POSTER: Demonstrating the Effectiveness of MOSESdroid for Separation of Execution Modes

Giovanni Russello
University of Auckland
Auckland, New Zealand
g.russello@auckland.ac.nz

Mauro Conti
Università di Padova,
Padova, Italy
conti@math.unipd.it

Bruno Crispo
Università di Trento
Trento, Italy
crispo@disi.unitn.it

Earlence Fernandes
Vrije Universiteit Amsterdam
The Netherlands
earlence@cs.vu.nl

Yury Zhauniarovich
Università di Trento
Trento, Italy
zhauniarovich@disi.unitn.it

ABSTRACT

In this poster, we describe a demo of a *light virtualisation* solution for Android phones. We named our solution **MOSESdroid** (MOde-of-uses SEcurity Separation for anDROID). MOSESdroid is a policy-based framework for enforcing software isolation of applications and data. In MOSESdroid, it is possible to define distinct *security profiles* within a single smartphone. Each security profile is associated with a set of policies that control the access to applications and data. One of the main characteristics of MOSESdroid is the dynamic switching from one security profile to another. Each profile is associated with a context as well. Through the smartphones sensors, MOSESdroid is able to detect changes in context and to dynamically switch to the security profile associated with the current context. Our current implementation of MOSESdroid shows minimal overhead compared to standard Android in terms of latencies and battery consumption.

Keywords

Android Security Extension, Separation of Modes, Light Virtualisation

1. MOTIVATION

Latest smartphone models provide computational power and storage capacity that only few years ago were exclusive realm of laptop and desktop computers. Today, smartphones enable the users to perform several tasks while being on the move carrying only a device that easily fits in a pocket. Users have since recognised the advantages of smartphones and it does not come as a surprise that in 2011 half a billion of smartphones have been sold world-wide [2].

Another incentive to smartphone sales comes from the proliferation of enterprises that adopt smartphone in their IT infrastructure. A large number of enterprises allow their employees' smartphones to connect to their IT infrastructure. Enterprise have recognised that opening their IT infrastructures to employee-owned smartphones vastly increases their productivity. This Bring-Your-Own-Device (BYOD) policy [4] has advantages for both parties: companies can save budget on acquiring and maintaining a fleet of smartphones; employees can avoid to carry around several devices (i.e. at least one for work, and one for private computing).

Given the success of smartphones, these devices are becoming an attractive target for attacks. As a consequence, we have witness a growth in the number of malware types on smartphones [1]. For instance, malicious applications may access emails, SMS and MMS messages stored in the smartphone. This poses serious security concerns to sensitive corporate data, especially when the standard security mechanisms offered by the platform are not sufficient to protect the users from such attacks [3].

One possible solution to this problem is to compartmentalise the phone, by keeping applications and data related to work separated from recreational applications and private/personal data. This separation can be achieved by supporting separate *security environments*. On the same device, one security environment can be dedicated to sensitive/corporate data and trusted applications while on another environment the user could install entertainment and third-party applications. As long as applications from the second environment are not able to access data of the first environment the risk of leakage of sensitive information can be greatly reduced.

An implementation of this solution is represented by virtualisation technologies where different instances of an OS can run separately on the same device. When deployed on fully-fledged devices such PCes and servers, virtualisation is an effective solution to create separate computation environments. Another approach that is taking momentum especially for smartphones is para-virtualisation [8]. Unlikely full virtualisation where the guest OS is not aware of running in a virtualised environment, in para-virtualisation it is necessary to modify the guest OS to boost performance. Current implementations include Trango, VirtualLogix, L4 microkernel, L4Android [6, 5]. However, these approaches are still too resource demanding for embedded systems such as smartphones especially in terms of the battery overhead during the switching between different environments.

2. MOSESDROID OVERVIEW

In this section, we provide an overview of MOSESdroid. More details of MOSESdroid can be found in [7]. As Figure 1 shows, the MOSESdroid framework extends some the modules of the Android middleware. The separation of execution is realised in MOSES-

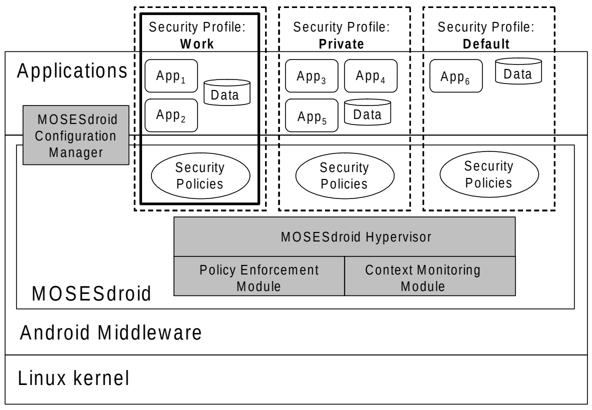


Figure 1: MOSESdroid Overview.

droid through the notion of **Security Profile** (SP). MOSESdroid supports several SP instances within the same device. By default, the "Default" SP is always present in MOSESdroid. This SP can be used for containing newly installed applications or new data stored in the smartphone (e.g., as an email attachment). The "Work" SP is used for accessing work-related data through company-approved applications. The "Private" SP is used by the user for accessing private information such as emails and SMS messages from family and friends. Also, in "Private" SP the user can install her preferred applications and games.

An SP is associated with a set of *security policies*. Through the enforcement of the security policies, MOSESdroid guarantees that applications within an SP can access only the data within the same SP. MOSESdroid achieves this fine-grained level of enforcement by means of data tainting implemented in the **Policy Enforcement Module** (PEM). Basically, when data is associated with an SP it is tainted with the SP name. The security policies specified in that SP enforce the constraint that applications can only access data tainted with the label of the same SP name. For instance, in Figure 1 the data in the "Work" SP is tainted with the label "Work". The security policies of the "Work" SP grant access to the data only to applications contained in the same SP.

The **MOSESdroid Hypervisor** (MH) is responsible for de/activating SPes. When an SP is activated, the MH loads the security policies of the SP in the PEM and enables the applications associated to the SP. Deactivating an SP is the opposite of activation: first the applications are disable and then the policies in the PEM are unloaded. However, it may require an extra step that is the killing of running applications. When an application requests access to a piece of information, the PEM grants access only if a security policy in the SP grants such request.

The switching between SPes can be manually controlled by the user. However, MOSESdroid provides a more sophisticated mechanism based on the context. In MOSESdroid, an SP can be associated with a context expression that is a boolean expression on contextual data such as location and time. When a given context expression is evaluated to true MH activates the respective SP. The evaluation of context expressions is a task performed by the **Context Monitoring Module** (CMM). For instance, the "Work" SP can be associated with a context expression that is true only during working hours and within the office facilities. In this way, the employee is allowed to access applications and data within her private profile only outside the working period and environment.

The creation of new SPes and the editing of existing ones can be done by means of the **MOSESdroid Configuration Manager** (MCM). The MCM allows the users to associate applications and data to an SP. Moreover, the MCM supports the specification of security policies and context expressions. The MSM supports the specification of *locked* SPes: a locked SP has its settings protected by a password. In order to edit the settings of a locked SP the user needs to supply a password first. For instance, the "Work" SP in Figure 1 is a locked profile where the password is only know to the IT administrator of the company where the user of smartphone works. In this way, the user owning the smartphone cannot edit. In this way, the company makes sure that its SP is not modified by the user once it is installed in the smartphone.

3. DEMO

In our demo, we demonstrate several aspects of MOSESdroid. First of all, we show how to define new SPes and how to edit existing ones. Second, we demonstrate how changing of context information drives the switching of SPes. Finally, we show the impact that MOSESdroid has on the execution of the applications.

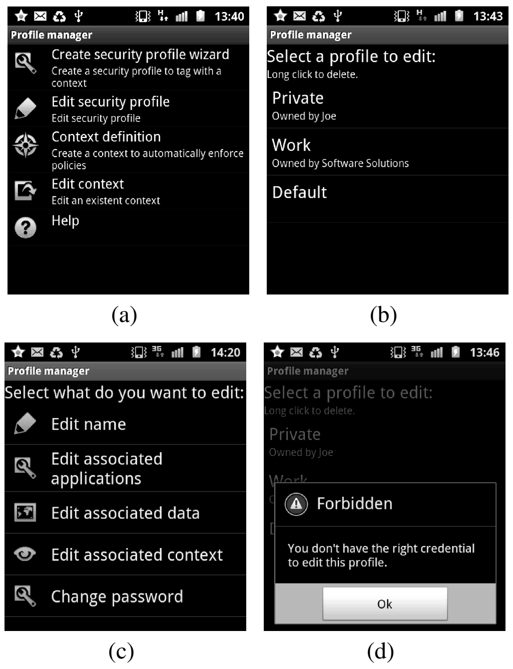
3.1 Security Profile Management

The **Profile Manager App** is an application that allows the user to create an SP and modify existing ones. The application also allows the user to define and edit context expressions that later the user can associate with an SP. The Profile Manger App stores and retrieves the context expressions to and from the **ContextDef** content provider. When a new context definition is stored in ContextDef, a conflict check is performed to avoid that the new context definition is overlapping with the context definitions already stored in the ContextDef. As a matter of fact, if two or more context definitions overlap then it might be the case that in a given situation more than one SP needs to be activated. We decided to have here a very restrictive approach by avoiding that overlapping context definitions can be stored in the ContextDef. However, as part of our future research direction we will explore conflict resolution strategies such as prioritising each SP to select the one with highest priority.

Figure 2(a) shows a screenshot of the main activity list of the Profile Manager App. If the user selects to edit an existing SP, the application retrieves the definitions of all the SPes stored in the Profile Store. The list of existing SPes is shown to the user as in Figure 2(b). Clicking on an SP in the list will bring the user to the editing activity list (Figure 2(c)). In MOSESdroid, each SP has assigned an *owner* that is the entity authorised to define and modify the SP. The owner of an SP can be the user of the device that creates her own SP. However, a user can deploy on her device SPes defined by third-parties. To protect the SP from unauthorised modification, we support several mechanisms for authenticating the SP owners, such as passwords, certificate, and biometric authentication. In case the user has no clearance, such as in the case of the "Work" SP, then a error message will be shown as in Figure 2(d).

3.2 Security Profile Switching

One of the main contributions of MOSESdroid compared to other similar approaches is the use of context for controlling the activation and deactivation of SPes. In MOSESdroid, each SP is associated with one or several contexts. A context is defined as a boolean expression over raw data from the device sensors (such as from GPS, clock, Bluetooth, etc.) and *logical sensors*, that is functions that combine raw data from physical sensors to capture specific user behaviours, such as detecting when the user is running. The evaluation of context expressions is executed by the Context Monitoring Module (CMM) (see Figure 1). When a new context expression



You don't have the right credential to edit this profile.

Figure 2: Screenshots of the Profile Manager App. (a) Main activity view. (b) View of the existing Security Profile. (c) Activities for editing an existing Security Profile. (d) Message to a user that tries to edit a profile with no clearance.

is satisfied, the CMM notifies the MOSESdroid Hypervisor (MH). If the new context expression is associated with the SP that is currently active then no further actions are needed. Otherwise, the MH initiates the SP switching.

The switching of SPes consists in executing the following steps. Firstly, the MH disables all the applications associated with the current SP. If applications are still active then the MH forces them to terminate. Secondly, the MH disables the set of security policies of the current SP that are stored in the Policy Enforcement Module (PEM). Thirdly, the set of security policies associated with the new SP are enabled in the PEM. Finally, the MH retrieves the list of applications of the new SP and enables them.

3.3 Security Policy Enforcement

The enforcement of the security policies happens within the Policy Enforcement Module (PEM). When an application requests access to a resource, the **Policy Enforcement Point** (PEP) intercepts such a request. The PEP collects information about application UID, the resource being accessed and the type of operation. The PEP forwards this information to the **Policy Decision Point** (PDP). The PDP uses the information received by the PEP to evaluate the security policies relevant to the request stored in the Policy Provider. Based on the evaluation of the policies, the PDP might decide either to allow or disallow the request. The PDP informs the PEP of the decision and then it is the responsibility of the PEP to take the necessary actions for the enforcement of such a decision.

In Android, several components are responsible for mediating access requests of applications to the device resources. Therefore, we need to connect several PEPs with these components within the Android Middleware to intercept such requests and to enforce the PDP decisions. The PEP-1 is connected with the `LibBinder` module for intercepting requests to access simple resources, such as device ID (IMEI), phone number and location data, as well as complex data such as user's calendar and contact entries.

In the `LibBinder`, we intercept the standard cursor from where we extract the `CursorWindow`. The `CursorWindow` provides methods that can be used for modifying the data contained in the cursor. Using the `CursorWindow` allows us to filter out from the cursor data only part of the information. In this way, our enforcement mechanism achieves a fine-grained filter capability. For instance, if a work application retrieves the contact entries from the contact provider, all the private contact entries can be filter out from the data contained in the `CursorWindow` before it is returned to the application.

Other PEPs are connected with some classes of the Java Framework Library (JFL) in the Dalvik Virtual Machine. In particular, the PEP-2 is connected with the `Socket` class for controlling network traffic even if sent over an encrypted socket (SSL). In the `Socket` class, we have modified the `socket.open(address)` method to inspect the address to where the data is sent. In this way, we can restrict the use of only authorised addresses or substitute the address specified by the application with an address defined by the user. By modifying the `sendStream()` method, we are able to intercept the data before it is sent and perform some actions, such as filtering or substitutions. Finally, for capturing operations on the file system, such as reading and writing on the local storage, the PEP-3 is connected with the `OSFileSystem` class.

4. REFERENCES

[1] Eric Chien. The motivations of recent android malware. `http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/motivations_of_recent_android_malware.pdf`.

[2] Gartner says worldwide smartphone sales soared in fourth quarter of 2011 with 47 percent growth. `http://www.gartner.com/it/page.jsp?id=1924314`.

[3] Gartner survey shows byod is top concern for enterprise mobile security. `http://www.gartner.com/it/page.jsp?id=2048617`.

[4] Unisys establishes a bring your own device (byod) policy. `http://www.insecureaboutsecurity.com/2011/03/14/unisys_establishes_a_bring_your_own_device_byod_policy/`.

[5] Technische Universitat Dresden and University of Technology Berlin. L4android.

[6] Matthias Lange, Steffen Liebergeld, Adam Lackorzynski, Alexander Warg, and Michael Peter. L4android: a generic operating system framework for secure smartphones. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM '11, pages 39–50, New York, NY, USA, 2011. ACM.

[7] Giovanni Russello, Mauro Conti, Bruno Crispo, and Earlence Fernandes. Moses: supporting operation modes on smartphones. In Vijay Atluri, Jaideep Vaidya, Axel Kern, and Murat Kantarcioglu, editors, *SACMAT*, pages 3–12. ACM, 2012.

[8] Yang Xu, Felix Bruns, Elizabeth Gonzalez, Shadi Traboulsi, Klaus Mott, and Attila Bilgic. Performance evaluation of para-virtualization on modern mobile phone platform. In *Proceedings of the International Conference on Computer, Electrical, and Systems Science, and Engineering*, 2010.