

Sicherheit von Betriebssystemen – VO 07: Einführung in Android

Paul Kalauner

Research Group for Industrial Software (INSO)

<https://www.inso-world.com>



**FACHHOCHSCHULE
WIENER NEUSTADT**
University of Applied Sciences - Austria





Agenda

- Android Grundlagen
- Android Applikationen
- Threatmodell
- Sicherheitsmodell
- Demo: App Aufbau & Dekompilierung

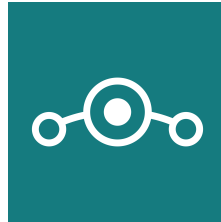
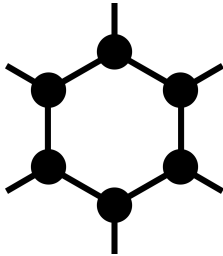
Android Grundlagen

Allgemein

- Basierend auf Linux
- Weitere Security-Maßnahmen durch SELinux
- Open Source: <https://source.android.com>
 - oftmals proprietäre Erweiterungen durch Smartphone(komponenten)-Hersteller

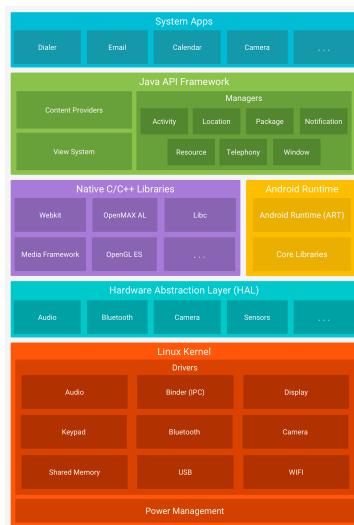
Android Distributionen

- CyanogenMod, das in LineageOS aufgegangen ist
- GrapheneOS
- Fire OS
- etc.



(Vergleiche <https://commons.wikimedia.org>)

Schematischer Aufbau von Android



(Vergleiche <https://developer.android.com/guide/platform>)

Entwicklung von Android und seinen Applikationen I

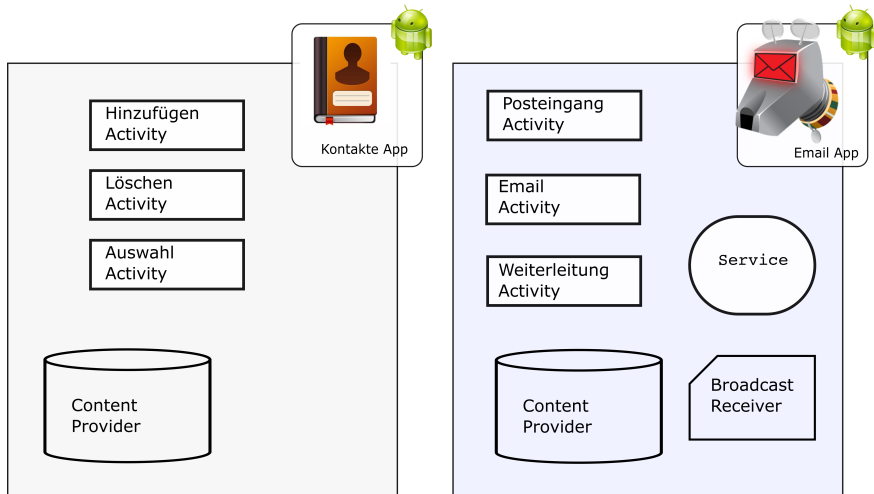
- Verwendung unterschiedlicher Sprachen zur Entwicklung von Android und Android Apps
- IDE: Android Studio - <https://developer.android.com/studio>
- Entwicklung des Android Systems zumeist in Systemprogrammiersprachen
 - C/C++
 - Rust - <https://play.rust-lang.org>
 - aber auch Java und Kotlin werden verwendet

Entwicklung von Android und seinen Applikationen II

- Entwicklung von nativen Apps erfolgte zuerst in Java und dann Kotlin - <https://play.kotlinlang.org>
 - mittlerweile existieren verschiedene Frameworks und Bibliotheken, welche die Entwicklung in anderen Programmiersprachen erlauben
 - ▶ React Native
 - ▶ Flutter
 - ▶ Xamarin/.NET MAUI
 - ▶ Apache Cordova
 - ▶ etc.
 - oftmals werden auch Shared Objects, die über das Java Native Interface (JNI) verfügbar sind, integriert

Android Applikationen

Komponenten Übersicht

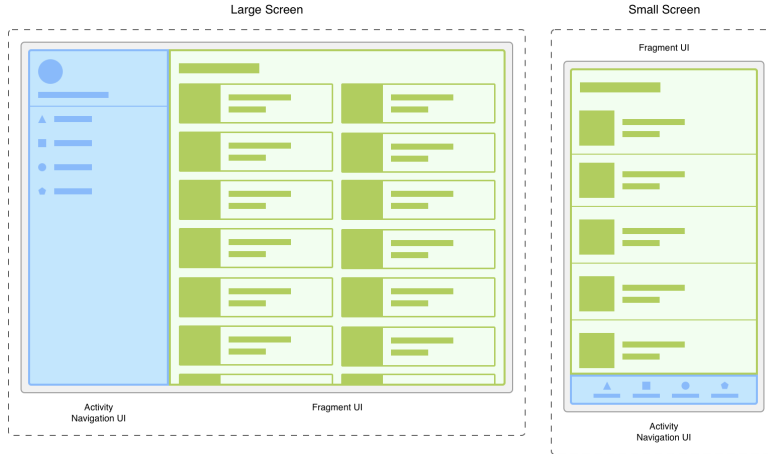


- App besteht aus Komponenten: **Activity**, **Service**, **BroadcastReceiver**, **ContentProvider**

Activity

- „Entry Point“ einer Android Applikation
- Beinhaltet Fragments und Layouts
- Muss im Android Manifest deklariert werden
- Heute: Meist einzige Activity, mehrere Fragments

Fragment



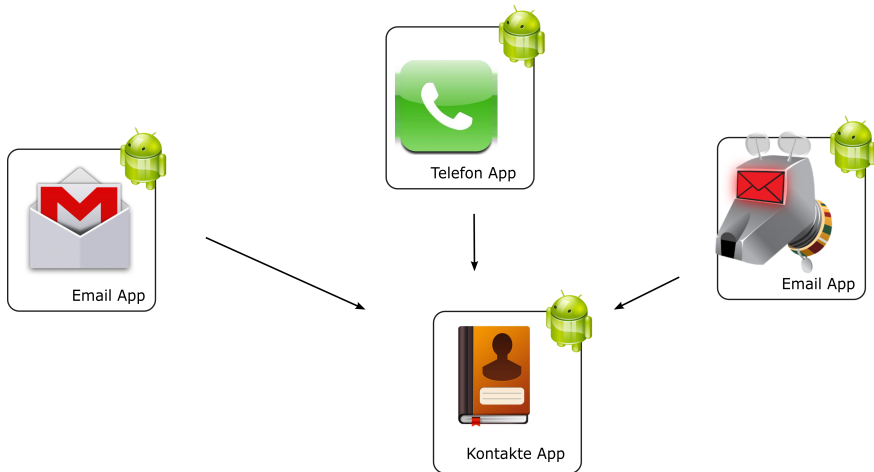
- (Wiederverwendbarer) Teil der UI
- Meist ein Fragment pro „Seite“

(Vergleiche <https://developer.android.com/guide/fragments>)

Service

- Für langlebige Hintergrund-Operationen
- Musik-Player, Downloads etc.
- Vordergrund-Services
 - Dauerhaft sichtbar für Nutzer (Benachrichtigung)
 - Ausführung auch wenn Nutzer andere App verwendet
- Hintergrund-Services
 - „Unsichtbar“ für Nutzer
 - Limitierte Funktionalität

Android-App Ökosystem



- Unterschiedliche Apps miteinander integriert: z.B. Email-Apps und Telefon-App verwenden Kontakte-App

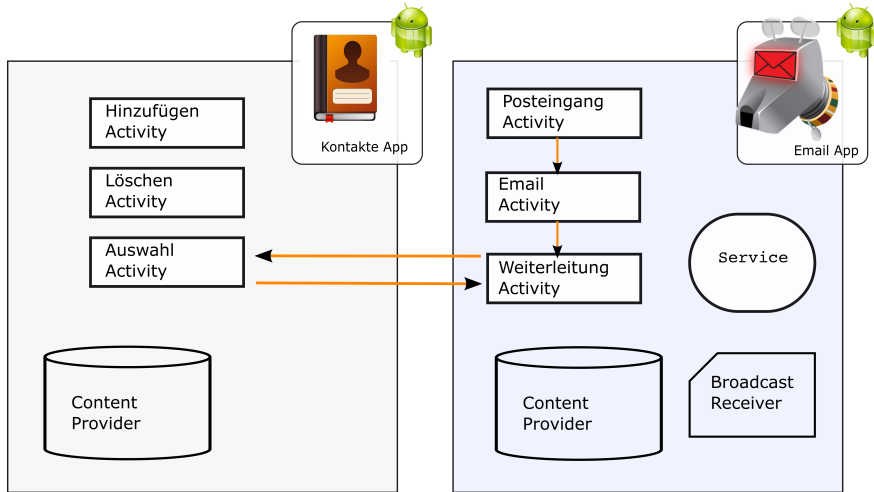
Content Provider

- Verwaltet Zugriff auf App-Daten
- Ermöglicht Zugriff auf Daten anderer Apps
- Ermöglicht es, Daten für andere Apps zugänglich zu machen

Broadcast Receiver

- Ermöglicht Reaktion auf globale Events (z.B. Erhalt einer SMS)
- Kann als weiterer „Entry Point“ einer App dienen
- z.B. Öffnen der App wenn Kamerataste gedrückt wurde

Inter Component Communication (ICC): Intents



- Asynchronous message passing System
- Intra- und Interapp Kommunikation

Datenspeicherung

- Shared Preferences (privat wenn `MODE_PRIVATE`, Key-Value)
- Internal Storage (privat wenn `MODE_PRIVATE`)
- Internal Database (privat, SQL)
- External Storage (public)

Hinweise zur Datenspeicherung

- `MODE_WORLD_READABLE` und `MODE_WORLD_WRITEABLE` vermeiden
- External Storage als nicht vertrauenswürdig einstufen
- So wenig wie möglich speichern
- **KeyStore** zum Speichern des Schlüssels verwenden
- Achtung bei Cloud Backups

Android Package Format (APK)

- ZIP-Datei
- Vollständige, installationsbereite Android Applikation
- Aus installierten Apps extrahierbar
- APKs können von unbekanntem Quellen installiert werden (Sideloadung)
 - USB
 - ▶ Android Debug Bridge
 - ▶ `https://developer.android.com/studio/command-line/adb`
 - ▶ `adb install <path_to_apk>`
 - Datei auf Smartphone (Download etc.)

Hauptinhalte einer APK

- **AndroidManifest.xml**: Deklariert App-Name, Komponenten etc.
- **lib/**: Libraries und kompilierter, plattformabhängiger Code
- **res/**: Ressourcen (z.B. Icons, Layouts etc.)
- **assets/**: Assets (z.B. Texturen)
- **classes.dex**: Im DEX-Format kompilierte Klassen

Dalvik EXecutable Format (DEX)

- Beinhaltet Dalvik Bytecode
- Konvertierbar in Smali (Human-Readable Representation)
 - smali/baksmali - <https://github.com/JesusFreke/smali>
 - Apktool - <https://ibotpeaches.github.io/Apktool>

Java-Code:

```
int x = 42
```

Dalvik Bytecode:

```
13 00 2A 00
```

Smali Repräsentation:

```
const/16 v0, 42
```

Android Runtime (ART)

- Führt DEX Bytecode aus
- Seit Android 5 (Ersetzt Dalvik VM)
- Ahead of Time (AOT) Kompilierung (bei Installation)
- Sandboxed (Eine ART Instanz pro App)
- Seit Android 7: „Hybrid“-Lösung AOT + JIT

Threatmodell

Mögliche Angriffe

- Forensic Attacks
- Network-based Attacks
- Code Execution Attacks
- Web-based Attacks
- Physical Proximity Attacks (USB, NFC)
- ...

Was eine App nicht ermöglichen sollte

- Störung anderer Apps
- Datendiebstahl, Spionage
- Unerwartete Kosten verursachen
- Denial of Service (z.B. Notruf)
- Auf alle OS-Funktionen (z.B. Mikrofon) zugreifen können
- ...

Dekompilierung

- Bytecode vergleichsweise einfach dekompilierbar
- Dalvik Bytecode (`classes.dex`) kann in Java Bytecode (JAR-Datei) konvertiert werden
⇒ dex2jar - <https://github.com/pxb1988/dex2jar>
- Resultierende JAR-Datei kann mit üblichen Java Decompilation Tools analysiert werden
⇒ jd-gui - <https://java-decompiler.github.io>
- Shortcut: DEX to Java Decompiler
⇒ jadx - <https://github.com/skylot/jadx>

Repackaging

- Bekannte App verwenden
- Disassemblieren und Payload einfügen
- Reassemblieren und in (in)offiziellm Store veröffentlichen
- die umverpackte App enthält oft Malware mit verschiedenen Funktionen
 - Diebstahl von Informationen
 - Diebstahl von Zugangsdaten
 - Premium-Rate Anrufe und SMS
 - Spam via SMS und Email
 - Erpressung des Opfers

Update-Angriff

- Ähnlich wie Repackaging
- Kein Payload sondern Update Routine
- Payload wird später nachgeladen
 - der Download ist versteckt
- umgeht so die Richtlinien und Checks von Google und Apple

Sicherheitsmodell

Sandboxing

- Apps laufen in eigener VM isoliert durch OS
- Jede App bekommt eigenen Linux User `a<number>_a<packagenumber>`
- Verzeichnis in `/data/user/<usernumber>/<packagename>`
- Durch Linux-Berechtigungen geschützt
- Mandatory Access Control (SELinux)

Android Berechtigungen

- Berechtigungen steuern Zugriff auf Ressourcen
 - Kamera and Mikrofon
 - Kontakte and Telefonie
 - Location Services
 - ...
- Explizites Einverständnis
- Einhaltung durch OS gegeben
- Feingranularer und widerrufbar ab Android 6

Speichern von Schlüsseln im KeyStore

- Vom OS zur Verfügung gestellt
- Unterstützte Cipher: `https://developer.android.com/training/articles/keystore.html#SupportedAlgorithms`
- `RSA/ECB/NoPadding`(sic!) und `RSA/ECB/PKCS1Padding` vor Android 6.0
 - Möglichst nicht verwenden
- `RSA/ECB/OAEP`Padding und `AES/GCM/NoPadding` nach Android 6.0

KeyStore Schutzmechanismen

- Keys non-exportable
 - Keys nie im Memory der App
 - Teilweise Support für Hardware-backed (`KeyInfo.isInsideSecureHardware()`)
- Schutz vor nicht authentifiziertem Zugriff konfigurierbar
 - Fixieren was mit dem Key gemacht werden kann (Signieren, Verifizieren, BlockMode, ...)
 - Nur eine Zeit lang gültig
 - Nur verwendbar wenn User kürzlich authentifiziert wurde (`KeyInfo.isUserAuthenticationRequired()`)

Signierung von Applikationen

- APKs sind mit selbstsignierten Zertifikaten signiert
- Apps im Play Store können nur mit selbem Zertifikat aktualisiert werden
- Allerdings: Code kann während Laufzeit nachgeladen werden
 - Via HTTP(S): Man-in-the-Middle möglich
 - Vom lokalen Dateisystem: Potentiell von anderen Apps veränderbar
 - Maliziöse App mit selbem Package-Namen könnte bereits installiert sein

⇒ Signierung liefert keine Information über ausgeführten Code

Android App Bundle (AAB)

- Verwendung ab August 2021 verpflichtend
- App Bundle wird mit Upload Key signiert und hochgeladen
- Google signiert AAB mit eigenem Key und generiert APK
 - APKs sind zugeschnitten auf verschiedene Geräte (Bildschirmauflösungen, Architekturen, ...)
 - Bei Verlust des Upload Keys kann neuer Key generiert werden

App Hardening

- Root/Jailbreak Detection
- Identifier Obfuscation (Proguard)
- String Obfuscation, Class Encryption, Junk Code Insertion, Call Graph Flattening
- Native Code vs. Byte Code Obfuscation

⇒ Wird in Einheit “Mobile Security – Vertiefung” behandelt

Demo:

App Aufbau & Dekompilierung

Zusammenfassung

- Mobile Applikationen sind omnipräsent
- Android basiert auf Linux
- Android OS stellt Sandboxing und Berechtigungs-Mechanismen zur Verfügung
- Komponenten einer Android App
 - Activity
 - Service
 - ...
- APKs enthalten vollständige Android Applikationen
- Dekompilierung liefert Einblick in die interne Funktionsweise von Android Apps

Literaturverzeichnis 1/2

- Android Runtime (ART) and Dalvik:
`https://source.android.com/devices/tech/dalvik`
- App Manifest Overview:
`https://developer.android.com/guide/topics/manifest/manifest-intro`
- Dalvik Executable format:
`https://source.android.com/devices/tech/dalvik/dex-format`
- Content provider basics: `https://developer.android.com/guide/topics/providers/content-provider-basics`

Literaturverzeichnis 2/2

- Broadcasts overview:
<https://developer.android.com/guide/components/broadcasts>
- Elenkov (2014): Android Security Internals
- Drake, et al. (2014): Android Hacker's Handbook
- Chell (2015): The Mobile Application Hacker's Handbook

Vielen Dank!

<https://establishing-security.at/>