Android Security Architecture

In Chapter 2, we looked at a simple example of how we can protect information using encryption. However, that example did not make use of Android's built-in security and permissions architecture. In this chapter, we will take a look at what Android is able to offer the developer and end user with regard to security. We will also look at some direct attacks that can take place on applications and how to take the necessary safeguards to minimize the loss of private data.

The Android platform has several mechanisms that control the security of the system and applications, and it attempts to ensure application isolation and compartmentalization at every stage. Each process within Android runs with its own set of privileges, and no other application is able to access this application or its data without explicit permissions provided by the end user. Even though Android exposes a large number of APIs to the developer, we cannot use all of these APIs without requiring the end user to grant access.

Revisiting the System Architecture

Let’s start by looking at the Android architecture once more. We covered the Android system architecture in Chapter 1, where you will recall that each process runs in its own isolated environment. There is no interaction possible between applications unless otherwise explicitly permitted. One of the mechanisms where such interaction is possible is by using permissions. Again in Chapter 1, we looked at a simple example of how we needed to have the RECORD_AUDIO permission set, so that our application can make use of the device's microphone. In this chapter, we will look at the permissions architecture in a little bit more detail (see Figure 3-1).
Figure 3-1 depicts a simpler version of the Android architecture than the one presented in Chapter 2; specifically, this figure focuses more on the applications themselves.

As we saw previously, Android applications will execute on the Dalvik virtual machine (DVM). The DVM is where the bytecode, or the most fundamental blocks of code, will execute. It is analogous to the Java Virtual Machine (JVM) that exists on personal computers and servers today. As depicted in Figure 3-1, each application—even a built-in system application—will execute in its own instance of the Dalvik VM. In other words, it operates inside a walled garden of sorts, with no outside interaction among other applications, unless explicitly permitted. Since starting up individual virtual machines can be time consuming and could increase the latency between application launch and startup, Android relies on a preloading mechanism to speed up the process. The process, known as Zygote, serves two functions: it acts first as a launch pad for new applications; and second, as a repository of live core libraries to which all applications can refer during their life cycles.

The Zygote process takes care of starting up a virtual machine instance and preloading and pre-initializing any core library classes that the virtual machine requires. Then, it waits to receive a signal for an application startup. The Zygote process is started up at boot time and works in a manner similar to a queue. Any Android device will always have one main Zygote process running. When the Android Activity Manager receives a command to start an application, it calls up the virtual machine instance that is part of the Zygote process. Once this instance is used to launch the application, a new one is forked to take its place. The next application that is started up will use this new Zygote process, and so on.

The repository part of the Zygote process will always make the set of core libraries available to applications throughout their life cycles. Figure 3-2 shows how multiple applications make use of the main Zygote process’s repository of core libraries.