Digital Right Management and Software Protection on Android Phones

Chen-Yuan Chuang[1,2] Yu-Chun Wang[1] Yi-Bing Lin[2]

[1]Telecommunication Laboratories, Chunghwa Telecom Co., Ltd., Taipei, Taiwan

[2]Department of Computer Science, National Chiao-Tung University, Hsinchu, Taiwan

zzy@cht.com.tw ycwang@cht.com.tw liny@csie.nctu.edu.tw

***Abstract*— Android is an open mobile phone platform. To accommodate value-added services such as selling wallpapers, ringtones, applications, and games on Android phones, it is essential to ensure copyright protection on these products. This paper studies how the Android source code to implement the Open Mobile Alliance (OMA) Digital Right Management (DRM) 1.0, software installation and protection. We also identify potential leaks of Android DRM and software protection in this study.**

Keywords-component; Android; OMA DRM; open source; Cupcake; Android Market

I. Introduction

Android is an open mobile phone platform and accommodate value-added services in feasible ways. As shown in Figure 1, the Android architecture consists of four layers [1]: *Linux kernel*, *Libraries* and *Android runtime*, *Application framework*, and *Applications*. The Android version 1.5 (Cupcake) adopts the *Linux kernel* 2.6.27 (Figure 1 (1)) to provide core system services and act as a hardware abstraction layer. The *Libraries* (Figure 1 (2)) expose various features to developers through the *Application framework* (Figure 1 (4)), e.g., multimedia playback, database, font rendering, and web browser engine. By using a virtual machine, the *Android runtime* (Figure 1 (3)) executes a compact Dalvik EXecutable (.dex) application, which is converted from Java bytecode. The built-in and third-party *Applications* (Figure 1 (5)) are developed with the framework APIs. All the code in a single Android PacKage (APK) file is considered to be an application, in which the Java classes can be derived from four major fundamental components described below.

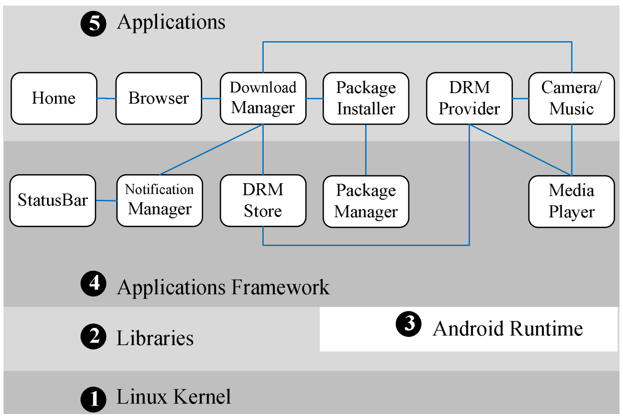


Figure 1. Android Archtecture

- *Activity* component provides a visual user interface and interacts with the user. An *Activity* can start another one and delivery parameters via *intent*, which acts as high-level Inter-Process Communication (IPC) in Android. In Figure 1(5), some classes in the applications, such as Home, Browser, Package Installer, Camera and Music, are derived from the Activity class.
- *Service* component performs a background process, such as audio playback, when a foreground one executes. In the *Service*'s lifecycle, another *Activity* can be a controller to start or stop the *Service*. Some classes in the applications, such as Download Manager, Package Manager, and MediaPlayer (Figure 1 (4)), are derived from the Service class.
- *Broadcast receiver* component does not display any user interface but starts an *Activity* when receiving broadcasted *intents* from the system or user application events, such as low battery warning or new photo has been taken. The DownloadReceiver class in the Download Manager (Figure 1 (5)) is derived from the BroadcastReceiver class.
- *Content provider* component shares data via Uniform Resource Identifier (URI) with content: scheme [1]. A *Content provider* maintains the shared data in the file system or a SQLite3 database [2], and other applications utilize ContentResolver objects to manipulate the data. Some classes in the Download Manager and the DRM Provider (Figure 1 (5)) are derived from the ContentProvider class.

Android phones accommodate value-added services such as selling wallpapers, ringtones, applications and games. It is essential to ensure copyright protection on these digital contents. In this paper, we used an Android-based HTC handset, called Dream [3], to develop Cupcake images based on the Android open source. We evaluated the implementations in Android Cupcake to identify problems of DRM support and software protection. Other commercial Android Cupcake phones, HTC Magic [4] and Hero [5], were also examined.

This paper is organized as follows. Section II presents the background of Android development. Section III describes DRM protection on Android phones. In Section IV, we show the software installation and some protection mechanisms. Conclusions are summarized in section V.

II. Background

This section elaborates on Android Software Development Kit (SDK), and Android software and runtime environment.

A. Android SDK

The Android SDK [6] provides Java API documents, phone emulator, USB connection driver, compiler, and debug tools executed in developer's PC hosts. The Android Debug Bridge (ADB) tool in the SDK provides shell command interface for console terminal, application install/uninstall and file pull/push between the phone and the PC host. The Dalvik Debug Monitor Service (DDMS) in the SDK provides process status, file explorer, screen capture, debug log messages and emulation features for incoming call/text message and location.

Android Applications were originally developed using pure Java programming language. Recently, Android also released a companion tool, the Native Development Kit (NDK), to combine native C/C++ code with Java code.

As the SDK, the download of the Android open source [7] is totally free. The open source gives a principle example for Android application and system development.

B. Android Software and Runtime Environment

An Android application is ZIP-archived into an APK file, which mainly contains a res folder, a classes.dex file, and an AndroidManifest.xml file. The res folder includes image and XML-based files that define window layouts, menus, and strings for user-interface design. The classes.dex file stores Dalvik bytecode, and the AndroidManifest.xml file describes activities, content providers, services, intents, broadcast receivers, and permission requirements if need in the application. For example, a conventional application with Google map may require permissions of accessing GPS device and Internet. Android's permission control prompts a user to grant permission during application installation, instead of performing authorization redundantly when the application is executed.

While /system/app folder stores system built-in APK files, user-installed APK files are located in /data/app folder. The file permission control inherits from Linux. Each application has a corresponding sandbox for individual runtime data in /data/data/<PackageName> folder with a unique Linux user and group ID. The sandbox contains two informative subfolders: The files/ subfolder stores the files, which are created and accessible by the application; the databases/ subfolder stores SQLite3 database files for the application.

For security reason, existing commercial phones restrain ADB pull and the /bin/ls console terminal command from accessing the sandbox files in /data partition. On our Dream prototype or the emulator, /data partition is fully accessible to fulfill the requirements of software development.

Some hackers have released their customized system images to facilitate users to "upgrade" their commercial phones to so-called rooted phones. The rooted phones can provide more features, e.g., Wi-Fi tethering [8] and access the whole file system, by modifying Linux configuration and kernel. We will investigate the problem of loopholes created by the rooted phones in DRM and software protection.

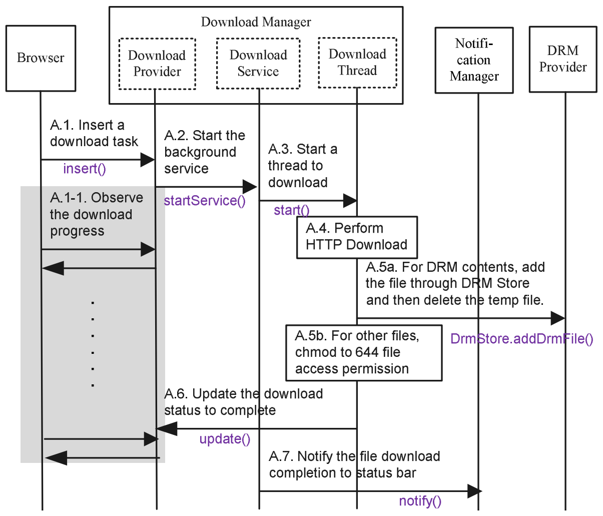
III. Download and OMA DRM Protection

On Android phones, the "Home" is a desktop application where a user can click on an icon to launch an application such as contacts, dialer, camera, music, settings or browser. Furthermore, the user can also click on an icon in the status bar at the top of screen to read some notifications, such as download completion, miss call, text message arrival or Wi-Fi connection activation.

This section describes the message flow of downloading a media file with OMA DRM Forward-Lock protection from an HTTP server. We will discuss the problem of cracking DRM protection.

A. Download Procedure

Figure 2 shows the message flow of downloading a file on an Android phone with the following steps:



Download Manager
Browser
Download Provider
Download Service
Download Thread
Notifi-cation Manager
DRM Provider
A.1. Insert a download task
insert()
A.2. Start the background service
startService()
A.3. Start a thread to download
start()
A.1-1. Observe the download progress
A.4. Perform HTTP Download
A.5a. For DRM contents, add the file through DRM Store and then delete the temp file.
DrmStore.addDrmFile()
A.5b. For other files, chmod to 644 file access permission
A.6. Update the download status to complete
update()
A.7. Notify the file download completion to status bar
notify()

Figure 2. Message Flow for File Download

Step A.1: In the browser, the user clicks on a web hyperlink to download a file. Derived from the ContentProvider base-class, the DownloadProvider allows the browser to insert the task into download list and returns a URI to the browser. After that, the browser obtains the download task URI content://downloads/download/<_id>, where the <_id> is a unique identify number of the task.

Step A.1-1: The browser repeats the process of obtaining URI at Step A.1 and shows the download progress in a display *Activity*.

Step A.2: The DownloadProvider maintains a list for all download tasks and initializes the background service DownloadService to manage all download tasks.

Step A.3: The DownloadService runs a loop and creates a DownloadThread instance to handle the download task.

In the loop, the DownloadService periodically queries the download list URI and notifies the status of progressing tasks to the NotificationManager. With that, the user can also observe the progress in the status bar. We omit these query and notification messages in Figure 2, except the last notification message at Step A.7.

Step A.4: The DownloadThread performs HTTP download and saves the downloaded file into the external storage (e.g., a SD card), which is mounted in /sdcard folder.

The DownloadThread checks the size of downloaded file and updates the status in DownloadProvider periodically. We omit these update messages in Figure 2, except the last update message at Step A.6.

Step A.5a: If the downloaded file is protected by OMA DRM Forward-Lock mechanism that Multipurpose Internet Mail Extensions (MIME) type is application/vnd.oma.drm.message, the DownloadThread calls the static method DrmStore.addDrmFile(), which decrypts the protected file and inserts the media content and metadata into the DrmProvider. The method returns an intent indicating the MIME type and the associating URI refers to content://drm/images/<_id> or content://drm/audio/<_id>. After that, the DownloadThread deletes the temporal file in /sdcard.

The DrmProvider maintains the media content files in its sandbox located in /data/data/com.android.providers.drm/. A database file drm.db stores MIME types, file sizes, file names and titles of the media contents. The files folder stores the unencrypted raw image and audio contents in individual files, named as DRM-<rand>.data, where the <rand> is a four-digit random number.

Step A.5b: If the downloaded file is DRM-free, the DownloadThread changes Linux file permission of the file to 644, which means that the file owner can read/write but other Linux users or processes can read only through the URI, file://sdcard/<FileName>. Note that Linux file permission is not applicable to FAT-formatted SD cards.

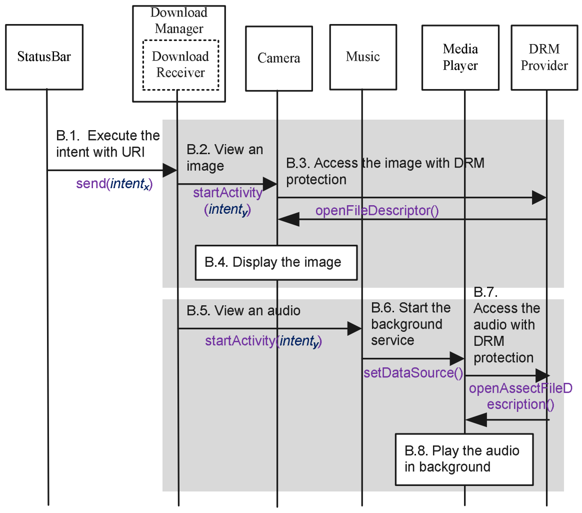
Step A.6: The DownloadThread updates the download task in DownloadProvider that the download status is Complete and the downloaded file can be retrieved from the URI obtained at Step A.5a or A.5b. Consequently, the browser shows the download completion to the user at the end of Step A.1-1.

Step A.7: The DownloadService sends a Notification object to the system service NotificationManager to indicate the completion of file download. The Notification object describes that the event should put an entry into the status bar and contains a data member contentIntent, which defines the $intent_x$ to execute when the status entry is clicked. As a result, an icon is placed in the status bar for opening the downloaded file.

At Step A.5a in Figure 2, an image/audio file with DRM protection is finally stored in a secured space. Oppositely, the DownloadThread leaves a downloaded APK file in the SD card (at Step A.5b), which can be freely accessed. For commercial software that needs copy protection, we can develop an application to download the APK file into its sandbox and set the Linux file permission to be readable for other applications, e.g., a software installation tool.

B. OMA DRM Protection

Figure 3 shows the message flow of accessing media contents with DRM support in Android with the following steps:



StatusBar
Download Manager
Download Receiver
Camera
Music
Media Player
DRM Provider
B.1. Execute the intent with URI
send($intent_x$)
B.2. View an image
startActivity ($intent_y$)
B.3. Access the image with DRM protection
openFileDescriptor()
B.4. Display the image
B.5. View an audio
startActivity($intent_y$)
B.6. Start the background service
setDataSource()
B.7. Access the audio with DRM protection
openAssectFileDescription()
B.8. Play the audio in background

Figure 3. Message Flow for Media File Access

Step B.1: When the user clicks on the icon in the status bar, the associated $intent_x$ with the content://downloads/download/<_id> URI is executed, so that the DownloadReceiver in the Download Manager is invoked and examines data source through content provider mechanism.

The DownloadReceiver obtains the content://downloads/download/<_id> URI in the $intent_x$ and then queries the MIME type and the file storage or content provider URI, e.g., file://sdcard/<FileName> or content://drm/images/<_id>, referring to the media content. After initializing the intent $intent_y$, the DownloadReceiver starts an activity for $intent_y$. Depending on the MIME type of the desired media content, either Steps B.2- B.4 or Steps B.5- B.8 are executed.

Step B.2: The system recognizes the MIME type in $intent_y$ and starts an image viewer, the Camera application.

Step B.3: If the image data source refers to content://drm/images/<_id> URI, the image is DRM protected. The Camera application retrieves the image content via a FileDescriptor to open a file stream through the DrmProvider. For DRM-free image, the file://sdcard/<FileName> URI is accessible and this step is skipped.

Step B.4: The Camera application displays the image.

Step B.5: The system recognizes the MIME type in $intent_y$ and then starts an audio player, the Music application.

Step B.6: The Music application simply acts as a MediaPlayer controller to start the MediaPlayer service to play the audio file.

Step B.7: Similar to the Camera application at Step B.3, the MediaPlayer retrieves both DRM protected and DRM-free audio files. If the audio data source refers to content://

drm/audio/<_id> URI, the audio is DRM protected. The MediaPlayer application retrieves the audio content via FileDescriptor which opens a file stream through the DrmProvider. For DRM-free audio file, the file://sdcard/<FileName> URI is accessible and this step is skipped.

Step B.8: The MediaPlayer service plays the audio in background.

Note that the sandbox on a rooted phone is accessible and the DRM protection is broken. Likewise, we can easily pull out all files in the sandbox by the ADB tool on the proposed prototype due to accessible /data partition.

On commercial phones, although we cannot ensure the implementation in details, the drm.db file does exist on these phones (as described at Step A.5a). If they follow the open source, the media contents can be copy once the users crack their phones.

IV. SOFTWARE INSTALLATION AND PROTECTION

As shown in Figure 1, the *Applications* layer provides a built-in software installation tool PackageInstaller. In the underlying *Application framework* layer, the PackageManager performs file verification and installation for the APK file.

In this section, we describe the message flow of built-in software installation procedure. Then, we investigate the software protection methods of both Android Market and SlideMe.

A. Installation Procedure

Figure 4 shows the message flow of installing an Android application with the following steps:

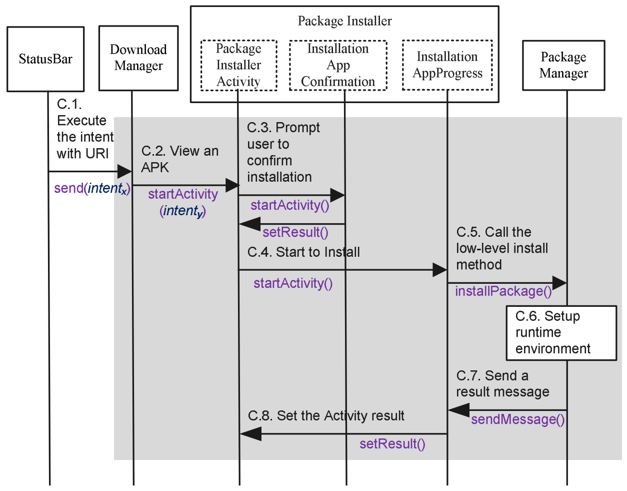

Package Installer
StatusBar
Download Manager
Package Installer Activity
Installation App Confirmation
Installation AppProgress
Package Manager
C.1. Execute the intent with URI
send($intent_x$)
C.2. View an APK
startActivity ($intent_y$)
C.3. Prompt user to confirm installation
startActivity()
setResult()
C.4. Start to Install
startActivity()
C.5. Call the low-level install method
installPackage()
C.6. Setup runtime environment
C.7. Send a result message
sendMessage()
C.8. Set the Activity result
setResult()

Figure 4. Message Flow for Software Installation

Step C.1: The user clicks on the icon in the status bar to pick up an APK file. This step is similar to Step B1 in Figure 3.

Step C.2: The DownloadReceiver in DownloadManager queries the MIME type and the file storage URI file://sdcard/<FileName>. Then the DownloadReceiver initializes the intent *intent*$_y$ and induces the system to start PackageInstallerActivity, which belongs to the PackageInstaller package.

According to the declarion in AndroidManifest.xml of the PackageInstaller package, the PackageInstaller should also accept to install APK through URI with content: scheme. In our observation, however, the PackageInstaller in Cupcake implementation does not support the APK data source through content provider mechanism.

Step C.3: The PackageInstallerActivity starts the activity InstallAppConfirmation to prompt the user to confirm the installation acceptance for the APK file. At this step, the InstallAppConfirmation shows the permission requirements of the APK file to the user. Accepting the installation implies to grant the permission for the application package.

Step C.4: The PackageInstallerActivity starts the activity InstallationAppProgress to activate the installation procedure and display an installation progress bar.

Step C.5: The InstallationAppProgress calls the static method PackageManager.installPackage() to install the APK.

A general application is unable to directly call the PackageManager.installPackage() method. As an alternative, the application should initialize an intent with a given APK file resident in the SD card or its sandbox, and then the built-in PackageInstaller application will be invoked by the system. This restriction prevents spyware applications from installing package without noticing the user.

Step C.6: The PackageManager performs underlying installation procedure, including constructing the sandbox in /data/data/<PackageName> and copying the APK file to the /data/app/<PackageName>.apk. If the package name is found duplicate with another existing application, the installation will fail. Note that the destination file name is different from the source file file://sdcard/<FileName>, but the file content is identical.

Steps C.7 and C.8: The PackageManager feeds an installation result to the InstallationAppProgress, which returns to the PackageInstallerActivity when the installation completes.

Before the installation procedure is executied, an APK is stored in the SD card as file://sdcard/<FileName>, which can be accessed by anyone. To protect the APK, one solution is to install this APK through content provider mechanism. Specifically, one can develop an application to serve the APK content to PackageInstaller and prevent the APK content to be stored in the file system (including the SD card), and therefore avoid unauthorized access. However, we found that PackageInstaller does not support the content: scheme in Cupcake (as described at Step C.2).

After the APK file installed, the APK file is installed in /data/app folder where the ADB tool can pull out the <PackageName>.apk from both our prototype and commercial phones. Although no one can identify the file name using /bin/ls command on commercial phones, he/she can use DDMS to observe a debug-level log that shows the file path /data/app/<PackageName>.apk. This log is generated at the moment of installing the APK file (at Step C.6) by PackageManager. Consequently, one can pull out the APK files from commercial phones and install them into other phones. We have not found any solution to this problem, and there may be a way of copying the installed APK file from one phone and distribute/install to other phones.

B. *Software Protection by Android Market*

Android Market [9] developed by Google is not included in the Android open source and only available on commercial phones with Google's license.

The Android Market application includes download and installation features. Downloading APK files from Android Market doesn't need an SD card. Therefore, the download problem observed in Section III-A is solved.

Without any additional code, software developers can enable the Market protection provided by the Android Market server [10]. After installing an APK file with Market protection, the destination file is located in /data/app-private folder, where the ADB tool cannot pull from the commercial phones. However, the Market protection cannot solve the problem on a rooted phone because it allows users to access the whole file system including the /data/app-private folder [11].

C. *Software Protection by SlideMe*

A third-party software store SlideMe [12] uses the Android built-in download and installation procedures mentioned in Section III-A and IV-A. This approach assists the software developers to involve the protection in their code.

SlideMe provides a Java library SlideLock to facilitate the developers in protecting their applications. A developer must define a SlideLock Key string associated with an application. During execution time, the application with SlideLock library sends back the Android phone information via an HTTPS connection to the SlideMe server. Then the SlideMe server checks the authority according to the SlideLock Key and the Android phone information [13].

The application holds and waits for the result in the HTTPS response. The SlideLock does not interrupt the execution of application when the SlideMe server is unreachable. One may purposely fail the connection attempt on his/her rooted phone by assigning an invalid IP address to the SlideMe server slideme.org in the Linux configuration file /etc/hosts, which is symbol link to the read-only /system partition. Rooted phones and our Dream prototype can remount the /system partition to be writable and then modify the /etc/hosts file.

V. CONCLUSION

We developed a prototype based on the Android Cupcake source code and analyzed the OMA DRM Forward-Lock protection for image and audio files. It is observed that the media content files can be pulled out in Android Cupcake. If a commercial phone utilizes the implementation of the Android Cupcake source code, the protection is cracked once the user "upgrades" the commercial phone to the rooted phone.

Software installation and protection were also studied. We illustrated the message flows of file download and software installation. The downloaded files are accessible in the SD card, and the installed software could also be pulled to PC easily. The Android Market and SlideMe are two typical solutions for this problem. The Android Market proprietarily implements download and installation procedures on some commercial phones. The SlideMe assists the developers to involve the protection mechanism in their software. While they can protect software on commercial phones, one can cracks these protections on rooted phones from the Linux root. Table I summarizes the examined results in our study.

TABLE I. SUMMARY OF DRM AND SOFTWARE PROTECTION

Phone Type	**Successful Protection?**			
	OMA DRM Forward-Lock Protection	***Download Manager and Package Installer***	***Android Market Protection***	***SlideMe Protection***
Our Prototype	X	X	Not Available	X
Commercial Phone	O	X	O	O
Rooted Phone	X	X	X	X

O: Protection Success on the Phone. X: Protection Failure on the Phone

REFERENCES

[1] The Developer's Guide, http://developer.android.com/guide/index.html.

[2] SQLite Home Page, http://www.sqlite.org/.

[3] HTC – Products – HTC – Dream – Specification, http://www.htc.com/www/product/dream/specification.html.

[4] HTC – Products – HTC – Magic – Overview, http://www.htc.com/www/product/magic/overview.html.

[5] HTC – Products – HTC – Hero – Overview, http://www.htc.com/www/product/hero/overview.html.

[6] Android SDK – Android Developers, http://developer.android.com/sdk/.

[7] Android Open Source Project, http://source.android.com/download.

[8] Tethering via WiFi, http://forum.xda-developers.com/showthread.php?t=474470.

[9] Android Market, http://www.android.com/market/.

[10] Upload an Application – Developer Console – Android Marktet, http://market.android.com/publish/Home.

[11] Android Market DRM busted < 12 hrs, http://strazzere.com/blog/?p=185.

[12] SlideME – Android Community and Application Marketplace, http://slideme.org/.

[13] SlideMe / SlideLock released - does it really protect?, http://strazzere.com/blog/?p=177.