Zhongguancun Standard

# 中　　关　　村　　标　　准

T/ZSA 3001.01-2016

# 企业移动智能终端应用开发、安装、运行管控机制指南

App developing, installing and running management and control mechanism guideline for enterprise mobile smart terminal

（2020 年 3 月修订）

2016-12-16 发布

中关村标准化协会　　发布

# 目　次

# 前　言

本标准依据GB/T1.1—2009《标准化工作导则 第1部分：标准的结构和编写》给出的规则起草。

本标准由中关村网络安全与信息化产业联盟企业移动计算工作组（EMCG）提出。

本标准由参与本标准制定的单位投票表决通过。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本标准主要起草单位：华为技术有限公司、联想集团、北京小米科技有限责任公司、奇虎360科技有限公司、北京元心科技有限公司、安天实验室、北京中油瑞飞信息技术有限责任公司、杭州安恒信息技术有限公司、江苏通付盾科技有限公司。

本标准的主要起草人：王克、王梓、刘长山、石晓宏、张建国、黄晟、潘宣辰、宋超。

# 引　言

随着 4G、5G 移动网络及智能移动终端技术快速发展，使用移动互联技术处理业务信息已成为政府提高办事效率，企业产业转型升级的重要手段，网络安全等级保护国家标准GB/T 22239-2019 对组织机构使用移动应用软件（App）开发、检测及签名等提出要求。为保障政企单位移动信息化建设发展，解决移动应用软件安全问题，需建立移动应用软件管控机制。

# 企业移动智能终端应用开发、安装、运行管控机制指南

## 1 范围

本标准规范了移动应用软件（App）开发准入、软件开发、软件签名及验证、软件审核、软件发布、软件安装及运行监管等过程。本标准适用于智能终端厂商、应用开发者、企业应用软件服务商实施企业移动应用软件（App）的生命期管理。

## 2 规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件，仅注日期的版本适用于本文件。凡是不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 22239-2019 网络安全等级保护基本要求

GB/T 25069-2010 信息安全技术 术语

GM/Z 0001-2013 密码术语

T/EMCG 001.1-2019 移动智能终端密码模块技术框架 第1部分：总则

## 3 术语和定义

GM/Z 0001-2013界定的以及下列术语和定义适用于本标准。

3.1

**移动智能终端 mobile smart terminal**

能够接入移动通信网，具有提供应用软件开发接口的开放操作系统，并能够安装和运行第三方移动应用软件的移动设备。包括手机、Pad，这些设备可以是市场通用型的，也可以是政企机构专用型的。

[T/EMCG 001.1-2019 5.移动智能终端]

3.2

**App**

英文application简写，移动应用软件。安装在移动智能终端上，能够利用移动智能终端设备上操作系统提供的开发接口，实现某项或某几项特定任务的应用程序，包括移动智能终端预置的应用程序，以及通过网站、应用商店等移动分发平台下载、安装和升级的应用程序。

3.3

**数字签名（密码签名）digital signature**

签名者使用私钥对待签名数据的杂凑值做密码运算得到的结果，该结果只能用签名者的公钥进行验证，用于确认待签名数据的完整性、签名者身份的真实性和签名行为的抗抵赖性。

[GM/Z 0001-2013 术语2.113]

3.4

**官方机构 official authority**

社会公认的组织，可以是政府机构，也可以是社会团体、企业、联盟、协会等实体。

3.5

**App 开发者 App developer**

企业移动智能终端应用软件开发组织。

3.6

**EMCG 企业 App 服务中心 enterprise software services ESS**

遵照 EMCG 标准建立的为企业移动终端提供软件管理的服务平台。

3.7

**软件市场 App 商店 software market App shop**

面向社会的移动终端软件商店。

3.8

**EMCG 终端 EMCG smart mobile terminal**

具有验证、安装官方机构签名 app 能力的移动智能终端。

3.9

**EMCG-App**

经官方机构颁发或授权数字证书签名的 App。

3.10

**App 测试证书 test certificate**

官方机构颁发的，用于开发者进行 APP 开发调试的数字证书。

3.11

**App 发布证书 publish certificate**

官方机构颁发的，用于开发者进行 APP 发布的数字证书。

3.12

**官方机构证书 official authority certificate**

官方机构数字证书，用于 EMCG-App 发布。

3.13

**App 内部发布证书 internal publish certificate**

官方机构颁发的，用于开发者进行APP企业内部发布的数字证书。

3.14

**企业内部 App 管理中心 internal App management centre**

App使用企业建立的用于企业App管理的系统平台。

3.15

**企业专用证书 enterprise special certificate**

社会各团体组织已经建立的CA管理的证书。

3.16

**授权证书 authorized certificate**

经官方机构批准授权（官方机构签名）的企业专用证书。

## 4 基本要求

企业移动计算App是指移动智能终端出厂后由用户下载安装使用的App，包括通用App、定制App、企业专用证书定制App。

## 4.1 通用 App 基本流程

通用App适用于企业使用的通用功能的App（如计算器、日历等）。通用App开发、安装、运行管控机制基本流程如图1所示。以下流程中App签名须使用国家密码管理部门核准的密码技术。

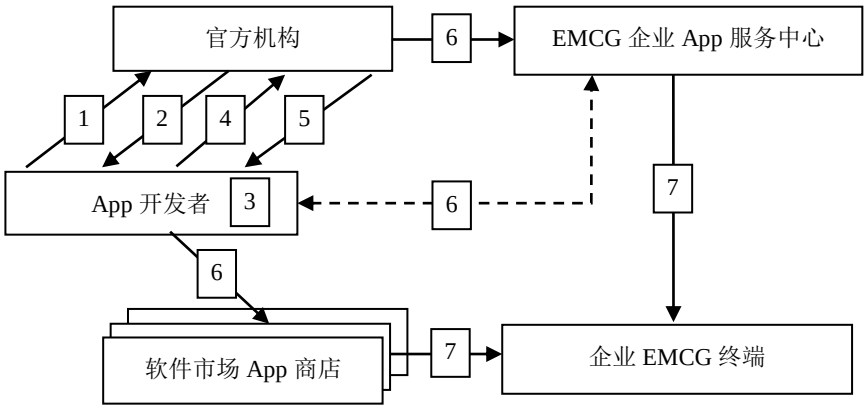



图 1 通用 App 开发、安装、运行管控机制基本流程

1) **开发申请** App 开发者向官方机构提交 App 开发申请。
2) **审核批准** 官方机构对开发者进行资质审核，审核通过，官方机构发给开发者 App 测试证书和 App 发布证书。开发者私钥自己保存，公钥发给官方机构。
3) **App 开发** 开发者开发企业移动 App 软件。开发过程中开发者使用 App 测试证书（对应的私钥）对开发的 App 进行密码签名，以便开发调试。参考附录 B EMCG 应用软件开发（安全）指南进行 App 开发。
4) **App 审核**
   ①开发者完成 App 开发后向官方机构提交，提交的 App 必须用发布证书（对应的私钥）进行密码签名。
   ②官方机构将按照附录 A EMCG 应用软件审核指南对提交的 App 进行审核。
5) **官方签名**
   ①官方机构使用官方机构证书（对应的私钥）对审核通过的 App 在开发者签名基础上再进行密码签名，生成 EMCG-App。
   ②将 EMCG-App 发给开发者。
6) **App 发布** 经官方机构密码签名的 EMCG-App 默认在 EMCG 应用管理系统中存档，开发者可以选择立即发布到 EMCG 企业软件服务中心，或者选择上架发布时间；上架后的 EMCG-App 也可以随时选择下架。开发者也可下载 EMCG-App 并自行发布到其他软件市场应用商店。
7) **App 安装运行** 企业 EMCG 终端可从软件市场 App 商店或 EMCG 企业 App 服务中心下载安装 EMCG-App；EMCG-App 安装和运行时，EMCG 终端进行 App 签名验证，只有经官方机构签名的 App 方可安装及运行。非 EMCG 终端也可从软件市场 App 商店下载 EMCG-App。

## 4.2 定制 App 基本流程

定制App适用于企业使用的专用功能的App。定制App开发、安装、运行管控机制基本流程如图2所示。以下流程中App签名须使用国家密码管理部门核准的密码技术。

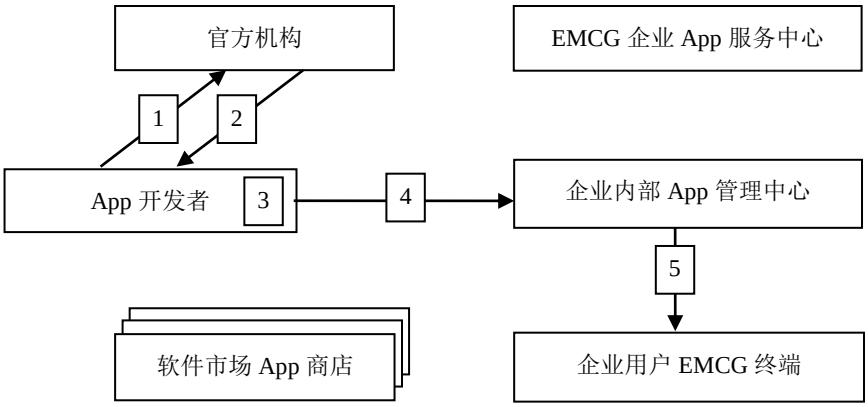



图 2 定制 App 开发、安装、运行管控机制基本流程

1) **开发申请** App 开发者向官方机构提交软件开发申请。
2) **审核批准** 官方机构对开发者进行资质审核，通过审核后官方机构发给开发者 App 测试证书和 App 内部发布证书。开发者私钥自己保存，公钥发给官方机构。
3) **App 开发** 开发者开发企业移动 App 软件。开发过程中开发者使用 App 测试证书（对应的私钥）对开发的 App 进行密码签名，以便开发调试。参考附录 B EMCG 应用软件开发（安全）指南进行 App 开发。
4) **App 签名发布**
   ①开发者完成软件开发后用 App 内部发布证书（对应的私钥）进行密码签名，生成 EMCG-App。
   ②开发者将 **EMCG**-App 发到企业内部 App 管理中心。
5) **App 安装运行** 企业用户 EMCG 终端从企业内部 App 管理中心下载安装 EMCG-App；EMCG-App 安装和运行时，EMCG 终端进行 App 签名验证，只有经官方机构证书签名的 App 方可安装及运行。

## 4.3 企业专用证书定制 App 基本流程

企业专用证书定制App适用于为已建立CA的企业开发本企业专用App。企业专用证书定制App开发、安装、运行管控机制基本流程如图3所示。以下流程中App签名须使用国家密码管理部门核准的密码技术。

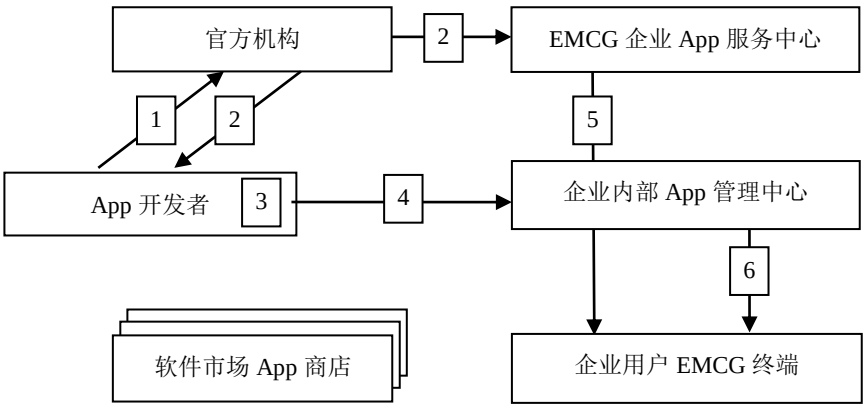



图 3 企业专用证书定制 App 开发、安装、运行管控机制基本流程

1) **开发申请** App 开发者向官方机构提交软件开发申请。
2) **批准及证书授权** 官方机构对开发者进行资质审核，通过审核后开发者申请证书许可，并将企业专用证书相关数据发送给官方机构，官方机构将企业专用证书（或由官方机构签名）放到 EMCG 企业 App 服务中心。
3) **App 开发** 开发者开发企业移动 App 软件。开发过程中开发者使用企业专用证书（对应的私钥）对开发的软件进行密码签名开发调试。参考附录 B EMCG 应用软件开发（安全）指南进行 App 开发。
4) **App 签名发布**
   ①开发者完成 App 开发后用企业专用证书（对应的私钥）进行密码签名，生成 EMCG-App。
   ②开发者将 EMCG-App 发到企业内部 App 管理中心。
5) **证书下载** 企业通过安全传输方式从 EMCG 企业 App 服务中心获得企业专用证书供企业 EMCG 终端 APP 签名验证。
6) **安装运行** 企业用户 EMCG 终端从企业内部 App 管理中心下载安装 EMCG-App；EMCG-App 安装和运行时，EMCG 终端进行 App 签名验证，只有经官方机构授权证书签名的 App 方可安装及运行。

# 附 录 A

（规范性附录）

# EMCG 应用审核指南

## A.1 条款与条件

本指南旨在帮助开发者的程序通过中关村网络安全与信息化产业联盟 EMCG 的认可，而不是为了修改和删除开发者任何其他协议中的条款。

## A.2 功能

A.2.1 有严重错误导致应用崩溃、闪退的程序将会被拒绝；

A.2.2 包含与开发者应用描述不符内容的应用将会被拒绝；

A.2.3 使用非公开 API 的应用将会被拒绝；

A.2.4 在用户不知情或者不通知用户的情况下，在后台进行任何与用户所使用的功能无关的操作的应用将会被拒绝；

A.2.5 提供不正确诊断或其他不准确数据的应用将会被拒绝；

A.2.6 与其他开发者应用外观、名称、内容、界面、功能等相似或混淆的应用可能会被拒绝；

A.2.7 用同一个功能模块开发大量相似产品，供用户反复下载的应用可能会被拒绝；

A.2.8 将主应用拆分为大量单一应用让用户反复下载，只接受主应用，其它单一应用可能会被拒绝；

A.2.9 应用内带有修改其他数据、存档等内容的功能；

A.2.10 应用内包含骚扰用户性质的模块，包含但不限于电话或短信轰炸机类应用；

A.2.11 存在非法窃取或上传用户隐私信息模块的应用将会被拒绝；

A.2.12 未经授权的情况下发布、披露、收集、查询他人的隐私和机密信息的应用将会被拒绝；

A.2.13 应用带有破解系统权限增加系统风险性功能的应用将会被拒绝；

A.2.14 应用中提供的服务于内容无法被认为是安全可靠的，可能存在导致用户的隐私受到侵犯或者财产受到损失的风险的应用可能会被拒绝；

A.2.15 伪造真实用户信息进行恶搞或者其他用途的应用将会被拒绝；

## A.3 安全性能

A.3.1 源码存在安全隐患的应用将会被拒绝；

A.3.2 不符合国家相关安全标准的应用程序将会被拒绝；

A.3.3 存在测试代码、数据和日志信息的应用将会被拒绝；

A.3.4 数据传输过程中存在安全威胁的应用将会被拒绝；

A.3.5 与应用关联的服务器层存在安全威胁的应用将会被拒绝；

## A.4 广告

A.4.1 内容主要是营销材料或者广告的程序将会被拒绝；

A.4.2 上传他人应用的破解版本、嵌入广告 SDK 因此而获利的应用将会别拒绝；

A.4.3 在应用中不合理地嵌入广告位，诱使用户误点而获利的应用将会被拒绝；

A.4.4 人工刷广告浏览量或广告点击率的应用将会被拒绝；

A.4.5 用不正当的方式提高应用的排名及好评率将会被拒绝；

## A.5 推送

A.5.1 首次推送通知或者要求推送通知之前未获得用户许可的应用将会被拒绝；

A.5.2 使用推送通知发送个人敏感信息或机密信息的程序将会被拒绝；

A.5.3 使用推送通知发送非请求消息，或用于钓鱼或群发垃圾信息用途的应用将会被拒绝；

A.5.4 应用程序不可使用推送通知发送广告、促销或任何类型的直销信息。

A.5.5 滥用推送通知给用户造成明显骚扰的应用将会被拒绝；

A.5.6 推送通知病毒、恶意软件、程序执行代码的应用将会被拒绝；

A.5.7 使用推送通知发送提醒信息的应用需要在通知栏提醒信息中包含应用名称或其它可被用户直接辨别来源的信息，否则应用将会被拒绝；

A.5.8 应用程序不能向使用推送通知服务的用户收取费用。

A.5.9 使用推送通知会过多利用 APN 服务的网络流量或给设备带来过度负担的程序将会被拒绝；

A.5.10　传送病毒、文件、计算机代码或程序，并且对 APN 服务的正常运行造成损害或中断的应用将会被拒绝；

## A.6 损害设备

A.6.1 怂恿用户以可能损害的方式使用移动设备的应用将会被拒绝；

A.6.2 快速耗光设备电量或产生过多热量的应用将会被拒绝；

A.6.3 能导致用户人身伤害的 APP 将会被拒绝；

## A.7 内容

A.7.1 包含讨论政治新闻、政治人物等政治敏感内容的应用将会被拒绝；

A.7.2 包含煽动性涉政新闻、散播谣言、扰乱社会稳定秩序内容的应用将会被拒绝；

A.7.3 含有暴力、色情素材、非法色情交易的应用将会被拒绝；

A.7.4 应用中出现人或动物被杀、致残以及枪击、刺伤、拷打等受伤情形的真实画面将会被拒绝；

A.7.5 鼓励过量饮酒、物质滥用等，或鼓励青少年饮酒、吸烟的应用；

A.7.6 包含描绘暴力或虐待儿童等内容的应用将会被拒绝；

A.7.7 包含诽谤、批判性内容、人身攻击性质、侵害他人或组织合法权益的应用内容将被拒绝；

A.7.8 游戏中出现的“敌人”不可指向一个特定种族、文化、一个真实存在的政府、企业或者其他任何现实中的实体；

A.7.9 对武器进行真实描述以怂恿非法使用或滥用这些武器的应用将会被拒绝；

A.7.10 涉及宗教、文化或族群的引用或评论包含诽谤或攻击性内容的应用，或会使特定群体遭受伤害或暴力的应用将被拒绝；

A.7.11 包含宣扬邪教、不健康文化等内容的应用将被拒绝；

A.7.12 涉及到宗教、民族的应用内容不应该具有煽动性言论或者破坏民族团结的内容；

A.7.13 包含令人反感、低俗或者激怒用户内容的应用将会被拒绝；

A.7.14 请求、促进或鼓励犯罪或明显鲁莽行为的应用将会被拒绝；

A.7.15 包括虚假、欺诈或误导性陈述的应用将会被拒绝

## A.8 隐私及权限

A.8.1 在未经用户事先许可，或未告知用户如何使用信息以及在何处使用信息的情况下，应用不能传输用户数据；未得用户同意，不得收集并上传用户敏感信息和用户行为统计等信息；

A.8.2 应公开用户数据的收集、使用及分享方式，数据应仅用于公开说明的用途，且须获得用户的

同意；

A.8.3 要求用户共享电子邮箱地址或者出生日期等私人信息才可使用其功能的应用将会被拒绝；

A.8.4 收集、传输以及分享未成年用户个人信息的应用程序必须遵守儿童隐私法律，并且必须包含隐私条款；

A.8.5 向终端用户或者任意第三方显示用户信息的应用将会被拒绝；

A.8.6 不得获取与软件无关的权限；

A.8.7 请求权限必须给用户提示授权。不得未得授权自动发短信、打电话、启动系统服务（蓝牙、GPS）、打开网络连接等。

### A.9 应用元数据

A.9.1 包含任何描述性文字，包含但不限于宣传口号、功能描述性文字的应用将会被拒绝；

A.9.2 应用标题、简介、描述、副标题中出现与应用本身功能、内容无关的其他应用或品牌关键词将会被拒绝；

A.9.3 应用名称、icon、截图与其他应用过度相似的可能会被拒绝；

A.9.4 不当使用或未经授权使用受版权保护的文学、戏剧、影视剧集、综艺节目等作品的应用将会被拒绝；

A.9.5 应用 icon 包含未经授权的角标将会被拒绝；

### A.10 特殊行业

A.10.1 包含彩票销售模块的应用必须由合法授权的应用开发者/公司发起；

A.10.2 包含股票交易的应用必须由合法授权的应用开发者/公司发起；

A.10.3 包含银行业务的应用必须由合法授权的应用开发者/公司发起；

A.10.4 包含理财产品销售模块的应用必须由合法授权的应用开发者/公司发起；

A.10.5 包含信用借贷类业务的应用必须由合法授权的应用开发者/公司发起；

A.10.6 包含医疗咨询、医药销售、医疗器材销售等模块的应用必须由合法授权的应用开发者/公司发起；

A.10.7 P2P 融资类应用、众筹类等涉及到用户资金财产安全的应用必须由合法授权的应用开发者/公司发起；

A.10.8　贵金属交易类应用必须由合法授权的应用开发者/公司发起；

A.10.9　移植、汉化、街机、热门开源类游戏需要出示原版授权，否则会被拒绝；

A.10.10 赌博类应用和游戏将会被拒绝；

A.10.11 包含可以向认证的慈善组织捐赠内容的应用必须是免费的。

## A.11 法律

A.11.1　应用必须遵守各地用户遵守的任何法律要求；

A.11.2　开发者有义务了解并遵守当地所有法律。

# 附 录 B

（资料性附录）

EMCG 应用软件开发（安全）指南

# 目 录

## B.1 Android 安全开发相关知识

本手册是关于 Android 应用开发的一系列安全建议，这里将讲述 Android 智能手机和平板电脑相关的通用安全设计和安全开发知识，在涉及后面的安全开发章节之前，我们建议你先熟悉下这个章节的内容。

### B.1.1 Android 应用安全

关于测试系统或应用的安全问题，业界已经有一种普遍被接受的思路。首先，我们得理解保护对象，我们称之为资产；接下来，我们要尽可能多的理解发生在资产上的攻击可能性，我们称之为威胁；最后，我们将测试并执行安全措施保护资产，抵御各种威胁，我们称之为安全措施。

上述提到的安全措施在这里我们把它叫做安全设计和安全开发，后面的章节会详细描述，本节内容，我们主要集中对资产和威胁进行解释说明。

#### B.1.1.1 资产-安全保护对象

在系统或应用里有两种保护对象：信息和功能。我们将它们称作信息资产和功能资产。信息资产是指那种只能被有权限的人访问或更改的信息，并且任何没有权限的人无法进行访问或更改。功能资产是指只能被有权限的人使用而其他人无法使用。

下面，我们将介绍在 Android 智能机和平板电脑里存在的信息资产类型和功能资产类型。当利用 Android 应用程序或 Android 智能机/平板电脑时，我们希望您能使用以下内容作为参考点来研究关于资产的内容。为了方便，我们将 Android 智能机/平板电脑简称为 Android 智能机。

##### B.1.1.1.1 Android 智能机的信息资产

下面表 B-1、表 B-2、表 B-3 列举了 Android 智能机包含的信息。恰当的保护是必要的，因为这些信息是个人信息、敏感信息或两者都有。

表 B-1 Android 智能手机管理的信息样例表

| 序号 | 信息 | 备注 |
|---|---|---|
| 1 | 电话号码 | 智能手机自身电话号码 |
| 2 | 通话记录 | 来电或去电的号码、时间和时间 |
| 3 | IMEI | 智能手机的设备 ID |
| 4 | IMSI | 国际移动用户识别码（IMSI：International Mobile Subscriber Identification Number） |
| 5 | 传感信息 | GPS，地理坐标、速率等 |
| 6 | 各种配置信息 | Wi-Fi 配置信息等 |

| 序号 | 信息 | 备注 |
|---|---|---|
| 7 | 账号信息 | 各种账号信息，认证信息等 |
| 8 | 多媒体数据 | 照片、视频、音乐、录音等 |
| 9 | ... | |

表 B-2 应用程序管理的信息样例表

| 序号 | 信息 | 备注 |
|---|---|---|
| 1 | Contacts 联系人 | 熟人等 |
| 2 | E-mail 地址 | 用户的 E-mail 地址 |
| 3 | E-mail 邮箱 | Content of incoming and outgoing e-mail, attachments, etc.<br>收件箱和发件箱、附件的内容等 |
| 4 | Web 收藏夹 | 收藏夹 |
| 5 | Web 浏览记录 | 浏览历史记录 |
| 6 | Calendar 日历 | Plans, to-do list, events, etc. 计划，任务列表，事件等 |
| 7 | 社交网络 | QQ、微信、微博、Facebook 等 |
| 8 | ... | |

表 B-3 联系人包含详细信息样例表

| 序号 | 信息 | 内容 |
|---|---|---|
| 1 | 电话号码 | 家庭电话、移动电话、传真、彩信等 |
| 2 | E-mail 地址 | 家庭 E-mail、工作 E-mail、移动电话 E-mail 等 |
| 3 | 照片 | 缩略图、大图片等 |
| 4 | 即使通讯 | QQ、微信、易信、飞信、Skype 等 |
| 5 | 昵称 | 首字母缩写、曾用名、昵称等 |
| 6 | 地址 | 国家、邮编、地区、区域、城镇、街道等 |
| 7 | 组成员关系 | 收藏联系人、家人、朋友、同事等 |
| 8 | 网站 | 博客、主页、FTP 服务器等 |
| 9 | 事件 | 生日、纪念日、其他 |
| 10 | 关系 | 配偶、子女、父亲、母亲、主管、助理、合伙人等 |
| 11 | SIP 网络电话 | 家庭、工作、其他等 |

直到现在，我们的焦点主要放在智能手机的用户信息，然而，应用也会处理其他重要信息，下图展示了一个应用内部信息的典型视图，被分为程序部分和数据部分。程序部分主要由应用开发者信息组成，数据部分则大部分是用户信息。由于应用的开发信息不想被用户访问，提供保护措施禁止用户访问或更改这类信息是非常重要的。

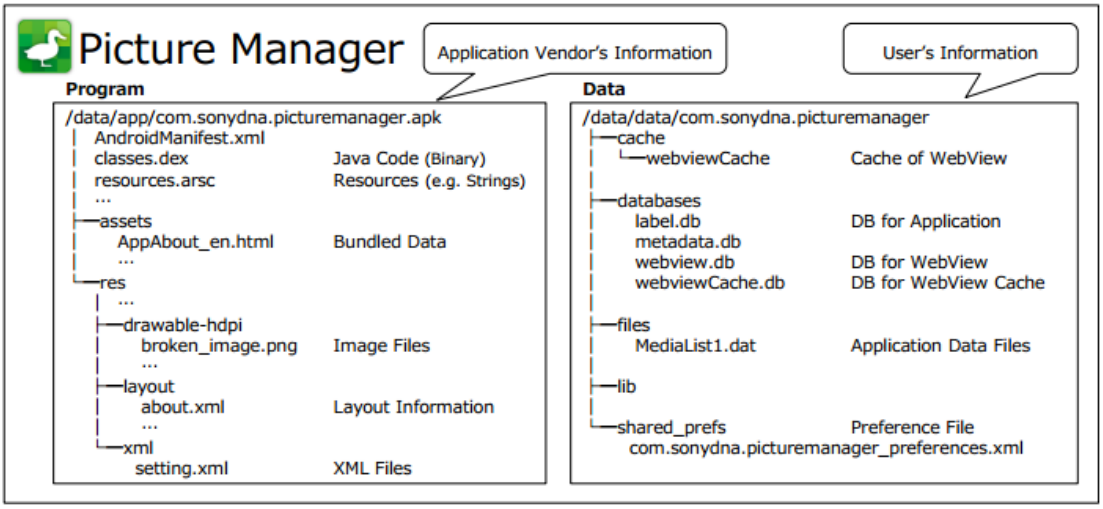



图 B-1 应用内部信息的典型视图

当创建一个 Android 应用时，非常重要的一件事是给图 B-1 中应用自身管理的信息采取合适的保护措施。然而，同等重要的是恰当地采取强健的安全策略保护 Android 智能手机自身包含的信息，同时也要保护被其他应用获取的信息，例如在表 B-1，表 B-2，表 B-3 中展示的信息。

#### B.1.1.1.2 Android 智能机的功能资产

表 B-4 展示了 Android OS 提供给应用的功能特性样例，当这些特性被恶意软件利用，意外的费用或隐私泄漏等损害将给用户带来严重损失。因此，恰当的保护措施应该作为信息资产的一个扩充被合理部布置。

表 B-4 Android OS 提供给应用的特性样例

| Function 功能 | 功能 |
|---|---|
| 发送和接收短信 | 相机 |
| 拔打电话 | 音量 |
| 网络通信 | 获取联系人列表和手机状态 |
| GPS | SD 卡 |
| 蓝牙通信 | 变更系统配置 |
| NFC 通信 | 读取日志数据 |
| 网络电话 (SIP) | 获取运行中的应用信息 |
| ... | ... |

除了 Android OS 提供给应用的功能之外，Android 应用的应用间通信组件也是功能资产的一部分。Android 应用可以允许其他应用通过访问他们内部组件来使用特性，我们称之为应用间通信。这是一个很方便的特性，然而，已经有很多例子，正是那些缺乏安全开发知识的开发人员，本应该只能被内部特定的应用使用的功能却错误地授权给其他应用访问。在设备本地还存在一些可能被恶意软件利用的应用功能，因此，非常有必要采取恰当的保护措施只允许合法应用访问这些功能。

### B.1.1.2 威胁-危害资产的各类攻击

在前面章节，我们谈论了关于 Android 智能机的资产，本章节，我们将讲解关于会对资产造成威胁的攻击。简而言之，对资产造成的威胁是指本应该没有权限的第三方可以访问、变更、删除或创建信息资产，或非法使用功能资产。这种直接地或间接地攻击那些资产的行为就叫做“威胁”。而且，犯做这些行为的恶意人员或应用被称之为威胁来源。恶意的攻击者和恶意软件是威胁源头，但并非威胁本身。我们定义的资产、威胁、威胁来源、脆弱性以及危害之间的关系如下图 B-2 所示：

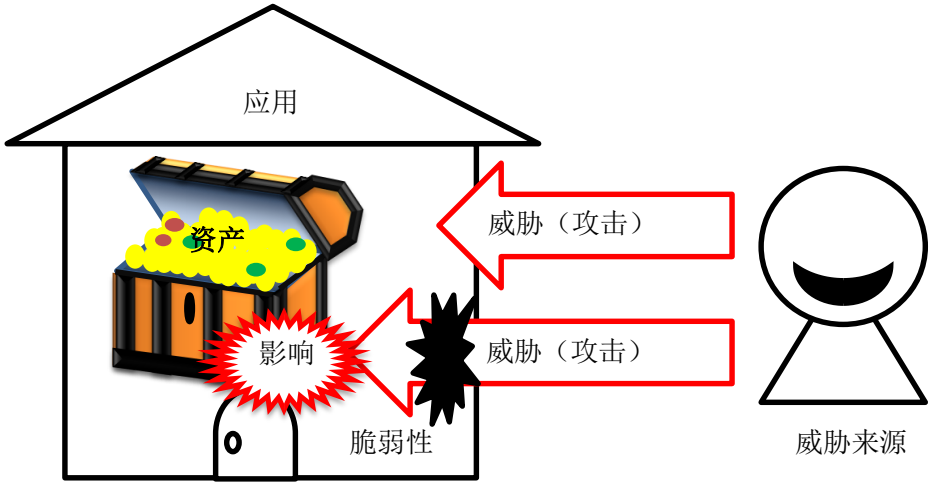



图 B-2 资产、威胁、威胁来源、脆弱性以及危害之间的关系

下图 B-3 展示的是 Android 应用行为的一个典型环境。从现在开始，为了使用这张图来拓宽说明一个 Android 应用所面对的威胁类型，首先我们将学习如何看懂这张图。

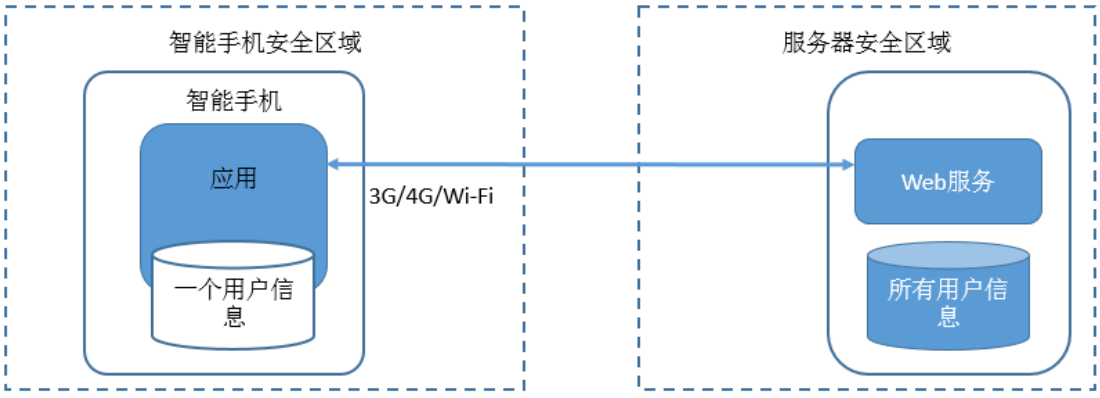



图 B-3 Android 应用行为的典型环境

图 B-3 中，智能手机在左侧而服务器在右侧，它们之间通过 3G/4G/Wi-Fi 通信，虽然一个智能手机里面存在多个应用，但我们在图中只显示单个应用以便更清晰的解释威胁。基于智能手机的应用主要处理用户信息，而基于服务器的 WEB 服务则集中管理所有用户信息。所以，和以往一样，这不会改变服务器安全的重要性，这里我们将不会涉及服务器安全相关话题，避免超出本手册范围。

#### B.1.1.2.1 威胁来自网络上的第三方

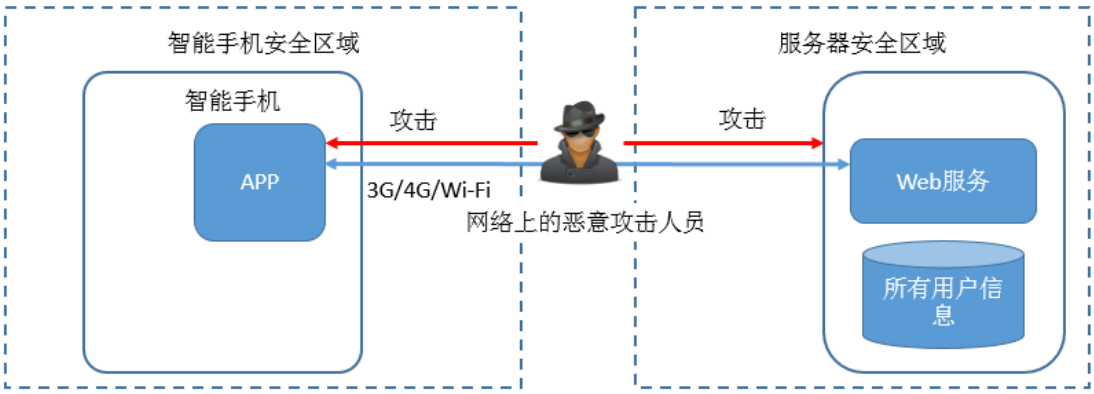



图 B-4 网络第三方攻击应用示意图

通常而言，一个智能手机应用是在某个服务器上管理用户信息，所以信息资产将在连接他们的网络上传输。如上图 B-4 所示，一个恶意的网络第三方可能访问（嗅探）通信之间的任何信息，或者试图修改信息（数据操作）。在中间的恶意攻击人员（也被称作“中间人攻击”）可以冒充真实的服务器来欺骗应用。不用说，基于网络的恶意第三方通常也会尝试攻击服务器。

#### B.1.1.2.2 威胁来自用户安装的恶意软件

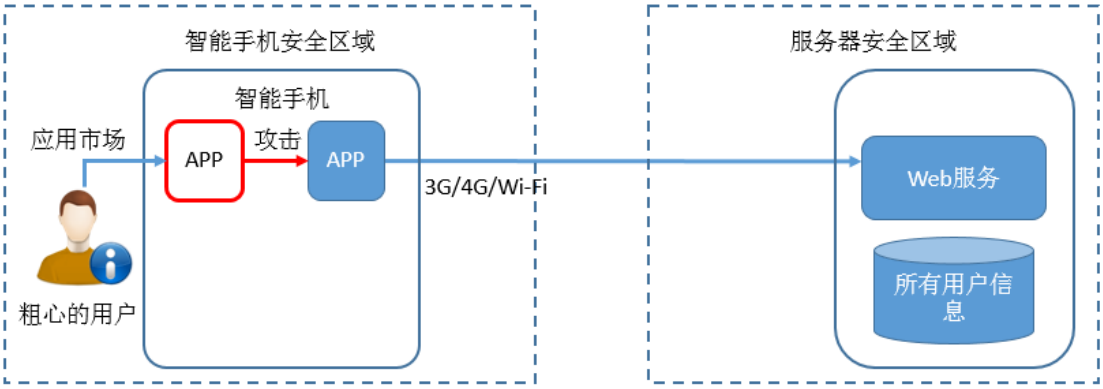



图 B-5 被用户安装的恶意软件攻击应用示意图

智能手机最大的卖点是可以从应用市场安装许多应用以扩大它的特性。给用户免费下载安装应用的站点有时也会错误地安装恶意软件。如上图 B-5 所示，恶意软件有可能会利用程序间通信功能或程序内部的漏洞，来达到访问信息或功能资产。

#### B.1.1.2.3 威胁来自能够利用应用漏洞的恶意文件

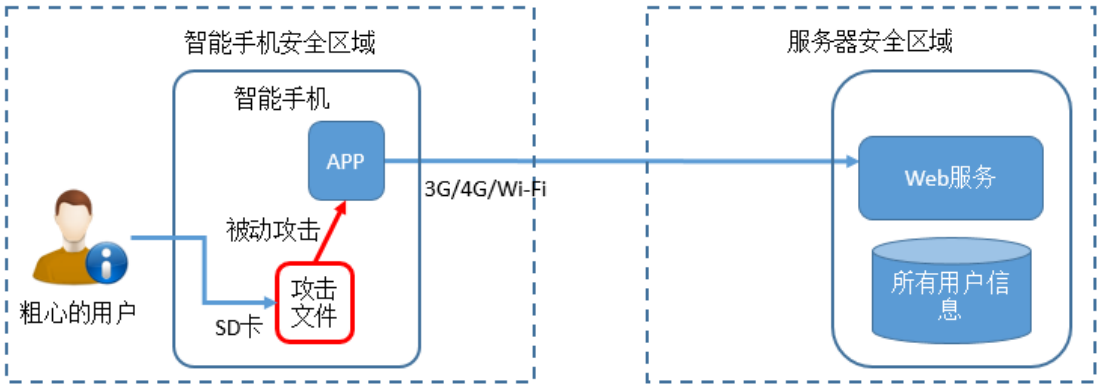



图 B-6：来自利用程序内部漏洞的恶意文件的攻击示意图

很多种文件，例如音乐、图像、视频和文件广泛存在互联网上，并且用户会下载很多文件到他们的 SD 卡以便在他们的智能手机上使用。此外，也很经常下载通过邮件发送的附件，这类文件后续会通过应用进行查看或编辑。

如果处理这些文件的应用存在功能上的一些漏洞，攻击者可以使用恶意文件来利用它并访问应用的信息或功能资产。特别是漏洞经常出现在处理一个具有复杂数据结构的文件格式上，当通过这种方法攻破一个应用，攻击者可以实现多种不同目的。如上图 B-6 所示，攻击文件处理休眠直接被某个存在漏洞的应用打开。一旦被打开，它将利用应用的漏洞产生严重破坏，我们将这种攻击方法叫做“被动攻击”。

#### B.1.1.2.4 威胁来自恶意手机用户

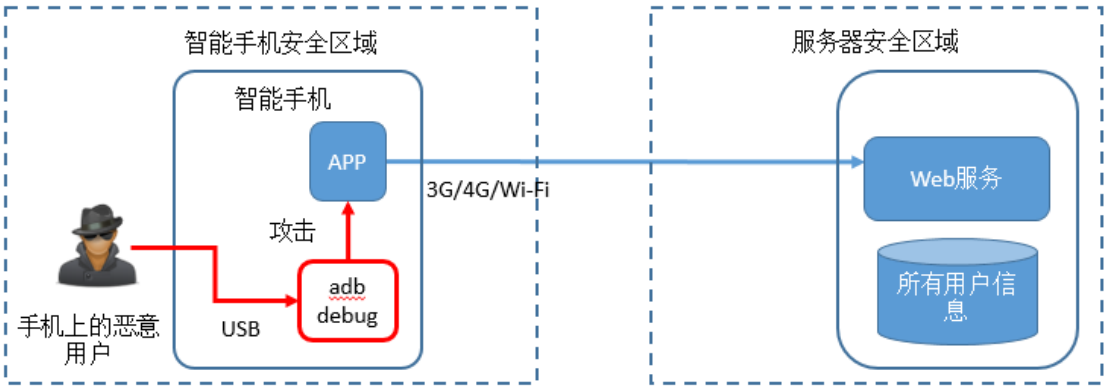



图 B-7 来自恶意手机用户的攻击示意图

至于 Android 智能手机的应用开发，它的环境和手机功能特性助于开发和分析应用一样，也是开放地提供给一般用户。在提供的功能特性当中，有用的 ADB 调试功能在没有注册或审查的情况下可以被任何人访问。这个特性允许一个手机用户轻易地执行系统或应用分析。

如上图 B-7 所示，一个带有恶意目的的智能手机用户通过利用 ADB 调试功能可以分析应用，并获得应用信息或功能资产的访问权限。如果应用包含的实际资产属于那个用户，那没什么问题；但是，如果那些资产是属于其他用户，如应用开发人员，那么将是一个问题。因此，我们需要注意合法手机用户也可以恶意地攻击应用里的资产。

### B.1.1.2.5 威胁来自智能手机附近的第三方

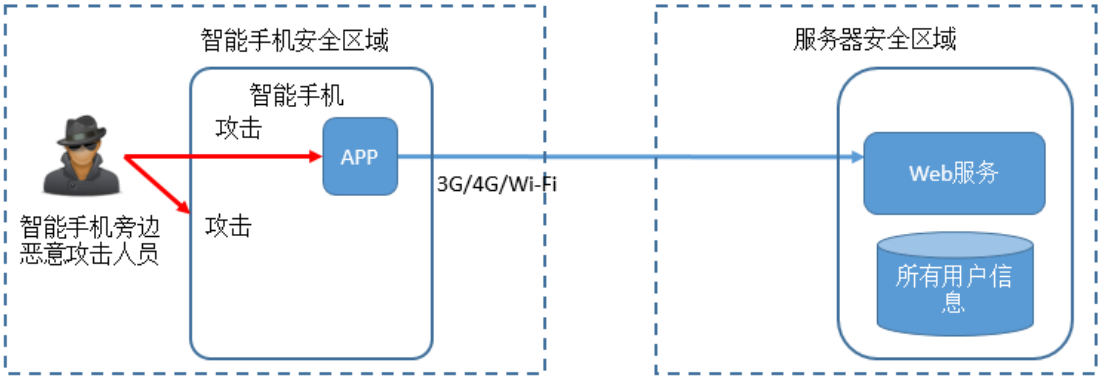



图 B-8 来自智能手机附近的恶意第三方攻击示意图

由于大部分智能手机具有各种近场通信机制，如 NFC、蓝牙和无线，我们绝不能忽视那种会发生来自物理临近手机的恶意人员的攻击。通过窥探某个正在输入的用户，攻击者可以窃取密码。或者如上图 B-8 所示，攻击者可以实施更复杂的，从远距离攻击一个应用的蓝牙功能。还有的威胁是，恶意人员可以窃取智能手机造成数据泄露风险，或甚至毁坏手机导致重要信息丢失。开发人员应该在设计阶段尽早的将这些风险考虑进去。

### B.1.1.2.6 安全威胁总结

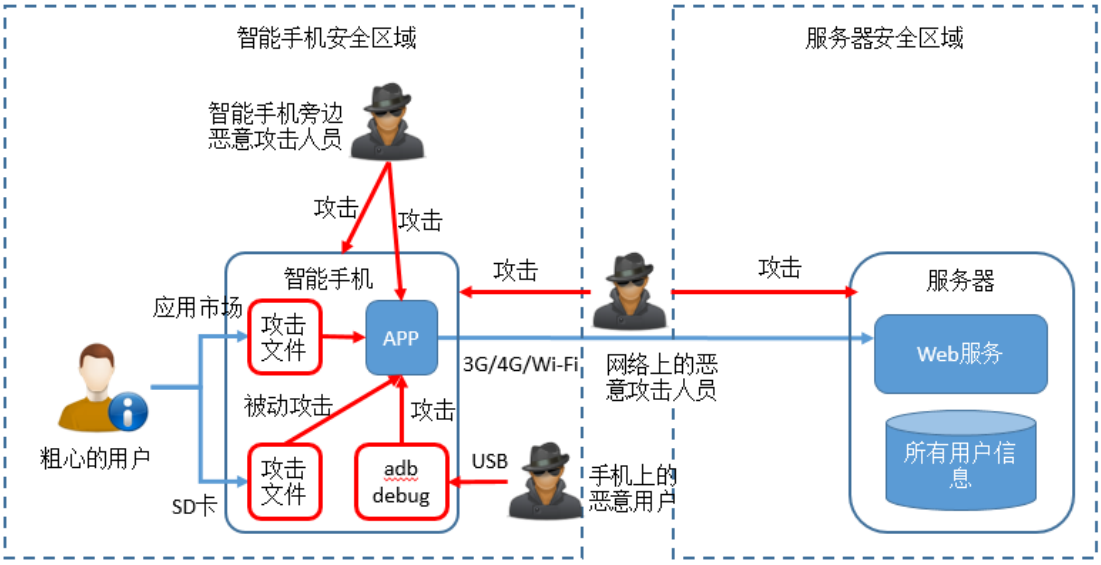



图 B-9 智能手机应用各种攻击汇总

上图汇总了前面章节描述的威胁的主要类型，智能手机面临着多种多样的威胁，上面的图示 B-9 并未包括所有威胁。通过我们日常的信息收集，我们需要传播关于 Android 应用面临多种威胁的意识，以及在应用安全设计和开发过程中要注意它们。

#### B.1.1.3 资源等级分类和保护措施

正如前面章节所探讨，Android 智能手机面临着各种各样的威胁，为抵御这些威胁要保护一个应用里的所有资产确实是非常困难，一是开发时间紧，二是技术受限制。因此，Android 应用开发人员应该测试资产保护措施的可行性和有效性，这可以根据开发人员判定的优先级来做，根据被查看到的资产的重要性以及可接受的损害水平，这是比较主观的一件事情。

为了帮助选定各个资产的保护措施，我们将进行资产分类并规定每个组的安全保护等级，可以通过检查法律依据，根据发生损害的影响以及开发人员（或组织机构）的社会责任来确定重要等级。当要选定如何处理这些资产以及实施的保护措施类型，这些将被证明是一种评判准则。因为这些将为应用开发人员和组织机构在决定如何处置资产和提供保护措施方面提供一种标准，所以有必要去明确与应用开发人员（或组织机构）境况一致的分类方法和相应的保护措施。

适用于本手册的资产分类和保护措施等级如下所示，以供参考。

| 资产分类 | 资产等级 | 保护等级 |
|---|---|---|
| 高 | 资产的损害程度对于组织机构或个体来说都是致命的、灾难性的。<br>例如：当属于这个级别的资产受侵害时，组织机构将无法继续开展业务 | ● 提供保护措施对抗可以攻破 Android OS 安全模型的复杂攻击并能阻止 root 权限修改 APK 的 dex 文件。<br>● 确保安全的优先级高于其他因素，比如用户体验等 |
| 中 | 资产的损害程度会对组织机构或个体来造成严重影响。<br>例如：当属于这个级别的资产受到侵害，组织的营收恶化，对业务造成不利影响 | ● 利用 Android OS 安全模型。它提供的保护措施涵盖这个级别的范围<br>● 确保安全的优先级高于其他因素，比如用户体验等 |
| 低 | 资产的损害程度会对组织机构或个体来造成一定程度的影响。<br>例如：当属于这个级别的资产受到侵害，组织的营收受到影响，但可以利用其他资源来弥补损失。 | ● 利用 Android OS 安全模型。它提供的保护措施涵盖这个级别的范围<br>● 安全保护措施与其他因素相比，如用户体验等，在这个级别，它的等级可能因为没有安全问题而优先于安全措施。 |

本手册描述的资产分类和保护措施提出的前提是一个 root 权限没有被突破的安全 Android 设备，而且它的安全策略是基于 Android OS 安全模型。特别要说的是，假设我们通过 Android OS 安全模型来设计安全保护措施的前提，是通过运行 Android OS 安全机制用来保护低于或等于中级别的资产。另一方面，我们也相信有必要对高级别资产采取保护以抵御能够导致突破 Android OS 安全模型的攻击，这类攻击包括突破 root 权限攻击，分析或修改 APK 二进制文件的攻击。为了保护这类资产对抗通过组合多种攻击方法的威胁，我们需要设计复杂的防御措施，如加密、混淆、硬件保护和服务器保护等。因为

关于这类防御技术集锦都编入本手册是很难的，并且恰当的防御体系设计会根据不同的环境而有所区别，所以我们认为它们不在本手册的范围之内。如果你的设备需要保护措施来抵抗复杂攻击，包括突破 root 权限攻击，分析或修改 APK 二进制文件的攻击，我们建议你去咨询一些精通 Android 防篡改设计的安全专家。

#### B.1.1.4 敏感信息

从现在开始，“敏感信息”将取代“信息资产”这个词语，虽然它在前面章节有被提到过，但我们必须确定应用程序所处理的每个信息资产的资产类别和保护措施等级。在本的册里，我们关注的是保护低于或等于中级别的资产。

### B.1.2 谨慎、安全地处理输入的数据

检验输入数据是最简单且最有效的安全编码方法。从外部来源，不管是直接地还是间接地输入到应用的所有数据都应该正确地被检验，为了展示输入数据检验的最佳实践，下面是一个例子：某个程序里的一个 Activity 从 Intent 接收数据。

Activity 很有可能会接收到已被攻击人员篡改的 Intent 数据。通过传输某种格式的数据或某个程序员不期望的值，攻击人员可以在程序内部引发故障，那将导致一些类型的安全事件。我们绝不能忽视用户也有可能变成攻击人员。

Intent 是由行为、数据和额外部分配置的，我们在接收所有可能被攻击人员控制的数据类型时必须保持警惕。在任何处理来自不可信源的数据的代码里，我们经常需要检验下列条目：

1) 可以接收与程序员指定格式匹配且值在可接收范围内的数据吗？

2) 即使接收到指定格式和值的数据，你能保证处理这些数据不会发生意外状况吗？

Sample Code that Displays HTML of a Remote Web page in TextView

```
TextView tv = (TextView) findViewById(R.id.textview);
InputStreamReader isr = null;
char[] text = new char[1024];
int read;
try {
    String urlstr = getIntent().getStringExtra("WEBPAGE_URL");
    URL url = new URL(urlstr);
    isr = new InputStreamReader(url.openConnection().getInputStream());
    while ((read=isr.read(text)) != -1) {
        tv.append(new String(text, 0, read));
    }
} catch (MalformedURLException e) { ...
```

从上述观点 1）来看，urlstr 是正确的 URL，使用新的 URL()通过一个 MalformedURLException 来确认。然而，这还是不够的，此外，当 urlstr 任命一个为“file://...”的格式化 URL 时，内部文件系统上的文件会被开打并在 TextView 里显示，而不是远程的 WEB 网页。这并不满足观点 2），因为它没有保证一定按程序员预想的执行。

下面的例子展示修复这个安全缺陷的一个修订版本。通过观点 1），通过检查“urlstr”是一个合法 URL 来确证输入数据且协议被限定为 HTTP 或 HTTPS。因此，即使根据观点 2），一个按互联网路径发送的 InputStream 获取值通过 url.openConnection().getInputStream()是可以保证的。

**Sample Code that Displays HTML of a Remote Web page in TextView**

```
TextView tv = (TextView) findViewById(R.id.textview);
InputStreamReader isr = null;
char[] text = new char[1024];
int read;
try {
    String urlstr = getIntent().getStringExtra("WEBPAGE_URL");
    URL url = new URL(urlstr);
    isr = new InputStreamReader(url.openConnection().getInputStream());
    while ((read=isr.read(text)) != -1) {
        tv.append(new String(text, 0, read));
    }
} catch (MalformedURLException e) { ...
```

验证输入数据的安全性称作“输入校验”，它是安全编码最基本的方法。
从输入检验的词意来推测，会经常出现一种情况，就是只注意观点 1）而忽视观点 2）。需要切记的是，数据传入程序时可以不会发生危害，但程序以不正确的方式使用数据时则会发生。

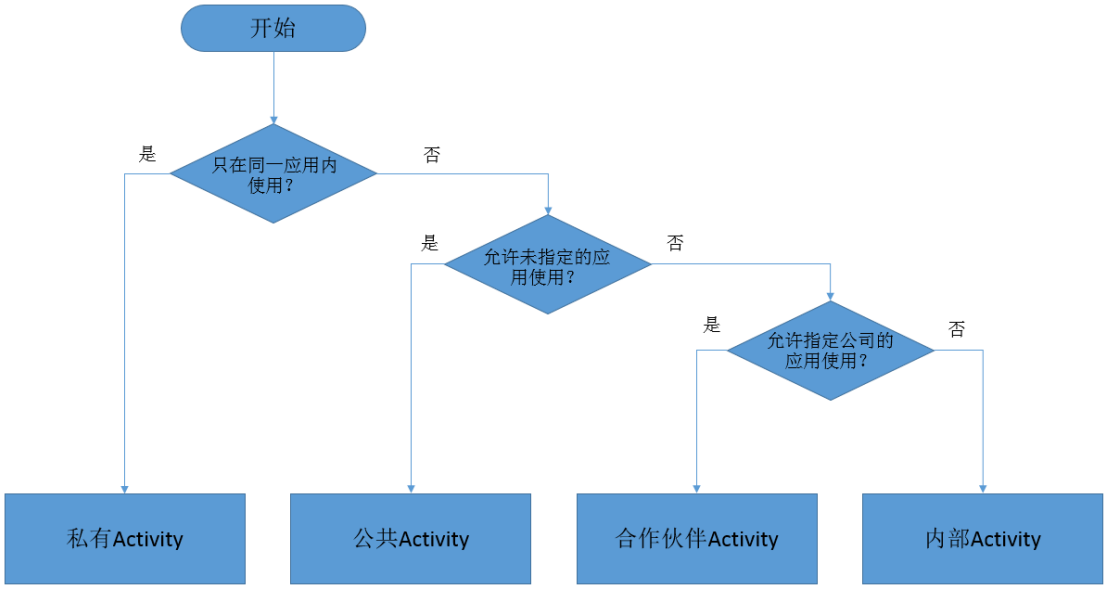
## B.2 创建/使用 Activities

### B.2.1 Activities 种类

以下例举 4 种常见的 Activities 使用场景，不同的的场景，安全开发实践也是不同的，我们会逐一进行介绍各种 Activities 的安全开发规范。

表 B-5 Activity 类型定义

| 类型 | 定义 |
|---|---|
| 私有 Activity | Activity 无法被其他应用程序运行，因此是最安全的 Activity |
| 公共 Activity | 假设 Activity 可以被大量未指定的应用程序使用 |
| 合作伙伴 Activity | Activity 只能被合作伙伴公司指定的应用程序使用 |
| 内部 Activity | Activity 只能被其他内部应用程序使用 |




### B.2.2 安全开发守则

#### B.2.2.1 只在应用程序内部使用的 Activities 必须设置成私有

只在单个应用程序内部使用的 Activities 不需要接收其他应用程序的 intents。开发人员通常假设 Activities 作为私有使用不会被攻击，但是非常有必要明确设置 Activities 成私有，确保不会接收到恶意的 intents。

Activities 只在只在单个应用程序内部使用时不能设置 intent filters。由于 Intent filters 的特点及运行特性，即使你只打算发送一个 Intent 给内部私有 Activity,如果你通过 intent filter 发送 Intent，将有可能无意地启动另一个 Activity。

#### B.2.2.2 不要指定 taskAffinity

在 Android 操作系统里，Activities 是由任务管理的。任务名称是由 root Activity 的 affinity 决定的。另一方面,对于 Activity 而言(而不是 root Activity),Activity 所属的任务不仅是由 affinity 决定，也要根据 Activity 的启动模式。

在默认情况下，每个 Activity 使用它的包名作为它的 affinity。因此，任务是根据应用程序分配的，所以所有在单个应用里的 Activities 都属于同于个任务。若要改变任务分配情况，你可以在 AndroidManifest.xml 文件为 affinity 中做一个明确的声明，或者在发送给一个 Activity 的 intent 里设置一个标志。然而，如果你改变任务分配，就会存在安全风险，即其他应用可以读取要发送给属于另一个任务的 Activities 的 Intents 信息。

切记不要在 AndroidManifest.xml 文件中指定 android:taskAffinity，并使用默认设置保持使用包名作为 affinity，目的是避免内部传送的敏感信息或接收到的 intents 被其他应用读取。

### B.2.2.3 不要指定启动模式

Activity 启动模式是用来控制启动一个 Activity 时所创建新任务和 Activity 实例的配置，简单地说就是 Activity 启动时的策略，默认设置是标准模式。在标准模式下，当启动一个 Activity 就会创建新的实例，任务跟随那个所属调用 Activity 的任务，因此，它不可能再创建一个新手任务。当一个新的任务被创建，调用的 intent 内容就有可能被其他应用读取，所以当在一个 intent 里包含敏感信息时必须使用 Activity 的标准启动模式。

Activity 的启动模式可以在 AndroidManifest.xml 文件中的 android:launchMode 属性里明确指定，但由于上述原因，它不应该在 Activity 声明里设置，并且应该保持默认的“标准模式”。

AndroidManifest.xml

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >

    <!-- Private activity -->
    <!-- *** POINT 2 *** Do not specify launchMode -->
    <activity
        android:name=".PrivateActivity"
        android:label="@string/app_name"
        android:exported="false" />
</application>
```

### B.2.2.4 不要为用来启动 Activity 的 Intents 设置 FLAG_ACTIVITY_NEW_TASK 标志

当执行 startActivity()或 startActivityForResult()时，Activity 的启动模式可以被修改，并且在某种情况下，可能产生一个新的任务。因此，非常有必要在执行时不要改变 Activity 的启动模式。

要改变 Activity 启动模式，通过使用 setFlag()或 addFlag()设置 Intent 标志，并使用这个 intent 作为 startActivity() 或 startActivityForResult()的参数。FLAG_ACTIVITY_NEW_TASK 是用来创建一个新任务的标志，当 FLAG_ACTIVITY_NEW_TASK 被设置，一个新的任务将会被创建，即使被调用的 Activity 不存在前台或后台。

可以使用 FLAG_ACTIVITY_NEW_TASK 同时设置 FLAG_ACTIVITY_MULTIPLE_TASK，在这种情况下，新的任务会一直被创建。被创建的新任务可能会使用随意配置，因此，当使用敏感信息时不应该这样设置 intents。

另外，你可能会认为还有一种方法可以来阻止一个 Intent 内容被读取，即通过设置 FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS 标志来创建一个新任务，然而，即使用这种方法，内容还是会被第三方读取，所以你应该避免使用 FLAG_ACTIVITY_NEW_TASK。

### B.2.2.5 谨慎、安全地处理接收到的 Intent

风险根据 Activity 的类型不同而有所差异，但当处理一个接收到的 Intent 数据，你第一个想到的应该是输入校验。由于公共 Activities 可以接收来自不信任源的 Intents，它们会被恶意软件攻击。另一方面，私有 Activities 永远不会直接接收来自其他应用的所有 Intents，但有一种可能性，目标应用的一个公共 Activity 可能转发一个恶意的 Intent 给一个私有 Activity，所以你不能假设私有 Activities 不会接收到所有恶意输入。由于合作伙伴 Activities 和内部 Activities 也存在风险，恶意的 Intent 也会被转发给它们，所以很有必要对这类 intents 进行输入检验。

### B.2.2.6 使用在内部应用里验证过的内部签名权限

创建 Activity 时，确认是通过定义内部签名权限来保护你的内部 Activities，因为在 AndroidManifest.xml 文件指定一个权限或声明一个请求权限不能提供相同安全级别。详细内容参见“如何在内部应用之间使用内部定义的签名权限进行通信”。

### B.2.2.7 返回结果时，注意来自目标应用造成信息泄露的可能性

当你使用 setResult()返回数据时，目标应用的可靠性将取决于 Activity 类型。当使用公共 Activity 来返回数据，目标有可能被换成恶意软件，这种情况下，信息可被恶意使用。对于私有和内部 Activities，不必太担心返回的数据会被恶意使用，因为你可以控制返回给哪些应用。合作伙伴 Activities 介于这两者之间。
综上所述，从 Activities 返回数据时，你必须注意来自目标应用造成的信息泄露。

Example of returning data.

```
public void onReturnResultClick(View view) {

        // *** POINT 6 *** Information that is granted to be disclosed to a partner application can be return
ed.
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Sensitive Info");
        setResult(RESULT_OK, intent);
        finish();
}
```

### B.2.2.8 若目的 Activity 已预定，使用显式 intents

通过隐式 intents 使用 Activity 时，是由 Android OS 决定 intent 发送给哪个 Activity。如果 Intent 错误地发送给恶意软件，信息泄漏就会发生。另一方面，当通过显示 intents 使用 Activity 时，只有预期在 Activity 才能接收到 Intent，所以它会更安全。

如果不是非常必要让用户决定 Intent 应该发送给应用的哪个 Activity，应该使用显式 Intents 并提前指定目标。

Using an Activity in the same application by an explicit Intent

```
Intent intent = new Intent(this, PictureActivity.class);
intent.putExtra("BARCODE", barcode);
startActivity(intent);
```

Using other applicaion's Public Activity by an explicit Intent

```
Intent intent = new Intent();
intent.setClassName(
    "org.jssec.android.activity.publicactivity",
    "org.jssec.android.activity.publicactivity.PublicActivity");
startActivity(intent);
```

然而，即使通过显式使用其他应用的公共 Activity 时，也有可能存在目标 Activity 是恶意软件。

这是因为即使你通过包名限制目标，它仍然有可能存在恶意应用伪造和真实应用一样的包名。为了消除这类风险，有必要考虑使用合作伙伴 Activity 或内部 Activity。

#### B.2.2.9 谨慎、安全地处理来自请求 Activity 的返回数据

虽然根据你访问 Activity 类型的安全风险只是细微的差别，当处理接收的 intent 数据作为返回值时，你需要对接收的数据一直进行输入校验。

当访问一个公共 Activity，公共 Activity 不得不从不可信源接受返回 Intent，那么就有可能返回的 Intents 实际上是由恶意软件发出的。我们经常错误地认为所有从私有 Activity 返回的 Intents 是安全的，因为它们来自同一个应用。然而，由于有可能接收到的一个来自不可信源但被直接转发的 intent，你不应该盲目地信任那个 intent 的内容。合作伙伴和内部 Activities 存在一定的风险介于私有和公共 Activities，必定也要对这类 Activities 进行输入校验。

#### B.2.2.10 检查和其他公司应用相关联的目标 Activity

和其他公司的应用相连接时一定要确认白名单，你可以这样做，通过保存一份该公司的证书哈希在你的应用里，并和目标应用的证书哈希做比对。这样可以阻止一个恶意应用欺骗 Intents。具体的实施方法详见开发范例章节“创建/使用合作伙伴 Activity”，具体的技术细节详见“验证请求的应用”章节。

#### B.2.2.11 间接提供资源时，应当采取相同的保护等级

当受权限保护的信息或功能资源间接提供给其他应用时，你要确保它拥有相同的访问权限。在 Android OS 权限安全模型中，只能正确授权的应用才能直接访问受保护的资源。然而，存在一些漏洞，因为拥有某个资源权限的应用可以作为代理并允许其他未授权应用访问资源。实质上，这和重新委派权限一样，所以它被称为权限委派问题，参见“权限委派问题”章节。

#### B.2.2.12 尽可能多的限制敏感信息的传送

你不应该发送敏感信息给不可信的合作方，即使当你和某个特定应用关联，还是存在一个疏漏，就是你无意地发送一个 Intent 给一个不同应用或恶意第三方（可以窃取你的 Intents）。参见“使用 Activities 时的日志输出”章节

当发送敏感信息给一个 Activity，你需要考虑信息泄露风险。你必须假设发送给公共 Activity 的 Intents 所有数据可以被恶意的第三方获取。此外，当发送 Intents 给合作伙伴或内部 Activities，还存在大量信息泄露风险，即使发送数据给私有 Activities，也有会存在一定的风险，即 Intent 数据可能通过 LogCat 泄漏。在 Intent 特殊部分的信息不会被输出到 LogCat，所以它是存储敏感信息最好的地方。

不论如何，一开始就不发送敏感数据是阻止信息泄露的最完美的解决方案，因此，应该限制发送敏感信息的数量越少越好，最佳实践是只发送给可信的 Activity 且确保信息不会被 LogCat 泄露。

此外，敏感信息应该永不发送给根 Activity。根 Activities 是任务被创建时第一个调用的那些 Activities。举个例子，启动栏运行的 Activity 永远是根 Activity。更多根 Activity 详细信息参见“读取发送给 Activity 的 Intents”和“根 Activity”。

### B.2.3 安全开发范例

#### B.2.3.1 创建/使用私有 Activity

当使用只在应用程序内部使用的 Activities 时，只要在类中使用显式 intents，你就不用担心它会意外的发送给其他应用。然而，当某个第三方应用可以读取一个用来启动 Activity 的 intent 时，就会存在安全风险。因此，当你在 intent 里放置敏感信息并用它来启动 Activity 时，就必须采取措施确保它不会被恶意的第三方应用程序读取。

创建私有 Activity 时遵循的几个关键点：

1) 不要指定 taskAffinity。

2) 不要指定 launchMode。

3) 明确地指定 exported 属性为 false。

4) 要谨慎、安全的处理接收到的 intent，即使这个 intent 是同一个应用程序发出的。

5) 敏感信息是可以被传送的，只要控制在同一个应用程序内部发送和接收。

代码示例

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.privateactivity" >

    <application
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Private activity -->
        <!-- *** POINT 1 *** Do not specify taskAffinity -->
        <!-- *** POINT 2 *** Do not specify launchMode -->
        <!-- *** POINT 3 *** Explicitly set the exported attribute to false. -->
        <activity
            android:name=".PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />

        <!-- Public activity launched by launcher -->
        <activity
            android:name=".PrivateUserActivity"
            android:label="@string/app_name"
            android:exported="true" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
```

使用私有 Activity 时遵循的几个关键点：

1) 当要启动一个 Activity 时，不要设置 intents 的 FLAG_ ACTIVITY_NEW_TASK 的标志。

2) 在指定的类里使用显式 intents 去调用同一个应用程序里的 Activity。

3) 当目标 Activity 在同一个应用程序时，可以使用 putExtra()传递敏感信息。

4) 要谨慎、安全的处理接收到的结果数据，即使数据是从同一个应用程序内的 Activity 发出的。

代码示例

```
package org.jssec.android.activity.privateactivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PrivateUserActivity extends Activity {

    private static final int REQUEST_CODE = 1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user_activity);
    }

    public void onUseActivityClick(View view) {

        // *** POINT 6 *** Do not set the FLAG_ACTIVITY_NEW_TASK flag for intents to start an activity.
        // *** POINT 7 *** Use the explicit Intents with the class specified to call an activity in the same applicati
on.
        Intent intent = new Intent(this, PrivateActivity.class);

        // *** POINT 8 *** Sensitive information can be sent only by putExtra() since the destination activity is in t
he same application.
        intent.putExtra("PARAM", "Sensitive Info");

        startActivityForResult(intent, REQUEST_CODE);
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (resultCode != RESULT_OK) return;

        switch (requestCode) {
        case REQUEST_CODE:
```

### B.2.3.2 创建/使用公共 Activities

因为公共 Activities 可以被大量未指定的应用程序使用，所以必须引起重视的是公共 Activities 可能会接收到恶意软件发送过来的 intents，另外，当使用公共 Acitivities 时，也要注意恶意软件同样可以接收或读取发送给它们的 intents。

创建公有 Activity 时遵循的几个关键点：

1) 明确设置 exported 属性为 true。

2) 谨慎、安全的处理接收到的 intent。

3) 当返回数据时，不要包含敏感信息。

代码示例：

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.publicactivity" >

    <application
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Public Activity -->
        <!-- *** POINT 1 *** Explicitly set the exported attribute to true. -->
        <activity
            android:name=".PublicActivity"
            android:label="@string/app_name"
            android:exported="true">

            <!-- Define intent filter to receive an implicit intent for a specified action -->
            <intent-filter>
                <action android:name="org.jssec.android.activity.MY_ACTION" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // *** POINT 2 *** Handle the received intent carefully and securely.
        // Since this is a public activity, it is possible that the sending application may be malware.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this, String.format("Received param: ¥"%s¥"", param), Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // *** POINT 3 *** When returning a result, do not include sensitive information.
        // Since this is a public activity, it is possible that the receiving application may be malware.
        // If there is no problem if the data gets received by malware, then it can be returned as a result.
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Not Sensitive Info");
        setResult(RESULT_OK, intent);
        finish();
    }
```

使用公有 Activity 时遵循的几个关键点：

1) 不要发送敏感信息。

2) 谨慎、安全的处理接收到的返回数据。

代码示例：

```
public void onUseActivityClick(View view) {

    try {
        // *** POINT 4 *** Do not send sensitive information.
        Intent intent = new Intent("org.jssec.android.activity.MY_ACTION");
        intent.putExtra("PARAM", "Not Sensitive Info");
        startActivityForResult(intent, REQUEST_CODE);
    } catch (ActivityNotFoundException e) {
        Toast.makeText(this, "Target activity not found.", Toast.LENGTH_LONG).show();
    }
}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    // *** POINT 5 *** When receiving a result, handle the data carefully and securely.
    // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
    if (resultCode != RESULT_OK) return;
    switch (requestCode) {
    case REQUEST_CODE:
        String result = data.getStringExtra("RESULT");
        Toast.makeText(this, String.format("Received result: ¥"%s¥"", result), Toast.LENGTH_LONG).show();
        break;
    }
}
```

#### B.2.3.3 创建/使用合作伙伴 Activities

合作伙伴 Activities 只能被特定的应用程序使用，经常使用在合作伙伴公司之间进行安全地共享信息和功能。当某个第三方应用可以读取一个用来启动 Activity 的 intent 时，就会存在安全风险。因此，当你在 intent 里放置敏感信息并用它来启动 Activity 时，就必须采取措施确保它不会被恶意的第三方应用程序读取。

使用合作伙伴 Activity 时遵循的几个关键点：

1) 不要指定 taskAffinity。

2) 不要指定 launchMode。

3) 不要定义 intent filter 并明确地设置 exported 属性为 true。

4) 通过事前定义白名单来确认发送请求的应用证书。

5) 谨慎、安全的处理接收到的 intent，即使它是从合作伙伴的应用程序发送过来的。

6) 只返回指定信息，即授权给合作伙伴应用程序的特定信息。

代码示例：

```
import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PartnerActivity extends Activity {

    // *** POINT 4 *** Verify the requesting application's certificate through a predefined whitelist.
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner application org.jssec.android.activity.partneruser.
        sWhitelists.add("org.jssec.android.activity.partneruser", isdebug ?
                // Certificate hash value of "androiddebugkey" in the debug.keystore.
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // Certificate hash value of "partner key" in the keystore.
                "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2527B8D 4C0EC35A");

        // Register the other partner applications in the same way.
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

       // *** POINT 4 *** Verify the requesting application's certificate through a predefined whitelist.
       if (!checkPartner(this, getCallingActivity().getPackageName())) {
           Toast.makeText(this,
                   "Requesting application is not a partner application.",
                   Toast.LENGTH_LONG).show();
           finish();
           return;
       }


       // *** POINT 5 *** Handle the received intent carefully and securely, even though the intent was sent from a p
artner application.
       // Omitted, since this is a sample. Refer to "3.2 Handling Input Data Carefully and Securely."
       Toast.makeText(this, "Accessed by Partner App", Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

       // *** POINT 6 *** Only return Information that is granted to be disclosed to a partner application.
       Intent intent = new Intent();
       intent.putExtra("RESULT", "Information for partner applications");
       setResult(RESULT_OK, intent);
       finish();
```

使用合作伙伴 Activity 时遵循的几个关键点：

1） 验证目标应用程序的证书是否在白名单中。

2) 启动 activity 的 intent 不要设置 FLAG_ACTIVITY_NEW_TASK 标志。

3) 使用 putExtra()传送只允许对合作伙伴公开的信息

4) 使用显式 intent 来调用合作方的 Activity。

5) 使用 startActivityForResult()调用合作方的 Activity。

6) 谨慎、安全的处理接收到的返回数据，即使它是从合作伙伴的应用程序发送过来的。

代码示例：

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.partneruser" >

    <application
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <activity
            android:name="org.jssec.android.activity.partneruser.PartnerUserActivity"
            android:label="@string/app_name"
            android:exported="true" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

### B.2.3.4 创建/使用内部 Activities

创建内部 Activity 时遵循的几个关键点：

1) 定义一个内部签名权限

2) 不要指定 taskAffinity

3) 不要指定 launchMode

4) 要求使用内部签名权限

5) 不要定义 intent filter 并明确地设置 exported 属性为 true。

6) 验证被内部应用程序定义的内部签名权限

7) 谨慎、安全的处理接收到的 intent，即使它是从内部应用程序发送过来的。

8) 可以返回敏感信息，因为请求的应用在内部。

9） 当导出 APK 文件，使用与目标应用程序相同的密钥进行签名

代码示例：

使用内部 Activity 时遵循的几个关键点

1） 声明你想使用的内部签名权限

2） 验证被内部应用程序定义的内部签名权限

3） 验证目标应用程序已经使用内部证书签名

4） 敏感信息可以通过 putExtra()发送，因为目标程序在内部

5） 使用显式 intents 调用内部 Activity

6） 谨慎、安全的处理接收到的数据，即使它是从内部应用程序发送过来的。

7） 当导出 APK 文件，使用与目标应用程序相同的密钥进行签名

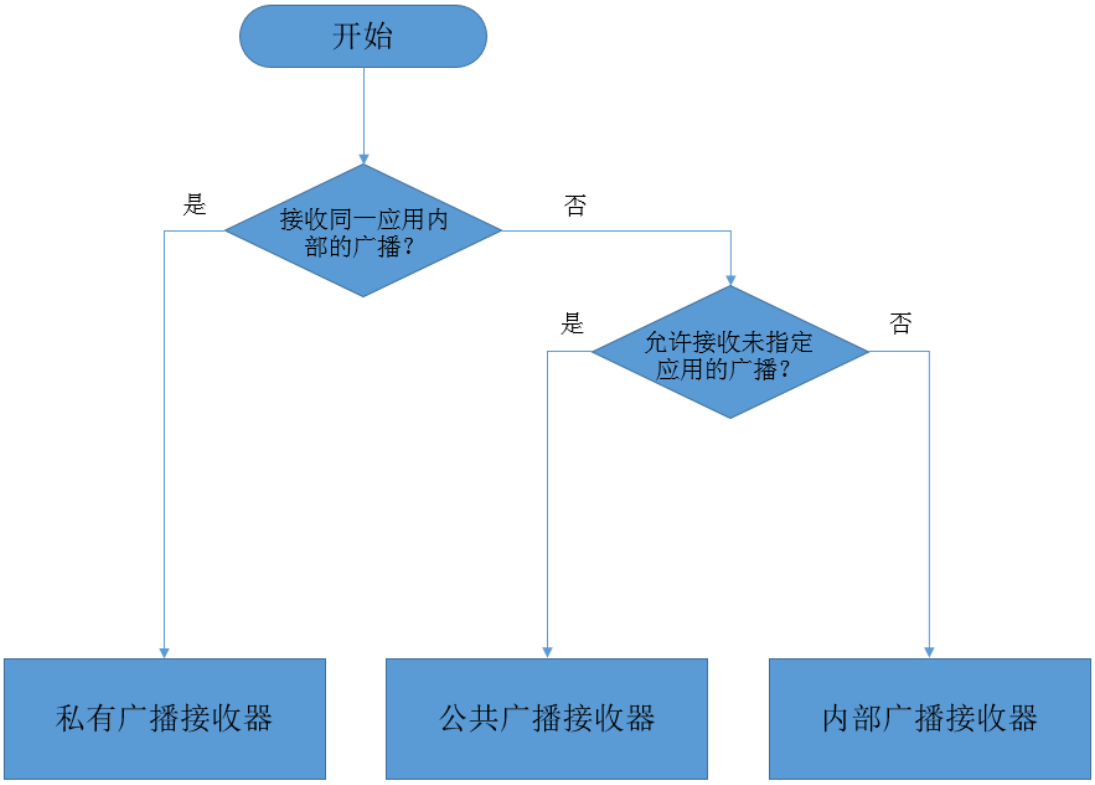
代码示例：（略）

## B.3 接收/发送广播

### B.3.1 广播类型

创建广播接收器是为了能够接到广播，使用广播接收器的安全风险及防范措施根据要接收的广播类型不同而有所区别。当接收广播的应用不能检查发送广播的应用包名，就不能与合作伙伴关联起来，从而导致无法为合作伙伴创建广播接收器。

| 类型 | 定义 |
|---|---|
| 私有广播接收器 | 广播接收器只能接收来自同一个应用内部的广播，因此它是最安全的 |
| 公共广播接收器 | 广播接收器可以接收来自大量未指定的应用程序的广播 |
| 内部广播接收器 | 广播接收器只能接收来自其他内部应用的广播 |




另外，广播接收器根据注册方式还可以分成两种类型，静态广播接收器和动态广播接收器。两者的区别如下表所示：

| 类型 | 注册方式 | 特征 |
| --- | --- | --- |
| 静态广播接收器 | ● 在 AndroidManifest.xml 文件中定义<receiver>元素 | ● 有限制，一些系统发送的广播无法接收，例如 ACTION_BATTERY_CHANGED<br>● 从应用初始化启动到卸载的广播都能接收到 |
| 静态广播接收器 | 在程序中使用 Context.registerReceiver 注册 | ● 静态广播接收器无法接收的都可以被动态广播接收器接收<br>● 接收广播的周期可以被应用控制，例如，只有当 Activity 在前台时广播才能被接收。<br>● 不能创建私有广播接收器 |

### B.3.2 安全开发守则

#### B.3.2.1 只在某个应用内部使用的广播接收器必须设置成私有

只能应用内部使用的广播接收器应该设置成私有，以免意外地接收到来自其他应用的广播。这样做可以避免应用功能滥用或异常行为。

只在同一个应用内部使用的广播接收器不应该设置 intent-filter。由于 intent-filter 的特性，即使只想调用同一个应用内部的私有广播接收器，也有可能会通过 intent-filter 意外地调用其他应用的公共广播

接收器。其他请参阅“组合使用 Exported 属性和 Intent Filter 设置（广播接收器）”章节。

#### B.3.2.2 谨慎、安全地处理 intent

由于不同类型的广播接收器的安全风险有所区别，在处理接收到的 intent 数据时应第一时间检查 intent 的安全性。

由于公共广播接收器会接收到来自大量不确定应用的 intents,它有可能接收到恶意软件的 Intent 攻击。私有广播接收器从不直接地接收来自其他应用的任何 intent，但某个公共组件从其他应用接收到的 intent 数据可能会被转发给私有广播接收器，所以不要认为私有接收器接收到的 intent（未经验证）总是安全的。内部广播接收器同样存在一定程度的风险，所以它也必须核实接收到的 intent 的安全性。

#### B.3.2.3 使用在内部应用里验证过的内部签名权限

只接收内部应用广播的内部广播接收器应该使用内部定义签名权限进行保护。在 AndroidManifest.xml 文件里声明权限定义/权限请求的保护程度是不够的，参阅“如何在内部应用之间使用内部定义的签名权限进行通信”。通过指定内部定义的签名权限给 receiverPermission 参数来结束广播也需要同样的校验。

#### B.3.2.4 返回结果信息时，注意来自目标应用的结果泄漏信息风险

通过 retResult()返回结果信息的应用程序的可靠性取决于广播接收器的类型。假设公共广播接收器，目标应用程序可能是恶意软件，那么就有可能存在结果信息被恶意使用的风险。假设私有广播接收器和内部广播接收器，返回结果信息的目标是内部开发的程序，也必须注意结果信息的处理。同上，当广播接收器返回结果信息时，必须注意的是来自目标应用的结果信息泄漏风险。

#### B.3.2.5 发送敏感信息给某个广播时，限制可能接收到的接收器

广播是广播信息给大量不确定应用或一次性定时通知它们,所以广播敏感信息需要谨慎的规划以避免信息被恶意软件非法获取。广播敏感信息时，只有可靠的广播接收器才能接收它，而其他广播接收器则不能。下面是关于广播发送方法的一些例子：

- 只发送给固定地址的方法。发送广播方使用显式 intent 发送广播给指定的可靠的广播接收器，这种方法分为两个部分：
  - 当它定位到同一个应用内的某个广播接收器，通过 intent#setClass(Context,Class)来指定目标位置，正确的编码方式请参阅安全开发范例的“私有广播接收器”章节。
  - 当它定位到其他应用的某个广播接收器，通过 ntent#setClassName(String, String)来指定目标位置。通过和白名单比对目标应用包 APK 签名的开发密钥来确认允许的应用并发送广播，实质上，下面的这种方法使用隐式 intent 更加合适。

- 内部发送广播的方法。通过指定内部定义的签名权限给 receiverPermission 参数，并让可靠的广播接收器声明使用这种签名权限。正确的编码方式请参阅安全开发范例的“内部广播接收器”章节，另外，执行这种广播发送方法需要遵循安全开发守则的“使用在内部应用里验证过的内部签名权限”章节。

#### B.3.2.6 敏感信息禁止包含在 StickyBroadcast 里

一般而言，当广播被接收器处理并接收后就会消失，然而，粘滞广播（以下粘滞广播包括有序粘滞广播）被接收器处理并接收后将不会从系统消失，并且通过 registerReceiver()可以再被接收。如果粘滞广播不再使用，可以用 removeStickyBroadcast()删除它。

粘滞广播预设是在隐式 intent 使用的，具有指定 receiverPermission 参数的广播将无法被发送。所以信息被粘滞广播发送可以发生在包括恶意软件的大量未明确的应用，因此，敏感信息不应该使用粘滞广播来传送。

#### B.3.2.7 注意：未指定接收器权限的有序广播不能被传送

未指定接收器权限参数的有序广播可以被包括恶意软件在内的大量不明确的应用接收。有序广播经常被用来接收从接收器返回的信息，并创建多个接收器一个接一个的执行处理，广播根据优先级发送给接收器。所以，如果高优先级的恶意软件率先接收到广播并执行 abortBroadcast()，广播将不会再传递给后面的接收器。

#### B.3.2.8 谨慎、安全地处理从广播接收器返回的结果数据

结果数据原则上都应该被安全地处理，考虑到接收到的可能是攻击数据，虽然根据广播接收器的不同类型安全风险会有很大差别。当发送方广播接收器是公共广播接收器的时候，它从大量不明确应用接收返回数据，所以它有可以接收到恶意软件的攻击数据。当发送方广播接收器是私有广播接收器的时候，它看起来没有风险，但其他应用接收到的数据有可能会被当作结果数据直接转发，所以结果数据没有经过验证就不应该认为是安全的。当发送方广播接收器是内部广播接收器的时候，它存在一定程度风险，所以需要一种安全方式来处理数据，考虑到结果数据可能是攻击数据。

#### B.3.2.9 间接提供资源时，应当采取相同的保护等级

当受权限保护的信息或功能资源间接提供给其他应用时，必须要求目标应用使用相同权限来保持保护标准。在 Android 权限安全模型里，只有应用直接访问受保护资源才进行权限分配管理，由于这个特性，获得的资源可能在没有声明权限的情况下提供给其他应用。实质上，这和重新委派权限一样，所以它被称为权限委派问题，参见“权限委派问题”章节。

### B.3.3 安全开发范例

#### B.3.3.1 私有广播接收器

私有广播接收器是最快的广播接收器，因为它只在应用内部传播。动态广播接收器无法被注册成私有，所以私有广播接收器只能由静态广播接收器组成。

接收广播时遵循的几个关键点：

1) 明确设置 exported 属性为 false

2) 谨慎、安全的处理接收到的 intent，即使它是从同一个应用程序内部发出的

3) 可以发送和返回敏感信息，因为请求是从同一个应用程序内部发出的

发送广播时遵循的几个关键点：

1) 在同一个应用内部使用类定义显式 intent 来调用接收器。

2) 可以发送敏感信息，因为目标接收器是在同一个应用程序内部。

3) 谨慎、安全的处理接收到的结果数据，即使数据是来自同一个应用程序内部的接收器

#### B.3.3.2 公共广播接收器

公共广播接收器是可以接收来自大量不明确的应用程序的广播的接收器,所以必须引起重视的是它可能接收到恶意程序的广播。公共广播接收器可以在静态广播接收器和动态广播接收器中使用。

接收广播时遵循的几个关键点：

1) 明确设置 exported 属性为 true

2) 谨慎、安全的处理接收到的 intent

3) 返回结果时不要包含敏感信息

代码示例：(略)

在动态广播接收器,注册/注销是在应用里通过调用 registerReceiver()或 unregisterReceiver()实现的，为了能够通过按钮操作执行注册/注销，按钮会被分配到公共接收器 Activity。因为动态广播接收器实例比公共接收器 Activity 长，所以它无法保持公共接收器 Activity 的成员变量。在这种情况下，保持动态广播接收器实例作为动态接收器服务的成员变量，然后从公共接收器 Activity 开始/结束动态接收器服务来直接地注册/注销动态广播接收器。

代码示例：（略）

当发送广播到公共广播接收器时，必须引起重视的是广播很有可能会被恶意程序接收。

发送广播时遵循的几个关键点：

1) 不要发送敏感信息

2) 接收结果数据时，必须谨慎、安全的处理。

代码示例：（略）

#### B.3.3.3 内部广播接收器

内部广播接收器只接收来自内部应用的广播，而不会接收其他任何应用的广播。它由多个内部应用组成，并被用来保护内部应用处理的信息或功能。内部广播接收器可以在静态广播接收器和动态广播接收器中使用。

接收广播时遵循的几个关键点：

1) 定义一个内部签名权限来接收广播

2) 声明使用内部签名权限来接收结果

3) 明确地设置 exported 属性为 true

4) 通过静态广播接收器定义要求使用内部签名权限

5) 要求使用内部签名权限来注册动态广播接收器

6) 验证内部应用是否定义内部签名权限

7) 谨慎、安全的处理接收的 intent，即使广播是由内部的一个应用发送来的

8) 可以返回敏感信息，因为发出请求的是内部应用

9) 当导出 APK 文件，使用与发送应用相同的开发密钥进行签名

代码示例：（略）

动态广播接收器是通过调用应用 registerReceiver()或 unregisterReceiver()执行注册/注销的。为了通过按钮操作执行注册/注销，按钮是由内部接收器 Activity 安排的。由于动态广播接收器实例比内部接收器 Activity 长，它无法维持作为内部接收器 Activity 的成员变量，所以，保持动态广播接收器实例作为动态接收器服务的成员变量,并从内部接收器 Activity 开始/结束动态接收器服务来直接注册/注销动态广播接收器。

代码示例：（略）

当发送广播给内部广播接收器，很重要的一点是要求广播接收方使用内部签名权限。所以必须引起重视的是它有一个限制就是 sendStickyBroadcast 无法使用。

发送广播时遵循的几个关键点：

1) 定义一个内部签名权限来接收结果

2) 声明使用内部签名权限来接收广播

3) 验证内部应用程序定义的内部签名权限

4) 可以返回敏感信息，因为请求方是另一个内部应用

5) 要求所有接收器使用内部签名权限

6) 谨慎、安全的处理接收的结果数据

7) 当导出 APK 文件，使用与目标应用相同的开发密钥进行签名

代码示例：（略）

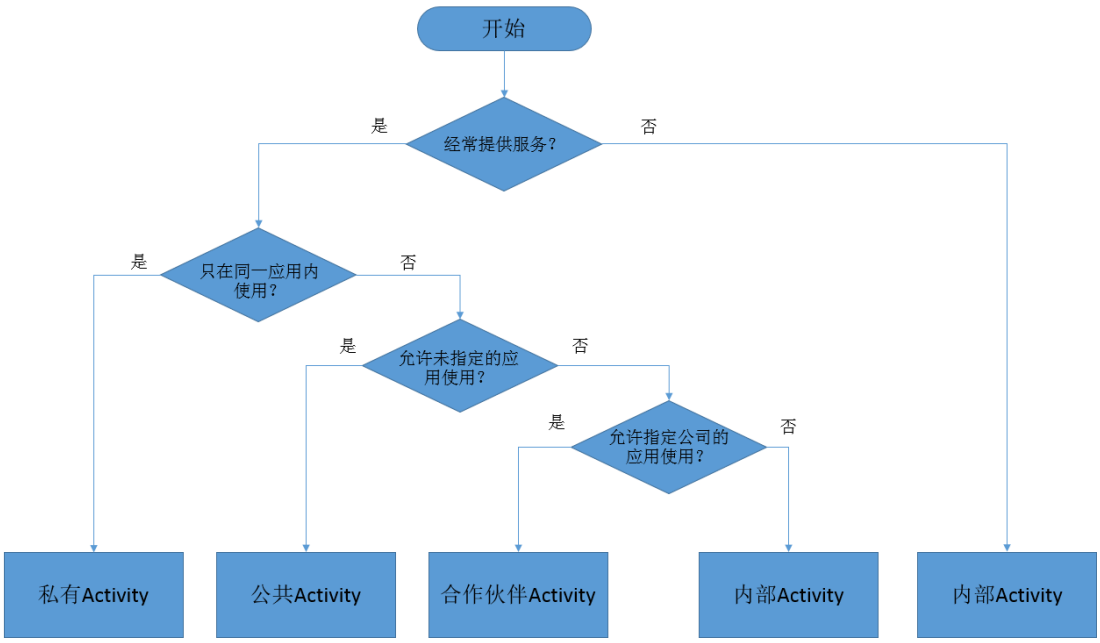
## B.4 创建/使用内容提供器

### B.4.1 内容提供器类型

由于内部提供器和 SQLite 数据库的接口很相似，所以经常错误地认为内容提供器和 SQLite 数据库是密切相关的。事实上，内容提供器是简单地为不同应用之间提供数据共享接口，必须注意的是它不干涉每个数据的存储格式。要存储数据到内容提供器，可以使用 SQLite 数据库和其他存储格式，如 XML 文件格式。

使用内容提供器的安全风险和应对措施，根据内容提供器的使用场景不同而有所区别。基于内容提供器的使用场景，我们将内容提供器分成 5 个类型，你可以从下表找到你要创建的内容提供器类型：

| 类型 | 定义描述 |
| --- | --- |
| 私有内容提供器 | 无法被其他应用使用的内容提供器，所以它是最安全的 |
| 公共内容提供器 | 假设会被大量未明确的应用使用的内容提供器 |
| 合作伙伴内容提供器 | 可以被信任的合作伙伴公司开发的指定应用使用的内容提供器 |
| 内部内容提供器 | 只能被其他内部应用使用的内容提供器 |
| 临时许可的内容提供器 | 它基本上是私有内容提供器，但允许指定的应用访问特定的 URI |



### B.4.2 安全开发守则

#### B.4.2.1 只在某个应用内部使用的内容提供器必须设置成私有

只在单个应用内部使用的内容提供器没有必要给其他应用访问，开发人员并不经常考虑这类访问会被攻击。系统主要使用内容提供器进行数据共享，默认是公共内容提供器。只在某个应用内部使用的内容提供器必须明确地设置成私有，并且只能是私有内容提供器。在 Android 2.3.1(API Level 9)及以后的系统，一个内容提供器可以通过在提供器元素里指定 android:exported="false"来设置成私有。

#### B.4.2.2 谨慎、安全地处理接收到的请求参数

根据内容提供器类型不同而安全风险有所差异，但当处理请求参数时，首先要做的是输入检验。虽然内容提供器的每个方法具有接收 SQL 语句的组件参数，事件上它只是简单传递系统里的任意字符串，所以，必须注意的是内容提供器端需要预测意外的参数会被提交。

由于公共内容提供器可以接收到不可信源的请求，所以会被恶意软件攻击，另一方面，私有内容提供器从来不会直接地接收到其他应用的任意请求，但在目标应用里的某个公共 Activity 可能转发一个恶意的 intent 给私有内容提供器，所以不能认为私有内容提供器不会接收到任何恶意输入。

由于其他内容提供器也存在接收到被转发过来的恶意 Intent 的风险，所以也有必要在这类请求进行输入检验。

#### B.4.2.3 使用在内部应用里验证过的内部签名权限

在创建内容提供器时，确保通过定义一个内部签名权限来保护你的内部内容提供器。因为在 AndroidManifest.xml 文件内定义一个权限或声明一个权限请求不能提供足够的安全性，请务必参阅后面章节“如何使用内部定义的签名权限在内部应用间进行通信”。

#### B.4.2.4 返回结果信息时，注意目标应用泄漏信息风险

假定执行 query()或 insert()时，指针或 Uri 会作为结果信息返回给发送请求的应用，当结果信息里包含敏感信息时，这些信息可能会被目标应用泄漏出去。假定执行 update()或 delete()时，若干个更新/删除记录会作为结果信息返回给发送请求的应用。在某些罕见案例中，根据应用声明，那些更新/删除记录可能是敏感的，所以也要注意这类情况。

#### B.4.2.5 间接提供资源时，应当采取相同的保护等级

使用权限来保护的信息资产或功能资产，也会二次的提供给其他应用，要确保访问这些资产需要有相同的权限保护。在 Android OS 权限安全模型里，只有被授予合适权限的应用才能直接访问被保护的资产，然而，这里存在漏洞，因为某个拥有访问资产权限的应用可以作为代理并允许访问没有权限的应用，实际上，这与重新分配权限一样，因此，它也被称作“权限重分配”问题。详见后续章节“权限重分配问题”。

#### B.4.2.6 谨慎、安全地处理从内容提供器返回的结果数据

风险根据内容提供器的类型而异，但处理某个结果数据时，第一个要做的是输入校验。如果目标内容提供器是公共内容提供器，假扮成公共内容提供器的恶意软件有可能返回带有攻击的结果数据。另外，如果目标内容提供器是私有内容提供器，那么它的风险会小一些，因为它接收的返回数据是来自同一个应用。但无法保证私有内容提供器不会接收到任何恶意的输入数据。其他类型的内容提供器也有接收到恶意数据的风险，因此有必要对那些返回数据执行输入校验。

### B.4.3 安全开发范例

#### B.4.3.1 创建/使用私有内容提供器

私有内容提供器只能被单个应用使用，是最安全的内容提供器。但是，有必要引起注意的是内容提供器的私有设置无法在 Android2.2（API Level 8）及更早的系统里运行。

创建私有内容提供器时遵循的几个关键点：

1） 不要在 Android2.2（API Level 8）及以前的系统里使用私有内容提供器

2） 明确地设置 exported 属性为 false

3） 谨慎、安全的处理接收到的请求数据，即使数据是来自同一个应用程序。

4） 可以传送敏感信息，因为它只在同一个应用内部传送和接收。

代码示例：（略）

1) 使用私有内容提供器时遵循的几个关键点：

2) 可以传送敏感信息，因为目标内容提供器在同一个应用内部。

3) 谨慎、安全的处理接收到的结果数据，即使数据是来自同一个应用程序。

代码示例：（略）

#### B.4.3.2 创建/使用公共内容提供器

公共内容提供器是批假设会被大量未明确的应用使用的内容提供器，必须注意的是由于它没有指定客户端，可能会被攻击或篡改。举个例子，数据可以通过 select()获取，通过 update()更新，或者可以通过 insert()/delete()插入/删除一些伪造的数据。

此外，当使用不是由 Android OS 提供的自定义公共内容提供器，必须注意的是请求参数可能会被假装成自定义内容提供器的恶意软件接收到，并且可能发送攻击性的结果数据。Android OS 提供的联系人和媒体库也是公共内容提供器，但恶意软件无法假扮它们。

创建公共内容提供器时遵循的几个关键点：

1） 明确地设置 exported 属性为 true

2） 谨慎、安全的处理接收到的请求数据

3） 返回结果时不要包含敏感信息

代码示例：（略）

使用公共内容提供器时遵循的几个关键点：

1） 不要发送敏感信息

2） 谨慎、安全的处理接收到的结果数据

代码示例：（略）

#### B.4.3.3 创建/使用合作伙伴内容提供器

合作伙伴内容提供器只能被特定的应用程序使用，安卓系统里包括合作伙伴公司的应用和内容应用，并被用来保护在合作伙伴应用与内部应用之间处理的信息和特性。

创建合作伙伴内容提供器时遵循的几个关键点：

1） 明确地设置 exported 属性为 true

2） 检查自己的白名单里是否已经注册了发送请求的应用的证书

3） 谨慎、安全的处理接收到的请求数据，即使它来自合作伙伴的应用

4） 可以返回已批准透露给合作伙伴应用的信息

代码示例：（略）

使用合作伙伴内容提供器时遵循的几个关键点：

1） 检查自己的白名单里是否已经注册了目标应用的证书

2） 可以返回已批准透露给合作伙伴应用的信息

3） 谨慎、安全的处理接收到的结果数据，即使它来自合作伙伴的应用

代码示例：（略）

#### B.4.3.4 创建/使用内部内容提供器

内部内容提供器不能在除内部应用之外的其他应用中使用。

创建内部内容提供器时遵循的几个关键点：

1） 定义一个内部签名权限

2） 要求内部签名权限

3） 明确的设置 exported 属性为 true

4） 检查内部应用是否定义内部签名权限

5） 检查参数的安全性，即使是内部应用发送的请求

6） 可以返回敏感信息，因为是内部应用发送的请求

7） 当导出 APK 文件，使用与请求应用程序相同的开发密钥进行签名

代码示例：（略）

使用内部内容提供器时遵循的几个关键点：

1） 声明要使用内部签名权限

2） 检查内部应用是否定义内部签名权限

3） 检查目标应用是否使用内部证书签名

4） 可以发送敏感信息，因为目标应用在内部

5） 谨慎、安全的处理接收到的结果数据，即使它来自某个内部应用

6） 当导出 APK 文件，使用与目标应用程序相同的开发密钥进行签名

代码示例：（略）

#### B.4.3.5 创建/使用临时内容提供器

通过发送一个指向目标应用的特定标志的 intent，向这些应用提供临时访问权限，内容提供器这方的应用可以主动地开通访问权限给其他应用，并且也可以被动地给请求临时访问权限的应用开通权限。

创建临时内容提供器时遵循的几个关键点：

1） 不要在 Android2.2（API Level 8）及以前的系统里使用临时内容提供器

2） 明确地设置 exported 属性为 false

3） 使用 grant-uri-permission 指定临时授权访问的路径

4） 谨慎、安全的处理接收到的请求数据，即使它来自那个临时授权访问的应用

5） 可以返回信息给经过批准的临时访问应用

6） 给授权临时访问的 intent 指定 URI

7） 给授权临时访问的 intent 指定访问权限

8） 给应用发送一个显式 intent 来授权临时访问

9） 给请求临时访问权限的应用返回 intent

代码示例：（略）

使用临时内容提供器时遵循的几个关键点：

1） 不要发送敏感信息

2） 当接收结果时，谨慎、安全地处理结果数据

代码示例：（略）

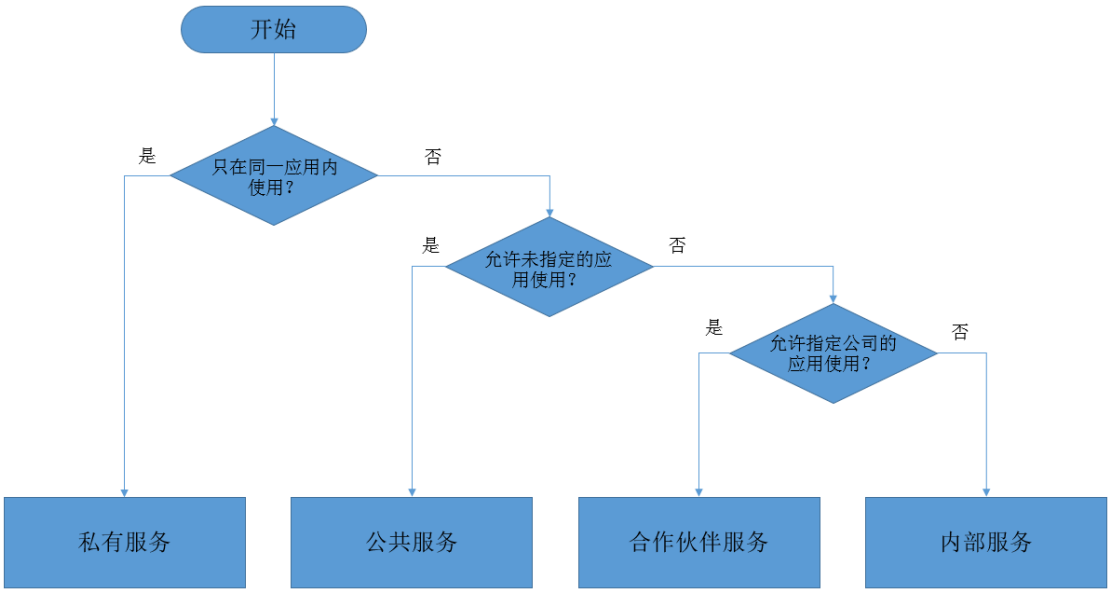
## B.5 创建/使用服务

### B.5.1 服务类型

使用服务的安全风险及防范措施根据服务的使用方式不同而有所区别。通过下面的表格 B-7 及图示，你可以找到你想创建的各种服务类别。因为基于服务的创建方式安全编码最佳实践也各有不同，因此，我们也将讲解服务的安全实现。

表 B-7：服务类型的定义

| 类型 | 定义 |
|---|---|
| 私有服务 | 无法被其他应用使用的服务，是最安全的服务 |
| 公共服务 | 假设可被大量不明确的应用使用的服务 |
| 合作伙伴服务 | 只能被由可信伙伴公司的指定应用使用的服务 |
| 内部服务 | 只能被其他内部应用使用的服务 |




这里有多种服务的实现方式，你可以选择与你想创建的服务类型相匹配的实现方式。下面表 B-8 列出了 5 种实现方式，“OK”代表可以组合使用，其他代表不能或很难组合。

表 B-8：5 种服务实现类型及组合

| 类别 | 私有服务 | 公共服务 | 合作伙伴服务 | 内部服务 |
|---|---|---|---|---|
| 启动服务类型 | **OK*** | OK | - | OK |
| Intent 服务类型 | OK | **OK*** | - | OK |
| 本地绑定类型 | OK | - | - | - |
| Messenger bind type | OK | OK | - | **OK*** |

| | | | | |
|---|---|---|---|---|
| AIDL 绑定类型 | OK | OK | **OK*** | OK |

更具体的服务实现方法请参阅后面章节“如何实现服务”以及各服务类型的代码示例。

### B.5.2 安全开发守则

#### B.5.2.1 只在应用内部使用的服务必须设置成私有

只在某个应用内部（或者相同的 UID）使用的服务必须设置成私有，它避免应用从其他不明确的应用接收 Intent，并且最终阻止安全危害，比如应用功能被使用或应用行为异常。

当在 AndroidManifest.xml 文件里定义一个服务时，你必须要做的是将 exported 的属性值设置为 false。

此外，还有一种不常见的情况是，只在应用内部使用的服务不要设置 Intent-filter。原因是，由于 Intent-filter 的特性，在其他应用里的公共服务可能会被意外的调用，虽然你要调用的是应用内部的私有服务。

#### B.5.2.2 谨慎、安全地处理接收到的数据

和 Activity 一样，当处理某个接收到的 Intent 数据，你要做的第一件事是输入检验。同样在服务的用户方，有必要核实来自服务的结果信息的安全性。详细信息参阅前面章节“谨慎、安全地处理接收到的 Intent”和“谨慎、安全地处理来自请求 Activity 的返回数据”。

在服务安全方面，还需要小心地实行调用方法和信息交换数据。详情参阅前面 2.2 章节“谨慎、安全地处理输入的数据”。

#### B.5.2.3 使用在内部应用里验证过的内部签名权限

创建服务时，一定要通过定义内部签名权限来保护你的内部服务。因为在 AndroidManifest.xml 文件里定义权限或声明权限请求不能提供足够的保护,请务必参阅后面章节“如何使用内部定义的签名权限在内部应用间进行通信”。

#### B.5.2.4 在 onCreate 函数不要下结论服务是否提供功能

诸如 Intent 参数校验或内部签名权限检验等安全检查不应该包含在 onCreate 函数,因为在服务运行期间接收到新的请求，onCreate 进程是不会执行的。所以，当要实现的服务是由 startService 启动的，务必要通过 onStartCommand 进行判断，如果是使用 IntentService,应该由 onHandleIntent 进行判断。同样，当要实现的服务是由 bindService 启动的，应该由 onBind 进行判断。

#### B.5.2.5 返回结果信息时，注意目标应用泄漏信息风险

取决于服务的类型，目标应用返回结果信息的可靠性是有区别的。必须认识到鉴于目标应用可能是恶意软件而引起信息泄漏的严重性。

更多详细信息查阅 Activity 的“返回结果时，注意来自目标应用造成信息泄露的可能性”章节。

#### B.5.2.6 如果目标服务是固定的，要使用显式 intent

通过显式 Intent 使用某个服务时，假设 Intent-filter 的定义是一样的，Intent 是被发送给早先就已安装的服务。如果事前安装的恶意软件有故意定义成一样的 Intent-filter， Intent 就会被发送给恶意软件，那么信息泄漏就会发生。另外，通过显式 Intent 使用某个服务时，只有预期的服务才能接收 Intent，所以更安全一些。

还有其他方面需要注意，请查阅“3.2.8 若目的 Activity 已预定，使用显式 intents”。

#### B.5.2.7 验证与其他公司应用有关联的目标服务

当与其他公司的应用连接时，务必确认下白名单。可以这么做：在你的应用里保存一份其他公司的证书哈希，并与目标应用的证书哈希做比较。这可以阻止某个恶意应用欺骗 intent。具体的实施方法请查看 “6.3.3 创建/使用合作伙伴服务”章节。

#### B.5.2.8 间接提供资源时，应当采取相同的保护等级

使用权限保护的某个信息或功能资产被间接的提供给其他应用时，你要确保必须需要同样的权限要求才能访问。在安卓系统的权限安全模型里，只有被授予合适权限的某个应用才能直接访问受保护的资产。然而，这里存在漏洞，因为某个拥有访问资产权限的应用可以作为代理并允许访问没有权限的应用，实际上，这与重新分配权限一样，因此，它也被称作“权限重分配”问题。详见后续章节“权限重分配问题”。

#### B.5.2.9 敏感信息不应该最大化传送

不应该向不可信的第三方传送敏感信息。当和某个服务交换敏感信息时需要考虑信息泄漏的风险，应该假设在 Intent 里传送给某个公共服务的所有数据都可能会被恶意第三方获取到。此外，根据实现方式，当发送 Intents 给合作伙伴服务或内部服务时也存在有各种各样的信息泄漏风险。

永远不传送敏感数据是阻止信息泄漏最完美的解决方案，因此，你应该尽可能多的限制敏感信息的发送。当必须发送敏感信息时，最佳做法是只发送给某个可信服务并确保这些信息不会通过 LogCat 泄漏出去。

### B.5.3 安全开发范例

#### B.5.3.1 创建/使用私有服务

当使用只能应用内部使用的私有服务时，只需要在类里面使用显式 Intent，就不担心会将它发送给任何其他应用了。

创建私有服务时遵循的几个关键点：

1) 明确设置 exported 属性为 false

2) 谨慎、安全的处理接收到的 intent，即使这个 intent 是同一个应用程序发出的。

3) 可以传送敏感信息，因为是发送请求的是同一个应用。

代码示例：（略）

使用私有服务时遵循的几个关键点：

1) 使用类指定显式 intent 来调用同一个应用里的服务

2) 以传送敏感信息，因为目标服务在同一个应用里

3) 谨慎、安全的处理接收到的结果数据，即使数据来自同一个应用程序里的服务

代码示例：（略）

#### B.5.3.2 创建/使用公有服务

公共服务是指那些假定可以被大量不明确的应用使用的服务，因此需要引起注意的是它可能会接收到恶意软件发送过来的信息（或 intent）。当使用公共服务时，也要注意发送出去的信息（或 intent）可能会被恶意软件接收。

创建公共服务时遵循的几个关键点：

1) 明确设置 exported 属性为 true

2) 谨慎、安全的处理接收到的 intent

3) 返回结果时不要包含敏感信息

代码示例：（略）

使用公共服务时遵循的几个关键点：

1) 不要发送敏感信息

2) 谨慎、安全的处理接收到的结果数据

代码示例：（略）

#### B.5.3.3 创建/使用合作伙伴服务

合作伙伴服务是指只能被特定应用使用的服务，系统由合作伙伴公司应用和内部应用组成，这用来保护某个合作伙伴应用与内部应用之间处理的信息和功能特性

创建合作伙伴服务时遵循的几个关键点：

1) 不要定义 intent filter 并明确地将 exported 属性设置为 true

2) 核实请求应用的证书是在白名单里注册过的

3) 不要使用 onBind（onStartCommand, onHandleIntent）来识别发送请求的应用是否是合作伙伴的。

4) 谨慎、安全的处理接收到的 intent，即使这个 intent 是从合作伙伴的应用程序发出的

5） 只返回已批准暴露给合作伙伴应用程序的信息

另外，关于更多如何核实白名单里指定的目标应用的证书哈希值，请参阅“如何核实某个应用证书的哈希值”。

代码示例：（略）

使用合作伙伴服务时遵循的几个关键点：

1） 核实目标应用的证书是否在白名单里注册过

2） 只返回已批准暴露给某个合作伙伴应用程序的信息

3） 使用显式 Intent 来调用某个合作伙伴的服务

4） 谨慎、安全的处理接收到的结果数据，即使数据是从某个合作伙伴的应用程序发出的

代码示例：（略）

#### B.5.3.4 创建/使用内部服务

内部服务是指只能在内部应用里使用而禁止被其他应用使用的服务。它们使用内部开发的应用，目的是安全地共享信息和功能特性。

创建内部服务时遵循的几个关键点：

1） 定义一个内部签名权限

2） 要求使用内部签名权限

3） 不要定义 intent filter 并明确地将 exported 属性设置为 true

4） 核实某个内部应用里定义的内部签名权限

5） 谨慎、安全的处理接收到的 intent，即使这个 Intent 是从某个内部应用程序发出的

6） 可以返回敏感信息，因为发送请求的是内部应用

7） 当导出 APK 文件，使用与请求应用程序相同的开发密钥进行签名

代码示例：（略）

使用内部服务时遵循的几个关键点：

1） 声明要使用内部签名权限

2） 核实某个内部应用里定义的内部签名权限

3） 核实目标应用是使用内部证书签名的

4） 可以发送敏感信息，因为目标是内部应用

5） 使用显式 Intent 来调用某个内部服务

6） 谨慎、安全的处理接收到的结果数据，即使数据是从某个内部应用程序发出的

7） 当导出 APK 文件，使用与目标应用程序相同的开发密钥进行签名

代码示例：（略）

## B.6 使用 SQLite

在下文中，我们将提出一些在使用 SQLite 创建/操作数据库时安全注意事项。重点是合理设置数据库文件的访问权限，以及 SQL 注入的应对措施。这里不包括允许从外部直接读写数据库文件（多应用间共享数据库）的数据库，而是后端内容提供器及应用内部自己使用的数据库。另外，如果不处理那么多敏感信息，建议采用以下提到的保护措施，这里假设是处理一定程度的敏感信息。

### B.6.1 安全开发守则

#### B.6.1.1 正确地设置 DB 文件位置和访问权限

就关于数据库文件数据的保护而言，数据库文件位置和访问权限设置是需要一起考虑的要素。举个例子，比如它被部署在一个访问权限无法设置的地方，即使正确设置了文件访问权限，某个数据库文件也可以被任务人访问，那么最终允许意料之外的访问。下面是一些关于正确部署和访问权限配置的关键点，以及它们的实现方法。

关于位置和访问权限配置，考虑到保护数据库文件方面，有必要按照下面的 2 点来执行。

1) 位置。文件路径定位可以通过 Context#getDatabasePath(字符串名字)获得，或者在某些情况下，目录可以通过 Context#getFilesDir 获得。

2) 访问权限。设置成 MODE_PRIVATE 模式（它只能被创建文件的应用访问）。

按照以上 2 点实施后，可以被创建不能被其他应用访问的数据库文件，这里是一些执行他们的方法。

1) 使用 SQLiteOpenHelper

2) 使用 Context#openOrCreateDatabase

当创建数据库文件时，可以使用 SQLiteDatabase#openOrCreateDatabase。然而，在某些安卓智能机设备使用这种方法时，将会创建可以被其他应用读取的数据库文件。所以建议避免使用这种方法，并使用其他方法，上述两种方法的特性如下所示：

1) 使用 SQLiteOpenHelper。当使用 SQLiteOpenHelper，开发人员无须担心很多事情。创建来自 SQLiteOpenHelper 的一个类，并指定数据库名称，然后符合上述安全要求的数据库文件会自动被创建。关于如何使用详见“7.2 创建/操作数据库”章节。

2) 使用 Context#openOrCreateDatabase。当使用 Context#openOrCreateDatabase 方法，应该通过选项来指定文件访问权限，即明确地指定 MODE_PRIVATE。

关于文件位置分配，指定数据库名的做法和 SQLiteOpenHelper 是一样的，在符合上述安全要求的文件会被自动创建。然而，全路径也可以被指定，所以要注意当指定到 SD 卡里，即使使用 MODE_PRIVATE，其他应用也能访问到。

可以参考以下 3 种访问权限设置包括 MODE_PRIVATE。MODE_WORLD_READABLE 和 MODE_WORLD_WRITABLE 可以被 OR 操作角色一起指定。由于除了 MODE_PRIVATE，其他方法在 API 17 级及以上有差异，你需要和应用需求一起考虑来小心使用它们。

1) MODE_PRIVATE：只有创建它的应用才能读写

2) MODE_WORLD_READABLE：创建它的应用能读写，其他应用只能读

3) MODE_WORLD_WRITABLE：创建它的应用能读写，其他应用只能写

#### B.6.1.2 与其他应用共享 DB 数据时，使用内容提供器进行访问控制

与其他应用共享数据库数据的方法是用 WORLD_READABLE 和 WORLD_WRITABLE 来创建数据库文件，给其他应用直接访问。然而，这种方法不能限制应用访问数据库或操作数据库，因此，数据可能会被不可预期的第三方读取或写入，导致会发生一些数据保密或数据完整的安全问题，甚至是成为恶意软件的攻击目标。

如上述所言，在安卓里与其他应用共享数据库数据时，强烈建议使用内容提供器。使用内容提供器有一些优点，不仅有实现数据库的访问控制的安全优点，还有在内容提供器隐藏数据库结构的设计优点。

#### B.6.1.3 在进行处理变量参数等 DB 操作时，必须使用占位符

就阻止 SQL 注入的意义而言，当合并任意的输入值到 SQL 表达式，应该使用占位符。下面有使用占位符来执行 SQL 表达式的 2 种方法

1) 使用 SQLiteDatabase#compileStatement() 获取 SQLiteStatement，之后，使用 SQLiteStatement#bindString() 或 bindLong()任命参数给占位符。

2) 当在 SQLiteDatabese 类里调用 execSQL()、insert()、update()、delete()、query()、rawQuery()和 replace()等，使用具有占位符的 SQL 表达式。

另外，执行 SELECT 命令时，通过使用 SQLiteDatabase#compileStatement()，会有一个限制，即只有第一个参数会被获取作为 SELECT 命令的结果，所以使用 SELECT 会被限制。

在其他方法中，根据应用要求，最好提前检查交给占位符的数据内容。下面是各种方法的进一步说明。

当使用 SQLiteDatabase#compileStatement()：数据是按下面步骤提交给占位符的。

1) 通过 SQLiteDatabase#compileStatement()获取包含占位符的 SQL 表达式

2) 通过使用类似 bindLong()和 bindString()的方法来设置占位符

3) 通过类似 ExecSQLiteStatement 对象的 execute()方法来执行 SQL

SQLite 数据库提供了 2 种数据库操作方法。一种是使用 SQL 表达式，另外一种是不使用 SQL 表达式。使用 SQL 表达式的方法是 SQLiteDatabase# execSQL()/rawQuery()。它的执行步骤如下：

1) 准备包含占位符的 SQL 表达式

2) 创建数据分配给占位符

3) 发送 SQL 表达式和数据作为参数并执行

此外，SQLiteDatabase#insert()/update()/delete()/query()/replace()是那种没有使用 SQL 表达式的方法。当使用它们，数据应该按照下面的步骤：

1） 如果有数据要插入/更新到数据库，注册到 ContentValues。

2) 将 ContentValues 作为参数发送并执行

### B.6.2 创建/操作数据库

在安卓应用里处理数据库时，合理的部署数据库文件和访问权限设置（拒绝其他应用访问的配置）可以通过使用 SQLiteOpenHelper 来完成。这里的例子是一个启动时会创建数据库的简单应用，并通过 UI 执行搜索/增加/修改/删除数据。代码示例是关于 SQL 注入的应对措施，避免从外部输入的不正确的 SQL 语句被执行。如图 B-10 所示。

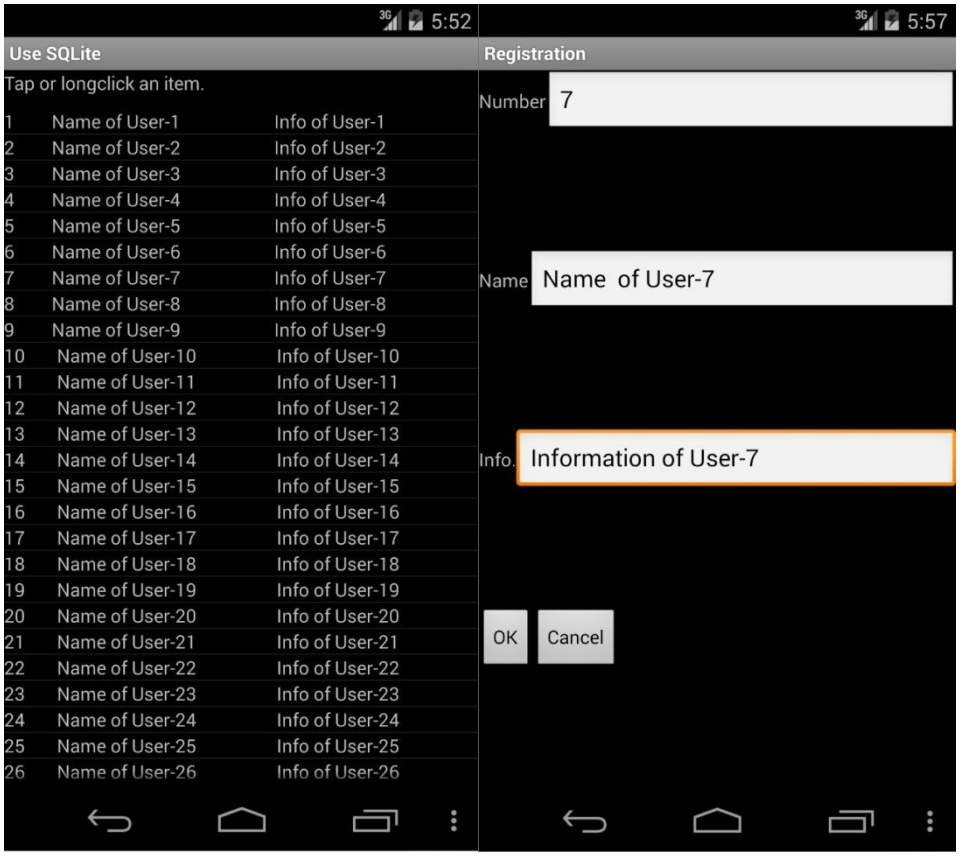



图 B-10：数据库示例图

创建/操作数据库时遵循的几个关键点：

1） 创建数据库时必须使用 SQLiteOpenHelper

2） 使用占位符

3） 根据应用需求校验输入的值

代码示例：（略）

T/ZSA 3001.01-2016
企业移动智能终端应用开发、安装、运行管控机制指南
APP developing ,installing and running management and control mechanism(guideline) for enterprise mobile smart terminal
中关村标准化协会
2016年12月发布