Android OS

Android OS

Why:

Open source – have access to system code

All over the place – dominates smartphone market

Interesting security model

A PKI would probably hamper deployment
but apps must be signed by developers

Application does not install without a valid certificate

Principle of least privilege

Permission to do something must be explicity given

Defense in depth

OS kernel – iptables, selinux, sandboxing,
mandatory access control

Applications – secure message passing

Users – approval to release resources

Android OS

Android is a software stack for touchscreen mobile devices, such as smartphones and tablet computers

Android Inc. was founded in 2003 to produce software for mobile phones

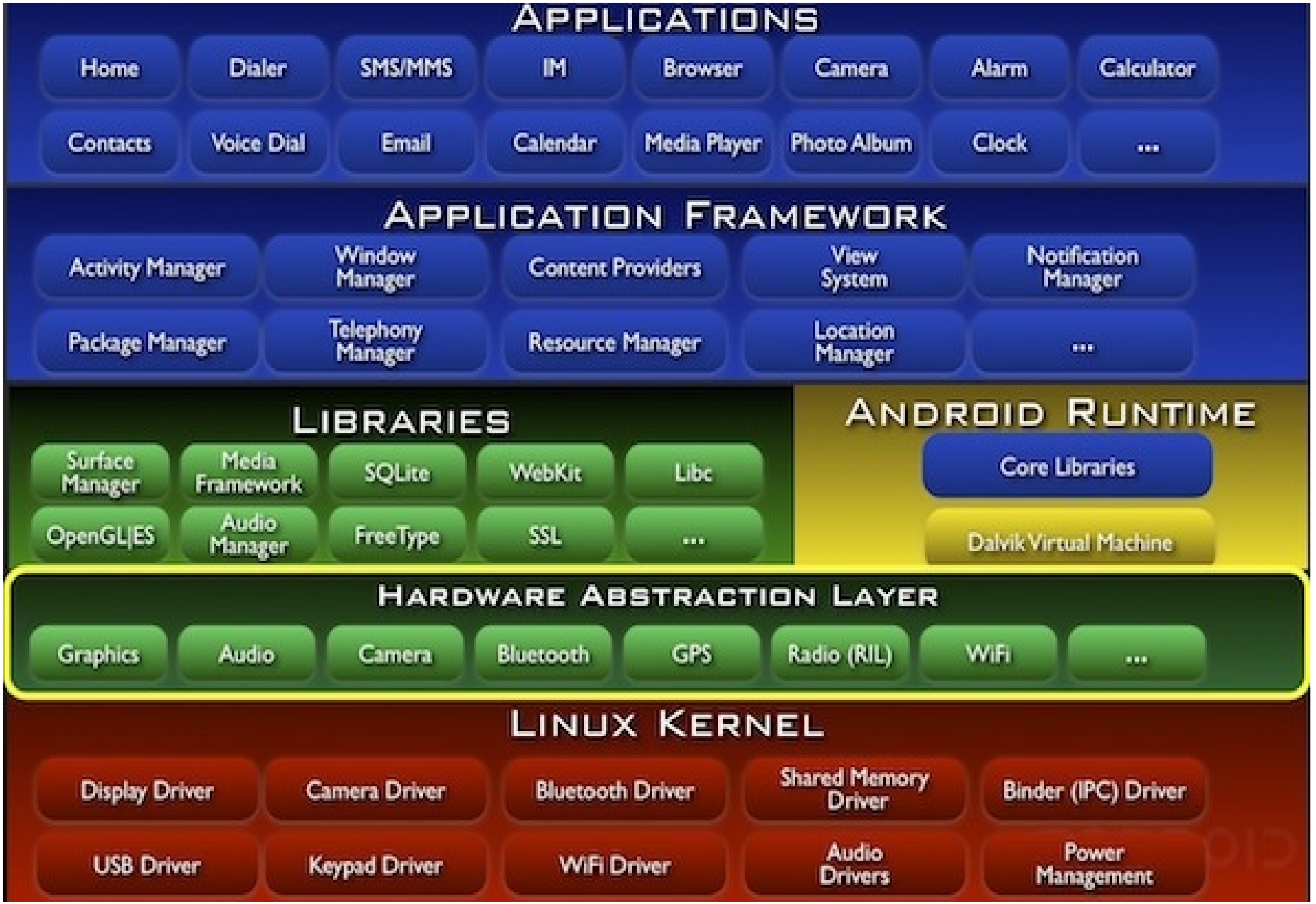
A little later Google was searching for partners to establish an improved secure/open platform for smartphones

Google acquired Android Inc. in 2005; key personnel continued development at Google on their OS

Google organized a group called Open Handset Alliance (announced in November 2007)
Google, LG U+, Samsung, T-mobile, Sony, HTC …

Rapidly improving since then, particularly security

Android OS



http://www.edureka.in/blog/beginners-guide-android-architecture/

Android OS

Provides the following services to the upper layers:

- hardware abstraction – via registration of device
- memory management – buddy, slabs, pmem, etc.
- security settings – SE for android, tomoyo, smack
- power management – by /sys/power/state (vfs)
  android aims to put system to sleep, apps can prevent this with a `WAKE_LOCK`
- device drivers – camera, bluetooth, wi-fi, audio, video
- file system support – ext4, fuse, fat, ntfs, ...
- network stack – socket buffers, protocols, etc.

http://www.edureka.in/blog/beginners-guide-android-architecture/

Android OS

Provides the following services to the upper layers:
(see `goldfish/drivers/staging/android`)

- binder – for interprocess communication (IPC)
  a remote method invocation system

  One process can call a function in another process and get a return object – ex: Window Manager ↔ client
  A process may get an object from one object and pass It on to another object
- logger – kernel msgs are saved to persistent memory
- ashmem – shared library support

http://www.edureka.in/blog/beginners-guide-android-architecture/

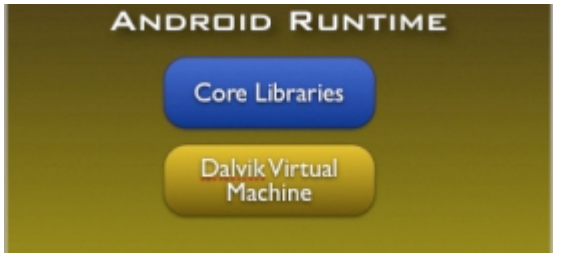
Android OS

Native libraries for handling different kinds of data:

- surface manager – composes windows on the screen
- SGL – 2D graphics
- OpenGL – 3D graphics
- media framework – codecs for playback/recording of audio/video formats
- freetype – fonts and font rendering
- WebKit – browser engine
- SQLite – database management
- OpenSSL – secure communication, certificates, etc.

http://www.edureka.in/blog/beginners-guide-android-architecture/

Android OS



Dalvik virtual machine runs the apps:

- Java class → bytecode → odex
- optimized for low memory usage
- allows several VMs to run efficiently simultaneously
- threads, memory management, process isolation are provided by the linux kernel
- register based instead of stack based
  bytecode: local variables are copied to/from stack
  odex: operations are directly applied to values in regs
- has its own library – does not use Swing, AWT, etc

http://www.edureka.in/blog/beginners-guide-android-architecture/

Android OS

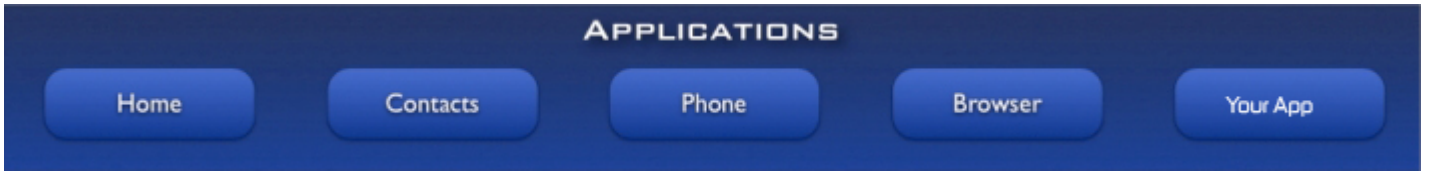
Manage basic phone functions:

(see `bin/frameworks/base/services/java/com/android/server`)

- Activity Manager – provides screen(s) in which user interacts with app (each is an activity)
- Content Providers – manage data sharing between apps
- Telephony Manager – manage voice calls
- Location Manager – GPS or cell tower locator
- Resource Manager – manage resources used by apps (memory, files, devices, locks, etc.)

http://www.edureka.in/blog/beginners-guide-android-architecture/

Android OS



Functions that are seen by normal users:

(see `bin/external/chromium_org...`)

- Phone - SMS/MMS client – texting/multimedia
- Phone - GSM
- Web Browser
- Contact Manager

http://www.edureka.in/blog/beginners-guide-android-architecture/

Android OS

Programming Model:

- Applications are written in Java but compiled to odex
- Native linux applications can be built and run
- Combined Java/C programs are possible with JNI

Android OS

Programming Model:

- Applications are written in Java but compiled to odex
- Native linux applications can be built and run
- Combined Java/C programs are possible with JNI

Android Applications:

- Apps are distributed in bundles as apk files
- These files are in effect zip files
- These files are self-contained (everything they need to run is contained in the bundle)

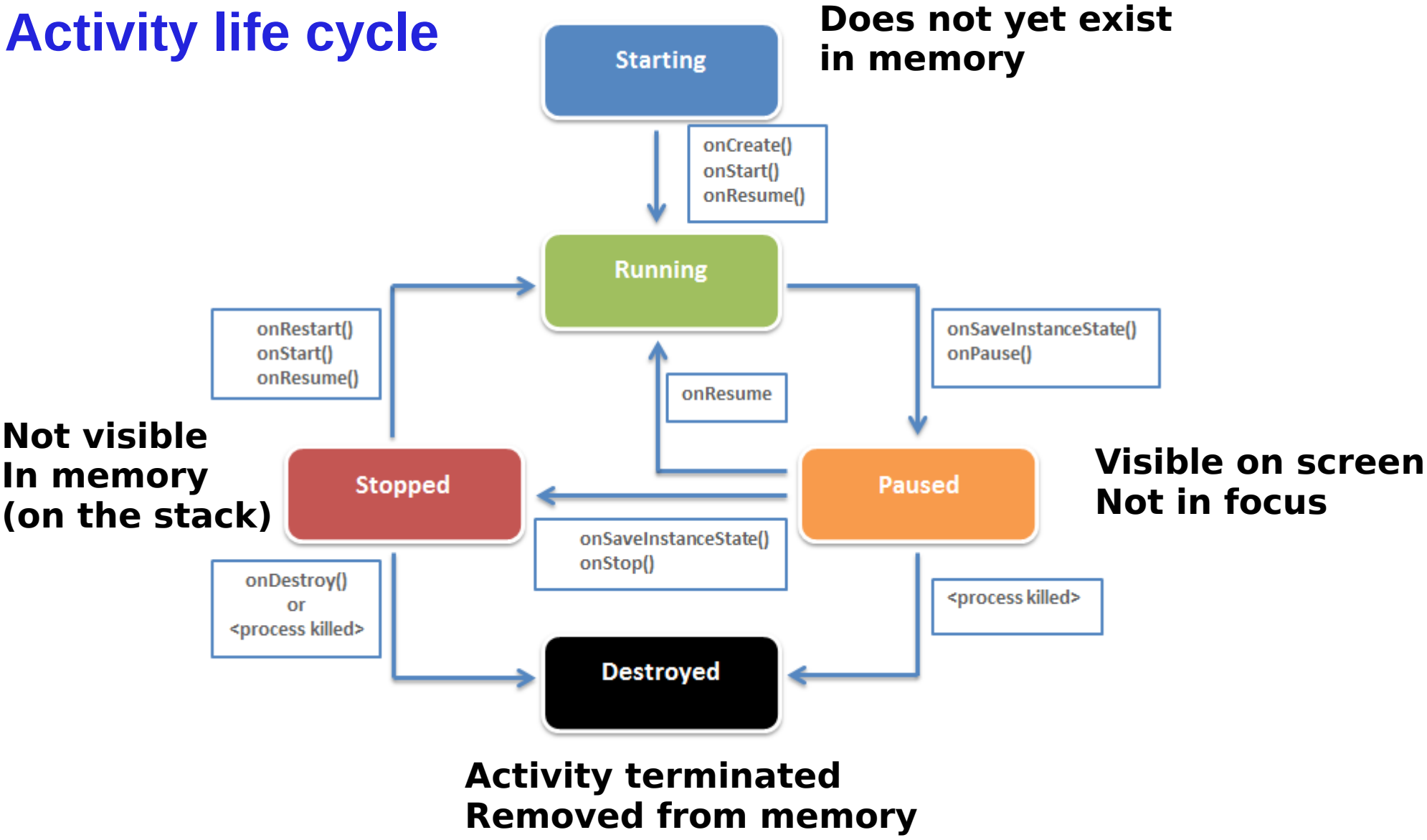
Android OS

Components:

- **Activity** - a single screen UI to interact with an app
  → everything in a UI is part of an activity
  → the only component that is directly visible to the user
- **Service** – a function running in the background
  → other components may “bind” to it
- **Broadcast Receiver** – receive notification of events
  → must register to receive specific notifications
- **Content Provider** – manage access to structured data
  → connect data in one process with code running in another process
  → encapsulate data
  → define data security

Android OS

Activity life cycle



Callbacks must be implemented to handle state changes
Each callback has a pre-defined behavior

Android OS

Intent

→ a message to communicate an action
→ contains a description of what should be done
→ are the communicators between Activities, Services and Broadcast Receivers
→ from one component an intent may request action on some other components

ex: an intent may be defined to communicate battery low to a user
ex: click on menu item in one Activity to switch to another

Android OS

Intent Object

→ contains all the information that needs to be conveyed
  information needed by the receiving component
    e.g. the action to be taken
  information needed by the Android OS
    e.g. description of the component category the intent is aimed at

→ consists of
- component name
- action
- data – URI of data including type (tel:<phone #>)
- category of the component that handles the intent
- extra – additional info to the receiver
- flags – to control the application of the action

Android OS

Explicit Intent

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

cause a second screen to open from a click on a button

Implicit Intent

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.edureka.in"));

startActivity(intent);
```

the component to be called is not specified
view the web page specified

Android OS

Intent Resolution

- Needed to determine the target component in implicit intents
- system compares contents of intent object against a set of intent filters (default descriptions)
- intent filters are a way for Android components to declare their capabilities: they define the collection of intent objects that they can receive
- the following is used to determine a match:
  - the action
  - the data type and URI
  - the category

Android OS

Service

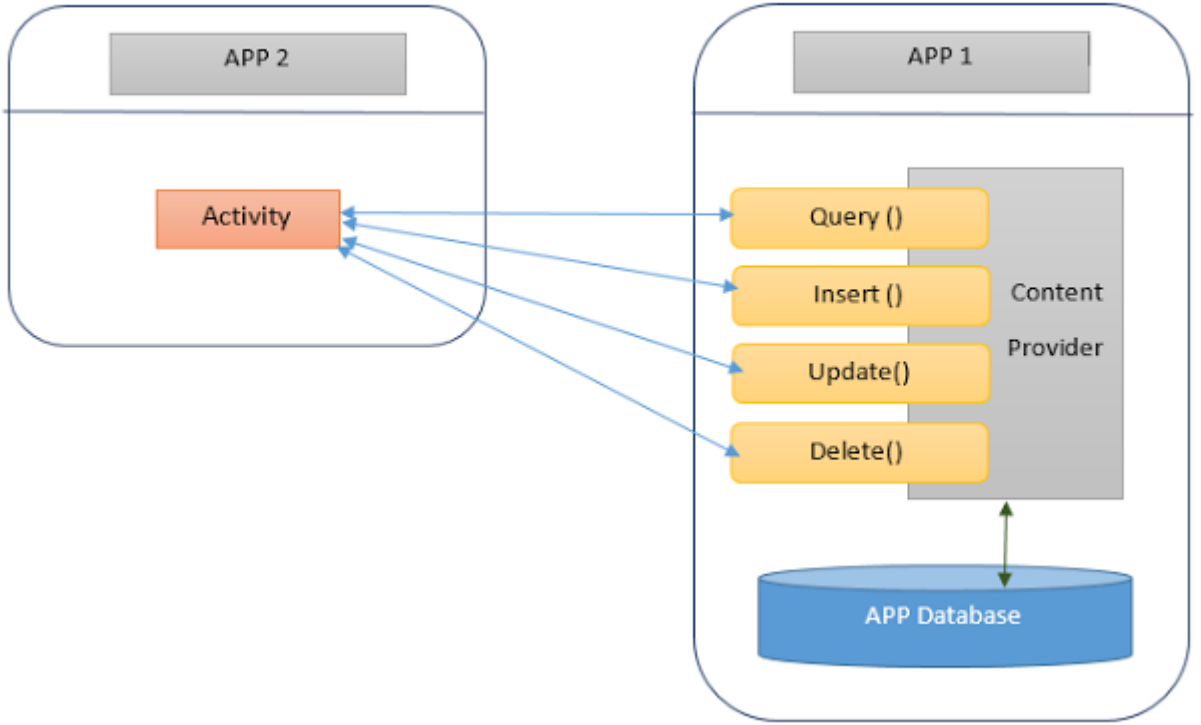
- component that runs in the background
- performs long-running tasks required by an application
- is unaffected by a switch in Activity state
- provides functionality to other applications ex: a music player, network download
- unbound: continues even if component starting it dies
- bound: service dies when component it is bound to unbinds it

Service life cycle

Android OS

Content Providers

- every app owns a data directory and protected memory
- content providers allow data sharing between apps
- is a set of data available using methods as shown below



Android OS

Broadcast Receiver

- pass a notification to the user in case a specific event occurs
- apps register for notification with a broadcast receiver
- many apps may register for the same event
  ex: you've got mail!

Event Listener

- used to capture events
- OnClick, OnTouch, OnDateSet

Android OS

Directory Structure:

- Directory structure is not fixed as in normal linux but the following are typical:

`/system/bin` – native utilities

`/system/xbin` – more native utilities

`/system/etc` – configuration files

`/system/apps` – location of apps as `apk` or `odex` files

`/system/framework` – application framework files (`jar`)

`/storage/sdcard` – accessible outside of the phone (`rw`)

`/data` – applications data (`rw`)

`/proc` – `proc` file system

`/sys` – `sys` file system

`/proc/crypto` – encryption/signing schemes

Android OS

Security Philosophy:

- Limited time and resources
- Humans have difficulty understanding risk
- It is reasonable to assume that
  - → Most developers do not understand security
  - → Most users do not understand security
- Security philosophy cornerstones
  - → Need to prevent security breaches from occurring
  - → Need to minimize the impact of a security breach
  - → Need to detect vulnerabilities and security breaches
  - → Need to react to vulnerabilities and security breaches swiftly

Android OS

Prevention of breaches:

- Android is open source and widely used so it cannot depend on obscurity for security
- Security design and tools are developed by several groups including Google, iSecPartners, NSA,...
- **Among useful prevention devices:**
  - → SE for Android... (mandatory access control)
  - → ProPolice stack overflow protection from IBM
  - → safe-iop macro library for C
- **Android Over-The-Air update system**
  - → security updates can be installed as soon as available
  - → user interaction is not necessary
  - → a user need not use any additional hardware to acquire the updates

From notes by Prabhaker Mateti

Android OS

Minimizing the impact of breaches:

- **Vulnerabilities and security breaches occur**
  - → users install malware, for example
  - → code may contain exploitable bugs (even the OS)
- **Contain the damage**
  - → typically there is one android user per system
    Linux can support many users
    Give every running app a user id and use the normal Linux permissions system to prevent one app from trampling on another
  - → apps can still talk to one another but checks can be made before completing a transaction
  - → this provides a sort of privilege separation that is transparent to the developer

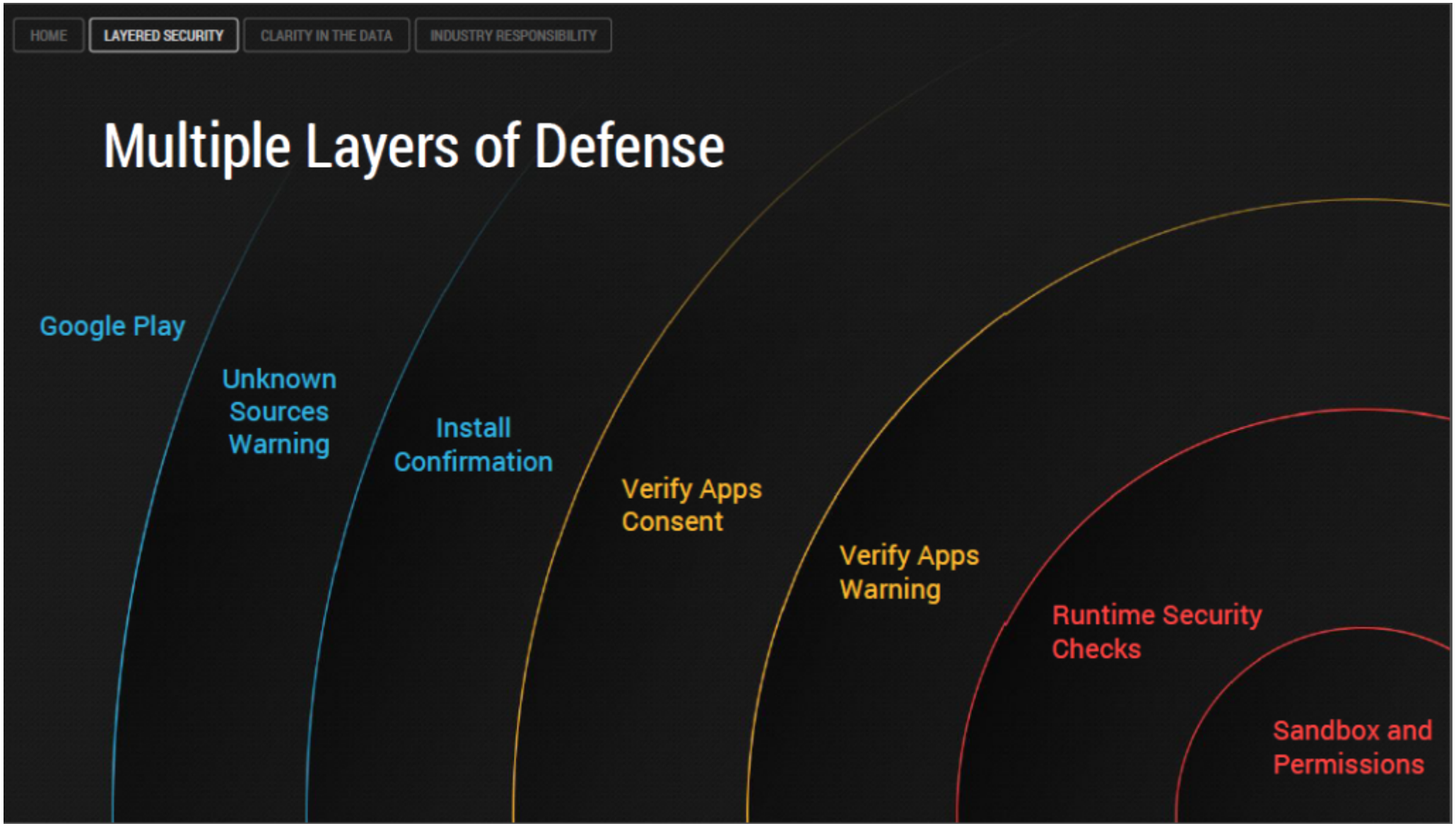
From notes by Prabhaker Mateti

Android OS

Detecting a breach:

- **Outside tools**
  - → audits
  - → simulate attacks
  - → fuzzing – unexpected data input to programs
- **Internal tools**
  - → activity monitors
  - → event logs

Android OS



http://qz.com/131436/contrary-to-what-youve-heard-android-is-almost-impenetrable-to-malware/

Android OS

Android platform security architecture summary:

- **Linux level security**
  - → logging
  - → process isolation via permissions and MAC application sandboxing
- **Native libraries**
  - → ssl encryption, authentication, integrity
  - → application signing so that developer is accountable for breaches
- **Application framework**
  - → application defined, user granted permissions

From notes by Prabhaker Mateti

Android OS

Android platform security basics:

- **Applications have no permissions by default**
- **Permissions may be added from a list, examples:**
  - → `ACCESS_NETWORK_STATE`
  - → `ACCESS_SURFACE_FLINGER`
  - → `WRITE_SMS`
  - → complete list:
    `http://developer.android.com/reference/android/Manifest.permission.html`
- **Permission declarations are static**
  - → user is prompted for approval upon installation
  - → permissions cannot be granted at runtime
  - → permissions are obtained via `uses-permission` tags

    ex: `<uses-permission android:name= "android.permission.RECEIVE_SMS" />`

Android OS

Android platform security extras:

- ARM “eXecute Never” (XN) protection against executing from the stack or heap
- ProPolice canaries to prevent stack/buffer overflows
- safe-iop macro library to pre-check over/underflow In integer operations
- `dlmalloc` extensions to prevent double free() vulnerabilities and heap exploits
- OpenBSD `calloc` to prevent integer overflows during memory allocation
- Linux `mmap_min_addr()` to mitigate null pointer dereference privilege escalation

From notes by Prabhaker Mateti

Android OS

Android platform system partition:

- Contains the bottom three layers: linux kernel, native Libraries, android runtime, and application framework
- Is set to be read only

When a user boots the device into Safe Mode:

- only core android applications are available
- no third-party applications are allowed to run

Android OS

OS protected APIs:

user grants permission for 3rd party apps to use

- **Cost-sensitive**:
  - → Telephony – end-to-end encryption
  - → SMS/MMS – lock out your friends
  - → Network/Data connections – built-in VPN, ssh app
  - → In-app billing – see best practices
  - → Near Field Communication access – 4cm or less
- **Sensitive data input devices**
  - → Location (GPS)
  - → Camera functions
  - → Microphone
- **Bluetooth**
- **Personal information** – screen lock, encryption, look at app permissions

Android OS

Interprocess Communication (IPC):

- **Existing in Linux**:
  - → file systems – e.g. proc, sys
  - → sockets – socket buffers encapsulate/carry content
  - → signals – e.g. semaphores
- **New in Android**:
  - → Binder – remote calls to other processes or between processes via a custom Linux driver
  - → Services – interfaces accessible through binder
  - → Intents – message object indicating an “intention” to have some service rendered
  - → Content providers – interfaces that connect data in one process with code running in another process provide mechanisms that define data security

Android OS

Application Signing:

- **All applications must be digitally signed**:
  - → with valid certificate whose private key is held by developer – validity checked before installation
  - → intended to establish trust of the application
  - → self-signed certificates are typical, no CA needed
  - → app cannot be installed if not signed
- **Why are self-signed certificates allowed?**
  - → OK since the market ties apps to known developers
  - → Notable failures with CAs have occurred
- **Updates and code sharing:**
  - → to update, certificates of old and new must match
  - → an app may expose functions to another app if
  - → apps signed by same certificate can run in same process – allows modular development

Android OS

User IDs and file access:

- **Each package is assigned a unique Linux UID**:
  - → but there is no password file for these!
  - → and different devices may have different UIDs for the same package
  - → the creator of the package is the owner
- **If two packages are signed with same certificate**
  - → they can share the same UID and therefore run in the same process

From notes by Prabhaker Mateti

Android OS

Application sandbox:

- **Principles**:
  - → separation of processes
  - → separation of file permissions
  - → authenticated IPC only
- **Each application**
  - → runs in its own Linux process
  - → has its own UID
  - → runs in its own Dalvik VM
- **Native code and system applications**
  - → are also sandboxed

Android OS

Application sandbox considerations:

- **Place access controls close to the resource**
  - → not in the VM
  - → it is easier to protect that way
- **Lock down user access for a default application**
  - → default Linux applications typically have too much power
  - → but locking down limits innovation
  - → however users generally make poor security choices

From notes by Prabhaker Mateti

Android OS

File system encryption:

- **May be enabled by the user**
- **Notes:**
  - → uses 128 AES with CBC and ESSIV:SHA256
  - → master key is a 128 bit random number taken from `/dev/urandom`
    key is encrypted with hash of password and salt, also from `/dev/urandom`, using PBKDF2 function from the openssl library
  - → the salt is kept in a user inaccessible part of `/data`
  - → every sector of the device is encrypted
  - → system checks that battery is fully charged before encrypting!

From notes by Prabhaker Mateti

Android OS

Sim Card Access:

- **What is a SIM card and what is it for?**
  - → a subscriber identity module
  - → identifies you to your carrier
  - → can be placed in several GSM phone bodies keeping phone #, data plan, etc.
  - → but is good for one carrier
- **No low-level access to a SIM card by 3rd party apps**
  - → OS handles all communication with the SIM card including information flow to/from personal info on behalf of an application
- **No low-level invocations of AT commands allowed**

From notes by Prabhaker Mateti

Android OS

Some early exploits:

KillingInTheNameOf:

protection for /dev/ashmem (shared memory) buggy
can be re-mapped to `R/W` permissions
allows user to root the phone

Exploid:

netlink establishes kernel/userspace communication
udev did not check who is doing it
user tricks udev into running any binary as root

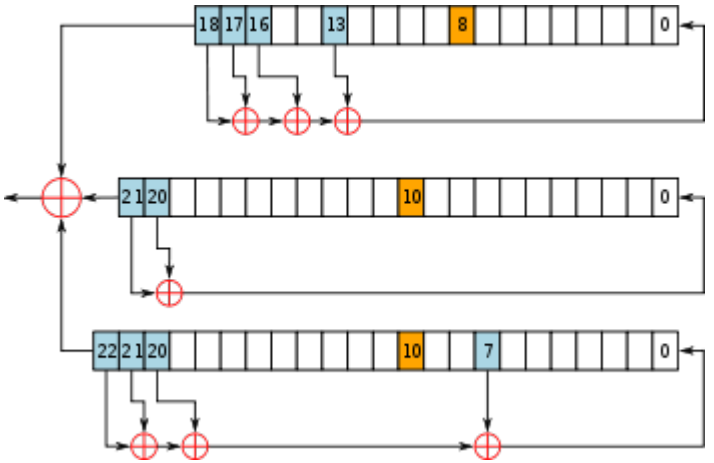
RageAgainstTheCage:

`adb` daemon `setuid` return value was not checked
Starts as root then switches to `uid`
User can cause that to fail to keep daemon running as root
`adb` shell is then available to user as root

Android OS

Attacks on transmission protocols:

- **GSM**
  - → largest mobile network – 3.8 billion phones
  - → see picture below for stream cipher A5/1
  - → can be broken with known plaintext attack
  - → acquire plaintext: send text message to phone to get subscription info and hardware ID
    Place a call and the call security exchange sends the same (known) plaintext as part of the set up



From notes by Prabhaker Mateti

Android OS

Attacks on transmission protocols:

- **SMS**
  - → used to send text messages – 160 character limit (140 byte limit)
  - → GSM is on 2 bands – 1 for control, 1 for data
  - → SMS operates only over the control band
  - → high volume text messaging can disable GSM

  - → SMS transmissions may pass through many different networks, even the internet, and its addresses can be spoofed

Android OS

Attacks on transmission protocols:

- **MMS**
  - → multimedia messaging service (videos, pics, etc.)
  - → extends SMS for longer length messages
  - → Vulnerable to attack on battery power!
  - → Attacker sends a message to a phone which is in the usual dormant state. The network pages the phone, waking it up, and requiring it to find its location. Lots of messages causes the phone to always be out of the dormant state, draining the battery!
  - → luckily, does not make phone useless and does not expose data – just annoys the victim

From notes by Prabhaker Mateti

Android OS

Attacks on transmission protocols:

- **Bluetooth**
  - → short distance file transfers between devices
  - → authentication is not required
  - → devices are often set to be discoverable
  - → bluesnarfing – theft of info due to misprogrammed software, no longer a threat
  - → bluejacking – send unsolicited messages such as pictures or advertisements to a victim
  - → bluebugging – take control over a victim's phone due to firmware issues in older phones
  - → Denial of Service – attacker keeps asking to pair draining the device's battery

Android OS

Mistakes:

- **Private data can be taken from the log file**
  - → use logcat to see the system log
  - → the log can contain email addresses, passwords, contact information, SMS info, bluetooth address, Wi-Fi MAC address...
  - → Malware can use `Runtime.getRuntime().exec("logcat")` and set `READ_LOGS` permission in the manifest
  - → Fixed by not allowing read of logs due to other apps

- **Private data is broadcast to other apps via IPC**
  - → SMS messages are sent as broadcast intents and can be read by apps with `READ_SMS` permission
  - → SMS transmissions are unencrypted and weakly authenticated

Android OS

Mistakes:

- **Intent spoofing**
  - → attacker forges an intent to cause malicious behavior
  - → victim has class expecting intents from same app
    victim exports the class: so any app can send to it
    victim's class blindly trusts whatever it gets
- **Leaking permission-protected data**
  - → app sends data to a client via IPC with permission
    protection but fails to ensure that the client applies
    the same protection to the data
- **Use IPC input to run activities w/o checking**
  - → attacker may cause nefarious activities to execute

From notes by Prabhaker Mateti

Android OS

Mistakes:

- **No null checks on intents from other applications**
  - → many apps do not make null checks on IPC input
  - → a null dereference can cause an app to crash
  - → may result in a denial of service

- **Many more here:**
  `http://developer.android.com/training/articles/security-tips.html`