COMP343 Assignment 2: Security analysis of Android APK file format and associated technologies within the Android Operating System

Adam Carmichael
adam.carmichael@mq.edu.au
Department of Engineering
Macquarie University

Monday Jun 3, 2013

Contents

1 Availability

LaTeX sources, and a PDF of this document can be downloaded from:
sources: https://github.com/carneeki/comp343-assignment2/
PDF: https://github.com/carneeki/comp343-assignment2/blob/master/out/COMP343%20Assignment%202.pdf

2 Summary

Android APK File Format - a zip file format used to package applications onto an Android device such as a tablet, phone, or smart television / set top box.

APK files must have several specific directories and files, some of which are of interest as they maintain the security through cryptographic certificates and hash functions, others will only be mentioned in summary as they are of minimal interest in the context of this assignment.

This document does not cover exploits where by the default install permitted network operators to send SMS codes to reset the phone to a default state, or to install arbitrary software and attempts to focus on the APK File Format and the permissions and access control structure surrounding it.

3 Detailed Description

The APK file format contains the following tree layout:

```
.
+-- AndroidManifest.xml
+-- assets
+-- classes.dex
+-- META-INF
|     +-- CERT.RSA
|     +-- CERT.SF
|     +-- MANIFEST.MF
+-- res
|     +-- anim
|     +-- drawable
|     +-- layout
|     +-- xml
+-- resources.arsc
```

- `AndroidManifest.xml` is a compiled XML file and is not human readable, but it may be converted to XML using a tool such as `apktool`[1]. The `AndroidManifest.xml` file contains information such as the application name, and access rights (called permissions) that the application would like to use. These permissions are presented to the user when the user goes to install the application, and are also shown in the Google Play Store (and other markets) when the user can browse the application. A user can opt to not install an application if it uses a permission the user is not willing to grant (such as send an SMS).
- `assets` are things such as sounds that the application may need to use.
- `classes.dex` Android operates a virtual machine called Dalvik which shares many similarities with Java. A `.dex` file is a Dalvik executable can be thought of as similar to a `.class` file in Java.

[1] https://code.google.com/p/android-apktool/

- `META-INF` is a directory which contains the following three files of interest:
  - `MANIFEST.MF` a listing of all files present in the APK and a digest of their content (default hash algorithm is SHA-1 represented as a base64 value).
  - `CERT.RSA` application certificate. Standard `openssl` commands from the Linux command line can be used to read and verify these certificates.
  - `CERT.SF` a `SHA-1` hash of each entry in the `MANIFEST.MF` file, which is then signed using the application certificate, `CERT.RSA`.

  Typically, in signing an `APK`, the compiled application is SHA-1 hashed (`MANIFEST.MF`), and then the hash entries are hashed as second time, signed using `CERT.RSA` to produce the signature file `CERT.SF`.

When an application is to be installed, it is done using the Package Manager within the operating system, which places calls to check `CERT.SF`. It also includes the prompt to the user showing the permissions required from `AndroidManifest.xml`, and this presents the weakest point of the security system - the end user.

4 Assets at Risk

Three assets that may be placed at risk are

- Phones
- Tablets
- Digital TVs / set top boxes

If one uses their phone or tablet for banking, and an attacker installs malicious software onto the victim's device, it is possible that banking data could be exposed. By the same token, confidential email data may be leaked, or, more simply, a phone could be turned into a zombie bot for sending SMS spam messages.
Information leakage is possible despite software running in sandboxed processes. One possible example of defeating the sandbox is to install a replacement keyboard - thereby reading all keystrokes.

5 Threat Surface

Three possible attack vectors in which an adversary may comprimise a device:

- Social engineering - asking a user to install a piece of malicious software under some false pretense.
- Physically obtaining the handset in an unlocked state and installing a piece of malicious software, perhaps in a socially engineered attempt to 'fix a problem'.
- Remote attack on running daemons - some handsets run the Dropbear SSH daemon; if this daemon runs as root and were to be comprimised (say buffer overflow), packages can be installed into the system partition and run at the next reboot.

6 Vulnerabilities

In two of these three possible attack vectors, the attacker is required to either have the device, or to trick the end user into installing software onto the device. Users must exercise discretion when installing software, and be cautious who they hand their unlocked devices to.
The third vulnerability is the most interesting, as it requires exploiting bugs in software that may not be running on the device. Attacks can be mitigated by turning off daemons that are not necessary (most users are not running Dropbear SSH by default).
Versions of Android prior to 4.0 can be plugged into a desktop computer and enter a debug mode whereby packages can be installed over a USB cable (this is also available over IP) using `ADB`[2] without a prompt from the user. This is typically turned off by default. More recently, on screen prompts to show the application permissions became the norm, however most recently, a handshake procedure between the handset and the computer similar to the SSH host key authentication process takes place. If a PC is not trusted, `ADB` refuses the connection and software cannot be installed.

7 Controls

The `AndroidManifest.xml` file presents a list of requested permissions that an APK requires to run, and at install time the list of permissions is shown to the user so they can determine if the end user should install the software or not. **Figure 1** shows the permissions listed in the `AndroidManifest.xml` for an application called **Remote for VLC**. This application is asking to read the status of the phone and the phone's identity, have full network communication, view wifi networks, and control the vibrate functionality. An observant user might ask *"Why does this application need to read the phone's identity?*.

This leads to a question concerning the trust of the manifest - so far the onus is placed on the developer to report the permissions that the application needs to run. What if a developer were to **not** list a certain permission (such as requesting superuser access?). **GetRIL** is an application which requires the superuser permission, however, it seems that the developer forgot to add this to the manifest. The operating system intercepts the request and displays a very stark warning as seen in **Figure 2**.

Signed certificate files are used to certify whether software is legitimate or not, and Android disallows unsigned software to be installed by default. Installing unsigned files requires delving into the system settings, and a warning message advising the user of unsafe practices appears when disabling the signature checks. **Figure 3** shows the checkbox for enabling the installation from unknown sources. **Figure 4** shows one of the VeriSign Certificate Authorities certificates which are installed by default, as well as a big *disable* button enabling a user to cease trusting the CA.

Host key authentication prevents a device being plugged into a PC and an APK

[2] Android Debug Bridge

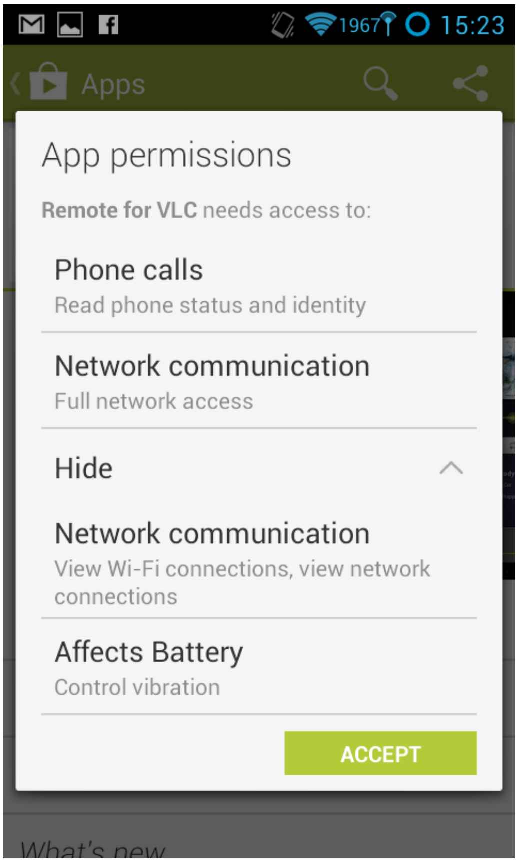

Figure 1: Install time permissions for *Remote for VLC* as listed by `AndroidManifest.xml`.

being pushed to the device because unrecognised devices are refused connection. Android 2.2 and up permits *corporate administrators* to lock down devices such that only specific staff may install software, however this requires tight integration with the Google Apps ecosystem which the author is still only just setting up, and without a secondary device with which to experiment, the author is reluctant to experiment with features that could lock him out of his own device. Google describes the Device Administration API as being able to[3]:

- Implement policies
  - Password strength
  - Require storage encryption
  - Disable Camera
- Security applications that do remote wipe
- Device management services and applications

[3]http://developer.android.com/guide/topics/admin/device-admin.html

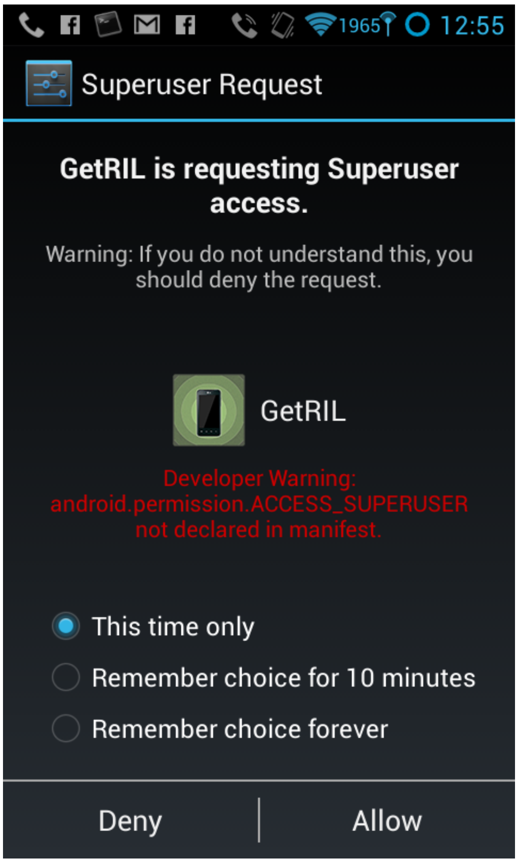

Figure 2: Superuser permission was not declared in the `AndroidManifest.xml` in *GetRIL*.

8 Risk Evaluation

In the default state, the greatest risk comes from the physical security of a device. The default state of a device is that the device itself has no PIN, passphrase or pattern to lock / unlock the device, however also are typically not running any daemons that are likely to be exploitable.

There are quite reasonable safeguards to prevent a remote attacker from installing software to exploit a device, and most users are reasonably wary enough to not physically hand a phone over to someone who is untrusted.

9 Conclusion

Social engineering an attack on a user to install malicious software on a device remains the easiest method of attack. Even so, the permissions at install time will alert the user as to what the software has the capability of doing. Taking the earlier example of an application to act as replacement keyboard, a user **should** ask *"Why does a keyboard need Internet access? Why does it need to*

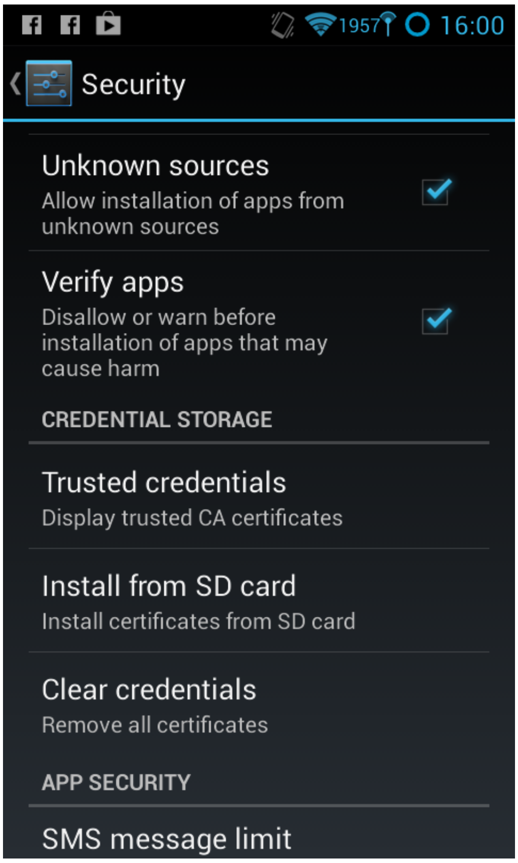

Figure 3: Nested deep in the security settings is the option to allow installation of applications from *unknown sources*.

send an SMS?". Other applications, such as launchers which download news and social media, as well as having dialers to make dialing a phone call (say you're driving a car) easier make asking these questions somewhat harder.

Anecdotal evidence suggests that most users do not actually read these permissions and usually confirm installation.

For a brand new device, simply exercising caution and putting a passphrase or PIN to unlock the device should suffice for most users.

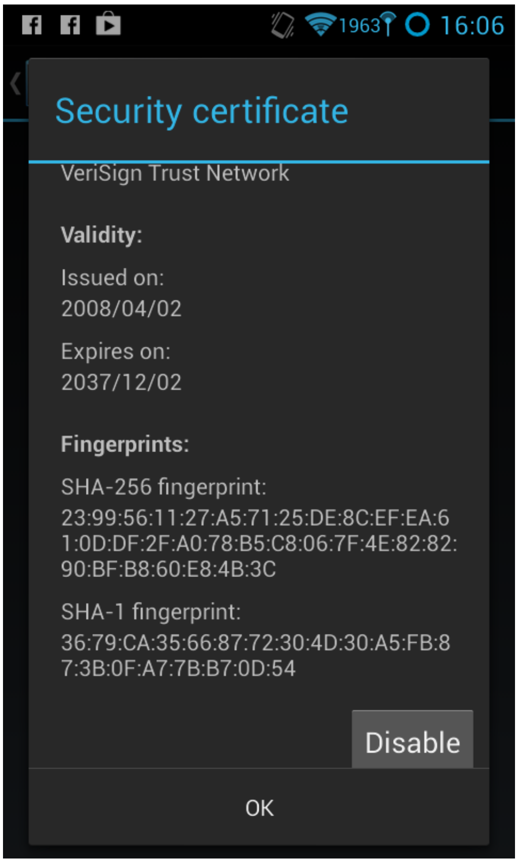

Figure 4: One of VeriSign's CA certificates, and SHA-1 and SHA-256 fingerprints, along with a button to disable should the user no longer trust this CA.