Applications Mobiles et Internet des Objets
Introduction a l'architecture d'Android

Thibault CHOLEZ - thibault.cholez@loria.fr

TELECOM Nancy - Universite de Lorraine

LORIA - INRIA Nancy Grand-Est

From Q. Jerome, R. State and T. Cholez: IM'13 tutorial "Android security overview"

11/01/2016

Plan

1. Introduction
2. Android Internals
3. The Android Paradigm
4. Android Security

Plan

1 Introduction

2 Android Internals

3 The Android Paradigm

4 Android Security

What is Android ?

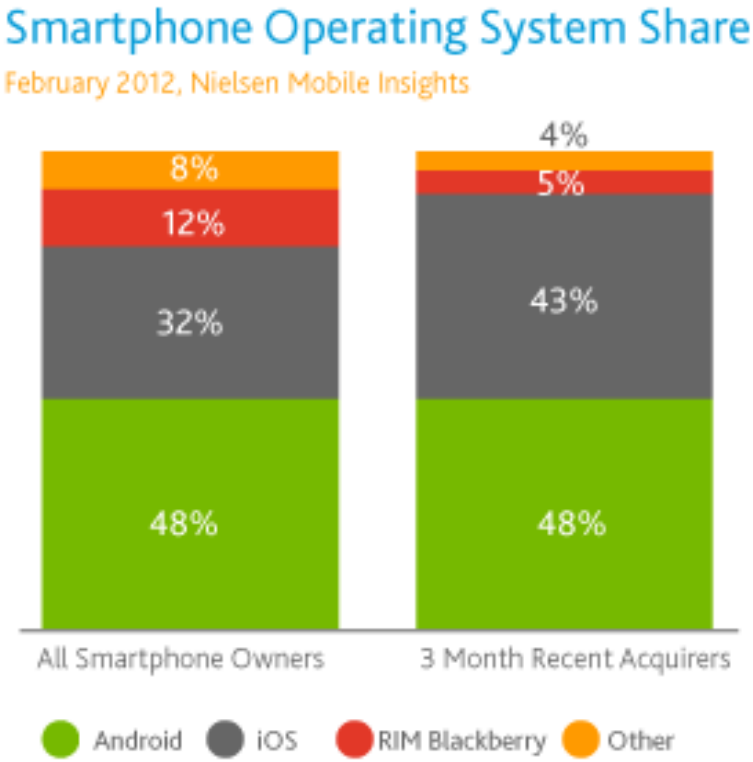
An Operating System for embedded devices

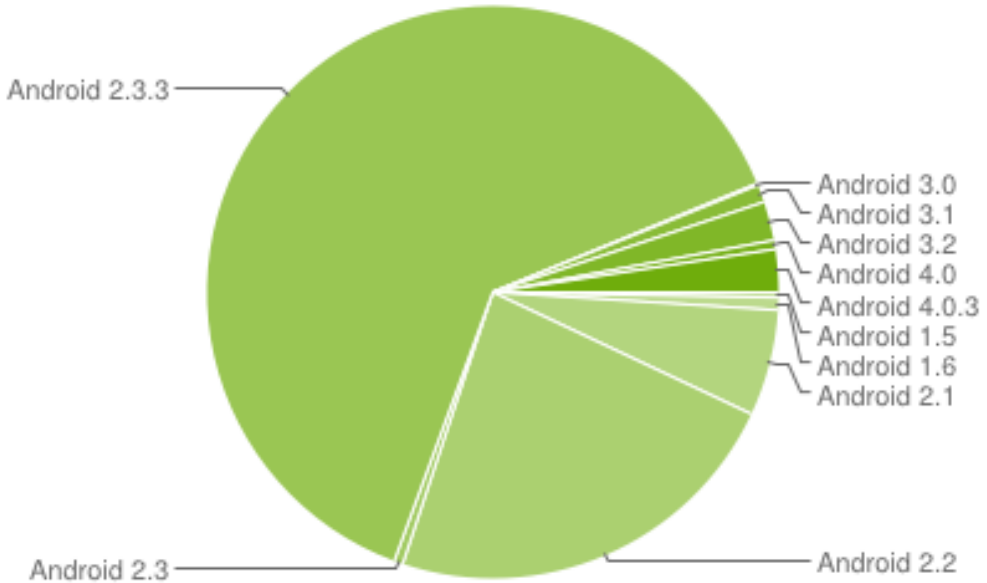
- Mobile phones
- Tablets
- Swatches
- Radios
- Fridges

Key points

- An open source project
- Project led by Google
- About 500 million devices
- Mobile OS most widely used

Some recent figures : the bright side





How Android fragmentation affects Security (1/2)

Security features progressively introduced in Android : earlier versions

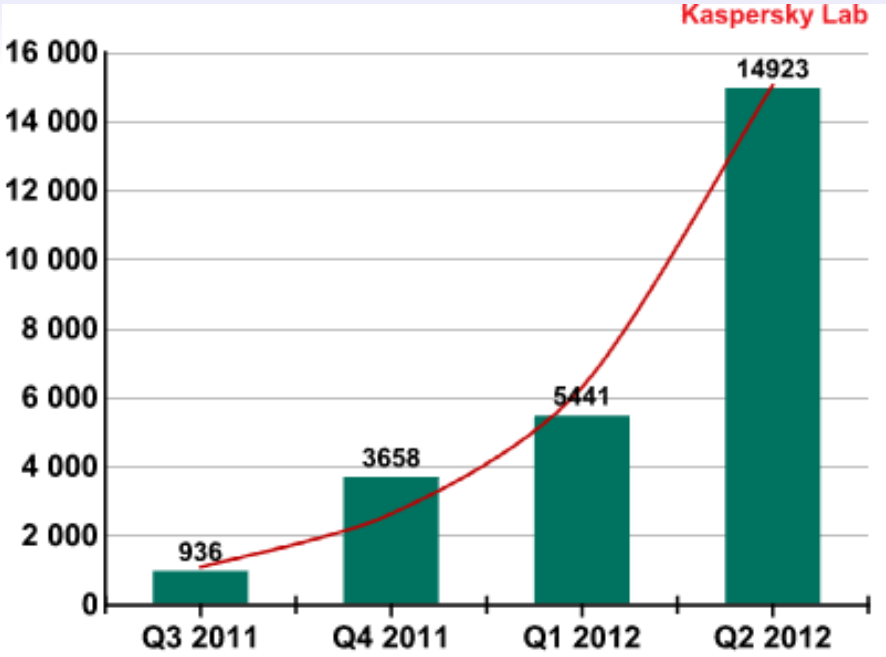
- Android 1.5+
  - ProPolice to prevent stack buffer overruns (-fstack-protector)
  - safe_iop to reduce integer overflows
  - Extensions to OpenBSD dlmalloc to prevent double free() vulnerabilities and to prevent chunk consolidation attacks.
  - OpenBSD calloc to prevent integer overflows during memory allocation
- Android 2.3+
  - Format string vulnerability protections (-Wformat-security -Werror=format-security)
  - Hardware-based No eXecute (NX) to prevent code execution on the stack and heap
  - Linux mmap_min_addr to mitigate null pointer dereference privilege escalation (further enhanced in Android 4.1)

How Android fragmentation affects Security (2/2)

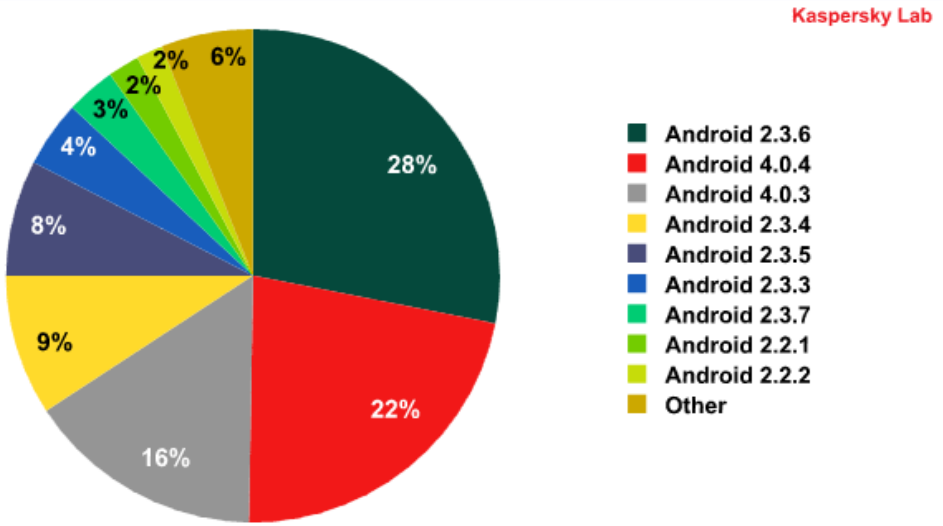
Security features progressively introduced in Android : latest versions

- Android 4.0+
  - KeyChain class to allow applications to use the system credential storage for private keys and certificate chains
  - Address Space Layout Randomization (ASLR) to randomize key locations in memory
- Android 4.1+
  - PIE (Position Independent Executable) support
  - Read-only relocations immediate binding (-Wl,-z,relro -Wl,-z,now)
  - dmesg_restrict enabled (avoid leaking kernel addresses)
  - kptr_restrict enabled (avoid leaking kernel addresses)
- Android 4.2+
  - SELinux support
  - FORTIFY_SOURCE for system code

Some recent figures : the dark side



Number of new malware on Android

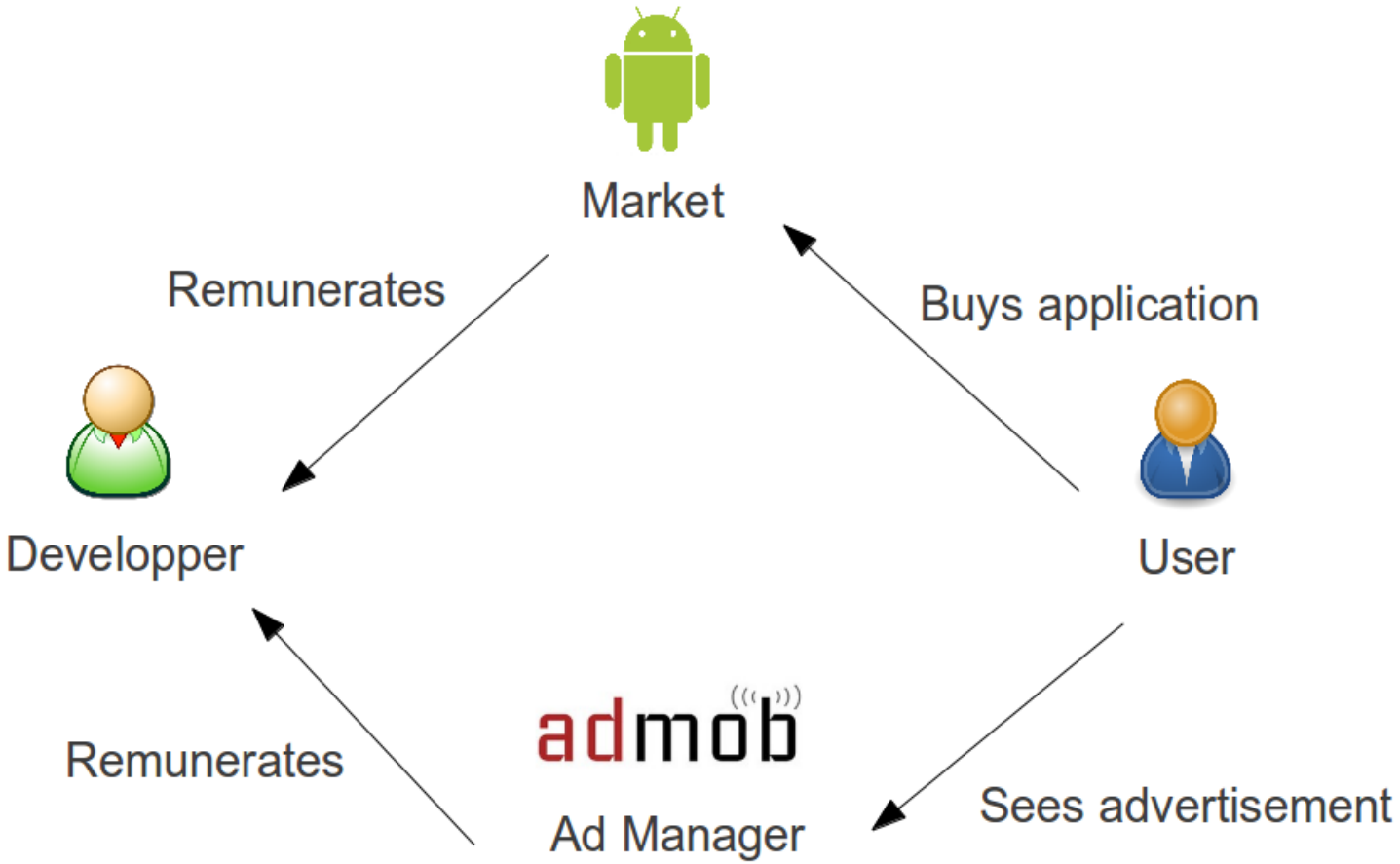


Malware repartition among versions, Q3 2012

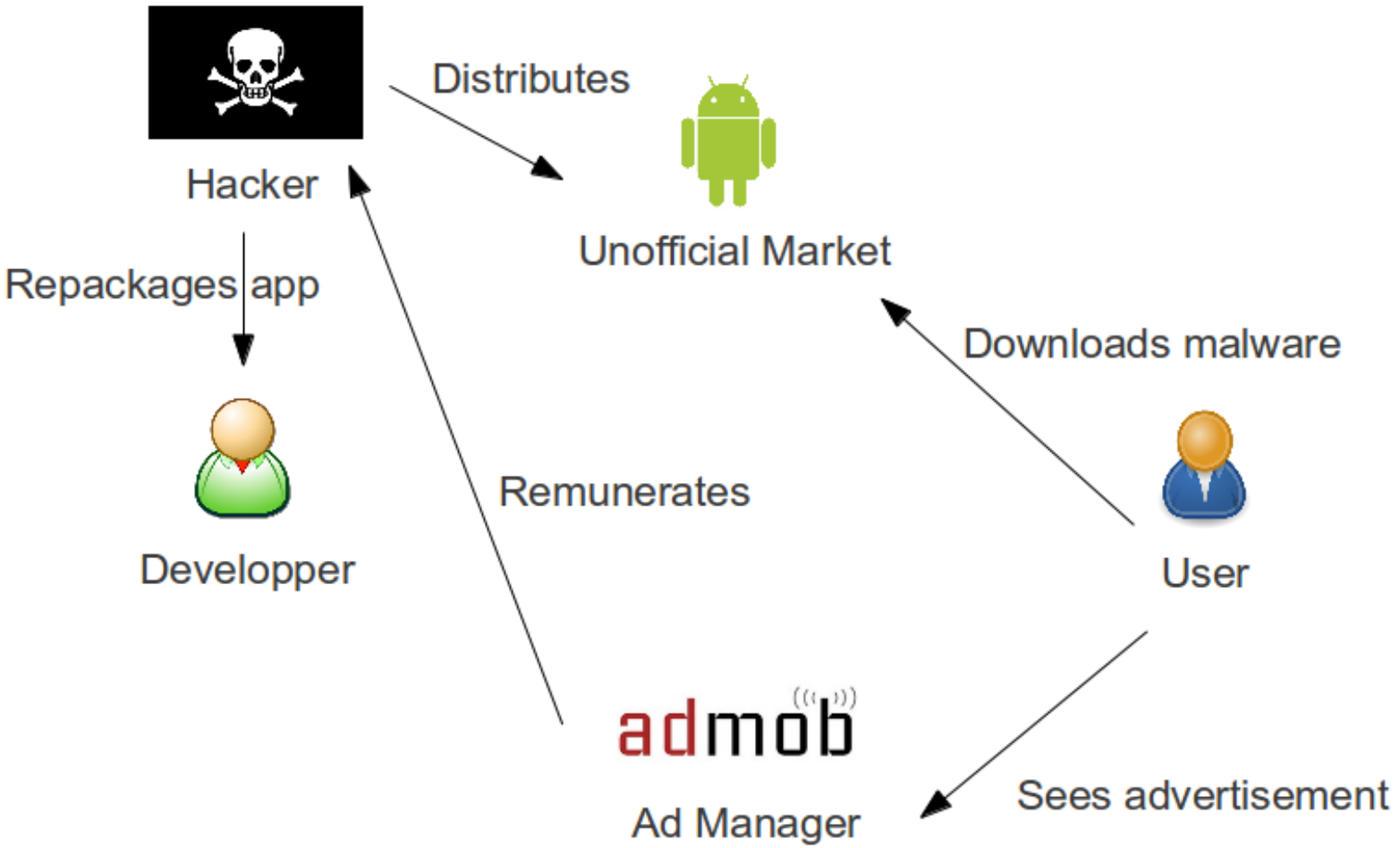
Why is it attractive ?

- Increasing number of activations
- The easy way to distribute apps
- A lot of alternative markets

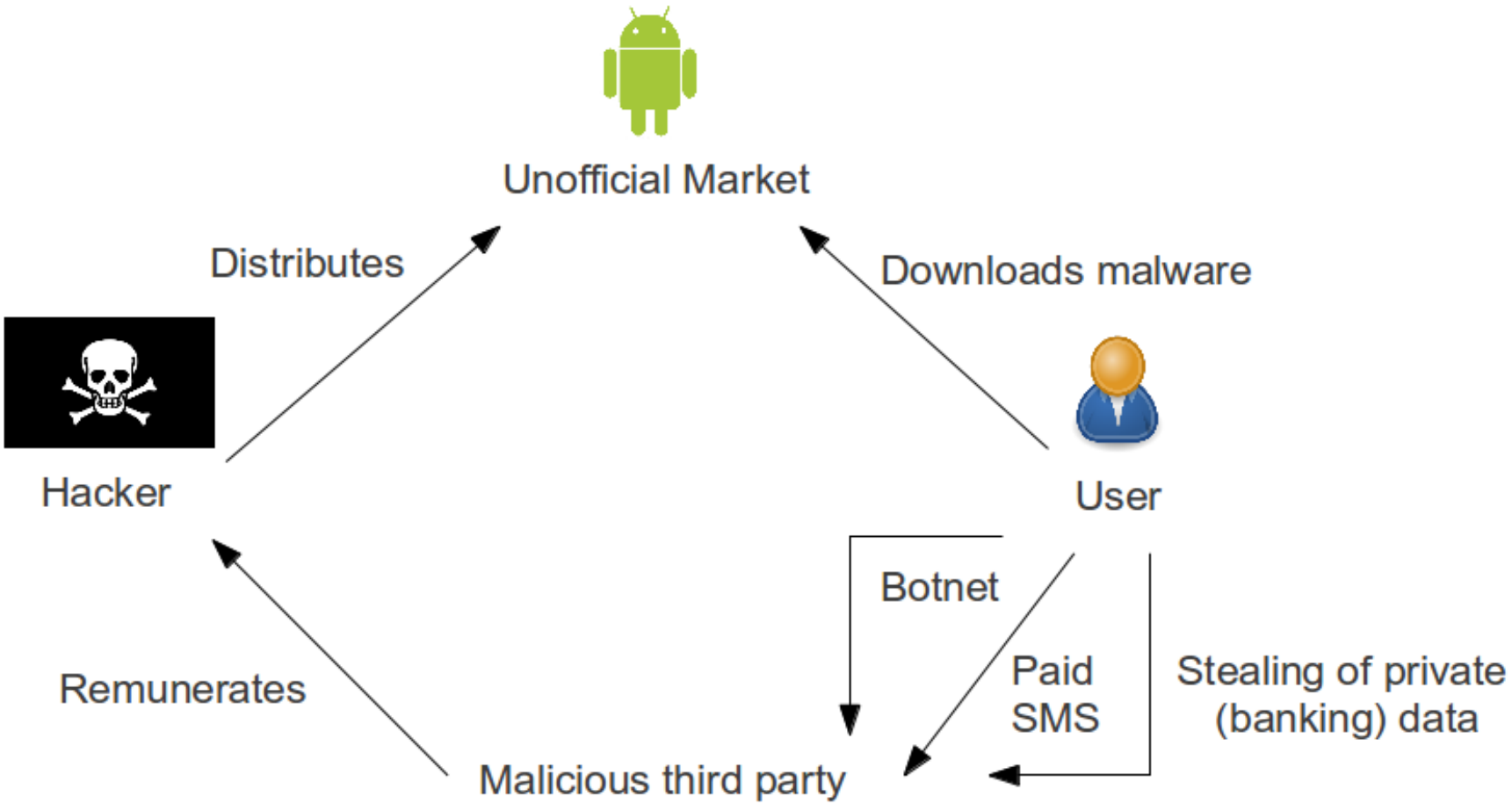
Business model of Android



Business model of Android



Business model of Android



Plan

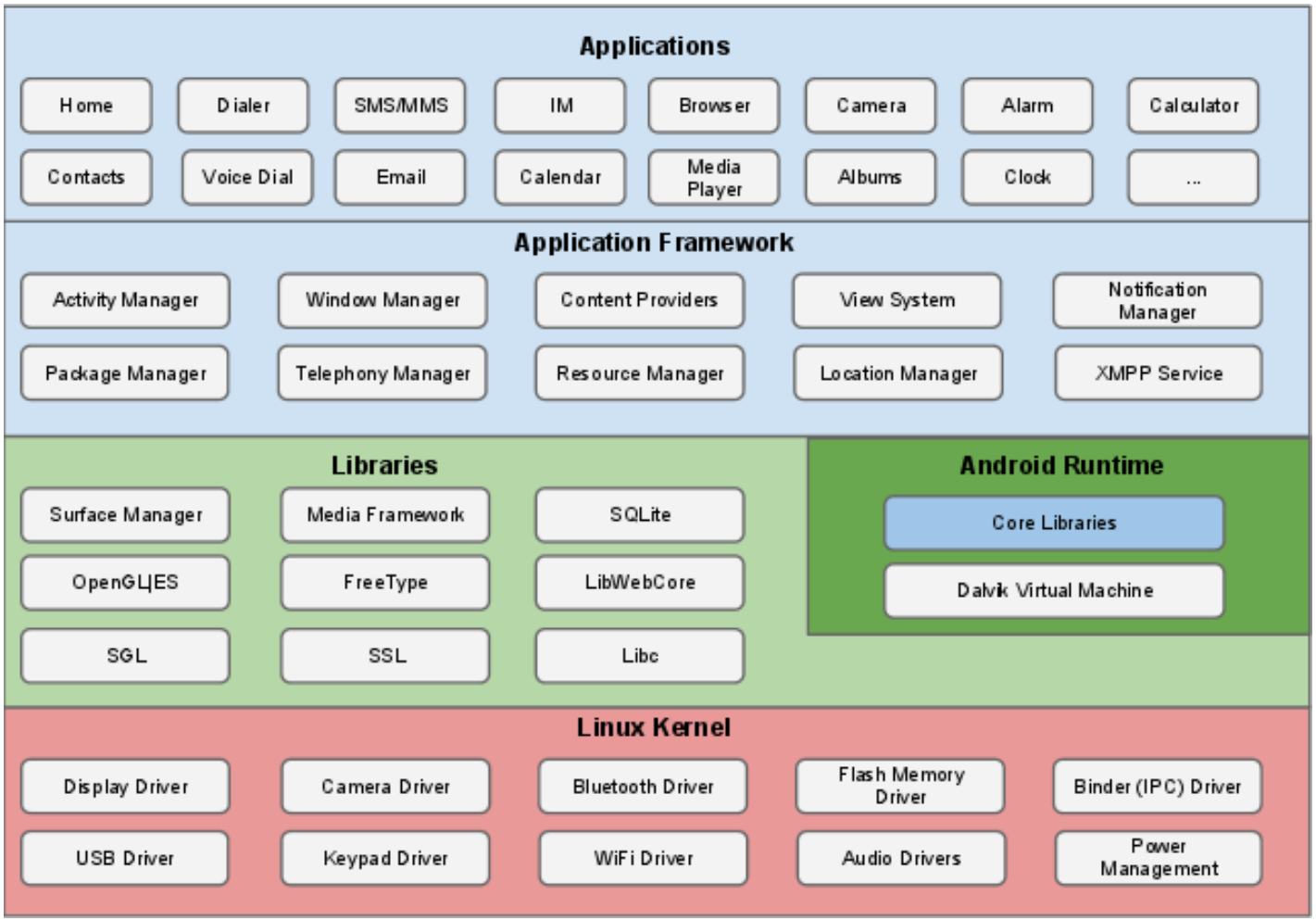
1 Introduction

2 Android Internals

3 The Android Paradigm

4 Android Security

The android framework



The android framework

public API for applications developpers

- Content Providers : access data from another application
- Resource Manager : access graphical assets, layout, strings
- Notification Manager : manages notification messages
- Activity Manager : manages the life cycle of an application

And many native libraries (multimedia, web engine, databases, libc, etc.).

The android SDK

SDK structure

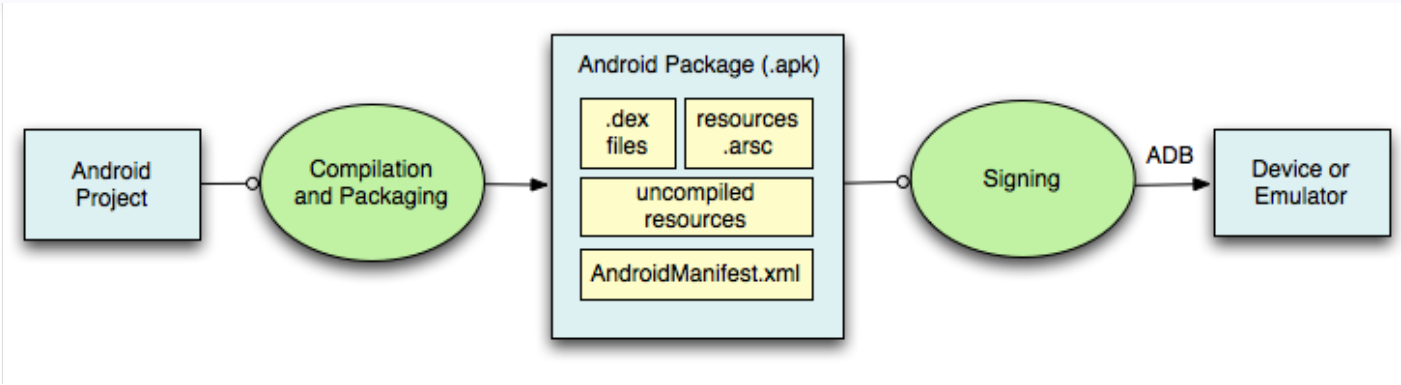
- $< sdk/tools >$: debugging tools (emulator, ddms, proguard, App excerciser Monkey, etc.)
- $< sdk/platform - tools >$: development tools (adb, aidl, appt)
- $< sdk/docs >$: API documentation
- $< sdk/platforms >$: system images (x86, ARM)

The android SDK

Main tools

- Android Virtual Device : creates and manages Android VM
- Android Debug Bridge : shell access, file copy, package installation on a virtual device
- Dalvik Debug Monitor Service : Debugging tool, monitor processes in a virtual device and resources usage

Application building process



Details

- .apk == .zip
- dex : Dalvik Executable (Dalvik bytecode)
- AndroidManifest.xml : ID card of the application (all activities, services, receivers, permissions, intents are declared)
- Ressources (static assets)
- apps can be written in Java as well as in C (JNI)

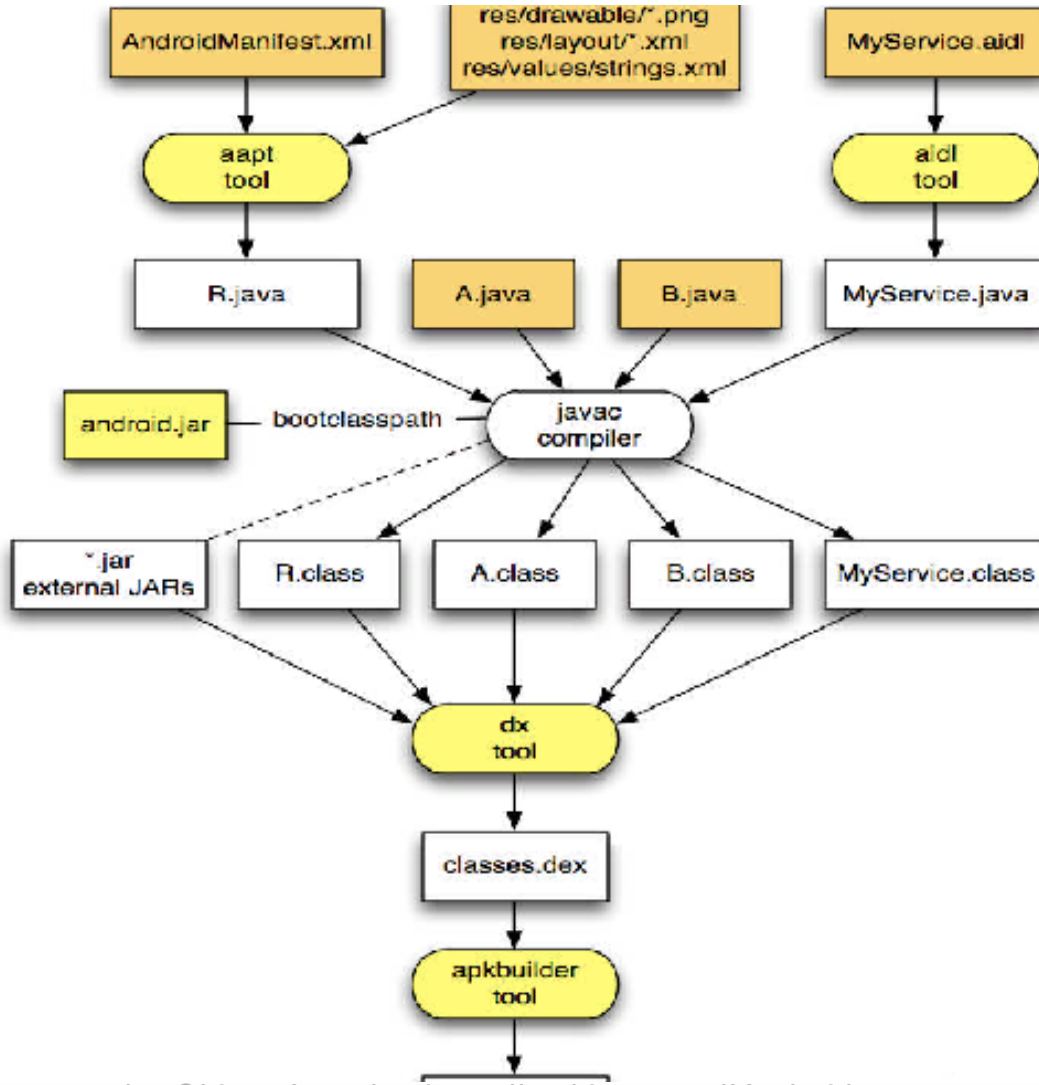
Application building process

Code source

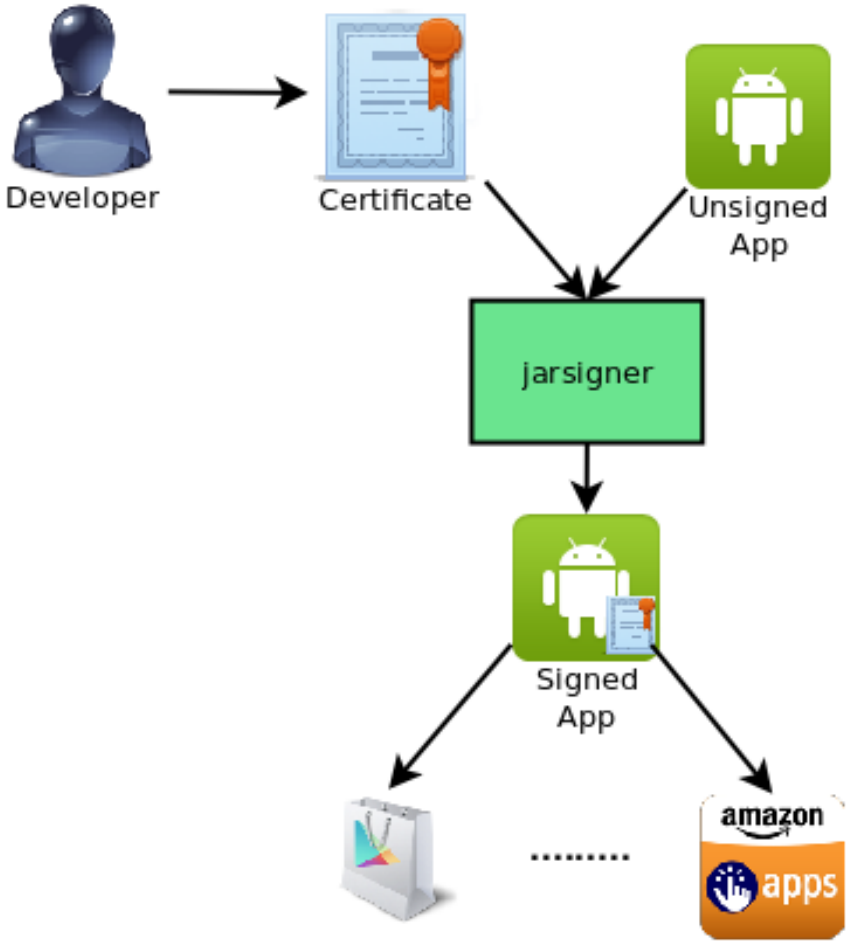
Byte Code Java

Byte code optimisé
Génération fichier.dex

Construction du package
destiné à être installé



Application signing process



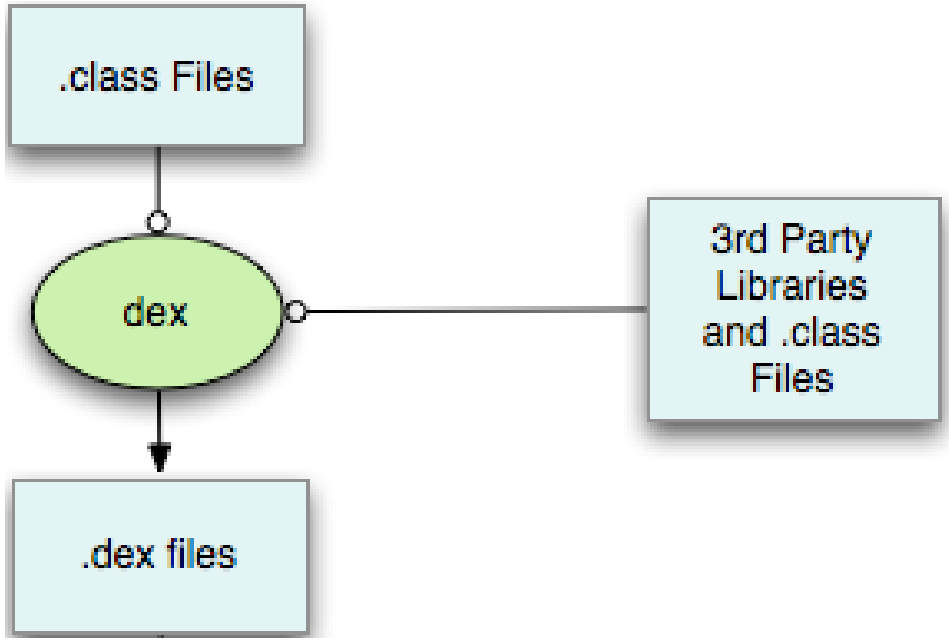
Details

- the certificate can be issued by a cert authority but usually developers self-sign their certificates (no trust chain)
- an unsigned app can not be installed on a phone
- Google does not seem to check for bogus certificates
- Apps are not checked by Google before submission

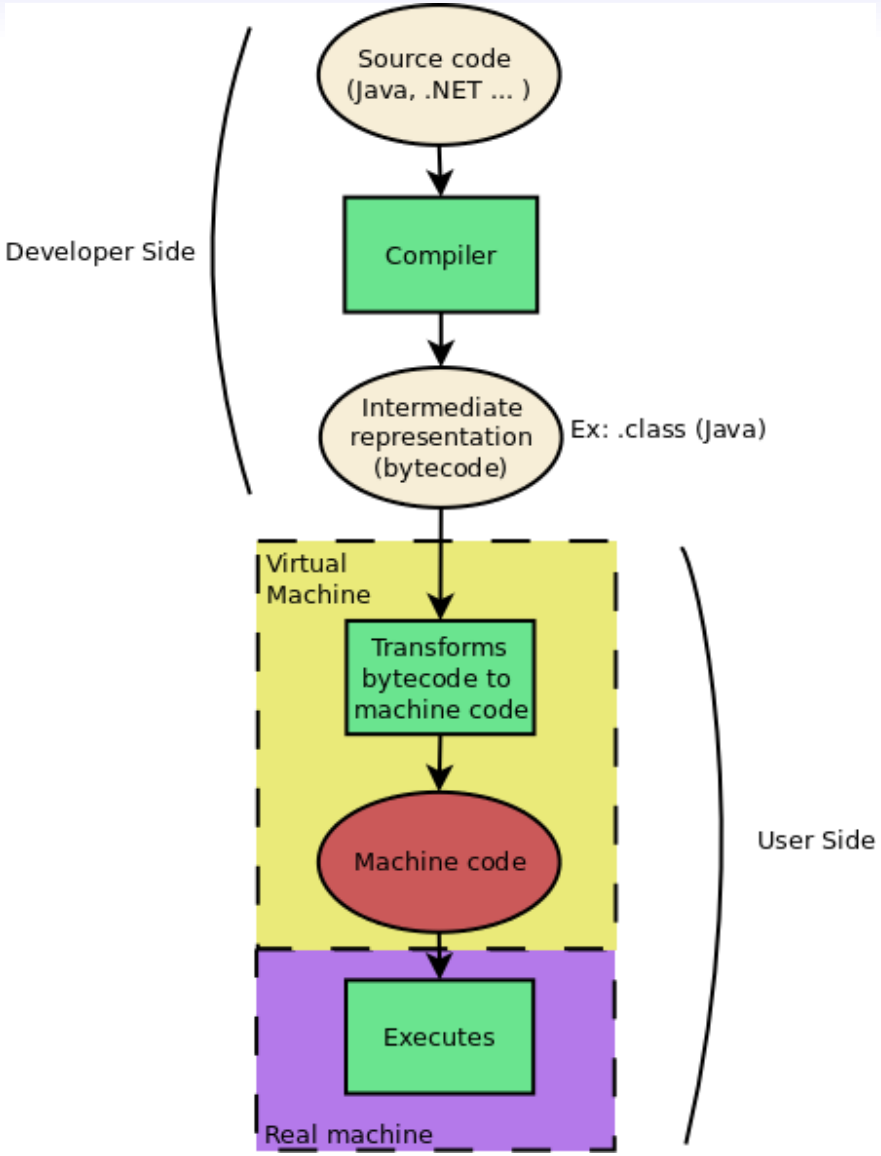
What is a dex file ?

Details

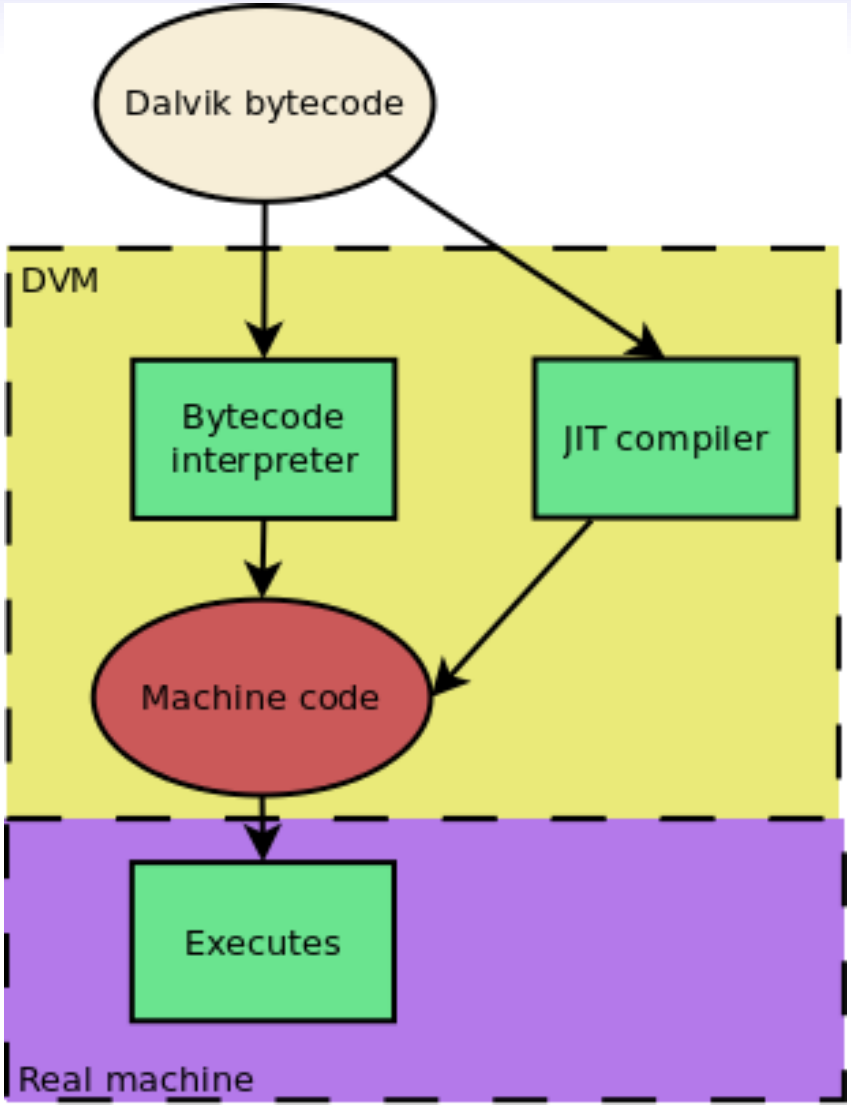
1. Java code gets compiled to .class (Java bytecode)
2. .class files and external libraries are injected to a dex compiler
3. the dex compiler compiles Java bytecode to Dalvik bytecode and generates a unique .dex file



Virtual Machine : General concepts



The Dalvik VM



Features

- Register based machine (less instruction than stack based one)
- Interprets Dalvik bytecode
- JIT compiler since Android 2.2

JIT (Just In Time) compiler

- watches out for often used code
- compiles it just once so that the interpreter does not loose time to compile it each time

Stack VS Registers

Add register a to register b and store the result in c

Stack based (Java)

push a
push b
add
pop c

Register based (Dalvik)

add a,b,c

Which wins ?

- a program written for a stack based machine takes much more in memory (not fitted to embedded systems)
- a program written for a register based machine tends to be slower than its equivalent for a stack based machine. But fast enough for embedded devices

Dalvik Bytecode snippet

Snippet

```
.method protected enforceReceiveAndSend(Ljava/lang/String;)V
        .parameter "message"
        iget-object v0, p0,
Lcom/android/internal/telephony/IccSmsInterfaceManager;->mContext :Lan-
droid/content/Context;
        const-string v1, "android.permission.RECEIVE_SMS"
        invoke-virtual v0, v1, p1, Lan-
droid/content/Context;->enforceCallingPermission(Ljava/lang/String;Ljava/lang/String;)V
        iget-object v0, p0,
Lcom/android/internal/telephony/IccSmsInterfaceManager;->mContext :Lan-
droid/content/Context;
        const-string v1, "android.permission.SEND_SMS"
        invoke-virtual v0, v1, p1, Lan-
droid/content/Context;->enforceCallingPermission(Ljava/lang/String;Ljava/lang/String;)V
        return-void
.end method
```

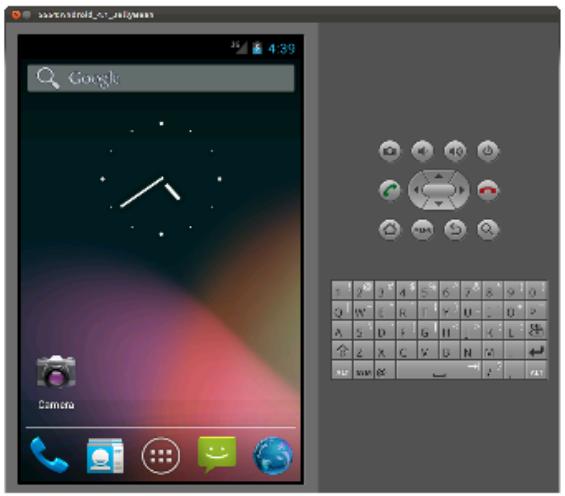
Details

- iget-object vA,vB : get the object vB and store it into vA
- const-string vA,S : put the String S into the register vA
- invoke-virtual vA,vB,vC,method : call method with the parameters vA,vB,vC

Android software development cycle

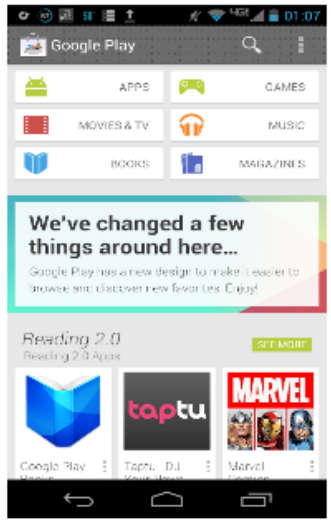
Several deployment steps needed

- Emulation within development framework (based on QEMU)
- Execution and testing of the code on a real smartphones
- Distribution within Google Play Store or other markets



Main development on emulator

Real smartphone(s) testing



Play Store distribution

Plan

1 Introduction

2 Android Internals

3 The Android Paradigm

4 Android Security

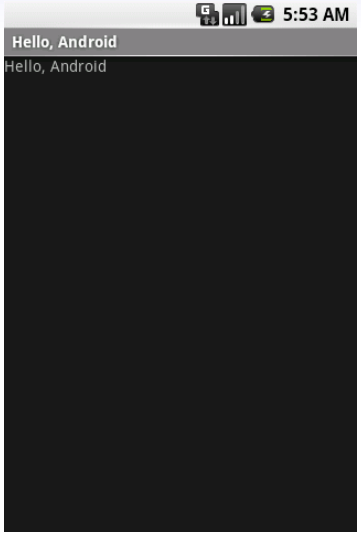
Generalities

- Each application is a different (Linux) User
- By default, each application has its own unique UID (User ID) and can access only to the files that it owns
- Each application runs in a unique process
- A process owns its own VM
- It is possible by setting the ShareUID parameter in the AndroidManifest.xml that allows two apps signed by the same certificate to run in the same process
- Android adopts a component based approach. An app can define different components (Activity, Service, Content Manager or Broadcast Receiver) which can be accessed by other apps according to the security policy

Components (1)

Activity

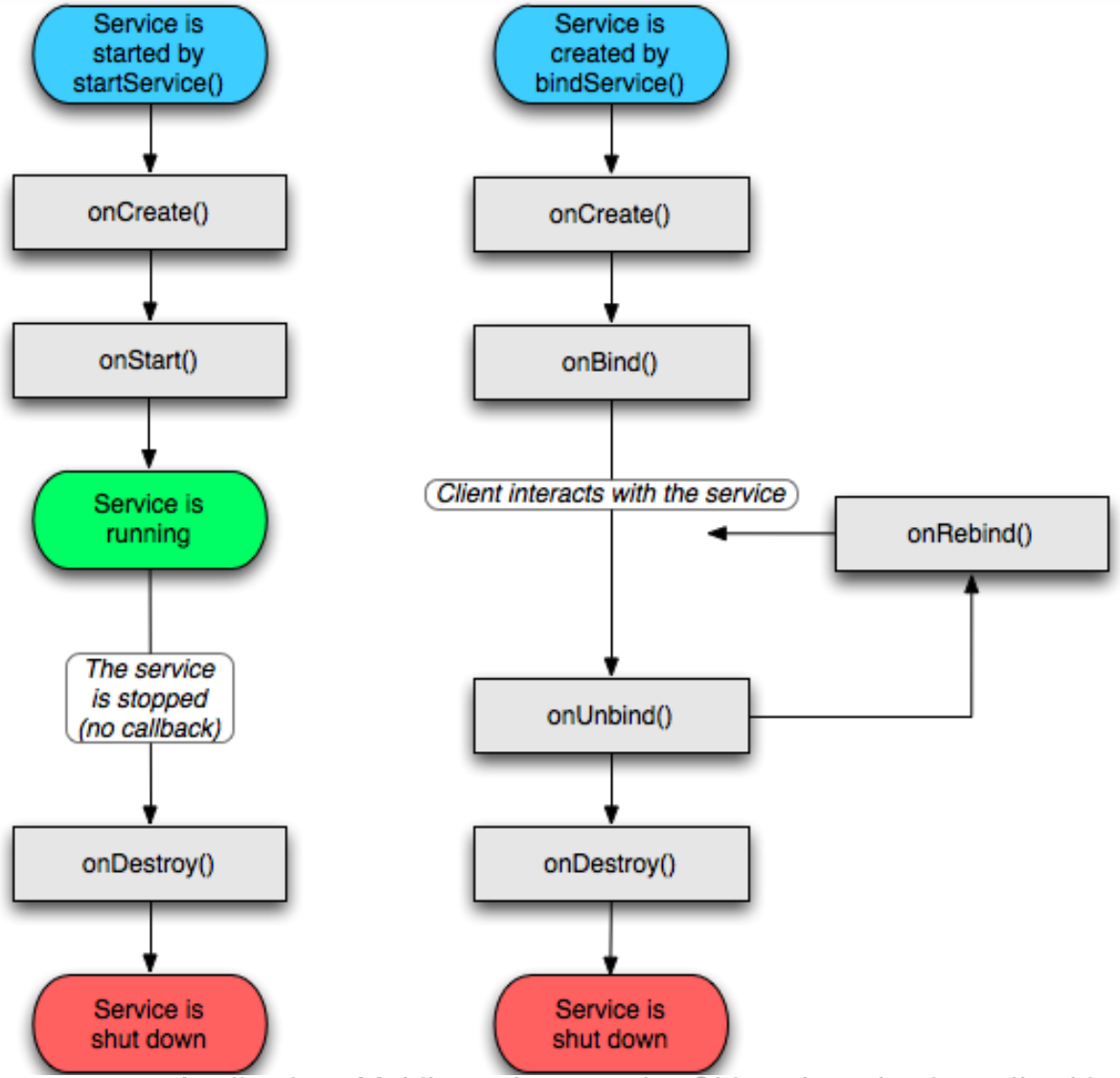
An activity is an interface used to interact with the user. As Android is component based, all activities from an app communicate through Intents (Inter Process Communication specific to Android).



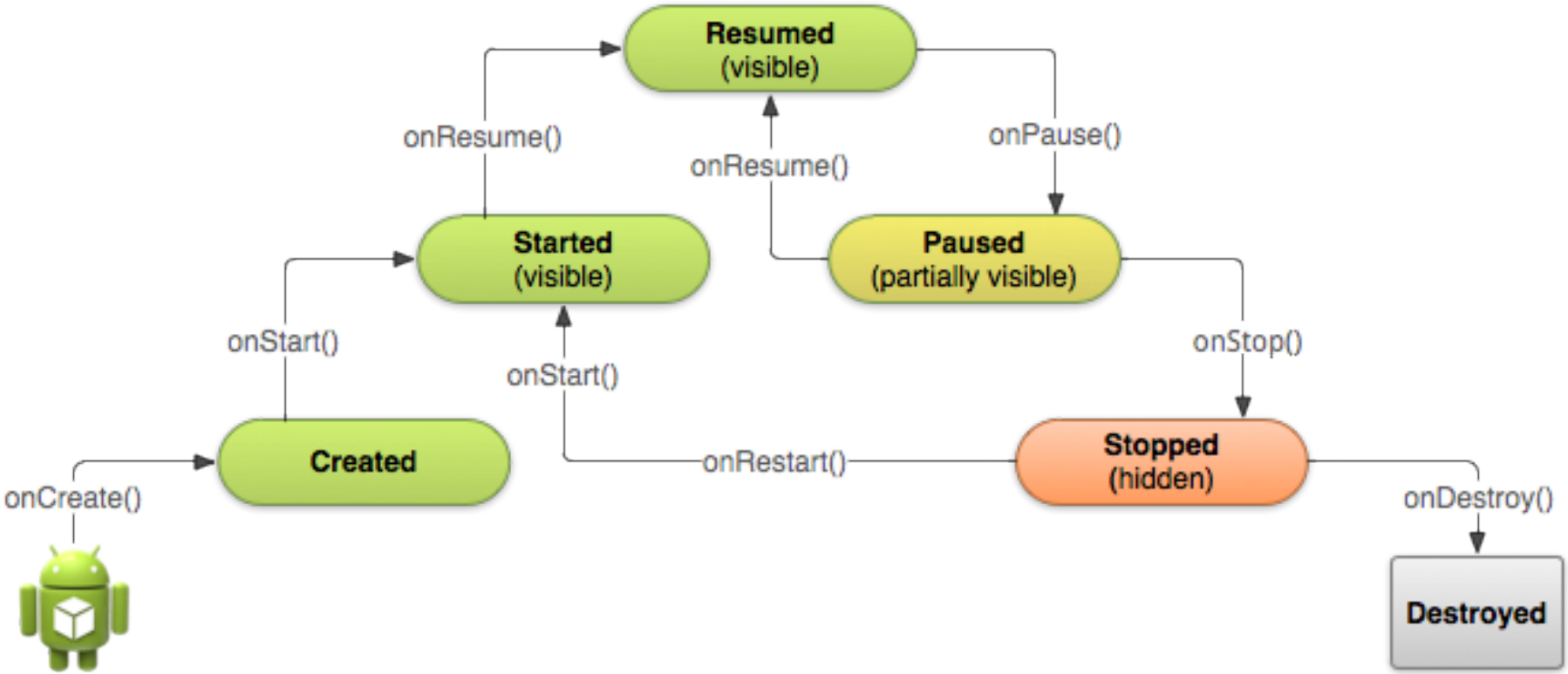
Services

Services are background task that runs without any User Interface. It is usually launched from an Activity or from a Broadcast Receiver via Intents. To interact with a service we can use either Intents or a Binder. A downloading deamon or a song player are examples of Service.

Service lifecycle



Activity lifecycle



Components (2)

Content Provider

A Content Provider is a database like component that can be queried in order to retrieve a specific content. Usually it is a way to share data between different apps. Thus we can protect contents by allowing access to specific apps. To access to "contacts", "call logs", ... you have to request their Content Provider.

Broadcast Receiver

It is used to catch "Broadcasts" (notifications) sent by the system (publish/subscribe model). For instance once the system is booted the system sends a Broadcast or when the battery is low. Usually Broadcast Receivers implement very straightforward functions like starting a Service.

Data Storage in Android

Different data storage options are possible :

- Shared Preferences : Store private primitive data in key-value pairs.
- Internal Storage : Store private data on the device memory.
- External Storage : Store public data on the shared external storage.
- SQLite Databases : Store structured data in a private database.
- Network Connection : Store data on the web with your own network server.

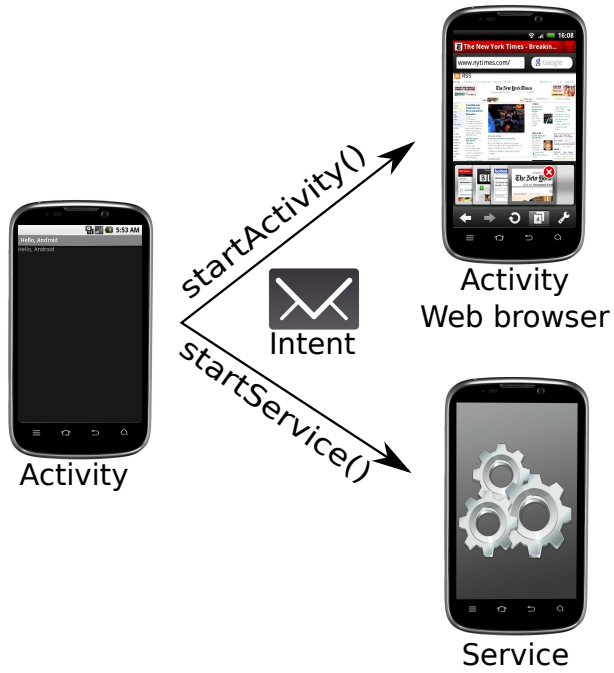
Data storage management

- Best solution depends on specific needs (if the data should be private to the application, accessible to others, how much space is required, etc.)
- Private data can also be exposed to other applications through a content provider (exposing read/write access)

Communication between components

Intents

- Can be used by all the components except Content Providers that are queried through Content Resolver. They are used to start/stop other component but also to send results back.
- They are asynchronous
- Even inside the same app Intents are used to communicate between components
- All the Intents are sent to the system that checks if the communication is allowed before going further.



Communication between components (2)

Binders

Binders are used to communicate between a Service and an Activity running in the same process (part of the same app). It creates a client-server architecture between an Activity and a Service. Cannot be used between processes.

Messenger

For continuous interaction between process we have to implement the Messenger interface in both client and server sides. Thread safe interface that can handle messages from different clients.

AIDL

Low level interface that allows implementation of IPC.

Intents (1)

Definition

An Intent is an abstract description of an action to performed, or in case of a Broadcast a description of an action that occurred.

Explicit Intents

explicit intent : this kind of Intent holds a Component parameter set with the name of one specific component. This is mostly used between components of the same app.

Details

- component : targetted component
- data : data to provide to the component
- extra : extra data depends on the target
- flags : set flags depends on the target

Intents (2)

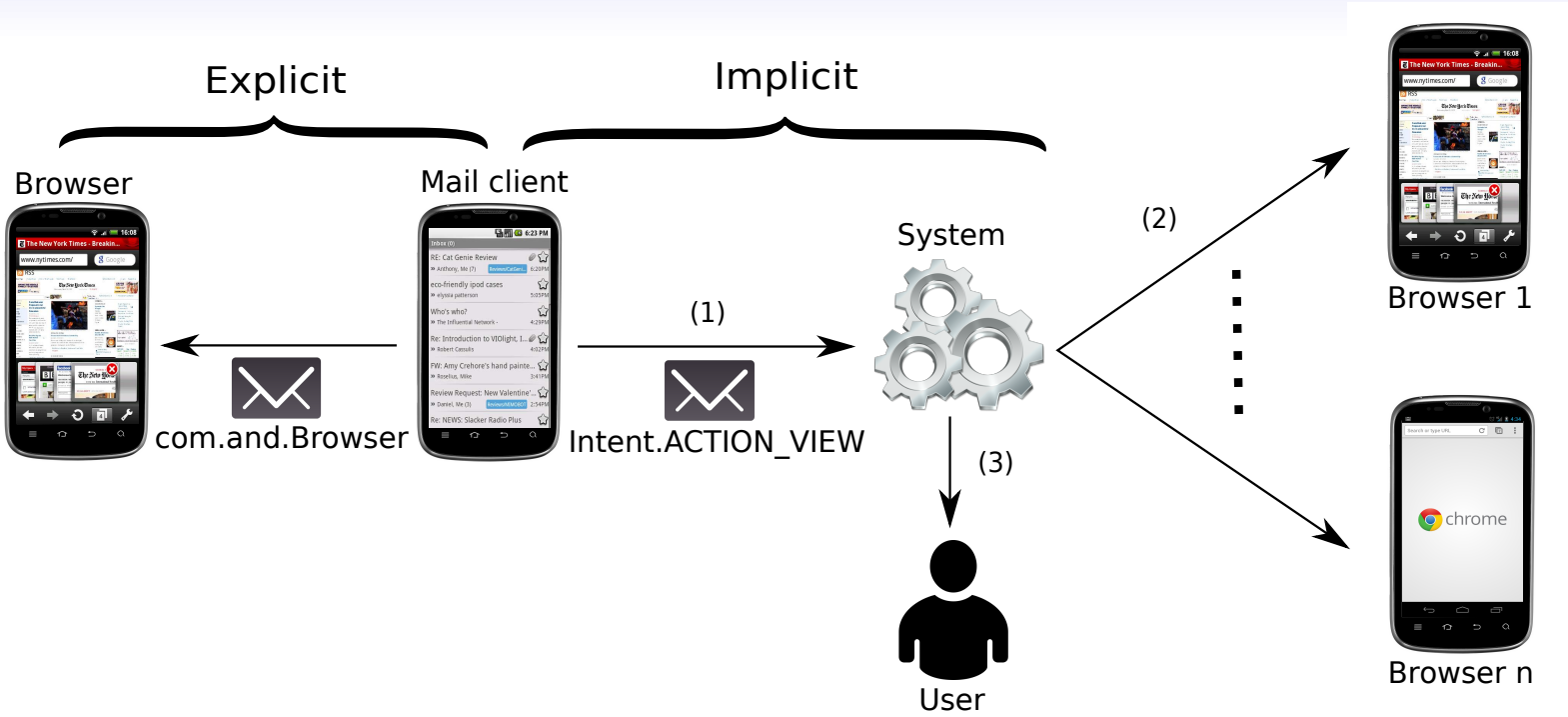
Implicit intents

implicit intent : it has its Component attribute void and specify an action that has to be performed, the data to use and the category of the target component. Then the system resolves which component is able to treat the Intent. When more than one component can deal with such an Intent the user is prompted. The system uses Intent Filters specified in the AndroidManifest.xml file to find out which component could handle the Intent.

Details

- data : data to act on
- action : action to perform
- category : category of the targeted component

Explicit VS Implicit



1. The mail client sends an implicit intent asking for a browser
2. The system resolves the intent and find all the component filtering this kind of intent
3. The system sends back the response to the user who can choose the browser he prefers

Resolution of Implicit intents : Intent Filters

Intent Filters

Intent filters describe the Implicit intents that a component will be able to respond to. If several components have same Intent filters, the system will ask the user to make a choice as described in the previous slide.

- These can be used by activities, services and receivers
- An intent filter must contain an action to perform
- Intent filters can contain either category or data information

```
<activity android:name=".MainActivity" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

NB : in this case, both action and category must be defined

Context

Context of an application

All what is linked to a given application, "Context" object created when the application starts, destroyed when it stops.

- Activity, services, broadcast receivers
- Resources, classes, intents
- Intent filters can contain either category or data information

Context.getApplicationContext or Activity.getApplication

Plan

1 Introduction

2 Android Internals

3 The Android Paradigm

4 Android Security

Android security

Generalities

Android is a permission based system. When a user installs an application, the permissions used by the app are exposed to the user. If the user does not want to grant the permissions he has to deny app installation. Doing so the permission set of an application is granted at installation time. NO PERMISSION CAN BE GRANTED AT RUNTIME.

What are permissions ?

Each permission stands for a user group at the Linux level. Thus, the only thing that the system does when an app is installed is to add the UID of the app to the users groups corresponding to the permissions. For instance the "android.permission.INTERNET" is nothing else that the inet users group.

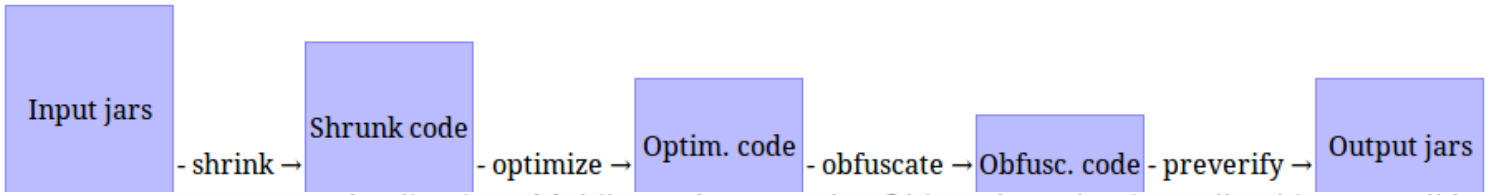
Code obfuscation with Proguard

What is Proguard ?

- Tool to shrink, optimize, and obfuscate code by removing unused code and renaming classes, fields, and methods with semantically obscure names.
- Purpose : make reverse engineering harder.

Proguard features :

- Fully integrated into the Android building system (automatically executed in "release mode")
- Generation of "mapping.txt" to allow programmers to debug obfuscated published applications.



Complement about Android security

Motivation

Android also specifies different protection levels associated to permissions when they are defined. According to its level, a permission might need further requirements than being declared in the Manifest in order to be used.

Example

An app that declares "android.permission.INSTALL_PACKAGES" should not be able to install packages just by declaring such a permission in the Manifest.

NB

Fines grained permissions can be granted for Content Provider : readPermission and writePermission.

Protection levels

normal	The default value. A lower-risk permission that gives requesting applications access to isolated application-level features, with minimal risk to other applications, the system, or the user.
dangerous	A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user.
signature	A permission that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission.
system	A permission that the system grants only to applications that are in the Android system image.

Static Permission Enforcement

In the AndroidManifest.xml file

- Developers are strongly recommended to define their own permissions in the Manifest and requiring them to access components of their apps
- This mechanism can restrict access to the component for which the permission has been defined

```
<manifest . . . >
    <permission android:name="com.example.project.DEBIT_ACCT" . . . />
    <uses-permission android:name="com.example.project.DEBIT_ACCT" />
    . . .
    <application . . .>
        <activity android:name="com.example.project.FreneticActivity"
                  android:permission="com.example.project.DEBIT_ACCT"
                  . . . >
            . . .
        </activity>
    </application>
</manifest>
```

FIGURE : Using permissions

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.me.app.myapp" >
    <permission android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"
        android:label="@string/permlab_deadlyActivity"
        android:description="@string/permdesc_deadlyActivity"
        android:permissionGroup="android.permission-group.COST_MONEY"
        android:protectionLevel="dangerous" />
    ...
</manifest>
```

FIGURE : Defining permissions

Dynamic Permission Enforcement

For all components

- All the components can protect their access in using dynamic permission checks. This can be done in using the *checkCallingPermission()* functions
- This is possible to check permissions in several ways like by PID or by package name
- This can be used only if the call comes from another process

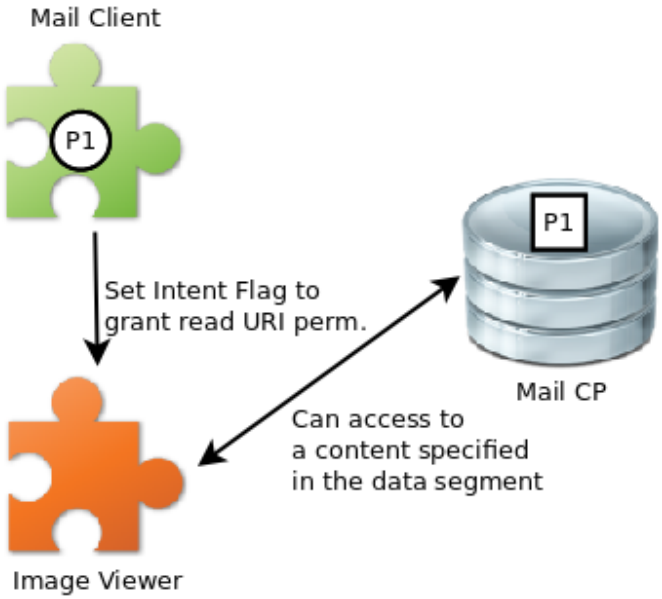
Enforcing permission for Broadcasts

When a Broadcast is sent, we can require that the Receiver should have the required permission. The permission is set in the *Context.sendBroadcast()* method.

Specific permission enforcement

Content Providers

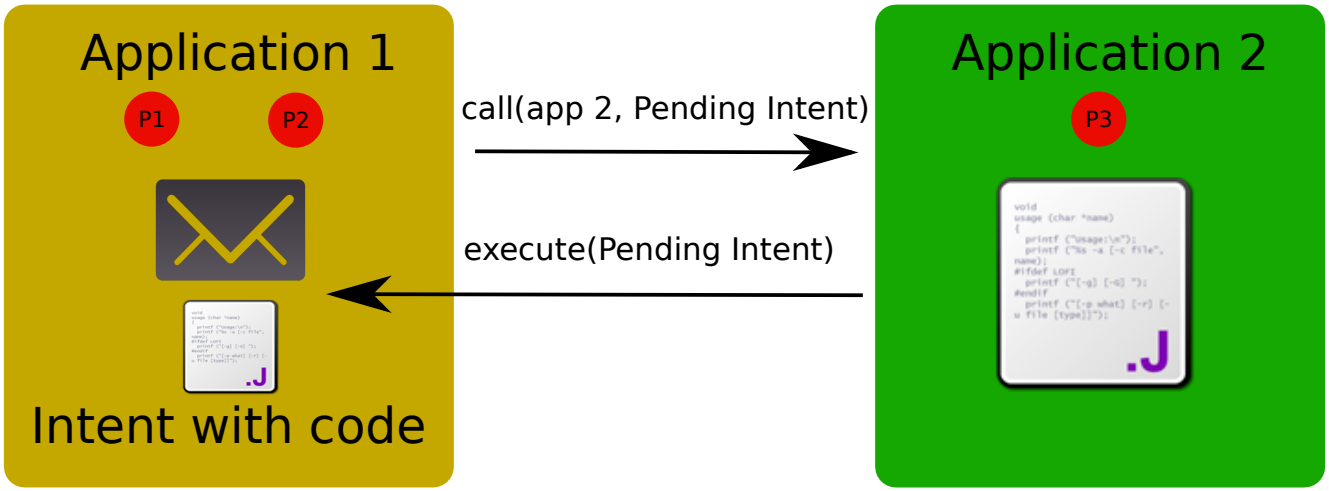
Android offers a mechanism to allow modification of a CP even though an app does not have required permissions. The CP can grant temporary access to one content by declaring "grant-uri-permissions" in its Manifest. This is done to limit the scope of permissions asked by an app that would only have to deal with a CP.



Pending Intents

Description

A Pending Intent is an Intent with a piece of code associated with it. The system registers pending intent as token accessible from any others applications. This means that any application can execute this piece of code as if it were the definer of the code. This mechanism allows a third application to execute a piece of code with the permission of the definer of the Pending Intent.



Discussion about android security (1)

Summary of security mechanisms

- Signing process : X.509 certificates used to identify who has developed an application...but mostly self-signed
- Permissions : used to manage rights of each application and to manage the access to components
- Permission system has several variants
  - Static declaration in Manifest
  - Dynamic permission check
  - Protection level : permissions granted only under constraints
  - Fine grained Content Provider permissions used to limit permission abuses

Discussion about android security (2)

Developers misusing the Permission system

This creates permission gaps, meaning that the app declares more permissions that it really needs. Moreover, developers can easily forget protecting sensible components by requesting the appropriate permissions.

Users misusing the Permission system

Users are prone to install over-privileged applications. Thus malware can be installed even if they are not misleading on their behaviour.

Consequence : Privilege escalation

It is possible to find android applications that do not protect their component which use critical components. This can lead to application level privilege escalation or data leaks.