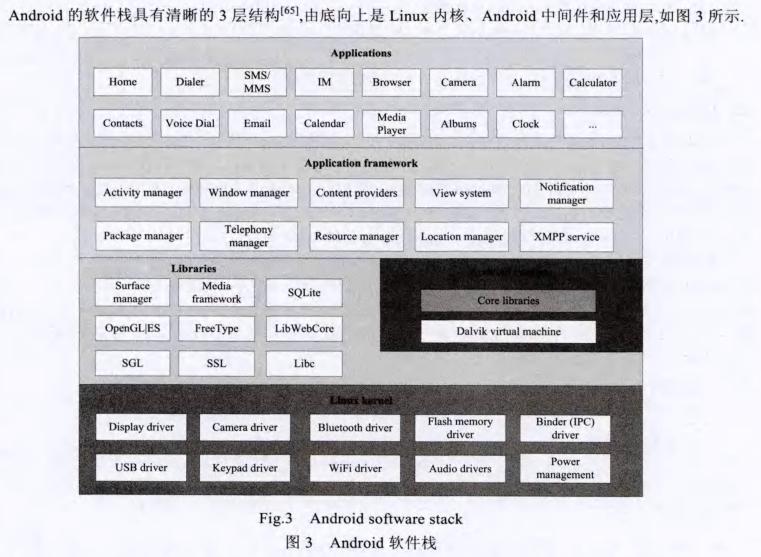
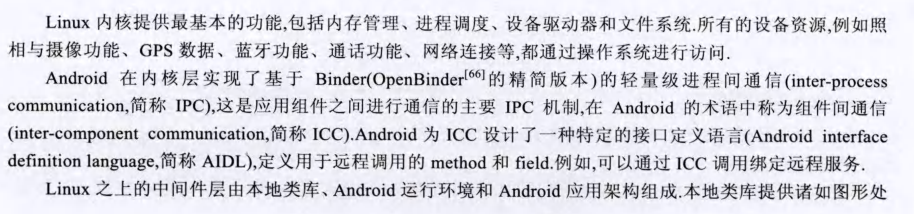
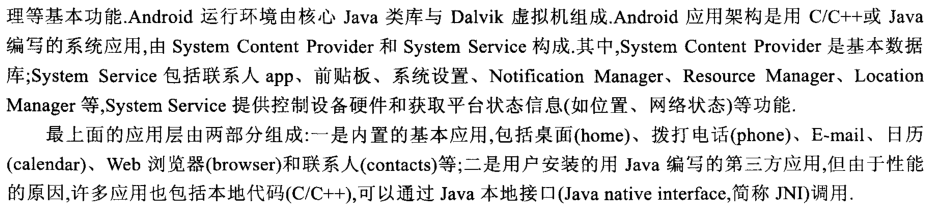
# Android体系结构

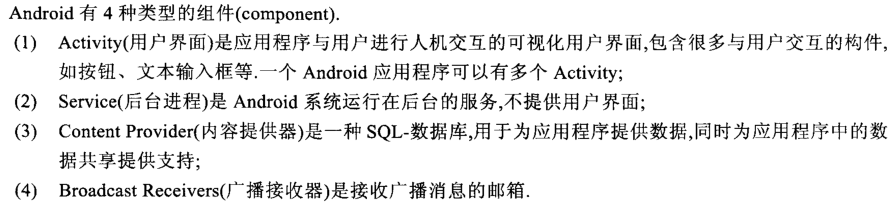
## Android软件栈

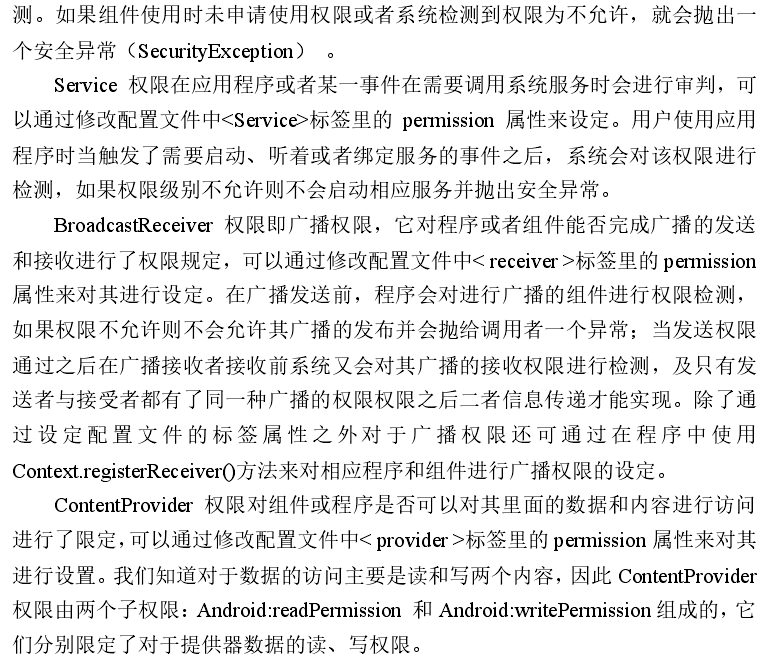




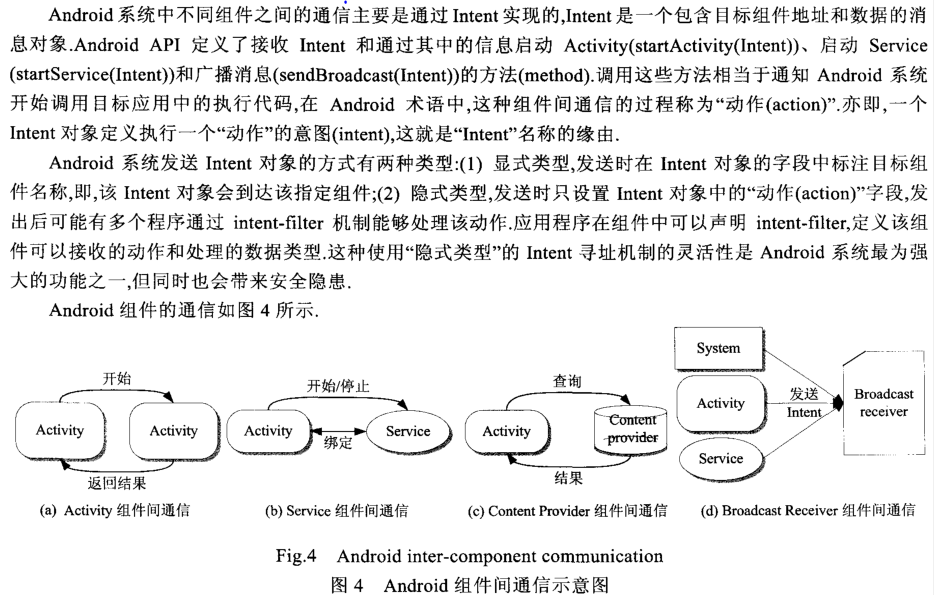


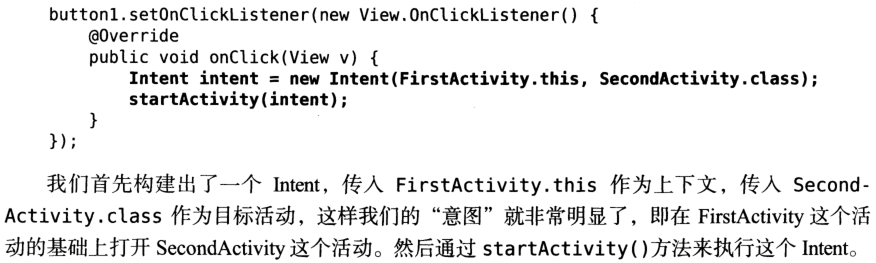
## Android组件

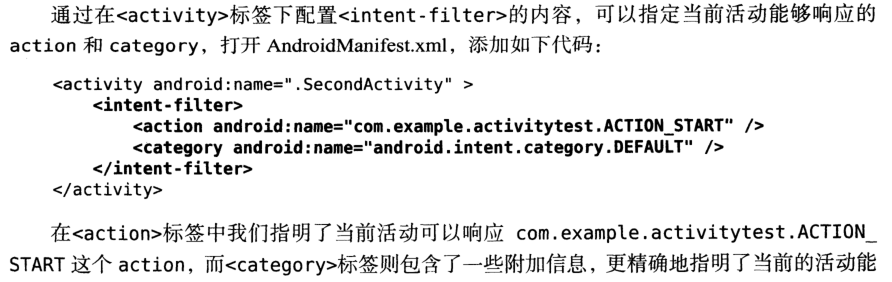




## Intent与组件通信

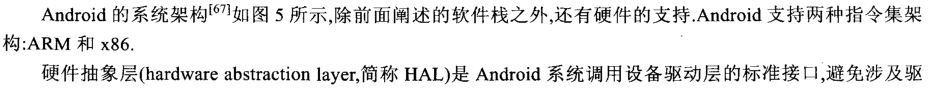


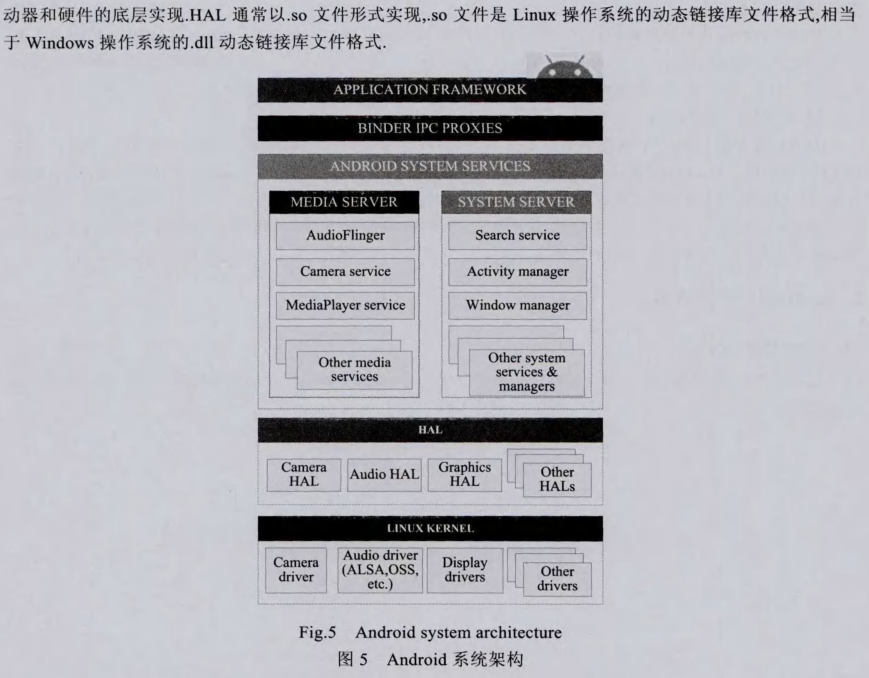






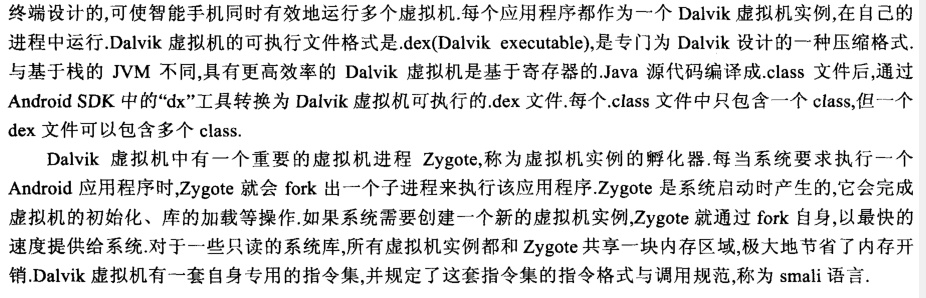
## Android系统架构





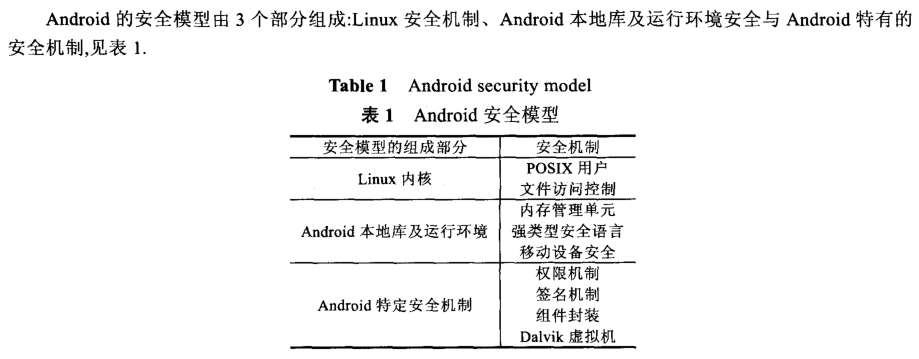
## Dalvik虚拟机



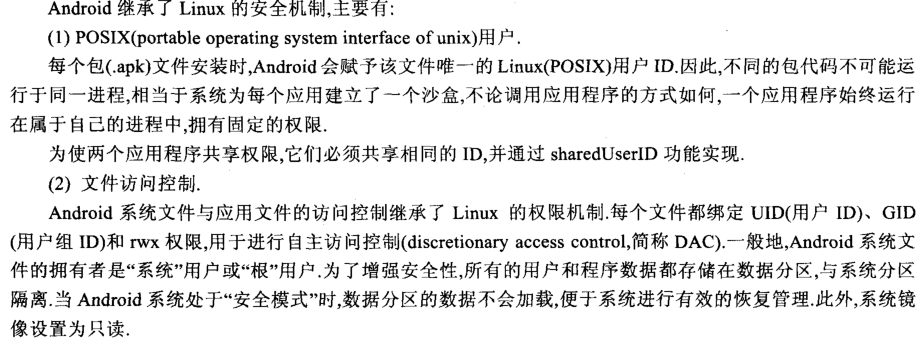


# Android安全机制

## Android安全模型

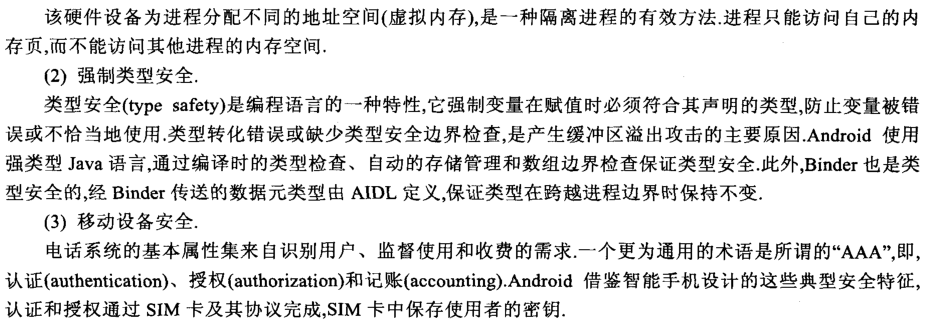


### Linux安全机制

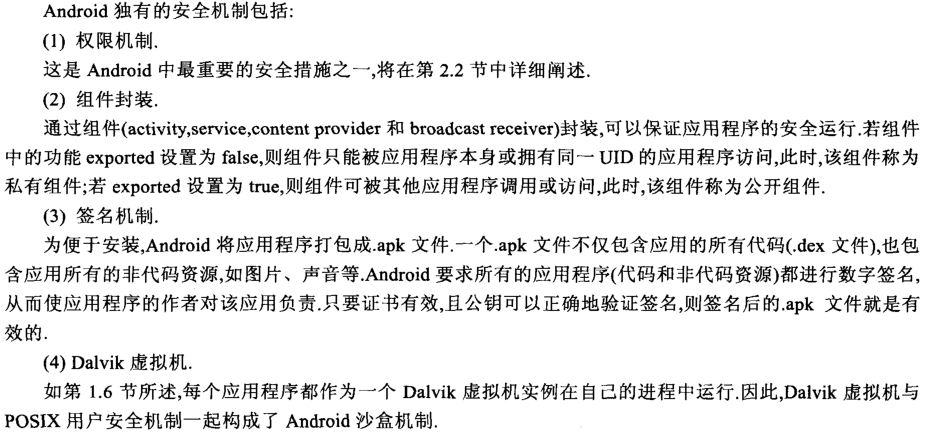


### Android本地库及运行环境安全

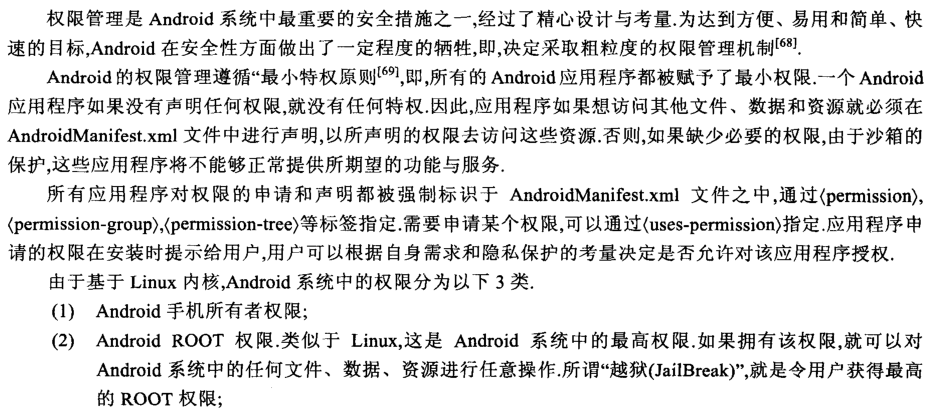


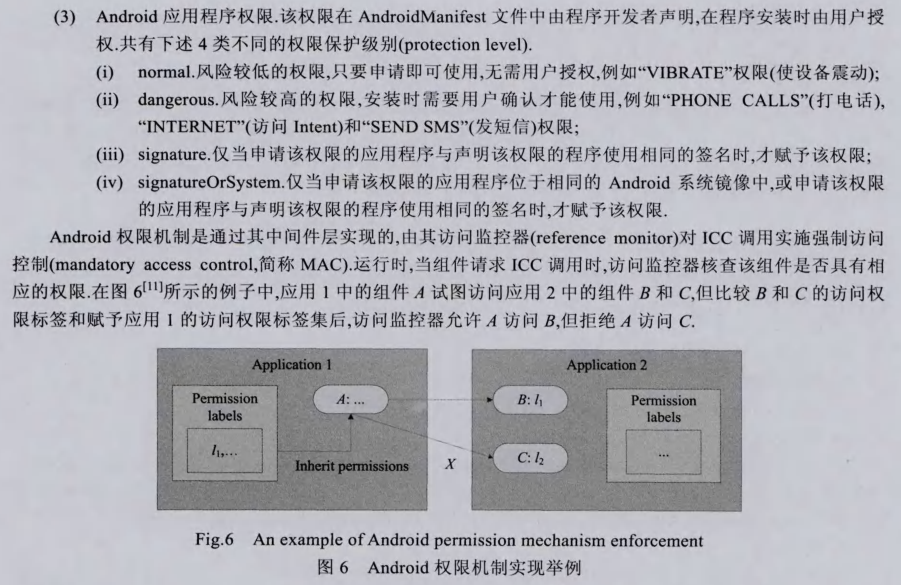


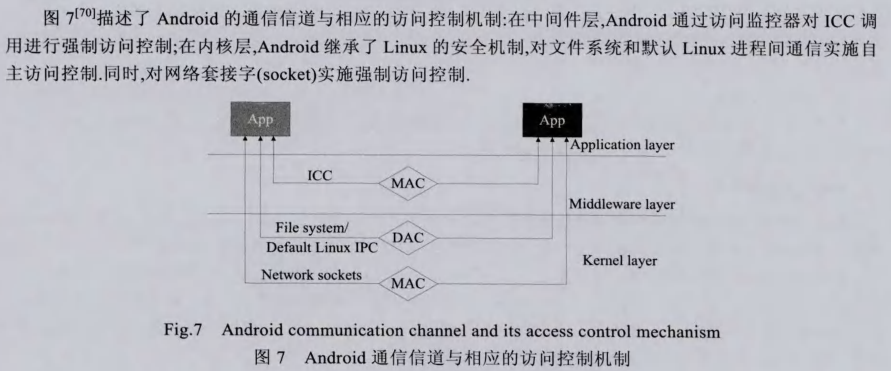
### Android特定安全机制



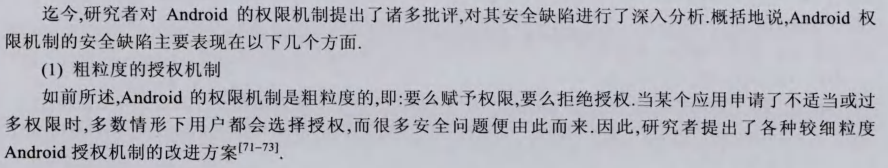
## Android权限机制

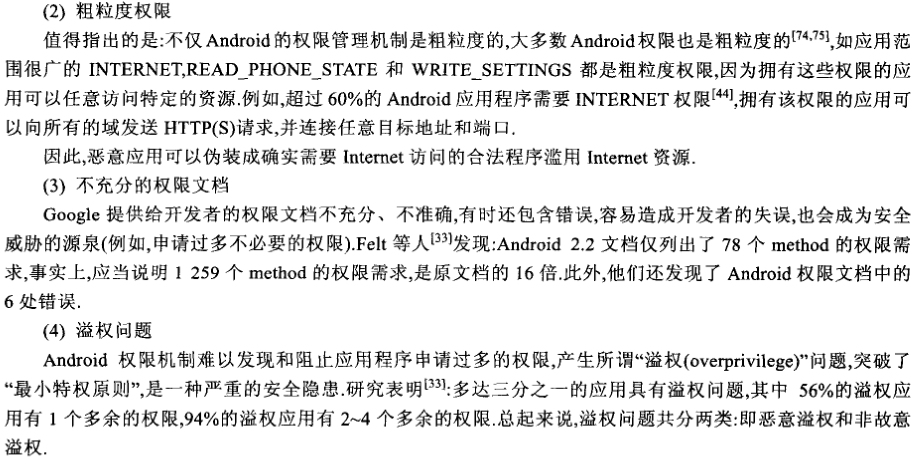


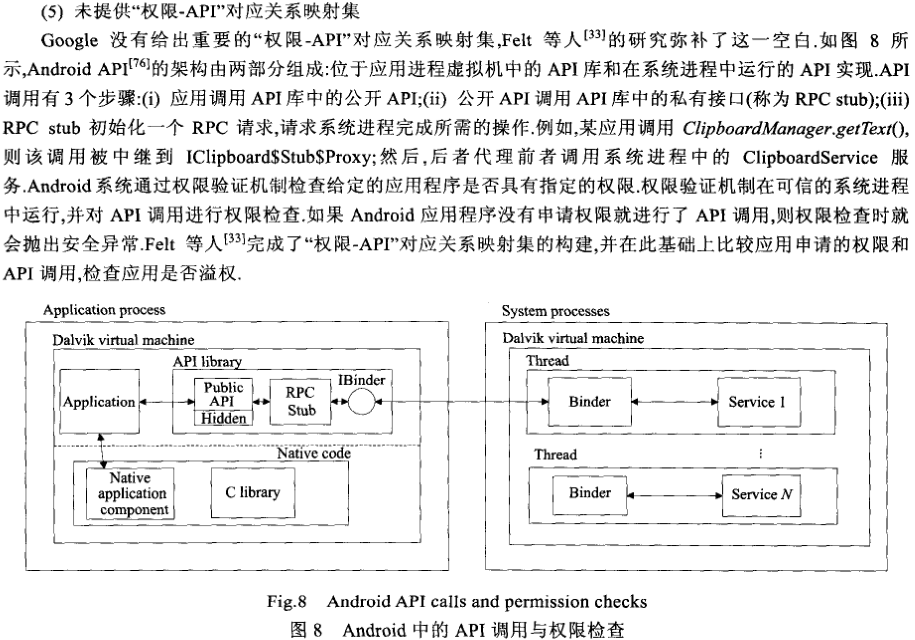




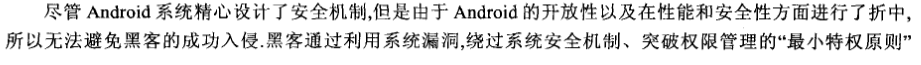
## Android权限机制的安全缺陷

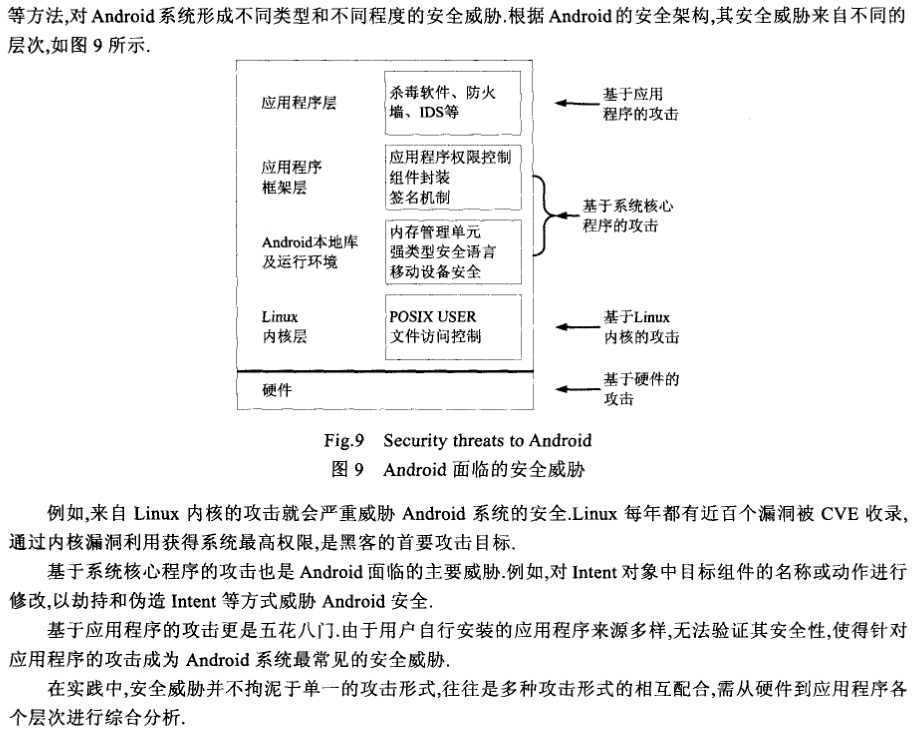






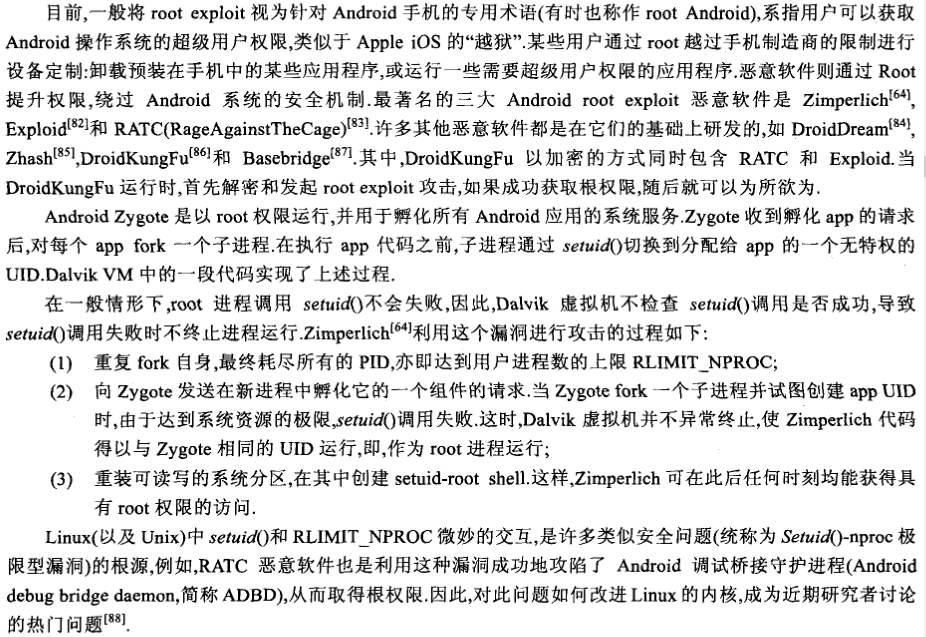
## Android的安全威胁



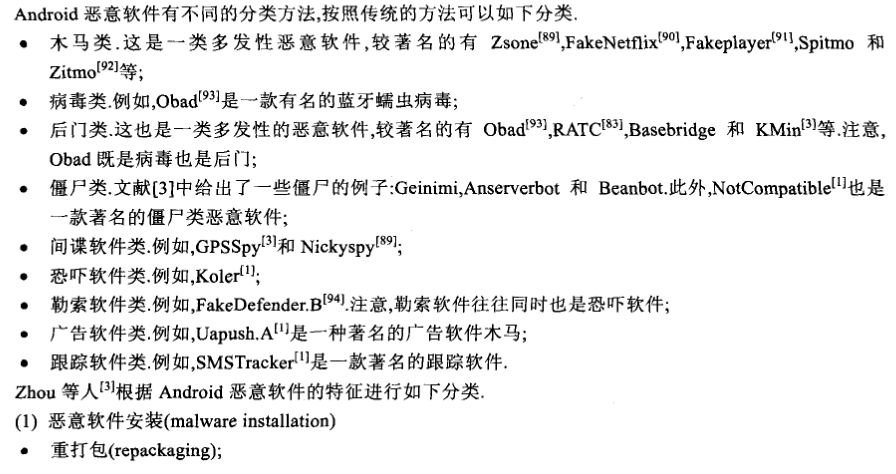


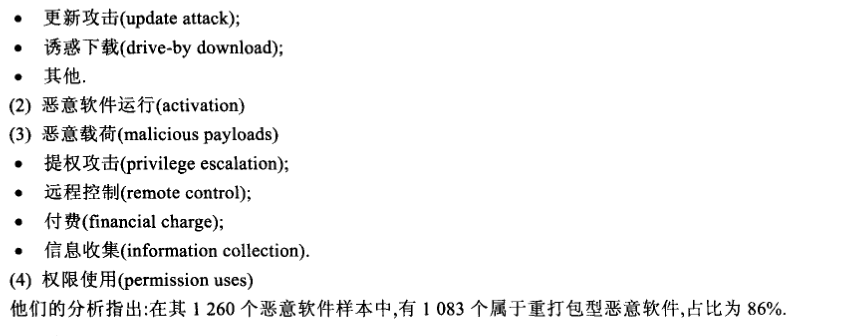
# Android恶意软件

## Zimperlieh

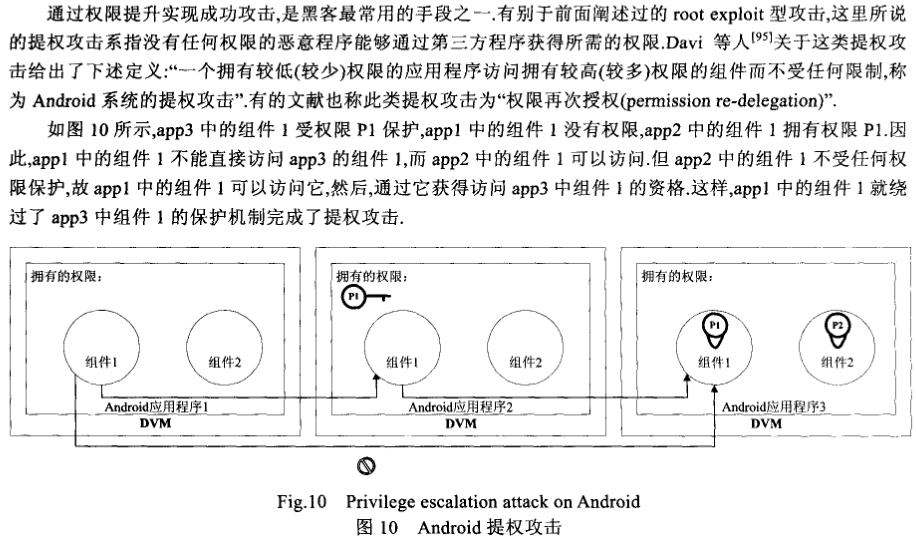


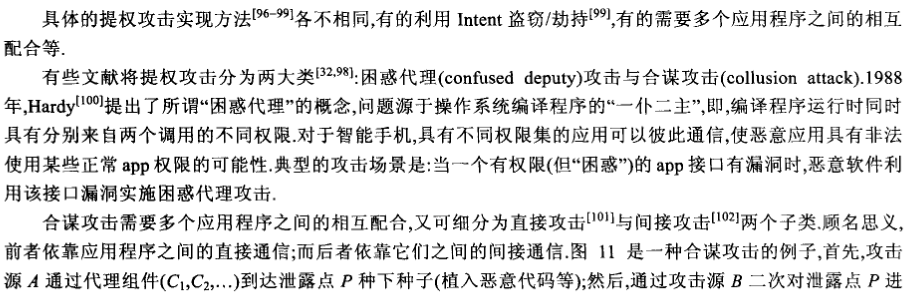
## Android恶意软件分类

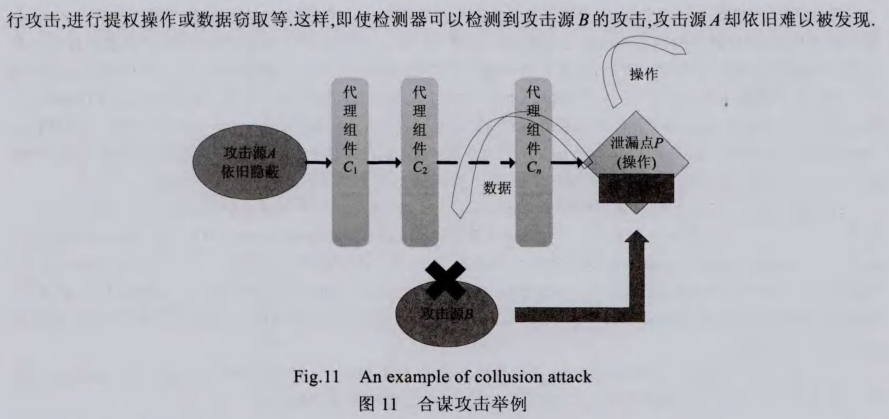




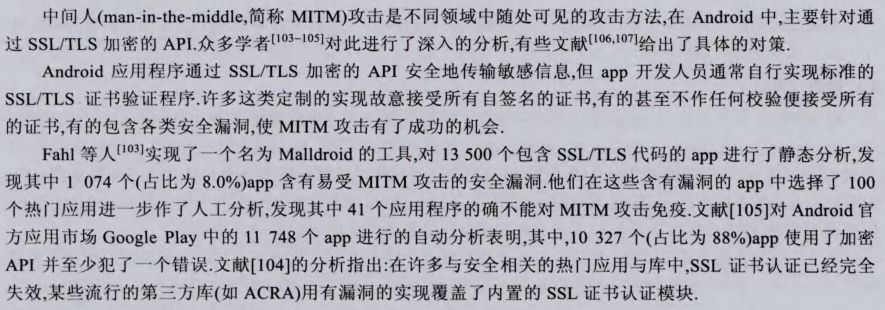
## 提权攻击



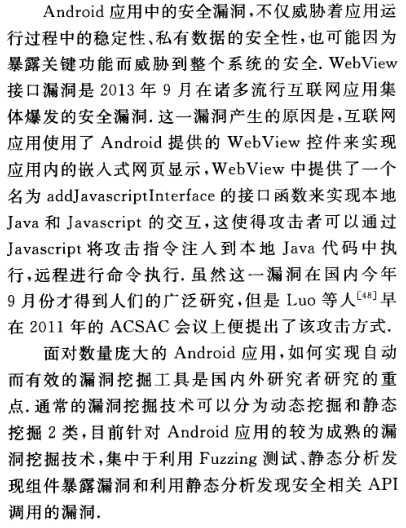




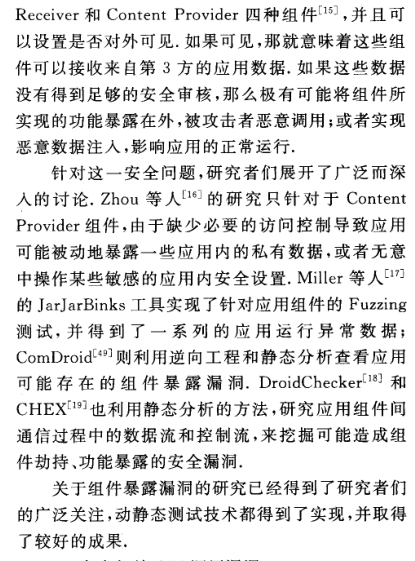
## 中间人攻击



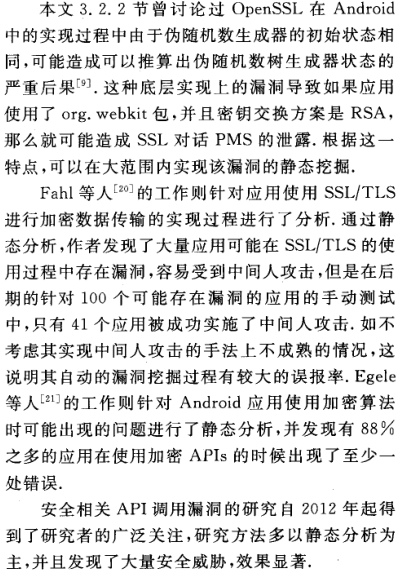
# Android应用漏洞



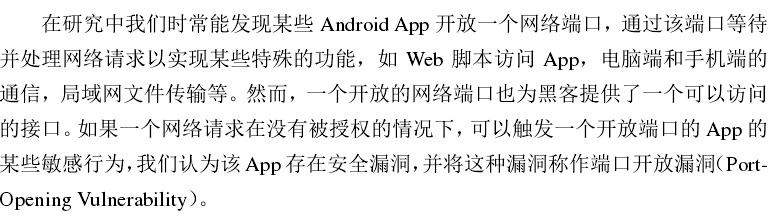
## 组件暴露漏洞

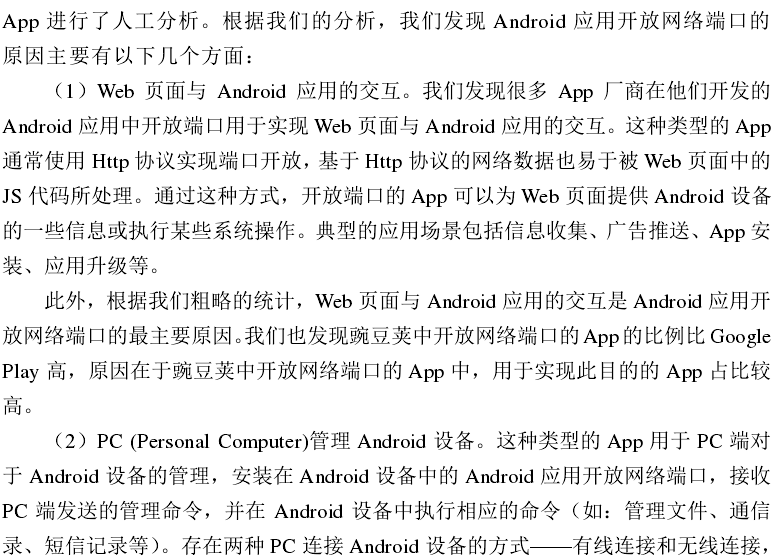


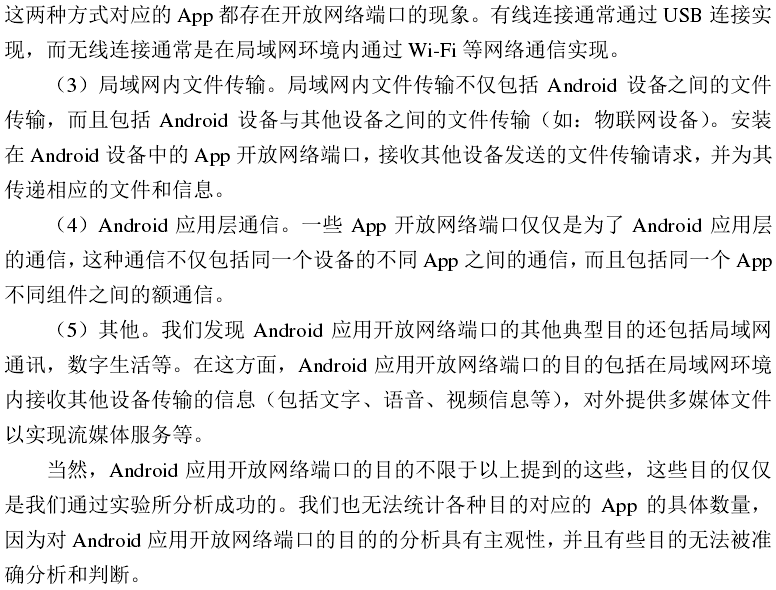
## 安全相关API调用漏洞

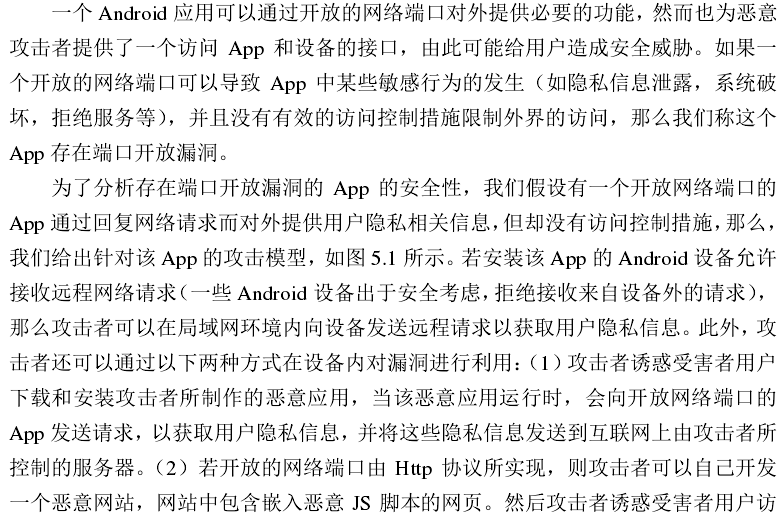


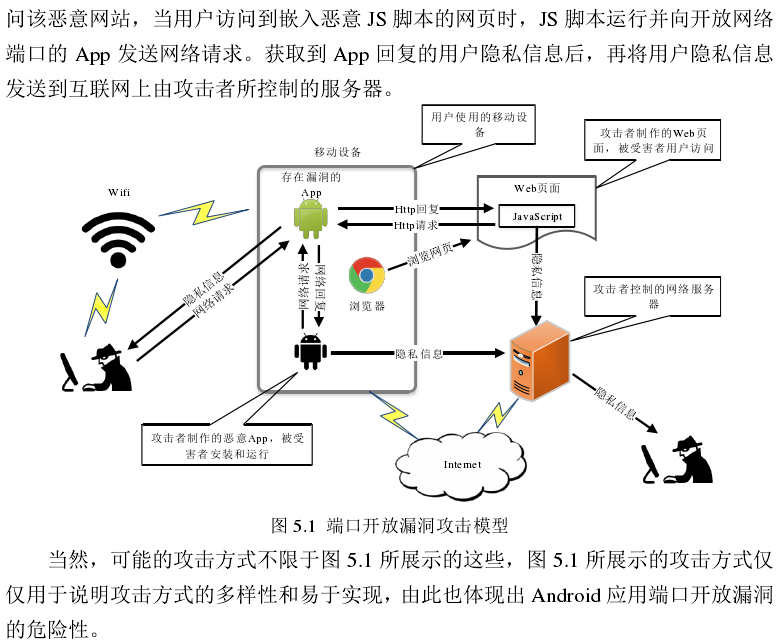
## 端口开放漏洞





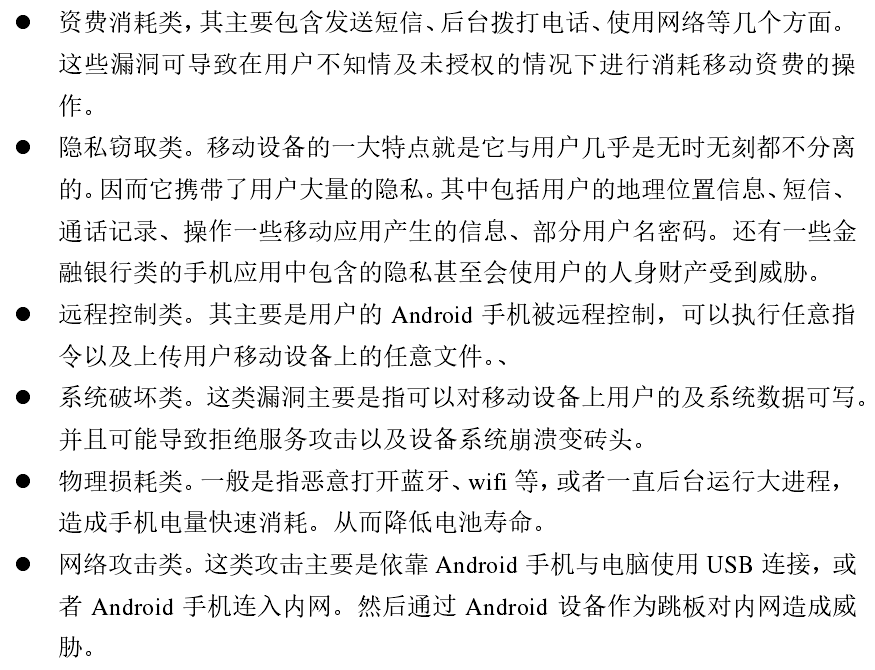




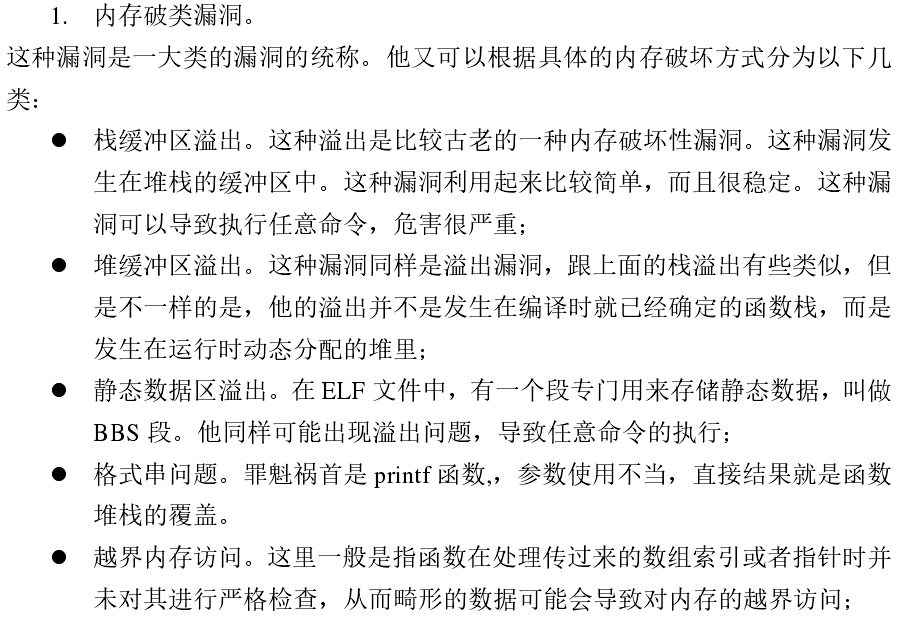


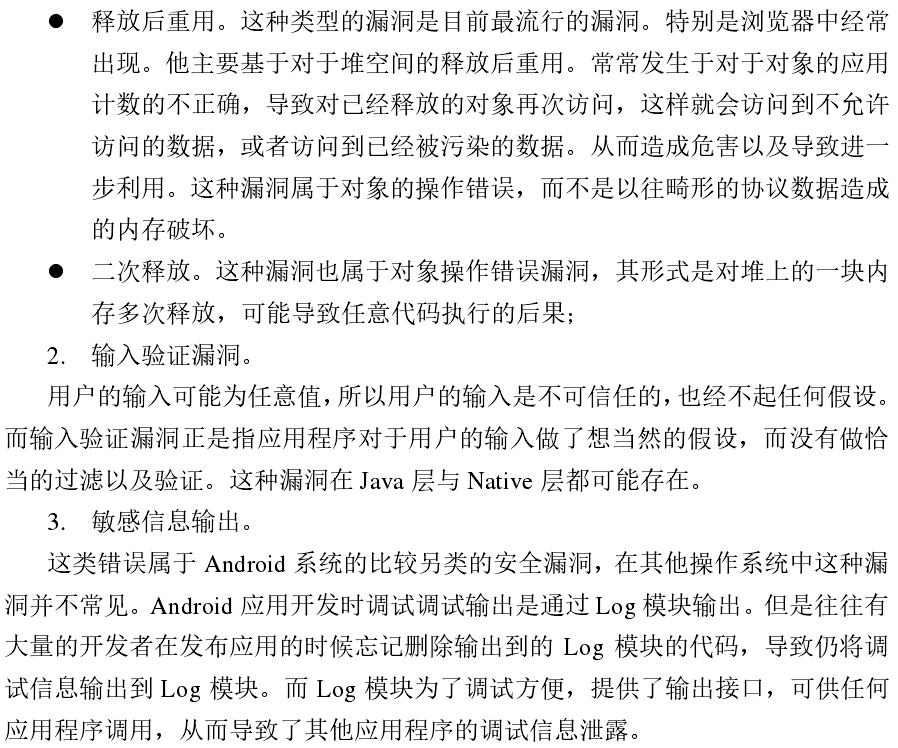
## 漏洞分类

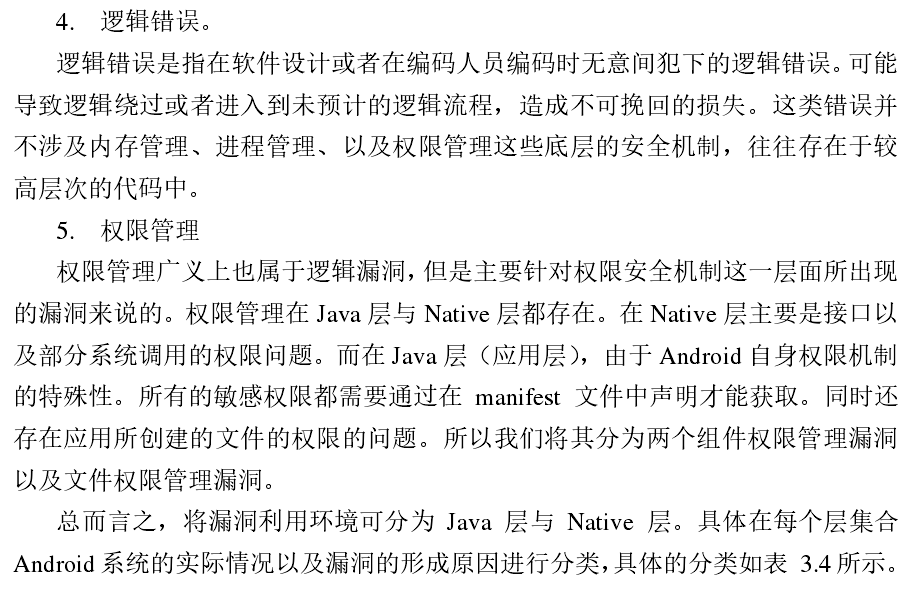




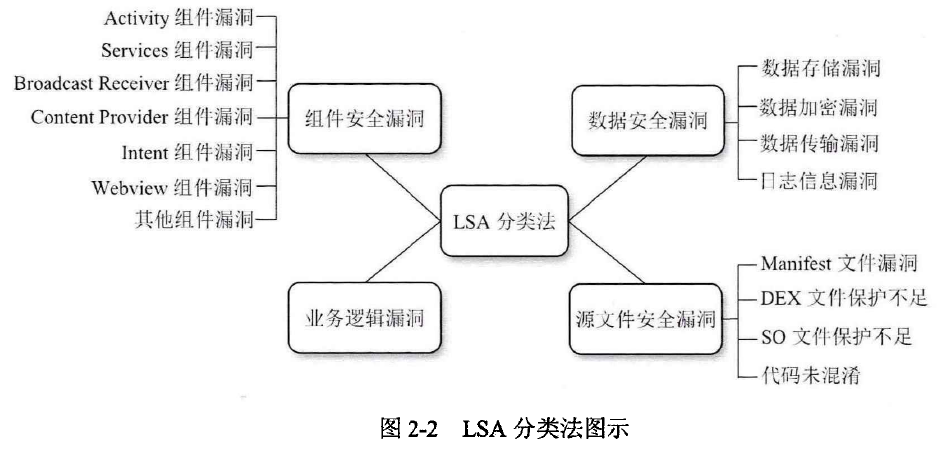












# Android漏洞挖掘技术

## 污点传播分析

污点传播分析是根据数据依赖关系对程序进行分析的一种技术，常用于检测数据相关漏洞，比如隐私泄露漏洞和组件劫持漏洞．按照是否需要目标程序运行，污点传播分析可分为静态和动态两种类型．静态方法主要利用控制流分析、数据流分析及结构分析等技术。

静态污点传播分析技术可用于检测Android应用组件劫持漏洞，通过在对外开放的接口和内部敏感sink之间进行传播分析来挖掘漏洞．静态污点传播分析技术还常用于检测隐私泄露漏洞．

动态污点传播分析技术是在定制的Android沙箱（sandbox）中动态连续跟踪数据的流

向，目前动态污点跟踪技术广泛用于漏洞挖掘等方面 ．将动态污点跟踪应用于检测隐私数据泄露时，将隐私数据作为source，然后动态监控其传播，当监测到这些污点数据通过sink被发送出去时，则可以判定该程序存在隐私泄露的问题 ．

## 可达路径分析

可达路径分析更关注控制流分析和控制流分支条件，因此在检测混淆代理漏洞（方面格外有效．混淆代理漏洞是由于存在预期外的控制流而使得预设的安全机制被绕过 而 产 生 的 漏 洞，包 括 功 能 泄 露、隐私泄露和污染、权限重委派、组件劫持等．如果能发现程序某个入口点和敏感API 调用点之间存在控制流上的可达路径，则可以认定存在混淆代理漏洞．

## 符号执行

符号执行技术是一种对程序变量进行精确计算的技术．符号执行采用抽象符号代替程序变量，并模拟程序执行．最终所有程序变量都是由符号组成的表达式构成，再通过约束求解引擎对其进行求解，获得深度语义信息．符号执行能够在复杂的数据依赖关系中发现变量之间本质的约束关系，比污点传播分析精度高，这种精确的数据流分析能帮助理解程序的内在逻辑；在模拟程序运行的过程中，符号执行还会精确记录执行路径上所有的约束条件，可以提高控制流分析中路径可达性问题判定的精确性．

## Fuzzing 测试

模糊测试，是黑盒测试技术，通过不断生成大量畸形测试数据来测试程序的鲁棒性和安全性．该技术的核心正是测试用例生成技术，良好的测试用例生成技术能保证更高的代码覆盖率和测试效率．而测试用例生成技术，又可细分为生成型和变异型两种，生成型根据预设的规则直接生成测试用例，而变异型是通过给定的种子测试用例不断生成新用例．

