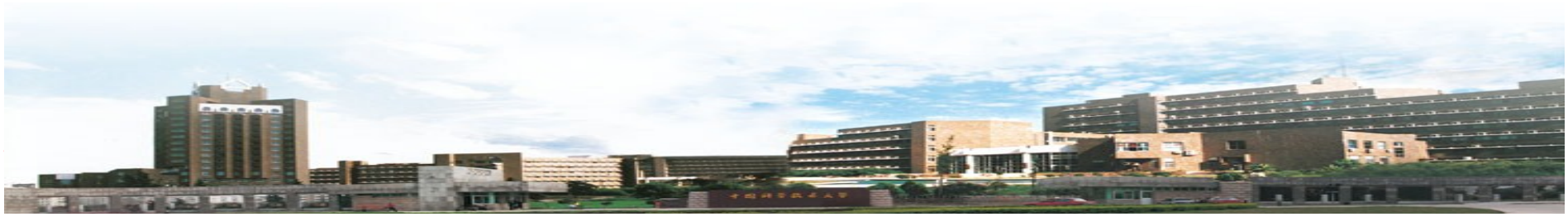
intel
Sponsors of Tomorrow.™

Android安全

程绍银

sycheng@ustc.edu.cn



引言

- 移动智能终端和移动互联网发展迅猛
- Android市场占有率已跃居第一
- Android应用安全问题日益严重

- 有必要了解Android安全问题，熟悉Android相关安全机制

本章内容

- Android概述
- Android安全问题
- Android安全机制
- Android系统机制的安全性
- 应用商店及其安全审核
- Android应用的安全检测

本章内容

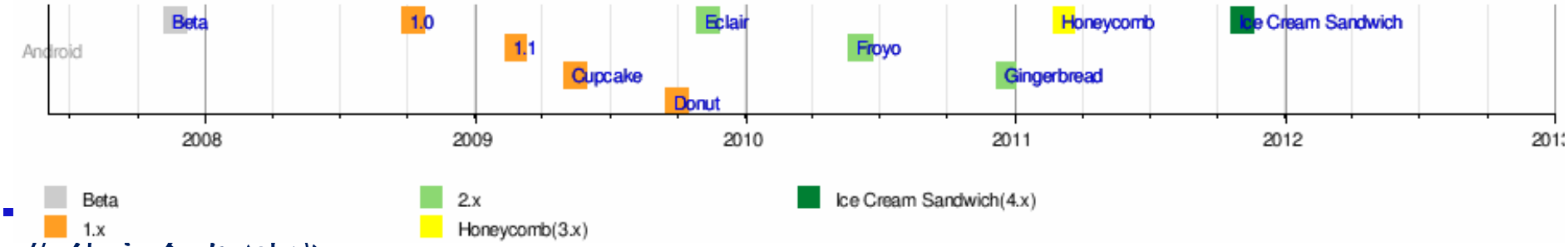
- ✓ Android概述
- Android安全问题
- Android安全机制
- Android系统机制的安全性
- 应用商店及其安全审核
- Android应用的安全检测

Android

- Android 安卓，机器人
- 基于Linux的开源手机OS
- Android公司
  - 2003.10，Andy Rubin创建
  - 2005.8，Google收购
- 2007.1.9，iphone横空出世
  - Steve Jobs：Apple reinvent the phone
- 2007.11.5，发布Android 1.0 beta版，同时宣布建立开放手持设备联盟（Open Handset Alliance）
  - 手机制造商；软件开发商；电信运营商； 芯片制造商

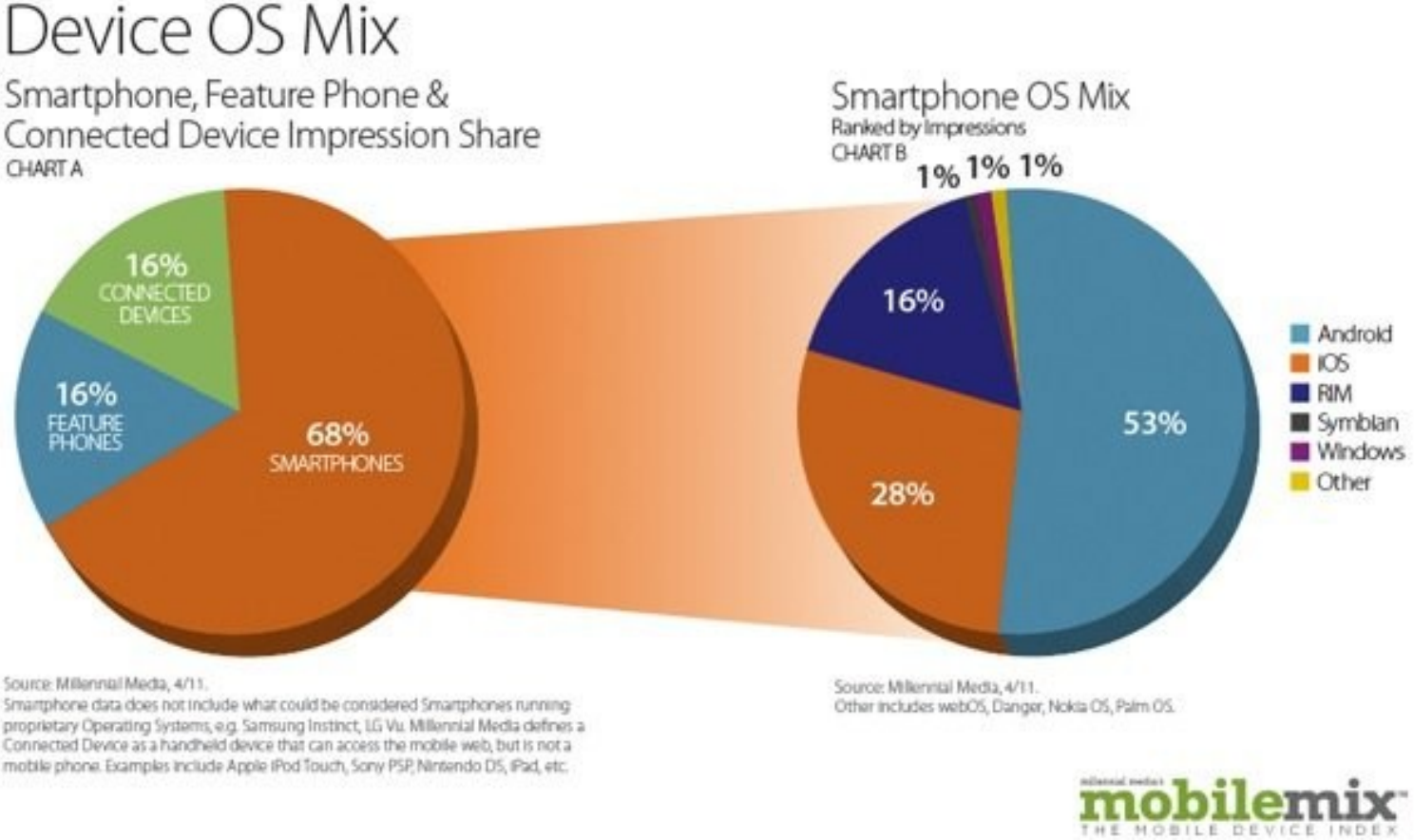
Android历史

- 2008.9, 1.0发布
  - 全球第一台Android设备HTC Dream（G1）
- 2009.10, 2.0发布
  - 强大终端产品支持，丰富应用，越来越多用户
  - 摩托罗拉 Milestone
- 2011.2, 3.0发布 平板电脑专用
- 2011.4, 4.0发布 统一手机和平板系统
  - 三星Galaxy s3



智能手机市场占有率（2011.4）

- 智能手机市场占有率68%
- 智能手机中，Android占53%，iOS占28%，



移动互联网

- 当前主要手机平台是Android操作系统和iPhone操作系统，2011年在智能手机系统中占的份额分别是39.5%和15.7%，其次是黑莓占14.9%
- IDC预测，在2015年Android手机系统将占45.9%，Windows Phone将占20.9%，iPhone将占15.3%
- Android实际发展速度，大大加快
  - 2012年上半年搭载Android系统的智能终端出货量达到一亿多部，市场占有率近70%

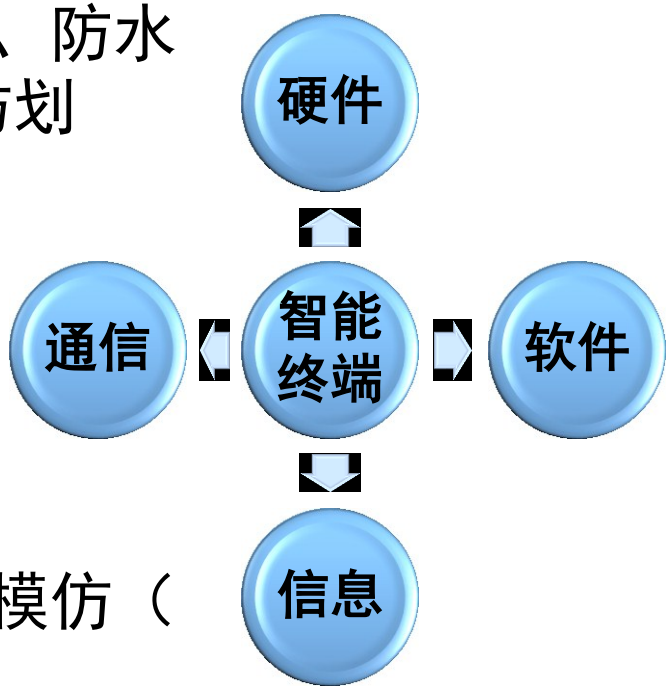
手机系统	Android	iPhone	Windows Phone 7 /Windows Mobile	BlackBerry	Symbian	others
2011年市场占有率	39.50%	15.70%	5.50%	14.90%	20.90%	3.50%
2015年市场占有率	45.90%	15.30%	20.90%	13.70%	0.20%	4.60%

主要操作系统阵营比较

	iOS	Android	Symbian	Black Berry	Windows Phone
当前最新版本	iOS4.2	Android2.3	Symbian3	Black Berry OS 6.0	Windows Phone7
开发公司	Apple	OHA联盟	Symbian 协会	RIM	Microsoft
系统家族	Unix Like	Unix Like	嵌入式	嵌入式	Windows CE
源码模式	封闭源码	开放源代码	开放源代码	封闭源码	部分开放源代码
是否支持多任务	部分支持	支持	支持	支持	支持
是否支持Flash	不支持	支持	支持	支持	不支持
内置浏览器	Safari	Mobile Chrome	Symbian浏览器	热点浏览器	Mobile IE
软件商店	App Store	Android Market	Ovi Store	App World	Market Place
游戏平台	Game Center	无	N-Gage	无	Xbox Live

智能终端安全问题

- 硬件
  - 设备丢失、设备保护（防摔，防尘、防水、防震）、Falsh防磨损、触摸屏防划
  - 具有后门特性的恶意电路
- 通信
  - 蜂窝、WiFi、蓝牙、互联网通信等
- 软件
  - 防破解、防反编译（混淆器）、防模仿（专利）、黑客攻击、服务器等
- 信息
  - 隐私、敏感数据、数据库（加密算法）等



移动智能终端带来的安全问题

- 移动智能终端
  - 与电脑相当的强大功能和业务能力
  - 记录并存储了大量用户隐私数据
  - 安全防护能力较弱及主流产品由国外企业掌控
  - 给移动互联网用户的个人隐私带来了潜在安全风险
- 移动互联网
  - 具有网络融合化、终端智能化、应用多样化、平台开放化等特点，使得信息安全问题日益凸显
  - 越来越多的黑客盯上了移动互联网用户，各种开放平台都会存在漏洞可被利用
  - 移动互联网平台的安全机制与环境还远远比不上传统互联网

移动互联网安全VS.传统互联网安全

	传统互联网安全	移动互联网安全
安全漏洞	传统计算机操作系统、应用程序都存在漏洞、网络设备也如此。	移动互联网的网络设备和智能手机面临同样困境，如IKEE.B是利用IPhone越狱后的sshd弱口令传播的蠕虫
恶意代码	大量的蠕虫、病毒、木马、僵尸网络程序泛滥	针对各种智能手机的病毒已经突破2000多种，并呈现激增趋势。
DDOS攻击	传统互联网僵尸网络发动的DDOS攻击防不胜防	移动互联网也出现相应的手机僵尸肉鸡，如BotSMS.A利用控制手机肉鸡发送大量的垃圾短信
钓鱼欺诈	钓鱼网站+网络木马轮番上阵，窃取网银、网游账号牟利	通过短信/彩信欺骗用户安装软件来实现恶意订购，如21CNread.A诱导用户下载欺诈软件从而订购21世纪手机报
垃圾信息	垃圾邮件常年泛滥，已经成为互联网的生态现象	垃圾短信方兴未艾，感染病毒的手机成为批量发送垃圾短信、彩信的新平台
信息窃取	大量木马在从事窃取个人隐私、敏感数据甚至国家秘密的工作	手机病毒不仅可以让手机成为窃听器，可将通话记录、短信内容、记事本内容全部偷走，效果更显著
恶意扣费	尚难以直接从终端电脑上扣费	手机天然带计费，传播会被扣费，上网也被扣费、恶意订购也被扣费，因此备受地下经济关注

移动智能终端的安全威胁

- 用户角度
  - 欺骗，信息泄露，信息分析，资料链接
  - 恶意软件，信息超载（拒绝服务、服务选择困境）
  - 配置的复杂性（设备和应用程序设置、安全参数）
- 网络角度
  - 被动威胁（窃听、流量分析）
  - 主动威胁（伪装、重放、消息修改、拒绝服务）
- 服务/内容提供商
  - 欺骗，拒绝服务，非支付，非法内容分发，无赖行为

本章内容

- Android概述
- ✓ Android安全问题
- Android安全机制
- Android系统机制的安全性
- 应用商店及其安全审核
- Android应用的安全检测

Android安全问题

- 中国互联网协会反网络病毒联盟2011年发布的《移动互联网恶意代码描述规范》
  - 恶意扣费
  - 隐私窃取
  - 远程控制
  - 恶意传播
  - 资费消耗
  - 系统破坏
  - 诱骗欺诈
  - 流氓行为

恶意扣费

- 恶意扣费是指在用户不知情或未授权的情况下，通过隐蔽执行、欺骗用户点击等手段，订购各类收费业务或使用移动终端支付，导致用户经济损失
- 如自动订购移动增值业务，自动利用移动终端支付功能进行消费，自动拨打收费声讯电话，自动订购其它收费业务，自动通过其它方式扣除用户资费等

隐私窃取

- 隐私窃取是在用户不知情或未授权的情况下，获取涉及用户个人信息
  - 包括短信内容、彩信内容、邮件内容、通讯录内容、通话记录、通话内容、地理位置信息、本机手机号码、本机已安装软件信息、本机运行进程信息、各类账号信息、各类密码信息、获取用户文件内容、记录分析用户行为、获取用户网络交易信息、获取用户收藏夹信息、获取用户联网信息、获取用户下载信息、利用移动终端麦克风、摄像头等设备获取音频、视频、图片信息和获取其它用户个人信息等

远程控制

- 远程控制是指在用户不知情或未授权的情况下，能够接受远程控制端指令并进行相关操作
- 包括由控制端主动发出指令进行远程控制或由受控端主动向控制端请求指令

恶意传播

- 恶意传播是指自动通过复制、感染、投递、下载等方式将自身、自身的衍生物或其它恶意代码进行扩散的行为
- 包括自动发送包含恶意代码链接的短信、彩信、邮件、WAP信息，自动发送包含恶意代码的彩信、邮件，自动利用蓝牙通讯、红外通讯、无线网络等技术向其它设备发送恶意代码，自动向存储卡等移动存储设备上复制恶意代码，自动下载恶意代码和自动感染其它文件等

资费消耗

- 资费消耗是指在用户不知情或未授权的情况下，通过自动拨打电话、发送短信、彩信、邮件、频繁连接网络等方式，导致用户资费损失
- 包括自动拨打电话，自动发送短信、彩信、邮件，频繁连接网络，产生异常数据流量等

系统破坏

- 系统破坏是指通过感染、劫持、篡改、删除、终止进程等手段导致移动终端或其它非恶意软件部分或全部功能、用户文件等无法正常使用的，干扰、破坏、阻断移动通信网络、网络服务或其它合法业务正常运行

诱骗欺诈

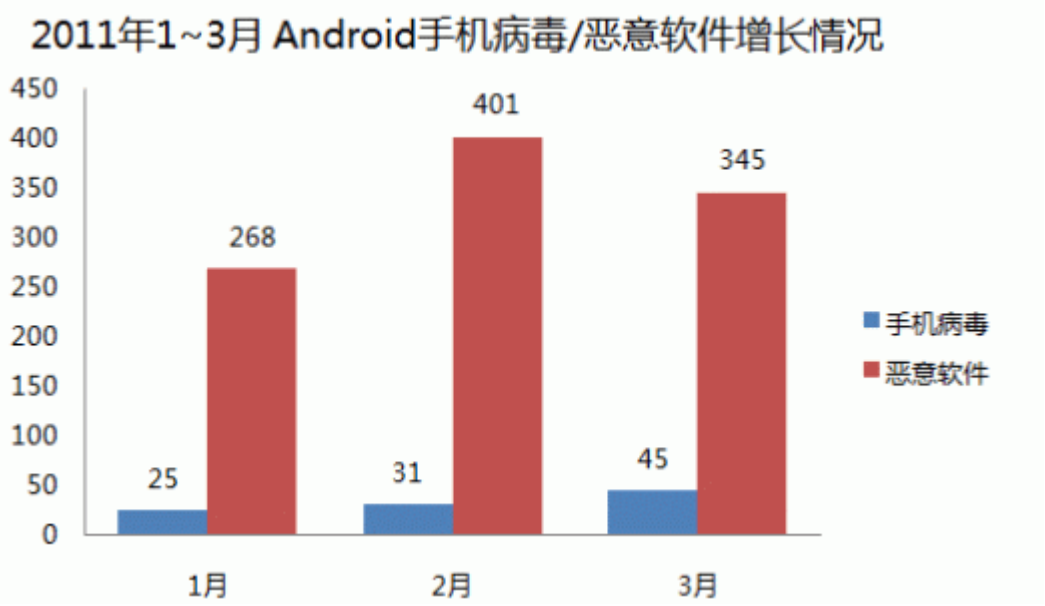
- 诱骗欺诈是指通过伪造、篡改、劫持短信、彩信、邮件、通讯录、通话记录、收藏夹、桌面等方式，诱骗用户，而达到不正当目的
  - 包括伪造、篡改、劫持短信、彩信、邮件、通讯录、收藏夹、通讯记录、用户文件、网络交易数据以诱骗用户，冒充国家机关、金融机构、移动终端厂商、运营商或其构和个人以诱骗用户，伪造事实，诱骗用户退出、关闭、卸载、禁用或限制使用其它合法产品或它机退订服务等

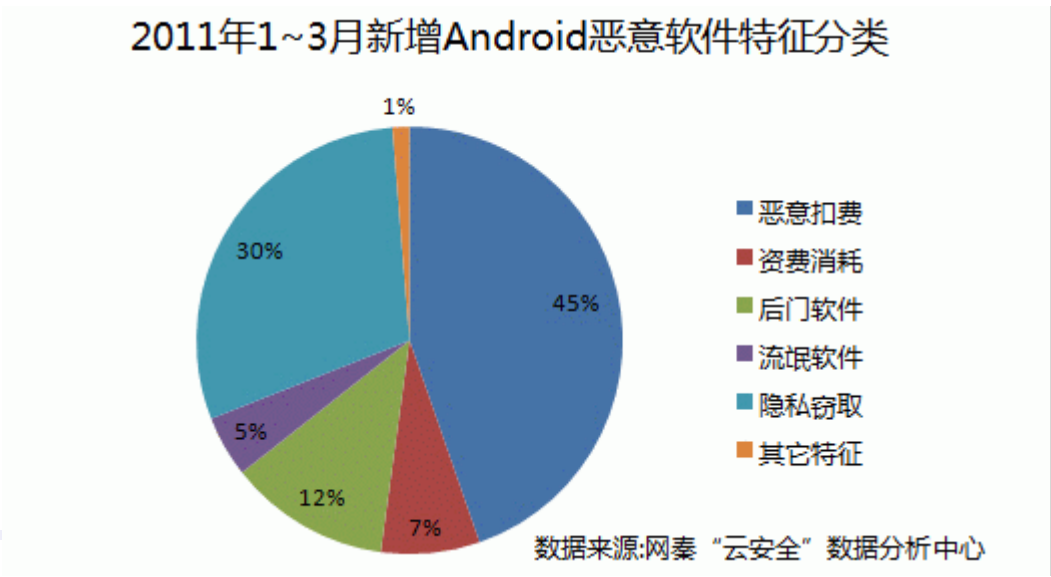
流氓行为

- 流氓行为是指执行对系统没有直接损害，也不对用户个人信息、资费造成侵害的其它恶意行为
- 包括长期驻留系统内存，长期占用移动终端中央处理器计算资源，自动捆绑安装，自动添加、修改、删除收藏夹、快捷方式，弹出广告窗口，导致用户无法正常退出，导致用户无法正常卸载、删除，在用户未授权的情况下，执行其它操作等

Android安全

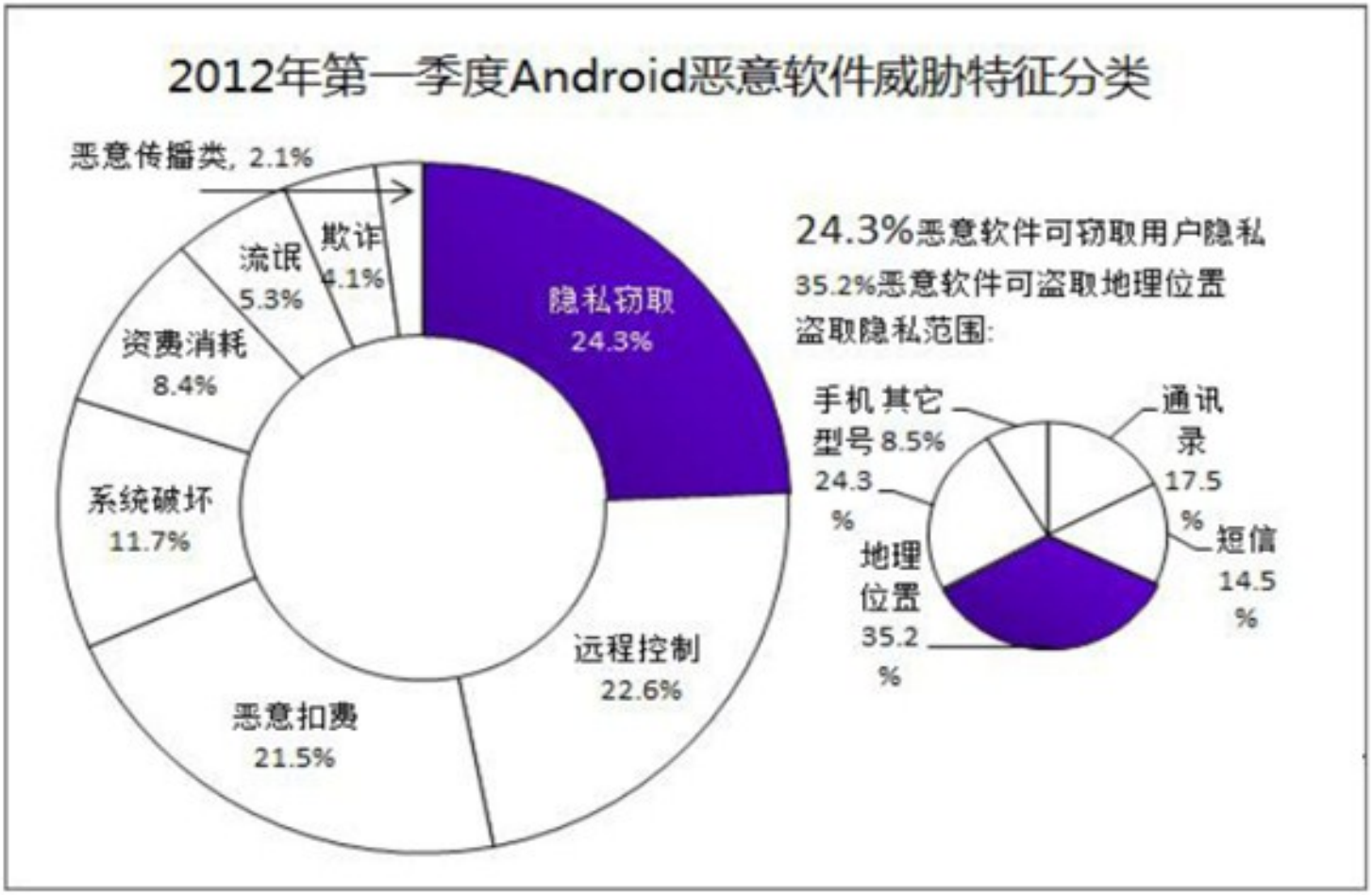
- Android安全趋势
  - 2010年底，应用超过20万，下载次数25亿
  - 恶意软件数量是手机病毒的10倍
  - 恶意扣费和隐私窃取占恶意软件的75%





Android安全

- 2012年第三季度查杀恶意软件2.3万余款，感染手机991万部，且94%的恶意软件集中在Android平台

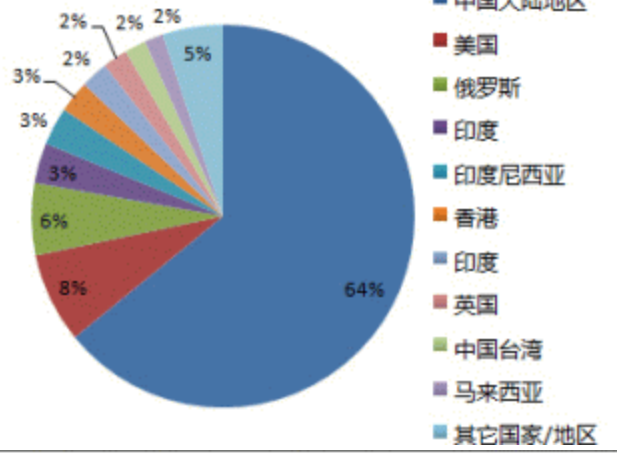


Android安全

- 感染地域分布
  - 中国大陆占64%
- 中国市场
  - 8%应用是恶意
  - 60%以上用户受到垃圾短信骚扰
  - 48%的用户出现过上网流量异常情况

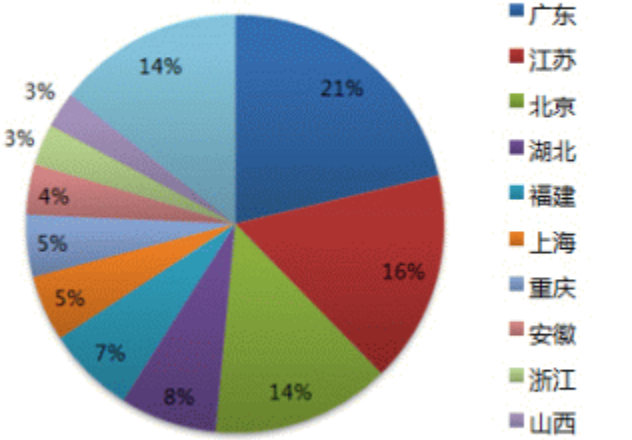
2011年1~3月 Android恶意软件感染地域分布（全球）

排名	感染国家/地区	感染比例
1	中国大陆地区	64.1%
2	美国	7.6%
3	俄罗斯	6.1%
4	印度	3.4%
5	印度尼西亚	3.2%
6	中国香港	2.7%
7	印度	2.3%
8	英国	2.1%
9	中国台湾	1.8%
10	马来西亚	1.6%
	其它国家/地区	5.1%



2011年1~3月 Android恶意软件感染地域分布（国内）

排名	感染省份/地区	感染比例
1	广东	21.3%
2	江苏	16.4%
3	北京	13.8%
4	湖北	7.5%
5	福建	6.8%
6	上海	5.1%
7	重庆	4.8%
8	安徽	3.9%
9	浙江	3.2%
10	陕西	2.8%
	其它地区	14.4%

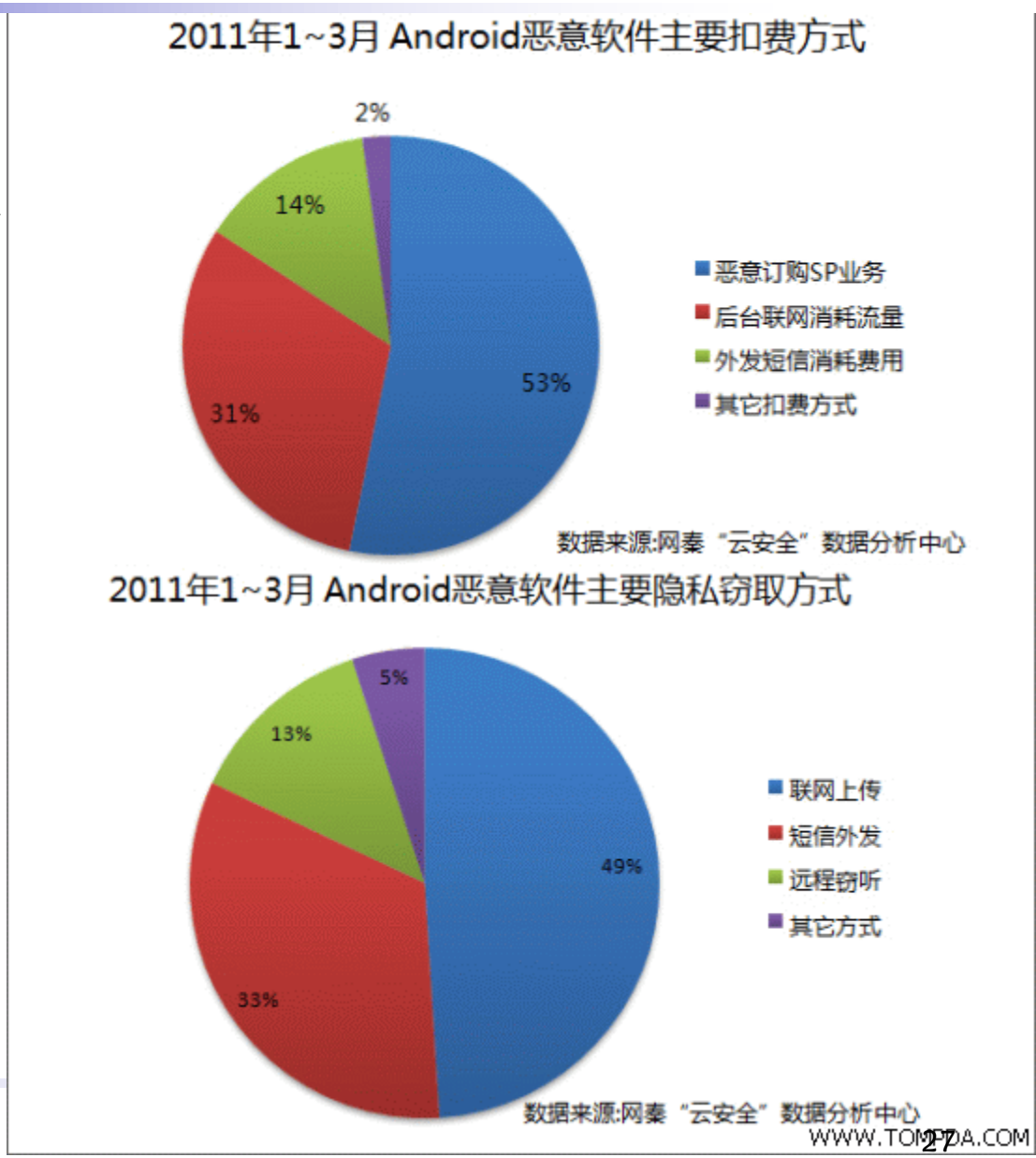


数据来源:网秦"云安全"数据分析中心

WWW.TOMPDA.COM

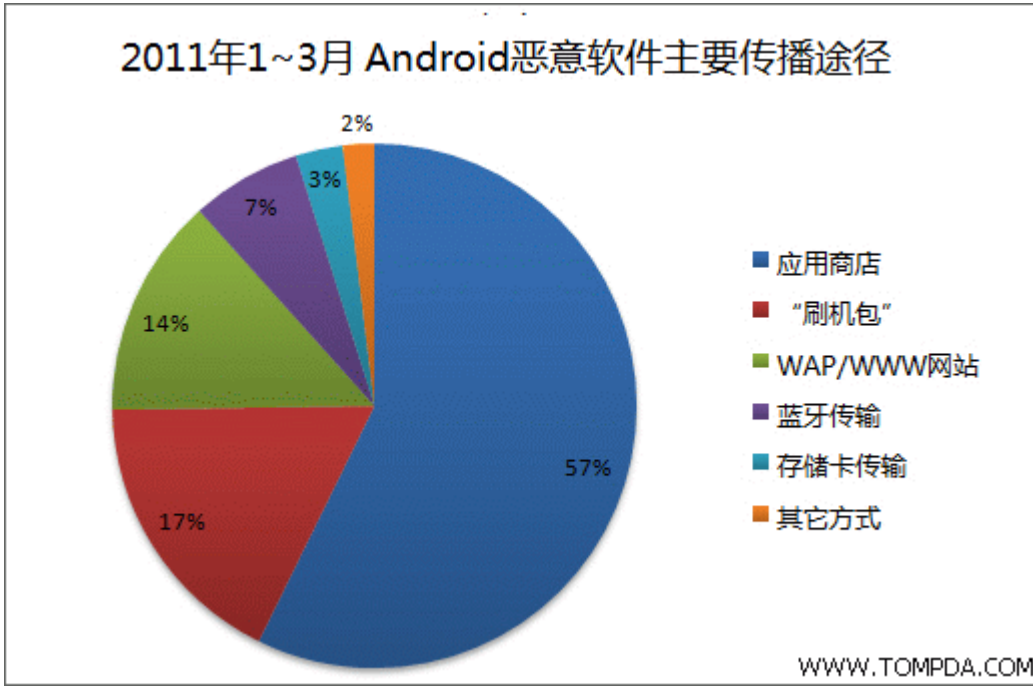
Android安全

- 主要扣费方式
  - 恶意订购SP业务
- 主要隐私窃取方式
  - 联网外传



Android应用商店成恶意软件传播源头

- 超过57%的用户通过Android应用商店下载感染
- 2011年03月04日，谷歌将Android Market应用商店中56款包含木马的应用移除
- 尽快建立应用商店的安全审核机制

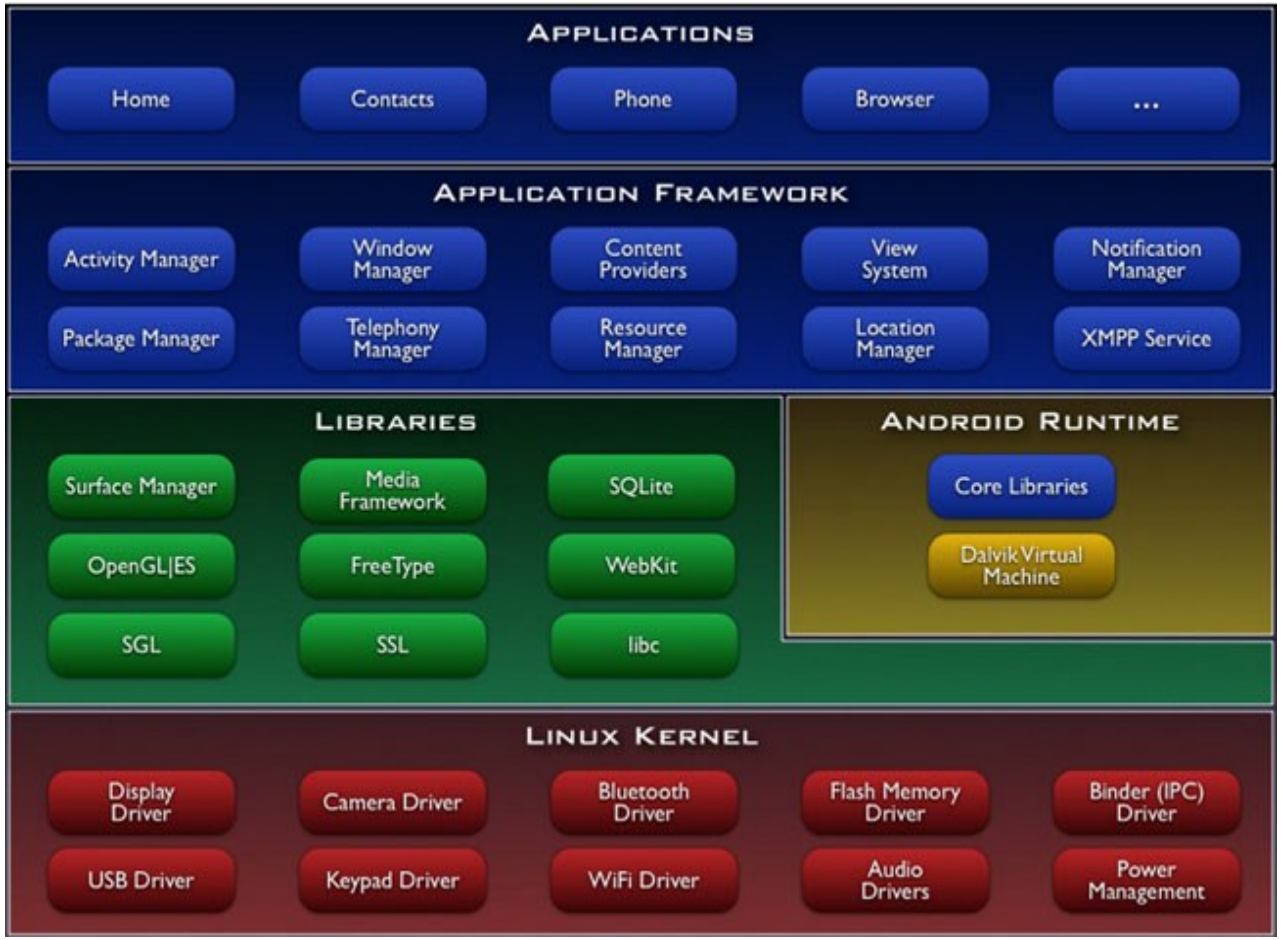


本章内容

- Android概述
- Android安全问题
- ✓ Android安全机制
- Android系统机制的安全性
- 应用商店及其安全审核
- Android应用的安全检测

Android系统架构

- 分层架构
  - 应用程序层
    - Java程序
  - 应用程序框架层
    - Dalvik虚拟机
  - 系统运行库层
    - C/C++程序
  - Linux核心层
    - 内核、驱动
- 应用开发语言
  - Java
  - C/C++



Android系统架构

Dalvik VM

- Dalvik是Google为Android系统设计的JVM，并不遵循Java官方的JVM规范
- 指令集基于寄存器架构，执行其特有的文件格式--dex字节码来完成对象生命周期管理、堆栈管理、线程管理、安全异常管理、垃圾回收等重要功能
- 核心内容是实现库（libdvm.so），大体由C语言实现
- 依赖于 Linux内核的一部分功能--线程机制、内存管理机制，能高效使用内存，并在低速CPU上表现出的高性能

Dalvik VM vs. Java VM

- Dalvik VM基于寄存器，而Java VM基于stack
  - Dalvik VM选择基于寄存器的方式是因为它对提前优化提供了更好的支持，而这对类似于移动电话这样的受限环境是很有利的
  - 在针对寄存器虚拟机和基于栈虚拟机更深入的比较分析得出的结论是，基于寄存器的虚拟机对于大的程序来说，在他们的编译的时候，花费的时间更短

Dalvik VM vs. Java VM

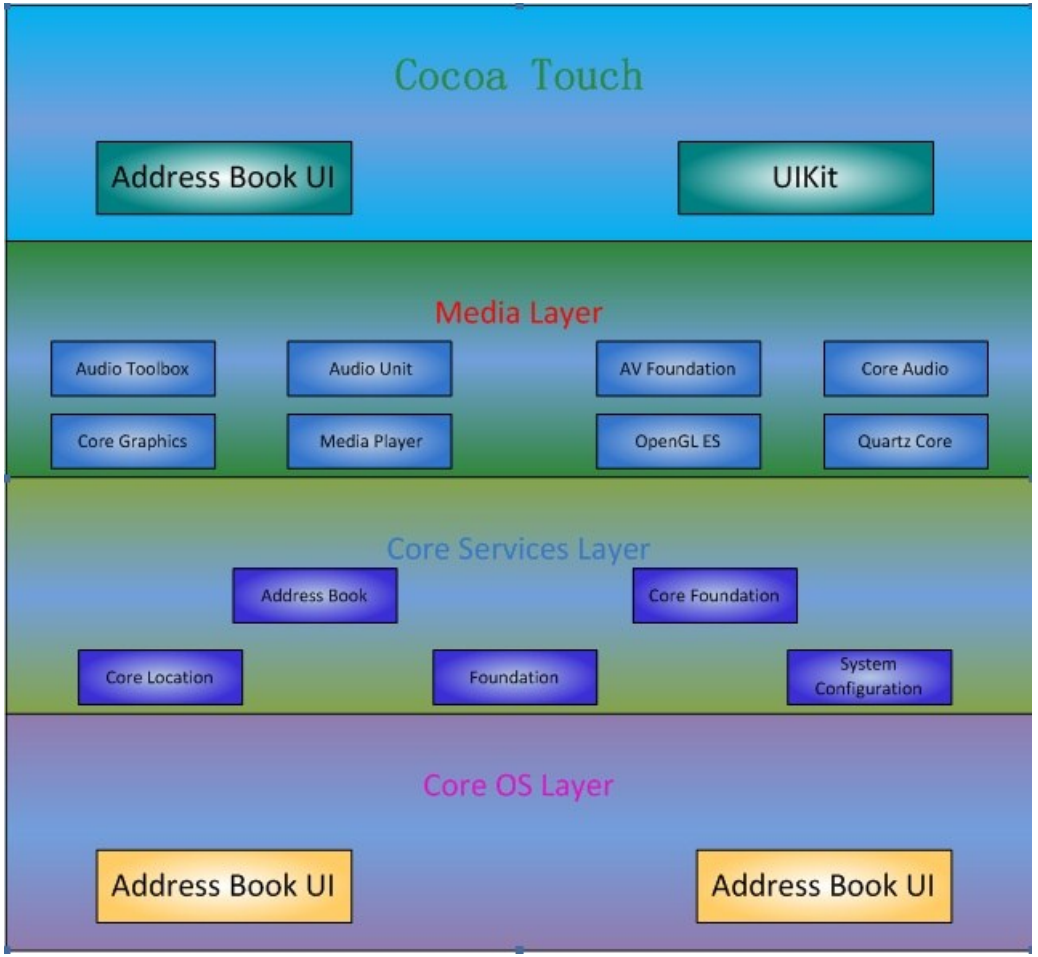
- Dalvik VM执行的是特有的DEX文件格式，而Java VM运行的是*.class文件格式
  - 一个应用中会定义很多类，编译完成后即会有很多相应的CLASS文件，CLASS文件间会有不少冗余的信息
  - dex字节码和标准Java的字节码（Class）在结构上的一个区别是dex字节码将多个文件整合成一个，这样，除了减少整体的文件尺寸，I/O操作，也提高了类的查找速度。原来每个类文件中的常量池现在由DEX文件中一个常量池来管理

Dalvik VM vs. Java VM

- 一个应用，一个虚拟机实例，一个进程
  - 每一个Android应用都运行在一个Dalvik虚拟机实例里，而每一个虚拟机实例都是一个独立的进程空间
  - 每个进程之间可以通信（IPC，Binder机制实现）
  - 虚拟机的线程机制，内存分配和管理，Mutex等都是依赖底层操作系统而实现的
  - 不同的应用在不同的进程空间里运行，当一个虚拟机关闭或意外中止时不会对其它虚拟机造成影响，可以最大程度的保护应用的安全和独立运行

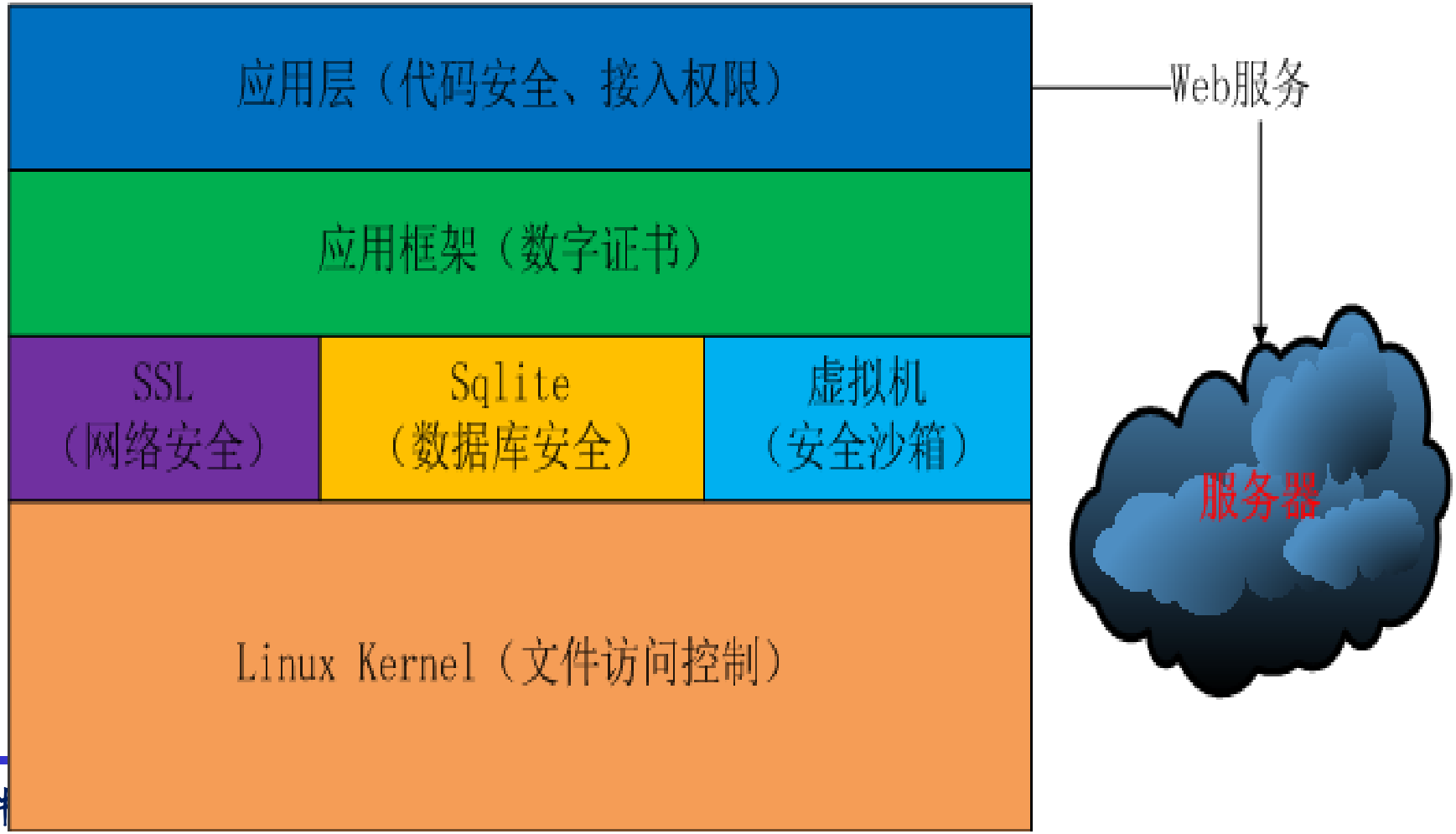
插曲：iPhone系统架构

- 分层架构
  - 触摸层
  - 媒体层
  - 核心服务层
  - 核心操作系统层
- 应用开发语言
  - Objective-C，扩充C的面向对象编程语言



Android安全机制

- 最重要的安全设计：沙箱和权限
- 最主要的安全机制：用户ID、权限、签名



Android安全机制

- Android是一个多进程系统，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
- 应用程序之间一般是不可以互相访问的，但是Android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

Android安全机制

- 安全架构
- 用户IDs和文件存取
- 权限
- 使用权限
- 自定义权限
- 组件权限
- 发送广播时支持权限
- 其它权限支持
- URI权限

安全架构

- Android安全架构的中心思想
  - 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。例如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等

安全架构

- 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

安全架构

- 所有的Android应用程序（.apk文件）必须用证书进行签名认证，而该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。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permissions，以及谁可以share 用户IDs

用户ID

- 在Android系统中，一个应用程序就是一个用户，每一个用户拥有一个独立的UID
- UID是在应用程序安装时，Android系统为之分配的，在该设备上一直保持同一个值
- Android系统的安全限制措施是在进程级实施的。在默认的情况下，应用程序不可以执行任何对其他应用程序以及系统带来负面影响的操作

用户ID与虚拟机

- 一个应用程序的进程就是一个安全的沙盒，系统根据其UID为之分配一个独立的虚拟机，其他程序不可以访问该程序中所使用的各种数据
- 但是如果其他程序拥有该应用程序的共享的UID，即设置自己的sharedUserID为该应用程序的UID，其他程序就可以突破沙盒的限制访问其数据

进程间共享数据

- 因为安全执行发生在进程级，所以一些不同包中的代码在相同进程中不能正常的运行，自从他们需要以不同Linux用户身份运行时
- 你可以使用每一个包中的AndroidManifest.xml文件中的manifest 标签属性sharedUserId 拥有它们分配的相同用户ID
- 通过这样做，两个包被视为相同的应用程序的安全问题被解决了，注意为了保持安全，仅有相同签名（和请求相同sharedUserId标签）的两个应用程序签名将会给相同的用户ID

权限

- Android系统的权限用来描述应用程序是否有做某件事的权利
- Android系统中权限划分非常细，如Android 4.0.x系统，权限总数就有124个
- 一个应用程序的权限是在其安装的时候确定的，而不是在运行的时候
- 如果一个应用程序在安装时没有申请完其运行时所有的权限，那么就必须重新安装

权限

- 一个权限主要包含三方面的信息
  - 名称
  - 属于的权限组
  - 保护级别
- 权限组是指把权限按照功能分成不同的集合
- 每个权限组包含若干具体权限
  - 例如PHONE_CALLS组中包含android.permission.READ_PHONE_STATE, android.permission. PROCESS_OUTGOING_CALLS等与接入和修改电话状态有关的权限

权限的保护级别

- 权限通过protectionLevel来标识保护级别
  - 普通级别(Normal)，不会给用户带来实质性的伤害，如调整背光
  - 危险级别(Dangerous)，可能会给用户带来潜在性的伤害，如读取电话簿、联网等，系统在安装应用时提示用户
  - 签名级别(Signature)，具有同一签名的应用才能访问
  - 系统/签名级别(Signature/System)，主要被设备商使用，不推荐使用

权限的申请

- 在默认情况下应用程序没有权限执行对其它应用程序、操作系统或用户有害的操作
- 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
- 普通级别和危险级别属于比较低的级别，申请即可授予
- 签名级别和系统/签名级别需要使用者的应用程序和系统使用同一个数字证书才可以申请成功

权限的声明

- 在应用程序的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：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>
```

签名

- 每一个Android应用程序都有必须有一个数字签名，目的是为了在应用程序的开发者和应用程序之间建立一种信任关系
- 例如，一个permission的protection Level为signature，那么只有那些有该数字签名的应用程序才可以获得该权限

签名

- Android数字签名可以是自签名的，不需要一个权威的数字证书机构进行签名认证
- 每一个签名都有有效期，系统只有在安装的时候才会检查应用程序的签名是否有效，安装后就不会对签名进 行检查，所以安装后的应用程序即使数字签名失效也可以正常使用
- 数字签名的另一个用途就是应用程序的升级，不过如果数字签名失效后，该应用程序就无法正常升级

使用权限

- 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：
  - 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
  - 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

自定义权限

- 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

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

组件权限

- 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
- 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

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

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

URI权限

- URI：Uniform Resource Identifier
- 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

URI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

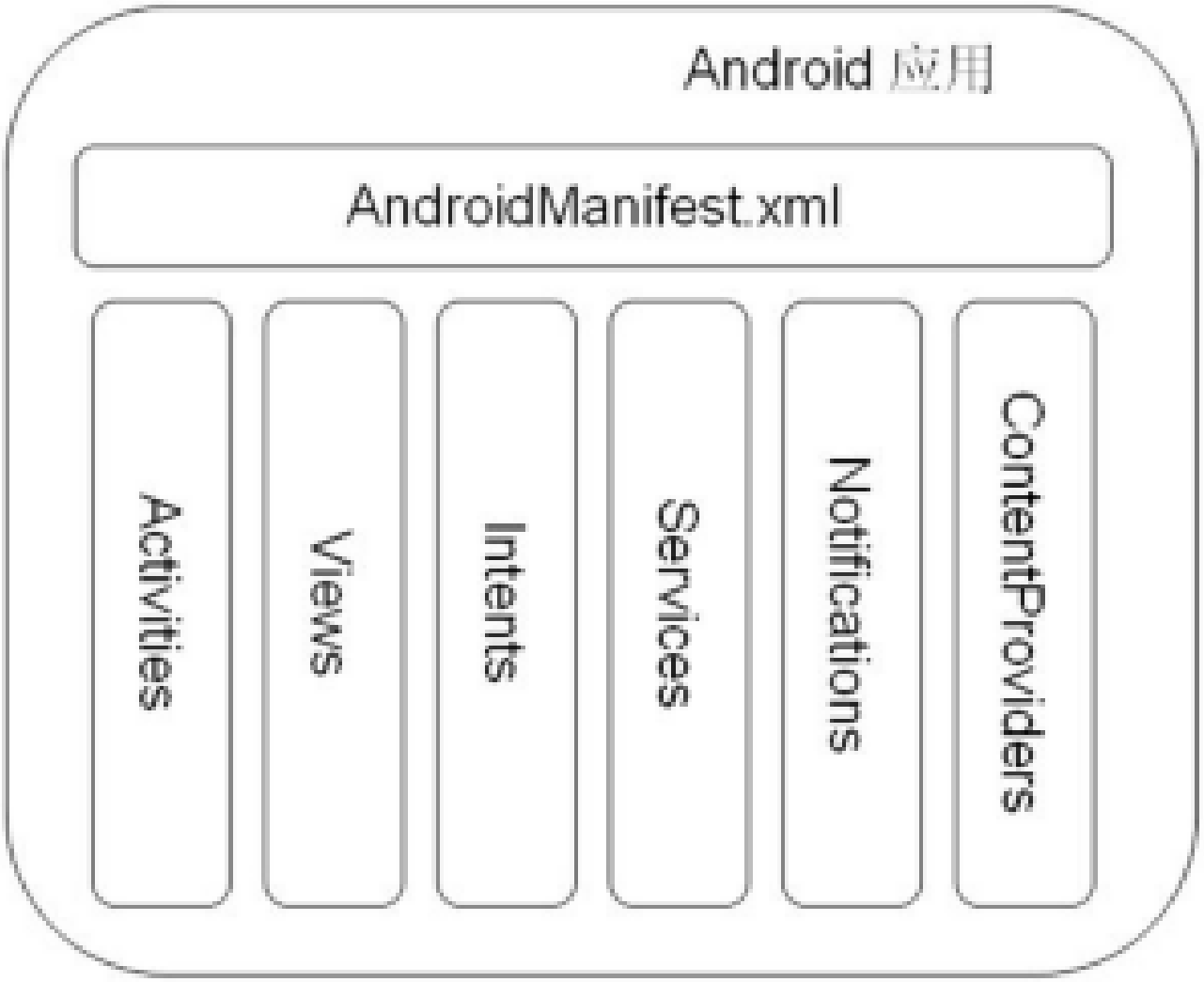
Android与经典访问控制

- SOA
  - 主体
  - 客体
  - 访问操作
- 访问控制中间结构

AndroidManifest.xml

- android程序的全局配置文件，每个程序必须有，位于根目录
- 描述了package中暴露的组件（activities, services, 等等），以及他们各自的实现类，各种能被处理的数据和启动位置
- 除了能声明程序中的Activities, ContentProviders, Services, 和Intent Receivers,还能指定permissions和instrumentation（安全控制和测试）

AndroidManifest.xml结构



简单AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"package="com.my_domain.app.helloactivity">
    <application android:label="@string/app_name">
        <activity class=".HelloActivity">
            <intent-filter>
                <action android:value="android.intent.action.MAIN"/>
                <category android:value="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

AndroidManifest.xml文件结构

manifest 根节点，描述了package中所有的内容。在它之下能放置：

uses-sdk 编译SDK的要求

uses-permission 请求你的package正常运作所需赋予的安全许可。一个manifest能包含零个或更多此元素。

instrumentation 声明了用来测试此package或其他package指令组件的代码。一个manifest能包含零个或更多此元素。

application 包含package中application级别组件声明的根节点。此元素也可包含application中全局和默认的属性，如标签，icon， 主题，必要的权限，等等。一个manifest能包含零个或一个此元素（不允许多余一个）。在它之下能放置零个或更多下列组件声明：

activity 是用来与用户交互的主要工具。当用户打开一个应用程序的初始页面就是一个activity，大部分被使用到的其他页面也由不同的activity所实现并声明在另外的activity标记中。注意：每一个activity必须要一个<activity>标记对应，无论它给外部使用或是只用于自己的package中。如果一个activity没有对应的标记，你将不能运行它。另外，为了支持运行时查找你的activity，你能包含一个或多个<intent-filter>元素来描述你activity所支持的操作：

intent-filter 声明了指定的一组组件支持的Intent值，从而形成了IntentFilter。除了能在此元素下指定不同类型的值，属性也能放在这里来描述一个操作所需的唯一的标签，icon和其它信息。action 组件支持的Intent action。category 组件支持的Intent Category.

receiver IntentReceiver能使得application获得数据的改变或者发生的操作，即使它当前不在运行。利用activity标记，你能选择地包含一个或多个receiver所支持的<intent-filter>元素；

service Service是能在后台运行任意时间的组件；

provider ContentProvider是用来管理持久化数据并发布给其他应用程序使用的组件。

本章内容

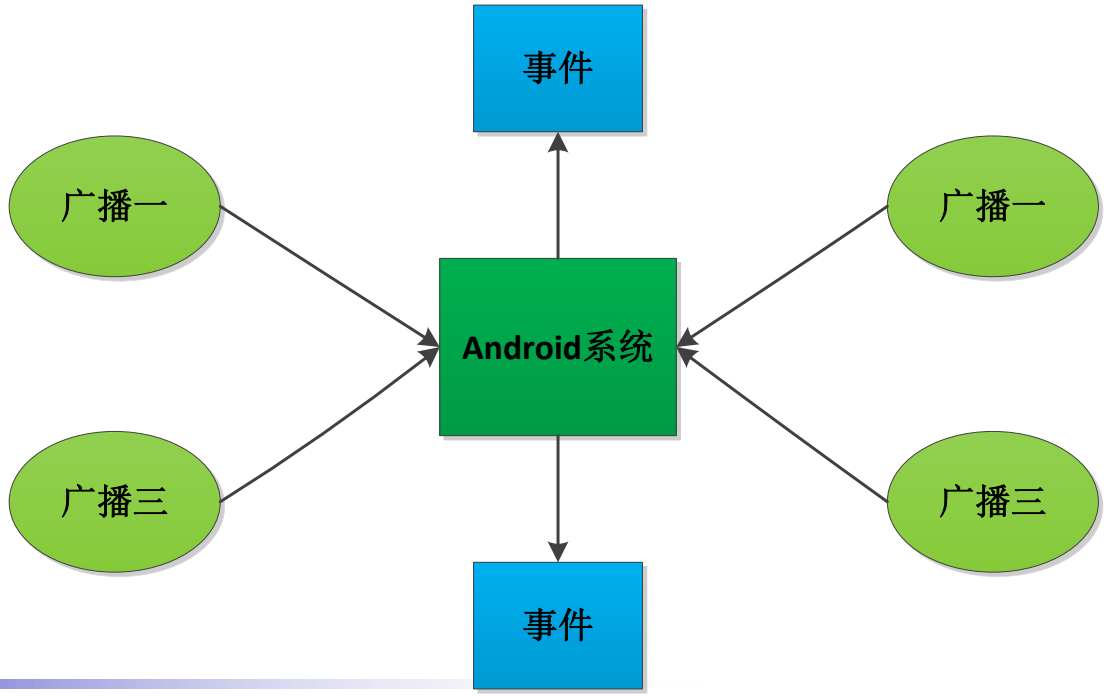
- Android概述
- Android安全问题
- Android安全机制
- ✓ Android系统机制的安全性
- 应用商店及其安全审核
- Android应用的安全检测

Android系统机制的安全性

- Android系统为了简化应用程序的开发以及各种软硬件资源和权限的管理，设计了很多的机制，例如广播与监听机制、服务机制等等
- 从开发和管理的角度上说，这些机制的确使开发和管理变的简单
- 从用户安全的角度上说，这些机制却将使用应用程序的用户暴露于极大的安全隐患之中

广播机制

- 在 Android系统中存在各种各样的广播，比如电池的使用状态，电话的接收和短信的接收都会产生一个广播，应用程序开发者也可以监听这些广播并做出程序逻辑的处理



广播机制

- 应用程序或系统在运行时便会在Android系统注册各种广播
- Android系统接收到广播会判断哪种广播需要哪种事件
- 不同的广播处理事件可能相同，相同的广播也可能处理事件不同，这时就需要Android 系统为我们做筛选

广播机制

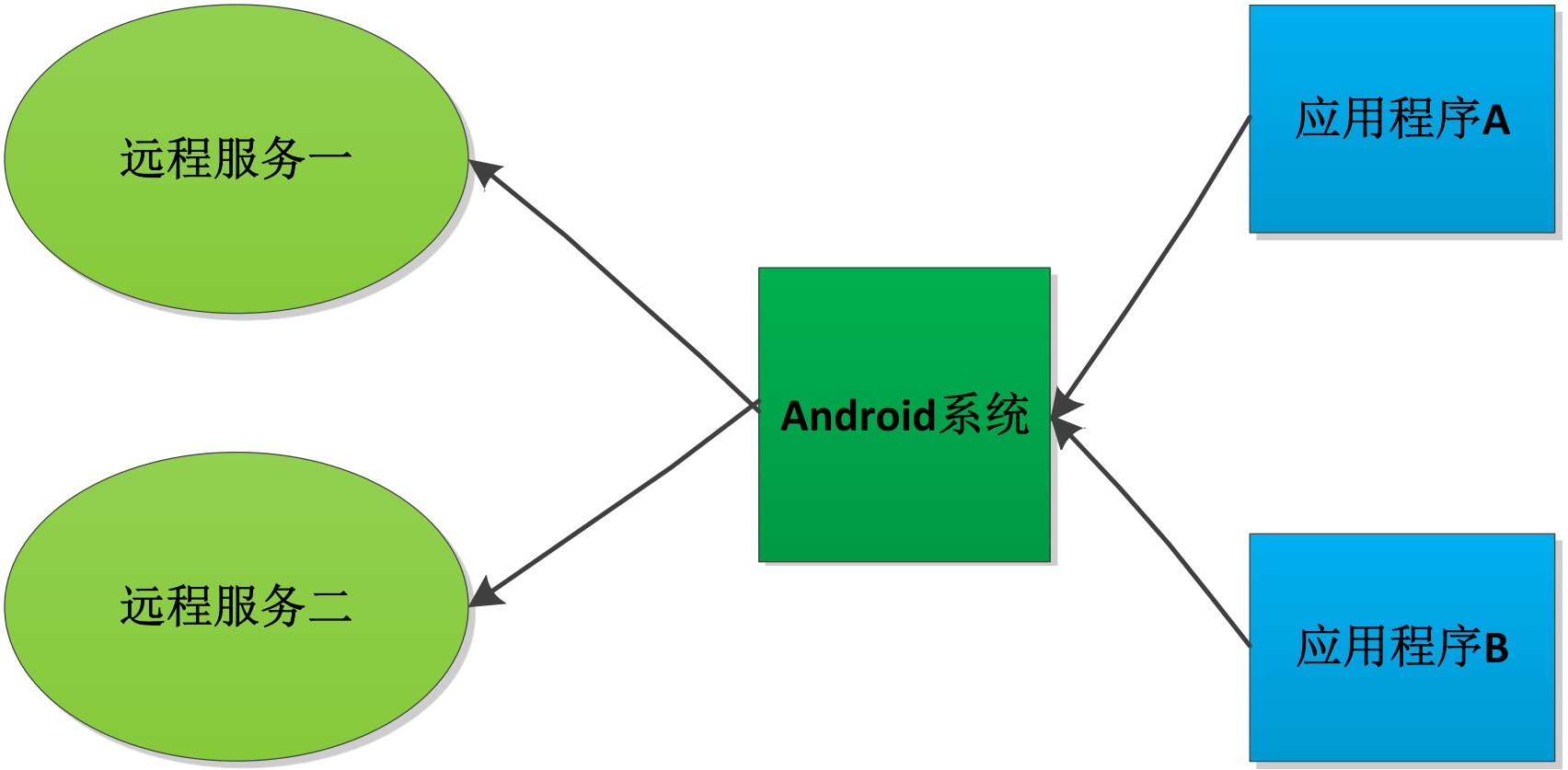
- **存在的危险性：**
- 恶意应用开发者可以在开发的应用程序中注册比较常用的广播监听器，比如电话和短信的广播
- 当监听到Android系统发送这类广播时，就可以在广播处理函数中做一些有害于应用使用者的行为

服务机制

- Service是Android系统的组件之一，它是不可见的，并且是在后台运行的
- Service一般是处理比较耗时以及长时间运行的操作
- Service分为两种：本地服务和远程服务
  - 本地服务是用于应用程序内部的一些耗时的任务，比如查询升级信息，并不占用应用程序比如Activity所属线程，而是单开线程后台执行
  - 远程服务是用于Android系统内部的应用程序之间的，可以被其他应用程序复用，比如天气预报服务，其他应用程序不需要再写这样的服务，调用已有的即可

服务机制

- 相比于本地服务，远程服务是很危险



服务机制

- **存在的危险性：**
- 服务可以提供很多方式供恶意应用程序开发者开发恶意应用，举个很简单例子，一个恶意应用程序在开发时不需要任何与外界交互的权限，比如发送短信和访问网络的权限，只要收集到感兴趣的用户信息，再通过一个具有以上权限的服务，将这些数据发送出去
- 这也是Android系统的IPC(Inter-Process Communication)机制

本地库加载机制

- Google为C/C++开发人员提供了NDK，同样可以开发Android应用程序
- NDK编写的程序就是本地库文件，在使用这些库前，必须要加载这些库
- Android系统提供了相应的函数，即load和loadLibrary，这两个函数可以将本地库加载到当前应用，同时Android系统还提供函数exec，能执行命令行命令

本地库加载机制

- **存在的危险性**：执行命令行的命令和执行本地库都存在着很大的危险性，因为本地库有可能是从网上下载的，这样会导致执行危险的代码。

应用程序升级机制

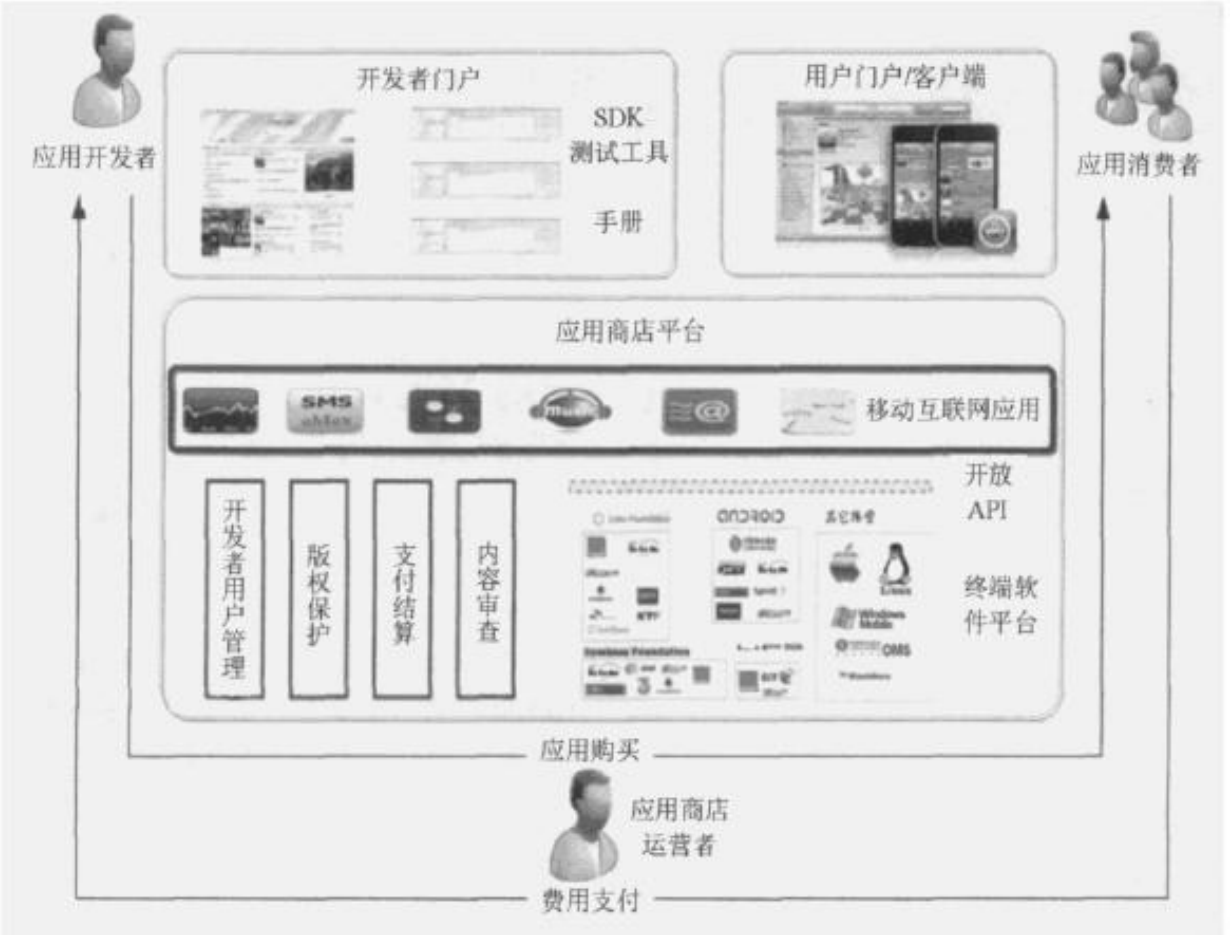
- Android系统提供应用程序的升级，而不是重新安装需要升级的程序
- 一个应用程序与需要升级到的版本使用同一个开发者签名
- 程序在升级过程中不需要重新向应用系统申请权限，可以直接使用之前版本已经申请的权限
- **存在的危险性**：升级前的应用程序是安全的，并申请一些与外界交互的权限，比如访问网络，而升级后的应用程序再使用这些权限，完成一个恶意应用软件的功能

本章内容

- Android概述
- Android安全问题
- Android安全机制
- Android系统机制的安全性
- ✓ 应用商店及其安全审核
- Android应用的安全检测

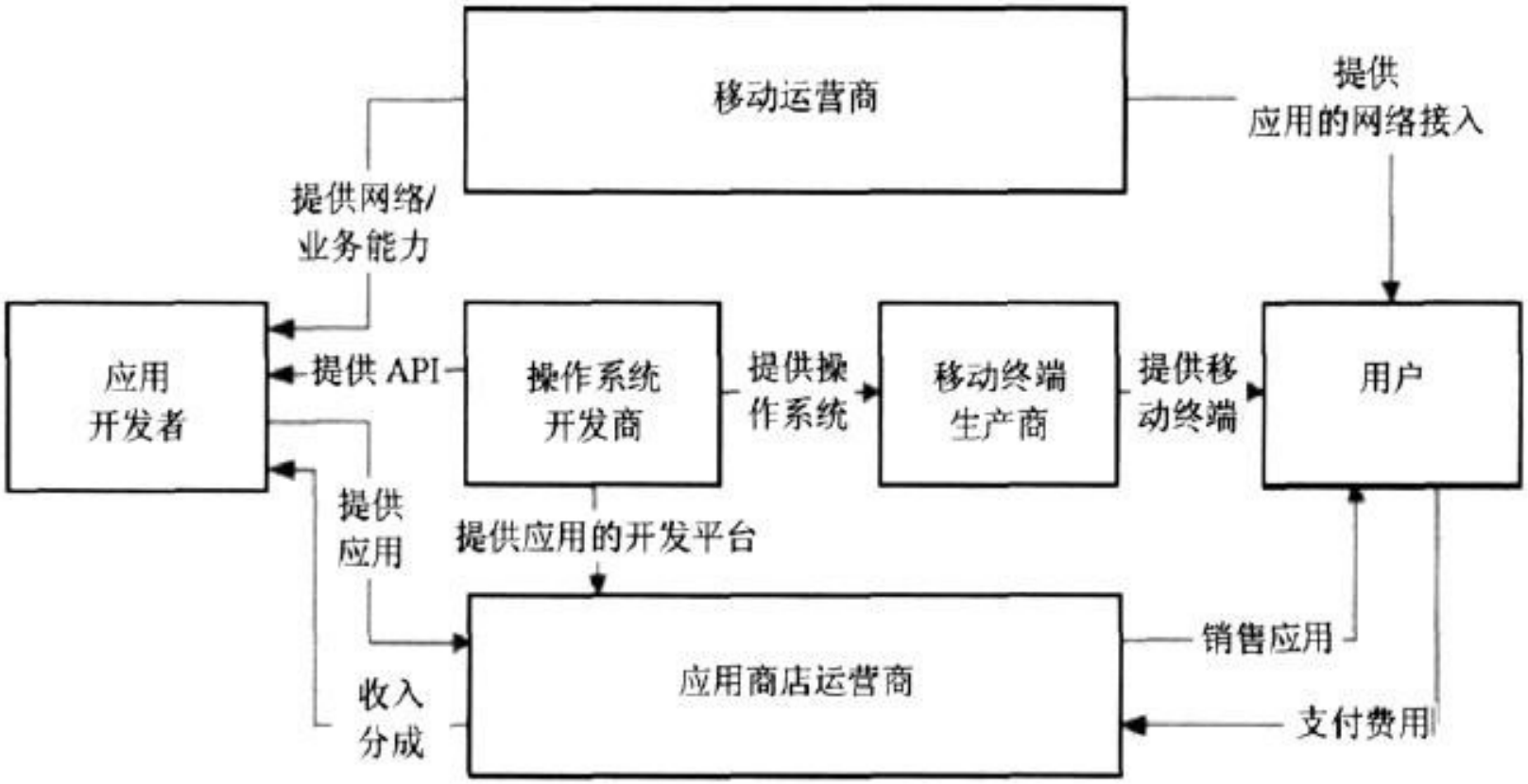
手机应用商店体系结构

- 3个部分
  - 应用商店平台
  - 开发者门户
  - 用户门户/客户端
- 3个参与主体
  - 应用商店运营者
  - 应用开发者
  - 应用消费者
- 2条业务流
  - 应用购买业务流
  - 费用支付资金流



手机应用商店的产业链

- 应用商店运营商可由移动运营商、操作系统开发商、移动终端生产商充当



安全保障涉及的产业链

- 操作系统开发商
  - 采用数字签名认证机制，禁止没获得签名的应用在操作系统上加载和安装
  - 提供安全认证沙盒，供开发者对应用在操作系统的安全性和可用性进行测试
- 应用商店运营商
  - 对申请上架的应用进行审查，确认应用不包含病毒、木马、恶意代码等后方可上架销售
- 移动运营商
  - 对应用调用的网络/业务能力（短信、流媒体等）进行鉴权认证
  - 对用户信息的使用进行分等级开放和鉴权
  - 对已使用应用进行拨测，保证应用不包含恶意代码、不良信息等
- 终端制造商
  - 通过预装杀毒软件、防火墙等防护软件，对下载的应用进行防护

手机应用商店的分类

- 以终端制造商为主导
  - 苹果App store、黑莓BB App world
  - 诺基亚Ovi Store、LG Application Store等
- 以操作系统开发商为主导
  - Google Android Market
  - 微软Windows Marketplace等
- 以移动运营商为主导
  - 中国移动 Mobile Market
  - 中国联通 “沃”商店
  - 中国电信 “天翼空间”

苹果App Store

- 2008年7月11日上线。手机软件业发展史上的一个重要的里程碑
- 截至2011年6月，应用超过25万个，下载量突破140亿次
- 在应用开发上处于主导地位
  - iOS开发者计划（iOS Developer Program）
- 应用审核
  - App Store 审查方针（Review Guidelines）
  - 应用审查不透明，取舍方式有时也非常武断
  - 不管新软件还是更新，审查期可能是数周到好几个月
- 白名单制度（whitelist scheme）

苹果App Store应用安全审核

- 1. 功能方面
  - 1）以任何方式或形式下载代码的程序将会被拒绝
  - 2）安装或释放其他可执行代码的程序将会被拒绝
- 2. 位置信息方面
  - 1）在采集、传送或使用位置数据之前未通知并获得用户同意的程序将会被拒绝
- 3. 推送通知方面
  - 1）使用推送通知发送敏感个人信息或机密信息的程序将会被拒绝
  - 2）如果程序能够传送病毒、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，该程序将会被拒绝

苹果App Store应用安全审核

- 4.隐私方面
  - 1）应用程序不能在未获用户允许或未向用户提供如何使用及在何处使用数据的相关信息情况下传输有关用户的数据。
  - 2）要求用户共享电子邮箱地址和出生日期等私人信息才可使用其功能的应用程序将会被拒绝。
  - 3）锁定未成年人进行数据收集的应用程序将会被拒绝。
- 5. 法律要件
  - 开发暗中收集用户密码或用户私人数据程序的开发者将会从iOS开发者项目中除名。

Google Play Store

- 发布应用流程
  - 首先需要注册，然后支付一定的费用后才能发布应用
  - 如果要发布付费应用，需要提供银行帐号供认证。规避了用户虚假身份信息，一旦有法律问题可以问责
  - 大部分应用是免费的，约占60%
- 没有二次认证的审核流程，上传的应用会立刻发布
  - 如果开发者发布了恶意应用，虽然被发现后会被下架，事后也可能会被追究法律责任，但是对在发布后和下架前的时间段内下载该软件的用户，已经造成了损失
  - 由于开发者能够通过服务器控制客户端恶意行为的发作时间，在应用通过审核上架后再来激活恶意行为，即使通过审核也难以完全确保程序安全性

Google Play Store

- Android Market的安全现状非常不容乐观，仅2011年7月到2011年11月，Android Market中的恶意软件数量就增长了472%，用户的安全面临着严重的威胁
- Google公司于2012年2月3日公布了一种叫Bouncer（保镖）的病毒扫描技术，意指将在Android Market中引入应用的安全审核机制，以应对日益严峻的恶意软件威胁
- 2012年6月5日，有报道Google Bouncer服务可被绕过，有一些恶意软件因此流入了Android应用软件市场
- 2012年7月11日，赛门铁克的研究人员在安卓在线商店上发现大量恶意程序，这一发现再次将谷歌研发的在线安全服务推向了风口浪尖

中国移动MM

- 全球首个由运营商发起的线上软件商店，2009年8月17日正式发布
- 移动MM由中国移动互联网基地运维，位于广州
- 截至2011年6月30日，移动MM应用下载量已经超过3.6亿次，上架应用目前也已接近7万款
- 移动MM注册开发者超过200万，注册客户突破8000万

中国移动MM

- 2011年4月，由中国移动互联网基地牵头，联合广东省消费者协会、终端厂商、软件开发商等，成立了首个手机应用“绿色安全诚信联盟”，倡议和发布了全国首个手机软件绿色安全诚信标准
- 2011年7月6日，中国移动MM绿色软件诚信联盟（GSIU）高峰论坛在广州举行
- 中国移动MM依托绿色软件诚信联盟为平台，以互联网基地为首，联合腾讯、金山、盛大、诺基亚、MOTO等硬件厂商等产业链的相关巨头，开展“反病毒、反盗版、倡服务”手机安全大行动

中国移动MM

- 中国移动互联网基地已经成立了国内首个专业的手机应用监控测试中心，并已经形成了一套从开发、测试到上线的规范流程和机制，力保移动MM所有产品绿色安全
- 审核环节：内容审查、安全扫描、预测试、正式测试、质检
  - 通过自动加人工的多重安全审核机制，对应用的内容进行审查，对隐藏的病毒和恶意插件进行全面扫描
  - 所有的上线应用都会由人工进行测试
  - 300多人的团队对应用程序的安全性、终端适配性能等进行检测
  - 一个8人小团队专门负责发现与推荐精品应用
- 移动MM面临的困难
  - 终端平台适配难
  - 版权保护环境差
  - 客户付费下载应用的意愿低
  - 缺乏精品应用

中国移动MM应用安全审核

《移动应用商场应用发布协议》中公布了一些有关应用程序安全和涉及用户隐私方面的问题：

1．应用安全方面

1）含有任何病毒、木马等恶意代码或功能的应用将被驳回。

2）可导致手机终端硬件损坏或功能故障的应用将被驳回。

2.用户隐私方面

1）在未经用户确认的情况下读取、复制、转发、编辑、传播或删除终端内储存的文件或数据的应用将被驳回。

2）在采集、传送或使用用户隐私数据之前未通知并获得用户同意的程序将会被驳回。

3）使用推送消息发送敏感个人信息或机密信息的程序将会被驳回。

4）开发利用程序暗中收集用户密码或用户私人数据的将会从 MM 开发者中除名。

5）未获得用户初次同意便发送推送消息的程序将会被驳回。

6）使用推送消息发送非请求消息或用于钓鱼或群发垃圾邮件用途的程序将会被驳回。

3.其他恶意操作

1）beta、trial 等测试版的收费应用将会被驳回

2）误使用户操作造成设备损害的应用软件将会被驳回

3）快速耗光设备电量或产生过多热量的应用软件将会被驳回

本章内容

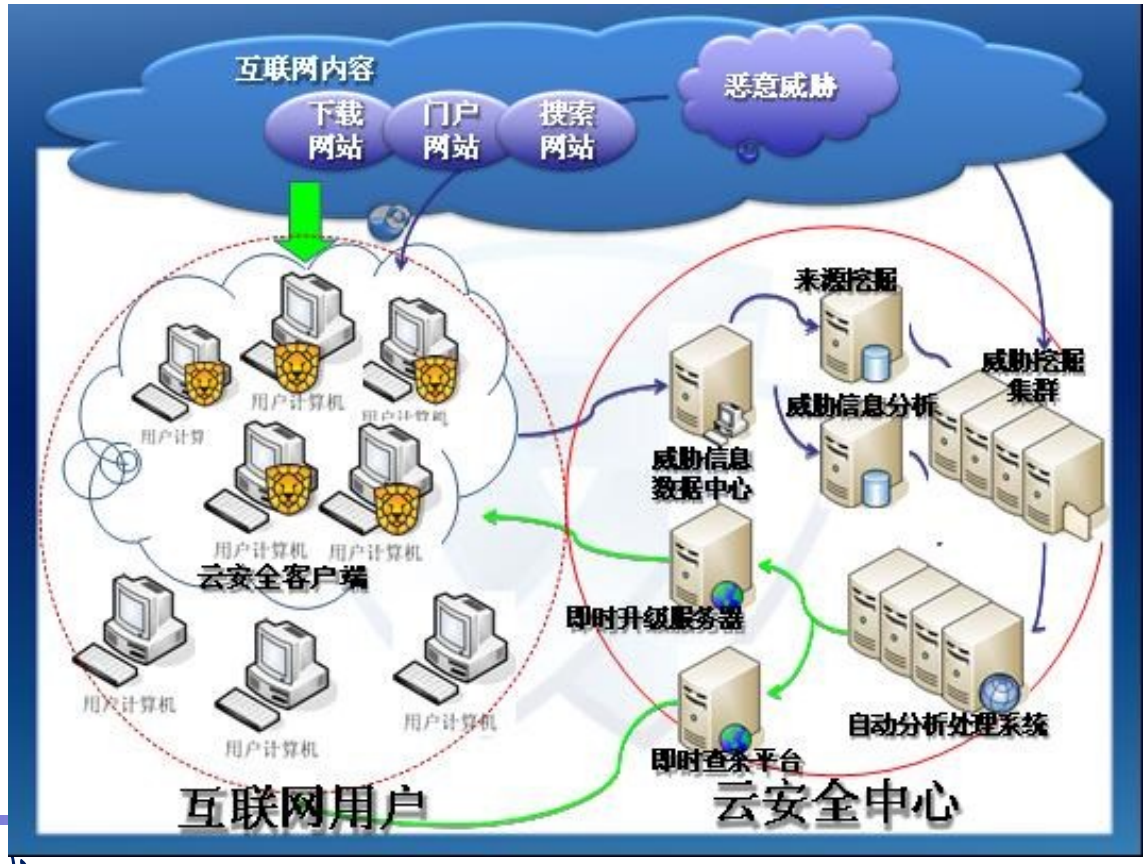
- Android概述
- Android安全问题
- Android安全机制
- Android系统机制的安全性
- 应用商店及其安全审核
- ✓ Android应用的安全检测

Android应用的安全检测

- 云安全
- 检测技术
  - 基于签名的检测技术
  - 基于行为的动态检测技术
  - 基于行为的静态检测技术
- 基于权限的恶意应用检测

云安全

- “云+端”结合的云安全
  - 云端：从各种渠道第一时间收集恶意软件
  - 终端：恶意软件的查杀和防护



云安全

- 各大安全公司都在推自己的“云安全”
- 问题
  - 只能查杀已知恶意软件
  - 效果依赖于“端”的覆盖率
  - “发现–分析–升级”，有时间空档期
  - 没有考虑应用程序的发布源头

基于签名的检测技术

- 当前手机平台上的反病毒措施主要依赖基于签名的检测，利用恶意软件的特征签名
- 特征签名：一段特殊代码或字符串
- 检测方法：匹配
- 无法检测新的和未知的恶意软件
- 基于单一签名的检测方法被加壳、代码混淆等技术所规避
- 必须保存每一个病毒新变种的特征签名。这导致特征库不断膨胀，进而使病毒特征检索复杂度变大，最终带来能耗的增加

手机平台恶意软件的特点

- 恶意软件数量是手机病毒的10倍
- 不需要像病毒或蠕虫一样大规模自我复制和分发，其只需放置到应用商店，即可得到大规模下载
- 恶意软件的特征签名比较隐蔽，绝大部分的恶意软件的实施恶意行为的代码都不尽相同，即签名都不一致
- 基于签名的检测技术，不太适合检测智能手机上的恶意软件

基于行为的检测技术

- 依靠监视文件的行为（如通过动态拦截或静态分析的方法获取程序的系统调用序列），结合已知的恶意行为模式（恶意软件的系列调用序列）
- 优点：
  - 特征库较小，无需频繁更新
  - 可用于预防未知病毒，因为行为模式类似
  - 能够抵抗混淆和加密攻击，因为其不改变程序行为模式
- 根据检测时机的不同，可分为动态和静态两种
  - 动态行为检测：在程序运行的过程中执行
  - 静态行为检测：在程序运行之前执行

基于行为的动态检测技术

- 由于在程序运行时执行
- 实时性要求较高，必须确保在恶意程序对系统产生损害前检测出威胁
- 能耗大，通常的解决方法是利用沙盒、虚拟机来模拟执行程序
- 漏报严重，因为恶意行为触发条件隐晦

基于行为的静态检测技术

- 程序运行之前，发现程序的潜在恶意行为
- 通过逆向工程手段，抽取程序的特征，如二进制序列，操作码序列，函数调用序列等
- 与动态行为检测相比，静态行为检测能耗更低（无需沙盒、虚拟机），风险更小，对实时性要求更低（在程序执行前进行检测）
- 适合应用商店或第三方测评机构的安全审核

基于权限的恶意应用检测

- 中科大提出，归属于基于行为的静态检测技术

- 恶意行为模式 = 权限使用方式
- 权限按照安全属性进行分类研究
- 基于权限的恶意行为分析

Android权限与安全

- 权限是 Android 平台的一种安全机制，以允许或限制应用程序访问受限的 API 和资源
- 默认情况下，Android 应用程序没有被授予任何权限，不允许它们访问设备上受保护的 API 或资源，从而保证了它们的安全
- 权限必须被请求，定义了定制的权限，文件和内容提供者就可以受到保护
- 确保在运行时检查、执行、授予和撤销权限

恶意行为模式

- 常规的软件行为模式
  - 二进制序列、操作码序列、函数调用序列等
- 以权限的使用方式作为行为模式
  - Android系统有着严格的权限管理机制
  - 恶意行为是对权限的恶意使用
  - 需要有能力跟踪权限的使用（通过污点传播）

权限的安全属性

- Android 2.3.3系统中有115个权限
- 通常按照功能来划分权限
- 需要研究权限的安全属性，并按照安全属性对权限进行分类，标记相应的危险等级
- 安全属性
  - 是否可能泄漏用户隐私
  - 是否可能控制系统资源
  - 是否可能产生费用等

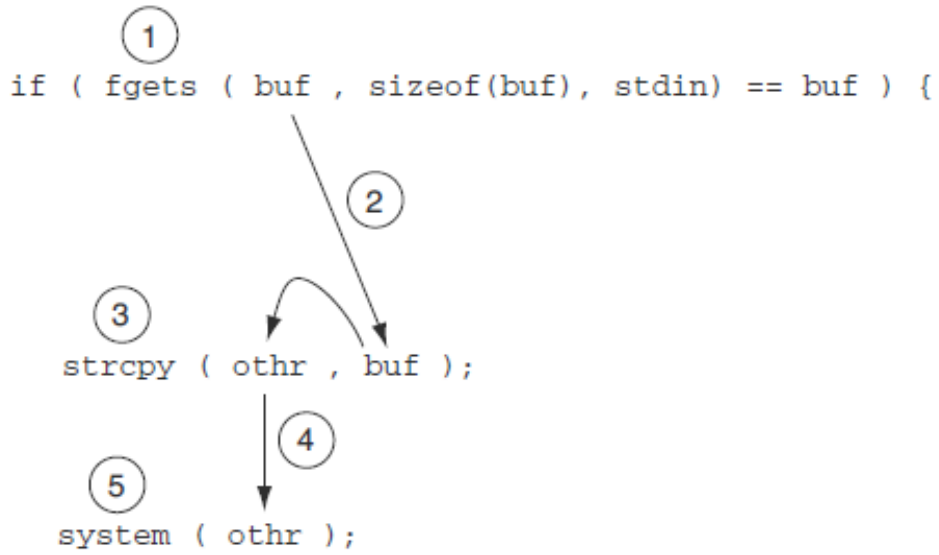
不可信行为相关权限初步分类

- 初步将不可信行为的相关权限初步分成四类：交互类、控制与系统资源类、隐私类和费用类

	交互类	控制与系统资源类	隐私类	费用类
包含的权限组	MESSAGES NETWORK	ACCOUNTS DEVELOPMENT_TOOLS HARDWARE_CONTROLS PHONE_CALLS STORAGE SYSTEM_TOOLS	PERSONAL_INFO LOCATION	CONST_MONEY
危险等级	最高	高	中	低

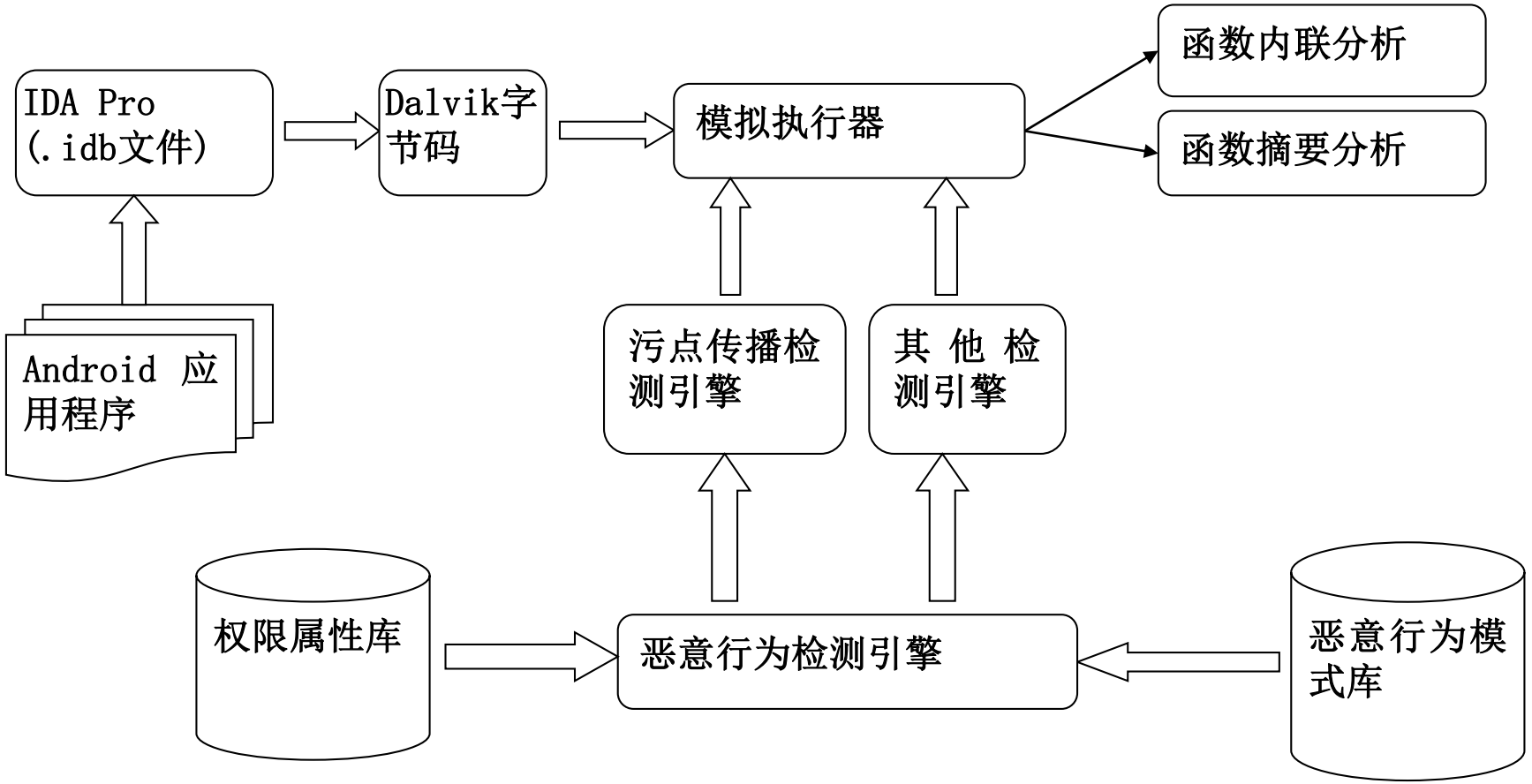
基于权限的恶意行为分析

- 引入污点传播算法
  - 效率和精度的平衡
  - 根据权限的属性设置锚点和污染源
- 恶意行为分析
  - 根据污点传播分析结果，结合恶意行为模式



1. A source rule for `fgets()` taints `buf` `othr`
2. Dataflow analysis connects uses of `buf`
3. A pass-through rule for `strcpy` taints
4. Dataflow analysis connects uses of `othr`
5. Because `othr` is tainted, a sink rule for `system()` reports a command injection vulnerability

基于权限的恶意软件检测系统



小结

- 了解Android系统安全机制
- 知道常见Android应用安全问题