Enhancing Android Permission through Usage Control: A BYOD Use-Case

[Full Paper]

Fabio Martinelli, Paolo Mori, Andrea Saracino
Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
Pisa, Italy
name.surname@iit.cnr.it

ABSTRACT

The Bring Your Own Device (BYOD) paradigm, where the employees of a company install an application on their mobile devices to access company privileged information, is becoming very popular in the business environment. In order to perform their tasks, BYOD applications typically require a large set of rights which, in Android mobile devices, must be statically granted in order to have the application installed. However, this access control model is too coarse grained for the BYOD scenario, because employees would like to have a finer control on the rights granted to such applications, for instance to protect their privacy when they are not on duty. To address this issue, we propose to enhance the Android permission system through a Usage Control-based framework enabling employees to write policies which are continuously enforced while BYOD applications are running. This framework acts as a dynamic permission manager, where usage control policies grants, revokes and restores permissions to running applications on the base of mutable attributes describing the current context. Context is observed by using Android device standard APIs to monitor attributes such as mobile device location, WiFi status, battery level, current date and time, and so on. External trusted attribute providers can also be exploited.

CCS Concepts

•**Security and privacy → Domain-specific security and privacy architectures;**

Keywords

Usage Control; Android; BYOD, Mobile Devices; Application Security

SAC 2016,April 04-08, 2016, Pisa, Italy

http://dx.doi.org/10.1145/2851613.2851797

1. INTRODUCTION

New generation mobile devices, such as smartphones and tablets, have become extremely popular in the last years. These devices have enough computational power to perform almost all of the same task of ordinary desktops and laptops. Moreover, thanks to their high connectivity, embodied by several connection interfaces such as mobile data (UMTS/LTE), WiFi and Bluetooth, these mobile devices can benefit from a seamless connectivity to Internet and to other neighbouring devices. Furthermore the high availability of mobile applications (apps), whose number is always increasing, daily brings new functionalities to smartphones and tablets.

Due to the high popularity of these devices, sometimes the same mobile device is used for both business and personal purpose. This gave a breeding ground to the development of the Bring Your Own Device (BYOD) paradigm. In BYOD environment, employees use their personal smartphones or tablets for both business and private purposes. Thus, BYOD brings benefits for both employer and employee: the employer does not need to buy devices for the employee, saving money, whilst the employees do not need to bring with them two distinct devices, and not being responsible for business devices which could be lost or stolen. However, security issues may arise from misuses of this paradigm. In particular, apps used for business purpose may handle company sensitive information, which should not be accessed by other applications on the device. To this end some frameworks [17] have been developed to implement separation between a business environment and a private environment, where the business environment can only be accessed by selected apps for business purposes. These frameworks also allows the definition and enforcement of business policies, aimed at improving security and productivity, that employees should respect when at work. However, the enforcement of these BYOD policies may damage the employee, for instance, if still effective when the user is not working [6]. As an example, let's consider a BYOD policy which requires the employee to stay in the company premises during working hours and exploits employee's device location interfaces to check the current position. Such a BYOD app might be able to continue to check the employee position even outside of the working hours, causing a privacy violation. At the same time, it is not considerable a viable solution to stop the

BYOD applications when the employer is not working, since this will nullify the separation between business and private environment.

This motivates the design of a system which is able to selectively remove usage rights to BYOD applications when they are violating employee's privacy or compromising the full device functionality when the user is not at work.

Among the existing platforms for mobile devices, in this work we will focus on the Android Operating System (OS). Currently, Android is the most popular operating system for mobile devices, powering 85% of active smartphones and tablets. Android is a multi level open source platform, which provides an environment for the execution of mobile applications. Android provides a robust security architecture based on the two paradigms of *app isolation*, i.e. each app interacts only with the OS and cannot interfere directly with the execution of other apps, and access control embodied by the *permission system*, which controls whether an app is allowed or not to access a specific resource or operation. Permissions are statically granted, in bulk (all-or-nothing) to apps by the device owner, at deploy time. By default, permissions cannot be revoked once granted, but in newest Android versions some apps exists to selectively remove and grant app permissions[1].

To address the previous issue, this paper extends the standard Android permission system in order to enforce Usage Control policies, i.e., policies which dynamically grant, revoke and restore permissions to the deployed applications depending on several factors that may change over time. In particular, in this paper, we propose *UC-Droid* a context-aware Usage Control (UCON) framework for automatic and flexible control of the permissions granted to Android apps. This framework addresses the need of a more flexible permission scheme, in which the right to perform an operation or access a resource can be granted and revoked on the base of the current context, which also contemplates the possibility of right-revoke during the execution of operations. Two contexts which are related to the BYOD environment, which are considered in this work are the *at-work* context, generally active during working hours and/or when the employee is in the company premises and *private* context, which is active for the rest of the day.

UC-Droid is embodied by an Android app, which does not require custom ROMs. UC-Droid exploits, in fact, the App Ops functionalities for permission revocation, which are included in Android and accessible till Android 4.3, while they require rooting on latest releases. This requirement will be removed in the upcoming new Android release (Android 6.0), where our model can be fully integrated.

The contributions of the paper can be summarized as follows:

- We propose UC-Droid, a lightweight UCON-based framework which extends the standard Android permission system by dynamically grant and revoke permissions to running applications on the base of mutable attributes.
- We present an extension to the standard UCON model [13], including the possibility to re-grant a right after it has been revoked.
- We discuss a BYOD use-case for the proposed framework, introducing an example of UCON policy for this specific use case.
- We introduce an implementation of UC-Droid as an Android app, reporting some architectural details and performance considerations.

The rest of the paper is structured as follows. Section 2 gives some background notions about security support provided by the Android operating system. Section 3 gives a brief description of the Usage Control model and of the proposed extension. Section 4 describes the adoption of the Usage Control model in our scenario, shows the architecture of the proposed framework, presents a simple example of usage control policy, and describes the prototype implementation. Section 5 describes some related work and, finally, Section 6 draws the conclusions.

2. ANDROID SECURITY OVERVIEW

The Android framework includes several elements to enforce security on the physical device, applications and user data. The Android native security mechanisms are the Permission System and Application Sandboxing, which enforce, respectively, access control and isolation. Through the permission system, every security critical resource (e.g., camera, GPS,Bluetooth, network, etc.), data or operation is protected by mean of a permission. If an application needs to perform a security critical operation or access a security critical resource, the developer must declare this intention in the app `AndroidManifest.xml` (manifest for short) file asking the permission for each needed resource or operation. Permissions declared by the application are shown to users when installing the app, to decide if he wants to consider the application secure or not. If the application tries to perform a critical operation without asking the permission for it, the operation is denied by Android. The manifest file is bound to the application by means of digital signature. The integrity check is performed at deploy time, thus the Android system ensures that if an application has not declared a specific permission, the protected resource or operation cannot be accessed. In the last Android versions, users can dynamically revoke and re-grant specific permissions to applications, however this practice requires a level of knowledge and expertise greater than that of average users.

On the other hand, isolation is enforced through the synergy of two elements: the Runtime Environment and the underlying Linux kernel. In Android every application runs in a virtual machine (VM), thus each application has its own memory space, can act like it is the only application running on the system and is isolated from other apps. Moreover each VM is registered as a separate user of the Linux kernel. This means that each installed app is considered a user at the kernel level, able to run its own processes and with its own home folder. The home folder of each application stores application files on the device internal memory, thus it is protected from unauthorized access by the Linux kernel itself. In fact, files stored in the home folder can be accessed

[1]https://goo.gl/A5o0nI

only by the application itself. However, since the device internal memory is limited, the amount of data that can be stored in the home folder is limited and generally using the internal memory is a deprecated practice.

3. AN EXTENSION OF THE USAGE CONTROL MODEL

The Usage Control (UCON) model [13, 19, 15] extends traditional access control models introducing new decision factors besides *authorizations*, i.e., *obligations* and *conditions*, and *mutable attributes* which require the continuous enforcement of the policy. This section summarizes the main features of UCON, and introduces the extension we propose for our scenario.

Authorization predicates are evaluated to determine whether a subject requesting access to an object holds the corresponding right. This decision making phase takes into account subject/object attributes, and the action that the subject requested to perform on the object. *Obligations* are predicates that state whether certain requirements have been fulfilled in order to access objects. *Conditions* are requirements that do not depend on subjects or objects. They evaluate environmental or system status (e.g., current time or current location) to decide whether to allow access or not.

Mutable attributes represent features of the subjects or of the resources which change their values over time [14], and this could affect the execution rights of the accesses that are in progress. Examples of mutable attributes in the mobile device scenario are the physical location of the device, the battery level, the WiFi status (on/off), the current WiFI SSID, the mobile data connection status, Bluetooth status, number of active processes and apps, the employee status (on duty or not), etc. Since mutable attributes can change their values during the usage of a resource, each decision factor, besides being evaluated before the access (as in traditional access control models, *pre-evaluation*), must be evaluated also during the usage of a resource (*ongoing-evaluation*). For example, the UCON model defines two categories of authorizations: pre-Authorizations (*preA*), where the decision phase is performed when the subject requests to access the object, and ongoing-Authorizations (*onA*), where the decision phase is continuously performed while the access is in progress.

In the original UCON model, when the *ongoing evaluation* detects a policy violation, the corresponding access is terminated. However some scenarios, such as the one of this paper, simply need to suspend the access as long as the Usage Control policy is violated, but the access should be resumed as soon as the Usage Control policy is satisfied again. Hence, we propose an extension of the UCON model to satisfy this need. To describe our extension we follow a similar approach to [19], where a set of *Usage Control actions* describes the actions performed by the subject and the ones performed by the UCON system. Hence, given that the triple (s, o, r) which represents an access request where a subject s wants to access an object o through an operation that requires the right r, we extend the UCON model as follows:

- $tryaccess(s, o, r)$: performed when the subject s requests a new access (s, o, r).
- $permitaccess(s, o, r)$: performed by the system as result of the *pre-evaluation* of the policy to allow the access request (s, o, r).
- $denyaccess(s, o, r)$: performed by the system as result of the *pre-evaluation* of the policy to deny the access request (s, o, r).
- $endaccess(s, o, r)$: performed when the subject s terminates an existing access (s, o, r).
- $update(s, a)$: performed by the system to update the attribute a of subject or object s.
- $suspendaccess(s, o, r)$: performed by the system as result of the *ongoing-evaluation* of the policy to suspend an ongoing access (s, o, r).
- $resumeaccess(s, o, r)$: performed by the system as result of the *ongoing-evaluation* of the policy to resume a suspended access (s, o, r).

With respect to the traditional UCON model, the *revokeaccess* action has been replaced by two new actions *suspendaccess* and *resumeaccess*. The *suspendaccess* action, instead of terminating an ongoing access, simply suspends it, and the *ongoing evaluation* of the policy is continued for that access until the subject terminates it. The *resumeaccess* is issued when the *ongoing evaluation* of the policy for a suspended access determines that the policy is satisfied again, to state that the suspended action must be restarted.

4. USAGE CONTROL ON BYOD APPLICATIONS

As previously discussed, BYOD apps need to regulate the device activity in order to preserve sensitive business information. To perform this task, BYOD apps generally require a large number of rights, requested through permissions. Thus a BYOD app can assume the control on several device interfaces, resources and operations which, if misused, can violate the user privacy, or limiting the normal device usage. In order to deploy a BYOD app on his mobile device, the device owner must grant it all the rights it requests at installation time (i.e., the permissions that are listed in the `AndroidManifest.xml` file of the app). As a matter of fact, denying one (or more) of the requested permission would cause the app not to be deployed on the mobile device. However, this access control model is too coarse grained, and the device owner would like to have a finer control on the operations performed by the BYOD app running on his device. For example, in order to avoid to be charged for the data downloaded from the Internet by the BYOD app, the device owner could want to allow the such app to connect to the Internet only if the mobile device is connected to the WiFi of the company. Hence, as soon as the mobile device connects to another WiFi network or exploits the telco carrier connection, the BYOD app should lose the right to access the Internet. Another example concerns the usage of the Global Position System (GPS). In particular, the device owner could want to authorize the BYOD app to use

the GPS system only when he is on duty, because he does not want that the app collects data that could reveal to his Company the places (restaurants, pubs, clubs, etc.) where he usually goes when he is not on duty, and hence his habits.

To overcome this limitation, this paper proposes to extend the standard Android permission system by enabling the enforcement of Usage Control policies, i.e., policies that are continuously enforced while the controlled app is running in order to dynamically suspend and restore the right granted to this app depending on a set of factors which change over time. These policies allow the device owner to have a finer control on the operations that the BYOD app is allowed to perform on the device, because the Usage Control policy dynamically changes the rights that are granted to the BYOD app while it is running. The enforcement of Usage Control policies requires the integration of our engine, called UC-Droid, in the Android OS in order to be able to intercept the app start and termination and to change the related permission while the app is running. With reference to the UCON model defined in Section 3, in our scenario the BYOD app is the subject s which performs the operations, the mobile device is the object o on which the operations are executed, and the Android permissions are the rights r which are granted (or not) to the app. Moreover, our scenario has a further actor, i.e., the owner of the mobile device, denoted with u in the following.

4.1 Architecture

The architecture of the UC-Droid system, shown in Figure 1, is based on the reference XACML architecture [12]. In our scenario, UC-Droid is in charge of executing the *ongoing-evaluation* of the policy (see Section 3) while the application is running. Instead, the traditional access control policy evaluation (called *pre-evaluation* in UCON) relies on the standard Android Permission system. To express Usage Control policies we exploit the UXACML language, which is an extension of XACML for dealing with Usage Control peculiarities. As a matter of fact, UXACML enable the mobile device user to write policies including ongoing authorization, conditions and obligations. For a detailed description of the UXACML language please refer to [3]. The components of the UC-Droid architecture are the following:

- the Policy Enforcement Points (PEPs) are embedded in the Android OS in order to intercept the beginning and the termination of applications to send the *tryaccess* and *endaccess* Usage Control actions to the UC-Droid system. In our scenario, while applications are running, PEPs are also able to receive the Usage Control actions *suspendaccess* and *resumeaccess* from the UC-Droid system, and to change the rights granted by the Android OS to the corresponding application accordingly.

- the Context Handler (CH) is the interface of the UC-Droid system, and it communicates with the Policy Enforcement Points embedded in the Android OS, according to the Usage Control actions defined in Section 3. The CH is responsible of starting and managing the ongoing policy evaluation processes, and of coordinating the other components of UC-Droid for evaluating the ongoing policy when necessary. The ongoing policy evaluation process for a given access starts when the CH receives the related *tryaccess* action and lasts until the CH receives the corresponding *endaccess* action.

- the Policy Information Points (PIPs) are in charge of managing attributes. They provide to the CH the interfaces to retrieve, subscribe (unsubscribe) and update attributes, and they manage the interactions with the real providers of the attributes. Hence, the CH always interacts with the PIPs for what concerns attributes, and the PIPs are in charge of implementing proper and efficient strategies for providing updated attribute values. In our scenario, PIP_o is in charge of managing the attributes related to the mobile device, PIP_s manages the attributes paired with the application, and PIP_u manages the attributes related to the mobile device user. These PIPs are also able to detect attribute changes, and to trigger the policy re-evaluation.

- the Policy Decision Point (PDP) is the component which is invoked by the CH to evaluate an access requests against a policy. It is a standard XACML engine, such as the WSO2 balana one[2].

- Session Manager (SM) is the component which manages the ongoing usage sessions. When an access starts, the CH asks the SM to create a new entry in the Access Table (AT) which stores the data concerning this access. The CH exploits the SM to obtain the list of the ongoing accesses whose right should be re-evaluated when an attribute changed its value.

The work-flow of the ongoing-evaluation of the policy performed by UC-Droid involves all the previous components as follows. When an application starts, the PEP embedded in the Android OS sends a *tryaccess(s,o,r,u)* message to the UC-Droid system for each of the permissions (r) that are statically paired to the application s, in order to start to control whether they should be suspended or resumed. The CH, for each of the received *tryaccess(s,o,r,u)* messages, asks the SM to create a session to continuously verify whether the ongoing policy is satisfied while the application is running. For each session, the SM creates an entry in the Access Table (AT), which includes the data related to the session, such as the session ID, the session state, the access request, an others. When the entry is created, the session state is set to "ACTIVE". The CH invokes the PIPs to retrieve and subscribe the updated values of the attributes of the subject, object, user and environment. The PIPs return the current values of attributes and, because of the subscription, they call back the CH as soon as these attributes change. The CH sends the received access request enriched with the collected attributes to the PDP, which evaluates it against the Usage Control policy, and returns back the evaluation result. If the result is "Deny", the right r of the application s should be suspended. Hence, the CH asks the SM to update the related entry in AT setting the session state to "SUSPENDED", and sends the message *suspendaccess(s,o,r,u)* to

[2]http://xacmlinfo.org/category/balana/

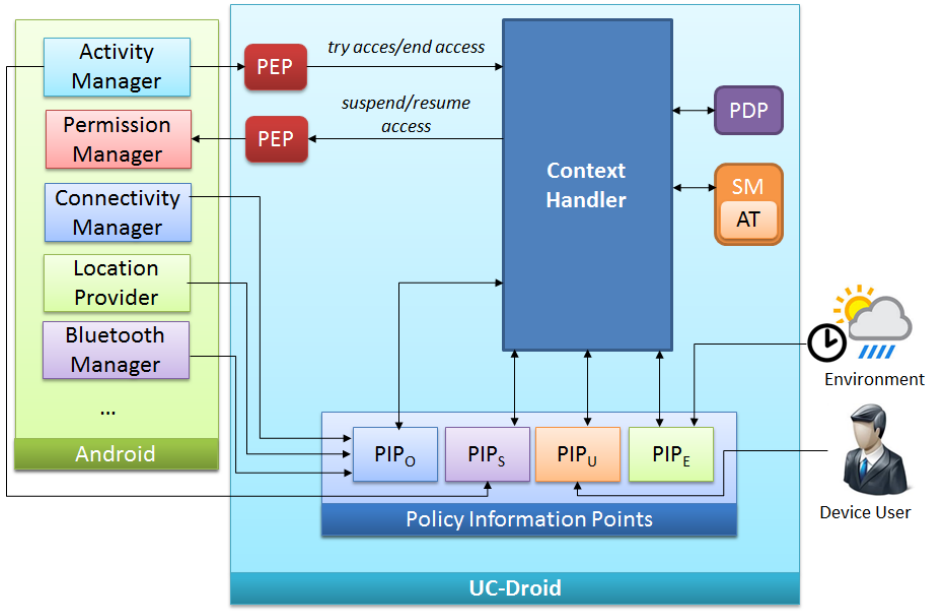

Figure 1: UC-Droid system model

the PEP, in order to temporary remove the Android permission implementing the right r to application s. Instead, if the result of the policy evaluation is "Permit", no actions are taken, because the session state is already "ACTIVE", i.e., the right is currently granted to the application.

While the application is running, as soon as one of the PIPs detects that the value of an attribute has changed, it invokes the CH to perform the policy re-evaluation process, in order to check whether the rights granted to the application must be modified. The CH retrieves from the SM the list of sessions which could be affected by this change, and the corresponding access requests are evaluated again, exploiting the new values of the attributes. For each of these sessions, the policy re-evaluation phase is the same as described before, i.e., the CH takes the original access request, enriches it with the updated attribute values retrieved from the PIPs, and invokes the PDP for a new evaluation of the Usage Control policy. Again, if the result returned by the PDP is "Deny" and the current state of the session is "ACTIVE", the right must be suspended. Instead, if the result returned by the PDP is "Permit" and the current state of the session is "SUSPENDED", the right must be restored. In these cases, the CH sends to the PEP the *suspendaccess(s,o,r,u)* and *restoreaccess(s,o,r,u)* message, respectively, to revoke or to restore the right r to the application s, and asks the SM to update the session status in the AT. If the result returned by the PDP confirms the current session status, no actions are required.

4.2 Example of Usage Control policy

This section shows a couple of rules of a possible Usage Control policy (Figure 1) which regulate the rights of using the Internet connection and the GPS system. For the sake of simplicity we don't show the full policy, which should include further rules regulating the other rights granted to applications. Moreover, we used a simple human readable policy language (although the language exploited by our framework to express Usage Control policies is UXACML, which is intended to be machine readable).

policy-1:	1
target:	2
(s.id = "BYODAppID")	3
rule-1:	4
target:	5
(r.id = INTERNET)	6
on-authorization:	7
(o.networkId ∈ COMPANY_NETWORKS) AND	8
(o.location ∈ COMPANY_LOCATIONS)	9
rule-2:	10
target:	11
(r.id = ACCESS_FINE_LOCATION)	12
on-authorization:	13
(u.onDuty = TRUE)	14

Table 1: Usage Control policy example

The policy in Figure 1 consists of two rules, rule-1 and rule-2. The target of the policy (line 3) specifies that the subject to which the rules are applicable is the BYOD application, which is identified by comparing the ID of the subject requesting the right (s.id in or example) with the ID of the BYOD application (that is known and static, "BYODAppID" in our example).

The first rule, rule-1, concerns the network usage right. In

particular, the predicate in line 6 compares the ID of the right to be controlled (represented by r.id) with the name of the corresponding Android permission, i.e., `INTERNET`. Lines 8-9 of rule-1 represent an ongoing-Authorization predicate which states that the right identified in line 6 is granted to the application as long as the WiFi network connected to the device is the company one. This implies that this right is suspended when the device is not connected to a company's network, and it is granted again when the device reconnects to one of the company's networks. To this aim, line 8 compares the attribute of the device which represent the name of the currently connected network (o.networkId) with the set of names of the company's networks (COMPANY_NETWORKS, which is known) and line 9 exploits the physical location attribute of the device (o.location) to check that the device is located in the the company area.

The second rule, rule-2, regulates the right to use the GPS device. In fact, the Android permission targeted by this rule is the `ACCESS_FINE_LOCATION` permission (line 12), and the ongoing-Authorization predicate in line 14 verifies that the device owner is on duty, by supposing to have a user attribute (u.onDuty) which is set to the value TRUE if the user is on duty and to the value FALSE otherwise.

4.3 Implementation

On the implementation side, UC-Droid is embodied by an Android app which mainly acts in background regulating the access of applications to device resources. Referring to the system model in Figure 1, the UC-Droid app is made of an internal logic part implemented by the Context Handler, Policy Decision Point and Session Manager, and an interface with the Android system implemented by Policy Enforcement Points and Policy Information Points. The CH is a bound Android service which implements client-server interactions. Initially, the PEP binds to the CH and then sends access requests using inter process communication (IPC) of Android OS. The Android IPC mechanism allows the CH to verify the identity of the PEP, and the CH is configured to accept only IPC invocations from the applications signed with the same key and this forbids other applications than the PEPs to invoke the CH. The CH may accept several access requests and each new request is evaluated in a separate thread. The PDP is implemented by customizing the available OW2 Balana XACML Engine for running on Android OS. The SM exploits the Android native SQLite library to implement the Access Table which keeps track of the data concerning active sessions. UC-Droid implements four PIPs to monitor respectively device attributes (PIP_O), the BYOD app's attributes (PIP_S), the user attributes (PIP_U) and environment attributes PIP_E. PIP_O exploits standard Android APIs to constantly monitor device's attribute. To minimize the overhead, this PIP is implemented through Services and Listeners waiting for status changes on the monitored attributes. The listeners are registered to the events that are important for the active UCON policies, and also implement the logic to notify meaningful attribute variations. For example, the listener we developed for the user location in the BYOD scheme only notifies an attribute change when the device location changes from inside the work premise to outside and vice-versa. In particular the current implementation listens to: (i) WiFi status change, monitored through a listener attached to the Network Manager, (ii) SSID name change, (iii) location change, (iv) active process list. New listeners can be easily added by following the same template of the currently active listeners in UC-Droid. The PIP_S interacts with the Activity Manager to monitor features of the BYOD app such as the amount of used CPU, memory and forked processes. The PIP_U reads information about the user status. In particular, in this BYOD use case, this PIP reports if the user (employee) is currently "on duty" or "not on duty". This information can be inferred either exploiting Android features, or received from an external server. In our implementation, we consider, for the sake of simplicity, that it is received from an external trusted server. A possible implementation will consider the moment in which the employee uses his badge to enter the work premises as starting time for the "on duty" situation. The same event for leaving the work premises will change the status to "not on duty". Finally the PIP_E monitors attribute changes concerning the environment.

The PEP keeps the association between the rights specified by policies and related Android permissions, removing and re-granting permissions at run time. The current implementation exploits the AppOps permission manager, which requires root access in systems with Android from version 4.4.2 to the latest. It is worth specifying that, if the BYOD app attempts to perform an operation for which the permission has been revoked, an exception is generated, which will likely cause the BYOD app to crash if the exception is not handled according to a graceful degradation approach. This depends from the specific implementation of the BYOD app. The graceful degradation is a developer best practice always encouraged, which will become almost mandatory with the upcoming Android 6.0, where permission revocation will be a stock feature.

4.4 Discussion

UC-Droid allows the mobile device owner to have a finer grained control on the app behavior, allowing him to define Usage Control policies which dynamically change the rights grated to apps to perform the operations protected by Android permissions.

One of the advantages of UC-Droid w.r.t. other run time policy enforcement frameworks (e.g., [20]) is that it does not require to intercept or hijack the actions performed by Android apps and, consequently, it does not require neither to install custom ROMs on the mobile device nor to modify the code of the installed apps. In fact, UC-Droid enforces the Usage Control policy by interacting with the Android permission manager, dynamically changing the permissions currently granted to the apps. The actual enforcement on the apps is demanded to standard routines of the Android system that are executed when these apps perform the actions, to check whether the corresponding permissions are currently granted or not. Consequently, UC-Droid does not introduce any direct delay in the execution of the controlled apps, because it is simply another app which runs in parallel. In fact, the policy evaluation performed by UC-Droid simply consumes computational time of the mobile device, and it does not cause any hindrance for the app execution, since it is unrelated. The overhead introduced by UC-Droid, due

to the policy evaluations, mainly depends on the number of controlled apps and rights, on the number of monitored attributes, and on how frequently these attributes change their values.

A weakness of UC-Droid is that the enforcement granularity is limited to permission level, i.e., it is not possible to regulate the usage of any resource which is not directly protected by an Android permission. For example, UC-Droid allows to prevent an app from sending text messages, but it is not possible to be selective on the amount of sent messages, or on the allowed recipients.

5. RELATED WORK

The UCON model has been exploited in many scenarios to regulate the usage of several kind of resources.

In [18], Sandhu et al propose the adoption of their model in collaborative computing systems, such as the GRID environment based on a centralized Attribute repository (AR) for attribute management. They use the eXtensible Access Control Markup Language (XACML) [12] to specify several aspects of the Usage Control model, exploiting more than one XACML policy for the different policies necessary in the Usage Control model.

Pretschner et al, in [9], propose an Usage Control enforcement mechanism for applications, showing an implementation for a common web browser and using this prototype to control data in a social network. The proposed mechanism allows the data owner to prevent data from being printed, saved, copied&pasted, etc., after this data has been downloaded by other users. Instead, in [2] Pretschner shows an application of the Usage Control model to preserve people privacy in video surveillance systems.

The work in [10], instead, addresses the problem of continuous Usage Control when multiple copies of a data object are stored in distributed systems, such as the Cloud. This work defines an architecture, a set of workflows, a set of policies and an implementation for the distributed enforcement. The policies, besides including access and usage rules, also specify the parties that will be involved in the decision process. In fact, the policy might be evaluated on one site, enforced on another, and the attributes needed for the policy evaluation might be stored in (many) other locations.

The Android permission system received several critics to be too coarse grained and of difficult interpretation for the users [7] [4]. Before the introduction of the AppOps tool to selectively remove permissions, a similar application, named TISSA, has been proposed in [20]. TISSA allows to selectively choose the permissions to revoke to apps after their installation and, differently from AppOps, apply a graceful-degradation approach for removed permissions concerning data reading, providing bogus data instead of not returning any result, avoiding apps to crash. However, in TISSA, it is still the user to choose which permissions to revoke. Differently, in our approach permissions are handled automatically, on the base of user defined Usage Control policies.

Access control on Android to enforce security is the paradigm of FireDroid: a security framework presented in [16]. FireDroid enforces security policies for controlling how processes execute system calls, but allowing policies specification at an higher level. Compared to UC-Droid, the FireDroid framework is more focused on malware detection. Moreover, FireDroid controls happens at a lower level (kernel) and requires system modification.

The framework presented in [11] enforces Usage Control policies on data shared by other users, which have been downloaded on Android devices. Although similar, the data UCON framework presented in [11] protects the shared data from the device owner, according to a Usage Control policy embedded in the data itself. UC-Droid, instead, protects the mobile device from the apps running on it by enforcing policies concerning the usage of a wide set of device resources and operations.

Another complex framework for Android which allows the enforcement of enhanced security policies on data and operations is presented in [8]. The framework, which is based on the TaintDroid tool [5], exploits information flows monitoring to have a finer grained control than Android permissions. However, differently from UC-Droid, this framework requires a custom Android ROM.

Finally, the *Android Security Framework* (ASF), presented in [1], provides security APIs to developers interested in writing security extensions for Android. Target of this framework are both manufacturer and government interested in enforcing specific security policies on users' mobile devices. Differently from UC-Droid, this framework is more oriented to control the action of the device user, whilst UC-Droid aims at improving the user protection against particularly intrusive apps.

6. CONCLUSIONS AND FUTURE WORK

This paper presented UC-Droid, a Usage Control framework for Android devices able to automatically enforce a more flexible control on permissions granted to Android apps. We mainly focused on a BYOD use-case, which represents a relevant application environment, and we described both the framework architecture and the prototype implementation. However, UC-Droid can be easily applied to other use-cases where the access control model provided by the standard Android permission system is too coarse, and the device owner needs a finer control of the right granted to apps. We envision several extension for the UC-Droid framework. For instance, we plan to provide a user friendly tool for policy authoring. Moreover, in order to be easily applied to a large number of use cases, the UC-Droid framework should be extended by defining Policy Information Points for a larger number of attributes. Furthermore, UC-Droid will be applied, with a proper set of policies and monitored attributes, to prevent malicious behaviors from infected apps (malware). Finally a large scale evaluation of UC-Droid performance will be conducted.

Acknowledgements

The work leading to this research has been partially funded by project EU-FP7 Coco-Cloud under agreement n. 610853 and EU roject H2020 NeCS under agreement n.675320.

7. REFERENCES

[1] Michael Backes, Sven Bugiel, Sebastian Gerling, and Philipp von Styp-Rekowsky. Android security framework: Extensible multi-layered access control on android. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, pages 46–55, New York, NY, USA, 2014. ACM.

[2] Pascal Birnstill and Alexander Pretschner. Enforcing privacy through usage-controlled video surveillance. In *10th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS 2013, Krakow, Poland, August 27-30, 2013*, pages 318–323. IEEE, 2013.

[3] Maurizio Colombo, Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. A proposal on enhancing XACML with continuous usage control features. In *proceedings of CoreGRID ERCIM Working Group Workshop on Grids, P2P and Services Computing*, pages 133–146. Springer US, 2010.

[4] Gianluca Dini, Fabio Martinelli, Ilaria Matteucci, Marinella Petrocchi, Andrea Saracino, and Daniele Sgandurra. A multi-criteria-based evaluation of android applications. In ChrisJ. Mitchell and Allan Tomlinson, editors, *Trusted Systems*, volume 7711 of *Lecture Notes in Computer Science*, pages 67–82. Springer Berlin Heidelberg, 2012.

[5] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information flow tracking system for real-time privacy monitoring on smartphones. *Commun. ACM*, 57(3):99–106, 2014.

[6] Ernst and Young Global Limited. Bring your own device. security and risk considerations for your mobile device program, 2013.

[7] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. Technical report, Electrical Engineering and Computer SciencesUniversity of California at Berkeley, 2012. http://goo.gl/rLEhT4. Last access, 27th of Aug, 2014.

[8] Denis Feth and Alexander Pretschner. Flexible data-driven security for android. In *Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on*, pages 41–50, June 2012.

[9] Prachi Kumari, Alexander Pretschner, Jonas Peschla, and Jens-Michael Kuhn. Distributed data usage control for web applications: a social network implementation. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy, CODASPY 2011*, pages 85–96, 2011.

[10] Aliaksandr Lazouski, Gaetano Mancini, Fabio Martinelli, and Paolo Mori. Architecture, worflows, and prototype for stateful data usage control in cloud. In *2014 IEEE Security and Privacy Workshop*, pages 23–30. IEEE Computer Society, 2014.

[11] Aliaksandr Lazouski, Fabio Martinelli, Paolo Mori, and Andrea Saracino. Stateful usage control for android mobile devices. In Sjouke Mauw and ChristianDamsgaard Jensen, editors, *Security and Trust Management*, volume 8743 of *Lecture Notes in Computer Science*, pages 97–112. Springer International Publishing, 2014.

[12] OASIS. eXtensible Access Control Markup Language (XACML) version 3.0, January 2013.

[13] Jaehong Park and Ravi Sandhu. The $UCON_{ABC}$ usage control model. *ACM Transactions on Information and System Security*, 7:128–174, 2004.

[14] Jaehong Park, Xinwen Zhang, and Ravi S. Sandhu. Attribute mutability in usage control. In *Research Directions in Data and Applications Security XVIII, IFIP TC11/WG 11.3 Eighteenth Annual Conference on Data and Applications Security*, pages 15–29, 2004.

[15] Alexander Pretschner, Manuel Hilty, and David A. Basin. Distributed usage control. *Commununcations of the ACM*, 49(9):39–44, 2006.

[16] Giovanni Russello, Arturo Blas Jimenez, Habib Naderi, and Wannes van der Mark. Firedroid: Hardening security in almost-stock android. In *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '13, pages 319–328, New York, NY, USA, 2013. ACM.

[17] SAP. Securing mobile apps in a byod world, 2013.

[18] Xinwen Zhang, Masayuki Nakae, Michael J. Covington, and Ravi Sandhu. Toward a usage-based security framework for collaborative computing systems. *ACM Transactions on Information and System Security*, 11(1):3:1–3:36, 2008.

[19] Xinwen Zhang, Francesco Parisi-Presicce, Ravi Sandhu, and Jaehong Park. Formal model and policy specification of usage control. *ACM Transactions on Information and System Security*, 8(4):351–387, 2005.

[20] Yajin Zhou, Xinwen Zhang, Xuxian Jiang, and Vincent W. Freeh. Taming information-stealing smartphone applications (on android). In *4th International Conference on Trust and Trustworthy Computing (TRUST 2011)*, pages 93–107, June 2011.