howest
hogeschool

Mobile Security

Week 1: Android - ADB - ROOT

Contents of this course

- Mobile application security
- Mobile device security
- Mobile application development
- ADB
- Rooting
- Static analysis
- Dynamic analysis
- Malware

Evaluation

- Combination of methods:
  - Theoretical exam
  - Project work in group
    - With peer evaluation
  - Individual and group assessment during oral exam
    - About **group** project and/or **individually** made labs
- Work on assignment during classes
- See guidelines for all the information

howest
hogeschool

Overview guidelines assignment

howest
hogeschool

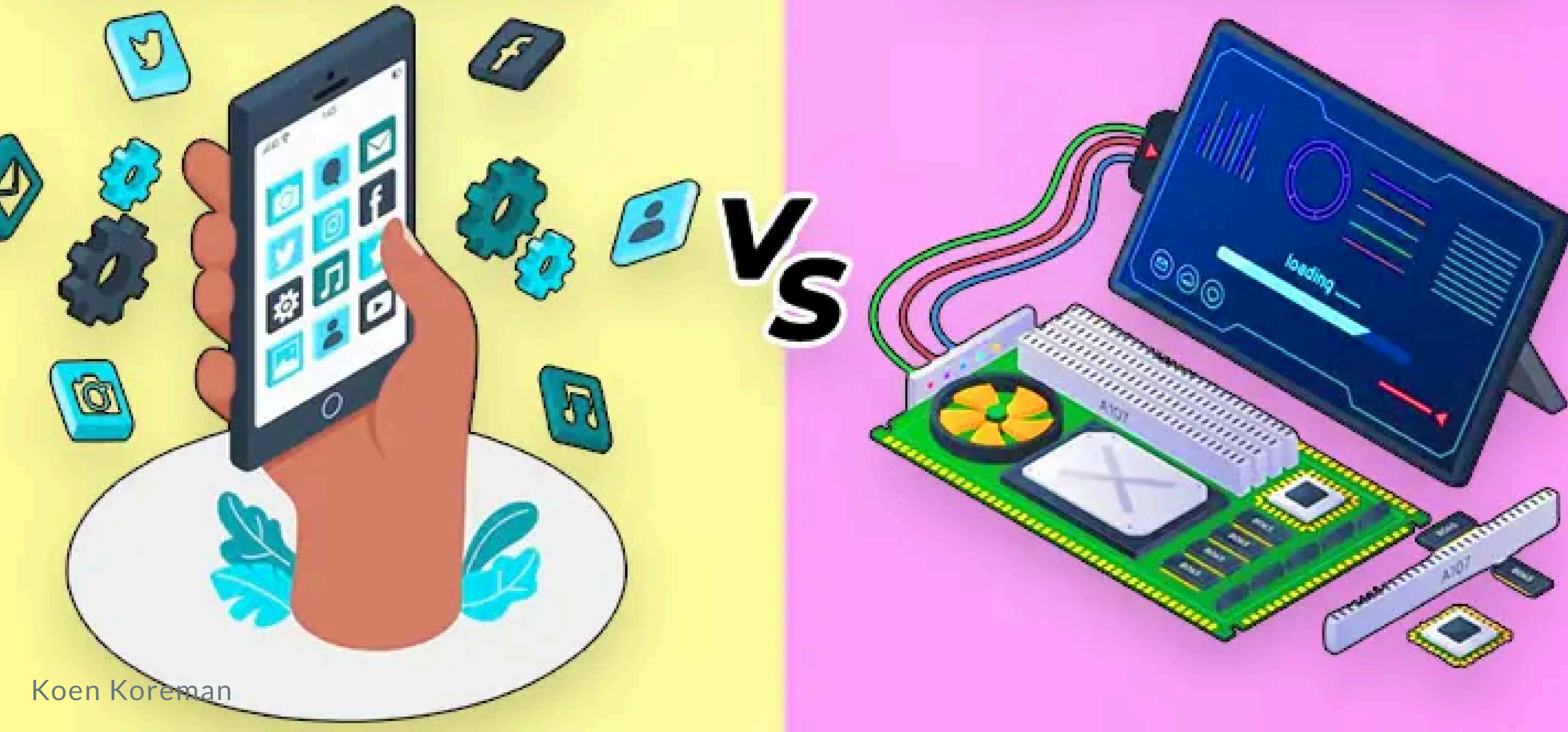
Introduction to Mobile Security

What is Mobile Security?

Mobile Security is divided into 3 parts.

- Hardware layer (modem, ...)
- OS layer (filesystem, ...)
- Software layer (applications, ...)

Software vs Hardware



Koen Koreman

Mobile Application Pentesting

- Lock Screen Security
- Encryption
- App Permissions
- Secure Working Frameworks

Mobile Application Pentesting

The mobile application pentesting methodology concentrates on the following items:

- Client-side safety
- File system
- Hardware
- Network security
- Application security

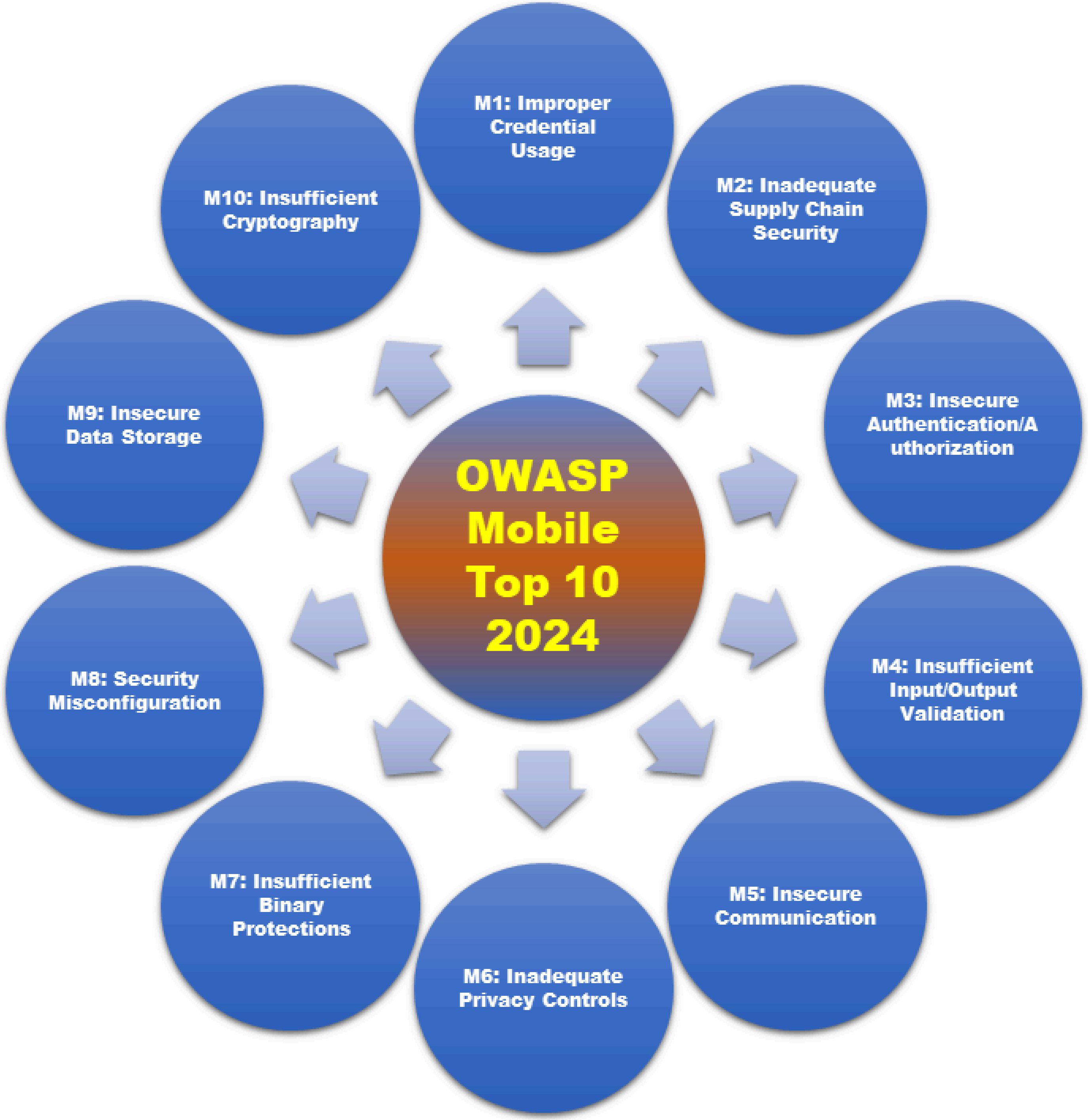
Many of the mobile app bugs are web bugs in disguise.

Key areas in Mobile App Security

- Local Data Storage
- Interaction with the Mobile Platform
- Code Quality and Exploit Mitigation
- Communication with Trusted Endpoints
- Authentication and Authorization

howest hogeschool

OWASP Mobile Top 10


OWASP Mobile Top 10 2024

M1: Improper Credential Usage

M2: Inadequate Supply Chain Security

M3: Insecure Authentication/Authorization

M4: Insufficient Input/Output Validation

M5: Insecure Communication

M6: Inadequate Privacy Controls

M7: Insufficient Binary Protections

M8: Security Misconfiguration

M9: Insecure Data Storage

M10: Insufficient Cryptography

Koen Koreman

OWASP Mobile Top 10

Vulnerabilities that didn’t make the place on the initial release list, but in the future, OWASP might consider them.

- Data Leakage
- Hardcoded Secrets
- Insecure Access Control
- Path Overwrite and Path Traversal
- Unprotected Endpoints (Deeplink, Activitity, Service …)
- Unsafe Sharing

howest
hogeschool

Mobile Operating Systems

vs

Koen Koreman

Types of Operating Systems

- iOS
- Android
- Windows 10 (11)
- Windows 10 Mobile (maintenance only)
- Blackberry OS (maintenance only)
- Tizen OS (Samsung)
- Chrome OS (for devices)
- watchOS (Palm/HP)

howest
hogeschool

Mobile Applications

Progressive Web Apps

VS

Native Apps

What’s the Difference?

Types of applications

- Native
- Progressive Web Apps
- Hybride

Native applications

Native apps are specific written for an OS. Usually don't interact with internet or web views.

iOS = Objective C or Swift
Android = Java, Kotlin, C(++), …

- Faster then other apps
- All device functionality possible
- Native look and feel OS

Native applications

Examples of Native applications:

- Clock
- Calculator
- File Manager

Hybride applications

Hybride apps are a combination of native views and web (PWA) views.

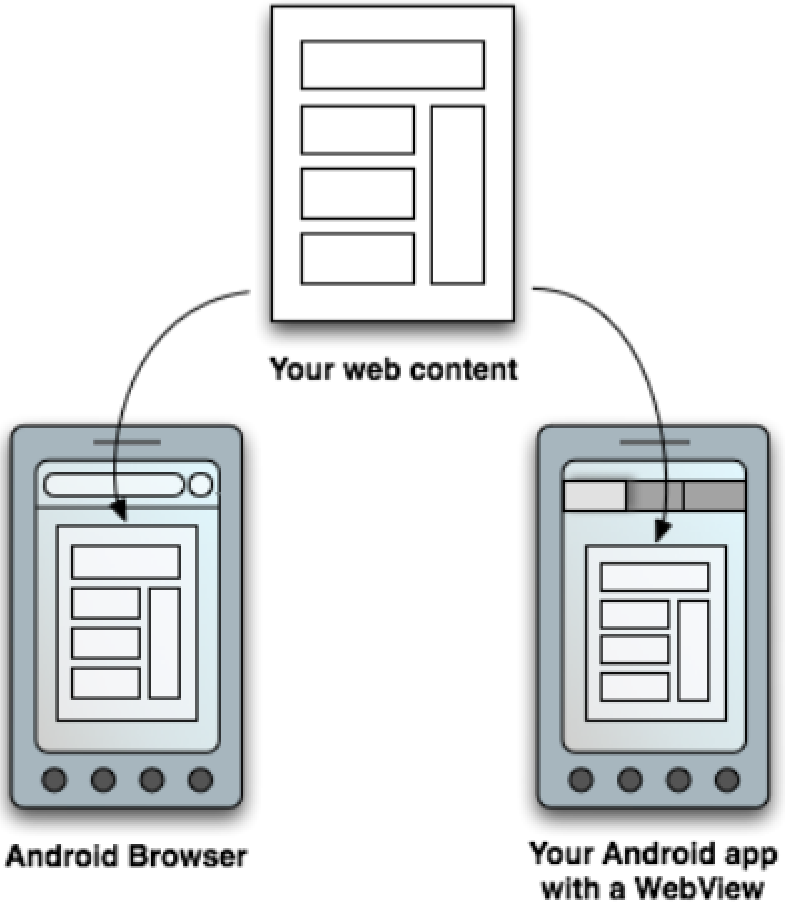
They can be installed like a native app, but they use web standards like PWAs, although they can also be used offline.

WebView

WebView objects allow you to display web content as part of your activity layout, but lack some of the features of fully-developed browsers.

A WebView is useful when you need increased control over the UI and advanced configuration options that will allow to embed web pages in a specially-designed environment for the app.

WebView


Your web content
Android Browser
Your Android app with a WebView

Hybride frameworks

- Apache Cordova (Phonegap from Adobe)
- Xamarin (Microsoft)
- React Native (Facebook)
- Flutter (Google)

Hybride frameworks

Some examples:

- WhatsApp (React Native)
- Spotify (React Native)
- Google Pay (Flutter)
- Ebay (Flutter)
- …

PWA

Progressive web apps

- A website that looks, feels and behaves like a mobile app
- Can take advantage of native features though HTML5 API's
- No need for app store
- Installed through web browser

PWA

Advantages:

- No need to develop for different platforms
- Can leverage on your existing HTML, CSS and JavaScript knowledge

PWA

Disadvantages:

- Not available through app stores
- No access to all native features

PWA

More information:

https://developer.mozilla.org/en-US/docs/Web/Apps/Progressive/Advantages

howest
hogeschool

Mobile Pentesting

Start Mobile Pentesting

Step 1: intelligence gathering is the most significant step in a penetration test.

- Architecture understanding
- Mapping the Application

Step 2: assessment of the intelligence with the correct tools

- File system analysis
- Package analysis
- Reverse engineering

Vulnerability Analysis

Vulnerability analysis is usually the process of looking for vulnerabilities in an app.

- Manually
- Automated scanners

Static and dynamic analysis are types of vulnerability analysis.

Static Analysis

During static analysis, the mobile app's source code is reviewed to ensure appropriate implementation of security controls.

- Manual code review
- Automated Source Code Analysis

Dynamic Analysis

The main objective of dynamic analysis is finding security vulnerabilities or weak spots in a program while it is running.

Dynamic analysis is conducted both at the mobile platform layer and against the back-end services and APIs, where the mobile app's request and response patterns can be analyzed.

Dynamic analysis is usually used to check for security mechanisms that provide sufficient protection against the most prevalent types of attack.

Standard Web Bugs

- SQL Injection
- Direct object reference
- Improper authentication/authorization
- Insecure uploads

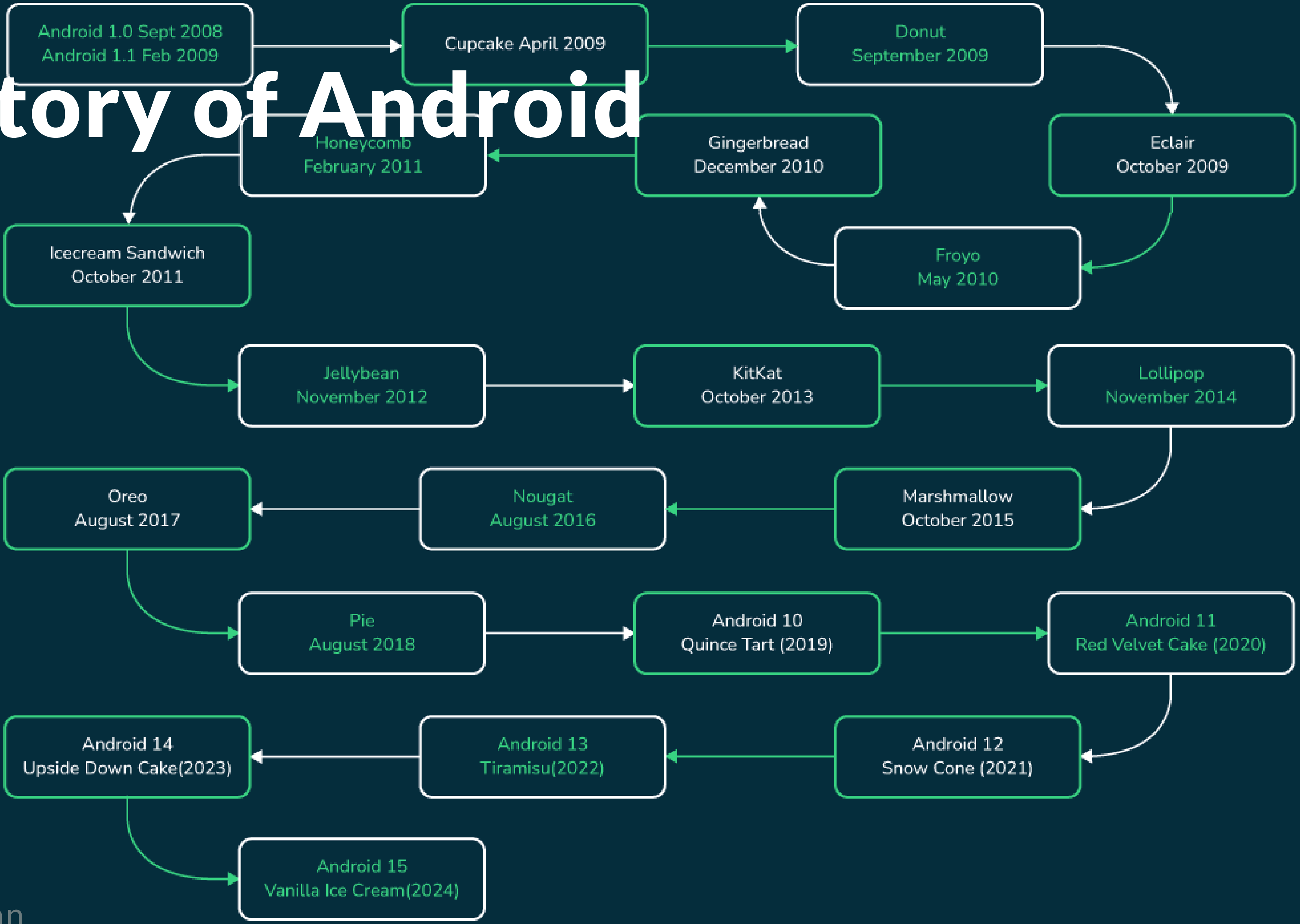
Android

Android

Android Security Architecture

- Android is a Linux-based open source platform
- Android's software stack is composed of several different layers. Each layer defines interfaces and offers specific services.

History of Android



Android 1.0 Sept 2008
Android 1.1 Feb 2009

Cupcake April 2009

Donut
September 2009

Eclair
October 2009

Froyo
May 2010

Gingerbread
December 2010

Honeycomb
February 2011

Icecream Sandwich
October 2011

Jellybean
November 2012

KitKat
October 2013

Lollipop
November 2014

Marshmallow
October 2015

Nougat
August 2016

Oreo
August 2017

Pie
August 2018

Android 10
Quince Tart (2019)

Android 11
Red Velvet Cake (2020)

Android 12
Snow Cone (2021)

Android 13
Tiramisu(2022)

Android 14
Upside Down Cake(2023)

Android 15
Vanilla Ice Cream(2024)

Koen Koreman

History of Android

Android dates back to 2003 when Andy Rubin, Rich Miner, Nick Sears, and Chris White co-founded a start-up Android Inc.

Google could sense the potential the product carried within and sealed a deal worth $50 Million to acquire Android in 2005.

The first public Android Beta Version 1.0 was finally published on 5th November 2007

Early Android (1.0–2.3 Gingerbread)

2008–2011: Android 1.0 debuted on the HTC Dream with Google apps and a new app market.

- Cupcake introduced third-party keyboard support and video recording.
- Donut brought CDMA support and better screen compatibility.
- Eclair, Froyo, Gingerbread: Added navigation, multi-touch, mobile hotspot, and NFC. Focus on UI, security, and battery improvements.

Tables and UI Evolution

3.x Honeycomb, 4.x Ice Cream Sandwich & Jelly Bean

- 2011–2013: Honeycomb focused on tablets with a virtual button interface and enhanced multitasking.
- Ice Cream Sandwich united phones/tablets and included Face Unlock and gesture navigation.
- Jelly Bean: Smoother UI, expandable notifications, Daydream screensavers, and Android Beam sharing.

Modern Design & Performance

4.4 KitKat–6.0 Marshmallow

- 2013–2015: KitKat launched "Ok Google" and Hangouts integration.
- Lollipop introduced Material Design, Android TV, and major battery enhancements.
- Marshmallow debuted fingerprint unlock, Doze mode for battery, Android Pay, and granular app permissions.

Personalization & Smarter Features

7.0 Nougat–v10

- 2016–2019: Nougat enabled split-screen multitasking, notification replies, and Google Assistant.
- Oreo: Autofill, picture-in-picture, notification dots, and faster boot times.
- Pie introduced gesture navigation, AI-powered brightness, and wellness features.
- Android 10: Dark mode, gesture navigation, Live Captions, and a new privacy focus. Sweets-based naming dropped.

AI, Privacy, and Device Flexibility

v11–v13

2020–2024:

- Android 11: Conversation bubbles, screen recording, smart home controls.
- Android 12: "Material You" design, customizable themes, improved privacy dashboard, and optimized for foldables.
- Android 13: More dynamic themes, per-app languages, spatial audio, and better foldable/tablet support.

AI, Privacy, and Device Flexibility

v14–v15

- Android 14: Lock screen customization, satellite messaging, app cloning, Health Connect integration.
- Android 15: AI-powered Theft Detection, Private Space, partial screen sharing, app archiving, high-quality webcam support, enhanced multitasking, and PDF handling.

Android 16

New features in Android 16, designed to boost productivity and enhance security.

- Usability Enchancements
- Multitasking tools
- Accessibility
- Changes in security and privacy

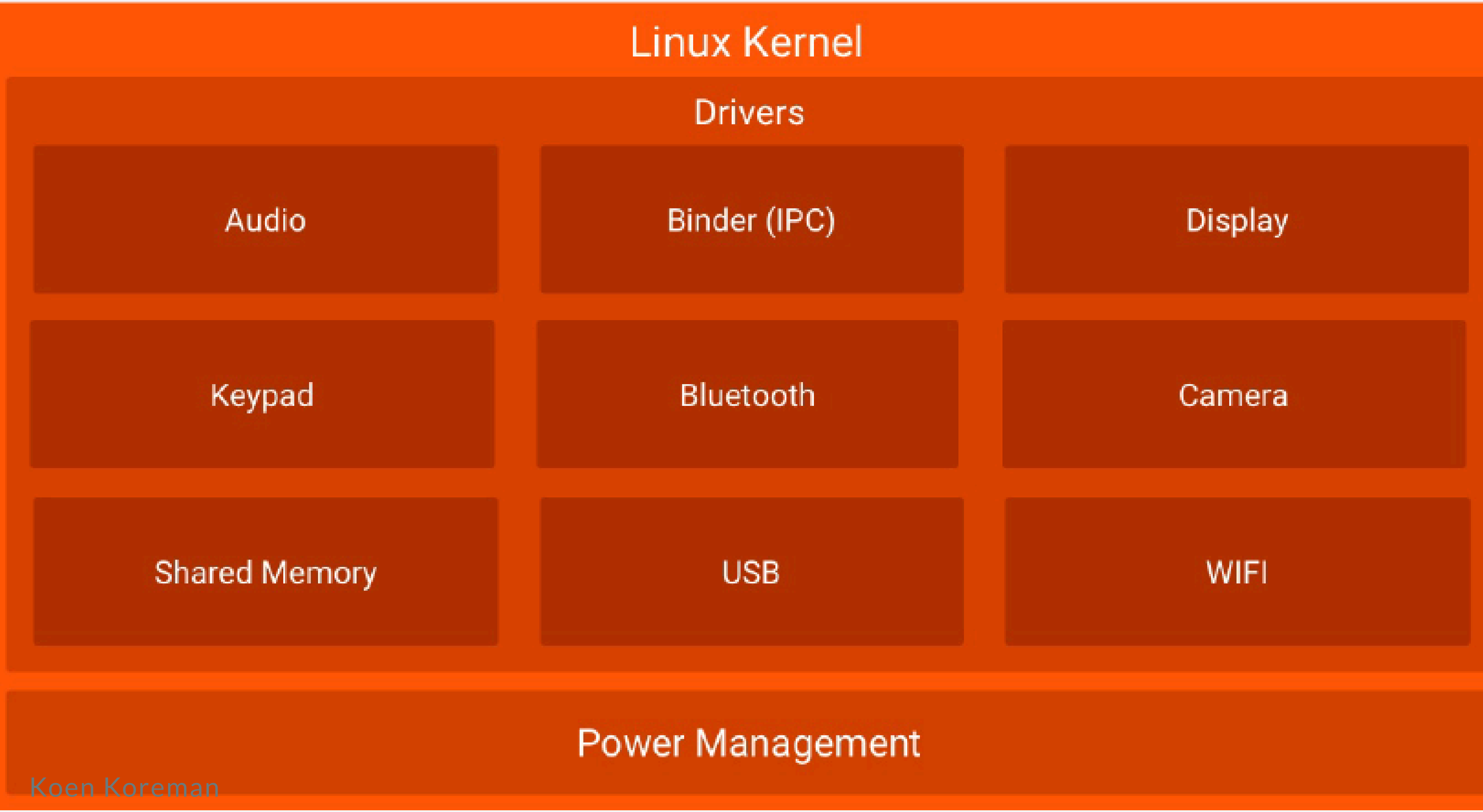
Release date: June 10, 2025

Android

At the lowest level, Android is based on a variation of the Linux Kernel.

For example, the Android Runtime (ART) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management.

Using a Linux kernel lets Android take advantage of key security features and lets device manufacturers develop hardware drivers for a well-known kernel.



Linux Kernel

Drivers

Audio	Binder (IPC)	Display
Keypad	Bluetooth	Camera
Shared Memory	USB	WIFI

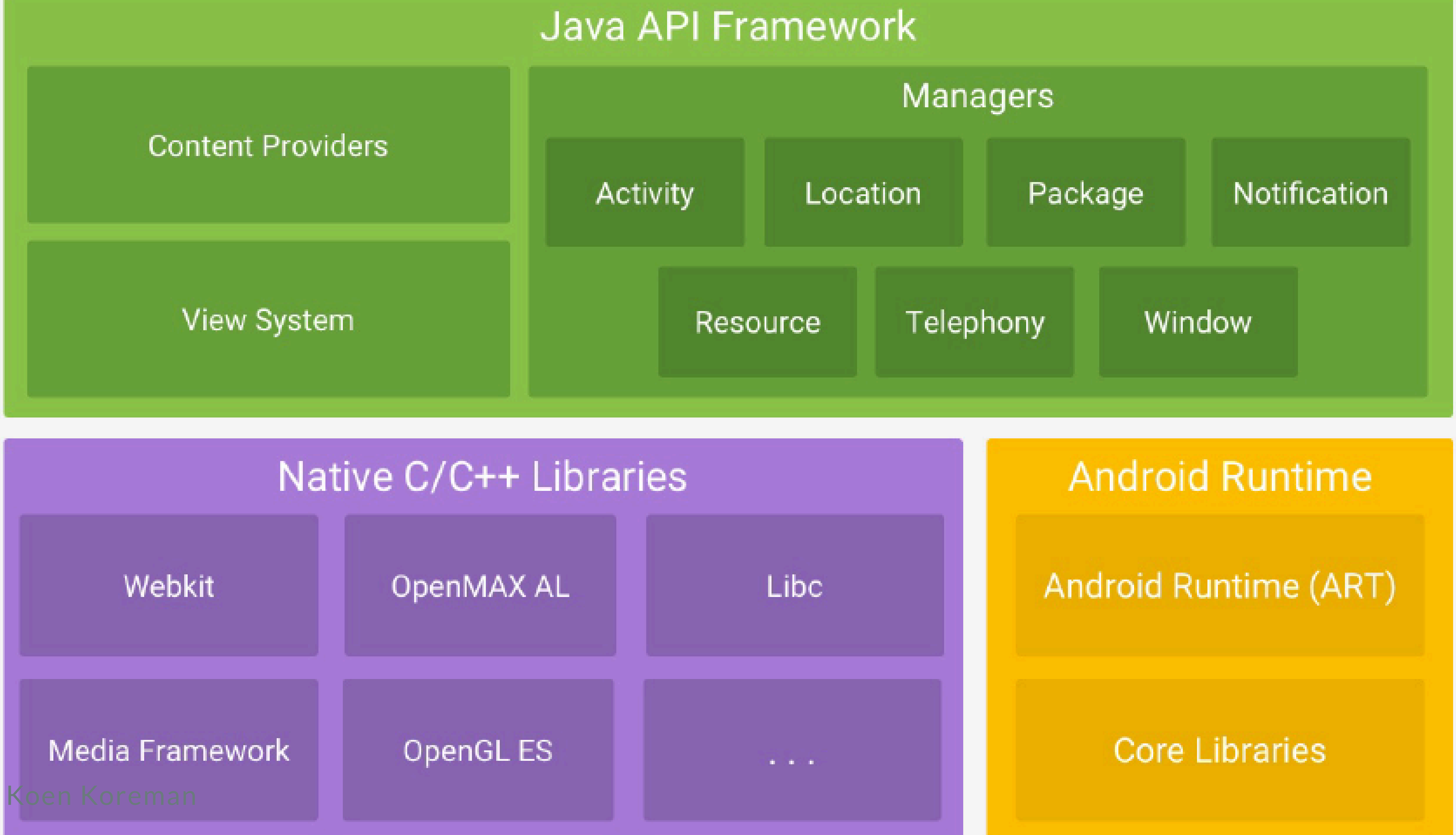
Power Management

Android

The Hardware Abstraction Layer (HAL) defines a standard interface for interacting with built-in hardware components.

The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component.

When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.



Java API Framework

Content Providers

View System

Managers

Activity

Location

Package

Notification

Resource

Telephony

Window

Native C/C++ Libraries

Webkit

OpenMAX AL

Libc

Media Framework

OpenGL ES

. . .

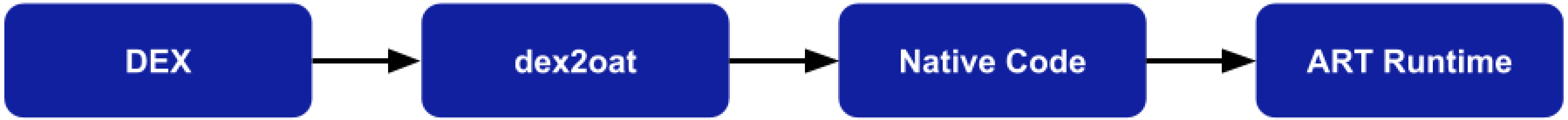
Android Runtime

Android Runtime (ART)

Core Libraries

Android ART

Android executes this bytecode on the Android runtime (ART).



Android ART

ART, in addition to using Just-in-time (JIT) compilation like Dalvik, also uses Ahead-of-time (AOT) compilation and Profile-guided compilation, a combination of these compilation modes will help Android apps Improve performance compared to using Dalvik.

Android ART

Just In Time (JIT): With the Dalvik JIT compiler, each time when the app is run, it dynamically translates a part of the Dalvik byte-code into machine code.

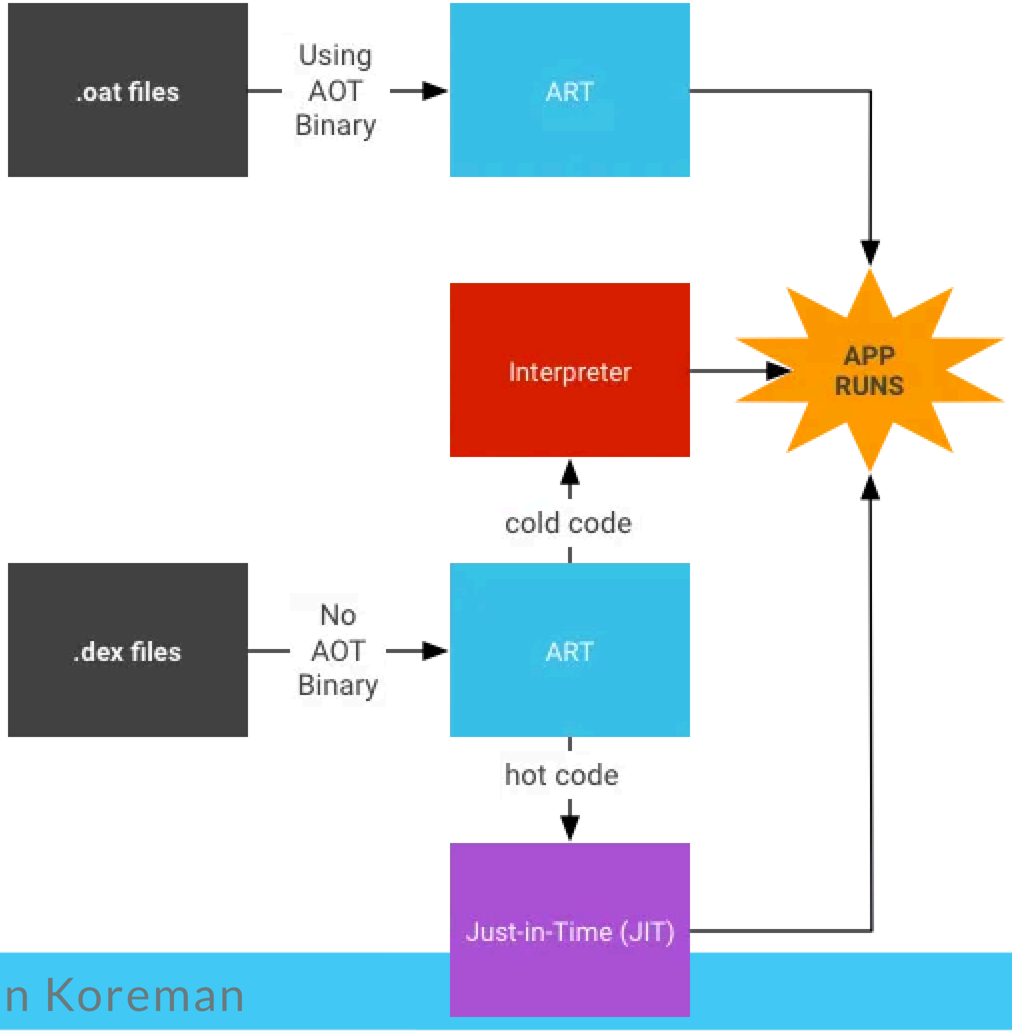
As the execution progresses, more byte-code is compiled and cached. Since JIT compiles only a part of the code, it has a smaller memory footprint and uses less physical space on the device.

Android ART

Ahead Of Time (AOT): It's a compiler before app executes, it statically translates the DEX byte-code into machine code and stores in the device's storage. It is run in the background.

Profile-guided compilation is a compiler optimization technique in computer programming that uses profiling to improve program runtime performance.

Android ART


.oat files
Using AOT Binary
ART
Interpreter
APP RUNS
cold code
.dex files
No AOT Binary
ART
hot code
Just-in-Time (JIT)

Android ART

After installing APK, the first few times the application runs with the Interpreted and JIT mechanisms.

The interpreter works with the mechanism of translating each line in the input file, then immediately executes that output stream, and then keeps iterating the translation process with the next lines.

Android ART

When the application executes, lots of methods called. If this time it runs with JIT, the application takes longer to start (because it translates into large blocks), so it is faster to use Interpreted now (because it can translate the line that is executed immediately).

When it has passed the start-up stage, the application will run both of them. The first part of the method will run with Interpreted, at the same time JIT will also translate that method. When JIT is finished translating, the Interpreted stops. The method which has been translated by JIT will run quick.

Android ART

When device is idle or charging, AOT compilation (dex2oat) daemon compiles frequently used methods (based on configuration from first few times running) to machine code and stores them in a .oat file.

When running with machine-code in the .oat file, it will be faster because it takes no time to translate dalvik byte-code ➡️ machine-code.

APK

Android - APK files

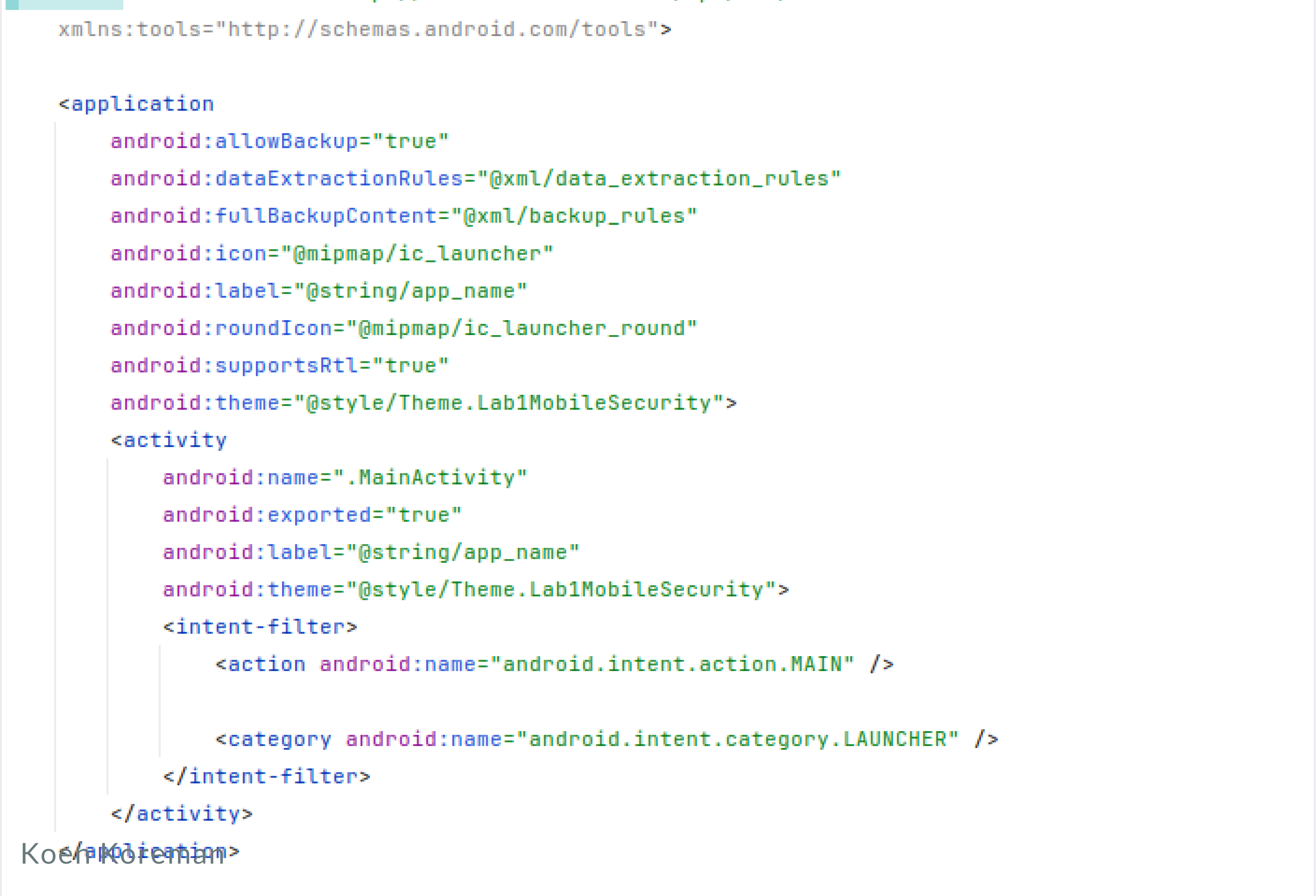
Android Package (APK) is the package file format used by the Android operating system for distribution and installation of mobile apps and middleware.

Android - APK files

An APK file is an archive that usually contains the following files and directories:

META-INF directory:

- MANIFEST.MF: the Manifest file
- The certificate of the application.
- CERT.SF: The list of resources and a SHA-1 digest of the corresponding lines in the MANIFEST.MF file
  lib: the directory containing the compiled code that is platform dependent



```
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Lab1MobileSecurity">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.Lab1MobileSecurity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

howest
hogeschool

Data storage

Data storage

Both Android and iOS provide secure credential storage for saved logins.

If an app doesn't use this API, it's likely the app saves credentials improperly.

For example: plain text in the file system.

Data storage: example requesting API

```
url = new URL("https://koenkoreman.be/mobilesecurity/index.php?user=KoenK");
HttpURLConnection urlConnection = null;
urlConnection = (HttpURLConnection) url.openConnection();
int responseCode = urlConnection.getResponseCode();

if (responseCode == 200) {

    InputStream inputStream = urlConnection.getInputStream();
    BufferedReader in = new BufferedReader(new InputStreamReader(inputStream));

    StringBuilder response = new StringBuilder();
    String currentLine;
    while ((currentLine = in.readLine()) != null) {
        response.append(currentLine);
    }
    in.close();
    userInfo = response.toString();
```

Example saving data plain text

```
try {
    FileOutputStream fileout=openFileOutput("data.json", MODE_PRIVATE);
    OutputStreamWriter outputWriter=new OutputStreamWriter(fileout);
    outputWriter.write("My data to save");
    outputWriter.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

Example saving data plain text

```
SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(COLUMN_data, data);
        values.put(COLUMN_moreData, moredata);
        db.insert(TABLE_USER, null, values);
        db.close();
```

Data storage

Android uses a file system that's similar to disk-based file systems on other platforms. The system provides several options for you to save your app data.

Data storage

App-specific storage: Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage. Use the directories within internal storage to save sensitive information that other apps shouldn't access.
Shared storage: Store files that your app intends to share with other apps, including media, documents, and other files.
Preferences: Store private, primitive data in key-value pairs.
Databases: Store structured data in a private database using the Room persistence library.

Data storage

All apps (root or not) have a default data directory, which is `/data/data/<package_name>`.

By default, the apps databases, settings, and all other data go here.

If an app expects huge amounts of data to be stored, or for other reasons wants to "be nice to internal storage", there's a corresponding directory on the SDCard: `Android/data/<package_name>`

Data storage

- databases/ ➡️ app's databases
- lib/ ➡️ libraries and helpers for the app
- files/ ➡️ other related files
- shared_prefs/ ➡️ preferences and settings
- cache/ ➡️ caches

Open databases

Android uses SQLite3 as database engine.
SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

To open a database use `sqlite3 database.db`
Inside the database `.tables` can be used to show all available tables or `.help` to get help information.

Logcat

Logcat is a command-line tool that dumps a log of system messages, including stack traces when the device throws an error and messages that you have written from your app with the Log class.

`adb logcat` gives all the log information back.

The `--help` option gives all information about this command.

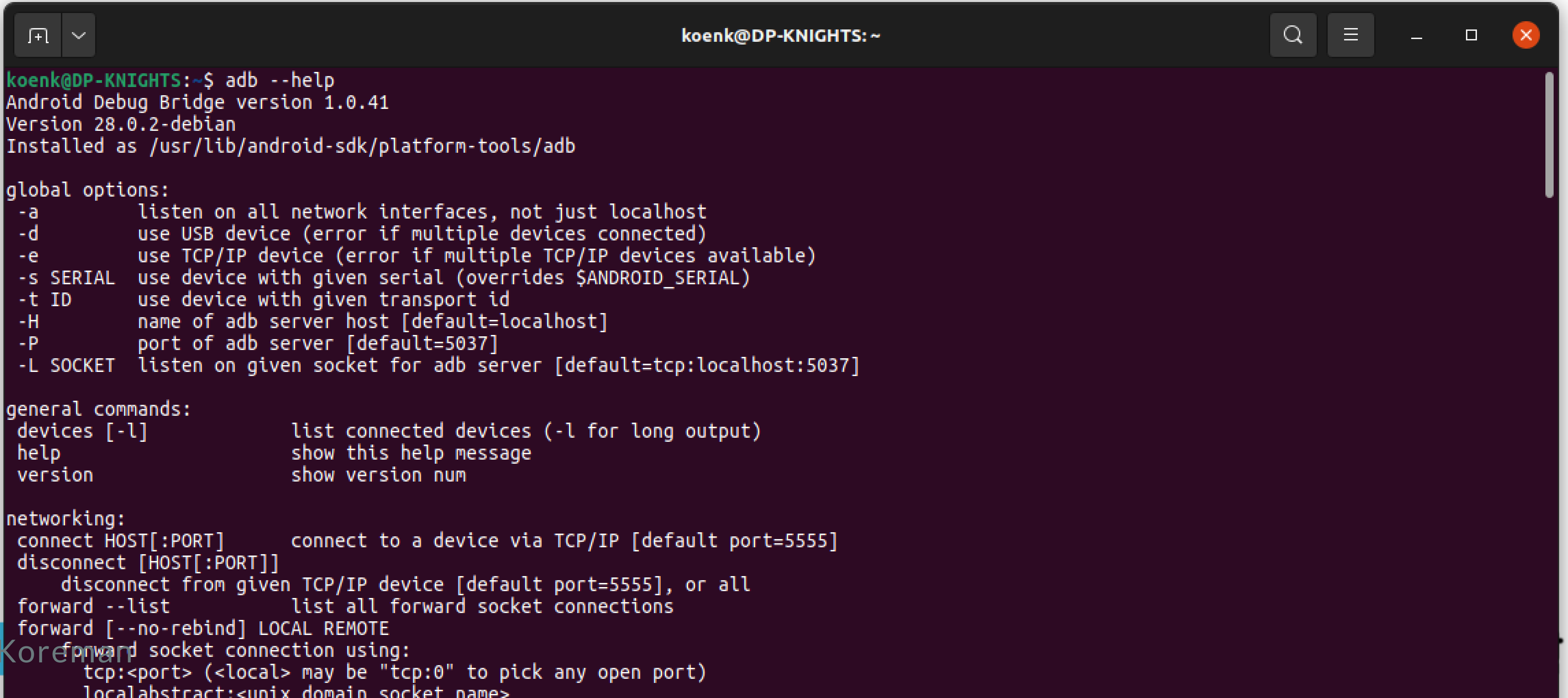
ADB

ADB

ADB (Android Debug Bridge), shipped with the Android SDK, bridges the gap between your local development environment and a connected Android device.

ADB allows testing apps on the emulator or a connected device via USB or WiFi.

ADB



```
koenk@DP-KNIGHTS:~$ adb --help
Android Debug Bridge version 1.0.41
Version 28.0.2-debian
Installed as /usr/lib/android-sdk/platform-tools/adb

global options:
 -a          listen on all network interfaces, not just localhost
 -d          use USB device (error if multiple devices connected)
 -e          use TCP/IP device (error if multiple TCP/IP devices available)
 -s SERIAL   use device with given serial (overrides $ANDROID_SERIAL)
 -t ID       use device with given transport id
 -H          name of adb server host [default=localhost]
 -P          port of adb server [default=5037]
 -L SOCKET   listen on given socket for adb server [default=tcp:localhost:5037]

general commands:
 devices [-l]             list connected devices (-l for long output)
 help                     show this help message
 version                  show version num

networking:
 connect HOST[:PORT]      connect to a device via TCP/IP [default port=5555]
 disconnect [HOST[:PORT]]
     disconnect from given TCP/IP device [default port=5555], or all
 forward --list           list all forward socket connections
 forward [--no-rebind] LOCAL REMOTE
     forward socket connection using:
       tcp:<port> (<local> may be "tcp:0" to pick any open port)
       localabstract:<unix domain socket name>
```

ADB

ADB provides many useful commands to run on the device.

The most powerful is to open a shell and enter the Android console on the device.

ADB

Android Debug Bridge (adb) is a versatile command-line tool that lets you communicate with a device.

It is a client-server program that includes three components:

1. A client, which sends commands. The client runs on your development machine. You can invoke a client from a command-line terminal by issuing an adb command.

ADB

2. A daemon (adbd), which runs commands on a device. The daemon runs as a background process on each device.
3. A server, which manages communication between the client and the daemon. The server runs as a background process on your development machine.

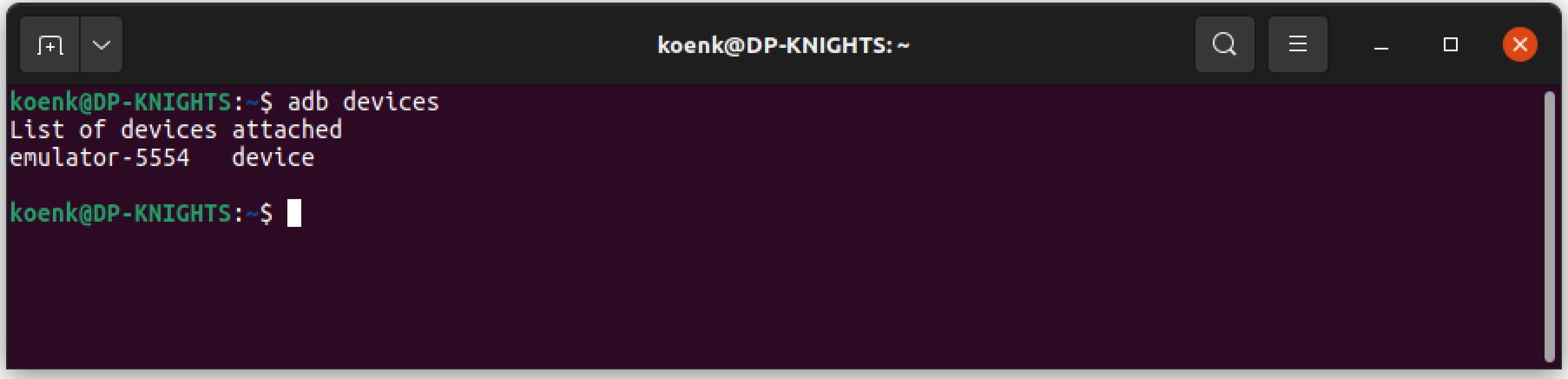
ADB - how ADB works

To use ADB with a device connected over USB, you must enable **USB debugging** in the device system settings, under **Developer options**.

Install the Android SDK that includes ADB in the Android SDK Platform-Tools package.

Connect your device with USB, verify that the device is connected by executing `adb devices` from the android_sdk/platform-tools/ directory. If connected, you'll see the device name listed as a "device."

ADB - how ADB works

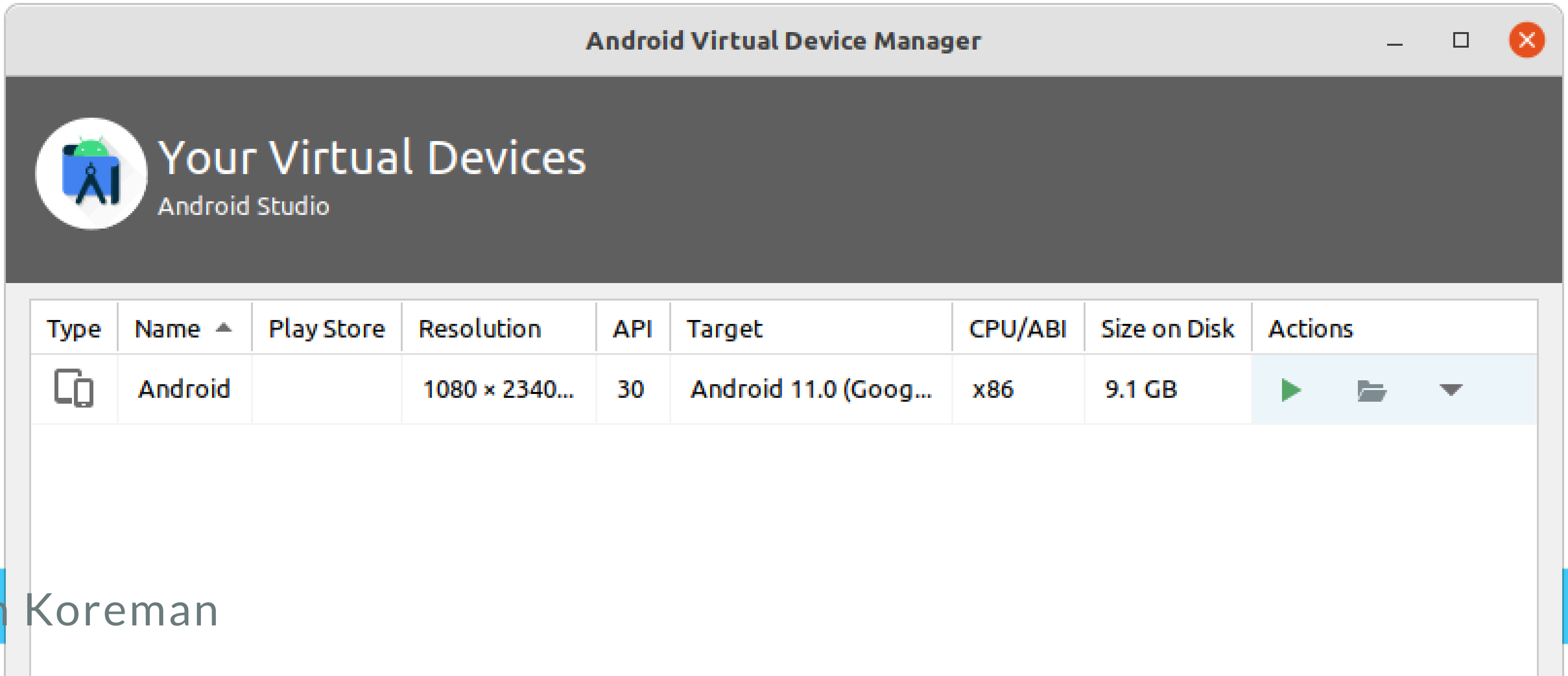


```
koenk@DP-KNIGHTS:~$ adb devices
List of devices attached
emulator-5554   device

koenk@DP-KNIGHTS:~$ 
```

ADB - how ADB works

Start the emulator in Android Studio (AVD manager)
Or plug in the actual device via USB. Make sure USB debugging is enabled via the Developer Tools.



Type	Name ▲	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Android		1080 × 2340...	30	Android 11.0 (Goog...	x86	9.1 GB	▶

ADB - commands

adb is installed in the folder:

```
C:\Users\%USERNAME%\AppData\Local\Android\Sdk\platform-tools
```

adb commands:

- adb devices
- adb connect [IP PORT]
- adb install
- adb forward

ADB - commands

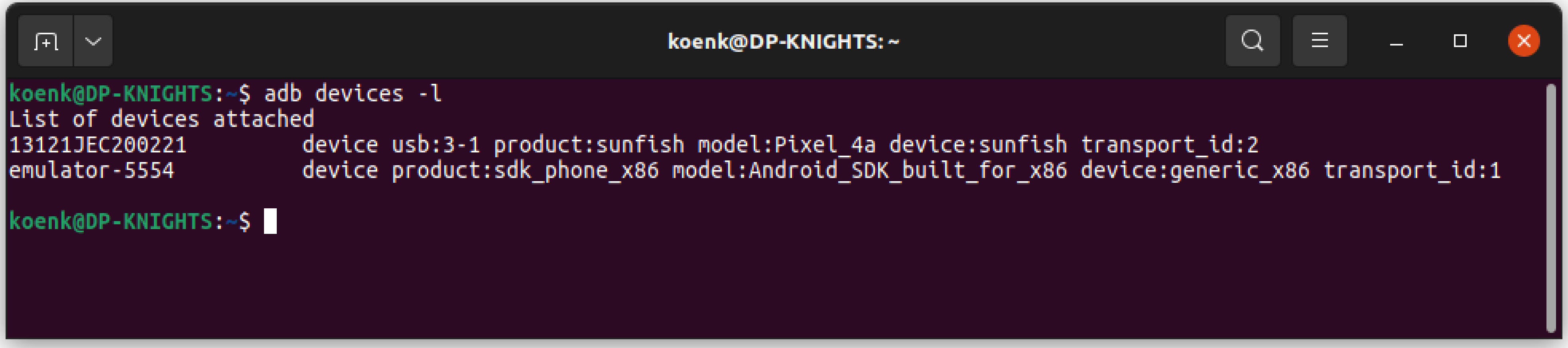
`adb devices`

Before issuing adb commands, it is helpful to know what device instances are connected to the adb server. You can generate a list of attached devices using the devices command.

`-l` gives detailed information about the device

- Serial number
- State (offline/device/no device)
- Description

ADB - commands



```
koenk@DP-KNIGHTS:~$ adb devices -l
List of devices attached
13121JEC200221         device usb:3-1 product:sunfish model:Pixel_4a device:sunfish transport_id:2
emulator-5554          device product:sdk_phone_x86 model:Android_SDK_built_for_x86 device:generic_x86 transport_id:1

koenk@DP-KNIGHTS:~$ 
```

ADB - commands

`install`

Install an APK to the connected device, sideload applications without using the Play Store

`forward`

Port forwarding on the device

ADB - commands

Call package manager (pm)
Within an adb shell, you can issue commands with the package manager (pm) tool to perform actions and queries on app packages installed on the device.

ADB - via WiFi

Android 11 and higher support deploying and debugging your app wirelessly from your workstation using Android Debug Bridge (adb).

The device needs to enable wireless debugging
Connection is via Pairing code

```
adb pair ip:port
```

ADB - via WiFi

Android 10 and lower support ADB via WiFi less then version 11 and higher.

```
adb tcpip 5555
adb connect ip
```

ADB

More information about ADB commands:

https://developer.android.com/studio/command-line/adb

howest
hogeschool

Demonstration

ADB connection and commands

Android bootloader

A bootloader is a vendor-proprietary image responsible for bringing up the kernel on a device. It guards the device state and is responsible for initializing the Trusted Execution Environment and binding its root of trust.

The bootloader enables recovery functionality of the device and verifies the integrity of the boot and recovery partitions before moving execution to the kernel and displays the warnings.

Android bootloader

```
FASTBOOT MODE
PRODUCT_NAME - bullhead
VARIANT - bullhead LGH791 16GB
 VERSION - rev_1.0
   DER VERSION - BHZ11f
     SION - M8994F-2.6.33.2.14
                  c1919b69
```

Android bootloader

Flashing images via the bootloader.

Download the official image, boot adb and flash the device with the downloaded image.

```
adb sideload ota_file.zip
```

Android bootloader

Unlock the bootloader to install custom ROMS of Android.

When the fastboot flashing unlock command is sent, the device should prompt users to warn them that they may encounter problems with unofficial images. After acknowledging, a factory data reset should be done to prevent unauthorized data access. The bootloader should reset the device even if it is not able to reformat it properly. Only after reset can the persistent flag be set so that the device can be reflashed.

Android custom rom

Custom Rom on Android – not official software for device

Unlock the bootloader and use fastboot to upload the image.

```
fastboot flash boot <recovery_filename>.img
```

Android custom rom

A custom ROM is an alternative Android version, based on the Android Open Source Project (AOSP) or an already existing ROM.

By installing a custom ROM, users can alter the source code of Android as desired.

More information: https://www.xda-developers.com/what-is-custom-rom-android/

Android fastboot

Fastboot is basically a diagnostic tool used to modify the Android file system from a computer when the smartphone is in bootloader mode. The commands are basic, and include, for example, to 'flash' (install) a boot image or a bootloader.

Root the device

Frameworks:

- Magisk Root
- Framaroot
- Towelroot
- CF-Auto-Root
- KingRoot

TWRP: TWRP is a recovery-level UI packed with powerful features that will make your rooted life easier.

Root the device

Rooting gets you to the bottom of the os. You get root privileges on the device.

This means:

- Access to the entire OS.
- Remove and replace the entire OS of the device.

It is just a standard Linux function that was removed.

Root the device

There are 2 ways to root the device:

- Systemless
- System root

Root the device

Systemless:

- Only the boot partition is modified.
- OS is >= Android marshmellow
- Cleaner
- Bypass Google SafetyNet (checks device tampering)

Root the device

System root:

- System partition is modified.
- Manufacturer can notice.
- Old way.

Root the device - cons

Unlocking the bootloader is OK!

Rooting & jailbreaking can void your warranty.
Please check in the device manual.

More vulnerable for malicious apps.

Could damage the phone.
Use tested methods or be aware of the consequences.

Root the emulator

There is a script that roots the emulator.

Prerequisites:

- Boot the AVD
- Clone rootAVD: `git clone https://gitlab.com/newbit/rootAVD`
- Working internet connection on AVD
- ABD is working

Root the emulator

Step 1: find the image file used for the AVD (ramdisk.img)

- `~/Android/Sdk/system-images/`
- `%localappdata%\Android\Sdk\system-images`

Execute the script with the correct ramdisk and open emulator.

```
/rootAVD.sh system-images/android-UpsideDownCakePrivacySandbox/google_apis_playstore/x86_64/ramdisk.img
```

Root the emulator

The script installs Magisk and patches the ramdisk with su.

After running the script the emulator is rooted, this can be tested via the app rootchecker.

Magisk

Magisk is a tool that can be used to gain systemless root access on your device, similar to SuperSU but it’s not limited to just that.

Magisk is a portal that enables all sorts of modifications on your Android phone. There are several Magisk Modules you can install on your phone for different purposes.

Magisk

There are modules for theming, ad blockers, enabling Camera2API, and a lot of other system-level modifications you can't do otherwise.

Magisk

- Magisk allows you to pass Google’s safety tests (Safety net).
- This open-source software allows you to add and modify files without any issues. If you know to code and want to make some changes to the default coding, you can do so.
- Magisk Mount feature will make changes to the core and partition level without any issues. You can divide your system, core files, and other media files and store them anywhere in the memory storage.
- Resetprop feature, in this you can make changes to your system prop files including read-only files. You can attempt changes in the build.

Magisk

Example of why to use Magisk:

- Use of a banking application on a rooted phone.
- 1controller module to support gamecontrollers (ps, xbox, ...)
- ARCore/Playground Patcher (enables ARCore on unsupported devices)

More information: https://github.com/topjohnwu/Magisk

Emulator

Using the emulator outside of Android Studio.
The emulator is an executable in the Android SDK. This can have several options while starting.

- -list-avds - list available AVDs
- -writable-system - make system & vendor image writable
- -cores - set number of CPU cores to emulator
- -help - prints the help content

Emulator

Example: start the emulator with this command

```
./emulator -avd NameAVD -writable-system -selinux permissive
```

This gives the emulator a state where the /system folder is writable and the security disabled.

howest
hogeschool

Lab time

Learn to work with the Emulator, ADB, rooting, …

See Lab1 on LEHO