理解Android安全机制

2016-01-26 安卓应用频道

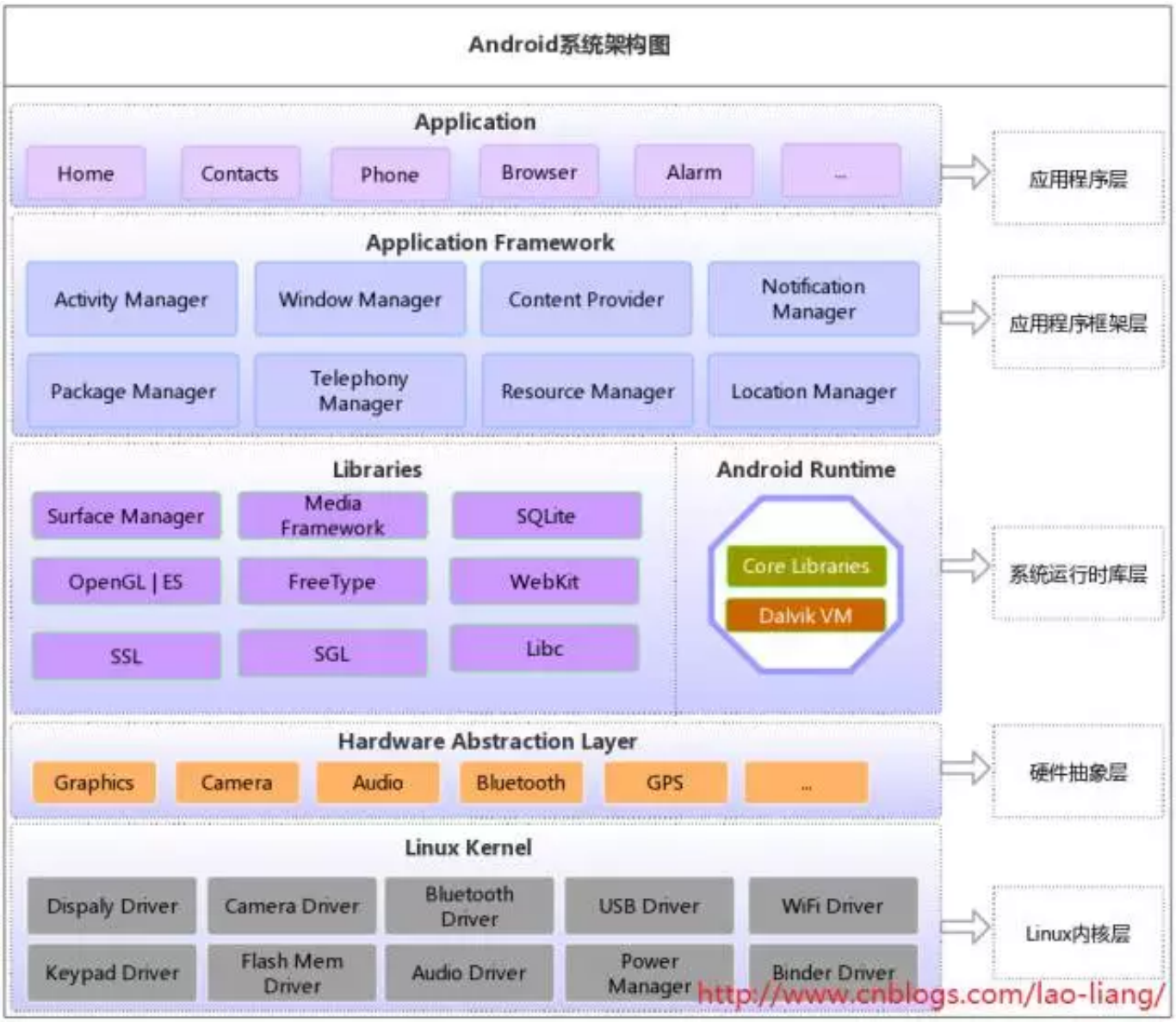
(点击上方公众号，可快速关注)

> 来源：LeoLiang
>
> 链接：http://www.cnblogs.com/lao-liang/p/5089336.html

本文从Android系统架构着手，分析Android的安全机制以SE Android，最后给出一些Android安全现状和常见的安全解决方案。

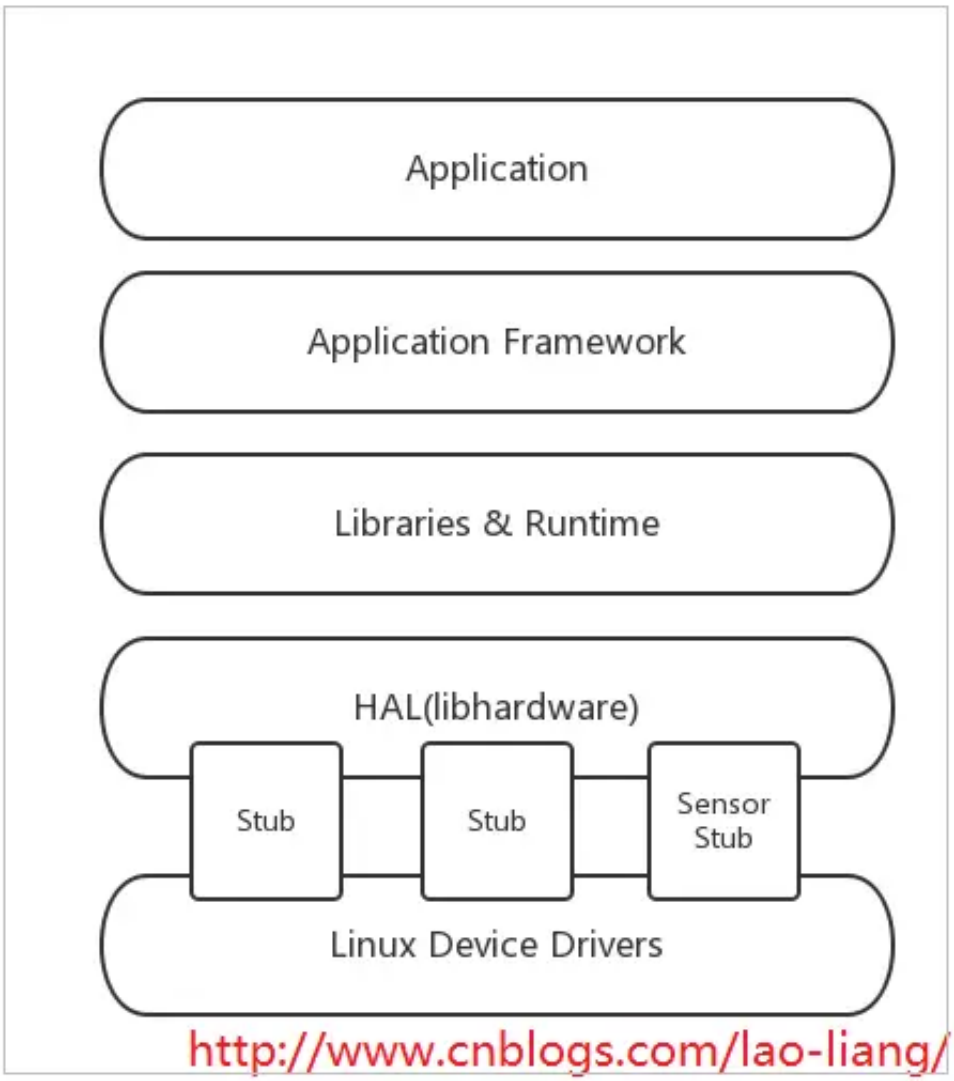
1、Android系统架构

Android采用分层的系统架构，由下往上分别是linux内核层、硬件抽象层、系统运行时库层、应用程序框架层和应用程序层。



Android以Linux操作系统内核为基础，实现硬件设备驱动、进程和内存管理、网络协议栈、电源管理等核心系统功能。除此以外，Android还增加了一些面向移动设备的特有功能，如低内存管理LMK(Low Memory Killer)、匿名共享内存(Ashmem: Anonymous Shared Memory)，以及进程间通信Binder机制。这些功能的增强进一步提升了Android在内存管理、进程间通信等方面的安全性。

Android之前的版本并不存在硬件抽象层。鉴于硬件厂商并不希望公开其设备驱动的源代码，Google对此将Linux内核驱动程序进行封装，屏蔽掉底层的实现细节，向上提供统一的接口，这就是硬件抽象层。



HAL(Hardware Abstraction Layer)规定了一套应用层对硬件层的读写和配置的统一接口，本质上是将硬件的驱动分为用户空间和内核空间，其中内核驱动程序运行在内核空间，HAL运行在用户空间。上图中的Stub，以so库的形式存在，可以理解为proxy。上层通过调用标识获得HAL的相关Stub，进而取得相应操作。

系统运行时库由系统类库和Android运行时构成。系统类库基本上用C/C++编写，基本功能说明如下：

系统类库	功能说明
Surface Manager	执行多个应用程序时，管理子系统显示，对2D和3D图形提供支持
Media Framework	基于PacketVideoOpenCore多媒体库，支持多种常用的音频视频格式录制和回放。支持的编码格式包括MPEG4、MP3、H.264、AAC、ARM
SQLite	本地小型关系型数据库，以代替传统的耗费资源JDBC API
OpenGL | ES	基于OpenGL ES 1.0 API标准实现的3D跨平台图形库
FreeType	用于显示位图和矢量字体
Webkit	Web浏览器软件引擎
SSL	安全套接层
SGL	底层2D图形引擎
Libc(bionic libc)	继承自BSD的C函数库bionic libc，更适合基于嵌入式Linux的移动设备

当然，还有Android NDK(Native Development Kit)，使得应用程序可以不依赖Dalvik虚拟机进行开发。Android运行时核心库提供android.os, android.net, android.media等核心API，而Dalvik虚拟机依赖Linux内核，实现进程隔离与线程调度管理、安全与异常管理、垃圾回收等功能，并被改进以适应低内存、低处理器速度的移动设备环境。

再往上就是应用程序框架层了。一系列的Android应用程序所需的类库，使得开发人员可以快速地进行程序开发，也可以通过继承实现个性化的扩展。如Activity Manager负责主线程ActivityThread的创建、Activity生命周期的维护，并为窗口提供交互的接口。

应用层就是与用户直接交互的应用程序，如SMS短信、图片浏览器、地图以及开发人员所开发的应用程序。

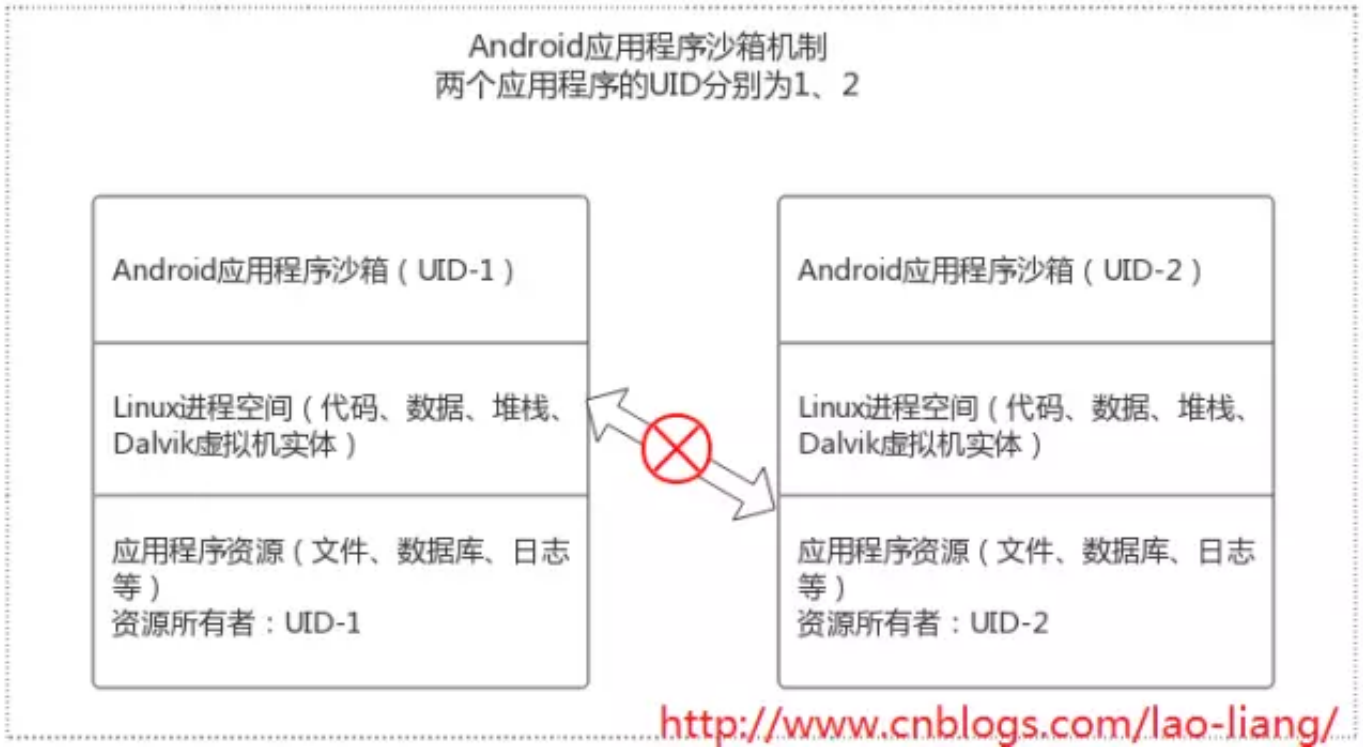
2、Android安全机制

Android将安全设计贯穿系统架构的各个层面，覆盖系统内核、虚拟机、应用程序框架层以及应用层各个环节，力求在开放的同时，也恰当保护用户的数据、应用程序和设备的安全。Android安全模型主要提供以下几种安全机制：

-
- 进程沙箱隔离机制
- 应用程序签名机制

- 权限声明机制
- 访问控制机制
- 进程通信机制
- 内存管理机制

进程沙箱隔离机制，使得Android应用程序在安装时被赋予独特的用户标识（UID），并永久保持。应用程序及其运行的Dalvik虚拟机运行在独立的Linux进程空间，与其它应用程序完全隔离。



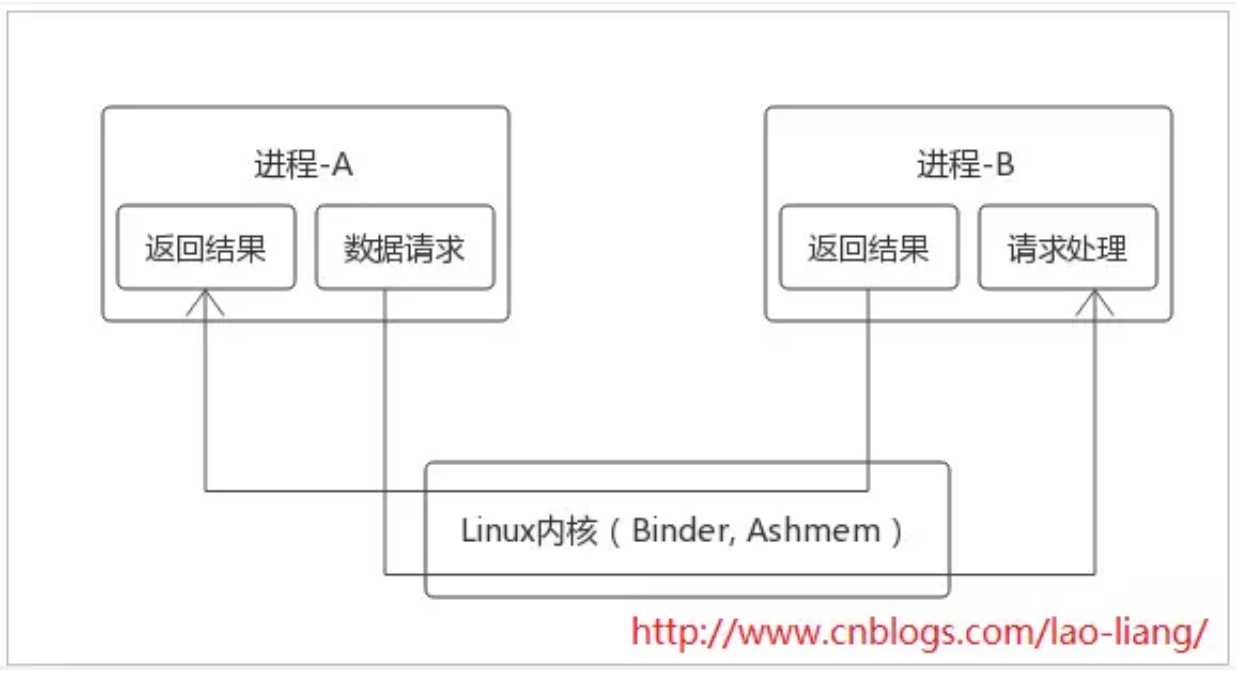
在特殊情况下，进程间还可以存在相互信任关系。如源自同一开发者或同一开发机构的应用程序，通过Android提供的共享UID（Shared UserId）机制，使得具备信任关系的应用程序可以运行在同一进程空间。

应用程序签名机制，规定APK文件必须被开发者进行数字签名，以便标识应用程序作者和在应用程序之间的信任关系。在安装应用程序APK时，系统安装程序首先检查APK是否被签名，有签名才能安装。当应用程序升级时，需要检查新版应用的数字签名与已安装的应用程序的签名是否相同，否则，会被当做一个新的应用程序。Android开发者有可能把安装包命名为相同的名字，通过不同的签名可以把他们区分开来，也保证签名不同的包不被替换，同时防止恶意软件替换安装的应用。

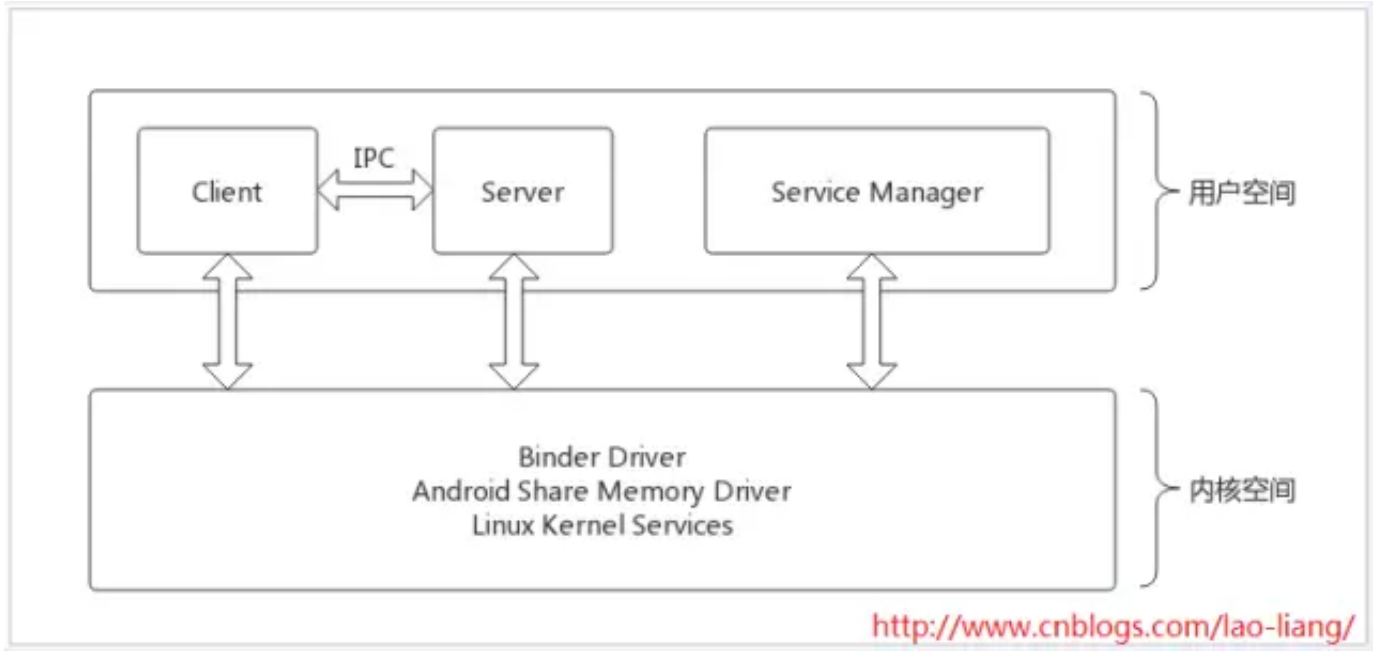
权限声明机制，要想获得在对象上进行操作，就需要把权限和此对象的操作进行绑定。不同级别要求应用程序行使权限的认证方式也不一样，Normal级申请就可以使用，Dangerous级需要安装时由用户确认，Signature和Signatureorsystem级则必须是系统用户才可用。

访问控制机制，确保系统文件和用户数据不受非法访问。

进程通信机制，基于共享内存的Binder实现，提供轻量级的远程进程调用（RPC）。通过接口描述语言（AIDL）定义接口与交换数据的类型，确保进程间通信的数据不会溢出越界。



Linux进程视角



应用程序视角

内存管理机制，基于Linux的低内存管理机制，设计实现了独特的LMK，将进程重要性分级、分组，当内存不足时，自动清理级别进程所占用的内存空间。同时，引入的Ashmem内

存机制，使得Android具备清理不再使用共享内存区域的能力。

正是因为Android采用多层架构，在保护信息安全的同时，也保证开放平台的灵活性。

3、SE Android

Android是一个基于Linux内核的系统，像传统的Linux系统一样，Android也有用户的概念。只不过这些用户不需要登录，也可以使用Android系统。Android系统将每一个安装在系统的APK都映射为一个不同的Linux用户。也就是每一个APK都有一个对应的UID和GID，这些UID和GID在APK安装的时候由系统安装服务PackageManagerService分配。Android沙箱隔离机制就是建立在Linux的UID和GID基础上。

这种基于Linux UID/GID的安全机制存在什么样的问题呢？

Linux将文件的权限划分为读、写和执行三种，分别用字母r、w和x表示。每一个文件有三组读、写和执行权限，分别针对文件的所有者、文件所有者所属的组以及除了所有者以及在所有者所属组的用户之外所有其它用户。这样，如果一个用户想要将一个自己创建的文件交给另外一个用户访问，那么只需要相应地设置一下这个文件的其它用户权限位就可以了。所以，在Linux系统中，文件的权限控制在所有者的手中。因此，这种权限控制方式就称为自主式的，正式的英文名称为Discretionary Access Control，简称为DAC。

在理想情况下，DAC机制是没有问题的。然而，一个用户可能会不小心将自己创建的文件的权限位错误地修改为允许其它用户访问。如果这个用户是一个特权用户，并且它错误操作的文件是一个敏感的文件，那么就会产生严重的安全问题。这种误操作的产生方式有三种：

- 用户执行了错误的命令
- 负责执行用户命令的程序有Bug
- 负责执行用户命令的程序受到攻击

后来，Linux内核采用了必要的访问控制机制：SE Linux（Security-Enhanced Linux），它采用了一种强制存取控制MAC（Mandatory Access Control）策略的实现方式，目的在于通过限制系统中的任何进程以及用户对资源的访问，保护内核安全。而SE Android（Security-Enhanced Android）是Android与SE Linux的结合，由美国NSA在2012年推出的Android操作系统安全强化套件，以支持在Android平台上使用SE Linux。

目前SE Android系统中的策略机制主要有三种：

- 安装时MAC（install-time MAC）
- 权限取消（permission revocation）

- 权限标签传播（tag propagation）

安装时MAC通过查找MAC策略配置来检查应用程序的权限。权限取消可以为已安装的应用取消权限，该机制在应用程序运行的权限检查时通过查找权限取消列表来取消应用的某些权限。权限标签传播是一种污点跟踪方式的应用，Android系统的权限作为抽象的标签映射到MAC策略配置文件中。

SE Android安全机制所要保护的对象是系统中的资源，这些资源分布在各个子系统中。实际上，系统中需要保护的资源非常多，除了文件之外，还有进程、socket和IPC等。SE Android是一个复杂的安全模型，本文就不进一步分析了。想了解更多，请参考：SEAndroid安全机制框架分析(http://blog.csdn.net/luoshengyang/article/details/37613135)

4、Android应用安全解决方案

Android应用会遇到各种各样的安全性问题，如何从宏观上了解各种安全隐患，积极采取适当的防御措施便变得尤为重要。那么，Android应用面临哪些安全问题呢？

- 病毒
- 关键信息泄露
- APP重打包
- 进程被劫持
- 数据在传输过程遭劫持
- Webview漏洞

病毒不用多说了，都是一些恶意软件。关键信息泄露，可能有些开发者并不十分留意。虽然Java代码可以做混淆，但是Android的几大组件的创建方式是依赖注入的方式，因此不能被混淆。而且目前常用的一些反编译工具比如apktool等能够毫不费劲地还原Java里的明文信息，native里的库信息也可以通过objdump或IDA获取。因此一旦Java或native代码里存在明文敏感信息，基本上就是毫无安全而言的。重打包即通过反编译后重新加入恶意的代码逻辑，重新打包一个APK文件。进程被劫持一般通过进程注入或者调试进程的方式来hook进程，改变程序运行的逻辑和顺序，从而获取程序运行的内存信息。hook需要获取root权限或者跟被hook进程相同的权限。如果手机没被root，被劫持的可能性还是较小。数据在传输过程遭劫持，一般来说是由于数据明文传输或没使用HTTPS。Webview漏洞一般由于JS注入。

现实中，出现的问题可能比上面提及的还要多。总的来说，应该从以下几个方面来应对Android开发的常见安全问题：

- 应用权限控制。通过控制应用程序的权限防止恶意应用对系统造成破坏，采取的措施包

括合理使用系统内置权限和应用程序自定义权限。

- 应用程序签名。采用数字签名为应用程序签名。
- 应用加固。应用加固包括病毒扫描、防注入、防调试、防篡改四个模块，目前行业内已经出现了很多的应用加固解决方案，如360应用加固、腾讯云应用加固、百度应用加固等等。
- 静态代码分析。通过静态代码分析工具lint监测安全隐患，对代码进行优化。
- 防火墙。必要时为Android设备安装防火墙，以防止远程网络攻击。
- 数据存储加密。采用加密的方式保护应用程序敏感数据，如利用SQLCipher加密SQLite数据库。
- 应用程序组件开发的安全要点。Activity, Service, Content Provider, Broadcast Receiver等组件在代码层面应采取的安全措施。它们每一个都可以通过隐式的Intent方式打开，所以这些组件只要不是对外公开的必须在AndroidManifest里面注明exported为false，禁止其它程序访问我们的组件。对于要和外部交互的组件，应当添加访问权限的控制，还需要要对传递的数据进行安全的校验。

参考：

《Android安全机制解析与应用实践》

SEAndroid安全机制简要介绍和学习计划

> http://blog.csdn.net/luoshengyang/article/details/35392905

Android应用安全现状与解决方案（学习资料）

> http://blog.csdn.net/yzzst/article/details/46471277