*International Journal of Innovative Computing, Information and Control*
Volume **8**, Number **2**, February **2012**

 ISSN 1349-4198
pp. **1221–1235**

# MODEL-DRIVEN SECURITY FOR TRUSTED SYSTEMS

Masoom Alam[1], Shahbaz Khan[1], Quratulain Alam[1], Tamleek Ali[1]
Sajid Anwar[1], Amir Hayat[2], Arfan Jaffar[3]
Muhammad Ali[1] and Awais Adnan[1]

[1]Security Engineering Research Group
Institute of Management Sciences
1-A, E-5, Phase VII, Hayatabad, Peshawar, Pakistan
{ masoom.alam; shaz; q.alam; tamleek; sajid.anwar; m.ali; awais }@imsciences.edu.pk

[2]School of Electrical Engineering and Computer Science
National University of Sciences and Technology
H-12, Islamabad, Pakistan
amir.hayat@seecs.edu.pk

[3]Foundation for Advancement of Science and Technology
National University of Computer and Emerging Sciences
A.K. Brohi Road, H-11/4, Islamabad, Pakistan
arfan.jaffar@nu.edu.pk

Received October 2010; revised February 2011

Abstract. *Security management in information societies is becoming extremely challenging where the resources are open and ubiquitous. The policies of the classified object change with its usage depending on the environment and its internal states. A classified object will have many stakeholders, with varying number of rights over it, all along its life. The relationship of stakeholders with objects is of our prime interest, where the object will be derived and/or distributed in its original form. In this contribution, we have classified stakeholders and objects, while concentrating on the specification for objects' usage at the stakeholder end. This will allow us to deploy our tools in an emulated environment, where these high level specifications will transform to platform independent administrative policies and administer the emulation in a dynamic manner. This work will further lead us to investigate and provide tooling support for more granular semantically integrated control for stakeholders on their objects in a ubiquitous environment. We have concentrated on resources focusing the mobile platforms, taking Android as our case study. We base our work on top of recognized standards like that provided by Trusted Computing Group's Mobile Phone Working Group. It allows us to use the strong service oriented business model available in information based societies.*
**Keywords:** Security, Trusted computing, Access control, Mobile platforms, Rights management

1. **Introduction.** Modern mobile platforms have taken over the roles of tools like the electronic gadgets, PC, netbook, camera and jukebox into an all in one package. They provide sophisticated services and they can use various modes of communications. Battery life time and performance limitations are now less of a consideration than they were a few years back. Unfortunately, the mobile platforms suffer from the same threats as its PC counterparts like malicious software and crackers. The attack vectors have been reduced by using restricted application runtime environment and strictly mediated access to file systems. Such methods enhance security by reducing the attack surface, but they do not provide any means to delegate authority to other service providers or stakeholders. This business model takes effect when the device manufacturer plays a trustworthy role

on which other service providers provide services to others, especially the user. This is mandatory for fair sharing of responsibility as the owner or user of a device is merely a consumer and she expects the service providers to take care of service delivery and its assurance.

Mobile devices have revolutionized the business model of computation, where the user consumes services but the platform is serviced by a few service providers. The mobile device and its user might be dealing with sensitive or private information, where monitoring the device for such use cases is not feasible. The user is not interested to detect and prevent intrusion by worms and viruses. Other stakeholders also expect the device to be in a condition where their data are being used according to their requirements using integrated tools of the platform with minimum possible input from the user. On the other hand, conventional anti-virus toolkits and intrusion detection modules consume battery power continuously, without any other benefit.

Such requirements can be met by regulating the business model. Normally, the device manufacturer will keep all the regulation in its control by deciding the operator and the usage constraints of the stakeholders of the device. This model works for the conventional device manufacturer's centered model but does not scale well to custom business models. The open business model for mobile computing is emerging to allow more flexibility for the stakeholders of the platform.

An example of a restricted business model is of the popular Apple's iPhone, where each developer has to submit her developed applications to an iPhone authority, and the authorities check the viability of the current application according to their mandated requirements and regulations. A device manufacturer has unlimited authority and it can be expected to be used for monopoly building. Consider the ban of Google voice application by iPhone [5], which had created a lot of distrust among the service providers and consumers.

The open model, pioneered by Google's Android, amongst others, is one prominent example of an open business model. Its web portal called Android Market [4] is where autonomous stakeholders can post their applications and user can choose among the applications. Both paid and free applications are available on the Android Market; however, this model also does not provide any flexible regulatory mechanism that fairly deals with the rights of the different stakeholders.

A generic open and fairly regulated mobile business model includes a few stakeholders using each others' resources to enable or provide service, while the user or a software component will consume these services. The user and the software component can also be expected to serve the consumed service as a derived service [6]. This is unlike the PC platforms where the platform user manages all these according to her preferences and experience, while the mobile user wants to be able to consume the platform and its services for some specific requirements and expects the service providers to assure the security of the platform with minimal input from the user.

A business model regulated by our proposed policy model for multi stakeholders forces a perspective change on how the resources on a platform should be modeled, administered and used. A resource on a platform will be used to support the user but might not be distributed by the platform provider. In such cases, the platform provider must enable the platform to provide resource and stakeholder deployments with regulations. It might as well require minimal input from user. The regulations provide the assurance and fair usage of the platform.

In our model, a resource on a platform has an owner and users. The owner can be a distributor, but she might delegate these rights to a third party, which is trusted by the resource owner and users. The distributor makes sure that the platform will use the

resource or service according to the specified regulations. It is important to note that, in order to deal with remote and different authoritative stakeholders, trust relationships have to be handled with automated policy management. Stakeholders' policies specify the regulations used to build trust relationships and manage conflicting requirements.

Our policy model is based on Trusted Computing Group's (TCG) [16] Mobile Phone Reference Architecture (MPRA) [9], which defines an open business model within its specification. In general, trusted computing deals with hardware assisted security that leverages software stack to extend the chain of trust using sophisticated policies and mechanisms. Using trusted computing to regulate policy management between stakeholders is mandatory to use the computational strength and authority of service providers, while the device user provides minimal or no input in this process. The TCG is a consortium of major vendors who specify the hardware and software requirements as a standard to be widely used. Their specifications for PC platforms are already deployed as the Trusted Platform Module (TPM) [1] in PCs and laptops.

According to MPRA specification, the manufacturer has the ultimate rights and it is the certification root for the platform and the provided operating system acts like a meta-trust engine, which provisions and allocates resources to the stakeholders. Resources of these stakeholders need to be managed at platform deployment and runtime. Most of the stakeholders, including the device owner, will use the certificates provided and rights provisioned by the manufacturer, but the manufacturer can also delegate this ability to another cloud provider, hence enabling the open business model that is *fairly* regulated by higher order legislative bodies like governments to safeguard against monopolies and unfair restrictions.

**Contributions:** The TCG MPRA specification provides only a high-level view on how a resource has to be shared among multiple stakeholders on a mobile platform. The specifications are more focused on the handling of MTM on a mobile platform and its cryptographic details with little emphasis on how concrete access and delegation policies will be defined and enforced.

Our main contributions in this paper are as follows:

- We propose a policy model called Multi stakeholder policy model (MSPM) built on top of MPRA specification by TCG mobile phone work group. Using multi stakeholder policy model, we define a concrete mechanism for access rights' management between different stakeholders on a mobile platform. In particular, how the device manufacturer's trust engine, which can closely be associated with the OS, mediates access control between services belonging to different stakeholders.
- Our target architecture is based on Android software stack, which is a sophisticated application virtual machine based OS. With the help of an example on the Android mobile platform, we show that our approach is feasible.

The rest of the paper is organized as follows. Section 2 provides the formal basis for the multi stakeholder policy model. Section 3 describes a detailed description on how multi stakeholder policy can be implemented using the Android software stack. Section 4 discusses the feasibility of our approach for the future mobile operating systems. Section 5 summarizes the related work and finally, Section 6 concludes the paper.

## 2. Model Driven Security for Trusted Systems.

A mobile platform has no single owner and multiple owners but ownership and usage rights are delegated. Thus, all resources are located on a single mobile platform but partly or fully owned by multiple stakeholders. A comprehensive policy model is needed that can depict the access rights management relationships between resources of multiple stakeholders residing on a single mobile platform at a fine grained level. Multi stakeholder policy model aims at formalizing

such relationships between different protected resources of multiple stakeholders in order to provide a secure mobile environment in which the stakes of all the owners are secured. Definition 2.1, formally defines the semantics of the multi stakeholder policy model.

Each service on a mobile device is owned by a stakeholder and is accessible by other services through a set of rights. Formally:

**Definition 2.1** (Stakeholder)**.** *A stakeholder is an entity that owns a set of services on a mobile platform. We define a function R that returns a set of rights through which a service s is accessible. Formally:*

$$R : S_v \rightarrow S_r$$

*A right $r_1$ of a service $s_1$ is written as $s_1.r_1$ where $R(s_1) \subseteq S_r$. A function K returns the stakeholder of a service, thus*

$$K : S_v \rightarrow S_k$$

Each stakeholder $s_k \in S_k$ can belong to one of the following stakeholder category sets: device manufacturer ($DM$), User ($U$), Service provider ($SP$) and communication carrier ($CC$). Except set $U$, each of the above sets can contain multiple elements. The reason is that only a single user is operating a mobile device at any time. These sets reflect the current set of stakeholders present through their services on a mobile device. For example, if a mobile device is shared between multiple employees of an organization according to their job hours, the set $U$ will contain the active user operating the mobile device. Similarly, if a flash is attached to a mobile device, the set $DM$ is updated accordingly by the mobile operating system.

Depending on the purpose for which a stakeholder has purchased a mobile device, the rights of the stakeholders in each of the category are decided. This factor is called ownership of the mobile device and has a direct impact on the access control scenarios occurring, following the mobile device is purchased.

**Example 2.1.** *As an example, consider if a hospital has purchased a set of mobile devices for its employees, then the rights from the device manufacturer have been delegated to the hospital which is a stakeholder in the service provider category. According to its organizational policy, the hospital can restrict access to certain mobile services for its employees. Alternatively, if the mobile device was directly purchased by a hospital employee and installed the medical application, she has a lot more control on the services running on the mobile device.*

Each category of the stakeholders can belong to the device owner (DO) set where $|DO| = 1$. Thus, each mobile device has a single owner.

A device owner can restrict access to certain services after purchasing the mobile device. Restriction to certain services in a multi stakeholder environment can be done at two levels: 1) at the level of device owner, it can be performed by associating constraints with the particular rights of its owned services and 2) at the level of mobile user, it can be done by performing a delegation of its owned services to other stakeholders and associating constraints with it. Delegation of rights from the device owner to other stakeholders is performed when device owner and user of the mobile device are not the same. Thus, delegation of rights explicitly describes a situation in which the mobile device is physically handed over to other stakeholders for use by the device owner. A device owner can delegate a subset of rights delegated to it from the device manufacturer, to other stakeholders including service providers, communication carriers and/or different mobile users, based on its organizational policy.

In case, mobile user is the same as the device owner, he/she can define a set of access policies regulating the usage of its owned services. The reason is that a mobile device is not physically handed over to other stakeholders. Without this aspect, both delegation of rights and access control can achieve the same regulations. Both of these levels are explained in the following subsections.

2.1. **Access control.** Each service is treated as a subject service or object service (or subject and object in short) running on a mobile device. Thus, $s \in S, o \in O \wedge S \cup O \in S_v$. We use access control matrix (ACM) to capture the behaviors of the subjects or object services running on a mobile platform. Formally:

**Definition 2.2.** *The ACM of a mobile device is defined as* $A : S \times O \rightarrow 2^{S_r}$.

Thus, if $r \in A(s, o)$, it means that subject $s$ is exercising right $r$ on object $o$. Also, if $r \in A(s, o)$, it means that the stakeholders of the subject and object are same, thus $K(s) = K(o)$. For simplicity, we assume that there is only one session for a single $(s, o, r)$ existing at one time. However, one subject can access multiple objects and one object can be accessed by multiple subjects at the same time.

Each service may have a finite set $A_{tt}$ of attribute names associated with it. Each attribute characterizing a subject or an object has a name $a$ and a domain denoted by *dom(a)* of possible values. A mobile service may consume these attribute values using a set of internal functions $F$ defined particularly for that service. These internal functions take attributes of other services as input and produces an output relevant for the corresponding service. An internal function of a service is written as $s.f$ where $s \in S_v$ and $f \in F$.

A service stakeholder can define regulations for the usage of its owned services. These regulations are enforced when services owned by other stakeholders access the corresponding service. Formally:

**Definition 2.3** (Access Control Scenario)**.** *An access control scenario describes a situation in which a service* $s$ *wants to exercise a right* $r$ *on another service* $o$*. We write* $s \rightsquigarrow_c o.r$ *to define an access control scenario, thus:*

$$\{s \rightsquigarrow_c o.r \rightarrow r \in A(s, o)\} \; if \; c = true$$
$$\{s \rightsquigarrow_c o.r \rightarrow r \notin A(s, o)\} \; if \; c = false$$

*where* $c \in C \cup \emptyset$, $s \in S$, $o \in O$, $S \cup O \in S_v$, $r \in R(o) \subset S_r$.

$C$ defines a set of constraints which are composed of attributes of services involved in a particular access control scenario or other services running on a mobile platform and a set of internal functions of the object service. A particular right of a service might be accessible without any constraint thus, $c = \phi$.

**Example 2.2.** *The medical application (introduced in Example 2.1) does not allow access to medical records if the location of the mobile device is not within hospital. thus*

$$(mobileuser \rightsquigarrow_c medicalapp.readrecord) \rightarrow$$
$$readrecord \in A(mobileuser, medicalapp)$$
$$if medicalapp.location\_within\_hospital(GPS.coordinates)$$
$$where \; mobileuser \in U, medicalapp \in SP,$$
$$location\_within\_hospital \in A_{tt}$$

Each stakeholder's service can have its own application specific usage control requirements. In our model, internal functions are used to model the behavior of individual services/applications on a mobile device. For example, the medical application has a

security requirement that determines the location of the mobile device from the GPS coordinates or a music application might allow access to music file after five mins, if payment has been made. In the above example, *location_within_hospital* is an internal function of the medical application that takes attributes of the GPS service – *coordinates*, as input and returns true or false based on the evaluation of the GPS coordinates.

**Example 2.3.** *A user defines a policy that a particular game application can use a dialer but only a specific number can be dialed thus*

$$\begin{aligned}&(gameapp \rightsquigarrow_c dialer.dial) \rightarrow \\ &\qquad\qquad dial \in A(gameapp, Ddialer) \\ &if(dialer.dialed\_number = x) \\ &where\ dialed\_number \in A_{tt}, \\ &gameapp \in SP, dialer \in DM.\end{aligned}$$

In the above example, *dialed_number* is an attribute of the *dialer* service which is matched against a predefined number $x$.

Due to the independence that each stakeholder can define constraints for its owned services and these constraints can include service attributes owned by other stakeholders, a constraint can trigger a set of access control scenarios, causing a permission dependency in two patterns 1) circular and 2) an indefinite loop. Formally:

**Definition 2.4** (Permission Dependency)**.** *A Permission dependency $P$ is defined as a situation in which the evaluation of a constraint triggers a set of access control scenarios. A permission dependency can be in two patterns: Circular and indefinite.*

*1 Circular)*

$$\{s \rightsquigarrow_c o.r, o \rightsquigarrow_{c1} s.r1, s \rightsquigarrow_{c2} o.r2, ...\}$$

*2 Indefinite loop)*

$$\{s \rightsquigarrow_c o.r, o \rightsquigarrow_{c1} s1.r1, s1 \rightsquigarrow_{c2} o1.r2, ...\}$$

In a circular pattern, the evaluation of a constraint triggers another access control scenario involving the same subject and object but their roles as subject and object are swapped. On the other hand, in an indefinite loop, the evaluation of constraints triggers other access control scenarios involving other services. A permission dependency can involve a combination of the above patterns.

In order to reduce the wastage of the limited computational power of a mobile device, a device owner can resolve permission dependencies by associating levels with it. These levels reflect the number of access control scenarios that can be triggered in response to the evaluation of a constraint.

**Example 2.4.** *The level 1 of a permission dependency means that if the medical application needs GPS coordinates for determining the location of the mobile device, the GPS service cannot call other mobile services before handing over coordinates to the medical application.*

2.2. **Delegation of rights.** A stakeholder belonging to device owner category may delegate a subset of its owned services to another stakeholder. By default, a device owner can delegate a service in two modes: Constrained (CD) and Unconstrained (UD). In unconstrained delegation, the delegator places no restriction on the respective delegation. Formally:

**Definition 2.5** (Unconstrained Delegation)**.** *An unconstrained delegation scenario describes a situation in which a stakeholder $s_{k_1} \in DO$ delegates its owned service $s_1$ to another stakeholder $s_{k_2}$. We define unconstrained delegation (UD) as a function that returns true or false if an unconstrained delegation has been done with the corresponding parameters.*

$$UD : (S_k, S_k, S_v) \rightarrow \{true, false\}$$

*Thus*

$$s_{k_1}, s_{k_2} \in S_k \wedge K(s_1) = s_{k_1} \wedge s_{k_1} \neq s_{k_2}$$
$$UD(s_{k_1}, s_{k_2}, s_1) \rightarrow K(s_1) = s_{k_2}$$

On the other hand side, the only difference in a constrained delegation scenario is that it places a restriction on the delegated services using a delegation constraint. Formally:

**Definition 2.6** (Constrained Delegation)**.** *A constrained delegation describes a situation in which a stakeholder $s_{k_1} \in DO$ delegates a service $s$ to another stakeholder $s_{k_2}$ associated with a delegation constraint. We define constrained delegation as a function*

$$CD : (S_k, S_k, S_v, DC) \rightarrow \{true, false\}$$

$DC$ defines a set of delegation constraints and a particular delegation constraint is a boolean expression, composed of a set of access control scenarios.

**Example 2.5.** *The hospital as a device owner delegates the GPS service to an employee with a restriction that he/she cannot deny GPS to the hospital's medical application.*

$$CD(hospital, mobileuser, GPS, dc_1)$$
$$dc_1 = (medicalapp \rightsquigarrow_c GPS.getGPS) \rightarrow$$
$$(getGPS \in A(medicalApp, GPS))$$
$$where\ hospital \in DO, mobileuser \in U$$

In our model, only a device owner can delegate a subset of permissions to other stakeholders. Thus, revocation is quite simple to handle. As an example, consider if a communication carrier acting as a device owner makes a restriction on the use of voice over ip (voip) applications on the respective mobile device, the mobile user should respect the corresponding restricted delegation. Otherwise, the communication carrier can block a mobile user (revocation in some sense) if a violation has been made or detected.

Assuming that a service $s$ is calling another service $o$ then, Algorithm 1 evaluates that whether access request is successful or not. In particular, before a service $s$ can make a request for an object $o$, it is verified that a delegation policy does not exist that denies the formation of the corresponding access control scenario. If both of these conditions did not meet, the algorithm finally verifies the possibility of a permission dependency by examining the access requests triggered as a result of this access control scenario. In case, there is a permission dependency, an error is returned with the message that a permission dependency exists. The algorithm is configured to operate at level 1 of any type of logical transgression. Depending on the computational power of a mobile device, the Algorithm 1 has a linear time and space complexity in the size of the policy.

In the following sections, the realization of multi stakeholder policy model using the Android software stack is detailed.

**Algorithm 1** Evaluation of an access control scenario

1. Input: $s \rightsquigarrow_c o.r$
2. Output: access request successful or not
3. $s_{k1} = K(s.r)$
4. $s_{k2} = K(o.r)$
5. $dc_1 = getDelegation(o, r)$
6. **if** $CD(s_{k_1}, s_{k_2}, o, dc1)$ **then**
7. **if** $dc_1 = s \rightsquigarrow_c o.r \rightarrow r \notin A(s, o)$ **then**
8. return 'denied'
9. **end if**
10. **end if**
11. **if** $(c1 => s \rightsquigarrow_{c2} o.r) \wedge (c2 => (o \rightsquigarrow_{c3} s.r1))$ **then**
12. return ("Error",Circular Permission dependency)
13. **end if**
14. **if** $(c1 => o \rightsquigarrow_{c2} s1.r) \wedge (c2 => (s1 \rightsquigarrow_{c3} s2.r1))$ **then**
15. return ("Error",Indefinite Computation permission dependency)
16. **end if**

3. **Target Architecture.** For the deployment of our proposed multi stakeholder policy model, we have chosen the popular open source mobile software stack Android. We believe that with the passage of time, this open source model with shape itself to a distributed and multi stakeholder environment where different stakeholders can have their application running on Android. In this section, we first briefly describe the architecture of Android along with the permission mechanism both at install-time and runtime. Afterwards, the changes we have made to this framework for the incorporation of the multi stakeholder policy model in Android, are explained.

3.1. **Android architecture.** Android is an open source platform for mobile devices with a complete stack that includes an operating system, middle ware and different applications. Android architecture is arranged in different layers. It is based on the Linux 2.6 kernel which acts as an hardware abstraction layer i.e., it includes Android's memory management programs, security settings, power management software and several hardware drivers. Above the kernel layer is the native libraries. These libraries are written in C and C++ and the core power of Android platform comes from this layer. Some of the core libraries are *Surface Manager* – which is responsible for composing different drying into the screen, *SGL* and *openGL* are used for the graphics, *Webkit* for the browser and *SQLLite* as the database.

The Next layer is the Android runtime. The main component of Android runtime is the *Dalvik Virtual Machine*. The Android runtime is designed specifically for Android to meet the needs of running an embedded environment i.e., limited battery, limited memory and CPU speed. The Dalvik VM runs the *Dex* files. These are the resultant byte codes of the .class files and .jar files at build time. These libraries are written in Java programming language and contains all of the collection classes, utilities and I/O.

Next is the Application framework layer, written in Java programming language. This is the toolkit that all applications uses i.e., built-in applications that comes with Android phone or applications written by third party developers.

The permission model of Android is an *all-or-nothing* policy model. During install, the `PackageInstaller` retrieves the permissions required by the application and shows a list of the collection to the user. The user has to click on "Install" and by doing so, she agrees

to grant all permissions to the installed application. These permissions are included in the `<uses-permission>` tag in the special manifest file – `AndroidManifest.xml`. Once the application is installed, it gets access to all the permissions requested by it in the manifest file.

Applications in Android consist of components that form parts of the whole for the sake of improved re-usability and enhanced performance. This means that any application can benefit from elements belonging to other applications, provided they have the proper permissions.

These components can be of type *Activity*, *Service*, *Broadcast Receiver* and *Content Providers*. They are initialized by the Android framework using specialized inter-process communication design patterns known as *Intents*. Intents raised for instantiation of application components are intercepted by the `ApplicationContext` class in the Android application framework. Using the different application manifests, this class makes sure that the calling application has the appropriate permissions associated with it (and thus granted at install-time). For this purpose, it creates a *Parcel* and transports it to the target application using the concepts of the `IBinder` interface and the `Binder` class. Parcel maintains some information as meta-data about its contents, which is used to manage `IBinder` object references in the buffer. The meta-data is used to maintain object references as the buffer moves across different processes.

The primary purpose of this parcel is to check whether the calling application has the permissions associated (or required by the target application). The parcel is handled by the `ActivityManager` class which passes the call to the `PackageManager` class which retrieves and then parses the manifest file associated with the application. If the permissions associated with the intent are indeed granted to the application, the `PackageManager` creates a resultant Parcel and returns the value true to the `ApplicaitonContext`. The intent can then be passed onto the target application which then resumes the operations as per its own semantics.

3.2. **Modifications to the base framework.** The core idea behind our target architecture is to make each mobile service configurable by its stakeholder, be it a service giving access to telephony related operations such as GPRS or calling facility or a simple client server application accessing a server for some data. In this way, multiple stakeholders can be supported on a single mobile platform. Using existing mobile architectures, where the responsibility of access control is taken care by a central authority or service, is no longer applicable to a multi stakeholder environment such as a mobile platform, the reason is that a central mobile service responsible for mediating access control cannot understand the access and usage control requirements of a specific service.

Existing Android framework revolve around checking the permission at a single point called `PackageManager` which is very much centralized in nature. Firstly, a single point check cannot capture the specific usage control requirements of different application services. The central security point of Android should be backed up by multiple security check points within individual services. Each service on a mobile platform whose attributes can change the state of other services such as GPS, GPRS, should be configurable by its stakeholders. For example, a GPRS service can have policy defined by its stakeholder for its restricted use in a specified time, or for limited services due to the expensive cost of the communication carrier. These application specific policies cannot be handled by Android base framework due to their varying nature. Secondly, the responsibility of the central security point is to check for the permission dependency issues such as circular or indefinite loop computation. Finally, the evaluation of these application specific

policies may require attributes of other services on a mobile platform, which is then the responsibility of the base framework to handle such requests.

The main objective of our experiment was to check the viability of multi stakeholder environment with minimal changes to the Android base framework. We have implemented the following scenario as a proof of concept implementation.

**Scenario:** A mobile user is installing a medical application on her Android which requires a GPS permission. The normal flow is that whenever an application is installed, it asks for the required permissions from the mobile user, which she can deny or grant depending on her preferences. In our case, the device owner – hospital has set a constrained delegation on the GPS service that the mobile user cannot deny GPS to its medical application thus, for the incorporation of the multi stakeholder policy model in the existing Android framework, we need to handle both the install-time and runtime aspects of the framework.

**Install-time:** The existing Android architecture uses the `AndroidManifest.xml` file for permission definition. Any application which is installed, dictates a set of required permissions from the mobile user using the `AndroidManifest.xml`. In the current Android settings, it is at the discretion of the mobile user to grant or deny the required permissions. Further, if a mobile user denies a permission, the application will not be installed. Thus, application installation is based on *all-or-nothing* policy.

In order not to have to change the schema specification of the manifest file, we have opted to include the application or service policies within the corresponding application. Each service has its own policy manifest file – `PolicyManifest.xml` – that includes the stakeholder's policies as applied to the application. The policy includes the constraints associated with the different permissions allowed by the application as well as any (constrained or unconstrained) delegation policies.

The GPS service on Android is handled by a *Java Native Interface* call to the operating system. The `com.android.internal.location` package contains several classes related to location services on the Android platform. It introduces the `LocationManager` system service, which provides an API to determine location. The `com.android.internal.location` provides `GpsLocationProvider` class which is a GPS implementation of `LocationProvider` used by `LocationManager`. It will call the `GpsLocationProvider` class which is not available to application developers in Android application development kit. This class contains Java methods e.g., `startNavigating()`, `stopNavigating()` and many others, that are used to get location. Basically, these methods call the native methods which have the actual implementation.

We have placed a hook at the point where the GPS functions makes the system call, namely `evaluatePolicy()`. This function evaluates the `PolicyManifest.xml` defined for the GPS service. In this way, the GPS service is made configurable with the help of a policy.

Figure 1 shows an example `PolicyManifest.xml` for the GPS service. The example policy dictates that the hospital as device owner imposes a delegated constraint that GPS access to the medical application cannot be denied. The GPS service on Android is accessible through the `ACCESS_FINE_LOCATION` permission. If an application is granted `ACCESS_FINE_LOCATION` permission at install-time, it can access the GPS service. In Android framework, these rights are assigned at install time only.

We have also extended the `PackageInstallerActivity` class which is responsible for handling all application installations. The extended package installer asks the GPS service directly for the `ACCSS_FINE_LOCATION` permission. Since the GPS service `PolicyManifest.xml` states that mobile user cannot deny GPS service to a specific medical application, therefore, extended package installer did not ask the permission from the mobile user. As a result, the user get the installation complete screen as shown in Figure 2. The enforce-

```
1  <policy id="0">
2   <constraineddelegation>
3       <delegator>
4             DO
5         <attribute id="name"/>
6         <attributevalue> Hospital A </attributevalue>
6          </delegator>
7         <delegatee>
8             U
9         <attribute id="name"/>
10        <attributevalue> User A </attributevalue>
11        <constraint effect="cannotDeny">
12        <service> com.hospitalA.medicalapp</service>
13         <right>GPS.ACCESS_FINE_LOCATION</right>
14        </constraint>
15 </constraineddelegation>
16 </policy>
```

FIGURE 1. PolicyManifest.xml for the GPS service

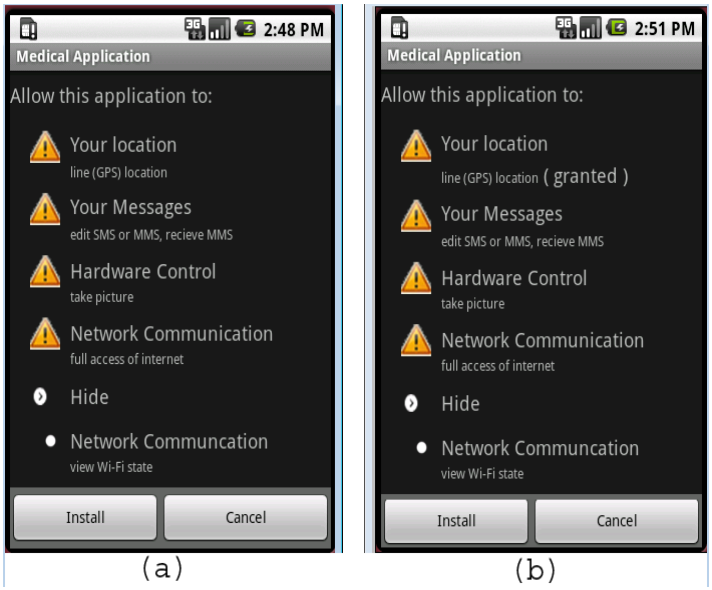


FIGURE 2. Android's **a)** normal package installer **b)** extended one

ment of stakeholder policies at runtime is described in the following section.

**Runtime:** Once the medical application is installed, the next task is to enable attribute values which the medical application or service may require for its policy evaluation. Figure 3 shows an example `PolicyManifest.xml` file for the medical application. The policy verifies the location of the mobile device through the internal function `locationwithinhospital` before handing over the medical records to the mobile user from hospital server (cf. Figure 4). A service `PolicyManifest.xml` may include attributes for the evaluation of internal functions such as GPS etc. This feature is provided in the Android through which an application can raise an intent for an attribute of a service. The `PackageManager` is already responsible for handling the permissions associated with applications. In our extended version, `PackageManager` is also responsible for managing the permission dependencies. The extended `PackageManager` resolves the attributes keeping in consideration the permission dependency rules defined in Section 2. If the attributes can be successfully resolved, they are returned to the target service

```
1  <policy id="1">
2    <accesscontrol>
3      <target>
4            <subject>mobileuser</subject>
5             <right>ReadHospitalRecord</right>
6      </target>
6      <condition function="locationwithinhospital">
7        <attribute id="name"/>
8        <attributevalue> GPS </attributevalue>
9      </condition>
10   </accesscontrol>
11 </policy>
```

FIGURE 3. PolicyManifest.xml for the medical application

which can then apply its policy internally as defined by its semantics.

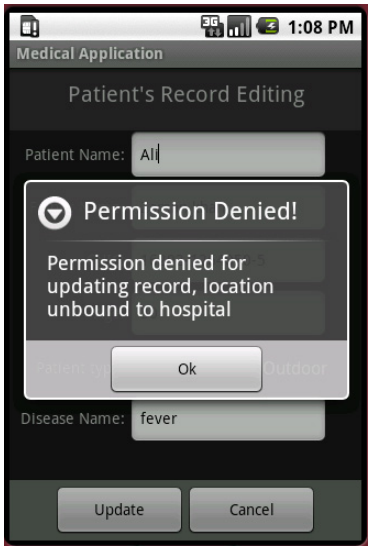


FIGURE 4. An example medical application

**Evaluation:** To measure the performance hit caused by execution of `PolicyManifest.xml`, we have carried out some preliminary experiments. These tests have been carried out on Android HTC handset developer version. The total time taken by the GPS service to evaluate policy and give results back to the medical application caused $27ms$ and with a policy in place, it took $35ms$ which is quite acceptable. We believe that this minimal performance hit, coupled with the usability of multi stakeholder policy model in the consumer market and increasing computational power of mobile devices will not make any difference.

4. **Discussion.** The basic idea behind the implementation of multi stakeholder policy model using the Android software stack is that every service on a mobile platform should be configurable using the policy of its stakeholder. This policy of a service dictates the delegation as well as the access control requirements imposed by a stakeholder. An issue is the change management of this policy on the mobile device. This means that if there is a change required within the policy of a service on a mobile platform, a stakeholder might set a special bit within the application, asking for the policy update for the corresponding

service. [13] has discussed the possibility of a proxy server between the mobile user and various stakeholders for policy change management.

Due to the feature of mobility, mobile devices are untrustworthy regarding the enforcement of the policy of a stakeholder. There is always a threat that mobile user is not fulfilling the requirements of the stakeholders of different services. Compared to its PC counterparts, stealth of data or misuse of services is rather easy on a mobile device. In order to fill this gap, trusted computing has launched the hardware root of trust called Trusted Platform Module (TPM) for the PC platform. Using TPM, a remote verifier can vouch the trusted status of a mobile device using a technique known as remote attestation.Nokia has launched a software mobile trusted module based on the specifications of MPWG [3]. The mobile trusted module will be also be launched in the near future thus, a hardware root of trust will also be available for the mobile device. Which can ensure that 1) integrity of nad 2) semantics of the policies is indeed enforced. This is currently under investigation and part of the future work.

Our work also paves the way for the analysis of the information flow properties for the mobile devices. Mobile kiosks can be in place that can make an analysis of the mobile devices and give a validity certificate upon verification. Thus, the device owner can have a tool, through which it can verify all the information flows occurring on a respective mobile device.

As the MPRA specification suggests that device manufacturer trust engine is the parent engine and should facilitate other stakeholders trust engines. A related issue is that the device manufacturer engine must take care of communication between different services and should allow only if the their stakeholders permit. In our proposed model, it is the responsibility of the corresponding service to provision resources associated with it. A device manufacturer engine or mobile operating system should protect the files of the corresponding service.

In our multi stakeholder policy model, only services are delegated and not their specific rights, thus it is a service based delegation model, as opposed to a permission based delegation model. This is due to limited computation power of a mobile device. If computational power of a mobile device permits, a more fine grained permission based delegation model can also be defined in the same way.

A policy editor both at the mobile phone and at the PC platform can be made available through which the different policies of stakeholders can be configured. For example, if the device owner and the mobile user are the same, the provision for associating constraints at run time should also be included. Android provides no support for run time policy management, and policy decisions have to be made at install time only which then can never be changed. Our current implementation is in line with the current philosophy of Android and permissions are verified at install time only. A provision for a mobile user only to change a policy at run time is discussed in [10].

Our whole idea is based on a set of reference monitors that are residing on a mobile platform, rather than a single security check point. These distributed security check points ensure the delegation as well as the access control requirements of various stakeholders.

5. **Related Work.** Over the past year, many efforts have been made at formalizing and extending the security mechanism of smart phones in the light of the existence of multiple stakeholders. Rao and Jaeger [13] have proposed a dynamic mandatory access control mechanism that is able to enforce the security policies of multiple stakeholders using runtime information. The idea is to introduce a policy proxy that is able to combine the policies associated with different stakeholder. However, the authors base their work on SELinux and are thus unable to take into consideration constraints involving predicates

of different services themselves. For example, they are unable to resolve predicates that restrict usage control requirements such as time constraints and periods of usage.

Ongtang et al. [11] have described SAINT – a mechanism aimed at Android that allows application developers to define install-time and runtime constraints on the policies associated with their applications. As the different applications running on smart phones work on the behalf of different stakeholders, this work also falls under the category of multi-stakeholder policy enforcement on smart phones. However, SAINT does not allow some core stakeholders such as users, device manufacturer and the network carrier to define their policies and is thus, in our viewpoint, an incomplete effort at this stage. On the other hand, Nauman et al. [10] describe another runtime policy enforcement framework Apex that allows users to define and enforce fine-grained usage requirements on the resources of the Android phone. This is a user-centric approach and thus does not allow some stakeholders to define their policies. It is therefore our belief that a comprehensive policy mechanism is required as presented in this paper.

Since our multi stakeholder policy model closely mimics the specifications of MPRA, it would be prudent to include some past efforts at realization of MTM on smart phones. The high-level concept of realizing stakes of multiple organizations on a mobile device using the MTM specifications was first described by Schmidt et al. [15]. In close compatibility with the MPRA specification, different stakeholders are each represented using a trusted engine which provide services to other stakeholders. This architecture forms part of our core idea in this paper. However, in the original paper, no description of the stakeholder policies or the conflict resolution between them was provided. We have described both these concepts in detail using our multi stakeholder policy model.

RBAC [14] defines a set of permissions associated with a role. In our model, the notion of a role can be assigned to different categories of stakeholders for example, a service belonging to a device manufacturer is considered as playing the role of device manufacturer on the mobile device. In contrast to RBAC, permission dependencies on services belonging to other stakeholders defines a hierarchy of permissions of services owned by different stakeholders.

Unlike traditional access control models, UCON [12] has defined two distinguishing features, namely continuity of access decisions and mutability of attributes. In our target architecture, we handle the usage control requirements of an individual service using its internal functions. Suppose, if a medical application wants to allow access to medical records five times only, or a specific number can be dialed using the dialer service by a specific application, all these requirements shall be handled within the corresponding application in our target architecture. Keeping in view variety of these usage control requirements [2, 7, 8], a global usage control engine at the mobile operating system level is neither practical nor feasible.

## 6. **Conclusion.**

In this paper, we have presented an open model for regulating rights for multi stakeholders on a mobile platform. Our policy model is built on top of the dominant standards like MPRA specified by the TCG's mobile phone work group. Resources on a mobile phone has no single owner and services owned by multiple stakeholders are running on a mobile platform. Through our approach, it is possible to realize fair behavioral control on information flows occurring on a mobile device. Further, it is possible to simulate information flows occurring on a mobile device. There are two types of policies that a stakeholder can specify namely delegation and access control. A classified object can have both of these policies configured by its stakeholder. we also presented that how this idea can be implemented using open source software stack, Android.

In this paper, we have demonstrated our experiences that focuses on packaged objects, while work is in progress to accommodate other type of objects. These *other* objects normally fall into the category of passive resources. Our work has matured to the extent that we can manage packaged objects based on our regulatory policy model for multiple stakeholders. We have demonstrated our current implementation. Our future work delves into resources that cannot be packaged. Some exceptional cases are also under our investigation.

## REFERENCES


[1] Trusted Computing Group, TCG specification architecture overview v1.2, *Technical Report*, pp.11-12, 2004.

[2] C. Chiu, C.-I Hsu and M. S.-H. Ho, The prediction of PKI security performance using PSO and Bayesian classifier, *ICIC Express Letters*, vol.3, no.4(A), pp.1031-1036, 2009.

[3] J. Ekberg and M. Kylänpää, *Mobile Trusted Module (MTM) – An Introduction*, http://research.nokia.com/files/NRCTR2007015.pdf, 2007.

[4] *Google's Android Market*, http://www.android.com/market/.

[5] *IPhone Removing Google Voice from AppStore*. http://www.crn.com/mobile/218700215; jsessionid=B2JR52TZYV1GVQE1GHRSKH4ATMY32JVN.

[6] A. Jhingran, Enterprise information mashups: Integrating information, simply, *Proc. of the 32nd International Conference on Very Large Data Bases*, VLDB Endowment, pp.4, 2006.

[7] K. Wang, J. Ren, C. Hu and R. Ma, A method for discovering security bugs of software based on AOE network, *ICIC Express Letters*, vol.3, no.4(A), pp.1081-1086, 2009.

[8] Y. Zhang, J. Ye, B. Fang and Z. Tian, A quantitative method for evaluating the security threats of grid system to tasks, *International Journal of Innovative Computing, Information and Control*, vol.5, no.4, pp.1125-1136, 2009.

[9] TCG MPWG, TCG mobile reference architecture specification, *TCG Specification Version 1.0*.

[10] M. Nauman, S. Khan, M. Alam and X. Zhang, *Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints*, 2010.

[11] M. Ongtang, S. McLaughlin, W. Enck and P. McDaniel, Semantically rich application-centric security in android, *Proc. of the Annual Computer Security Applications Conference*, 2009.

[12] J. Park and R. Sandhu, Towards usage control models: Beyond traditional access control, *Proc. of the 7th ACM Symposium on Access Control Models and Technologies*, pp.57-64, 2002.

[13] V. Rao and T. Jaeger, Dynamic mandatory access control for multiple stakeholders, *Proc. of the 14th ACM Symposium on Access Control Models and Technologies*, pp.53-62, 2009.

[14] R. Sandhu, Rationale for the RBAC96 family of access control models, *Proc. of the 1st ACM Workshop on Role-based Access Control*, pp.9, 1996.

[15] A. U. Schmidt, N. Kuntze and M. Kasper, On the deployment of mobile trusted modules, *Arxiv Preprint arXiv:0712.2113*, 2007.

[16] Trusted Computing Group, *http://www.trustedcomputinggroup.org/*.