Reputation Based Security Model for Android Applications

Shanthakumar H C,
Associate Prof ,
Dept. of CS&E,
SJBIT, Bangalore
ShanthkumarHC@gmail.com

Shruthi K ,
1st sem,
Mtech (cs&e),
SJBIT, Bangalore
Shruthigowda1992nov26@gmail.com

Abstract: *The market for smart phones has been booming in the past few years. There are now over 400,000 applications of the Android market. Over 10 billion Android applications have been downloaded from the Android market. Due to the Android popularity, there are now a large number of malicious vendors targeting the platform. Many honest end users are being successfully hacked on a regular basis. In this work, a cloud based reputation security model has been proposed as a solution which greatly mitigates the malicious attacks targeting the Android market. The Reputation based security solution takes advantage of the fact that each application in the android platform is assigned a unique user id (UID). Our solution stores the reputation of Android applications in an anti-malware providers' cloud (AM Cloud).*

Keywords - Smartphones; Android OS; Reputation based security; Inter Process Communication

I.INTRODUCTION

Android is a Linux-based open-source operating system, with a layered structure of services including core native libraries and application frameworks. Android isolates different applications and provides safety through OS primitives and environmental features such as type safety. At the application level, each software package is partially sandboxed by the kernel, making Android a widely deployed system that employs privilege separation as a matter of course. More specifically, by default, each application is not allowed to read, update or delete another application's files. On the other hand, the Android user has the ability to allow or deny one application to communicate with other applications via inter process communication (IPC). If the user grants this permission, malware applications may be able to compromise the Android system. If the application requests this permission, and the user denies this permission, the possibly valid application will not install.

1.1 Android Operating System

The Android Operating System (OS) provides a rich set of IPC capabilities. Unfortunately there are many security issues in the OS. Since the Android phones are limited in processing power, memory and speed, there are some additional constraints in running strong security defenses such as live antivirus and firewall programs. Privacy issues are common in all OS's. However with Android's Global Positioning System (GPS) and Locations Based Services (LBS), hackers may be able to track the movements and location of the Android phone owner. Microsoft Windows applications do not have their own UID. When a Microsoft Windows application runs, it uses the security impersonation feature, which allows the application to run, in the security context of the user who runs the application. When the Android application runs, they run in the security context of their own UID. Therefore, with Android, it is possible to have more control over the security of the application. i.e., the Android applications do not, by default, have the security permissions of the phone user. However, we are missing some tools to fine tune the control the applications' security, based on these unique application UID's.

1.2 Components of Android

The different types of application components in the Android OS are: Activities, Services, Content providers and Broadcast receivers. A service in Android runs in the background. Other components, from various different applications can interact with the service and exchange data. When a service is installed, the developer has the option to make its service available to the applications. The installation manifest file must include a permission statement and give a name to the shared service (for example SuperService). If a user installs an application, which requires the use of SuperService, that application, must request access to SuperService via its installation manifest file. There are three problems with the above permissions strategy:

1. The installation options are either take it or leave it. If the user wants to prevent from accessing just one service, the user is must not install the application. It would be better if the user could install the application, but just prevent the application from accessing that one service.
2. The permission to use SuperService, is only checked at installation time. To make any permission changes (revoke or grant), requires the user to completely uninstall and reinstall the application, which uses SuperService.

3. It is the end user who must make the decision, if a newly installed application should be allowed to use the SuperService. Users are not in a good position to know which combinations of applications services are used by malware.

Powerful and well-connected smartphones are becoming increasingly common. Their features are provided by focused applications that users can easily install from application market places. With hundreds of thousands of applications available, however, there is only limited control over the quality and intent of those applications. Mobile code and extensibility is one of the key issues that increase the complexity of software security. To counter this threat, mobile operating systems impose security restrictions for each application. The Android mobile operating system is one of the major smartphone platforms. The Android security model enforces the least-privilege principle through application-level permissions that can be requested by the applications. End users need to grant the permissions at install time and decide on the adequacy of the required permissions and the trustworthiness of the individual application. Often it cannot be assumed that a user fully understands the risks related to granting permissions to applications. The purpose of Reputation Based Security Model is to present a solution to identify the reputation of applications. These reputations can be stored in the AM Cloud which will mitigate all three of these Android permission security limitations.

II.SYSTEM DESIGN AND IMPLEMENTATION

Early on in development, the core Android development team recognized that a robust security model was required to enable a vigorous ecosystem of applications and devices these precautions, security issues may occur after shipping, which is why the Android project has created a comprehensive security response process. A full-time Android security team constantly monitors Android-specific and the general security community for discussion of potential vulnerabilities. Upon the discovery of legitimate issues, the Android team has a response process that enables the rapid mitigation of vulnerabilities to ensure that potential risk to all Android users is minimized. These cloud-supported responses can include updating the Android platform (over-the-air updates), removing applications from Google Play, and removing applications from devices in the field.

*2.2 Android Platform Security Architecture*Android seeks to be the most secure and usable rating system for mobile platforms by repurposing traditional operating system security controls to: Protect user data Protect system resources (including the network) Provide application isolation To achieve these objectives, Android provides the following key security features:

2.2.1 Security Features:

Robust security at the OS level through the Linux kernel Mandatory application sandbox for all applications Secure interprocess communication Application signing Application-defined and user-granted permissions The built on and around the Android platform and supported by cloud services. As a result, through its entire development lifecycle, Android has been subjected to a professional security program.

2.1 Android Security

The Android team has had the opportunity to observe how other mobile, desktop, and server platforms prevented and reacted to security issues and built a security program to address weak points observed in other offerings.

2.1.1 The key components of the Android Security Program: Design Review: The Android security process begins early in the development lifecycle with the creation of a rich and configurable security model and design. Each major feature of the platform is reviewed by engineering and security resources, with appropriate security controls integrated into the architecture of the system. Penetration Testing and Code Review: During the development of the platform, Android-created and open-source components are subject to vigorous security reviews. These reviews are performed by the Android Security Team, Google's Information Security Engineering team, and independent security consultants. The goal of these reviews is to identify weaknesses and possible vulnerabilities well before the platform is open-sourced, and to simulate the types of analysis that will be performed by external security experts upon release. Open Source and Community Review: The Android Open Source Project enables broad security review by any interested party. Android also uses open source technologies that have undergone significant external security review, such as the Linux kernel. Incident Response: Even with all of

security components and considerations of the various levels of the Android software stack are summarized in fig 2.1. Each component assumes that the components are properly secured. With the exception of a small amount of Android OS code running as root, all code above the Linux Kernel is restricted by the Application Sandbox.

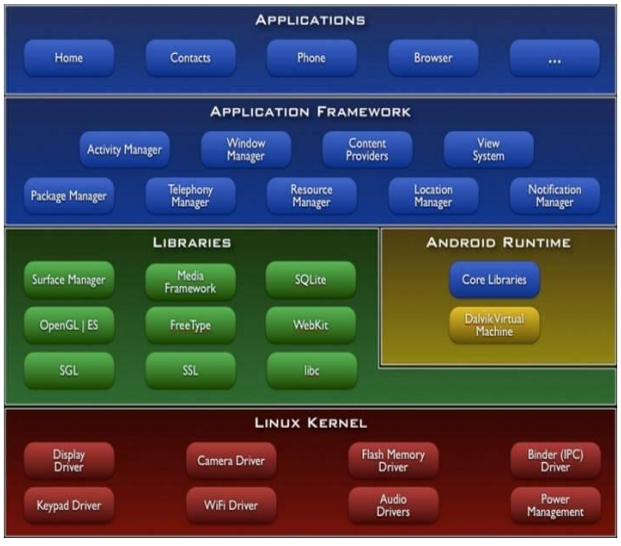

Fig. 2.1 Android software stack.

2.3 System and Kernel Level Security

At the operating system level, the Android platform provides the security of the Linux kernel, as well as a secure inter-process communication (IPC) facility to enable secure communication between applications running in different processes. These security features at the OS level ensure that even native code is constrained by the Application Sandbox. Whether that code is the result of included application behavior or exploitation of an application vulnerability, the system would prevent the rogue application from harming other applications, the Android system, or the device itself.

2.4 Linux Security

The foundation of the Android platform is the Linux kernel. The Linux kernel itself has been in widespread use for years, and is used in millions of security-sensitive environments. Through its history of constantly being researched, attacked, and fixed by thousands of developers, Linux has become a stable and secure kernel trusted by many corporations and security professionals.As the base for a mobile computing environment, the Linux kernel provides Android with several key security features. As a multiuser operating system, a fundamental security objective of the Linux kernel is to isolate user resources from one another. The Linux security philosophy is to protect user resources from one another. Thus, Linux:

2.5 The Application Sandbox

The Android platform takes advantage of the Linux user-based protection as a means of identifying and isolating application resources. The Android system assigns a unique user ID (UID) to each Android application and runs it as that user in a separate process. This approach is different from other operating systems (including the traditional Linux configuration), where multiple applications run with the same user permissions. The kernel enforces security between applications and the system at the process level through standard Linux facilities, such as user and group IDs that are assigned to applications. By default, applications cannot interact with each other and applications have limited access to the operating system. If application A tries to do something maliciouslike read application B's data or dial the phone without permission (which is a separate application), then the operating system protects against this because application A does not have the appropriate user privileges. The sandbox is simple, auditable, and based on decades-old UNIX-style user separation of processes and file permissions.

2.6 Android Components

Application components are the essential building blocks of an Android application. Each component is a different point through which the system can application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your application's overall behavior. There are four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

2.6.1 Activity components - define an application's user interface. Typically, 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.

2.6.2 Service components - perform background processing. When an activity needs to perform some operation that must continue after the user interface disappears (such as download play music), it commonly starts a service specifically designed for that action. The developer can also use services as application- specific daemons, possibly starting on boot. Services often define an interface for Remote Procedure Call (RPC) that other system components can use to send commands and retrieve data, as well as register callbacks.

2.6.3 Content provider- components store and share data using a relational database interface. Each content provider has an associated ―authority‖ describing the content it contains. Other components use the authority name as a handle to perform SQL queries (such as SELECT, INSERT, or DELETE) to read and write content. Although content providers typically store values in database records, data retrieval is implementation specific—for example, files are also shared through content provider interfaces.

2.6.4 Broadcast receiver – components act as mailboxes for messages from other applications. Commonly, application code broadcasts messages to an implicit destination. Broadcast receivers thus subscribe to such destinations to receive the messages sent to it. Application code can also address a broadcast receiver explicitly by including the namespace assigned to its containing application.

2.7 Component Interaction

The primary mechanism for component interaction is an *intent*, which is simply a message object containing a destination component address and data. The Android API defines methods that accept intents and uses

that information to start activities, start services (start Service (Intent)), and broadcast messages. The invocation of these methods tells the Android framework to begin executing code in the target application. The process of inter component communication as described above is known as an *action*. Simply put, an intent object defines the ―intent‖ to perform an ―action.‖ One of Android's most powerful features is the exibility allowed by its intent-addressing mechanism. Although developers can uniquely address a target component using its application's namespace, they can also specify an implicit name. In the latter case, the system determines the best component for an action by considering the set of installed applications and user choices. The figure 2.2 shows the interaction between components in the Friend Tracker and Friend Viewer applications and with components in applications defined as

part of the base Android distribution. In each case, one component initiates communication with another.

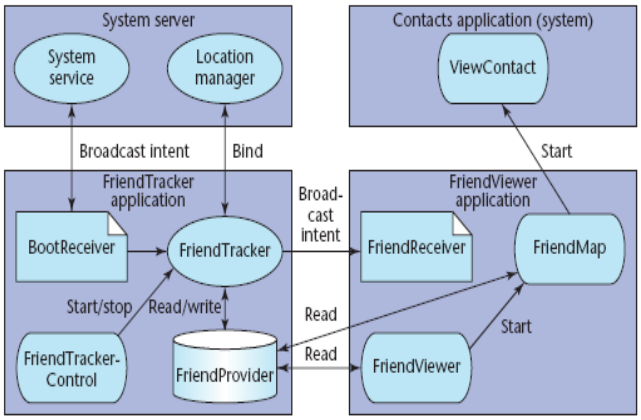

Fig. 2.2 Component Interaction

III. SYSTEM ANALYSIS

Analysis approach aims to identify information flows between different Android applications/components. To analyze a larger set of Applications, as it usually exists on an end user's phone, interprocess communication is considered.

3.1 Interprocess Communication

Processes can communicate using any of the traditional UNIX-type mechanisms. Examples include the filesystem, local sockets, or signals. However, the Linux permissions still apply. Android also provides new IPC mechanisms:

3.1.1 IPC Mechanisms Binder: A lightweight capability-based remote procedure call mechanism designed for high performance when performing in-process and cross-process calls. Binder is implemented using a custom Linux driver. Services: Services (discussed above) can provide interfaces directly accessible using binder. Intents: Intent is a simple message object that represents an "intention" to do something. For example, if your application wants to display a web page, it expresses its "Intent" to view the URL by creating an Intent instance and handing it off to the system. The system locates some other piece of code (in this case, the Browser) that knows how to handle that Intent, and runs it. Intents can also be used to broadcast interesting events. ContentProviders: A ContentProvider is a data storehouse that provides access to data on the device; the classic example is the ContentProvider that is used to access the user's list of contacts. An application can access data that other applications have exposed via a ContentProvider, and an application can also define its own ContentProvider to expose data of its own.

3.2 Android Developer Tools

The Android SDK provides the API libraries and developer tools necessary to build, test, and debug apps for Android. ADT Bundle includes the essential Android SDK components and a version of the Eclipse IDE with built-in ADT (Android Developer Tools) to streamline your Android app development.

3.3 System Requirements

All computer software needs certain hardware components or other software resources to be present on a computer. These prerequisites are known as system requirements.

3.4 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.

3.4.1 Example Application

A Pair of applications was developed to describe how Android applications operate. Consider a location-sensitive social networking application for mobile phones in which users can discover their friends' locations. The functionality of the applications is split into two components: one for tracking friends and one for viewing them. Consider a location-sensitive social networking application for mobile phones in which users can discover their friends' locations. The functionality of the applications is split into two components: one for tracking friends and one for viewing them.

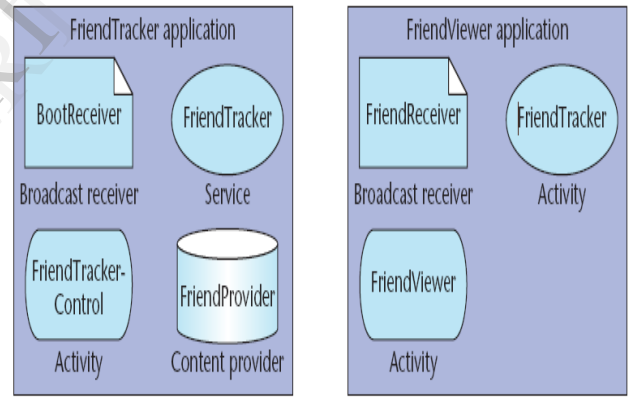

Fig. 3.1 FriendTracker and FriendViewer applications

3.5 Security Enforcement

As shown in Fig 3.2, 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 general case, each application runs as a unique user identity, which lets Android limit the potential damage of programming rows. For example, the Web browser vulnerability discovered recently after

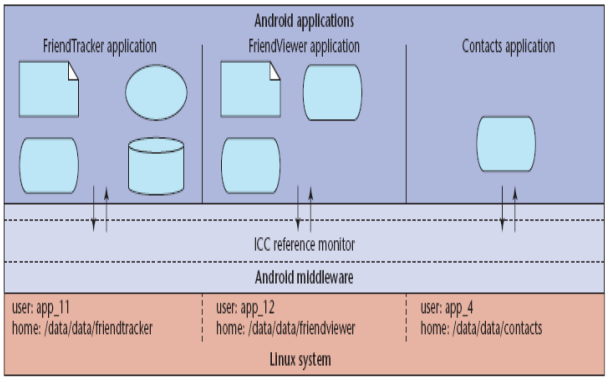

Fig. 3.2 Security Enforcement in Android

A similar vulnerability in Apple's iPhone gaveway to the ―jail breaking‖ technique, which let users replace parts of the underlying system, but would also have enabled a network-based adversary. ICC isn't limited by user and process boundaries. In fact, all ICC occurs via an I/O control command on a special device node, /dev/binder. Because the file must be world readable and writable for proper operation, the Linux system has no way of mediating ICC. Although user separation is straightforward and easily understood, controlling ICC is much more subtle and warrants careful consideration. 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. Below figure depicts this logic.

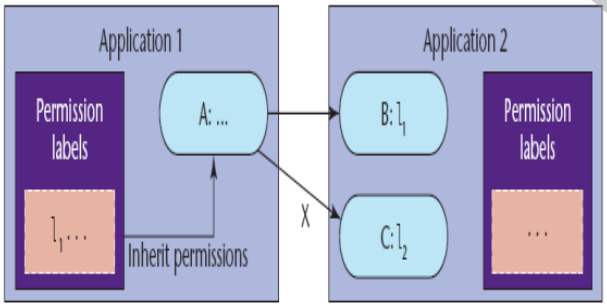

Fig. 3.3 Access permission logic.

As shown in Fig 3.3, the Android middleware implements a reference monitor providing mandatory access control (MAC) enforcement about how applications access components. The basic enforcement model is the same for all component types. Component A's ability to access components B and C is determined by comparing the access permission labels on B and C to the collection of labels assigned to application 1. The developer assigns permission labels via the XML manifest file that accompanies every application package. In doing so, the developer denies 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.

IV.CONCLUSION AND FUTURE WORK

Android smart phones are becoming very popular among the different smart phone platforms and this is expected to increase in popularity. Despite its popularity momentum, its open source software and programmable framework behavior make it vulnerable to virus attacks. A Reputation based security model for android applications, takes into consideration the fact that Smart phones are memory, battery and speed constrained and hence exploiting the cloud to do the reputation index computation of a given application. By referring to the calculated matrix of reputation built by a given application, the model will notify users on the potential risk of the application before installation. Applications can be classified as highly risky, medium risk, less risk and genuine all based on reputation they have built in the cloud. The experimental results show that some application need to be regarded as highly risky and therefore warn users not to install them until they improve their reputation by passing the threshold set by the reputation based security model.

REFERENCES

[1]. A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. Google Android: Comprehensive Security Assessment. In IEEE Security & Privacy, Volume 8, Issue 2, 35– 44, March–April 2010.

[2]. T. Bläsing, L. Batyuk, A.-D. Schmidt, S.A. Camtepe, and S. Albayrak. An Android application Sandbox system for suspiciou software detection. In Proceedings of 5th International Conference on Malicious and Unwanted Software (MALWARE 2010), Nancy, France, Oct. 19–20, 2010.

[3]. M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically Rich Application Centric Security in Android. In Proceedings of the Annual Computer Security Applications Conference (ACSAC '09), Austin, TX, USA, December 6–10, 2009.

[4]. W. Shin, S. Kiyomoto, K. Fukushima, and T. Tanaka. Towards Formal Analysis of the Permission-Based Security Model for Android. In Proceedings of Fifth International Conference on Wireless and Mobile Communications (ICWMC '09), Cannes/La Bocca ,France, August 23–29, 2009.

[5]. P. Teufl, C. Orthacker, S. Kraxberger, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder and O. Prevenhueber. Android Market Analysis with Activation Patterns, In Proceedings of 3rd International ICST Conference on Security and Communication Systems (MOBISEC 2011), Aalborg, Denmark, May 17–19, 2011.

[6]. C. Orthacker, P. Teufl, S. Kraxberger, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder , Prevenhueber. Android Security Permissions - Can we trust them? In Proceedings of 3rd International ICST Conference on Security and Privacy in Mobile Information and Communication Systems (MOBISEC 2011), Aalborg, Denmark, May 17–19, 2011.

[7] J. Burns. Developing Secure Mobile Applications for Android—An Introduction to Making Secure Android Applications,

[8] E. Chin, A. Porter Feltm, K. Greenwood, and D. Wagner. Analyzing the Inter-application Communication in Android, University of California, Berkeley, Berkeley, CA, USA.

[9] T. Vidas, D. Votipka, and N. Christin. All Your Droid Are Belong To Us: A Survey of Current Android Attacks, INI/CyLab, Carnegie Mellon University