Context-Aware Usage Control for Android

Guangdong Bai, Liang Gu, Tao Feng, Yao Guo *, and Xiangqun Chen

Key Laboratory of High Confidence Software Technologies (Ministry of Education),
Institute of Software, School of EECS, Peking University. Beijing. China.
{baigd08,guliang05,fengtao09,yaoguo,cherry}@sei.pku.edu.cn

Abstract. The security of smart phones is increasingly important due to their rapid popularity. Mobile computing on smart phones introduces many new characteristics such as personalization, mobility, pay-for-service and limited resources. These features require additional privacy protection and resource usage constraints in addition to the security and privacy concerns on traditional computers. As one of the leading open source mobile platform, Android is also facing security challenges from the mobile environment. Although many security measures have been applied in Android, the existing security mechanism is coarse-grained and does not take into account the context information, which is of particular interest because of the mobility and personality of a smart phone device. To address these challenges, we propose a context-aware usage control model ConUCON, which leverages the context information to enhance data protection and resource usage control on a mobile platform. We also extend the existing security mechanism to implement a policy enforcement framework on the Android platform based on ConUCON. With ConUCON, users are able to employ fine-grained and flexible security mechanism to enhance privacy protection and resource usage control. The extended security framework on Android enables mobile applications to run with better user experiences. The implementation of ConUCON and its evaluation study demonstrate that it can be practically adapted for other types of mobile platform.

Key words: security, access control, mobile platform, context-aware, Android

1 Introduction

During the past few years, smart phones, combining the functionalities of traditional mobile phone and increasing computing and storage capabilities, have become prevalent. They are serving more and more individuals and organizations as extensions of desktop computers. As a result, many critical applications are moved to smart phones. Unfortunately, security risks and attacks on traditional PCs have since shifted to smart phones as well [19, 12, 15].

Compared to the security of traditional computing platforms, the security of mobile devices faces more challenges [21] because they possess many unique features, including *Personalization, Mobility, Pay-for-service and Limited resources*. These distinct features require special privacy protection and resource usage constraints compared to

---

* corresponding author

PCs. *Personalization* increases the requirement for data confidentiality and privacy. *Mobility* increases the risk of device loss and theft, which leads to privacy exposure, as well the risk of classified information theft in a confidential environment, a business meeting and a military conference, for instance. *Pay-for-service* and *limited resources* make the phone prone to overcharge attacks and DoS attacks.

Android [18], a Google-led open source mobile platform, is one of the most popular mobile platforms. A series of security mechanisms such as UIDs, permission label, application signing and sandbox have been adopted into Android to enhance its security [26]. However, the permission model of Android is coarse-grained and incomplete [22]. For example, an Android application requests a list of permissions at installation; the user can only choose to either allow all these permissions or none. In addition, the user cannot revoke or change the permissions of an application once he grants the permissions, unless the application is re-installed. It cannot provide data protection and resource usage constraints in a fine-grained manner. Furthermore, there is no mechanism for the user to enforce context-aware constraints on data and resources on Android.

Some approaches have attempted to enhance the security of Android (or similar smart phone platforms) through malware detection [8, 9, 29], application certification [14] and access control [22]. However, to the best of our knowledge, no existing studies have combined context information to provide fine-grained security/privacy measures on smart phone platforms, especially on Android.

To address these challenges, we propose a context-aware usage control mechanism for the Android platform. We first present a Context-aware Usage CONtrol model (ConUCON) based on the previously proposed UCON model [27]. By taking into account the context information, such as the spatial and temporal data during runtime enforcement, ConUCON is able to support flexible data protection and resource usage constraints. Based on ConUCON, we also extend the existing security mechanism to implement a new policy enforcement framework on the Android platform. The new framework offers several new security features, such as allowing the user to grant permissions in a fine-grained manner, and supporting revocations and modifications on an application's permission at runtime.

We make the following main contributions in this paper.

– We propose a context-aware usage control model ConUCON, extending the UCON model to support context-ware protection for mobile platforms. It enables smart phone users to employ fine-grained and flexible security mechanisms to enhance the privacy protection and resource usage control.
– ConUCON provides continuous usage control because its usage decisions are not only performed prior to the access, but also during the access.
– We extend the policy specification interface of Android according to the proposed ConUCON model to provide an interface for the user to express his policy on data and resources in a context-aware and fine-grained manner. As a result it could provide better user experiences with this extended framework.
– Finally, as our extended mechanism is implemented by introducing minimal changes to the existing one, it is transparent and could easily support existing applications.

The rest of this paper is organized as follows: Section 2 describes the background, including the motivating scenarios, UCON model and Android security. Section 3

presents the ConUCON model formally. Section 4 shows the framework based on the ConUCON model. Section 5 presents the implementation and evaluation. Section 6 introduces the related works, and finally, Section 7 concludes this paper.

2 Background

2.1 Motivating Scenarios

Confidentiality and Privacy Protection A smart phone user may store private data such as photos and calendar on his/her phone. Assume that a user Alice loses her smart phone and it is picked up (or maybe stolen) by Bob. Then Bob takes it home (We can safely assume that it is a strange location) and tries to browse the data for malicious purpose (or just out of curiosity). If this unfamiliar context is detected, or Alice has ever enforced context constraints on her privacy data, an authentication will be required, which would prevent the exposure of Alice's privacy.

Resources Usage Constraints Some services such as GPRS and voice calls may charge extra fees according to the usage time and user's location, that is, incurring significantly more expenses at a certain time in the day, or if the user roams out of a certain area. Thus, the user may tend to restrict applications' usage on these resources during specific periods or at locations with higher charge rate.

In a government or military meeting, in which confidentiality is specially concerned, the participants are required to disable certain functions of the phone such as audio capture. If the participants' phones can detect the meeting-related contexts (time and location), it would be possible to disable corresponding functions automatically.

2.2 UCON Model

UCON [24, 27, 23, 34] is a generalized security model proposed by Sandhu *et. al* to cover a variety of security aspects including obligations, conditions, continuity and mutability, etc. The UCON model consists of eight components: subjects, subject attributes, objects, object attributes, rights, authorizations, obligations, and conditions. The first five hold similar meanings with the concepts in traditional access control models, while authorizations, obligations and conditions impact on the usage decisions. Authorizations permit or deny an access from a subject to an object with a particular right based on attributes of subject and object. Obligations require the subject to perform specific actions before (pre) or during (ongoing) an access. Conditions are environmental factors.

2.3 Android Security

Android is a software stack for mobile devices, and it contains an operating system, middleware and key applications. The applications in Android consist of four different types of components: activities, services, broadcast receivers and content providers. Most security mechanisms on Android are enforced at the application level. Each application is assigned with a unique UID at install-time. At runtime, by adopting a sandbox

mechanism to run applications as separate processes, Android protects them from modifying or controlling each other.

To use some protected resources, such as the dialer or GPRS, an application must include a file named `AndroidManifest.xml`, which contains several <uses-permission> tags to declare the required permissions. During installation, the package installer will list these permissions to the user, who can then choose to grant all permissions to the application, or deny all permission requests and withdraw the installation. Once all permissions are granted, the application will be allowed to use the resources without reminding the user all the time. The user cannot revoke the permissions unless the application is reinstalled.

3 ConUCON: A Context-aware Usage Control Model

This section presents the proposed context-aware usage control model ConUCON, which consists of three major parts: model components, user policy specifications and runtime usage decisions. ConUCON can leverage the context information to enhance the security of mobile computing platforms, and it serves as the foundation of our extended security framework for Android platform.

3.1 Model Components

ConUCON contains the following components: subjects, objects, states (which include subject attributes and object attributes), rights, permissions, obligations, and contexts. We will introduce the definitions and descriptions of these components in this section.

The concepts of subjects and objects remain similar with those in traditional access control models as well as the UCON model.

Definition 1. *(**Subjects** and **Objects**) A **subject** is an entity that holds and exercises certain rights on objects. An **object** is an entity that subjects can access or use. **Subject set** and **object set** are denoted by S and O, respectively.*

Example 1. For example, subjects can be applications and components in Android, while objects can be files, resources, and services.

Definition 2. *(**Attributes**) An **attributes** is a property used in usage decisions,* such as UID, software producer, permission label and path of an object. All attributes are contained in the **attribute set** (AT).

Each subject or object is associated with a corresponding attribute set, which can be queried with the function $\tau : S \cup O \rightarrow \mathscr{P}(AT)$. For a subject or object $so \in S \cup O$ that holds an attribute $at \in \tau(so)$, the value of the attribute $so.at$ can be retrieved with the function $\upsilon : (S \cup O) \times \tau(S \cup O) \rightarrow ran(\tau(S \cup O))$, where $ran(a)$ is the value of attribute a.

Example 2. A Telecom Provider may provide a specific number of free SMSes for users every day, and will charge fees for any extra message. As a result, the user may wish to prohibit the corresponding applications to send messages once the quota for a day is exhausted. Thus, the usage history of the SMS service should be recorded as an attribute of the object and be involved in the usage decision process.

Definition 3. *(States) A **state** is defined as a set, whose elements are triples*
$$(so, at, val), where\ so \in S \cup O,\ at \in \tau(so)\ and\ val = \upsilon((so, at)).$$

A state element consists of an attribute set, the owner of the attribute set, and the values of these attributes in the set. A state is a subset of the **State Set**(ST), which contains all the attributes and their values.

An update action is defined as a function $\mu : \mathscr{P}(ST) \rightarrow \mathscr{P}(ST)$.

Definition 4. *(Rights) A **right** is an operation that a subject can perform on an object. All Rights comprise the **Right set**(R).*

Rights can be divided into several functional categories. For files and other data, the rights include read, write, delete, etc.; while for resources and services, the rights include use, disable, etc. The Right set is defined by users.

Definition 5. *(Permission Labels) A **permission label** is a credential to allow a subject to perform a specific right on corresponding objects, which are assigned to subjects and objects. All permission labels comprise the **Permission label set**(P).*

For a subject, its permission labels determine which objects it can access, while the labels for an object determine which subjects can access it. Each subject owns a permission label set, which can be retrieved using the function $\varphi_s : S \rightarrow \mathscr{P}(P)$.

Each resource object and service object can be attached with a permission label [26] to declare the permission required to use it. The function $\varphi_o : O \rightarrow P$ is defined to query the label.

It is a bit more complex for data objects. Each of the objects has two labels, one for confidentiality and the other for integrity. The confidentiality label is an element of the ***confidentiality label set(CL)***, while all integrity labels comprise the ***integrity label set(IL)***. A subject is also associated with these two labels to indicate its confidentiality level and integrity level, respectively. The orders of the confidentiality level and integrity level are denoted by $\{\leqslant_c, \geqslant_c\}$ and $\{\leqslant_i, \geqslant_i\}$, respectively.

The functions $\varphi_c : S \cup O \rightarrow CL$ and $\varphi_i : S \cup O$ are defined to retrieve the confidentiality and integrity labels of a data object or a subject, respectively.

Definition 6. *(Obligations) An **obligation** is a mandatory action that must be performed before or during an access. It is an element of the **obligation set**(OB).*

Example 3. To avoid privacy exposure caused by trojans such as Pbstealer.A [16], a user may require all applications that access the contacts to disable Bluetooth before and during the access. Thus the obligation for these applications is to disable Bluetooth by themselves or to agree the usage control decision process to disable it.

Definition 7. *(Contexts) A **context** is defined as a property of environment and system. The type of a property is the **context type**, which is an element of **context type set**(CT).*

The examples of context types are CPU rate, battery, device location and time. For ConUCON, We focus on the contexts related with the system and environment. In addition, there is a subtle difference between context and attribute: a context is a property of systems or physical environment, whereas an attribute is a property directly related to a subject or an object.

Continuous evaluation is critical on mobile platforms because of their features mentioned before. Thus, the evaluation of context constraint in ConUCON is performed before (pre) and during (ongoing) an access. For example,

Example 4. A user has required the permission to read a confidential article which is restricted to be read only in a specific area. While browsing the article, the user roams out of the restricted area unconsciously. The smart phone should trigger a warning as soon as it detects this situation.

3.2 Environment Contexts

For a smart phone platform, environment contexts such as spatial and temporal contexts are especially important.

Spatial Context *A* **spatial context** is defined as a spatial property. **Spatial context** $\in$ CT. We adopt the geometric model of GEO-RBAC [13] to model the positions.

Definition 8. *(**Features and Feature Types**) A **feature** is an object which indicates an entity that occupies a space in real world, which is identified by feature name. The features are included in **feature set**(F). Each feature has a **feature type** contained in **feature type set**(FT).*

*A feature can be mapped to a **geometry** on Earth. A **geometry** is an object in Euclidean space with a coordinate, which is an element in the **geometry set**(GEO).*

The functions $\gamma : F \to FT$ and $\xi : F \to GEO$ are used to get the feature type and the geometry of a feature.

Definition 9. *(**Feature Order and Feature Type Order**)*

- *feature type order* $(\leqslant_{ft}) :$ $ft_1 \leqslant_{ft} ft_2$ iff $\forall f_1 \in F \wedge \gamma(f_1) = ft_1, \exists f_2 \in F \wedge \gamma(f_2) = ft_2, \xi(f_1) \subseteq \xi(f_2)$
- *feature order* $(\leqslant_f) :$ $f_1 \leqslant_f f_2$ iff $\gamma(f_1) \leqslant_{ft} \gamma(f_2) \wedge \xi(f_1) \subseteq \xi(f_2)$

Example 5. Office 2E315, Pentagon, Arlington, Virginia are examples of features, whose feature types are Room, Building, County, State, respectively. The Room and Building satisfy the $\leqslant_{ft}$ order, while Arlington and Virginia satisfy the $\leqslant_f$ order.

Definition 10. *(**Real Position and Logical Position**) A **real position** is a position on the Earth and can be obtained using a device such as a GPS based equipment, while a **logical position** is a semantic representation of a position. **Real position set** and **logical position set** are denoted as RP and LP, respectively.*

Obviously, a real position corresponds to a geometry and a logical position corresponds to a feature. A real position may correspond to one or more logical positions under different feature types. For example, a region may correspond to a room or part of a city, when assigning it with these two feature types.

The function $\rho_{ft} : RP \to LP$ is used to map a real position to the corresponding logical position under feature type ft.

Thus, we can define the inclusion relation between a real position and a logical position $\sqsubseteq_p$: $rp \sqsubseteq_p lp$, where $rp \in RP \wedge lp \in LP$ iff $\rho_{\gamma(lp)}(rp) \leqslant_f lp$.

Temporal Context *A* **temporal context** is defined as a temporal property. **Temporal context** $\in$ CT.

Definition 11. *(Time Instants) A time instant is a time point that has the form*
$$TI := mm/dd/yy\_hh : ii : ss \quad where$$
$mm \in \{1, 2, ..., 12\} \wedge dd \in \{1, 2, ..., 31\} \wedge yy \in \mathbb{N} \wedge hh \in \{0, 1, ..., 23\} \wedge ii,\ ss \in \{0, 1, ..., 59\}$.

The definition of the periodic expression in ConUCON is based on past studies in [6, 31, 5]:

Definition 12. *(Periodic Expression) The* **periodic expression** *is defined as*
$PE := Y | W$
$Y := R.years | R.years \rhd S.years | R.years + M$
$W := weeks | weeks + D$
$M := R.months | R.months \rhd S.months | R.months + D$
$D := R.days | R.days \rhd S.days | R.days + H$
$H := R.hours | R.hours \rhd S.hours | R.hours + M$
$M := R.minutes | R.minutes \rhd S.minutes$
$where\ \ R \in 2^{\mathbb{N}} \cup \{all\},\ S \in \mathbb{N}.$

Example 6. We can use the periodic expression $years + 7.months \rhd 6.months$ to indicate the second half of every year and the expression $weeks + \{1, 2, ...5\}Days + 9.hours \rhd 8.hours$ to indicate working hours of every week.

As a result, we can define the inclusion relation between a time instance and a periodic time [6] $\sqsubseteq_t$: $ti \sqsubseteq_t \langle [begin,\ end],\ P \rangle$ if and only if there exists a time interval $it \in \Pi(P)$ such that $ti \in it$ and $begin \leqslant ti \leqslant end$, where $\langle [begin,\ end],\ P \rangle$ is a periodic time, $begin$ and end are two time instants, $\Pi(P)$ is the set of time intervals corresponding to the periodic expression P.

Example 7. $PT = \langle$ [01/01/2010_00:00:00, 12/31/2012_23:59:59], $weeks + \{1, 2, ...5\}.Days + 9.hours \rhd 8.hours \rangle$ indicates the working hours during the year 2010 and year 2012. A time instant 4/19/2010_14:30:00 $\sqsubseteq_t$ PT.

3.3 User Policy Specification

The policy specification allows a user to specify his security policies on usage, i.e. data and resources. The security policies describe:

– Which permission label should be assigned to a resource object? Which confidentiality label and integrity label should be assigned to a data object? Which permission label set should be assigned to a subject?
– If a subject requests to perform a specific action (right) on an object, what authorizations, obligations and contexts should be satisfied before (pre) and during (ongoing) the access?

Definition 13. *(**Label Policies**) Define function $\varpi_{o_1} : O \to P$ to impose a permission label to a resource object, function $\varpi_{o_2} : O \to CL \times IL$ to impose confidentiality label and integrity label to a data object, and function $\varpi_s : S \to \mathscr{P}(P)$ to grant a permission label set to a subject.*

Definition 14. *(**Usage Control Policy**) The **usage control policy** is used to specify authorizations, obligations and contexts that should be satisfied before (pre) and during (ongoing) a subject performing a specific action (right) on an object. All the usage control policies are included in the **usage control policy set** (UP).*

$UP \subseteq S \times O \times R \times PreOb \times OnOb \times StateConstraint \times PreContext \times OnContext \times Update, \ where$

- $PreOb, \ OnOb \in \mathscr{P}(OB)$
- *StateConstraint := (StatePredicate) | ¬ StateConstraint | StateConstraint ∨ StateConstraint | StateConstraint ∧ StateConstraint. (StatePredicate is a relational expression, with form of $f(\mathscr{P}(S \cup O \times AT))relator\langle value\rangle$, where f is an operation expression using the attributes as operands, while $relator$ is a logical operator.)*
- $PreContext, OnContext \subseteq ContextConstraint$, where *ContextConstraint := (ContextPredicate) | ¬ ContextConstraint | ContextConstraint ∨ ContextConstraint | ContextConstraint ∧ ContextConstraint*
  $ContextPredicate := \langle CT\rangle relator\langle value\rangle \ | \ PeriodicTime \ | \ LP$

Example 8. Let's consider the *Resources Usage Constraint* scenario in Section 2.1 to illustrate the User Policy Specification. At first, a smart phone user Alice may restrict the usage of the camera in her phone as following:

- To prevent conflict, all applications that apply for using the camera must close or remind the user to close the other application that are using the camera.
- To keep privacy, all applications that are recording video must pause recording when an incoming call comes.
- To preserve battery for more critical functions, the camera should be disabled when the remaining battery power is blow 30%.
- If one application was denied a short time ago(one minute, for instance), its request should be denied automatically.

Now suppose there is a confidential meeting in the company where Alice works during 10:00 to 12:00 every Wednesday and Thursday in 2010, and video recording is not allowed at the meeting. We can specify the policy as in Table 1.

3.4 Runtime Usage Decisions

We employ the Bell-LaPadula model [4] for confidentiality and the Biba model [7] for integrity to express the authorizations in ConUCON. The other appropriate security models can be used to express specific application constraints in ConUCON.

Definition 15. *(**Authorizations**) **Authorizations** are used to check whether a subject is allowed to perform an action on an object, according to specified security model, such as integrity models and confidentiality models. The function $\Omega : S \times O \times R \to \{true, false\}$, which is used to get the authorization result, is defined as:*

Table 1. An example of usage control policy

Components	Constraints
Subject	All
Object	Camera
Right	Use
Pre-obligation	ObligationID$_1$ (predefined as closing or reminding user to close the other application that is using the camera)
On-obligation	ObligationID$_2$ (predefined as pausing recording if an incoming call comes)
State	$currentTime - lastForbiddenTime \geqslant 1minute$
Pre-context	$(batterypower \geqslant 30\%)$ $\wedge((\neg\langle[01/01/2010\_00:00:00, 12/31/2010\_23:59:59], weeks+ \{3,4\}.day+10hours \rhd 2hours\rangle) \vee (\neg meetingroom))$
Ongoing-context	$(batterypower \geqslant 30\%)$ $\wedge((\neg\langle[01/01/2010\_00:00:00, 12/31/2010\_23:59:59], weeks+ \{3,4\}.day+10hours \rhd 2hours\rangle) \vee (\neg meetingroom))$
Update	$if(forbidden)lastForbiddenTime = currentTime$

$$\Omega(s,\ o,\ r) \Rightarrow \begin{cases} \varphi_o(o) \in \varphi_s(s), \textit{if } o \textit{ is a resource object} \\ \varphi_c(s) \geqslant_c \varphi_c(o) \wedge \varphi_i(s) \leqslant_c \varphi_i(o), \textit{if } o \textit{ is a data object, and } r = read \\ \varphi_c(s) \leqslant_c \varphi_c(o) \wedge \varphi_i(s) \geqslant_c \varphi_i(o), \textit{if } o \textit{ is a data object, and } r = write \end{cases}$$

Definition 16. *(**Usage Decision**) The **usage decision** determines whether an access should be permitted or an ongoing access should be revoked based on authorizations, obligations, contexts, and states. The usage decision is performed as below:*

– $allow(s,\ o,\ r) \Rightarrow \Omega(s,\ o,\ r) \wedge fulfill(preOb) \wedge fulfill(preContext) \wedge fulfill(stateConstraint)$

– $revoke(s,\ o,\ r) \Leftarrow \neg fulfill(onOb) \vee \neg fulfill(onContext)$

– $update(state)$

4 A Usage Control Framework for Android

Based on the above ConUCON model, we developed a continuous context-aware usage control framework for Android.

4.1 Framework Overview

Figure 1 describes the architecture of the framework. The framework consists of a Policy Enforcement Point (PEP), a Policy Decision Point (PDP), a Policy Information Point (PIP) and a Policy Administration Point (PAP). These components communicate with each other with the messaging mechanism listed in Table 2.

The circled numbers in the figure indicate the processing flow during one usage decision process. When an application tries to access an object, the PEP perceives the request and invokes the PDP using $request(s, o, r)$. Then the PDP performs authorization and activates the PIP with $evaluate(s, o, r)$. The PIP then invokes the Policy Resolver
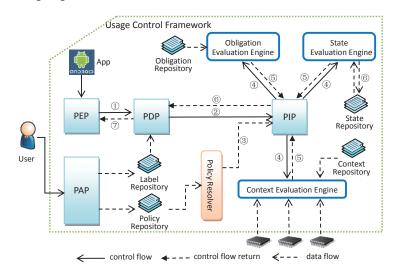
Fig. 1. ConUCON Framework

Table 2. Message types transmitted among components.

Message	Source	Destination	Meaning
$request(s, o, r)$	PEP	PDP	The subject s is requesting to perform the right r on the object o.
$permit(s, o, r)$	PDP	PEP	The $request(s, o, r)$ is permitted.
$deny(s, o, r)$	PDP	PEP	The $request(s, o, r)$ is denied.
$terminate(s, o, r)$	PEP	PDP	The subject s terminates access to the subject o.
$revoke(s, o, r)$	PDP	PEP	Revoke the $request(s, o, r)$.
$evaluate(s, o, r)$	PDP	PIP	Perform obligation, state and context evaluation.
$fulfill(s, o, r)$	PIP	PDP	The obligation, state and context policies are enforced.
$violate(s, o, r)$	PIP	PDP	Not all obligations, state and context policies are enforced.
$withdraw(s, o, r)$	PDP	PIP	Withdraw all continuous evaluations.

to resolve predefined policies related to s and o, and then invokes the Evaluation Engines to check the pre-policies. After that, the PIP sends a result(i.e. a $fulfill(s, o, r)$ or $violate(s, o, r)$ message) to the PDP, which synthesizes the received result and the authorization result to decide whether the access should be permitted or denied, and notifies the PEP of the decision by a $permit(s, o, r)$ or a $deny(s, o, r)$ message. The State Evaluation Engine also updates the states accordingly.

Two cases are not illustrated in Figure 1 for the sake of simplicity. The first case is continuous evaluation. After permitting the access, the Evaluation Engines begin to evaluate ongoing policies continuously. Once a violation is detected, the Engines notify the PIP, which then sends a $violate(s, o, r)$ message to the PDP. And the PDP will send a $revoke(s, o, r)$ message to the PEP immediately to revoke the access at once. The

other case occurs when an application terminates the access. The PEP notifies the PDP by a $terminate(s, o, r)$ message, which then sends a $withdraw(s, o, r)$ message to the PIP to withdraw the continuous evaluation on this session.

Notice that the PDP can send a $revoke(s, o, r)$ message to the PEP on its own initiative before the PEP sends a $terminate(s, o, r)$ message. Similarly, the PIP can send a $violate(s, o, r)$ message to the PDP once the Engines detect a violation, no matter whether the PDP sends an $evaluate(s, o, r)$ message or not. This is an important improvement upon existing access control research, whose usage decision only occurs before the access.

4.2 Framework Components

Policy Enforcement Point (PEP) The PEP takes charge of perceiving access request and termination, invoking the PDP to perform the usage decision and enforcing the usage control according to the PDP's response.

When the PEP captures a request, it invokes the PDP with a $request(s, o, r)$ message. The PEP allows the access only if the PDP responds a $permit(s, o, r)$ message. After permitting the access, the PEP shifts to a listening state. If any of the ongoing policies is violated, the PEP will be noticed by PDP with a $revoke(s, o, r)$ message to terminates the access. In addition, The PEP should perceive the termination of the access. Once a subject terminates the access, the PEP sends a $terminate(s, o, r)$ message to the PDP, and then the PDP stops monitoring policies.

To capture the access requests to all objects, the PEP should be integrated in the relatively low level, the application framework layer of Android platform, for instance. It should also implement specific messaging mechanism to communicate with the PDP.

Policy Decision Point (PDP) The PDP is the component that performs usage decisions. The PDP is responsible for activating the Policy Resolver and the PIP, authorizing (i.e. checking the permission labels), and notifying the PEP of the usage decision result after merging the authorization result and the responding result of the PIP. The PDP is invoked by the PEP when access actions including request and termination occur.

The PDP is invoked by the PEP using a $request(s, o, r)$ message. After being invoked, the PDP retrieves the permission labels of s and o from label repository and performs authorization based on Definition 15. If the result is true, the PDP then invokes the PIP to gather information related with usage decision (i.e. pre-policy evaluation result). If any policy is violated, the PDP responds a $deny(s, o, r)$ message to the PEP to deny the access. Otherwise, it returns the PEP a $permit(s, o, r)$ message, and listens on both the PEP and the PIP to process the $violate(s, o, r)$ from the PIP and $terminate(s, o, r)$ from the PEP.

Policy Information Point (PIP) The PIP is the component that provides the PDP with evaluation information on obligation, state and context both before (pre) and during (ongoing) the access, with the aid of the Obligation Evaluation Engine, the State Evaluation Engine and the Context Evaluation Engine.

The PIP is invoked by the PDP with an $evaluate(s, o, r)$ message. The PIP then calls Policy Resolver to resolve policies that contain pre-policies and ongoing-policies.

After that, the PIP invokes Evaluation Engines to evaluate pre-policies at first. If any engines returns false, the PIP responds a $violate(s, o, r)$ to the PDP. Otherwise, the PIP returns $fulfill(s, o, r)$, then invokes Evaluation Engines to fork daemons to evaluate ongoing polices continuously. If any of ongoing policies is violated, the PIP notifies the PDP of a $violate(s, o, r)$ message. The PIP also listens on the PDP after the pre-policies evaluation. When it receives a $withdraw(s, o, r)$ message from the PDP, it withdraws the ongoing evaluation.

Evaluation Engines The Evaluation Engines are invoked by the PIP to perform corresponding policy evaluation for obligation, state and context.

The Obligation Evaluation Engine monitors the execution of obligations. If the obligation is an action that can be carried out by the Engine directly, the Engine can require the subject to perform the obligation or carry out directly. Recall Example 8. The Engine can ask for the application to remind the user to close another application that is using the video recorder, or just close it directly. If the obligation can only be observed, the Engine does not return true until the obligation is observed. The obligations defined by Definition 6 are stored in the Obligation Repository, which can be accessed using their obligation IDs or paths specified by the user.

The State Evaluation Engine is invoked to evaluate the state constraints. It first resolves the attribute type from the state constraint expressions, and retrieves the corresponding attribute values from the State Repository. Then it evaluates whether the constraint is satisfied, and notifies the PIP of the evaluation result. Besides, the State Evaluation Engine will update the state values into the state repository if needed.

The Context Evaluation Engine evaluates the context policies and monitors the change of the context. Similarly with the State Evaluation Engine, it first resolves the context types from context constraint expressions. Then it interacts with underlying systems and sensors to retrieve context value such as the coordinate, CPU utilization and battery power.

Policy Administration Point (PAP) The PAP is a component that interacts with the user, which allows the user to administrate the usage policies for the data and resources in his smart phone. The user can impose or deprive the permission labels, confidentiality labels and integrity labels to a subject or an object as Definition 5.

The PAP then formats the user's policy specification and stores the policies into the Label Repository and Policy Repository, respectively. The policies are formatted in XML, which will be discussed in Section 4.3.

4.3 Policy Specification

Through the PAP, the user specifies his usage policies on his data and resources according to Definition 13 and 14. In order to facilitate policy storage and transmission among the components, the policies are represented in an XML format. The primary tags used are listed as follows.

– `<Subject>`, `<Object>` and `<Right>` specify the subject, object and right associated with the policy.

– <Obligation> tag specifies an obligation. The *ObligationTime* specifies whether the obligation must be performed before (pre) or during (ongoing) the access, while the *ObligationID* specifies the ID of the obligation, the Obligation Evaluation Engine retrieves the action stored in Obligation Repository using this ID. The user can specify a new obligation by assigning the action path to the *ObligationID*, and use several < Parameter > tags to specify the parameters to execute the action.

– <State> tag specifies the state constraint in the policy. The <Attribute> tag indicates the attribute in this state constraint, while the attribute *Owner* indicates owner of this attribute and the *Type* is the attribute type. The <Expression> tag specifies the logic expression expected, which is defined in Definition 14.

– <Context> tag specifies the context constraint in the policy. The meaning of *ContextTime* is similar with *ObligationTime*. The context consists of several < ContextComposition > tags, which are connected with "∧". Each < ContextComposition > consists of several < Factor > tags, connected with the *Operator*. The <Factor> is a context predicate defined in Definition 14, the *Type* specifies the context type defined in Definition 7.

– <Update> tag specifies an update policy. The *UpdateTime* declares the time to perform this update, which is in {Allow, Deny, Ongoing, Post}. The <Attribute> tag indicates the attribute to be modified. It is stored in State Repository and identified by *Name*, while the default value of the attribute is *Default*. An <Expression> tag specifies an assignment expression that is executed to update the state.

Figure 2 illustrates the XML representation of Example 8. The root node <Policies> contains all the policies. It includes several <Policy> tags, each indicates a user-specified policy defined in Definition 14.


```
<Policies>
  <Policy Effect="Permit">
    <Subject>All</Subject>
    <Object>Camera</Object>
    <Right>Use</Right>
    <Obligations>
      <Obligation ObligationTime="Previous" ObligationID =
          "com:android:conUcon:obligationID1"></Obligation>
      <Obligation ObligationTime="Ongoing" ObligationID=
          "com:android:conUcon:obligationID2"></Obligation>
    </Obligations>
    <States>
      <State>
        <Attribute Owner="Camera"
            Type = "lastForbiddenTime"></Attribute>
        <Expression>System.currentTime-
            Camera.lastForbiddenTime>=1</Expression>
      </State>
    </States>
    <Contexts>
      <Context ContextTime="Previous">
        <ContextComposition Operator="~">
          <Fator Type="Temporal">[01/01/2010_00:00:00,
              12/31/2010_23:59:59],  weeks + {3, 4}.day + 10 hours ->
              2hours</Fator>
          <Fator Type="Spatial">Meeting Room</Fator>
        </ContextComposition>
        <ContextComposition>
          <Fator Type="BatteryPower">batteryPower >= 30%</Fator>
        </ContextComposition>
      </Context>
      <Context ContextTime="Ongoing">...</Context>
    </Contexts>
    <Updates>
      <Update UpdateTime="Deny">
        <Attribute Owner="Camera" Name = "lastForbiddenTime"
            Default="01/01/1900_00:00:00"></Attribute>
        <Expression>Camera.lastForbiddenTime
            =System.currentTime</Expression>
      </Update>
    </Updates>
  </Policy>
</Policies>
```

Fig. 2. XML representation of Example 8 in Section 3.3

5 Implementation and Evaluation

We have implemented the above framework on Android, which will be described in this section. The framework monitors the accesses to the resources, data and files (i.e. the Objects in ConUCON) performed by the applications and application components (i.e. the Subjects in ConUCON). The identities of subjects and objects, i.e. a subject's UID and an Object's URI, are included in their attribute sets. The attribute sets also contain other information such as the software producer, usage times and attributes defined by the user, like `lastForbiddenTime` in Example 9. The attributes can be retrieved and maintained by the usage decision process. Some frequently used obligations are predefined and hard-coded in the Obligation Evaluation Engine, an interface is also provided to the user to assign a new obligation in the way described in section 4.3. Context types are confined to frequently used property in our implementation, such as temporal, spatial, battery, signal strength, acceleration, Bluetooth state, WiFi state, CPU utilization, and memory amount, which can be easily retrieved in Android.

The framework components are implemented and deployed according to their responsibility. The PEP, PDP and PIP are integrated in the application framework layer on Android. We implement the Policy Resolver as a parser to resolve the `xml` file which stores the policies. The Evaluation Engines are implemented as daemon threads to monitor and evaluate the ongoing policies continuously. The messages described in Section 4.1 can be implemented as procedure calls and inter component communications (ICC). The Repositories are stored in the `\system` directory on Android and are managed using the `Content Provider` component, which provides inherent isolation and protection.

5.1 Usage Decision

The applications in Android retrieve the resources and data using an ICC mechanism. Intent is used to encapsulate the information related to the ICC. An Intent object is passed to `Context.startActivity()`, `Context.startService()`, or other limited number of methods. These methods are implemented by the `ApplicationContext` class, which then transmits control to the `ActivityManagerService` class, where the Intent is resolved to determine the component which will handle it. Then the UID and permission required for accessing the component are used as parameters to call the `checkComponentPermission()`. The `checkPermission()` in `ActivityManagerService`, which is claimed (by the comments in source code) as the only public entry point [22], actually calls the `checkComponentPermission()` to perform permission check. Thus, we hook this function to insert our usage decision `conUconPDP()`.

After performing the existing permission check in the `checkComponentPermission()`, the `conUconPDP()` takes over the control. It first extracts and analyzes the object information from the Intent. If the object is a file object (including the pictures, contacts, and regular files), it retrieves the confidentiality and integrity labels of the subject and object from the label repository and checks the permission. After that, it invokes the `conUconPIP()` to perform evaluation on obligations, states and contexts.

The `conUconPIP()` calls the Policy Resolver to get the policies. For the obligations, the `conUconPIP()` executes the hard-coded instructions according to the `obligationID` or calls the routine specified by the user. For the states, it checks whether the constraint is fulfilled. It also creates representations for the new attributes and maintains them in the State Repository. For contexts, it invokes different managers to get the context information and evaluates it. If the evaluation is passed, it returns true. Meanwhile, if necessary, it may create daemon threads to evaluate the ongoing policies before returning. The daemons periodically check whether the constraints are violated. If a violation is identified, the daemons will terminate the session.

5.2 Policy Specification

An activity `com.android.conUcon.contextDefine` is implemented to provide a usable interface for the user to define his context information such as examples illustrated in Table 3. Besides, we modify the `PackageInstallerActivity` to allow a user to impose his policies on an application at install-time. The existing framework lists the permission that the application requires. We modify this interface to enable the user to set his obligation, state and context constraints on this permission. An activity `com.android.conUcon.policyAdministrator` is implemented to enable the user to specify his policies after installation. This activity lists all the installed applications, all the resources and all the data (i.e. contacts, pictures, files and so on). The user can associate the confidentiality and integrity labels to the subjects and data objects and specify the policy on these applications, resources and data. The specifications are resolved by the activity, which then generates the corresponding data and stores them in the Repositories. The activity even provides an interface for expert users to specify his policies by editing the policy file.

Table 3. A context information stored in the Context Repository

Context Type	Context Name	Context Value
temporal	weekday	$periodic\ expression =$ $''<[0,\infty], weeks + \{1,2,,5\}.days >''$
logic position	my university	$featuretype ='' school'', realposition =$ $''(o = (39.99° N, 116.30° E), r = 1530m)''$
battery power	low power	$''batterypower \leqslant 10\%''$

5.3 Performance Evaluation

Because the usage decision in ConUCON framework performs extra actions to evaluate the obligation, state and context policies, an overhead will be introduced. To evaluate this overhead, we carry out some experiments on the Android emulator to measure the execution time. The actions we choose are frequently used in the daily life.

To keep authenticity, we associate different policies on these actions. For example, The applications starting dialer must perform the obligation to check whether the audio capture is closed. The call duration should be maintained as a state, meanwhile, its

constraint should consider restrictions on location and time. The constraints on all these actions come into three categories: Obligation, State and Context in Table 4. If an experiment with ConUCON considers specific types of constraints in its security policy, the corresponding column are marked as"$\sqrt{}$". The performance contrast between existing mechanism and our framework are illustrated in Table 4. The overhead caused by our usage decision is quite acceptable when we restrict context types within what can be retrieved locally, such as temporal, WiFi and battery power. Other information such as location that needs to be retrieved by querying network or satellite will consume a little longer time (the numbers within parentheses in Table 4). However, if the data and resources are extremely important, it is worthwhile sacrificing a little performance.

Table 4. Performance Comparisons

Actions	Existing mechanism (ms)	ConUCON framework			
		Obligation	State	Context	Time (ms)
sarting WiFi	102.5	$\sqrt{}$	-	$\sqrt{}$	117.3 (195.3)
sending SMS	69.8	-	$\sqrt{}$	$\sqrt{}$	76.0
starting dialer	49.7	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	80.6(150.8)
accessing a contact	95.3	-	$\sqrt{}$	$\sqrt{}$	116.5
accessing a picture	55.8	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	68.5(153.8)

6 Related Work

Some literatures have proposed solutions for enhancing the security on smart phone platforms. Malware detection on smart phone is already widely concerned [8, 9, 29, 32]. Zhang et al. [33] proposed an isolation technique for mobile platform by realizing the TCG's Trusted Mobile Phone specification and by leveraging SELinux which provides a generic domain isolation concept at the kernel level. Schmidt et al. [29] demonstrated how to monitor a smart phone running Symbian and Windows Mobile in order to extract features for anomaly detection.

Access control models play an important role in security mechanisms. Some researchers have extended the RBAC model [28], the most popular access control model nowadays, to include context information in authorization decisions. Damiani et.al proposed GEO-RBAC [13] to support spatial roles. Bertino et al. proposed TRBAC [6] to support temporal roles. Other extensions include GRBAC [20, 11], STARBAC [1] and LRBAC [25]. Context-awareness has attracted much attention in the security issues of mobile platforms as well, some literatures have already focused on context-aware access control in the networks [10, 3, 2].

Android security is also widely concerned in recent researches. Asaf Shabtai et.al analyzed and assessed the security mechanisms incorporated in Android by identifying the threats and potential dangers, as well as solutions in Android platform [30]. SCANDROID [17] is a tool for reasoning automatically about the security applications, which checks whether data flows through an application are consistent with its specifications. Enck et al. [14] proposed Kirin security service for Android, which performs

lightweight certification of applications to mitigate malware at install-time. Apex [22] presents a policy enforcement framework to enable the the user grant permissions in a fine-grained manner and enforces policy user defined at runtime. However, the context information is not taken into consideration in these approaches. Our work refers to the Apex [22] in policy specification and implementation, yet we focus on performing a continuous usage decision including obligations, states and contexts.

7 Conclusion

The existing security mechanism on the Android platform is facing great challenges because of the mobility and openness of mobile computing environment. This paper proposes a context-aware usage control mechanism to enhance data protection and resource usage constraints on Android. We propose a context-aware Usage CONtrol model ConUCON, which is able to take obligations, states and contexts into consideration at usage decisions. Based on ConUCON, we extend the existing security mechanism to implement a policy enforcement framework on Android, which enables the user to grant permissions in a fine-grained manner and to support revocations and modifications on an application's permissions at runtime. We also evaluate our mechanism with some frequently used actions, which shows that the overhead introduced by the proposed scheme is acceptable. We will further study the application of our ConUCON model on other types of mobile platform.

References

1. Subhendu Aich, Shamik Sural, and Arun K. Majumdar. STARBAC: Spatio temporal role based access control. In *OTM Conferences*, 2007.
2. Jalal Al-Muhtadi, Anand Ranganathan, Roy H. Campbell, and M. Dennis Mickunas. Cerberus: A context-aware security scheme for smart spaces. In *PerCom*, page 489, 2003.
3. Matteo Bandinelli, Federica Paganelli, Gianluca Vannuccini, and Dino Giuli. A context-aware security framework for next generation mobile networks. In *MobiSec*. Springer, 2009.
4. D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations. Technical Report ESD-TR-73-278, MITRE Corporation, 1973.
5. Elisa Bertino, Claudio Bettini, Elena Ferrari, and Pierangela Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM Trans. Database Syst.*, 23(3):231–285, 1998.
6. Elisa Bertino, Piero A. Bonatti, and Eiena Ferrari. TRBAC: A temporal role-based access control model. In *RBAC-00*, pages 21–30, N.Y., July 26–27 2000. ACM Press.
7. K. J. Biba. Integrity considerations for secure computer systems. MTR-3153, Rev. 1, The Mitre Corporation, 1977.

8. Abhijit Bose, Xin Hu, Kang G. Shin, and Taejoon Park. Behavioral detection of malware on mobile handsets. In *MobiSys '08*, pages 225–238, New York, USA, 2008. ACM.
9. Jerry Cheng, Starsky H.Y. Wong, Hao Yang, and Songwu Lu. Smartsiren: virus detection and alert for smartphones. In *MobiSys '07*, pages 258–271, New York, USA, 2007. ACM.
10. Michael J. Covington, Prahlad Fogla, Zhiyuan Zhan, and Mustaque Ahamad. A context-aware security architecture for emerging applications. In *ACSAC*, pages 249–260, 2002.
11. Michael J. Covington, Matthew J. Moyer, and Mustaque Ahamad. Generalized role-based access control for securing future applications, November 03 2000.
12. D. Dagon, T. Martin, and T. Starner. Mobile phones as computing devices: the viruses are coming! *Pervasive Computing, IEEE*, 3(4):11 – 15, oct.-dec. 2004.
13. Maria Luisa Damiani, Elisa Bertino, Barbara Catania, and Paolo Perlasca. Geo-rbac: A spatially aware rbac. *ACM Trans. Inf. Syst. Secur.*, 10(1):2, 2007.
14. William Enck, Machigar Ongtang, and Patrick Drew McDaniel. On lightweight mobile phone application certification. In *Proceedings of CCS 2009*, pages 235–245. ACM, 2009.
15. F-Secure. Cabir. `http://www.f-secure.com/v-descs/cabir.shtml`.
16. F-Secure. Pbstealer.A. http://www.f-secure.com/v-descs/pbstealer_a.shtml.
17. Adam P. Fuchs, Avik Chaudhuri, and Jeffrey S. Foster. Scandroid: Automated security certification of android applications.
18. Google. Android. `http://www.android.com,`.
19. Mikko Hypponen. Mobile Malware. `http://www.usenix.org/events/sec07/tech/hypponen.pdf`. USENIX Security Symposium, August 2007. Invited Talk.
20. M.J. Moyer and M. Abamad. Generalized role-based access control. In *Distributed Computing Systems, 2001. 21st International Conference on.*, pages 391 –398, apr 2001.
21. C. Mulliner. Security of Smart Phones. Master's thesis, Department of Computer Science, University of California Santa Barbara, June 2006.
22. Mohammad Nauman, Sohail Khan, Masoom Alam, and Xinwen Zhang. Apex: Extending android permission model and enforcement with user-defined runtime constraints. In *ASI-ACCS 2010, Beijing, China, April 13-16, 2010*. ACM.
23. Jaehong Park and Ravi Sandhu. The UCON$_{ABC}$ usage control model. *ACM Transactions on Information and System Security*, 7(1):128–174, February 2004.
24. Jaehong Park and Ravi S. Sandhu. Towards usage control models: beyond traditional access control. In *SACMAT*, pages 57–64, 2002.
25. Indrakshi Ray, Mahendra Kumar, and Lijun Yu. LRBAC: A location-aware role-based access control model. In *ICISS*, pages 147–161. Springer, 2006.
26. Android reference. Develope Guide. `http://developer.android.com/guide/index.html`.
27. Sandhu and Park. Usage control: A vision for next generation access control. In *MMMACNS*, 2003.
28. Ravi S. Sandhu. Role-based access control. *Advances in Computers*, 46:238–287, 1998.
29. Aubrey-Derrick Schmidt, Frank Peters, Florian Lamour, and Sahin Albayrak. Monitoring smartphones for anomaly detection. In *MOBILWARE '08*. ICST, 2007.
30. Asaf Shabtai, Yuval Fledel, Uri Kanonov, Yuval Elovici, Shlomi Dolev, and Chanan Glezer. Google android: A comprehensive security assessment. *IEEE Security & Privacy*, 2010.
31. J. Stevenne and M. Niezette. An efficient symbolic representation of periodic time. In *Int. Conf. on Information and Knowledge Management, Baltimore*, November 1992.
32. Liang Xie, Xinwen Zhang, A. Chaugule, T. Jaeger, and Sencun Zhu. Designing system-level defenses against cellphone malware. In *SRDS '09.*, pages 83 –90, sept. 2009.
33. Xinwen Zhang, Onur Aciiçmez, and Jean-Pierre Seifert. A trusted mobile phone reference architecture via secure kernel. In *STC*, pages 7–14. ACM, 2007.
34. Xinwen Zhang, Francesco Parisi-Presicce, Ravi Sandhu, and Jaehong Park. Formal model and policy specification of usage control. *TISSEC*, 8(4):351–387, November 2005.