device, he or she must eventually regain all of his original privileges. Therefore, restrictions must be limited in time. For mExam, mMeet, or mTicket strapps, constraints will be withdrawn, e.g., when an exam time limit expires, a meeting ends, or a museum visitor finishes the tour, respectively. In addition, to prevent the enforcement of constraints against the user's will, the user must be informed of the constraints that the strapp requires and explicitly authorize them. This process occurs in the three example strapps, when a student agrees

on performing an exam (mExam), when meeting participants turn on the private mode (mMeet), and when ticket holders validate their tickets (mTicket).

3. Remote attestation capability is required. There is typically the need to assure an interested party that the relevant constraints are in place before the critical activity proceeds. Interested parties vary from case to case. They can be: the user itself, a single third party, or multiple third parties. mExam and mTicket illustrate the second case, in which there is a single interested third party, namely the examiner and the ticket issuer, respectively. mMeet illustrates both the first and the third case, because each user and its peers from a private meeting constitute interested parties.

Our goal is then to devise a security abstraction for MOSes that can accommodate the requirements of strapps. Our abstraction must be simple to understand by users and developers, and require little changes to existing MOSes. We assume that the hardware, kernel and services of the MOS are correct and, therefore, are part of the trusted computing base (TCB).

3 Trust Lease Security Architecture

To serve our goals, we propose a novel OS abstraction that enables temporary dropping of user privileges on behalf of a specific strapp running on the user's device. Because such a privilege reduction implies a reduction of trust in the user's actions and occurs for a limited amount of time, we name our abstraction *trust lease*, in which the user is the lessor, and the strapp the lessee.

Figure 1 illustrates how trust leases work. Essentially, a trust lease transitions the state of the mobile device between two modes: *unrestricted mode* and *restricted mode*. In unrestricted mode, the device is not constrained. The user can launch or terminate apps arbitrarily (A_i) and grant or deny access to resources (R_j). A_i refers to the threads and processes associated with a specific user application i. R_j represents an operation on a given system object j, which can target a peripheral (e.g., "access the camera") or a system service (e.g., "access the contact list"). Today's devices run solely in unrestricted more.

We introduce the restricted mode in which specific system constraints are enforced for a limited duration. System constraints c establish configuration permissions for a set of applications and resources: these permissions are immutable while the device is restricted. An application can be set to either stop or sticky. In the first case, the application is suspended, forcing all its threads and processes to terminate; the application cannot be started while in restricted mode. In the second case (sticky), the application is launched and cannot be terminated while in restricted mode. A resource can be set to either grant or deny, respectively preventing

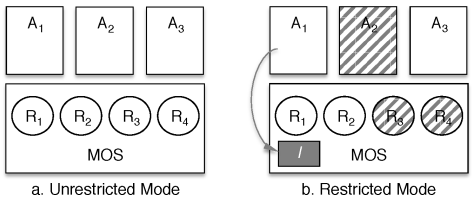

Figure 1: OS operation modes with trust leases: A_i denote applications, R_j resources, and l a trust lease.

or forcing a resource to be blocked. The duration for restricted mode depends on the occurrence of a given event e, which can be as simple as a timeout. For example, if $e = \{t = 5 \text{ min}\}$, the device will return from restricted mode after a 5 minute timeout.

To enter restricted mode, an application must issue a trust lease request to the MOS. A trust lease can be seen as a capability that controls the state of the device while in restricted mode. It consists of a tuple containing three parts: constraints c, termination events e, and the identity of the caller application a. For example, the trust lease l shown in Figure 1.b can be described as:

$l = \langle c, e, a \rangle$, where
$c = \{A_2 : \mathsf{stop}, R_3 : \mathsf{deny}, R_4 : \mathsf{deny}\}$
$e = \{\text{t} = 5\text{min}\}, a = ID(A_1)$

When an application issues a MOS call to request a trust lease, the request must be authorized by the user. This decision can be taken based on any of the elements of the lease tuple. The user can deny the authorization if he considers the constraints c too strict or too loose, judges the termination events e inadequate, or finds that the application a is not trustworthy based on elements of the application identity, such as its version, digest, developer certificate, etc. If the user denies the request, the device remains in unrestricted mode and an error is returned to the caller application. Otherwise, if the request is granted, the MOS creates a corresponding trust lease object, switches the device to restricted mode, and returns the success status and a trust lease reference. Since the caller application plays the lessee role, it can proceed executing sensitive strapp logic that requires constraints c to be in place.

The MOS remains in restricted mode until the termination event e occurs. e expresses a logical condition over a set of possible triggers, and it fires when that condition fails. Several types of triggers can be used:

- time: tests the value of timer t, which counts the time starting from the moment the trust lease was issued. As mentioned above, it can be used to set a lease timeout (e.g., $e = \{\text{t} = 5 \text{ min}\}$).
- space: tests the value of a given location provider s, such as the GPS sensor, a WiFi-based location system, or alike. This trigger can be used to

confine the restricted mode to a certain area (e.g., $e = \{s_C = \langle X, Y \rangle \wedge s_r = R\}$, which refers to a circular area centered in $\langle X, Y \rangle$ with radius R.

- abort: tests if the lessee has issued a lease abort call to the MOS. This allows the lessee application to terminate the lease explicitly. It is used as a boolean variable (e.g., $e = \{\text{abort} = true\}$)

Different triggers can be used in a termination event. For example, $e = \{\text{abort} = true \vee t = 5\text{min}\}$ fires the lease termination event as soon as either the lessee aborts the lease or a timeout of 5 minutes fires. To make sure that the lease always terminates, the system automatically adds to e the condition $t \leq T_{max}$, which imposes a predefined maximum timeout T_{max}. It is also possible to define application-specific triggers that allow the user to exit restricted mode, e.g., to make a phone call while a movie is screening, in case of emergency.

As discussed in Section 2, interested parties may require guarantees that constraints are in place. To address this need, the trust lease model provides a mechanism called *trust session*. A trust session is a secure communication channel between an interested third party, called *client*, and a local strapp that holds a trust lease. To convince a remote client that the local device did transition into restricted mode, the trust session implements a remote attestation protocol based on trusted hardware, namely ARM TrustZone technology [27]. Attestation signatures are issued by a cryptographic key bundled into the hardware by the device manufacturer. These signatures include information about the MOS identity and the current trust lease state. When a trust lease expires, the trust session is terminated and the client signaled. For flexibility reasons, the client can close and reopen a session while the trust lease is active.

4 Use Case: mExam

Trust leases provide the missing primitive to enable the implementation of strapps on mobile devices. We illustrate how by showing a concrete utilization of this primitive for the mExam use case described in Section 2. Note that this description is a caricature of a realistic mExam application. Our purpose is to convey how trust leases are concretely used and are crucial for security.

Figure 2 represents the parties of a potential mExam usage scenario and relevant interaction steps among these parties. The professor is responsible for compiling the exam questions, assigning registered students (examinees) to individual examination rooms, and assigning an examiner to each room. Examiners are responsible for supervising the examination process for the students present in their assigned rooms. Examinees will use their tablets for filling out the exam by running the mExam strapp on their devices. Examiners

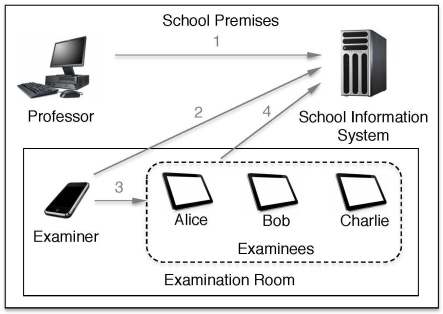

Figure 2: Usage scenario for mExam strapp example.

use their smartphones to control the examination process. For this purpose they must execute a helper mobile app called mKeeper. In addition, there is a central service—the School Information System (SIS)—which manages the examination procedures of the school.

The examination process is as follows. First, the professor sets up relevant exam details on SIS (step 1). On the predefined exam date, examinees and examiners head to their respective rooms carrying their mobile devices with them, and run the mExam and mKeeper applications, respectively. After validating the registration of every student in the room (step 2), examiners authorize examinees to start the exam (step 3) who can then retrieve the exam questions from SIS and submit the responses (step 4). For a given examinee, the exam process ends when he submits the final responses, the exam duration expires, or for some reason the examiner cancels its submission.

In this scenario, trust leases play a fundamental role to ensure that the students' devices are properly secured for the exam. During the exam, each examinee's device will be restricted by a trust lease requested by the mExam strapp running locally on each device. There are two interested parties which require trust lease enforcement guarantees before the exam questions are sent to the examinees: the examiner and the SIS. Examiner (through mKeeper) and SIS obtain these guarantees by opening trust sessions to the examinees' devices. Each examinee must explicitly approve the trust lease, after being informed of the lease request conditions. The lease constraints must deny the execution of all but the mExam application, and block all resources and system services except the required ones by mExam. The lease termination event must specify at least a timeout for the exam duration and an abort condition. Abort is issued by mExam, e.g., when the exam is submitted to SIS.

5 Implications for MOS Design

In this section, we highlight the main implications of the trust lease model to the design of a MOS. For this discussion, we take Android as the baseline MOS.

1. Security policy specification: In Android, security policies define permissions of apps in the system. E.g., android.permission.INTERNET authorizes an app to connect to the Internet. Trust leases introduce system-wide permissions that need to be specified by developers and approved by the users. It is necessary to investigate whether existing permissions can be reused in the specification of trust lease constraints. Others need to be added, namely the ability to block the execution of applications (A_i), permission that Android currently does not support. Trust leases add complexity to policies. Care must be taken to not overwhelm users and developers with cumbersome specification rules.

2. Security policy enforcement: To enforce trust lease security policies, Android's standard DAC and MAC mechanisms must be complemented with novel mechanisms. First, when a trust lease is requested to the system it will be necessary to ensure that no existing application or service holds access permissions to a resource that is disallowed by the trust lease. For example, if a trust lease requires exclusive access to the network and the browser is running, the MOS will need to take proper measures to revoke network permissions from the browser. Entering restricted mode may also require suspending running applications. The MOS must terminate all threads and processes of an app without breaking the application semantics (e.g., by losing state). When the device returns to unrestricted mode, applications must be resumed to their prior state.

3. Trust bootstrapping and remote attestation. Mobile devices running Android or other popular MOSes such as iOS, implement some form of secure boot scheme to check the integrity and authenticity of the TCB, i.e., firmware, bootloader, and operating system. This process is usually supported by dedicated trusted hardware available on the device, such as ARM TrustZone. With trust leases, stricter guarantees need to be granted. First, when establishing a trust session, a client will need to attest the integrity of the TCB and the integrity and authenticity of the lessee application. Trust session protocols implemented by the MOS and client endpoints must enable clients to remotely check the enforcement of such conditions. This can be achieved by incorporating existing remote attestation protocols [27, 28]. Second, it is necessary to ensure secure persistence of trust lease state. Otherwise, a device can become unrestricted by simply rebooting it.

6 Related Work

A large body of work aims to improve data security from untrusted, potentially malicious, applications for Android. Different systems attain this goal in several ways. Some systems focus on detecting overprivileged mobile apps [33]. Other systems [14, 18, 30] adopt information flow techniques to detect and / or prevent data leaks. Several proposals focus on improving access control mechanisms by: enabling fine-grained permissions [6, 8, 12, 19, 20, 23, 31, 32], mitigating confused deputy attacks [11, 13, 15, 16, 26], implementing mandatory access control (MAC) [7, 9, 10, 29], and providing hooking APIs to implement user-level access control models [5, 17]. The focus of all this body of work, however, is complementary to ours. In these systems the user is fully trusted. Generally they offer either application- or system-wide restrictions, rarely both. Our trust lease architecture complements these systems by introducing a restricted MOS operation mode, in which the user is no longer trusted, and where individual apps (strapps) can enforce global system constraints.

Regarding the enforcement of global system restrictions, we highlight: Digital Rights Management (DRM) and Trusted Execution Environments (TEE) systems. DRM systems [24, 25] aim at protecting copyrighted content (e.g., preventing copies between applications). The TEE considers a stronger threat model. TEE systems [2, 3, 21, 27] leverage trusted hardware to enable secure execution of small pieces of application code from potentially compromised MOSes. Systems like Flicker [22] fall in this category. All these models—DRM, TEE, and trust leases—allow for the enforcement of global system restrictions. However, the nature of these restrictions is different: DRM restricts access to content, TEE restricts access to the runtime state of a small piece of app code, and trust leases restrict access to full-blown applications and system resources.

When compared to Android's device administrator applications [1], trust leases provide a richer set of restrictions and the ability to negotiate restrictions and termination conditions dynamically without relying on a trusted system administrator.

7 Conclusions

This paper proposes a novel security model for mobile operating systems (MOS) called *trust leases*. Trust leases aim to enable the transition of commodity mobile devices between two operation modes: one in which the device is fully configurable but unsafe (*unrestricted mode*), and another in which some of the device's features are constrained, enabling it to carry out certain tasks in a trustful manner (*restricted mode*). We describe novel mobile application scenarios enabled by trust leases, and discuss the technical challenges of implementing this primitive in modern MOSes.

Acknowledgments: We would like to thank the anonymous reviewers for their feedback and insight. This work was supported by national funds through Fundação para a Ciência e Tecnologia (FCT) with reference UID/CEC/50021/2013.

References

[1] Android Device Administration API Overview. http://developer.android.com/guide/topics/admin/device-admin.html.

[2] GlobalPlatform. http://www.globalplatform.org.

[3] Samsung KNOX. https://www.samsungknox.com/en.

[4] ARM. ARM Security Technology – Building a Secure System using TrustZone Technology. ARM Technical White Paper, 2009. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.

[5] M. Backes, S. Bugiel, S. Gerling, and P. von Styp-Rekowsky. Android Security Framework: Enabling Generic and Extensible Access Control on Android. In *Proc. of ACSAC*, 2014.

[6] M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. von Styp-Rekowsky. AppGuard: Enforcing User Requirements on Android Apps. In *Proc. of TACAS*, 2013.

[7] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. MockDroid: trading privacy for application functionality on smartphones. In *Proc. of HotMobile*, 2011.

[8] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry. Towards Taming Privilege-Escalation Attacks on Android. In *Proc. of NDSS*, 2012.

[9] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry. Practical and Lightweight Domain Isolation on Android. In *Proc. of SPSM*, 2011.

[10] S. Bugiel, S. Heuser, and A. Sadeghi. Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies. In *Proc. of USENIX Security*, 2013.

[11] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing Inter-Application Communication in Android. In *Proc. of MobiSys*, 2011.

[12] M. Conti, V. T. N. Nguyen, and B. Crispo. CRePE: Context-Related Policy Enforcement for Android. *Information Security*, 6531:331–345, 2011.

[13] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach. QUIRE: Lightweight Provenance for Smart Phone Operating Systems. In *Proc. of USENIX Security*, 2011.

[14] 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 *Proc. of OSDI*, 2010.

[15] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin. Permission Re-Delegation: Attacks and Defenses. In *Proc. of USENIX Security*, 2011.

[16] M. C. Grace, Y. Zhou, Z. Wang, and X. Jiang. Systematic Detection of Capability Leaks in Stock Android Smartphones. In *Proc. of NDSS*, 2012.

[17] S. Heuser, A. Nadkarni, W. Enck, and A.-R. Sadeghi. ASM: A Programmable Interface for Extending Android Security. In *Proc. of USENIX Security*, 2014.

[18] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These Arent the Droids Youre Looking For: Retrofitting Android to Protect Data from Imperious Applications. In *Proc. of CCS*, 2011.

[19] J. Jeon, K. K. Micinski, J. A. Vaughan, N. Reddy, Y. Zhu, J. S. Foster, and T. Millstein. Dr. Android and Mr. Hide: Fine-grained security policies on unmodified Android. Technical Report CS-TR-5006, University of Maryland, Department of Computer Science, 2011.

[20] M. Kern and J. Sametinger. Permission Tracking in Android. In *Proc. of UBICOMM*, 2012.

[21] K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala. On-board Credentials with Open Provisioning. In *Proc. of ASIACCS*, 2009.

[22] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. In *Proc. of EuroSys*, 2008.

[23] M. Nauman, S. Khan, and X. Zhang. Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In *Proc. of ASIACCS*, 2010.

[24] OMA. Enabler Release Definition for DRM V2.0.1, 2008.

[25] M. Ongtang, K. Butler, and P. Mcdaniel. Porscha: Policy oriented secure content handling in Android. In *Proceedings of ACSAC*, 2010.

[26] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically Rich Application-Centric Security in Android. *Security and Communication Networks*, 5(6):658–673, June 2012.

[27] N. Santos, H. Raj, S. Saroiu, and A. Wolman. Using ARM TrustZone to Build a Language Runtime for Mobile Applications. In *Proc. of ASPLOS*, 2014.

[28] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu. Policy-Sealed Data: A New Abstraction for Building Trusted Cloud Services. In *USENIX Security Symposium*, 2012.

[29] S. Smalley and R. Craig. Security Enhanced (SE) Android: Bringing Flexible MAC to Android. In *Proc. of NDSS*, 2013.

[30] R. Spahn, J. Bell, M. Lee, S. Bhamidipati, R. Geambasu, and G. Kaiser. Pebbles: Fine-Grained Data Management Abstractions for Modern Operating Systems. In *Proc. of OSDI*, 2014.

[31] Y. Wang, S. Hariharan, C. Zhao, J. Liu, and W. Du. Compac: Enforce Component-Level Access Control in Android. In *Proc. of CODASPY*, 2011.

[32] R. Xu, H. Saïdi, and R. Anderson. Aurasium: Practical Policy Enforcement for Android Applications. In *Proc. of USENIX Security*, 2012.

[33] W. Xu, F. Zhang, and S. Zhu. Permlyzer: Analyzing Permission Usage in Android Applications. In *Proc. of ISSRE*, 2013.