

# UAMDroid: A User Authority Manager Model for the Android Platform

Zemin Liu, Choon-Sung Nam, Dong-Ryeol Shin

School of Electrical and Computer Engineering, Sungkyunkwan University

300 Cheoncheon-dong, Jangan-gu, Suwon, Gyeonggi-do 440-746, Korea

{zeminliu, namgun99, drshin}@skku.edu

**Abstract**— Smartphones are taking over the task of Personal Digital Assistant and becoming pocket computers due to their fully customizable software and personalization in terms of the personal data stored on them. Thus, how to prevent the private and sensitive information from exposing to outside world is necessary to be taken into account. However, current smartphone operating systems support only a single user mode which is suitable for the phone owner to work with. In this paper, we present a user authority manager model namely UAMDroid for the Android platform that allows phone owner to specify what applications the phone borrowers should be prohibited to access or uninstall, as well as whether extra applications should be allowed to install. We detail our model that extends the existing Android application framework and describe its prototype of implementation. The performance evaluation shows that UAMDroid only produces a minimal impact on the normal usage of smartphones which is tolerable. Our proposal is lightweight and can efficiently secure personal data in multi-user case.

**Keywords**— Smartphone; Android; security; multi-user; authority manager

## I. INTRODUCTION

In recent years, smartphones have been widely used due to their capabilities of allowing the end user to install and run many advanced applications based on a specific platform. Android is an open source platform introduced by Google, along with the Open Handset Alliance [1] formed by major hardware and software developers such as Intel, HTC, ARM, and Motorola. It is a software stack for mobile devices that consists of an operating system, middleware, and a set of core applications such as contacts, web browser, dialer and alarm clock. Android has a large community of developers writing application programs that extend the functionality of the mobile devices. Third party developers are allowed to submit their own applications to Android Market [2] for other users to download and install. A report from AndroLib [3] shows that there are over 100,000 applications currently available in Google's Android Market, which makes it the most popular open source mobile development environment [4].

Developers write managed Java code based on Google-developed application framework to provide additional functionalities, such as GPS, online video and maps, which may require building a data connection network, accessing the

media store, or exposing contact. While giving the device owner a higher level of user experience, it also brings a serious security concern of protecting privacy and sensitive data when sharing phone with others.

Consider the following example. Many people lend their mobile phones to someone else for various reasons. However in most cases, the mobile owner wants to keep his/her personal information (e.g., pictures) private or prohibit the borrower from accessing specific functions such as web browsing, installing malicious applications or uninstalling critical applications. Thus, the device owner needs to control accessing authority of all installed applications. However originally, smartphone OSes only support a single user mode. They do not provide such function of user authority management. Although smartphone OSes such as Android OS, iOS and Windows Mobile are ported from multi-user desktop OSes such as Linux, Mac OS and Windows, respectively, they have no concept of user home directory for its file storage, and thus traditional multi-user management is not available on the smartphone OSes.

To address this problem, we propose a User Authority Manager model namely UAMDroid that extends the existing application framework released by Google for the Android platform. It resides on top of the Android security model that performs user authority checking. UAMDroid classifies the smartphone users into owner and normal users, and gives the owner the highest authority to manage phone resources usage. Our contributions in this paper are as follows: (1) we describe the extensions to the existing application framework of Android platform; (2) we implement a prototype of UAMDroid on Android emulator; (3) we evaluate the system performance on the application execution.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 presents background. Section 4 details our prototype of UAMDroid on Android platform including architecture, implementation and evaluation. Section 5 concludes the paper.

## II. RELATED WORK

Some work has been done on smartphone application access control. But most of the approaches has centered around validating permission assignment at applications installed on local smartphone. Apex [5]-[6] is a resource usage

control model for Android. It provides a comprehensive policy enforcement mechanism extending the existing Android permission framework that allows users to specify detailed runtime constraints to restrict the use of sensitive resources by applications, whereas our user authority manager model helps the phone owner to limit the use of sensitive resources by other users. The policy mechanism of Apex checks the application permission at run-time to prevent user information from being stolen by malicious software, but it does not support checking user permission when the smartphone has multi-users.

Basically, mobile phones' primary function is telephony, which is the reason that smartphone OSes only support a single user who is the smartphone owner. In [7], AT&T provides a service called Smart Limits for Wireless that manages monthly usage limits, time-of-day restrictions and content filters to limit phone resources usage for children. However, users must pay for this service and it only affects the AT&T services. Xudong *et al.* [8] applied a differentiated user access control model named DiffUser for Android operating system, which is aimed at supporting multi-users. DiffUser classifies smartphone users into several classes - administrator, normal users, guests, and other configured users - based on certain set of user access privileges. When a different user is login the system, the Home screen information is reloaded automatically that makes different users in different classes see different available applications. This work is similar to our user authority manager model UAMDroid, except that: (1) UAMDroid has no incorporation with the Android's build-in permission-based security model. Instead, it runs on top of it; (2) the Home Activity has no need to reload the application information to make some application invisible, which could mislead the user to install an application intentionally or unintentionally if that application is the one that the owner does not want to share; (3) if necessary, the mobile phone owner can prohibits the non-host users from installing or uninstalling applications, or even execute the installation program, which improves the system security when sharing phones.

### III. BACKGROUND

#### A. Android Architecture

The Android operating system software stack consists of several layers which are application layer, application framework layer, Android runtime and libraries, and Linux kernel [9]. At the core of the Android platform is Linux kernel version 2.6.29, responsible for device drivers, resource access, power management, and other OS duties. Sitting at the next level, on top of the kernel, are a number of C/C++ libraries such as Secure Socket Layer (SSL), WebKit, etc. Most of the application framework accesses these core libraries through the Dalvik Virtual Machine (DVM), the gateway to the Android platform. The Android Java applications runs on a Java based object oriented application framework which includes activity manager, package manager, window manager, etc.

In Android, there are four types of essential components namely activities, services, broadcast receivers, and content providers that an application are composed of [10]. Android

Java application marks one activity as the first one to be presented to the user when the application is launched, instead of having one single entry point for everything in the application, such as *main()* method. Activities provide visible user interface of the application. Services run in the background for an indefinite period of time rather than having a visual user interface. To communicate with an ongoing service, an interface that the service exposes must be used. The broadcast receiver component receives and reacts to broadcast message that generated by applications. Broadcast receiver may use the *Notification Manager* to alert the user in various ways such as vibrating the device or playing a sound. Finally, Android allows applications to store and retrieve data in different types of data stores (such as SQLite database) through a REST (REpresentational State Transfer)-like abstraction called content provider. The application framework layer enables the use of those low-level components, and provides programming API to the application developers.

Intents are asynchronous messages that activate three components - activities, services, and broadcast receivers [11]. An intent encapsulates the action as well as the extra information associated with the call to be performed. Intent messaging is used for late run-time binding between components in the same or different applications. Every application in Android runs in its own Linux process which has its own virtual machine (VM). Components of an application can interact with other component within the application based on intents, or outside the application itself via a specialized inter-process communication (IPC) mechanism [12].

#### B. Problem Description

Nowadays, thanks to their fully customizable software and extensible functionalities such as web browsing and online games, smartphones are getting more and more popular in the world. With such attractive features, it also brings some serious security concerns. How can the smartphone owner manage all installed applications to keep his/her personal information safe in all possible use cases?

There are several issues that this paper takes into account: (1) the borrower is not allowed to access some certain applications; (2) the borrower is not allowed to uninstall some certain applications; (3) the owner can give the borrower the authority to install applications, otherwise such action is prohibited.

We address these issues by proposing a user authority manager model for Android platform that enables the phone owner to limit resources usage and manage borrowers' use of phone.

### IV. ANDROID USER AUTHORITY MANAGER MODEL

In this section, we present our user authority manager (UAMDroid) model on Android platform based on the problems described in the previous sections. The architecture of our prototype is shown in Figure 1.

In Android, applications are composed of components that

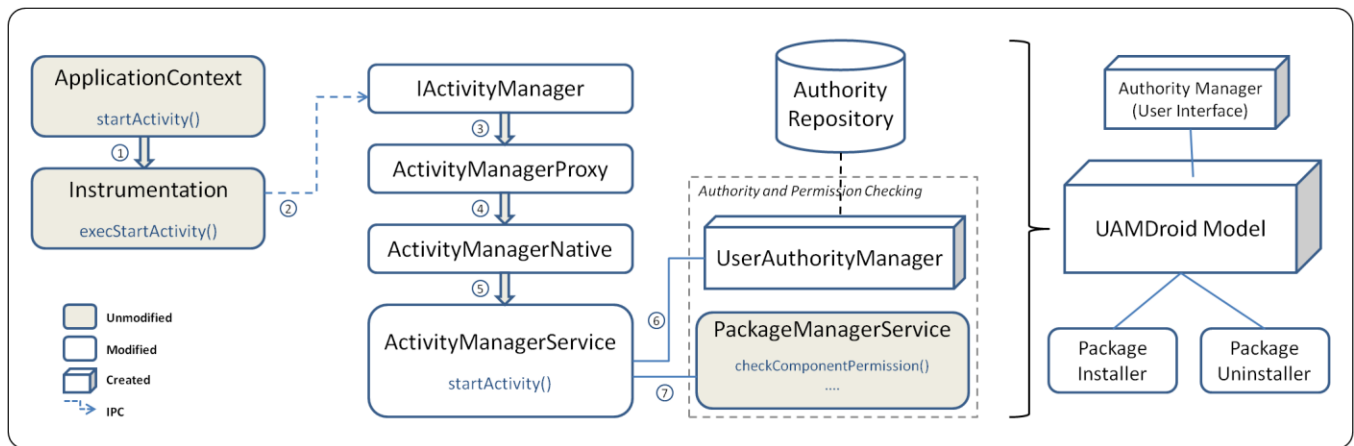


Figure 1. UAMDroid Architecture

can be executed on their own when the need arises. Activities provide not only the visible interface the user can undertake, but also the entry point for the application to start. Before Android can start an application component, it much checks whether the component exists, by reading the *AndroidManifest.xml* file that applications use to declare their components and identify permissions. In manifest file, the intent filter informs Android what kind of intent the component is able to handle. The activity with the action of *android.intent.action.MAIN* and category of *android.intent.category.LAUNCHER* is marked to start when the application is launched.

An intent object is put together with the required information and passed to the *startActivity()* method of the *ApplicationContext* class, when the activity is being started. It then pass the intent to the *execStartActivity()* method of the *Instrumentation* class, which is instantiated before any of the application code and monitors all of the interaction between the system and applications.

Two independent processes are able to talk, by using the specialized inter-process communication (IPC) mechanism supported by Android. The *ActivityManagerNative* and *ActivityManagerProxy* both use the concept of *Binders* and *Parcels*, which are the workhouse of Android IPC. *Binder* implements the *IBinder* interface that provides a simple functionality of synchronous method invocation. Calling the *transact()* method on the binder is able to transmit a *java.os.Parcel* instance carrying IPC messages between the client and the implementation. The *ActivityManagerProxy* creates a parcel with information about the targeted intent, and sends it to the *ActivityManagerNative*. Then the arguments are extracted from the parcel and passed to the *startActivity()* method of the *ActivityManagerService*. The *ActivityManagerService* is in charge of managing activities. Once the first component such as activity of an application is successfully executed, the *ActivityManagerService* creates a process for that application with its package name as the process name by default. The *IActivityManager* that the *ActivityManagerNative* and *ActivityManagerService* implement is exposed to clients. The *Instrumentation*

therefore can pass the intent to the *ActivityManagerService* by simply invoking a method via the IPC mechanism.

For managing the execution of an application, we have modified the *ActivityManagerService* class and placed a hook in the *startActivity()* method to our *UserAuthorityManager*. The *UserAuthorityManager* first check whether the action and category of the upcoming intent is *ACTION\_MAIN* and *CATEGORY\_LAUNCHER*, respectively. If matched, the authority resolver is invoked; otherwise the *UserAuthorityManager* won't do anything but let the *ActivityManagerService* continue its work. The authority resolver first resolves the current user authority and the target applications' accessing authority, and then draws a comparison. If the former has lower authority than the latter does, the *UserAuthorityManager* would propagate the result indicating that start permission is denied to the application layer via the Android IPC mechanism.

Our *UserAuthorityManager* has an Authority Repository that contains related information about the user authority and accessing authority of all installed applications. By using a lightweight and efficient XML serializer *FastXmlSerializer* that Android provides and a parser based on *XmlPullParser* [13], we store all information of each application as well as the user authority as an XML file in *UserAuthorityDir* created in *SystemDir* of the Android filesystem. Each XML file uses the package name of the application it related to as its file name, while the file named *UserAuthority* specifies the users' authority. Figure 2 shows the XML representation of the browsers' accessing authority. There are basically four attributes defined for it: the first one specifies the name of the application; the second one presents the accessing authority which has two levels – *Administrator* and *Normal*; the *category* tells whether the application is system specified; and

```
<UserAuthority
  name="com.android.browser"
  authority="Normal"
  category="User_App"
  state="Protected" />
```

Figure 2. XML Representation Sample of Browser

the *state*, which is classified into two groups – *Protected* and *Public* – indicates whether the application is allowed to be modified, such as uninstall.

For interacting with user interface, we have modified the *ActivityManager* interface, and thus the *ActivityManagerNative*, *ActivityManagerProxy*, and *ActivityManagerService* class that implement it. We have added several methods (e.g. *getUserAuthority()*, *getAppState()*, *updateAppAuthority()*, etc) into the *IActivityManager* interface, so that the end user with the highest authority can simply call to communicate with our *UserAuthorityManager* class, and obtain a specific applications' accessing authority or modify it directly.

We have also modified the Android application installer and uninstaller by putting a hook into the *installer* and *uninstaller* class. When the user tries to uninstall an application, the *uninstaller* checks its *State* value stored in Authority Repository. If the application is signed as *Protected*, the uninstallation would be terminated. Likewise, if the installation program is marked as *Protected*, then any installation would be failed.

The UAMDroid is running on top of the existing Android permission-based security mechanism. It performs accessing authority checking at the beginning of starting an application. Once it confirms that the application is not limited by user authority, the transaction is back to the *ActivityManagerService* and continued such permission checking. Otherwise, the starting application is forced to stop automatically without performing any such permission checking.

#### A. Implementation of UAMDroid

We present UAMDroid implementation in this section.

The development environment is Eclipse 3.5 with Google's Android Developer Tools (ADT) plugin on Linux operating system Ubuntu 10.04. All tests have been carried out on the Android emulator. The debug tools are Dalvik Debug Monitor Service (DDMS), which connects the IDE to the applications running on the device or Android emulator, and Android Debug Bridge (adb), which manages the state of an Android emulator instance or smartphone. The source code for our user authority manager model is written in Java with Android 2.1-update1.

Screenshots of the end user interface are shown in Figure 3. The phone user authority is set to *Normal* as default value, when the smartphone is powered on. Under this mode, the user only can access those applications whose accessing authority is *Normal*. If he/she tries to execute a *non-Normal* application, an alert window would be popped up, and thus the execution would be failed. To enter the *Administrator* mode, the user can press the *Login* item in the home Menu list, and input the correct user name and password. Under the *Administrator* mode, the user gains full authority to play with all installed applications on the device. Simultaneously, the *Login* item on home Menu is changed into *Logout*, *User Setting*, and *Authority Manager* items. The *User Setting* gives user the right to change his/her login name and password. The *Authority Manager* opens a user interaction interface showing

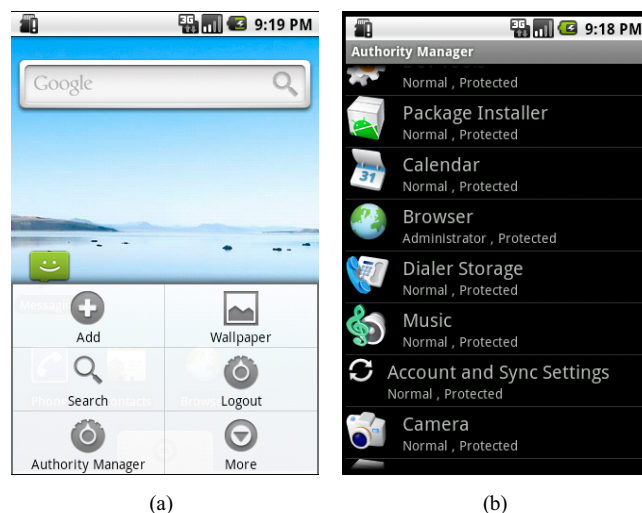


Figure 3. User Interface: (a) Home Menu after login; (b) Authority Manager

all installed applications' icon, name, current accessing authority, and current state. The user is not allowed to uninstall an application, if the *State* value of the target application is set as *Protected*. Through the authority manager, the phone owner can even prohibit the borrower from executing the Android installation or uninstallation program.

#### B. Performance Evaluation

Unlike traditional desktop computers, smartphone devices are mobile and handheld that carries limited resources and lacks of computational capability and power. Moreover, the phone users are unwilling to waste time on opening an application. In particular, they cannot wait for using some specific functions, such as an emergency call. In this case, performance cannot be sacrificed for extended functionalities or supports, such as security. Our user authority manager model is therefore necessary to make the impact on the starting time of applications as small as possible.

To evaluate our system performance, we have taken the Dialer, Browser, Music, and Message applications as test samples. We have recorded the starting time taken in the existing Android application framework, and in UAMDroid, respectively. The results are shown in Figure 4. The starting mean time in UAMDroid is 254ms with a minimum of 245ms and a maximum of 262ms, which is not much higher than that

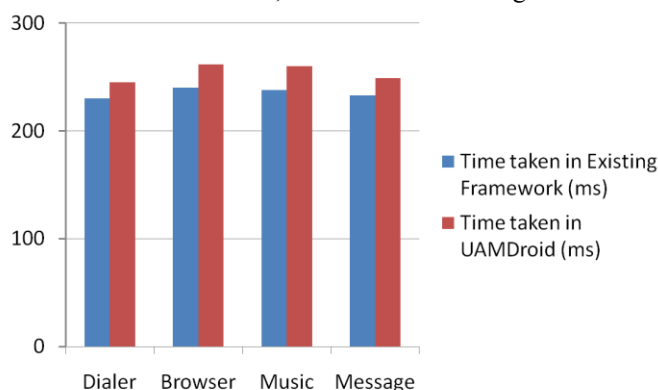


Figure 4. Mean time taken for starting the sample applications

**TABLE 1.** PERFORMANCE EVALUATION RESULT

| Application | Mean time taken for authority checking (ms) |
|-------------|---------------------------------------------|
| Dialer      | 15                                          |
| Browser     | 22                                          |
| Music       | 22                                          |
| Message     | 16                                          |
| Overall     | 19                                          |

in the existing framework. The mean time taken for authority checking of the sample applications are given in Table 1. The overall mean time is approximately 19ms. This range of increment is tolerable. The UAMdroid system thus has a minimal impact on the usage of smartphones.

## V. CONCLUSIONS

In these days, smartphones play a role as handheld personal computers that attract widely attentions due to its powerful and fully customizable functionalities. People can access the Internet, play high quality games, read emails, and watch TV on the small device they carry every day. As the mobile phones are becoming more and more human-based, we start worrying about the private personal information stored on the device. What if someone else uses our phone?

In this paper, we have described UAMdroid – a user authority manager model that extends the existing Android application framework. In multi-user case, UAMdroid controls user accessing applications by assigning each application a specific authority. Only the user with a higher authority can start an application. UAMdroid allows the phone owner who has the highest authority to specify an application's accessing authority and related information, so as to keep his/her sensitive resources private and safe. We have also presented UAMdroid implementation including a

user interface that interacts with the end user and our user authority manager. Finally, the evaluation result shows that our model is lightweight and efficient that has a minimal impact on the system performance when incorporating with the Android framework.

## ACKNOWLEDGMENT

This work is financially supported by the Ministry of Education, Science and Technology (MEST), the Ministry of Knowledge Economy (MKE) through the fostering project of HUNIC.

## REFERENCES

- [1] Open Handset Alliance Home Page. [Online]. Available: <http://www.openhandsetalliance.com/>
- [2] Google Android Market. [Online]. Available: <http://www.android.com/market.html>
- [3] AndroLib Home Page. [Online]. Available: <http://www.androlib.com/>
- [4] David Murphy, Extrapolating the Apple-Android Showdown: Who's Right? [Online]. Available: <http://www.pcmag.com/article2/0,2817,2366624,00.asp>, Aug. 2010.
- [5] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constrants," in ASIAN ACM, 2010.
- [6] M. Nauman, and S. Khan. "Design and Implementation of a Fine-grained Resource Usage Model for the Android Platform," in IJAIT, 2010.
- [7] AT&T Smart Limits for Wireless. [Online]. Available: <http://www.wireless.att.com/learn/articles-resources>
- [8] X. Ni, Z. Yang, X. Bai, A. C. Champion, and D. Xuan, "DiffUser: Differentiated User Access Control on Smartphones," in Mobile Adhoc and Sensor Systems, 2009.
- [9] Google, What is Android? [Online]. Available: <http://developer.android.com/guide/basics/what-is-android.html>
- [10] Google, Application Fundamentals. [Online]. Available: <http://developer.android.com/guide/topics/fundamentals.html>
- [11] Google, Intents and Intent Filters. [Online]. Available: <http://developer.android.com/guide/topics/intents/intents-filters.html>
- [12] Google, Designing a Remote Interface Using AIDL. [Online]. Available: <http://developer.android.com/guide/developing/tools/aidl.html>
- [13] S. Haustein, and A. Slominski, XML Pull Parsing. [Online]. Available: <http://www.xmlpull.org/>