ISSN 2277-3061

Security in Android

Amritesh Kumar Sharma1, Arun Kumar Singh2, Pankaj P. Singh3
M.Tech Scholar IIMT Group of Colleges, Meerut[1]
amriteshkumarsharma@gmail.com
Associate Professor, IIMT Group of Colleges, Meerut[2]
arun_apjs@yahoo.co.in
Associate Professor, IIMT Group of Colleges, Meerut3
pankajpratapsinghcs@gmail.com

ABSTRACT

New technologies have always created new areas of concern for information security teams. Usually it provides time for the development of effective security controls. The rapid growth of the smartphone in market and the use of these devices for so many sensitive data have led to the emergence of security threat. A malicious user or malware on a device can create a number of risks for an organization, and so the fact that these devices are not necessarily connected does not translate to a lack of security risks.

This paper will discuss why it is important to secure an Android device, what some of the potential vulnerabilities are, and security measures that can be introduced to provide a baseline of security of data on Google's mobile OS

Indexing terms/Keywords

Academic Discipline And Sub-Disciplines

Technology – Computer science

SUBJECT CLASSIFICATION

Computer science – Operating Systems - Android

TYPE (METHOD/APPROACH)

Technical Analysis

INTRODUCTION

Android is a mobile platform which enables application development while making use of local as well as server side data using advanced hardware and software. To protect that value, the platform must offers application environment that ensures the security of user's, data, applications, the device, and the network.

The multi-layered security of Android provides the flexibility in terms of security which is must for an open platform, while providing protection for all users of the platform. The great design of Android was done with device users in mind. They have been provided visibility that how applications work, and control over those applications. It includes the expectation the some attacker would attempt to perform common attacks, for example social engineering attacks to convince device users to install malware, and attacks on third-party applications on device. Android takes care of both reduction in probability of these attacks and greatly limit the impact of the attack if it happen [2].

This document shows the goals of the Android security program, describes the Android security architecture as much as possible, and answers the most pertinent questions. It focuses on the security related features of Android's core platform and does not include security issues that are unique to specific applications, for example related to the browser or SMS application. Recommended best practices for building Android devices, deploying Android devices, or developing applications for Android are not the goal of this document at all.

1. ANDROID PLATFORM MAIN BUILDING BLOCKS

- **Device Hardware**: Android is hardware unspecific it runs on a wide range of hardware configurations including smart phones, tablets, and set-top-boxes and can also be configured for many more. Android is processor-agnostic, but it does take advantage of some hardware-specific security capabilities such as ARM etc.
- **Android Operating System**: The Android core operating system is built on top of the Linux kernel and device resources, like camera functions, GPS data, Bluetooth functions, telephony functions, network connections, etc. are accessed through the operating system.
- **Android Application Runtime**: Android applications are most often written in the Java programming language and run in the Dalvik virtual machine. However, many applications, including core Android services and applications are native applications or include native libraries. Both Dalvik and native applications run within the same security environment, contained within the Application Sandbox. Applications get a dedicated part of the file system in which they can write private data, including databases and raw files.

1.1 Android Security Program Overview

After observing several mobile and desktop platforms in regards to security issues the Android team built a security program which addresses weak points observed in other offerings. As a result the Android has been subjected to a professional security program through its entire life cycle.

Android Security Program, The key components: [3]

- **Design Review**: Security process in Android begins early in the development lifecycle with the creation of a rich and configurable security model and design.
- **Code Review and Penetration Testing**: 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
    - **Open Source**: The Android Open Source Project enables broad security review by any interested party. Android uses open source technologies that have undergone significant external security review, such as the Linux kernel.
    - **Incident Response**: 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.

1.2 Android Platform Security Architecture [3]

Android security architecture has been divided into three parts as under

- Protection of user data
- Protection of system resources (including the network)
- Provide application isolation

And the most important, to achieve these android provide following security features.

- OS level through the Linux kernel
- Mandatory application sandbox for all applications
- Secure interprocess communication
- Application signing

Following section describes these and other security features of the Android platform. *Figure below* summarizes the security components and considerations of the various levels of the Android software stack. Each component assumes that the components below are properly secured. [4]

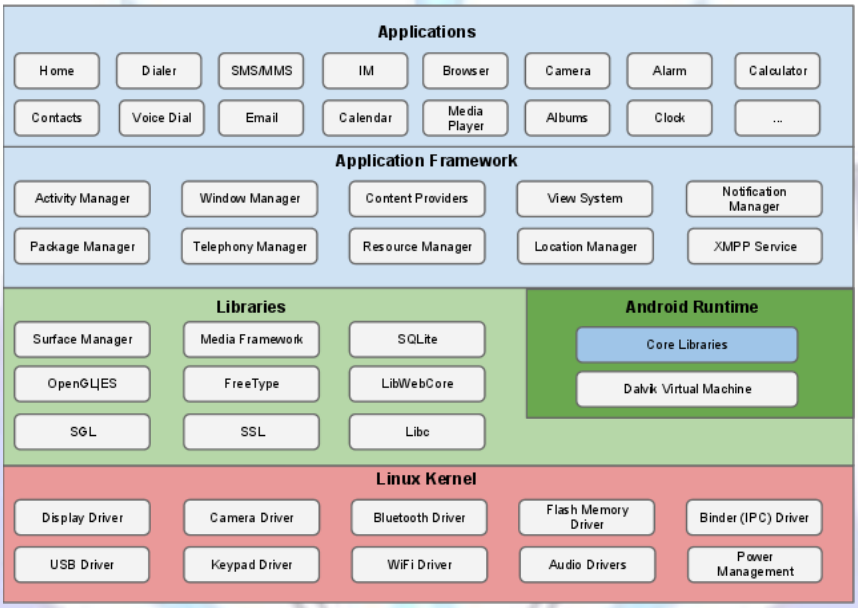

Figure 1.1: Android platform

2. SYSTEM AND KERNEL LEVEL SECURITY

Android provides the security of Linux kernel at the operating system level 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. This feature will prevent the rogue application from harming other applications, the Android system, or the device itself.

2.1 Linux Security

The Android platform has been built on 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.

On the ground of mobile computing environment, the Linux kernel provides Android with several key security features, including:

- A user-based permissions model
- Process isolation
- Extensible mechanism for secure IPC
- The ability to remove unnecessary and potentially insecure parts of the kernel

As we all know Linux, being 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:

- Prevents user A from reading user B's files
- Ensures that user A does not exhaust user B's memory
- Ensures that user A does not exhaust user B's CPU resources
- Ensures that user A does not exhaust user B's devices (e.g. telephony, GPS, Bluetooth etc.)

2.2 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.

This sets up a kernel-level Application Sandbox. 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 malicious like 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.

Since the Application Sandbox is in the kernel, this security model extends to native code and to operating system applications. All of the software above the kernel in *Figure 1.1*, including operating system libraries, application framework, application runtime, and all applications run within the Application Sandbox. On some platforms, developers are constrained to a specific development framework, set of APIs, or language in order to enforce security. On Android, there are no restrictions on how an application can be written that are required to enforce security; in this respect, native code is just as secure as interpreted code.

In some operating systems, memory corruption errors generally lead to completely compromising the security of the device. This is not the case in Android due to all applications and their resources being sandboxed at the OS level. A memory corruption error will only allow arbitrary code execution in the context of that particular application, with the permissions established by the operating system.

Like all security features, the Application Sandbox is not unbreakable. However, to break out of the Application Sandbox in a properly configured device, one must compromise the security of the Linux kernel.

2.3 Filesystem Permissions

Filesystem permissions ensure that one user cannot alter or read another user's files. In the case of Android, each application runs as its own user. Unless the developer explicitly exposes files to other applications, files created by one application cannot be read or altered by another application[5].

2.4 Cryptography [5]

Android provides a set of cryptographic APIs for use by applications. These include implementations of standard and commonly used cryptographic primitives such as AES, RSA, DSA, and SHA. Additionally, APIs are provided for higher level protocols such as SSL and HTTPS.

3. USER SECURITY

3.1 Filesystem Encryption

Full filesystem encryption was provided by Android 3.0 and later, so all user data can be encrypted in the kernel using the dmcrypt implementation of AES128 with CBC and ESSIV: SHA256. The encryption key is protected by AES128 using a key derived from the user password, preventing unauthorized access to stored data without the user device password. To provide resistance against systematic password guessing attacks (e.g. “rainbow tables” or brute force), the password is combined with a random salt and hashed repeatedly with SHA1 using the standard PBKDF2 algorithm prior to being used to decrypt the filesystem key. To provide resistance against dictionary password guessing attacks, Android provides password complexity rules that can be set by the device administrator and enforced by the operating system. Filesystem encryption requires the use of a user password; pattern-based screen lock is not supported.

3.2 Password Protection

Android can be configured to verify a user-supplied password prior to providing access to a device. In addition to preventing unauthorized use of the device, this password protects the cryptographic key for full filesystem encryption. Use of a password and/or password complexity rules can be required by a device administrator.

3.3 Device Administration

Android 2.2 and later provide the Android Device Administration API, which provides device administration features at the system level. For example, the built-in Android Email application uses the APIs to improve Exchange support. Through the Email application, Exchange administrators can enforce password policies — including alphanumeric passwords or numeric PINs — across devices. Administrators can also remotely wipe (that is, restore factory defaults on) lost or stolen handsets.

3.4 Credential Storage

By default, Android includes a set of predefined Certificate Authorities (CAs) that are trusted for operations such as establishing SSL connections within the browser. In Android 4.0 and later, users can disable preinstalled CAs within the system settings. Users can also add trusted CAs or certificates to the system by importing them from USB storage. Android 4.1 and later adds the ability for OEMs to add hardware-backed Key Chain storage which binds private keys to the device on which they are stored.

3.5 Virtual Private Network

Android provides support for PPTP, L2TP, and IPsec VPNs by a built-in VPN client. In addition, Android 4.0 introduced the VPN Service class to support third-party VPN solutions.

4. APPLICATION SECURITY

4.1 Elements of Applications

Android applications are most often written in the Java programming language and run in the Dalvik virtual machine. However, applications can also be written in native code. Applications are installed from a single file with the .apk file extension[7].The application building blocks are:

- **AndroidManifest.xml**:The AndroidManifest.xml file is the control file that tells the system what to do with all the top-level components (specifically activities, services, broadcast receivers, and content providers described below) in an application. This also specifies which permissions are required.
- **Activities**: An Activity is, generally, the code for a single, user-focused task. It usually includes displaying a UI to the user, but it does not have to -- some Activities never display UIs. Typically, one of the application's Activities is the entry point to an application.
- **Services**: A Service is a body of code that runs in the background. It can run in its own process, or in the context of another application's process. Other components "bind" to a Service and invoke methods on it via remote procedure calls. An example of a Service is a media player: even when the user quits the media-selection UI, the user probably still intends for music to keep playing. A Service keeps the music going even when the UI has completed.
- **Broadcast Receiver**: A BroadcastReceiver is an object that is instantiated when an IPC mechanism known as Intent is issued by the operating system or another application. An application may register a receiver for the low battery message, for example, and change its behavior based on that information.

4.2 The Application's Permission Model

4.2.1 Accessing Protected APIs: By default, an Android application can only access a limited range of system resources. The system manages Android application access to resources that, if used incorrectly or maliciously, could adversely impact the user experience, the network, or data on the device. These restrictions are implemented in a variety of different forms. Some capabilities are restricted by an intentional lack of APIs to the sensitive functionality (e.g. there is no Android API for directly manipulating the SIM card). In some instances, separation of roles provides a security measure, as with the per-application isolation of storage. In other instances, the sensitive APIs are intended for use by trusted applications and protected through a security mechanism known as Permissions.These protected APIs are:

- Camera functions
- Location data (GPS)
- Bluetooth functions
- Telephony functions
- SMS/MMS functions
- Network/data connections

These resources are only accessible through the operating system. To make use of the protected APIs on the device, an application must define the capabilities it needs in its manifest. When preparing to install an application, the system displays a dialog to the user that indicates the permissions requested and asks whether to continue the installation. If the user continues with the installation, the system accepts that the user has granted all of the requested permissions. The

user can not grant or deny individual permissions -- the user must grant or deny all of the requested permissions as a block.

Once granted, the permissions are applied to the application as long as it is installed. To avoid user confusion, the system does not notify the user again of the permissions granted to the application, and applications that are included in the core operating system or bundled by an OEM do not request permissions from the user. Permissions are removed if an application is uninstalled, so a subsequent re-installation will again result in display of permissions.

In the event that an application attempts to use a protected feature which has not been declared in the application's manifest, the permission failure will typically result in a security exception being thrown back to the application. Protected API permission checks are enforced at the lowest possible level to prevent circumvention. An example of the user messaging when an application is installed while requesting access to protected APIs is shown in figure next.

4.2.2 Interprocess Communication: In android 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 has its own new 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.[9]
- **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. [10]
- **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. [11]

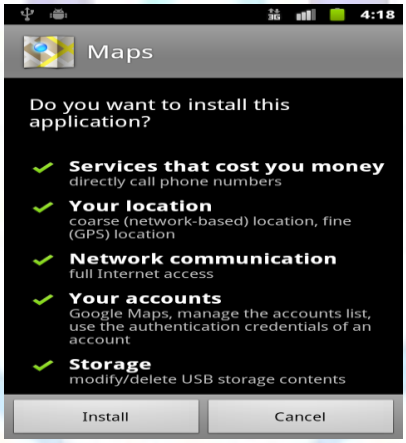

Figure 4.1: Interprocess Communication

4.3 Application Signing

Application/Code signing allows developers to identify the author of the application and to update their application without creating complicated interfaces and permissions. Every application that is run on the Android platform must be signed by the developer. Applications that attempt to install without being signed will rejected by either Google Play or the package installer on the Android device.

On Android, application signing is the first step to placing an application in its Application Sandbox. The signed application certificate defines which user id is associated with which application; different applications run under different user IDs. Application signing ensures that one application cannot access any other application except through well-defined IPC.

When an application (APK file) is installed onto an Android device, the Package Manager verifies that the APK has been properly signed with the certificate included in that APK. If the certificate (or, more accurately, the public key in the certificate) matches the key used to sign any other APK on the device, the new APK has the option to specify in the manifest that it will share a UID with the other similarly-signed APKs.

Applications can be signed by a third-party (OEM, operator, alternative market) or self-signed. Android provides code signing using self-signed certificates that developers can generate without external assistance or permission. Applications do not have to be signed by a central authority. Android currently does not perform CA verification for application certificates.

4.4 Application Verification

Android 4.2 and later support application verification. Users can choose to enable “Verify Apps" and have applications evaluated by an application verifier prior to installation. App verification can alert the user if they try to install an app that might be harmful. If an application is especially bad, it can block installation.

5. CONCLUSION

In this document we discussed security in Android, existing techniques and details and architecture of security in Android operating system, filesystem permissions and cryptography in Android. This document shows the details of Application security and the device and the operating system protection itself that how all these things have been implemented in such a small sized operational software. The most important and beautiful thing with Android, which must always be admired is the use of Linux kernel itself as a core operating system component which makes it more reliable, robust and secure most in comparison to other mobile operating systems present today.

After going through all these details we come to know that calling Android a complete ecosystem is not a wonder as it provides a complete fully functional environment to applications running on it and protect others also including operating system itself.

REFERENCES

[1] Artem Russakovskii "Android 4.4.2 (KOT49H) Is Already Rolling Out To All Nexus Devices - Here Are The OTA ZIP Links For Manual Updating". Android Police. December 9, 2013.

[2] Android developers (2011, Mar 03). Android Architecture;http://developer.android.com/guide/basics/what-is-android.html

[3] Shah, Agam "Google's Android 4.0 ported to x86 processors". Computerworld. International Data Group. December 1, 2011.

[4] Mahapatra, Lisa "Android Vs. iOS: What’s The Most Popular Mobile Operating System In Your Country?". November 11, 2013.

[5] Ingrid Lunden "Android, Led By Samsung, Continues To Storm The Smartphone Market, Pushing A Global 70% Market Share". TechCrunch. AOL Inc. July 1, 2013.

[6] Android Open Source Project, Android Technical Information; http://source.android.com/license.html

[7] Tim Strazzere and Jay Freeman, Overview of android security strazzere.com/blog/?p=104

[8] Android Open Source Project, Android Security Tips,https://developer.android.com/training/articles/security-tips.html

[9] Gartner Inc., “Android to overtake iPhone in 2012,” October2009,http://www.computerworld.com/s/article/9139026/

[10] BURNS J., 2009. Exploratory Android Surgery – Digging into Droids. Presented in Black Hat Conference, USA.

[11] Android Development Team. Webviewclient hooks list.http://developer.android.com/reference/

android/webkit/WebViewClient.html.

Author’ biography with Photo