



# 移动互联网安全

## 第六章 Android 安全基础

黄 玮



# 内容提纲

- Android安全机制概览
- Android应用软件基础
- Android应用软件的安全策略与机制
- 典型安全漏洞原理与实验

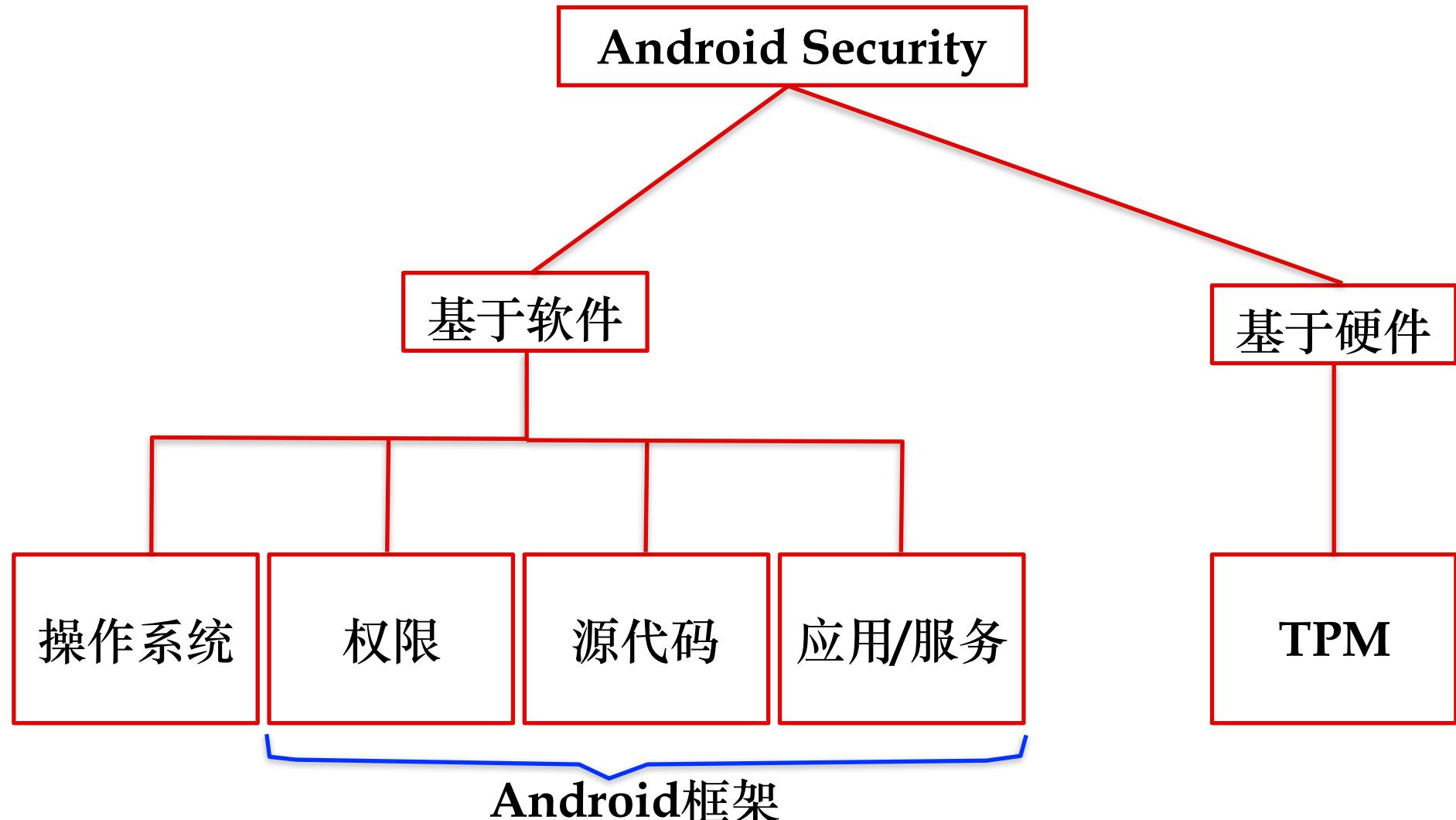


# ANDRID安全机制概览

中国传媒大学



# 层次化的Android安全机制





# Android安全机制——操作系统

## • Linux 内核

```
shell@android:/ $ uname -a
Linux localhost 3.4.0-geaea5c4 #1 SMP PREEMPT Fri May 23 21:19:37 CST 2014 armv7l GNU/Linux
```

## • 进程隔离

- 每个进程拥有自己的地址空间
- 进程彼此间不可访问其他进程内存
- (用户态) 进程无法访问内核内存
- 进程必须使用系统调用才能访问内核

```
shell@android:/data/system/users # ps | grep -v root
USER     PID   PPID  VSZ RSS   WCHAN   PC           NAME
nobody   173    1    6012  4   ffffffff 0000c4c0 S /sbin/rmt_storage
system   315    1    1400  632  c05d3f28 402e2008 S /system/bin/servicemanager
radio    328    1    16660 3276  ffffffff 4019c7b4 S /system/bin/rild
system   329    1    63104 8436  ffffffff 402e3008 S /system/bin/surfaceflinger
drm     331    1    13516 3388  ffffffff 40294008 S /system/bin/drmserver
media   332    1    40420 8732  ffffffff 40288008 S /system/bin/mediaserver
bluetooth 333    1    1816  724  c014c1cc 401b6f7c S /system/bin/dbus-daemon
install  336    1    1328  712  c073a550 40260d98 S /system/bin/installd
keystore 338    1    2288  1196  c06894b4 40193ab8 S /system/bin/keystore
system   342    1    1272  472   c014c1cc 40280f7c S /system/bin/qrnd
camera   343    1    11728 1948  c014c1cc 401e2f7c S /system/bin/mm-qcamera-daemon
system   345    1    8400  2148  ffffffff 4030cf04 S /system/bin/cnd
gps     346    1    5640  1060  ffffffff 402b9e18 S /system/bin/wiperface
system   348    1    8232  1080  ffffffff 4025f128 S /system/bin/ATFWD-daemon
system   349    1    4384  560   ffffffff 401c2f7c S /system/bin/ssr_diag
system   350    1    2580  688   c008c4ec 402d9da4 S /system/bin/qseecomd
system   351    1    7948  948   ffffffff 4027e128 S /system/bin/time_daemon
system   352    1    1516  612   c041d3dc 40279d98 S /system/bin/wcnss_service
shell    360    1    4576  224   ffffffff 00014e94 S /sbin/adbd
radio    386    1    15160 800   ffffffff 4022a12c S /system/bin/qmuxd
gps     390    1    7424  956   ffffffff 4015a128 S /system/bin/gpsone_daemon
radio    393    1    8152  1028  ffffffff 401c012c S /system/bin/qmiproxy
gps     395    1    1316  640   c014c1cc 400dc12c S /system/bin/location-mq
radio    397    1    8180  880   ffffffff 40208128 S /system/bin/netmgrd
gps     399    1    11148 1884  ffffffff 40270128 S /system/bin/quipc_main
gps     401    1    5292  1496  ffffffff 40256d98 S /system/bin/quipc_igsn
system   427    350  4680  484   ffffffff 402d9744 S /system/bin/qseecomd
media   537    1    9664  2592  ffffffff 402bd008 S /system/bin/bootqrdsound
system   832    330  612112 62608 ffffffff 40225008 S system_server
```



# Android安全机制——操作系统

- UID/GID

```
shell@android:/ $ id  
uid=2000(shell) gid=2000(shell) groups=1003(graphics),1004(input),1007(log),1009(mount),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003/inet),3006(net_bw_stats)  
shell@android:/ $ ls -l | grep -v root  
drwxrwx--- system cache 1970-02-11 14:29 cache  
drwxrwx--x system system 2014-09-21 13:04 data  
dr-xr-x--- system system 1970-01-01 08:00 firmware  
drwxrwx--x system system 2009-01-02 21:00 persist  
dr-xr-x--- system sdcard_r 1970-01-01 08:00 storage
```

- 每个进程都有一个UID/GID
- 相同UID和GID的进程拥有相同的访问权限
  - 文件系统访问权限
  - 信号
  - ...



# Android安全机制——操作系统

- Capabilities

**CAPABILITIES(7)**      Linux Programmer's Manual      **CAPABILITIES(7)**

**NAME**  
capabilities - overview of Linux capabilities

**DESCRIPTION**  
For the purpose of performing permission checks, traditional UNIX implementations distinguish two categories of processes: privileged processes (whose effective user ID is 0, referred to as superuser or root), and unprivileged processes (whose effective UID is nonzero). Privileged processes bypass all kernel permission checks, while unprivileged processes are subject to full permission checking based on the process's credentials (usually: effective UID, effective GID, and supplementary group list).

Starting with kernel 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled. Capabilities are a per-thread attribute.

**Capabilities List**  
The following list shows the capabilities implemented on Linux, and the operations or behaviors that each capability permits:

**CAP\_AUDIT\_CONTROL** (since Linux 2.6.11)  
Enable and disable kernel auditing; change auditing filter rules; retrieve auditing status and filtering rules.

**CAP\_AUDIT\_WRITE** (since Linux 2.6.11)  
Write records to kernel auditing log.

Manual page capabilities(7) line 1 (press h for help or q to quit)



# Android安全机制——操作系统

- Capabilities

- root权限无所不能

- 有些时候仅需要部分root权限

- 最小化授权原则

- Android上的安装包守护进程installd仅用到了部分root权限而并非使用完整root权限

- 精细化授权机制

```
shell@android:/ $ ps | grep install
install  336   1      1328    712    ffffffff  00000000 S /system/bin/installd
```



# Android安全机制——操作系统

- SELinux

- Linux系统默认采用的自主访问控制策略
  - 操作系统用户自主决定是否允许访问/操作
- SELinux在已有的Linux自主访问控制机制基础之上增加了强制访问控制机制
  - 所有操作除非声明允许，否则即使root权限也无法操作
- 优点：即使恶意进程提升权限到root，也无法突破SELinux的权限限制
- 缺点：和已有应用程序的兼容性较差



# Android安全机制——操作系统

- 文件系统分区

Filesystem	Size	Used	Free	Blksize
/dev	430M	136K	430M	4096
/mnt/secure	430M	0K	430M	4096
/mnt/asec	430M	0K	430M	4096
/mnt/obb	430M	0K	430M	4096
/system	1009M	667M	341M	4096
/data	1G	835M	276M	4096
/cache	189M	4M	184M	4096
/persist	7M	4M	3M	4096
/firmware	63M	44M	19M	16384
/storage/sdcard1	979M	5M	974M	4096
/storage/sdcard0	7G	442M	6G	32768
/mnt/secure/asec	7G	442M	6G	32768

Android  
独有的



# Android安全机制——操作系统

- 文件系统分区
  - RAM disk => Read-Only
  - System image => Read-Only (unless update)
  - Data image => Read-Write (specific user permissions needed)
  - Cache => Read-Write
  - Recovery => Not mounted by default
  - Virtual filesystems (proc, sysfs, etc.)
  - “sdcard” => Read-Write (world readable/writable)



# Android安全机制——操作系统

- init.rc
  - 定义了Android系统用户态程序的启动过程和步骤

```
# Copyright (C) 2012 The Android Open Source Project
#
# IMPORTANT: Do not create world writable files or directories.
# This is a common source of Android security bugs.
#
import /init.usb.rc
import /init.${ro.hardware}.rc
import /init.trace.rc
import /init.hw.debug.rc

on early-init
    # Set init and its forked children's oom_adj.
    write /proc/1/oom_adj -16

    # Set the security context for the init process.
    # This should occur before anything else (e.g. ueventd) is started.
    setcon u:r:init:s0

    start ueventd

/init.rc
```



# Android安全机制——操作系统

- /dev/\*
  - 所有硬件设备都通过device nodes访问
  - device nodes的权限设置和普通文件一样
  - ueventd进程负责初始化上述device nodes
  - App对绝大多数设备无访问权限

```
shell@android:/ $ ps | grep ueventd
root      137     1    480     208   ffffffff  00000000 S /sbin/ueventd
```



# Android安全机制——操作系统

- /dev/socket/\*

```
shell@android:/dev/socket $ ls -l
srw-rw---- system    system          2009-01-02 21:00 adbd
srw-rw---- bluetooth bluetooth      2014-09-21 13:04 bluetooth
srw-rw---- root      inet            2009-01-02 21:00 cnd
srw-rw---- bluetooth bluetooth      2009-01-02 21:00 dbus
srw-rw---- bluetooth bluetooth      2014-09-21 13:04 dbus_bluetooth
srw-rw---- root      inet            2009-01-02 21:00 dnsproxyd
srw----- system    system          2009-01-02 21:00 installd
srw-rw-rw- root      root            2009-01-02 21:00 keystore
srw-rw---- root      system          2009-01-02 21:00 mdns
srw-rw-rw- root      system          2014-09-21 13:04 mpctl
srw-rw---- root      system          2014-09-21 13:04 mpdecision
srw-rw---- root      system          2009-01-02 21:00 netd
srwxrwx--- system    system          2009-01-02 21:00 oeminfoserver
srw-rw-rw- root      root            1970-01-01 08:00 property_service
drwxrws--- media    audio           2014-09-21 13:04 qmux_audio
drwxrws--- bluetooth bluetooth      2014-09-21 13:04 qmux_bluetooth
drwxrws--- gps      gps             2014-09-21 13:04 qmux_gps
drwxrws--- radio    radio           2014-11-28 17:19 qmux_radio
srw-rw---- root      radio           2009-01-02 21:00 rild
srw-rw---- radio    system          2009-01-02 21:00 rild-debug
srw-rw---- root      root            2009-01-02 21:00 thermal-recv-client
srw-rw---- root      camera          2009-01-02 21:00 thermal-send-client
srw-rw---- root      mount           2009-01-02 21:00 vold
srw-rw---- wifi     wifi            2014-09-21 13:04 wpa_wlan0
srw-rw---- root      system          2009-01-02 21:00 zygote
shell@android:/dev/socket $
```



# Android安全机制——操作系统

- 原生守护进程

- 身份认证方式接受访问

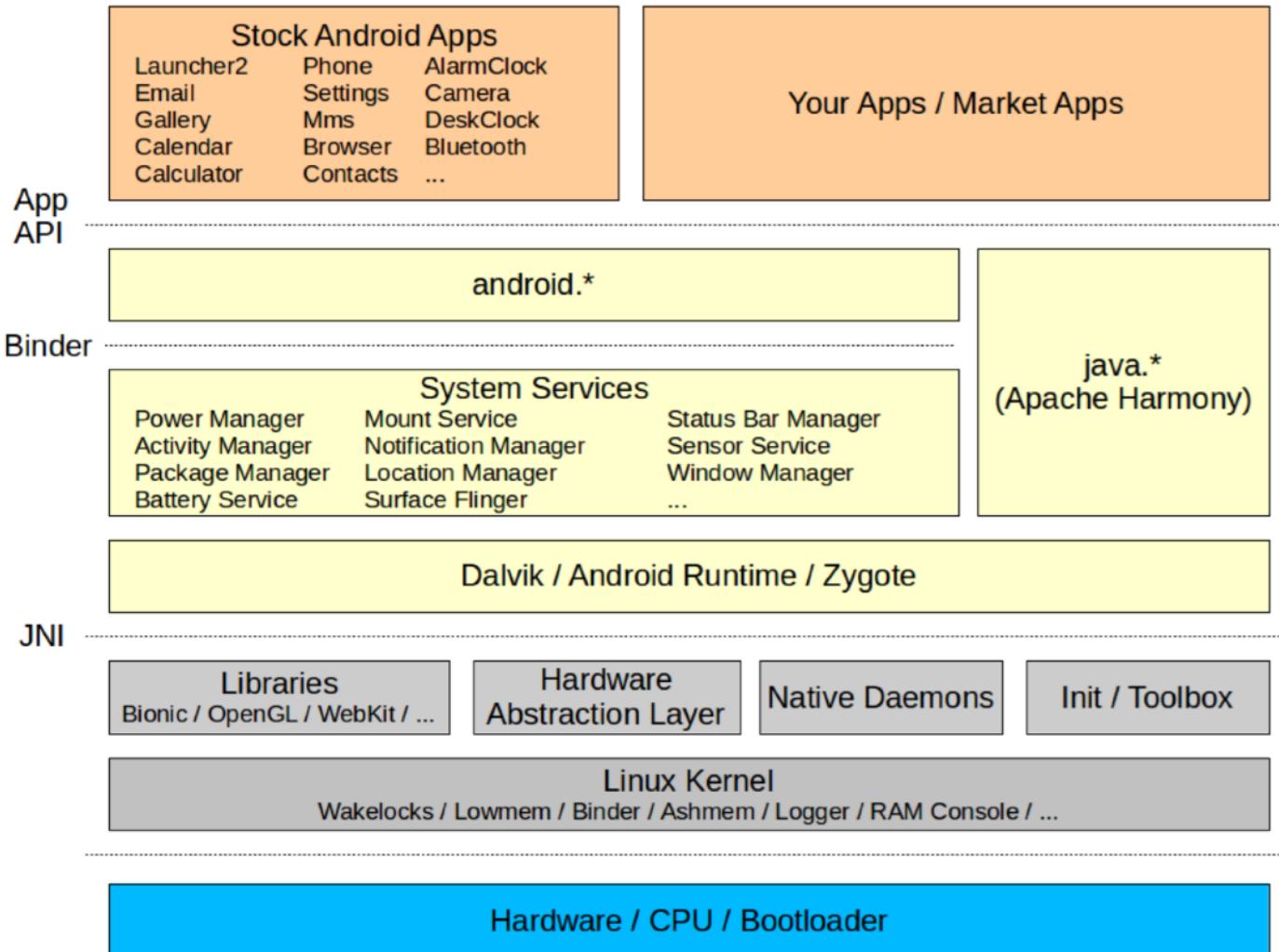
- servicemanager
    - init property service

- 权限屏蔽方式限制访问

- vold / netd / rild



# Linux 内核与 Android 框架的关系





# Android安全机制——Android框架

- 基于Android框架的权限机制
- 应用程序签名
- 多用户支持
- 设备管理
- SEAndroid



# Android安全机制——Android框架

- 基于Android框架的权限机制
  - 通过Binder访问受保护的系统资源（例如/dev/\*）
  - Binder机制本身不强制要求访问控制
  - 系统服务对每次系统调用进行权限检查
  - App运行时通过checkCallingOrSelfPermission()检查是否具备某项访问权限

```
2. vim checkCallingOrSelfPermission.java (Python)
1
2 private boolean checkWriteExternalPermission()
3 {
4     String permission = "android.permission.WRITE_EXTERNAL_STORAGE";
5     int res = getContext().checkCallingOrSelfPermission(permission);
6     return (res == PackageManager.PERMISSION_GRANTED);
7 }
```

NORMAL >> <ngOrSelfPermission.java java < utf-8[unix] < 14% : 1: 0 < ! trailing[6]



# Android安全机制——Android框架

## • 应用程序签名

- 应用程序分发商必须对App签名
- 应用程序分发商的身份认证非强制
  - 应用商店自行制定和执行认证规范，例如上传营业执照、身份证件扫描件等
- 基于Java SDK中的keytool

```
➜ - keytool -genkey -v -keystore poc.keystore -alias haha -keyalg RSA -keysize 2048 -validity 10000  
输入密钥库口令：  
再次输入新口令：  
您的名字与姓氏是什么？  
[Unknown]: huang  
您的组织单位名称是什么？  
[Unknown]: CUC  
您的组织名称是什么？  
[Unknown]: cs  
您所在的城市或区域名称是什么？  
[Unknown]: Beijing  
您所在的省/市/自治区名称是什么？  
[Unknown]: Peking  
该单位的双字母国家/地区代码是什么？  
[Unknown]: CN  
CN=huang, OU=CUC, O=cs, L=Beijing, ST=Peking, C=CN是否正确？  
[否]: 是  
  
正在为以下对象生成 2,048 位 RSA 密钥对和自签名证书 (SHA256withRSA) (有效期为 10,000 天):  
CN=huang, OU=CUC, O=cs, L=Beijing, ST=Peking, C=CN  
输入 <haha> 的密钥口令  
(如果和密钥库口令相同，按回车):  
[正在存储 poc.keystore]
```

## 实验：对APK进行签名



# Android安全机制——应用程序签名

- 所有的应用程序都必须有数字证书，Android系统不会安装一个没有数字证书的应用程序
- Android程序包使用的数字证书可以是自签名的，不需要一个权威的数字证书机构签认认证
- 如果要正式发布一个Android，必须使用一个合适的私钥生成的数字证书来给程序签名，而不能使用adt插件或者ant工具生成的调试证书来发布
- 数字证书都是有有效期的，Android只是在应用程序安装的时候才会检查证书的有效期。如果程序已经安装在系统中，即使证书过期也不会影响程序的正常功能
- Android使用标准的java工具 Keytool and Jarsigner 来生成数字证书，并给应用程序包签名



# Android安全机制——应用程序签名

- 包名相同的APK被安装到系统中的过程是应用『升级』
- 包名相同但『签名不同』的APK无法执行『升级』
- 应用『升级』会覆盖原有的应用程序安装目录但不会覆盖应用程序数据目录
- 应用程序模块化：Android 系统可以允许同一个证书签名的多个应用程序在一个进程里运行，系统实际把他们作为一个单个的应用程序，此时就可以把我们的应用程序以模块的方式进行部署，而用户可以独立的『升级』其中的一个模块
- 代码或者数据共享：Android 提供了基于签名的权限机制，那么一个应用程序就可以为另一个以相同证书签名的应用程序公开自己的功能。以同一个证书对多个应用程序进行签名，利用基于签名的权限检查，你就可以在应用程序间以安全的方式共享代码和数据了



# Android安全机制——Android框架

## • 多用户支持

/data/user

每个app的数据目录

```
shell@android:/dev # ls -l /data/user/
lrwxrwxrwx root root 2009-01-02 21:00 0 -> /data/data/
-rw----- root root 16384 2014-10-26 12:40 log_12139.etb
-rw----- root root 16384 2014-09-18 15:21 log_13064.etb
-rw----- root root 16384 2014-09-20 09:37 log_16070.etb
-rw----- root root 16384 2014-09-20 09:37 log_16097.etb
-rw----- root root 16384 2014-11-30 17:13 log_1664.etb
-rw----- root root 16384 2014-09-14 13:38 log_28745.etb
-rw----- root root 16384 2014-11-27 17:10 log_30166.etb
-rw----- root root 16384 2014-09-14 13:47 log_30245.etb
-rw----- root root 16384 2014-09-14 13:54 log_31742.etb
-rw----- root root 16384 2014-08-31 22:50 log_3688.etb
-rw----- root root 16384 2014-11-18 11:50 log_3801.etb
-rw----- root root 16384 2014-11-16 23:30 log_4130.etb
-rw----- root root 16384 2014-11-30 16:56 log_472.etb
-rw----- root root 16384 2014-10-01 07:16 log_690.etb
-rw----- root root 16384 2014-11-14 18:24 log_8314.etb
-rw----- root root 16384 2014-11-14 18:24 log_8321.etb
shell@android:/dev # ls -l /data/system/users/
drwx----- system system 2014-11-28 18:15 0
-rw----- system system 156 1970-01-02 08:00 0.xml
-rw----- system system 124 2014-09-21 13:04 userlist.xml
```

/data/system/users  
每个用户的账户数据目录



# Android安全机制——Android框架

- 设备管理

- 提供BYOD管理API
- 功能简陋
- 唯一有效的应用是强制密码复杂度
- 不支持
  - 批量配置
  - 不同app的个性化（权限、功能等）定制配置



# Android安全机制——Android框架

- SEAndroid
  - Android框架层的强制访问控制机制支持
  - Android 4.4首次默认启用SELinux
    - 对部分root权限后台进程设置了安全标签，限制访问权限
  - 已经进入Android 5.0 AOSP主干分支代码
    - 所有服务和app强制启用了SELinux
  - MAC中间件未被加入Android 5.0源代码主干分支
    - 扩展了对Android权限和进程间通信模型的保护



# Android安全机制——Android框架

## • SEAndroid

- 所有访问操作先进行DAC检查，通过后，再进行MAC检查
- ls -Z
- ps -Z
- 《网络安全》chap0x02介绍的操作系统MAC相关理论基础

```
➜ teaching adb devices
List of devices attached
d07ab5bbf830    device
emulator-5554    device

➜ teaching adb -s emulator-5554 shell
root@generic_x86:/ # ls -Z
drwxr-xr-x root      root          u:object_r:rootfs:s0 acct
drwxrwx--- system   cache          u:object_r:cache_file:s0 cache
dr-x----- root      root          u:object_r:rootfs:s0 config
lrwxrwxrwx root     root          u:object_r:rootfs:s0 d -> /sys/kernel/debug
drwxrwx--- system   system         u:object_r:system_data_file:s0 data
-rw-r--r-- root     root          u:object_r:rootfs:s0 default.prop
drwxr-xr-x root     root          u:object_r:device:s0 dev
lrwxrwxrwx root     root          u:object_r:rootfs:s0 etc -> /system/etc
-rw-r--r-- root     root          u:object_r:rootfs:s0 file_contexts
-rw-r----- root     root          u:object_r:rootfs:s0 fstab.goldfish
-rwxr-x--- root     root          u:object_r:rootfs:s0 init
-rwxr-x--- root     root          u:object_r:rootfs:s0 init.environ.rc
-rwxr-x--- root     root          u:object_r:rootfs:s0 init.goldfish.rc
-rwxr-x--- root     root          u:object_r:rootfs:s0 init.rc
-rwxr-x--- root     root          u:object_r:rootfs:s0 init.trace.rc
-rwxr-x--- root     root          u:object_r:rootfs:s0 init.usb.rc
drwxrwxr-x root     system        u:object_r:rootfs:s0 mnt
dr-xr-xr-x root     root          u:object_r:proc:s0 proc
-rw-r--r-- root     root          u:object_r:rootfs:s0 property_contexts
drwx----- root     root          u:object_r:rootfs:s0 root
drwxr-x--- root     root          u:object_r:rootfs:s0 sbin
lrwxrwxrwx root     root          u:object_r:rootfs:s0 sdcard -> /storage/sdcard
-rw-r--r-- root     root          u:object_r:rootfs:s0 seapp_contexts
-rw-r--r-- root     root          u:object_r:rootfs:s0 sepolicy
drwxr-x-x root     sdcard_r      u:object_r:rootfs:s0 storage
dr-xr-xr-x root     root          u:object_r:sysfs:s0 sys
drwxr-xr-x root     root          u:object_r:system_file:s0 system
-rw-r--r-- root     root          u:object_r:rootfs:s0 ueventd.goldfish.rc
-rw-r--r-- root     root          u:object_r:rootfs:s0 ueventd.rc
lrwxrwxrwx root     root          u:object_r:rootfs:s0 vendor -> /system/vendor
```



# Android安全机制——TPM

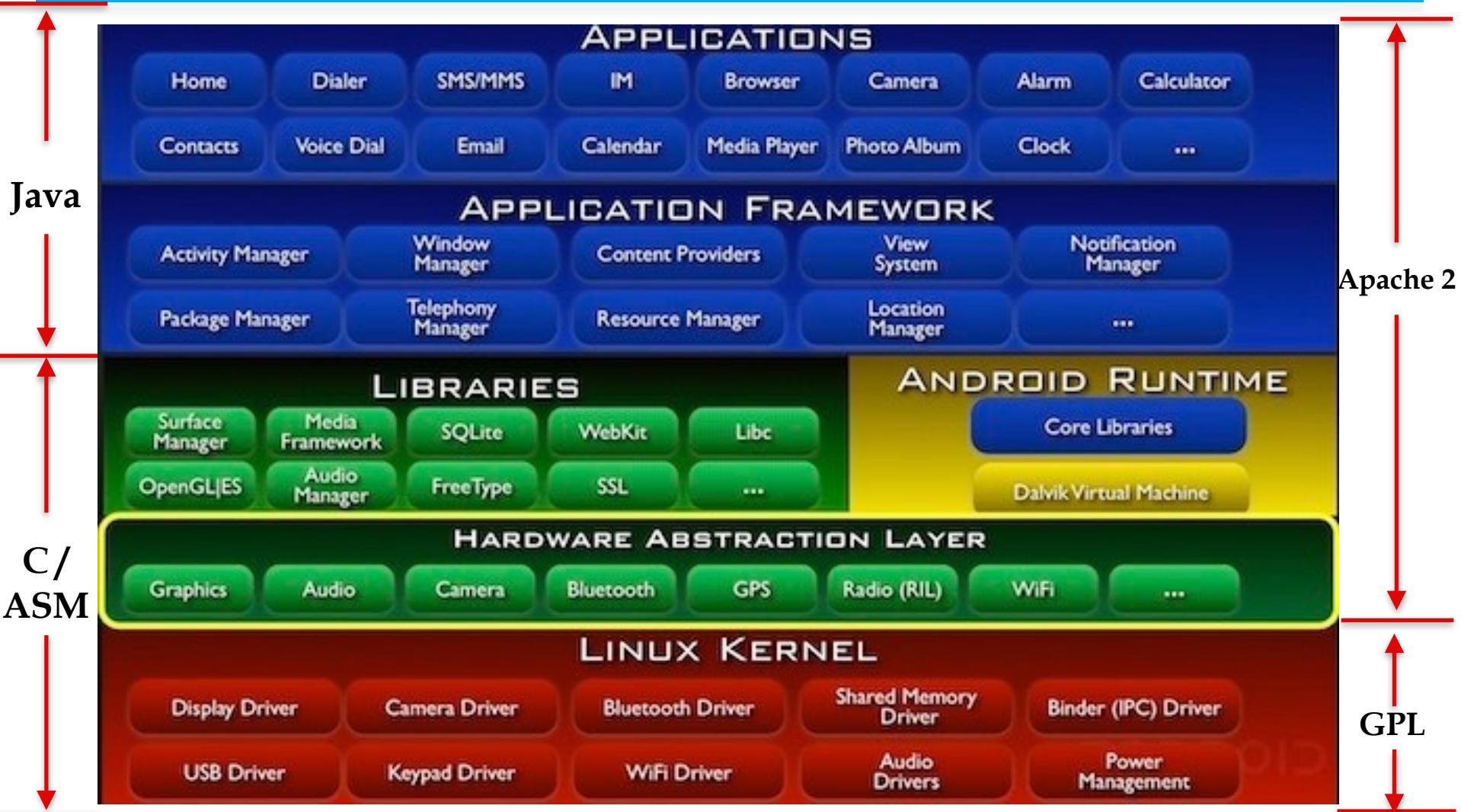
- 已有学术研究成果，工业界暂时未有完整解决方案
  - 学术成果：基于软件方式实现的TPM
  - 工业界：基于CPU的部分运行时完整性保护机制
- CPU
  - 特权指令
  - 安全引导
  - 密码学（专用芯片）硬件加速(ARM v8)
  - 可信区域（TrustZone）(ARM CPU部分型号支持)



# 总结：ANDROID VS. LINUX

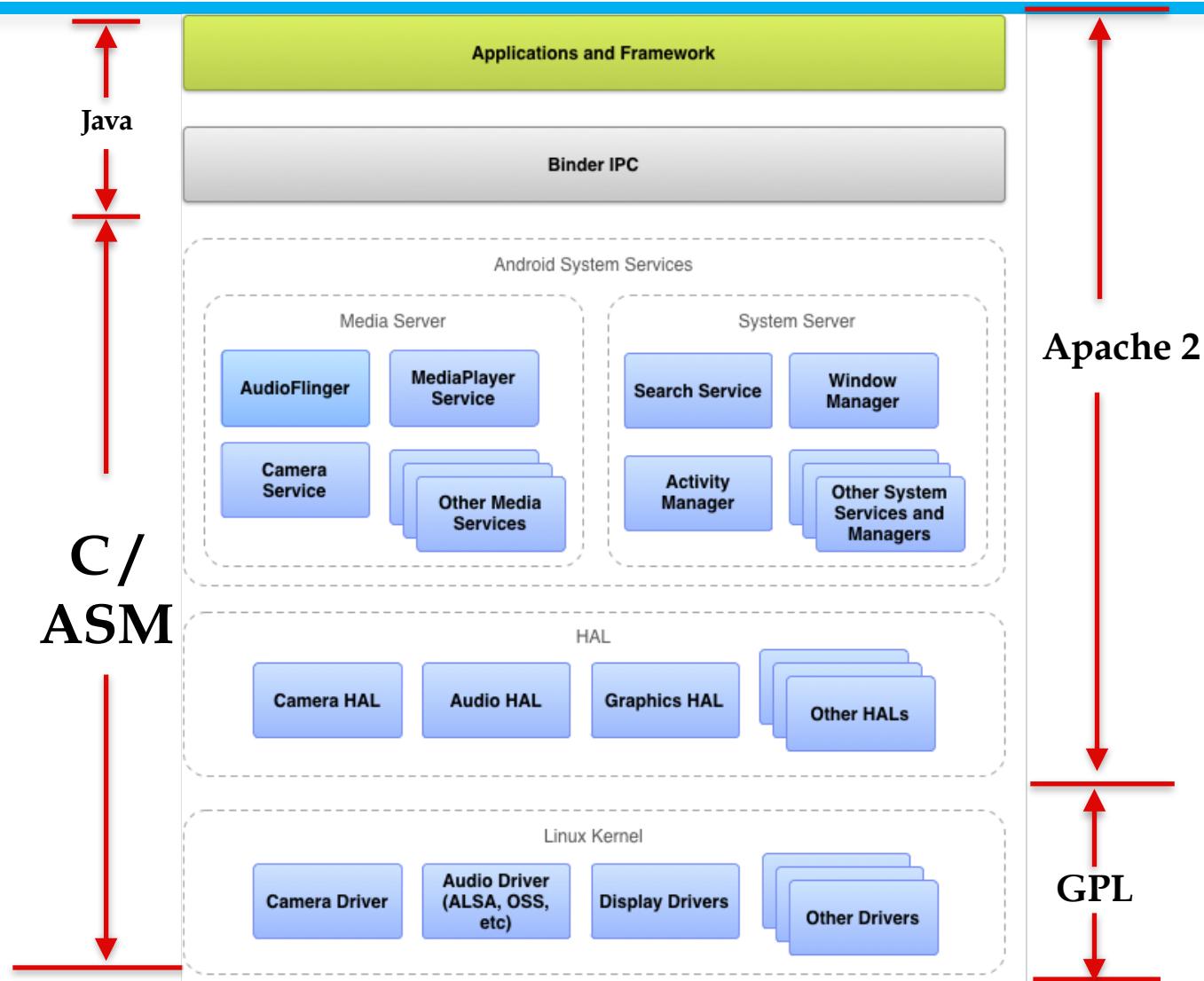


# 重温Android系统组件架构





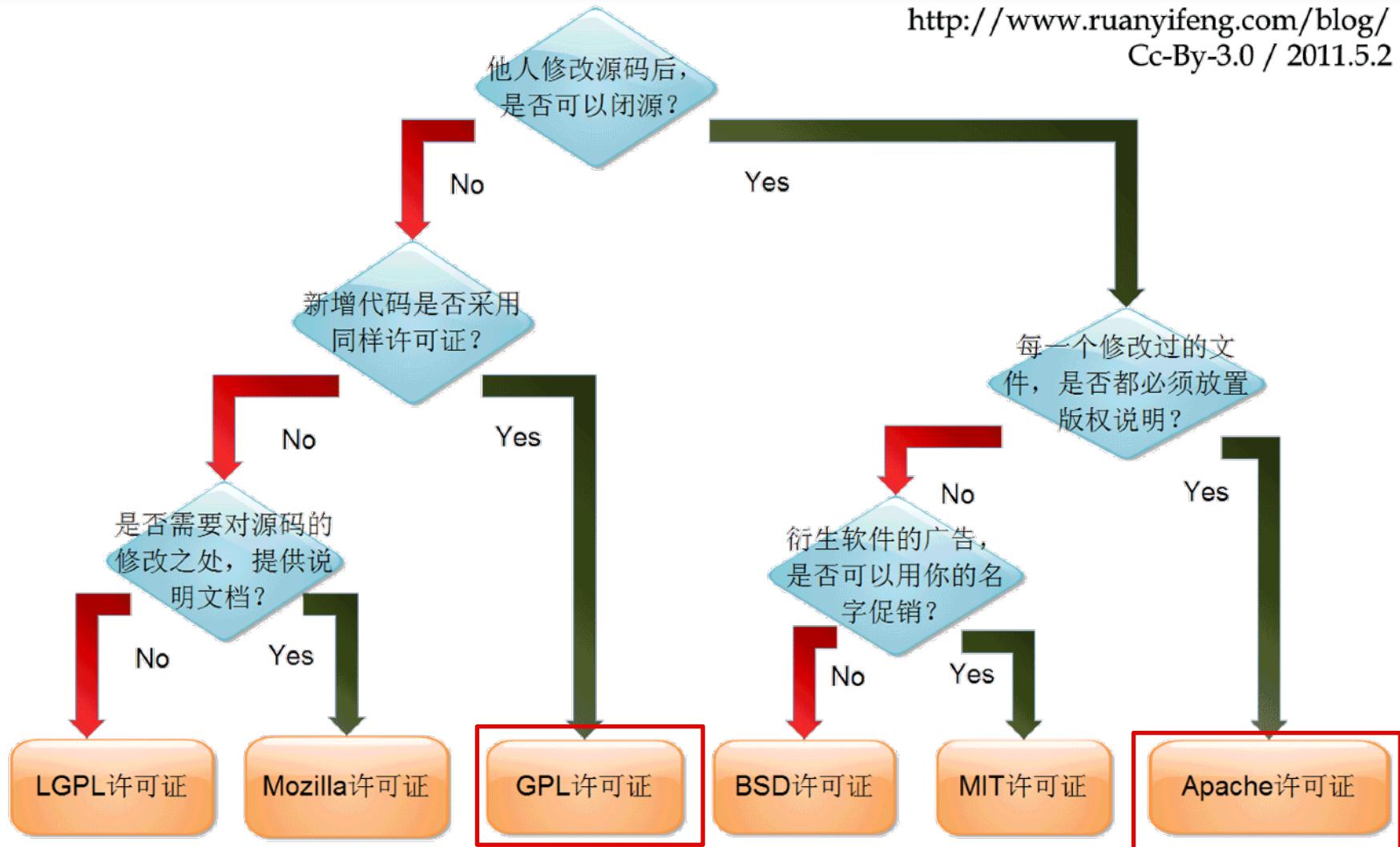
# 重温Android系统组件架构





# 一张图看懂关于开源授权协议差异

[http://www.ruanyifeng.com/blog/  
Cc-By-3.0 / 2011.5.2](http://www.ruanyifeng.com/blog/Cc-By-3.0/)



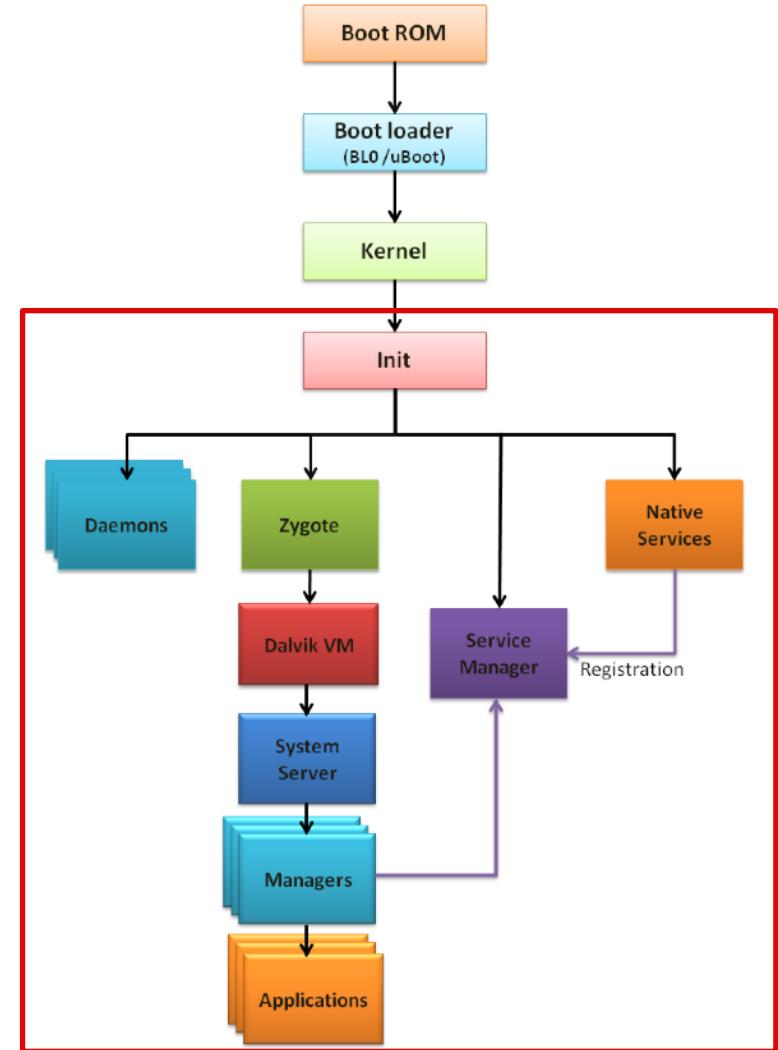
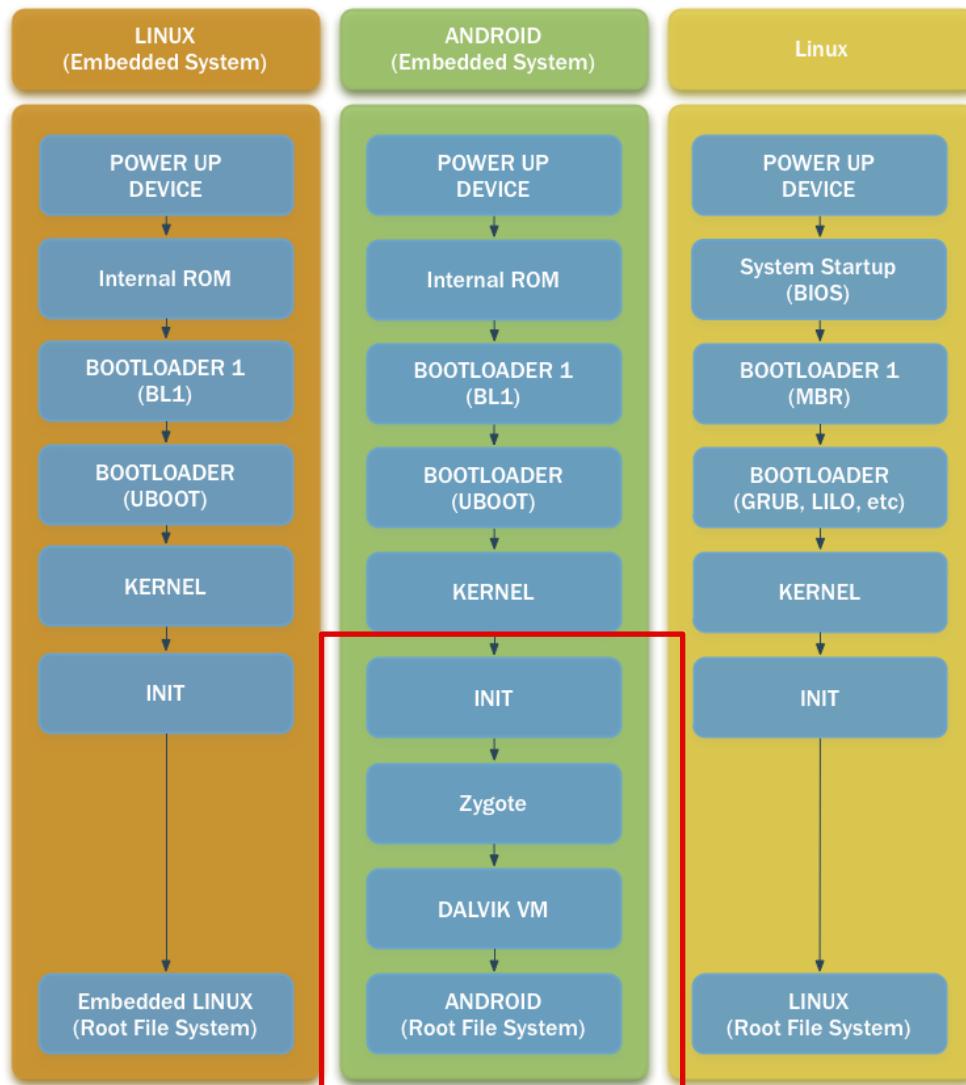


# 重温：Android系统组件架构与生态圈

- 基于已有第三方开源成果
  - Linux 内核
  - libc / WebKit / OpenSSL / Free Type / SGL: 2D Graphics / SQLite / Surface Manager / Open GL|ES / Media Framework / ...
- AOSP
- Google私有代码
- OEM私有代码
- 第三方APP私有代码



# 系统引导流程——Android VS. Linux





# bootloader

- 不同设备厂商各自独立实现
  - 加锁 vs. 不加锁
    - 设备厂商防止用户破解（逆向分析）设备和出厂原生操作系统及内置应用程序
  - （待写入）系统镜像强制签名 vs. 非强制
    - 设备厂商防止用户刷（装）入第三方系统



## 内容提纲

- Android安全机制概览
- **Android应用软件基础**
- Android应用软件的安全策略与机制
- 典型安全漏洞原理与实验



# Android应用软件基础

- Android开发基本流程
- APK静态结构
  - AndroidManifest.xml
- APK签名与分发（安装）
- 应用生命周期管理



# 创建Android项目

- 安装 Android Studio
  - 依赖软件安装: JDK
- 创建本课程的第一个Android应用程序
- 运行应用
- 构建简单用户界面
- 启动另一个Activity



# Android App的源代码基本组成

- AndroidManifest.xml
- src
- res
- libs
- gen
- bin

动手看一看

IDE创建的项目源代码  
VS.  
SDK创建的项目源代码

```
AndroidManifest.xml
ant.properties
bin
build.xml
gen
libs
local.properties
proguard-project.txt
project.properties
res
src
```



# AndroidManifest.xml

```
3. vim AndroidManifest.xml (Python)
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="edu.cuc.cs"
4     android:versionCode="1"
5     android:versionName="1.0">
6     <application android:label="@string/app_name" android:icon="@drawable/
ic_launcher">
7         <activity android:name="MainActivity"
8             android:label="@string/app_name">
9             <intent-filter>
10                 <action android:name="android.intent.action.MAIN" />
11                 <category android:name="android.intent.category.LAUNCHER" />
12             </intent-filter>
13         </activity>
14     </application>
15 </manifest>
16
```

NORMAL >> AndroidManifest.xml      xml << 100% : 16: 0



# AndroidManifest.xml

- Android项目的清单文件，包含如下信息
  - 应用程序包名，该包名将会作为应用的**唯一标识**
  - 应用程序所包含的组件，如：Activity、Service、BroadcastReceiver和ContentProvider等
  - 应用兼容的最低版本
  - 应用所需使用的系统权限声明
    - 当前应用作为主体，访问系统资源或其他应用程序
    - 部分硬件设备的访问有专门的权限名称，例如GPS
  - 其他程序访问该程序所需的权限声明
    - 其他应用作为主体，访问当前应用的特定功能模块所需要使用的权限声明关键字



# AndroidManifest.xml 自定义权限

```
<permission android:description="string resource"
            android:icon="drawable resource"
            android:label="string resource"
            android:name="string"
            android:permissionGroup="string"
            android:protectionLevel=[ "normal" | "dangerous" |
                "signature" | "signatureOrSystem" ] />
```

- android:label: 权限名字，显示给用户的，值可是一个 string 数据，例如“自定义权限”。
- android:description: 比 label 更长的对权限的描述。值是通过 resource 文件中获取的，不能直接写 string 值，例如”@string/test”
- android:name: 权限名字，如果其他 app 引用该权限需要填写这个名字。
- android:protectionLevel: 权限级别，分为 4 个级别：
  - normal: 低风险权限，在安装的时候，系统会自动授予权限给 application
  - dangerous: 高风险权限，系统不会自动授予权限给 app，在用到的时候，会给用户提示
  - signature: 签名权限，在其他 app 引用声明的权限的时候，需要保证两个 app 的签名一致。这样系统就会自动授予权限给第三方 app，而不提示给用户
  - signatureOrSystem: 这个权限是引用该权限的 app 需要有和系统同样的签名才能授予的权限，一般不推荐使用



## res 目录

- 存储Android应用所用的全部资源，包括图片资源、字符串资源、颜色资源、尺寸资源等
  - Android按照约定，将不同类型的资源放在不同的文件夹内，AAPT工具通过扫描这些资源，自动生成资源清单类：R.java



# R.java

```
1 /* AUTO-GENERATED FILE.  DO NOT MODIFY.
2 *
3 * This class was automatically generated by the
4 * aapt tool from the resource data it found.  It
5 * should not be modified by hand.
6 */
7
8 package edu.cuc.cs;
9
10 public final class R {
11     public static final class attr {
12     }
13     public static final class drawable {
14         public static final int ic_launcher=0x7f020000;
15     }
16     public static final class layout {
17         public static final int main=0x7f030000;
18     }
19     public static final class string {
20         public static final int app_name=0x7f040000;
21     }
22 }
```



# APK详细制作过程

- 建议通过Android SDK 创建一个Android App 工程，通过阅读Ant的 build.xml 了解Android APP的详细制作过程
  - debug模式发布
  - release模式发布

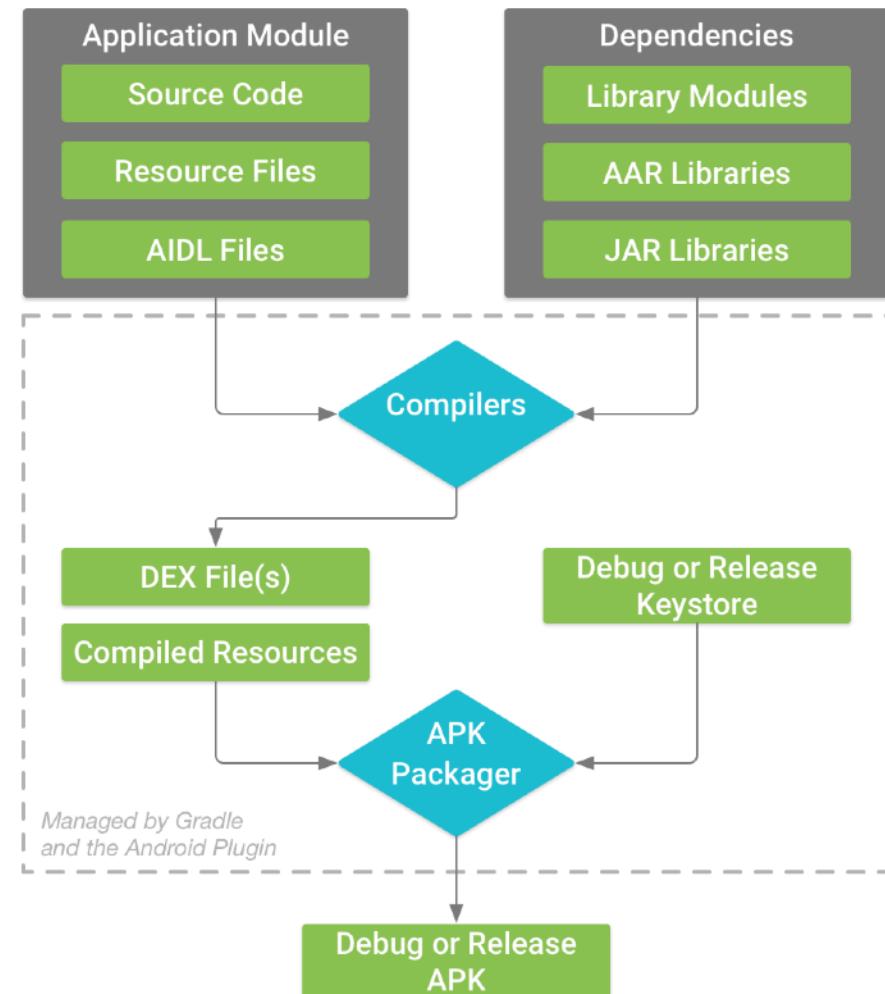


Figure 1. The build process of a typical Android app module.



# Android APP基本组件

- Activity
  - Intent和IntentFilter
- Service
- BroadcastReceiver
- ContentProvider



# Activity和View

- Activity是Android应用中负责与用户交互的组件
- View是所有UI控件、容器控件的基类
  - 用户看到的部分
  - View组件需要被放到容器组件中或使用Activity将它显示出来
  - setContentView()

```
1 package edu.cuc.cs;
2
3 import android.app.Activity;
4 import android.os.Bundle;
5
6 public class MainActivity extends Activity
7 {
8     /** Called when the activity is first created. */
9     @Override
10    public void onCreate(Bundle savedInstanceState)
11    {
12        super.onCreate(savedInstanceState);
13        setContentView(R.layout.main);
14    }
15 }
```



# Service

---

- 与Activity地位并列
- 后台运行，一般不需要与用户交互，无图形用户界面
- 所有的Service组件需要继承Service基类
- 一个Service组件被运行起来之后，拥有自己独立的生命周期
  - 为其他组件提供后台服务或监控其他组件运行状态



# BroadcastReceiver

- 广播消息接收器
  - 类似于事件编程中的事件监听器
    - 普通事件监听器监听的事件源是程序中的对象；BroadcastReceiver监听的事件源是Android应用中的其他组件
- 开发者实现自己的BroadcastReceiver子类，重写onReceive(Context cx, Intent it)方法即可
- 其他组件通过sendBroadcast()、sendStickyBroadcast()或sendOrderedBroadcast()方法发送广播消息时，如该BroadcastReceiver也对该消息『感兴趣』，onReceive方法会被自动调用
  - 配置IntentFilter实现『感兴趣』
  - 在Java代码中使用Context.registerReceiver()方法注册BroadcastReceiver
  - 在AndroidManifest.xml中使用<receiver ... />元素完成注册



# AndroidManifest.xml 中的 BroadcastReceiver

```
367      <!-- 广播接收者：开启启动电话监听器 -->
368      <receiver android:name=".receiver.BootPhoneListener" >
369          <intent-filter>
370              <action android:name="android.intent.action.BOOT_COMPLETED" />
371              <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
372              <action android:name="android.intent.action.USER_PRESENT" />
373          </intent-filter>
374      </receiver>
```



# ContentProvider

- Android OS限制每个应用运行在自己独立的 Dalvik虚拟机实例中
  - 应用间需要实时数据交换：ContentProvider
- 开发者实现自己的ContentProvider需要实现以下抽象方法
  - insert、delete、update和query
    - 上述方法的第一个参数都是Uri
- 应用A通过ContentProvider暴露自己的数据访问接口，应用B通过ContentResolver来访问数据



# Intent和IntentFilter

- Android应用内不同组件之间通信的载体
  - 可以用于启动一个Activity或Service组件
  - 还可以发送一条广播消息来触发系统中所有已注册的 BroadcastReceiver
- Intent封装了当前组件需要启动或触发的目标组件信息
  - 显式Intent：明确指定需要启动或触发的组件类名
  - 隐式Intent：仅指定需要启动或触发的组件应满足的条件
    - IntentFilter声明当前应用能处理哪些隐式Intent



# 显式Intent VS. 隐式Intent

```
3 // 显式Intent - 1
4 Intent intent = new Intent();
5 intent.setClassName("com.samples.intent.simple" ,
6                 "com.samples.intent.simple.TestActivity" );
7 startActivity(intent);
8 // 显式Intent - 2
9 Intent intent = new Intent(A.activity,B.class);
10 startActivity(intent);
```

```
13 // 隐式Intent - 1
14 Intent intent = new Intent(Intent.ACTION_DIAL );
15 startActivity(intent);
16 // 隐式Intent - 2
17 Intent intent = new Intent("com.test.broadcast");
18 intent.putString("PASSWORD","123456");
19 sendBroadcast(intent);
20 // 隐式Intent - 3
21 Intent intent = new Intent("com.test.service");
22 intent.putString("USERNAME","test");
23 startService(intent);
```



# AndroidManifest.xml 中的IntentFilter

```
275 <activity  
276     android:name="com.hlx.lite.activity.SearchResultActivity"  
277     android:label="@string/searchresult" >  
278     <intent-filter>  
279         <action android:name="Result_SearchActivity" />  
280  
281         <category android:name="android.intent.category.DEFAULT" />  
282     </intent-filter>  
283 </activity>
```

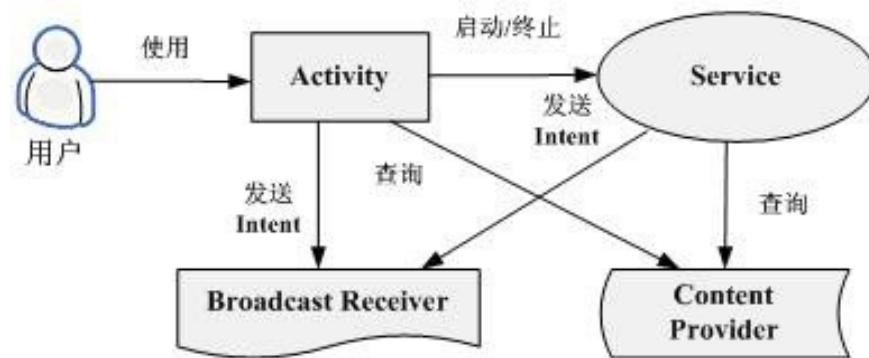
```
84 <activity  
85     android:name="com.hlx.lite.activity.MainActivity"  
86     android:label="@string/app_name"  
87     android:screenOrientation="portrait" >  
88  
89     <!--  
90     <intent-filter android:label="@string/title_call_contact" > 联系人详情打电话中用到  
91         <action android:name="android.intent.action.CALL_PRIVILEGED" />  
92         <action android:name="android.intent.action.CALL" />  
93         <category android:name="android.intent.category.DEFAULT" />  
94         <data android:scheme="tel" />  
95         <data android:scheme="callto" />  
96     </intent-filter>  
97     -->  
98     <intent-filter>  
99         <action android:name="android.intent.action.DIAL" />  
100  
101         <category android:name="android.intent.category.DEFAULT" />  
102     </intent-filter>  
103     <intent-filter>  
104         <action android:name="android.intent.action.VIEW" />  
105  
106         <category android:name="android.intent.category.DEFAULT" />  
107         <category android:name="android.intent.category.BROWSABLE" />  
108  
109         <data android:mimeType="vnd.android.cursor.dir/calls" />  
110     </intent-filter>  
111 </activity>
```

『捕获』  
自定义  
Intent

『捕获』 系统Intent



# Android各组件关系图

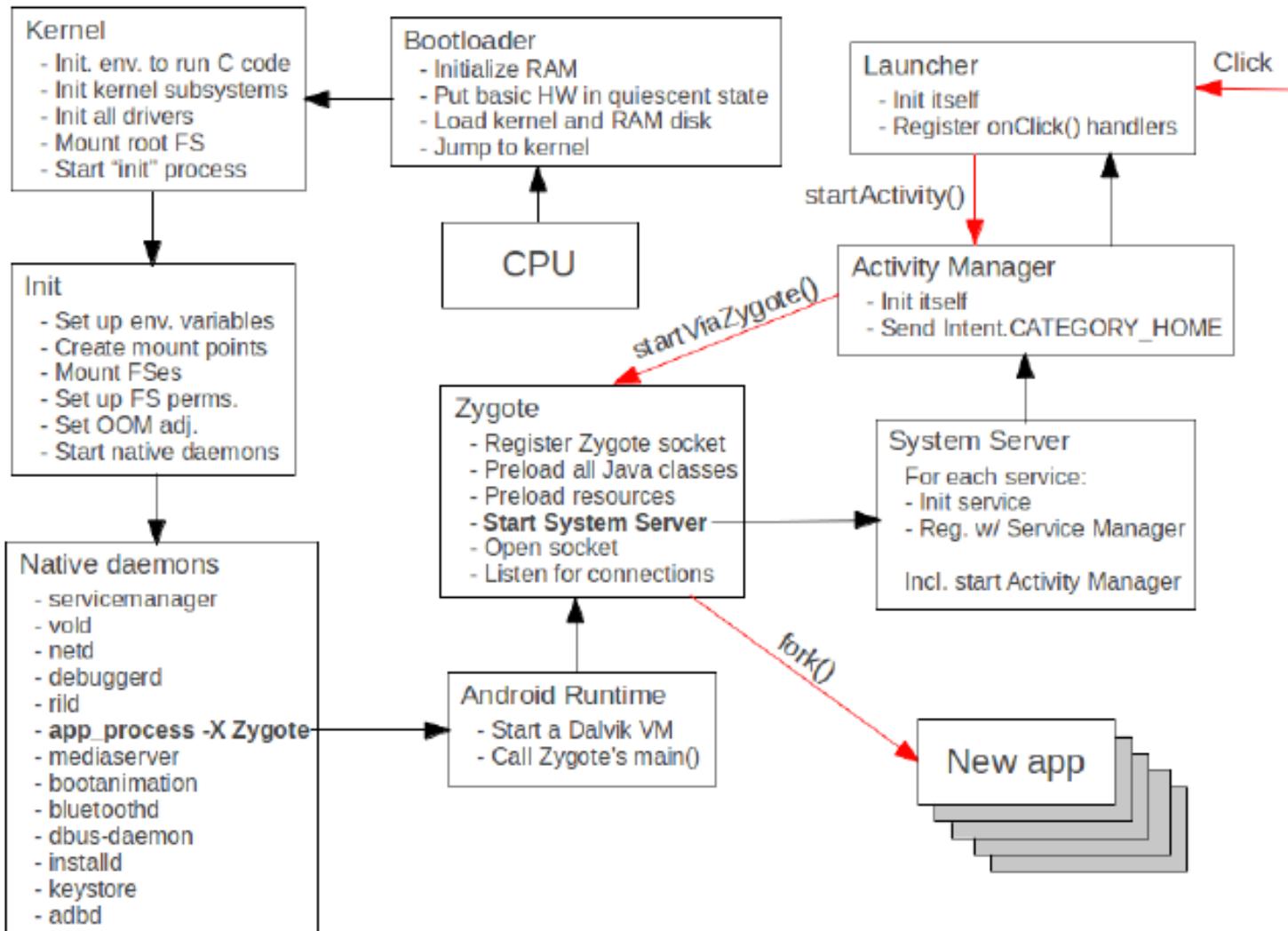


- 组件间需要通信需要在AndroidManifest.xml文件中『暴露』组件

组件名称	方法名称
<i>Activity</i>	<i>startActivity()</i> <i>startActivityForResult()</i>
<i>Service</i>	<i>startService()</i> <i>bindService()</i>
<i>Broadcast Receiver</i>	<i>sendBroadcast()</i> <i>sendOrderedBroadcast()</i> <i>sendStickyBroadcast()</i>

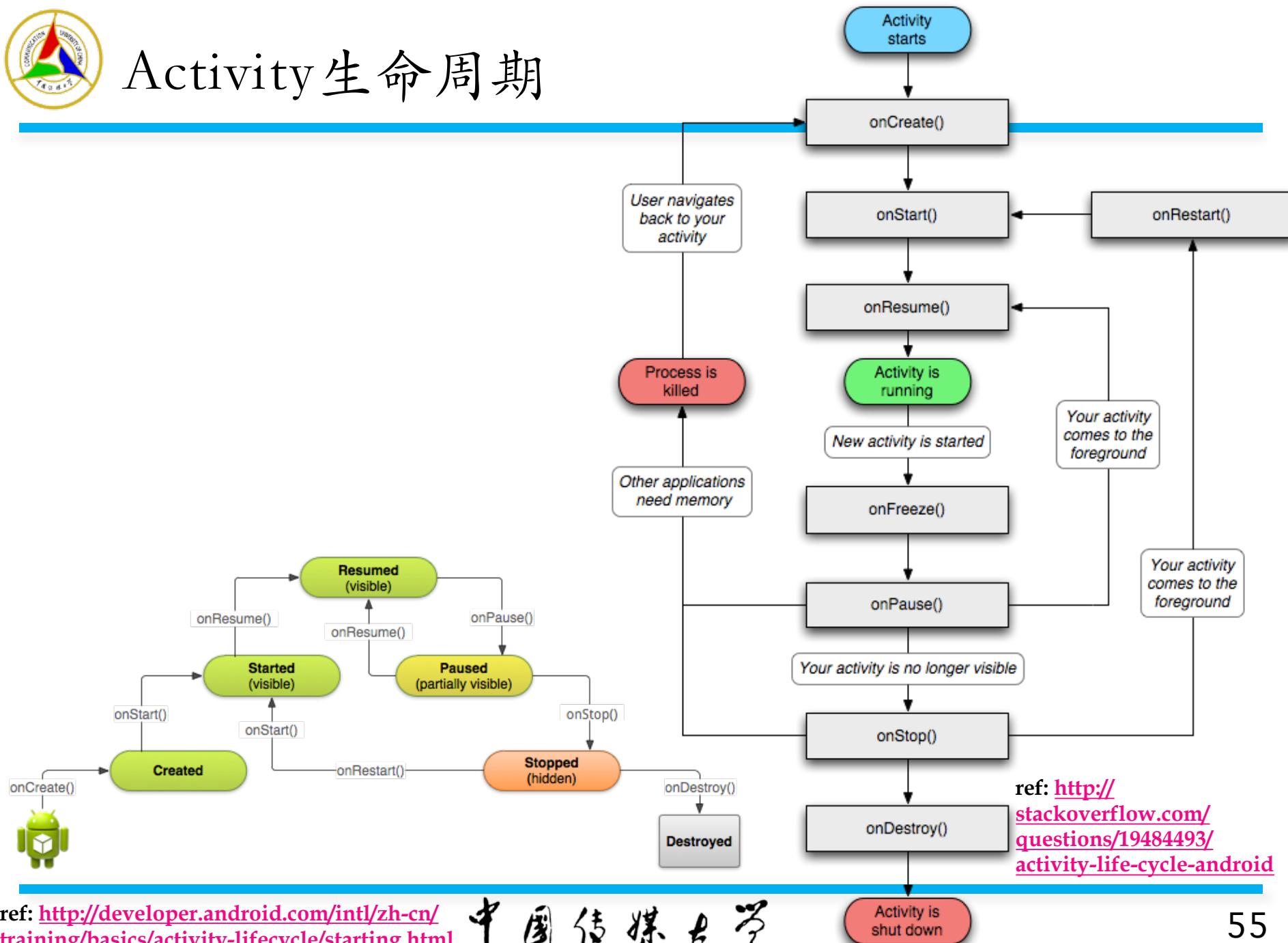


# App启动流程



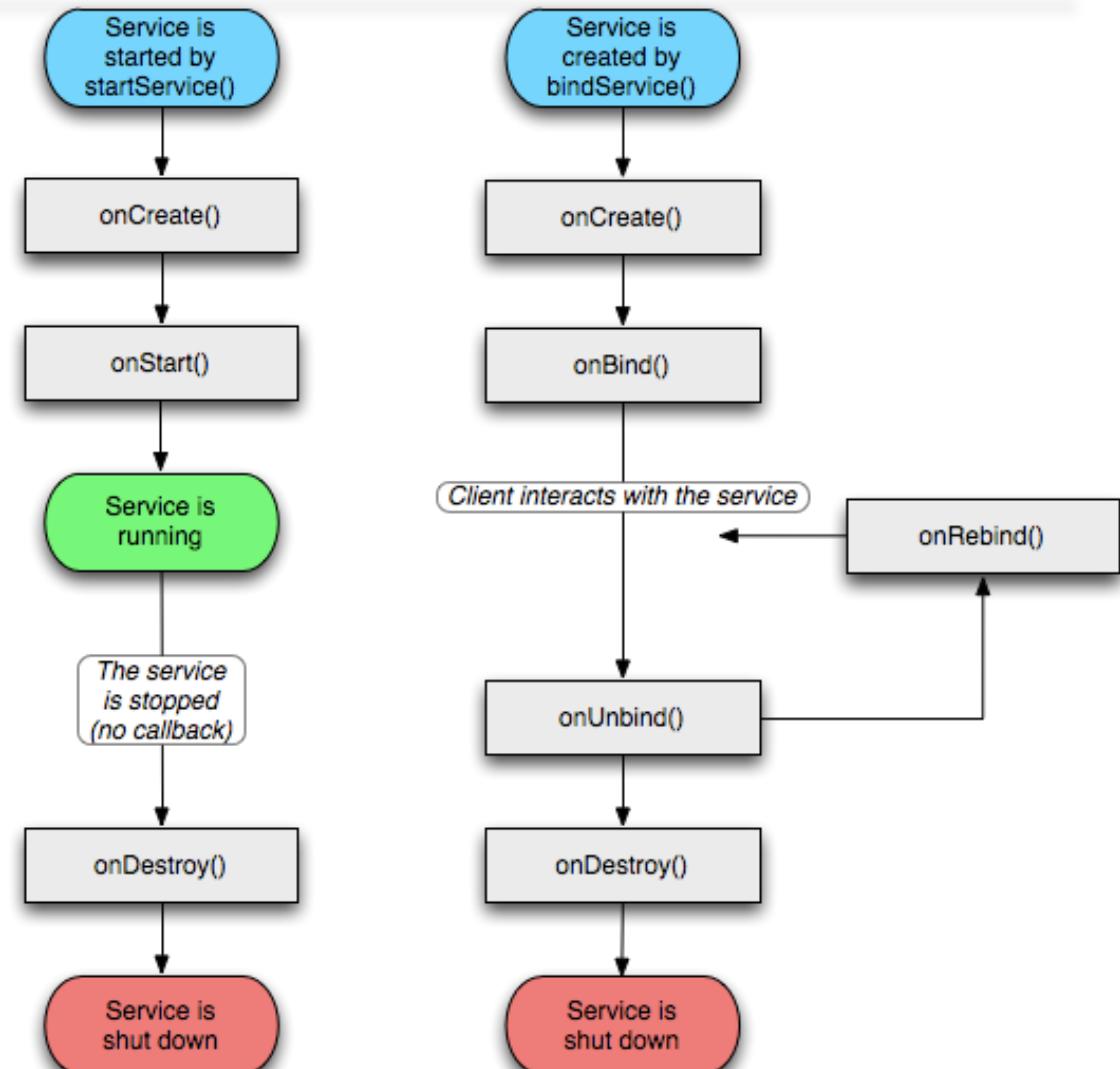


# Activity生命周期





# Service生命周期





# APK签名与验证

- 签名的方法与意义
- 生成的APK解压缩后目录及文件内容分析
  - 第7章詳解
- 验证签名



# 实验：对APK进行签名

- 证书和密钥库
- 调试构建时IDE的自动签名过程
  - 用于针对调试签署 APK 的自签署证书的有效期为 365 天，从其创建日期算起
    - openssl pkcs12 -in ~/.android/debug.p12 -info
    - keytool -list -keystore ~/.android/debug.keystore -v  
提示输入查看密码时输入： android
- 签署您的发布构建
- 从您的构建文件中移除签署信息（团队开发或开源时必读最佳安全实践）



## 验证签名

- 程序自检测
  - 下载组件完整性检测
- 可信赖第三方检测
  - 云端证书指纹检测
- 系统限定安装
  - 防止恶意同包名程序覆盖安装合法应用



# 安装应用

- 系统应用安装——开机时完成，没有安装界面
- 网络下载应用安装——通过Google Play完成，没有安装界面
- ADB工具安装——没有安装界面
- 第三方应用安装——通过SD卡里的APK文件安装，有安装界面，由PackageInstaller.apk应用处理安装及卸载过程的界面
  - 设置-安全-未知来源的设置影响的应用安装时检查
    - Android源码默认设置为禁止从未知来源（Google Play之外）安装应用  
Frameworks/base/packages/SettingsProvider/res/values/defaults.xml
    - 国行手机或定制ROM大多修改了该项设置，允许从第三方应用市场下载安装应用

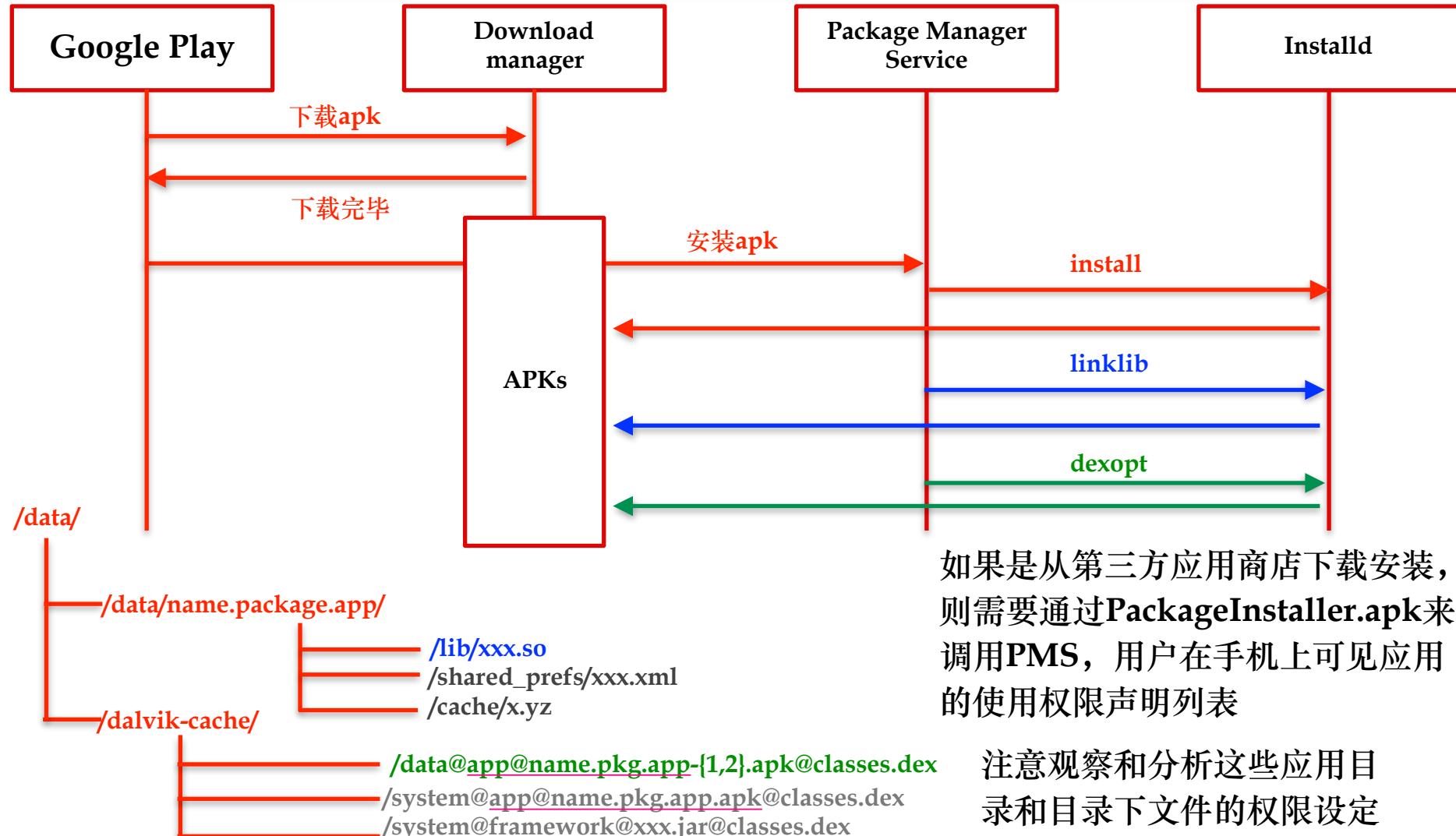


# Android应用在系统上的驻留踪迹

- 系统应用
  - /system/app/\*.apk
- 第三方应用（用户自行安装）
  - /data/app/\*.apk
- 数据目录
  - /data/data/com.xx.yy/\*
- 运行时缓存
  - /data/dalvik-cache



# 典型应用安装流程



如果是从第三方应用商店下载安装，则需要通过 **PackageInstaller.apk** 来调用 PMS，用户在手机上可见应用的使用权限声明列表

注意观察和分析这些应用目录和目录下文件的权限设定



# 第三方定制ROM（操作系统）的预装App

- odex文件与APK文件的关系
  - APK中的classes.dex文件通过dex优化过程将其优化生成一个dex文件单独存放，原APK中的classes.dex文件一般会被移除
  - 存储在/system/app/下
  - 节省文件存储占用空间，缩短系统启动时间
  - 极大增加了被重打包和篡改文件内容的难度
- deodex化
  - 同时保留APK中的classes.dex



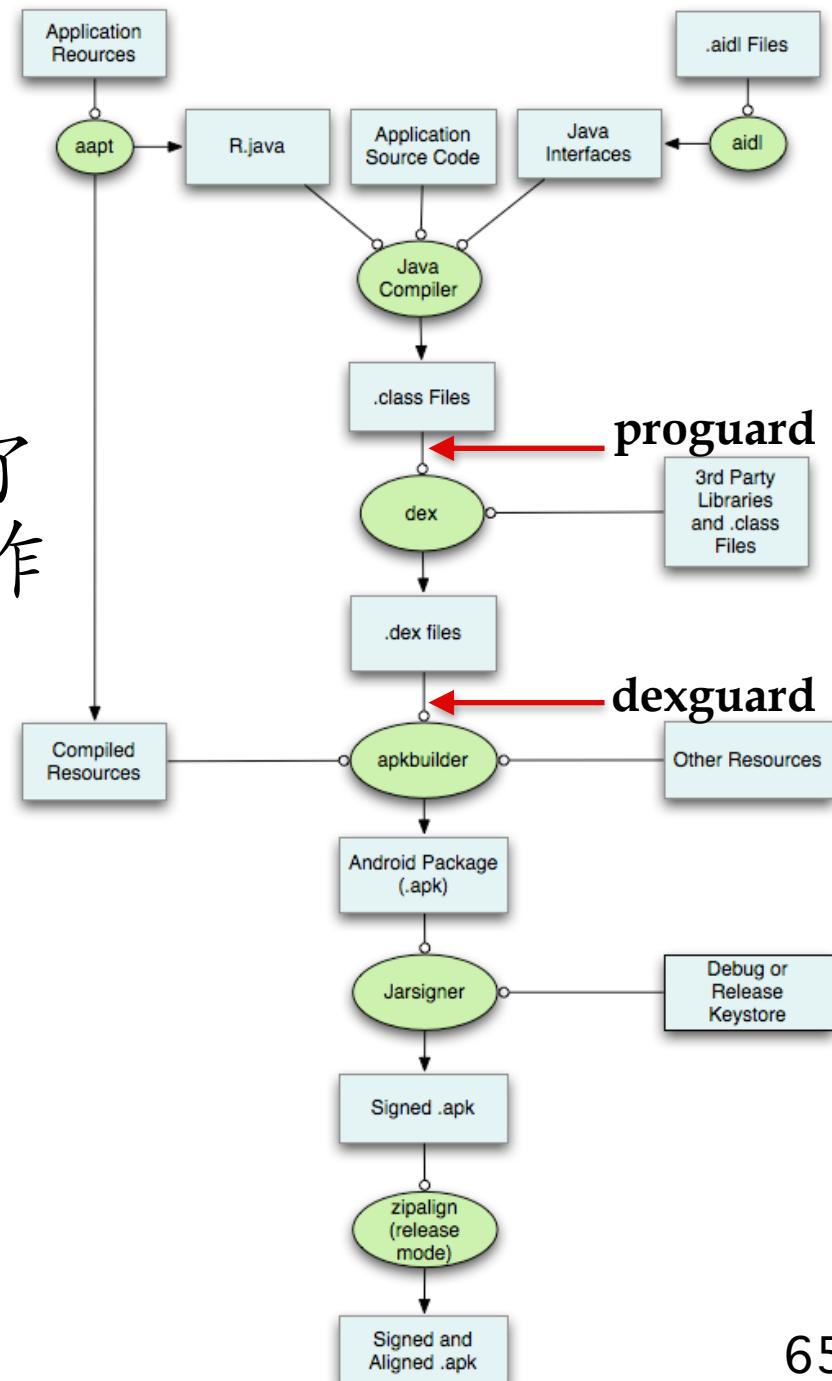
## 内容提纲

- Android安全机制概览
- Android应用软件基础
- Android应用软件的安全策略与机制
- 典型安全漏洞原理与实验



# APK详细制作过程

- 建议通过Android SDK创建一个Android App工程，通过阅读Ant的build.xml了解Android APP的详细制作过程
    - debug模式发布
    - release模式发布
    - 代码混淆
    - proguard





# 使用Ant编译发布Android项目过程解析

```
1163      <!-- This runs -package-release and -release-nosign first and then runs
1164          only if release-sign is true (set in -release-check,
1165          called by -release-no-sign)-->
1166      <target name="release"
1167          depends="-set-release-mode, -release-obfuscation-check, -package, -post-
  package, -release-prompt-for-password, -release-nosign, -release-sign, -post-build"
1168          description="Builds the application in release mode.">
1169      </target>
```

```
1082 <target name="-release-obfuscation-check">
1083     <echo level="info">proguard.config is ${proguard.config}</echo>
1084     <condition property="proguard.enabled" value="true" else="false">
1085         <and>
1086             <isset property="build.is.mode.release" />
1087             <isset property="proguard.config" />
1088         </and>
1089     </condition>
1090     <if condition="${proguard.enabled}">
1091         <then>
1092             <echo level="info">Proguard.config is enabled</echo>
1093             <!-- Secondary dx input (jar files) is empty since all the
1094                 jar files will be in the obfuscated jar -->
1095             <path id="out.dex.jar.input.ref" />
1096         </then>
1097     </if>
1098 </target>
```

```
59 <!-- Packages the application. -->
60 <target name="-package" depends="-dex, -package-resources">
61     <!-- only package apk if *not* a library project -->
62     <do-only-if-not-library elseText="Library project: do not package apk..." >
63         <if condition="${build.is.instrumented}">
64             <then>
65                 <package-helper>
66                     <extra-jars>
67                         <!-- Injected from external file -->
68                         <jarfile path="${emma.dir}/emma_device.jar" />
69                     </extra-jars>
70                 </package-helper>
71             </then>
72             <else>
73                 <package-helper />
74             </else>
75         </if>
76     </do-only-if-not-library>
77 </target>
```

```
890     <!-- Converts this project's .class files into .dex files -->
891     <target name="-dex" depends="-compile, -post-compile, -obfuscate">
892         <do-only-if-manifest-hasCode elseText="hasCode = false. Skipping...">
893             <!-- only convert to dalvik bytecode if *not* a library -->
894             <do-only-if-not-library elseText="Library project: do not convert bytecode...">
895                 <!-- special case for instrumented builds: need to use no-locals and need
896                     to pass in the emma jar. -->
897                 <if condition="${build.is.instrumented}">
898                     <then>
899                         <dex-helper nolocals="true">
900                             <external-libs>
901                                 <fileset file="${emma.dir}/emma_device.jar" />
902                             </external-libs>
903                         </dex-helper>
904                     </then>
905                     <else>
906                         <dex-helper />
907                     </else>
908                 </if>
909             </do-only-if-not-library>
910             </do-only-if-manifest-hasCode>
911     </target>
```

```
801     <!-- Obfuscate target
802     This is only active in release builds when proguard.config is defined
803     in default.properties.
804
805     To replace Proguard with a different obfuscation engine:
806     Override the following targets in your build.xml, before the call to <setup>
807         -release-obfuscation-check
808             Check whether obfuscation should happen, and put the result in a property.
809         -debug-obfuscation-check
810             Obfuscation should not happen. Set the same property to false.
811         -obfuscate
812             check if the property set in -debug/release-obfuscation-check is set to true.
813             If true:
814                 Perform obfuscation
815                 Set property out.dex.input.absolute.dir to be the output of the obfuscation
816     -->
817     <target name="-obfuscate">
818         <if condition="${proguard.enabled}">
819             <then>
820                 <property name="obfuscate.absolute.dir" location="${out.absolute.dir}/proguard" />
821                 <property name="preobfuscate.jar.file" value="${obfuscate.absolute.dir}/original.jar" />
822                 <property name="obfuscated.jar.file" value="${obfuscate.absolute.dir}/obfuscated.jar" />
```



# ProGuard——Java class代码混淆工具

- 免费Java class字节码压缩，优化，混淆，静态验证工具
  - 检测并删除无用类、成员、方法和属性
  - 优化Java字节码并删除无用指令
  - 使用简短无意义名称重命名类名、成员名和方法名（代码混淆）
- 压缩生成的二进制程序体积
- 增加二进制Java程序被逆向工程得到源代码和解读理解的难度



# DexGuard

- 商业Android平台Dalvik二进制字节码压缩，优化，混淆，静态验证工具
  - 加密字符串常量、类、原生库、资源文件等
  - 隐藏敏感API调用行为，改用运行时反射方法等价调用
  - 程序完整性检测与保护机制
  - 发布版代码自动清理所有日志行为代码
- 进一步增加二进制Android平台Java程序被逆向工程得到源代码和解读理解的难度



# APP制作过程的安全机制小结

- 使用ProGuard和DexGuard类解决方案防止已发布的二进制程序被逆向工程恢复源代码

- 增加攻击者通过源代码审计，发现漏洞的难度和破解工作量
- 开发者编写的某些类方法如果使用ProGuard混淆会导致程序运行出错，所以会配置禁止ProGuard对这些代码的混淆，这给了攻击者还原出这部分代码的便利

android proguard webview javascript interface

Web Shopping Images Videos News More Search tools

About 30,600 results (0.30 seconds)

[android - How to configure proguard for javascript interface ...](#)  
stackoverflow.com/.../how-to-configure-proguard-for-javascript-interfac... ▾  
Jul 13, 2013 - I have implemented a Webview which takes use of JavascriptInterface. It's working fine when not obfuscating, but at once Proguard is active, ...

[Proguard mess Javascript Interface functions when targeting ...](#)  
stackoverflow.com/.../proguard-mess-javascript-interface-functions-whe... ▾  
Jan 21, 2014 - I have a custom Webview in my android project as shown below: public class MyWebView extends WebView { public MyWebView(Context ...

[Android proguard Javascript Interface problem - Stack ...](#)  
stackoverflow.com/.../android-proguard-javascript-interface-problem ▾  
Mar 18, 2011 - The class names from that original thread are specific to that users Java classes, and not generic to all javascript interfaces. The javascript interface ...

[Android Proguard Javascript Interface Fail - Stack Overflow](#)  
stackoverflow.com/questions/.../android-proguard-javascript-interface-fa... ▾  
Jun 7, 2011 - http://lexandera.com/2009/01/extracting-html-from-a-webview/ ... according to this this answer Android proguard Javascript Interface problem.

[Proguard stops Javascript in WebView from working - Stack ...](#)  
stackoverflow.com/.../proguard-stops-javascript-in-webview-from-worki... ▾  
Jul 23, 2013 - If your Javascript interface methods are annotated with ... Browse other questions tagged android-webview proguard or ask your own question.

[android proguard webview javascript interface - CSDN blog](#)  
blog.csdn.net/wsyjx22/article/details/19481419 ▾ Translate this page  
Feb 19, 2014 - 项目中使用WebView + Javascript 相互使用时，在签名打包后发现，js无效，问题就是proguard造成的。解决方法如下：-keepclassmembers class ...

[Android经验: proguard 阻碍webView 正常工作 - CSDN blog](#)  
blog.csdn.net/span76/article/details/9065941 ▾ Translate this page  
Jump to [Android 与 JavaScript 的交互](#) - Calls into the javascript interface for the activity -->; <a onClick="window.demo.clickOnAndroid()"></div ...



# APP制作过程的安全机制小结

- 使用AndroidManifest.xml 声明程序运行所需的操作权限
  - 用户如果同意安装应用，剩下的访问控制机制实现完全交给操作系统来保证监督
  - 问题一：普通用户会在安装应用时阅读权限声明清单？
  - 问题二：普通用户能看懂这些晦涩专业的权限术语吗？
  - 零『敏感』权限声明的App也可以实现高危操作
  - 例如：借助第三方高权限应用的访问控制机制缺陷执行高危操作

视频操作演示：利用Google Voice认证缺陷执行高危操作



# APP制作过程的安全机制小结

- AndroidManifest.xml 中的其他安全相关静态特征
  - 请求访问哪些硬件设备，例如：GPS
  - App组件信息，例如：已知恶意软件组件名称
  - filtered intents扫描，例如：恶意软件通过注册 BOOT\_COMPLETED 消息实现开机自启动
- 反汇编APK和DEX文件
  - 危险API调用和敏感权限相关API调用
  - 硬编码的关键字特征，例如：域名、IP等



# APP制作过程的安全机制小结

- 静态分析方法面临的挑战

- 代码混淆、变形、加密
- 动态代码加载技术

- 动态分析方法面临的挑战

- 运行环境检测
  - 对抗虚拟运行分析
- 延迟加载
  - 对抗检测逻辑缺陷
    - Google Bouncer 沙盒检测绕过



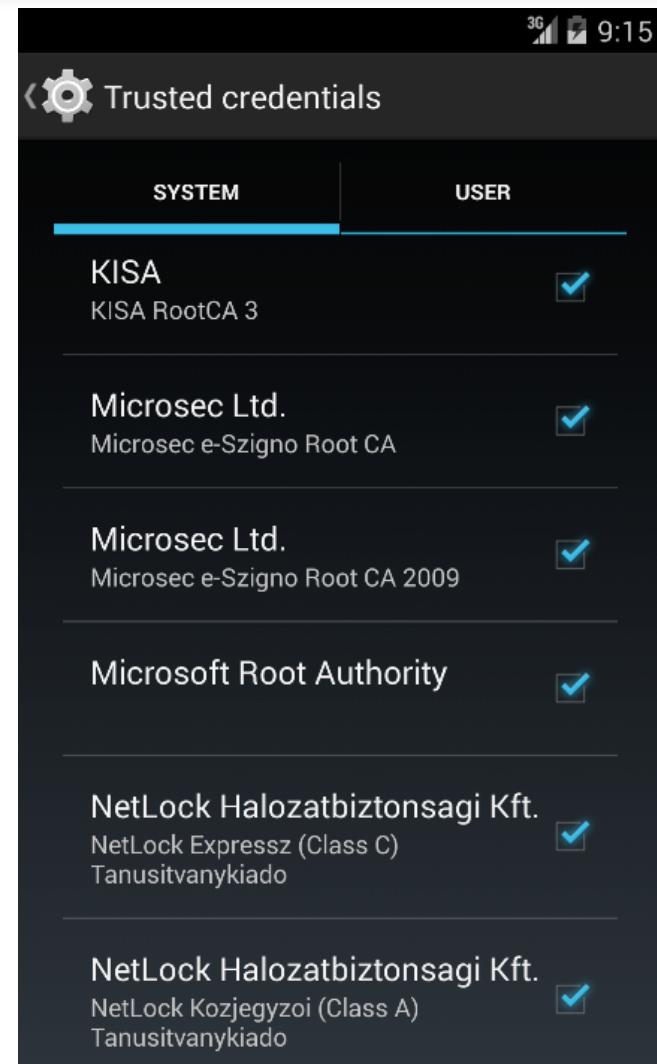
# 重温零『敏感』权限应用的高危操作

- 当某apk权限请求与权限声明不对等的时候，那么恶意apk在没有对应<uses-permission>请求下，可以通过访问用户安装其他合法apk的Activity、Broadcast、Service方式来突破
  - 恶意APK构造Intent访问高权限APK暴露的访问接口
- 拥有敏感数据的APK通过ContentProvider暴露敏感数据给任意APP
- SQL注入Android应用本地数据库SQLite
- . . .



# 系统内置CA证书

- /system/etc/security/carets
  - 查看证书详情
    - openssl x509 -in xxx -noout -text
  - 网络安全的保障





# 『root』原理

- 利用Android系统的本地提权漏洞获得root权限
- 获得root权限后重新以可写模式挂载/system
- 将root权限管理应用Superuser.apk写入/system/app，将su写入/system/xbin/和/system/bin/
- Superuser.apk负责对普通用户的root权限请求进行管理



# 『root』 利弊

- 利
  - 用户对自己的手机拥有完全控制权
- 弊
  - 系统不稳定
    - 提权漏洞利用效果的平台差异性通常较大
  - 恶意代码的攻击手段可以更丰富
    - 静默安装、静默卸载、静默访问任意数据等
  - 如果安装的Superuser.apk本身已经被恶意替换，后果不堪设想



# 组件安全

	攻击手段
<i>Activity</i>	* 构造Intent直接调用，实现非授权访问 * 后台守护进程通过进程枚举，直接启动新Activity覆盖到当前Activity进行『点击』劫持，实现钓鱼攻击（零权限要求）
<i>Service</i>	* 构造Intent直接调用，实现非授权访问
<i>Broadcast Receiver</i>	* 构造Intent直接发送虚假广播消息，实现非授权访问 * 注册同名IntentFilter，实现广播消息监听和劫持
<i>Content Provider</i>	* 直接访问暴露的URI，实现非授权访问



## 组件安全——危害

- 恶意调用Activity
- 恶意接收数据（监听/截获敏感数据，例如无序广播）
- 仿冒应用，例如（恶意钓鱼，启动登陆界面）
- 调用组件并接受组件返回数据
- 拦截（有序）广播
- 非授权访问或恶意篡改数据（针对Content Provider接口的SQL注入）



# 实验

中国传媒大学



# 组件安全加固

- 最小化组件暴露
- 设置组件访问权限
- 暴露组件对应代码的内部运行时检查



# 组件安全加固——最小化组件暴露

- 不参与跨应用调用的组件添加`android:exported="false"`属性

```
9084 <receiver
9085     android:name="com.alipay.android.app.LiveConnectReceiver"
9086     android:exported="false"
9087     >
9088     <intent-filter
9089         >
9090         <action
9091             android:name="com.alipay.android.app.pay.ACTION_CREATE_LIVE_CONNECT"
9092             >
9093         </action>
9094     </intent-filter>
9095 </receiver>
```

```
40 <service
41     android:name="com.taobao.tao.pay.PayService"
42     android:exported="false"
43     >
44 </service>
45 <service
46     android:name="com.taobao.cache.service.ChocolateCacheService"
47     android:exported="false"
48     >
49     <intent-filter
50         >
51         <action
52             android:name="com.taobao.cache.IMultiCacheService"
53             >
54         </action>
55     </intent-filter>
56 </service>
```

```
        <activity
            android:theme="@7F0F0005"
            android:name="com.taobao.open.oauth.OauthActivity"
            android:exported="false"
            android:launchMode="1"
            android:screenOrientation="1"
            android:configChanges="0x000000A0"
            >
        </activity>
```



# 组件安全加固——设置组件访问权限

## 参与跨应用调用的组件或者公开的广播、服务设置权限

- (1) 组件添加android:permission属性
- (2) 声明<permission>属性
- (3) 调用组件者声明<uses-permission>

```
<permission
    android:name="com.tencent.qqhead.permission.getheadresp"
    android:protectionLevel="0x00000002"
    >
</permission>
<uses-permission
    android:name="com.tencent.qqhead.permission.getheadresp"
    >
</uses-permission>

public static final int FLAG_COSTS MONEY
Flag for flags, corresponding to costsMoney value of permissionFlags.
Constant Value: 1 (0x00000001)

public static final int PROTECTION_DANGEROUS
Dangerous value for protectionLevel, corresponding to the dangerous value of protectionLevel.
Constant Value: 1 (0x00000001)

public static final int PROTECTION_NORMAL
A normal application value for protectionLevel, corresponding to the normal value of protectionLevel.
Constant Value: 0 (0x00000000)

public static final int PROTECTION_SIGNATURE
System-level value for protectionLevel, corresponding to the signature value of protectionLevel.
Constant Value: 2 (0x00000002)

public static final int PROTECTION_SIGNATURE_OR_SYSTEM
System-level value for protectionLevel, corresponding to the signatureOrSystem value of protectionLevel.
Constant Value: 3 (0x00000003)
```

```
<service
    android:name="com.taobao.android.sso.internal.AlipayAuthenticationService"
    android:permission="android.permission.ACCOUNT_MANAGER"
    android:enabled="false"
    android:exported="true"
    >
    <intent-filter
        >
        <action
            android:name="android.accounts.AccountAuthenticator"
            >
        </action>
    </intent-filter>
    <meta-data
        android:name="android.accounts.AccountAuthenticator"
        android:resource="@+id/AccountAuthenticator"
        >
    </meta-data>
    <meta-data
        android:name="android.accounts.AccountAuthenticator.customTokens"
        android:value="true"
        >
    </meta-data>
    <meta-data
        android:name="com.taobao.android.sso.Version"
        android:value="@+id/Version"
        >
    </meta-data>
</service>
```



# 暴露组件对应代码的内部运行时检查

Android 提供各种 API 来在运行时检查、执行、授予和撤销权限。这些 API 是 android.content.Context 类的一部分，这个类提供有关应用程序环境的全局信息。

```
3 if (context.checkSelfPermission("com.test.custempermission")  
4     != PackageManager.PERMISSION_GRANTED) {  
5         // The Application requires permission to access the  
6         // Internet");  
7 } else {  
8     // OK to access the Internet  
9 }
```



# 数据安全

- 存储
  - 内部存储
  - 外部存储
- 传输（通信）
- 边信道信息泄露



# 实验



# 外部存储安全

- AndroidManifest.xml中声明以下权限就可以读写外部存储设备上的任意数据

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE">
```

- 相关漏洞实例

- 小米MIUI系统造成用户大量敏感数据泄露
- 手机QQ 2012 (Android) 3.0可导致用户聊天信息泄露
- 印象笔记客户端设计缺陷可导致用户信息泄露
- 人人网Android客户端可能导致劫持或恶意软件安装
  - 从不可信位置安装软件未做文件真实性和完整性检查



# 内部存储安全

- 应用使用内部存储时，数据默认是只能被当前应用访问（基于文件访问权限设置）
- 一旦系统被root或代码编写错误，受保护数据将被其他程序访问到

```
try {  
    FileOutputStream fos = openFileOutput("config.txt", MODE_PRIVATE);  
    fos.write("hello world".getBytes());  
    fos.close();  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

```
root@generic_x86:/data/data/edu.cuc.cs/files # ll  
-rw-rw---- u0_a65 u0_a65 11 2014-12-07 23:42 config.txt  
root@generic_x86:/data/data/edu.cuc.cs/files # cat config.txt  
hello worldroot@generic_x86:/data/data/edu.cuc.cs/files #
```



# 传输（通信）安全

- 软件与软件
  - Intent通信机制安全
    - Broadcast Receiver
- 软件与网络（服务器）
  - SSL加密通信
  - （基于系统内置授权CA证书，） 检查服务端证书合法性
  - 检查服务器端证书配置（有效时间、CN等）



# 边信道信息泄露

- 日志
- 系统剪贴板信息
- URL缓存
- 浏览器Cookie对象
- 第三方统计数据



# WebView安全

- addJavascriptInterface API相关漏洞利用
  - CVE-2012-6636/CVE-2013-4710/CVE-2014-1939/CVE-2014-7224
  - 远程代码执行漏洞实例索引
  - 远程执行任意Java对象方法，可被用于网页挂马，进而实现系统级别的入侵事件
- Webkit内核漏洞利用UXSS
  - 绕过系统内置浏览器（Webkit内核）的同源性访问本地文件或非当前域的任意cookie等



# WebView安全

- 漏洞实例
  - 58同城app远程代码执行
  - 迅雷APP远程代码执行漏洞
- 漏洞检测
  - Android Webview挂马漏洞在线检测
  - addjsif漏洞在线检测
  - UXSS漏洞在线检测



# WebView安全加固

- addJavascriptInterface API相关漏洞修补
  - 如果无需与JS交互， 禁止调用addJavascriptInterface方法
  - 在载入页面时对URL进行白名单判定， 只有存在白名单中的域才允许导出或调用相关的Java类或方法
  - Android 4.2及以后版本， 使用新增的@JavascriptInterface 显式声明需要暴露给JS访问的Java对象方法



# WebView安全加固

- Webkit内核漏洞利用UXSS修补
  - 此问题属于android webkit的漏洞，请尽量使用最新版的android系统
  - 服务端禁止iframe嵌套X-FRAME-OPTIONS:DENY。详见：<http://drops.wooyun.org/papers/104>
  - 客户端使用setAllowFileAccess(false)方法禁止webview访问本地域。详见：setAllowFileAccess(boolean)
  - 客户端使用onPageStarted (WebView view, String url, Bitmap favicon)方法在跳转前进行跨域判断
  - 客户端对iframe object标签属性进行过滤



# 其他Android安全编码最佳实践

- 使用Safe Browsing API检查URL安全性
- 动态加载代码
- 避免使用监听localhost的IP通信来交换隐私和重要数据
  - 设备上所有应用都可以访问localhost
    - 本地暴力破解、服务枚举、数据遍历等风险
  - 优先使用内置认证机制的Android IPC，例如Service
  - 绑定到 INADDR\_ANY 比监听localhost还要糟糕，因为这样  
一来，您的应用可能会收到任何位置发来的请求



## 内容提纲

- Android安全机制概览
- Android应用软件基础
- Android应用软件的安全策略与机制
- 典型安全漏洞原理与实验



# 命令行工具

- 查看环境变量
  - echo \$PATH
- 基本
  - cd / ls / pwd / ps / grep /kill / cat / chmod /chown / mkdir /echo / touch / du / df / set / uptime / top / ifconfig / more
- 进阶
  - su / iptables / iftop / lsof / mount / vmstat / wpa\_cli / sqlite3



# 重要系统目录和文件

- /etc
  - /etc/hosts 静态域名解析记录本地配置，域名解析时最高优先级记录
  - /etc/apns-conf.xml APN(Access Point Name)，移动网络接入点配置
- /data/misc WIFI联网记录等存在此目录下
- /proc 进程信息目录，特别是/proc/<pid>/
- /dev 设备信息目录

更多有价值的目录相关信息请在Linux主机上man hier



# 强大的adb shell

- 包管理
  - pm
- 构造Intent（三大组件管理）
  - am
- 访问ContentProvider
  - content
- 服务管理
  - service



# 典型安全漏洞原理与实验

- 细数历史上的经典Android锁屏绕过漏洞
- 隐私信息的明文存储
  - wifi口令等
  - 第三方应用的不安全数据存储
    - 明文sqlite数据库、SharedPreferences、明文内部存储
- 关闭开发人员选项



# 国行安卓手机的默认信任SSID

- Evil Twins 的最爱和首选

默认出厂设置  
未root手机无法删除

```
shell@android:/data/misc/wifi # cat wpa_supplicant.conf
ctrl_interface=wlan0
driver_param=use_p2p_group_interface=1
update_config=1
device_name=G716-L070
manufacturer=HUAWEI
model_name=HUAWEI G716-L070
model_number=HUAWEI G716-L070
serial_number=[REDACTED]
device_type=10-0050F204-5
config_methods=physical_display virtual_push_button keypad
p2p_no_group_iface=1

network={
    ssid="CMCC"
    key_mgmt=NONE
}

network={
    ssid="CMCC-EDU"
    key_mgmt=NONE
}
```



```
config_methods=physical_display virtual_push_button keypad
p2p_no_group_iface=1

network={
    ssid="[REDACTED]"
    scan_ssid=1
    psk="[REDACTED]"
    key_mgmt=WPA-PSK
    disabled=1
}

network={
    ssid="[REDACTED]"
    psk="[REDACTED]"
    key_mgmt=WPA-PSK
    priority=1
}
```



root手机编辑  
/data/misc/wifi/wpa\_supplicant.conf  
删除默认信任接入点相关配置



# ADB安全

- 已开启USB调试功能的Android 4.2.2+真机设备通过USB连接到电脑

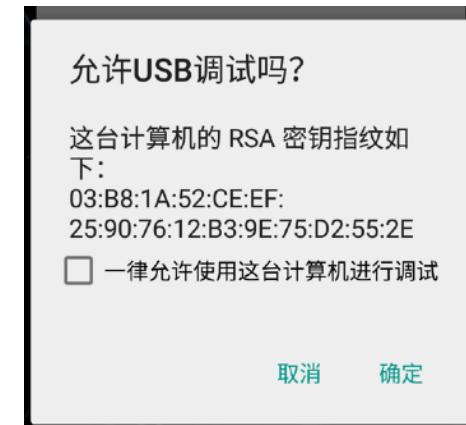
```
root@KaliRolling:~/android# lsusb  
Bus 001 Device 002: ID 05c6:6765 Qualcomm, Inc.
```

- 终端输入adb devices提示设备“未授权”

```
root@KaliRolling:~/android# adb devices  
List of devices attached  
5155ef17      unauthorized
```

- 手机解锁后在主屏幕上弹出对话框
- 只有用户点击“确定”之后才能完成USB调试接口连接

— 手机对电脑进行单向认证



- 电脑不认证手机的风险：利用手机USB连接攻击电脑



# ADB安全

- 如果用户勾选了“一律使用这台计算机进行调试”，则在手机的/data/misc/adb目录下的adb\_keys文件保存当前计算机进行ADB连接所使用的公钥

```
~/.android >>> ls adb*
adb_usb.ini adbkey      adbkey.pub
~/.android >>> head -n 5 adbkey
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQC5TsU1kFSLEBUj
f0LQmnraYq9Vp1jb/dgXt8H2otW/tTQWu3KXXIKf+W/5J/INOxVeXHPDBFH8cx
PzUe12dqBP3IwcscuBSSQSnyH7jd04C5+Z5r5s6DSlesuGfaJmpl3IhWD9e7HDAI
ITqQsD/rdmVegvJ4gqjVM7XJPp79ToTHY+huFWz/udfsa3p1lQNwW7G0W+4ihXfb
~/.android >>> cat adbkey.pub
QAAAAAKeiUiTpGjNTbzz8XT0HlQ00g7WhofxktnmsM/5y0uuuzV5HfcnxklmtAJze+cFh
mvs17n/bBVu6GPHhE79nj7JtTPVqIJ48oJeZXbrP7CQ0iEIMBy71w9WiNxlalaibazis
rAeprQQn8jFRCLVJA1xU65SkAlpdFaTgBIzmlctgfk6LvmCVlgJ+IF60FXXdkoTN3g8
Fbh9zKEEjpcvByNm0CFeKxnvF9uiuXYgrt1rZzBrMFYCy/HggJXxeE7rQMX2JWtYNZ3
3zCFkZwyY14XYd0Tw1+cqCe+vEHUgVOzjgEAAQA= huangwei@c4pr1c3rmbp.local
~/.android >>> pwd
/Users.huangwei/.android
```

```
root@KaliRolling:~/.android# awk '{print $1}' < ~/.android/adbkey.pub | openssl base64 -A -d -a | openssl md5 -
(stdin)= 03:b8:1a:52:ce:ef:25:90:76:12:b3:9e:75:d2:55:2e
```

```
root@A0001:/data/misc/adb # cat adb_keys
QAAAAAKeiUiTpGjNTbzz8XT0HlQ00g7WhofxktnmsM/5y0uuuzV5HfcnxklmtAJze+cFh
mvs17n/bBVu6GPHhE79nj7JtTPVqIJ48oJeZXbrP7CQ0iEIMBy71w9WiNxlalaibazis
rAeprQQn8jFRCLVJA1xU65SkAlpdFaTgBIzmlctgfk6LvmCVlgJ+IF60FXXdkoTN3g8
Fbh9zKEEjpcvByNm0CFeKxnvF9uiuXYgrt1rZzBrMFYCy/HggJXxeE7rQMX2JWtYNZ3
3zCFkZwyY14XYd0Tw1+cqCe+vEHUgVOzjgEAAQA= huangwei@c4pr1c3rmbp.local
QAAAAN+S4EThVrgrnB2AItmVcWNaVu094cPP7PAmF0xFqkMq7CPNMfosUw
jdCe+IirunxUoWdGticigf7+s6NV2bcBqtscafmxX3IcajDOEyWyg0ID+w
xVMmD/bajPnPbkF7XTBC+dkIdFUzrdsvYM+9AMj51mWGxD5T03r1c1xkZ
QsV4kAhwUXbHXSU2+2UTdZGWjqViw01YdV68UpEbDu/IgK9FwTx9Q6IEod
vqNmojidMCI3uH2KHl+nHqX1iq3leRyCUgEAAQA= unknown@unknown
```

```
root@A0001:/data/misc/adb # ls -l
-rw-r----- system shell 1445 2015-11-03 00:13 adb_keys
```



## 参考文献

- Official Android Security Overview
- Android官方术语字典
- Android Architecture For Beginners 2013.4
- Yuksel A S, Zaim A H, Aydin M A. A Comprehensive Analysis of Android Security and Proposed Solutions[J]. 2014.
- Android应用安全之android平台的跨应用攻击
- Android安全架构及权限控制机制剖析



## 参考文献

- Attacks on WebView in the Android System



## 延伸阅读

- Android 4.3中SEAndroid展望
- 极客学院视频教程：签名、权限、组件安全、文件与用户安全
- Android Webview UXSS漏洞攻防
- droidsec wiki
- 灵犀语音助手锁屏认证绕过设计缺陷
- 小米手机屏幕图案解锁设计缺陷可以绕过