NATIONAL CONFERENCE ON EMERGING TRENDS INTELLIGENT COMPUTING AND COMMUNICATION
APRIL 13-14, 2012

ANDROID SECURITY MODEL PROVIDES WITH BASE OF OPERATING SYSTEM

Ahmad Talha Siddiqui
Assistant Professor (CS)
School of Engineering & Technology
Noida International University
ahmadtalha2007@gmail.com

Jawed Ahmad
Assistant Professor
Department of Computer Science
Jamia Hamdard University
jawed2047@rediffmail.com

Shoeb Ahad Siddiqui
M.Tech (Computer Science)
Department of Computer Science
Jamia Hamdard University
siddiqui.shoeb3@gmail.com

Abstract
Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The android provides the tools and APIs necessary to begin developing applications on the Android platform using programming language. Android is a widely anticipated open source operating system for mobile devices that provides a base operating system, an application middleware layer, a Java software development kit (SDK), and a collection of system applications. Android has a unique security model, which focuses on putting the user in control of the device. Android devices however, don't all come from one place, the open nature of the platform allows for proprietary extensions and changes. This paper we should already be familiar with Android's basic architecture and major abstractions including: Intents, Activities, Broadcast Receivers, Services, Content Providers and Binder. As android is open source we should also have this code available to us. Both the java and C code is critical for understanding how Android works, and is far more detailed than any of the platform documentation.

Keywords: Android, Android Tools, API, System Application, Security Model and Architecture of Android.

1. INTRODUCTION

THE next generation of open operating systems won't be on desktops or mainframes but on the small mobile devices we carry every day. The openness of these new environments will lead to new applications and markets and will enable greater integration with existing online services. However, as the importance of the data and services our cell phones support increases, so too do the opportunities for vulnerability. It's essential that this next generation of platforms provide a comprehensive and usable security infrastructure. Developed by the Open Handset Alliance (*visibly led by Google*), Android is a widely anticipated open source operating system for mobile devices that provides a base operating system, an application middleware layer, a Java software development kit (SDK), and a collection of system applications. Although the Android SDK has been available since late *2007*, the first publicly available Android-ready "G1" phone debuted in late *October 2008*.
Since then, Android's growth has been phenomenal: T-Mobile's G1 manufacturer HTC estimates shipment volumes of more than 1 million phones by the end of *2008*, and industry insiders expect public

ISBN: 9788175157262

adoption to increase steeply in *2009*. Many other cell phone providers have either promised or plan to support it in the near future. A large community of developers has organized around Android, and many new products and applications are now available for it. One of Android's chief selling points is that it lets developers seamlessly extend online services to phones. The most visible example of this feature is—unsurprisingly—the tight integration of Google's Gmail, Calendar, and Contacts Web applications with system utilities. Android users simply supply a username and password, and their phones automatically synchronize with Google services. Other vendors are rapidly adapting their existing instant messaging, social networks, and gaming services to Android, and many enterprises are looking for ways to integrate their own internal operations (such as inventory management, purchasing, receiving, and so forth) into it as well.

2. ANDROID SECURITY MODEL

Android is a Linux platform programmed with Java and enhanced with its own security mechanisms tuned for a mobile environment3. Android combines OS features like efficient shared memory, pre-emptive multi-tasking, Unix user identifiers (UIDs) and file permissions with the type safe Java language and its familiar class library. The resulting security model is much more like a multi-user server than the sandbox found on the J2ME or Blackberry platforms. Unlike in a desktop computer environment where a user's applications all run as the same UID, Android applications are individually soiled from each other. Android applications run in separate processes under distinct UIDs each with distinct *permissions*. Programs can typically neither read nor-write each other's data or code, and sharing data between applications must be done explicitly. Mobile platforms are growing in importance, and have complex requirements including regulatory compliance6. Android supports building applications that use phone features while protecting users by minimizing the consequences of bugs and malicious software. Android's process isolation obviates the need for complicated policy configuration files for sandboxes.

Android permissions are rights given to applications to allow them to do things like take pictures, use the GPS or make phone calls. When installed, applications are given a unique UID, and the application will always run as that UID on that particular device. The UID of an application is used to protect its data and developers need to be explicit about sharing data with other applications7. Applications can entertain users with graphics, play music, and launch other programs without special permissions.

2.1 ANDROID APPLICATIONS

The Android application framework forces a structure on developers. It doesn't have a main() function or single entry point for execution—instead, developers must design applications in terms of components.

The user then uses the Friend Viewer application to retrieve the stored geographic coordinates and view friends on a map. Both applications contain multiple components for performing their respective tasks; the components themselves are classified by their component types. An Android developer chooses from predefined component types depending on the component's purpose (such as interfacing with a user or storing data).

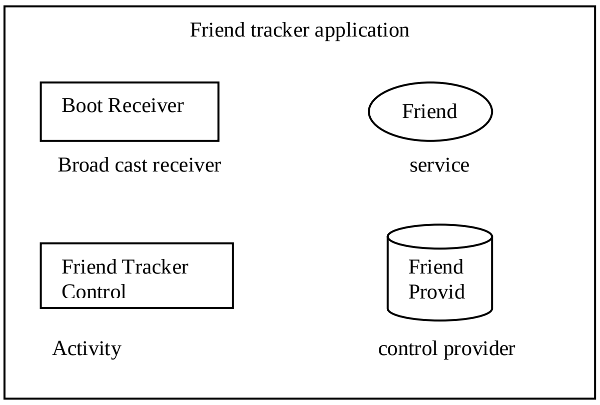
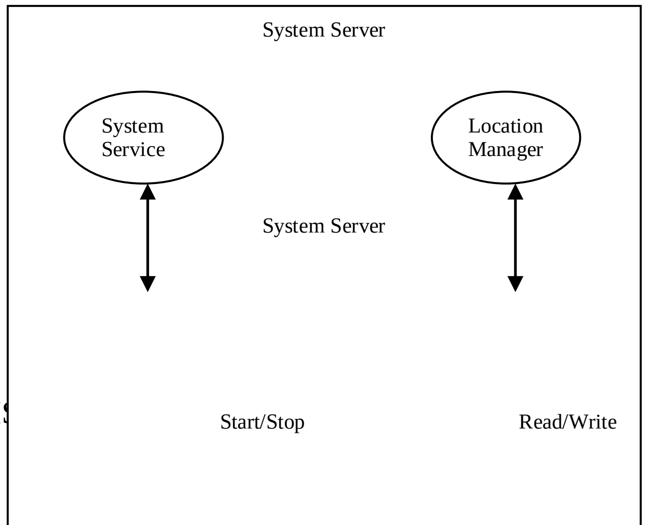


Fig-1: Android Application

2.2 ACTIVITY AND INTERACTION OF COMPONENTS

An application developer defines one activity per “screen.” Activities start each other, possibly passing and returning values. Only one activity on the system has keyboard and processing focus at a time; all others are suspended. In Figure 2, the interaction between components in the FriendTracker and FriendViewer applications and with components in applications defined as part of the base Android distribution. In each case, one component initiates communication with another. For simplicity, we call this inter-component communication (ICC). In many ways, ICC is analogous to inter-process communication (IPC) in Unix-based systems. To the developer, ICC functions identically regardless of whether the target is in the same or different application, with the exception of the security rules defined later in this article. The available ICC actions depend on the target component. Each component type supports interaction specific to its type for example, when FriendViewer starts FriendMap, the FriendMap activity appears on the screen. Service components support start, stop, and bind actions, so the FriendTrackerControl activity, for instance, can start and stop the FriendTracker service that runs in the background. The bind action establishes a connection between components, allowing the initiator to execute RPCs defined by the service. In our example, FriendTracker binds to the location manager in the system server.



IS

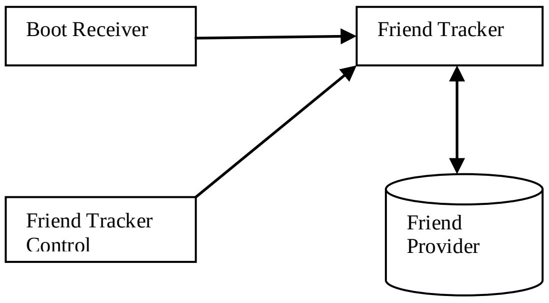
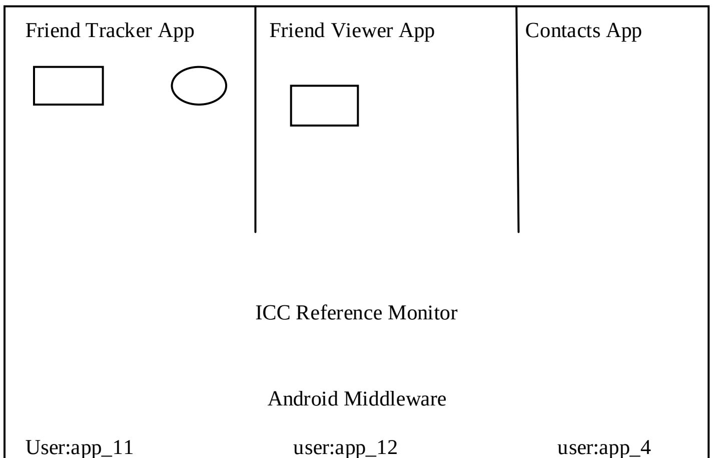


Fig-2: Component Interaction

Once bound, FriendTracker invokes methods to register a callback that provides updates on the phone’s location. Note that if a service is currently bound, an explicit “stop” action won’t terminate the service until all bound connections are released. Broadcast receiver and content provider components have unique forms of interaction. ICC targeted at a broadcast receiver occurs as an intent sent (broadcast) either explicitly to the component or, more commonly, to an action string the component subscribes to. For example, FriendReceiver subscribes to the developer-defined “FRIEND_NEAR” action string. FriendTracker broadcasts an intent to this action string when it determines that the phone is near a friend; the system then starts FriendReceiver and displays a message to the user. Content providers don’t use intents—rather, they’re ad-dressed via an authority string embedded in a special content indicates a table in the content provider, and <id> optionally specifies a record in that table. Components use this URI to perform a SQL query on a content provider, optionally including WHERE conditions via the query API.



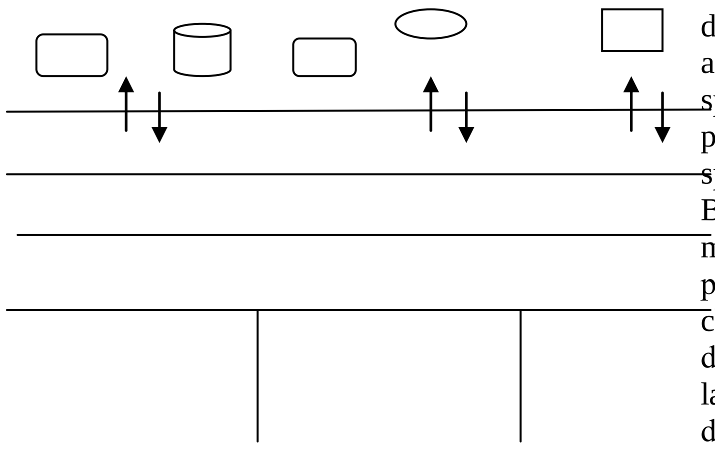


Fig-3: Protection

As Figure 3, Android protects applications and data through a combination of two enforcement mechanisms, one at the system level and the other at the ICC level. ICC mediation defines the core security framework and is this article's focus, but it builds on the guarantees provided by the underlying Linux system. In the general case, each application runs as a unique user identity, which lets Android limit the potential damage of programming flaws. A reference monitor provides mandatory access control (MAC) enforcement of how applications access components. In its simplest form, access to each component is restricted by assigning it an access permission label; this text string need not be unique. Developers assign applications collections of permission labels. When a component initiates ICC, the reference monitor looks at the permission labels assigned to its containing application and—if the target component's access permission label is in that collection—allows ICC establishment to proceed. If the label isn't in the collection, establishment is denied even if the components are in the same application.

3. ACCESS PERMISSION LOGIC

The developer assigns permission labels via the XML manifest file that accompanies every application package. In doing so, the developer defines the application's security policy—that is, assigning permission labels to an application specifies its protection domain, whereas assigning permissions to the components in an application specifies an access policy to protect its resources. Because Android's policy enforcement is mandatory, as opposed to discretionary, all permission labels are set at install time and can't change until the application is reinstalled. How-ever, despite its MAC properties, Android's permission label model only restricts access to components and doesn't currently provide information flow guarantees, such as in domain type enforcement. Security Refinements Android's security framework is based on the label-oriented ICC mediation described thus far, but our description is incomplete.

4. SECURITY RESPONSIBILITIES FOR DEVELOPERS

Developers writing for Android need to consider how their code will keep users safe as well as how to deal with constrained memory, processing and battery power. Developers must protect any data users input into the device with their application, and not allow malware to access the application's special permissions or privileges. How to achieve this is partly related to which features of the platform an application uses, as well as any extensions to the platform an Android distribution has made. One of the trickiest big-picture things to understand about Android is that every application runs with a different UID. Typically on a desktop every user has a single UID and running any application launches runs that program as the users UID. On Android the system gives every application, rather than every person, its own UID. Android requires developers to sign their code. Android code signing usually uses self-signed certificates, which developers can generate without anyone else's assistance or permission. One reason for code signing is to allow developers to update

ISBN: 9788175157262

their application without creating complicated interfaces and permissions. Applications signed with the same key (and therefore by the same developer) can ask to run with the same UID. This allows developers to upgrade or patch their software easily, including copying data from existing versions.

5. ANDROID PERMISSIONS REVIEW

Normal	Permissions for application features whose consequences are minor like VIBRATE which lets applications vibrate the device. Suitable for granting rights not generally of keen interest to users, users can review but may not be explicitly warned.
Dangerous	Permissions like WRITE_SETTINGS or SEND_SMS are dangerous as they could be used to reconfigure the device or incur tolls. Use this level to mark permissions users will be interested in or potentially surprised by. Android will warn users about the need for these permissions on install.
Signature	These permissions can only be granted to other applications signed with the same key as this program. This allows secure coordination without publishing a public interface.
Signature Or System	Similar to Signature except that programs on the system image also qualify for access. This allows programs on custom Android systems to also get the permission. This protection is to help integrate system builds and won't typically be needed by developers.

Applications need approval to do things their owner might object to, like sending SMS messages, using the camera or accessing the owner's contact database. Android uses manifest permissions to track what the user allows applications to do. An application's permission needs are expressed in its AndroidManifest.xml and the user agrees to them upon install. When installing new software, users have a chance to think about what they are doing and to decide to trust software based on reviews, the developer's reputation, and the permissions required. Deciding up front allows them to focus on their goals rather than on security while using applications. Permissions are sometimes called ―manifest permissions or ―Android permissions to distinguish them from file permissions. To be useful, permissions must be associated with some goal that the user understands. it's possible to secure the use of all the different Android inter-process communication (IPC) mechanisms with just a single kind of permission. Starting Activities, starting or connecting to Services, accessing Content Providers, sending and receiving broadcast Intents, and invoking Binder interfaces can all require the same permission. Therefore users don't need to understand more than ―My new contact manager needs to read contacts. Once installed, an application's permissions can't be changed. By minimizing the permissions an application uses it minimizes the consequences of potential security flaws in the application and makes users feel better about installing it. When installing an application, users see requested permissions in a dialog similar to the one shown in Installing software is always a risk and users will shy away from software they don't know, especially if it requires a lot of permissions. From a developer's perspective permissions are just strings associated with a program and it's UID. There are only four protection levels for permissions.

ISBN: 9788175157262

6. CONCLUSION

Android applications have their own identity enforced by the system. Applications can communicate with each other using system provided mechanisms like files, Activities, Services, BroadcastReceivers, and ContentProviders. If you use one of these mechanisms you need to be sure you are talking to the right entity — you can usually validate it by knowing the permission associated with the right you are exercising. If you are exposing your application for programmatic access by others, make sure you enforce permissions so that unauthorized applications can't get the user's private data or abuse your program. Make your applications security as simple and clear as possible. When communicating with other programs, think clearly about how much you can trust your input, and validate the identity of services you call. Before shipping, think about how you would patch a problem with your application.

REFERENCES

[1] J.P. Anderson, Computer Security Technology Plan ning Study, tech. report ESDTR-73-51, Mitre, Oct. 1972.

[2] M.A. Harrison, W.L. Ruzzo, and J.D. Ullman, "Pro tection in Operating Systems," Comm. ACM vol. 19, no. 8, 1976, pp. 461–471.

[3] L. Badger et al., "Practical Domain and Type En forcement for UNIX," Proc. IEEE Symp. Se curity and Privacy, IEEE CS Press, 1995, pp. 66–77.

[4] J. Saltzer and M. Schroeder, "The Protection of Infor mation in Computer Systems," Proc. IEEE, vol. 63, no. 9,S 1975, pp. 1278–1308.

[5] I. Krstic and S.L. Garfinkel, "Bitfrost: The One Laptop per Child Security Model," Proc. Symp. Usable Privacy and Security, ACM Press, 2007, pp. 132–142.

[6] N. Li, B.N. Grosof, and J. Feigenbaum, "Dele gation Logic: A Logic-Based Approach to Dis tributed Authori zation," ACM Trans. Informa tion and System Security, vol no.1, 2003, pp. 128–171.

[7] W. Enck, M. Ongtang, and P. McDaniel, Miti gating Android Software Misuse before It Happens, tech. report NAS-TR-0094-2008, Network and Security Re search Ctr., Dept. Computer Science and Eng., Pennsyl vania State Univ., Nov. 2008.

[8] Chu, E. (2008, August 28). Android Market: a user- driven content distribution system. Re trieved August 30, 2008, from Android Devel oper's Blog.

[9] Google Inc. (2008, August 29). Security and Permis sions in Android. Retrieved August 30, 2008, from An droid - An Open Handset Al liance Project.

[10] Burns, Jesse (2009, October) developing secure mobile applications for android.

ISBN: 9788175157262