第九章 Android 安全访问机制

Android 是一个多进程系统，在这个系统中，应用程序（或者系统的部分）会在自己的进程中运行。系统和应用之间的安全性通过 Linux 的 facilities（工具，功能）在进程级别来强制实现的，比如会给应用程序分配 user ID 和 Group ID。更细化的安全特性是通过"Permission"机制对特定的进程的特定的操作进行限制，而"per-URI permissions"可以对获取特定数据的 access 专门权限进行限制。 所以，应用程序之间的一般是不可以互相访问的，但是 anroid 提供了一种 permission 机制，用于应用程序之间数据和功能的安全访问。

9.1 安全架构

Android 安全架构中一个中心思想就是：应用程序在默认的情况下不可以执行任何对其他应用程序，系统或者用户带来负面影响的操作。这包括读或写用户的私有数据（如联系人数据或 email 数据），读或写另一个应用程序的文件，网络连接，保持设备处于非睡眠状态。

一个应用程序的进程就是一个安全的沙盒。它不能干扰其它应用程序，除非显式地声明了“permissions”，以便它能够获取基本沙盒所不具备的额外的能力。它请求的这些权限“permissions”可以被各种各样的操作处理，如自动允许该权限或者通过用户提示或者证书来禁止该权限。应用程序 需要的那些“permissions”是静态的在程序中声明，所以他们会在程序安装时被知晓，并不会再改变。

所有的 Android 应用程序（。apk 文件）必须用证书进行签名认证，而这个证书的私钥是由开发者保有的。该证书可以用以识别应用程序的作者。该证书也不需要 CA 签名认证（注：CA 就是一个第三方的证书认证机构，如 verisign 等）。Android 应用程序允许而且一般也都是使用 self- signed 证书（即自签名证书）。证书是用于在应用程序之间建立信任关系，而不是用于控制程序是否可以安装。签名影响安全性的最重要的方式是通过决定谁可以进入基于签名的 permisssions，以及谁可以 share 用户 IDs。

9.2 用户 IDs 和文件存取

每一个 Android 应用程序（。apk 文件）都会在安装时就分配一个独有的 Linux 用户 ID，这就为它建立了一个沙盒，使其不能与其他应用程序进行接触（也不会让其它应用程序接触它）。这个用户 ID 会在安装时分配给它，并在该设备上一直保持同一个数值。

由于安全性限制措施是发生在进程级，所以两个 package 中的代码不会运行在同一个进程当中，他们要作为不同的 Linux 用户出现。我们可以通过 使用 AndroidManifest。xml 文件中的 manifest 标签中的 sharedUserId 属性，来使不同的 package 共用同一个用户 ID。通过这种方式，这两个 package 就会被认为是同一个应用程序，拥有同一个用户 ID（实际不一定），并且拥有同样的文件存取权限。注意：为了保持安全，只有当两个应用程序被同一个签名签署的时候（并且请求了同一个 sharedUserId）才会被分配同样的用户 ID。

所有存储在应用程序中的数据都会赋予一个属性——该应用程序的用户 ID，这使得其他 package 无法访问这些数据。当通过这些方法 getSharedPreferences(String, int)，openFileOutput(String, int) 或者 openOrCreateDatabase(String, int, SQLiteDatabase.CursorFactory) 来创建一个新文件时，你可以通过使用 MODE_WORLD_READABLE and/or MODE_WORLD_WRITEABLE 标志位来设置是否允许其他 package 来访问读写这个文件。当设置这些标志位时，该文件仍然属于该应用程序，但是它的 global read and/or write 权限已经被设置，使得它对于其他任何应用程序都是可见的。

例如：APK A 和 APK B 都是 C 公司的产品，那么如果用户从 APK A 中登陆成功。那么打开 APK B 的时候就不用再次登陆。 具体实现就是 A 和 B 设置成同一个 User ID：

packagename APK A 的 AndroidManifest：

```
<manifest xmlns:android="http://schemas。android。com/apk/res/android"
    package="com.Android.demo.a1"
    android:sharedUserId="com.c">
```

packagename APK A 的 AndroidManifest：

```
<manifest xmlns:android="http://schemas。android。com/apk/res/android"
    package="com.Android.demo.b1"
    android:sharedUserId="com.c">
```

这个"com.c" 就是 user id。APK B 就可以像打开本地数据库那样打开 APK A 中的数据库了。APK A 把登陆信息存放在 A 的数据目录下面。APK B 每次启动的时候读取 APK A 下面的数据库判断是否已经登陆：

APK B 中通过 A 的 package name 就可以得到 A 的 packagecontext：

```
friendContext = this.createPackageContext( "com.android.demo.a1",
                                           Context,CONTEXT_IGNORE_SECURITY);
```

通过这个 context 就可以直接打开数据库。

9.3 权限(permission)

权限用来描述是否拥有做某件事的权力。Android 系统中权限分为普通级别(Normal)，危险级别(dangerous)，签名级别(signature)和系统/签名级别(signature or system)。系统中所有预定义的权限根据作用的不同，分别属于不同的级别。

对于普通和危险级别的权限，我们称之为低级权限，应用申请即授予。其他两级权限，我们称之为高级权限或系统权限，应用拥有 platform 级别的认证才能申请。当应用试图在没有权限的情况下做受限操作，应用将被系统杀掉以警示。

系统应用可以使用任何权限。权限的声明者可无条件使用该权限。

目前 Android 系统定义了许多权限，通过 SDK 文档用户可以查询到哪些操作需要哪些权限，然后按需申请。

为了执行你自己的权限，你必须首先在你的 AndroidManifest.xml 中使用一个或多个

<permission> 标签声明。例如，一个应用程序想用控制谁能启动一个 activities，它可以为声明一个做这个操作的许可，如下：

```
<manifest xmlns:android="http://schemas。android。com/apk/res/android"
   package="com.me.app.myapp" >
   <permission android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"
      android:label="@string/permlab_deadlyActivity"
      android:description="@string/permdesc_deadlyActivity"
      android:permissionGroup="android.permission-group.COST_MONEY"
      android:protectionLevel="dangerous" />
</manifest>
```

9.4 使用权限(uses-permission)

应用需要的权限应当在 users-permission 属性中申请，所申请的权限应当被系统或某个应用所定义，否则视为无效申请。同时，使用权限的申请需要遵循权限授予条件，非 platform 认证的应用无法申请高级权限。

所以，程序间访问权限大致分为两种：

- 第一种低级点的（permission 的 protectlevel 属性为 normal 或者 dangerous），其调用者 apk 只需声明<uses-permission>即可拥有其 permission。
- 第二种高级点的（permission 的 protectlevel 属性为 signature 或者 signatureorsystem），其调用者 apk 就需要和被调用的 apk 一样拥有相同的 signature。

若想拥有使用权限，必须在 AndroidManifest.xml 文件中包含一个或更多的<uses-permission>标签来声明此权限。

例如低级权限需要监听来自 SMS 消息的应用程序将要指定如下内容：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.android.app.myapp" >
   <uses-permission android:name="android.permission.RECEIVE_SMS" />
</manifest>
```

应用程序安装的时候，应用程序请求的 permissions 是通过 package installer 来批准获取的。package installer 是通过检查该应用程序的签名来确定是否给予该程序 request 的权限。在用户使用过程中不会去检查权限，也就是说要么在安装的时候就批准该权限，使其按照设计可以使用该权限；要么就不批准，这样用户也就根本无法使用该 feature，也不会有任何提示告知用户尝试失败。

例如高级权限用有 system 级别权限设定的 api 时，需要使其 apk 拥有 system 权限。比如在 android 的 API 中有供给 SystemClock.setCurrentTimeMillis()函数来修改系统时间。有两个方法：

- 第一个方法简单点，不过需要在 Android 系统源码的情况下用 make 来编译：
  1. 在应用程序的 AndroidManifest.xml 中的 manifest 节点中插手 android:sharedUserId="android.uid.system"这个属性。
  2. 修改 Android.mk 文件，插手 LOCAL_CERTIFICATE := platform 这一行
  3. 使用 mm 命令来编译，生成的 apk 就有修改系统时间的职权范围了。

- 第 2 个方法麻烦点，不外不消开虚拟机跑到源码情况下用 make 来编译：
  1. 同上，插手 android:sharedUserId="android.uid.system"这个属性。
  2. 使用 eclipse 编译出 apk 文件，但是这个 apk 文件是不能用的。
  3. 使用针系统的 platform 密码钥匙来从头给 apk 文件签名。signapk platform.x509.pem platform.pk8 input.apk output.apk

9.5 自定义 Permission

Android 系统定义的权限可以在 Manifest.permission 中找到。任何一个程序都可以定义并强制执行自己独有的 permissions，因此 Manifest.permission 中定义的 permissions 并不是一个完整的列表（即能有自定义的 permissions)。

一个特定的 permission 可能会在程序操作的很多地方都被强制实施：
- 当系统有来电的时候，用以阻止程序执行其它功能。
- 当启动一个 activity 的时候，会阻止应用程序启动其它应用的 Acitivity。
- 在发送和接收广播的时候，去控制谁可以接收你的广播或谁可以发送广播给你。
- 当进入并操作一个 content provider 的时候。
- 当绑定或开始一个 service 的时候。

9.6 组件权限

通过 AndroidManifest.xml 文件可以设置高级权限，以限制访问系统的所有组件或者使用应用程序。所有的这些请求都包含在你所需要的组件中的 android:permission 属性，命名这个权限可以控制访问此组件。

- Activity 权限 (使用 <activity> 标签) 限制能够启动与 Activity 权限相关联的组件或应用程序。在 Context.startActivity() 和 Activity.startActivityForResult() 期间检查；
- Service 权限(应用 <service> 标签)限制启动、绑定或启动和绑定关联服务的组件或应用程序。此权限在 Context.startService(), Context.stopService() 和 Context.bindService() 期间要经过检查；
- BroadcastReceiver 权限(应用 <receiver> 标签)限制能够为相关联的接收者发送广播的组件或应用程序。在 Context.sendBroadcast() 返回后此权限将被检查，同时系统设法将广播递送至相关接收者。因此，权限失败将会导致抛回给调用者一个异常；它将不能递送到目的地。在相同方式下，可以使 Context.registerReceiver() 支持一个权限，使其控制能够递送广播至已登记节目接收者的组件或应用程序。其它的，当调用 Context.sendBroadcast() 以限制能够被允许接收广播的广播接收者对象一个权限（见下文）。
- ContentProvider 权限(使用 <provider> 标签)用于限制能够访问 ContentProvider 中的数据的组件或应用程序。

如果调用者没有请求权限，那么会为调用抛出一个安全异常(SecurityException)。在所有这些情况下，一个 SecurityException 异常从一个调用者那里抛出时不会存储请求权限结果。

9.7 发送广播时支持权限

当发送一个广播时你能总指定一个请求权限，此权限除了权限执行外，其它能发送 Intent 到一个已注册的 BroadcastReceiver 的权限均可以。通过调用 Context.sendBroadcast()及一些权限字符串，为了接收你的广播，你请求一个接收器应用程序必须持有那个权限。注意，接收者和广播者都能够请求一个权限。当这样的事发生了，对于 Intent 来说，这两个权限检查都必须通过，为了交付到共同的目的地。

9.8 其它权限支持

在调用 service 的过程中可以设置任意的 fine-grained permissions（更为细化的权限）。这是通过 Context.checkCallingPermission()方法来完成的。使用一个想得到的 permission string 来进行呼叫，然后当该权限获批的时候可以返回给呼叫方一个 Integer（没有获批也会返回一个 Integer）。需要注意的是这种情况只能发生在来自另一个进程的呼叫，通常是一个 service 发布的 IDL 接口或者是其他方式提供给其他的进程。

Android 提供了很多其他的方式用于检查 permissions。如果你有另一个进程的 pid，你就可以通过 Context 的方法 Context.checkPermission(String, int, int)去针对那个 pid 去检查 permission。如果你有另一个应用程序的 package name，你可以直接用 PackageManager 的方法 PackageManager.checkPermission(String, String) 来确定该 package 是否已经拥有了相应的权限。

9.9 URI 权限

到目前为止我们讨论的标准的 permission 系统对于 content provider 来说是不够的。一个 content provider 可能想保护它的读写权限，而同时与它对应的直属客户端也需要将特定的 URI 传递给其它应用程序，以便其它应用程序对该 URI 进行操作。一个典型的例子就是邮件程序处理带有附件的邮件。进入邮件需要使用 permission 来保护，因为这些是敏感的用户数据。然而，如果有一个指向图片附件的 URI 需要传递给图片浏览器，那个图片浏览器是不会有访问附件的权利的，因为他不可能拥有所有的邮件的访问权限。

针对这个问题的解决方案就是 per-URI permission：当启动一个 activity 或者给一个 activity 返回结果的时候，呼叫方可以设置 Intent.FLAG_GRANT_READ_URI_PERMISSION 和/或 Intent.FLAG_GRANT_WRITE_URI_PERMISSION . 这会使接收该 intent 的 activity 获取到进入该 Intent 指定的 URI 的权限，而不论它是否有权限进入该 intent 对应的 content provider。

这种机制允许一个通常的 capability-style 模型，这种模型是以用户交互（如打开一个附件，从列表中选择一个联系人）为驱动，特别获取 fine-grained permissions（更细粒化的权限）。这是一种减少不必要权限的重要方式，这种方式主要针对的就是那些和程序的行为直接相关的权限。

这些 URI permission 的获取需要 content provider（包含那些 URI）的配合。强烈推荐在 content

provider 中提供这种能力，并通过 android:grantUriPermissions 或者<grant-uri-permissions>标签来声明支持。

更多的信息可以参考 Context.grantUriPermission()， Context.revokeUriPermission() 和 Context.checkUriPermission() methods。

QA

1.拥有 signature 的权限是否可以不用声明<uses-permission>就能 access 带 normal 或 dangerous 权限设定的数据或功能？
只要 signature 相同，就算不显式声明<ues-permission>也能 access 设定了 normal 或 dangerous 权限设定的数据或功能。

2.若需要 system 级别权限使用系统 api（即使用 system 级别的签名），如何同时使用其他 signature 权限设定（即使用 signature 级别的签名）的其他 apk 的功能？
拥有 system 级别权限的使用者可以 access 其他普通 signature 权限声明设定过的功能。
所以，设定为拥有 system 级别权限即可。