esse

Mobile Security – VO 6: Environment Checks

Raphael Kiefmann, Paul Kalauner,
Vanessa Hohenegger, Rafael Vrecar,
Florian Fankhauser

Agenda

Einführung

Motivation

Root Detection

Hook Detection

Emulator & Custom Firmware Detection

Jailbreak Detection

Bootloader Unlock Detection

Remote Attestation

Einführung

- Besonders relevant in Apps mit medizinischem oder finanziellem Bezug
- „Environment Checks“ sind Tests, die zur Laufzeit sicherstellen, dass Gerät in „sicherem“ Zustand ist
  - Anwendungen erfordern teilweise, dass Gerät **nicht gerootet** ist, keine **Custom Firmware** einsetzt etc.
- Je nach Typ wird auf verschiedenste Eigenschaften getestet
  - Werte in Dateien
  - Vorhandensein verschiedenster Dateien & Ordner
  - etc.

Motivation

- Mechanismen, die bereits zur Compile-Zeit angewandt werden, können teils sehr einfach zur Laufzeit ausgehebelt/nachvollzogen werden
  - Beispiel Code Encryption
  - Die .dex-Datei liegt ursprünglich in verschlüsseltem Format vor
  - Spätestens vor Ausführung muss Datei entschlüsselt & in Speicher geladen werden
  - Sobald die .dex-Datei im Speicher landet, kann sie leicht aus Speicher extrahiert werden
- Environment Checks sorgen beispielsweise dafür, dass gerootetes Gerät oder Versuch, sich mit frida einzuklinken, erfolgreich erkannt wird

Detection Mechanismen

- Root Detection
- Hook Detection
- Emulator & Custom Firmware Detection
- Jailbreak Detection
- Bootloader Unlock Detection
- Remote Attestation

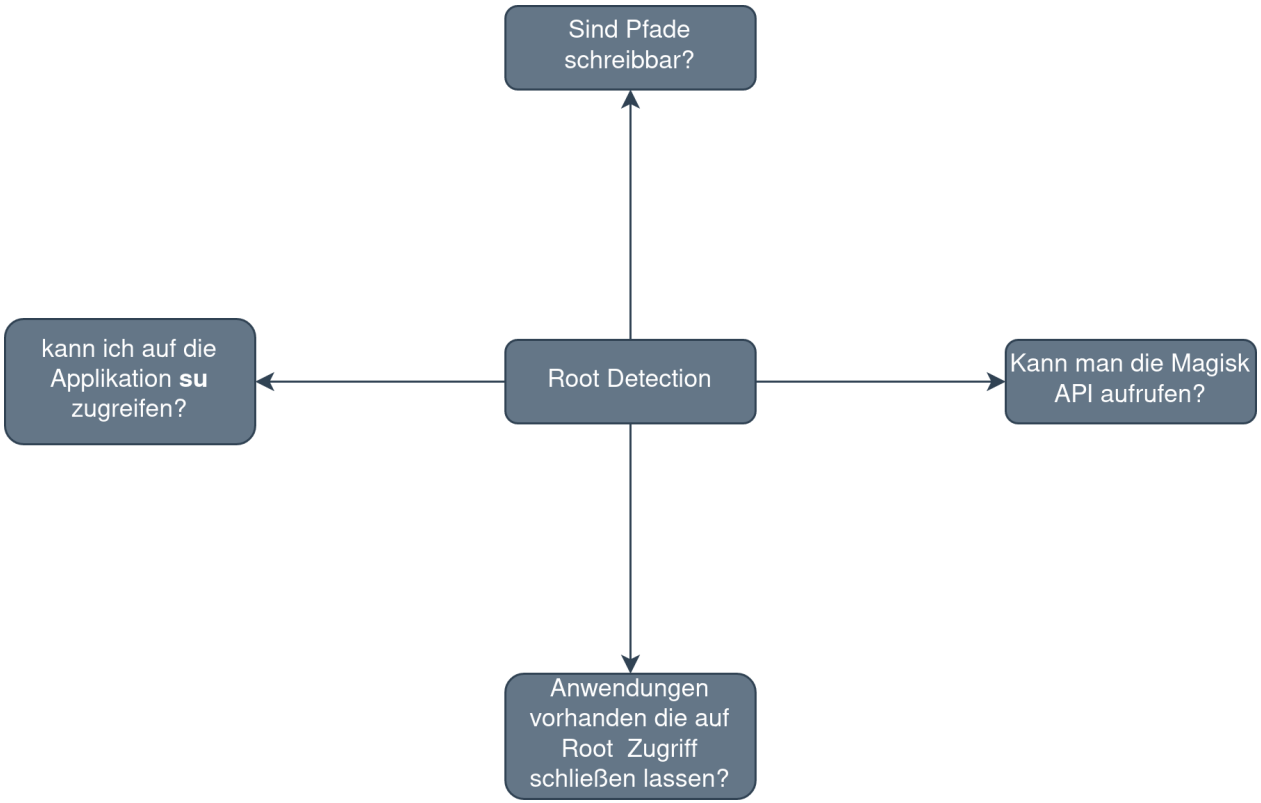
Root Detection

- Dient dazu, dass gerootetes Gerät erkannt wird
- Zugriff auf **root** Benutzer:in ist Indiz für etwaige weitere Modifikationen

Brainstorming zur Root Detection

Worauf würden Sie achten, wenn Sie eine Root Detection implementieren müssten?

Brainstorming zur Root Detection - Mindmap



Tests der Root Detection

- Vorhandensein von verschiedenen Dateien
  - `/usr/bin/su`
  - `/sbin/su`
  - etc.
- Vorhandensein von verschiedenen Applikationen
  - Magisk (`com.topjohnwu.magisk`)
  - SuperSU (`eu.chainfire.supersu`)
  - etc.
- Möglichkeit, Befehle als `root` Benutzer:in auszuführen
  - Magisk bietet eine eigene Bibliothek an

Detection Mechanismen

- Root Detection
- Hook Detection
- Emulator & Custom Firmware Detection
- Jailbreak Detection
- Bootloader Unlock Detection
- Remote Attestation

Hook Detection

- Dient dazu, dass erkannt wird, ob Applikation gehookt wird
- Durch Hooken können andere Mechanismen & Hooking selbst verschleiert werden

Brainstorming zur Hook Detection

Worauf würden Sie achten, wenn Sie eine Hook Detection implementieren müssten?

Brainstorming zur Hook Detection - Mindmap

Tests der Hook Detection 1/2

- Vorhandensein von Dateien in `/data/local/tmp`
  - `frida`
  - `frida-server`
  - etc.
- Verbindung zu Port 27042, welcher standardmäßig von **frida** aufgemacht wird
- **frida** verwendet Named Pipes zur Kommunikation
  - Auf Android (immer) unter dem Pfad `/data/local/tmp/pipe-[a-f0-9]{32}` zu finden

Tests der Hook Detection 2/2

- Überprüfung von Werten im virtuellen Verzeichnis `/proc`
  - `/proc/self/maps` beinhaltet die verwendeten geteilten Bibliotheken $\Rightarrow$ **frida** taucht darin auf
  - `/proc/self/maps` kann benutzt werden, um Speicherregionen zu öffnen & z. B. nach Wort „frida“ zu durchsuchen
  - Wird `/proc/self/fd` mit `readlink` aufgelöst, beinhaltet bei Verwendung von **frida** retournierten Pfad **linjector**
  - `/proc/self/tasks` beinhaltet bei Verwendung von **frida** Werte **pool-frida** & **gmain**

Detection Mechanismen

- Root Detection
- Hook Detection
- Emulator & Custom Firmware Detection
- Jailbreak Detection
- Bootloader Unlock Detection
- Remote Attestation

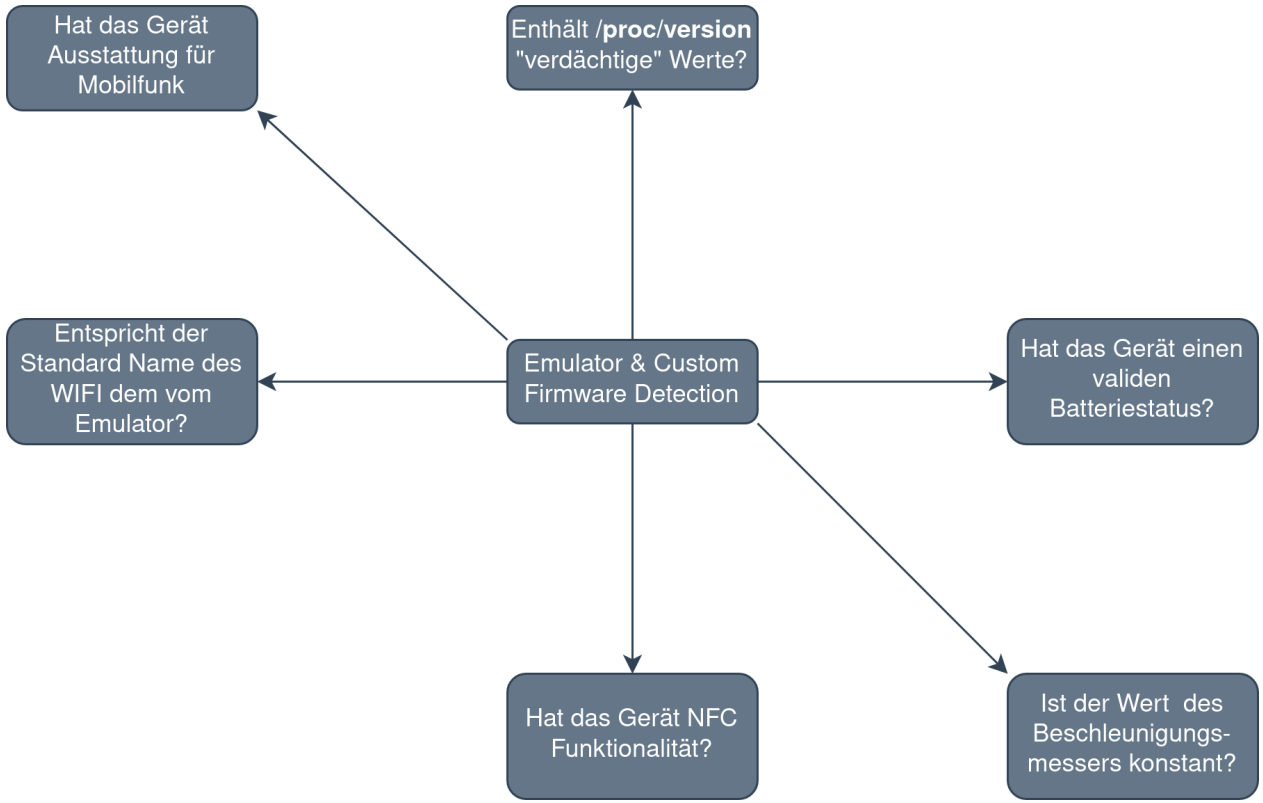
Emulator & Custom Firmware Detection

- Dient dazu, dass erkannt wird, ob inoffizielles Betriebssystem auf Gerät läuft oder Gerät emuliert wird
- Custom Firmware ist oftmals Indiz für weitere Modifikationen

Brainstorming zur Emulator & Custom Firmware Detection

Worauf würden Sie achten, wenn Sie eine Custom Firmware Detection implementieren müssten?

Brainstorming zur Emulator & Custom Firmware Detection - Mindmap



Tests der Emulator & Custom Firmware Detection 1/3

- Vorhandensein von gewissen Werten
  - In virtueller Datei (vom Kernel generiert) `/proc/version` sind oft Informationen zu Entwickler:innen des Kernels/der Custom ROM enthalten
- Auslesen der **System Properties**
  - Werden oftmals beim Bauen integriert & sind daher nicht einfach änderbar
  - Enthalten je nach ROM zusätzlich Einträge bzw. Werte, welche Rückschlüsse auf Custom Firmware zulassen

Tests der Emulator & Custom Firmware Detection 2/3

```
[ro.build.display.id]: [lineage_crosshatch-userdebug 11 RQ3A.211001.001 a384684437]
[ro.build.flavor]: [lineage_crosshatch-userdebug]
[ro.lineage.build.version]: [18.1]
[ro.lineage.build.version.plat.rev]: [0]
[ro.lineage.build.version.plat.sdk]: [9]
[ro.lineage.device]: [crosshatch]
[ro.lineage.display.version]: [18.1-20220421-NIGHTLY-crosshatch]
[ro.lineage.releasetype]: [NIGHTLY]
[ro.lineage.version]: [18.1-20220421-NIGHTLY-crosshatch]
[ro.lineagelegal.url]: [https://lineageos.org/legal]
```

Tests der Emulator & Custom Firmware Detection 3/3

- Vorhandensein von gewissen Pfaden
  - `/vendor/bin/qemu-props`
  - `/dev/qemu_pipe`
  - `/sys/qemu_trace`
  - `/vendor/bin/qemu-props`

Detection Mechanismen

- Root Detection
- Hook Detection
- Emulator & Custom Firmware Detection
- Jailbreak Detection
- Bootloader Unlock Detection
- Remote Attestation

Jailbreak

- Ziel: iOS-Sicherheitsmechanismen durchbrechen (z. B. Sandbox)
- Nutzung einer Reihe von Schwachstellen im System
- Deaktivieren der „Chain of Trust“
- Kernel-Modifikationen: Installation von unsignierten/benutzer:innendefinierten Anwendungen
- **Jailbreak Detection**:
  - Exklusiv auf iOS Geräten zu finden
  - Erkennung von Jailbreak ist meistens Indiz dafür, dass noch mehr Modifikationen durchgeführt wurden

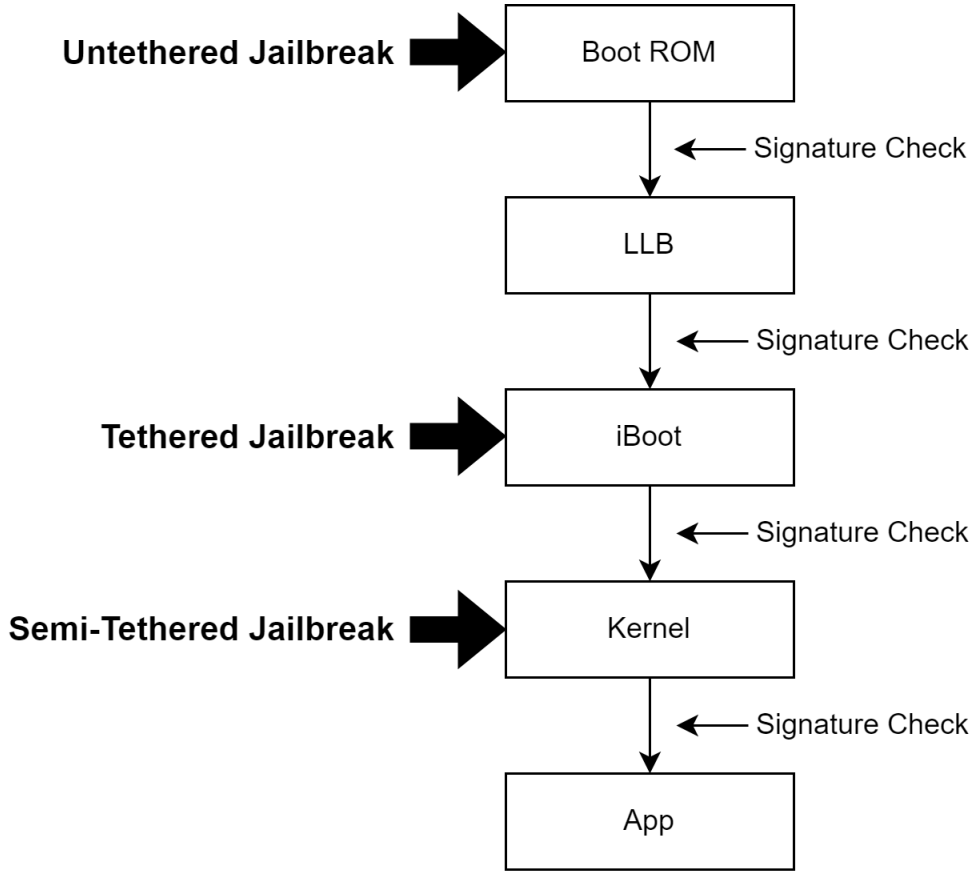
(Semi-)Tethered Jailbreaks

- **Tethered Jailbreak**
  - Computer erforderlich
  - Verändert Bootvorgang (iBoot-Stufe)
  - Kann nur in einem einzigen Bootvorgang verwendet werden ⇒ temporär
  - Nach Aus- & Einschalten ⇒ Recovery Mode
- **Semi-Tethered Jailbreak**
  - Wie Tethered, jedoch landet man nicht im Recovery Mode

(Semi-)Untethered Jailbreaks

- **Untethered Jailbreak**
  - Kein Computer erforderlich
  - Während des Bootvorgangs wird Kernel automatisch exploited $\Rightarrow$ permanent
  - Schwierig zu implementieren
- **Semi-Untethered Jailbreak**
  - Kein Computer erforderlich
  - Verwendet eine App, Website oder andere Software

Jailbreak Arten in Chain of Trust



1

(Vergleiche Liu et al. (2016), Kellner et al. (2019), Ali et al. (2019))

[1] LLB = Low Level Bootloader

Cydia

- Paketmanager für alternative Apps & Erweiterungen, welche nicht im offiziellen App Store erhältlich
- Jailbreak-Tools installieren Cydia oft zusätzlich
- Advanced Package Tool (APT) für iOS-Geräte
  - Grafische Benutzer:innenoberfläche
  - Unterstützt Installation von Apps & einigen Tools zur Manipulation von Apps
- Registriert ein URL Scheme (`cydia://`)

(Vergleiche `https://cydia.saurik.com`)

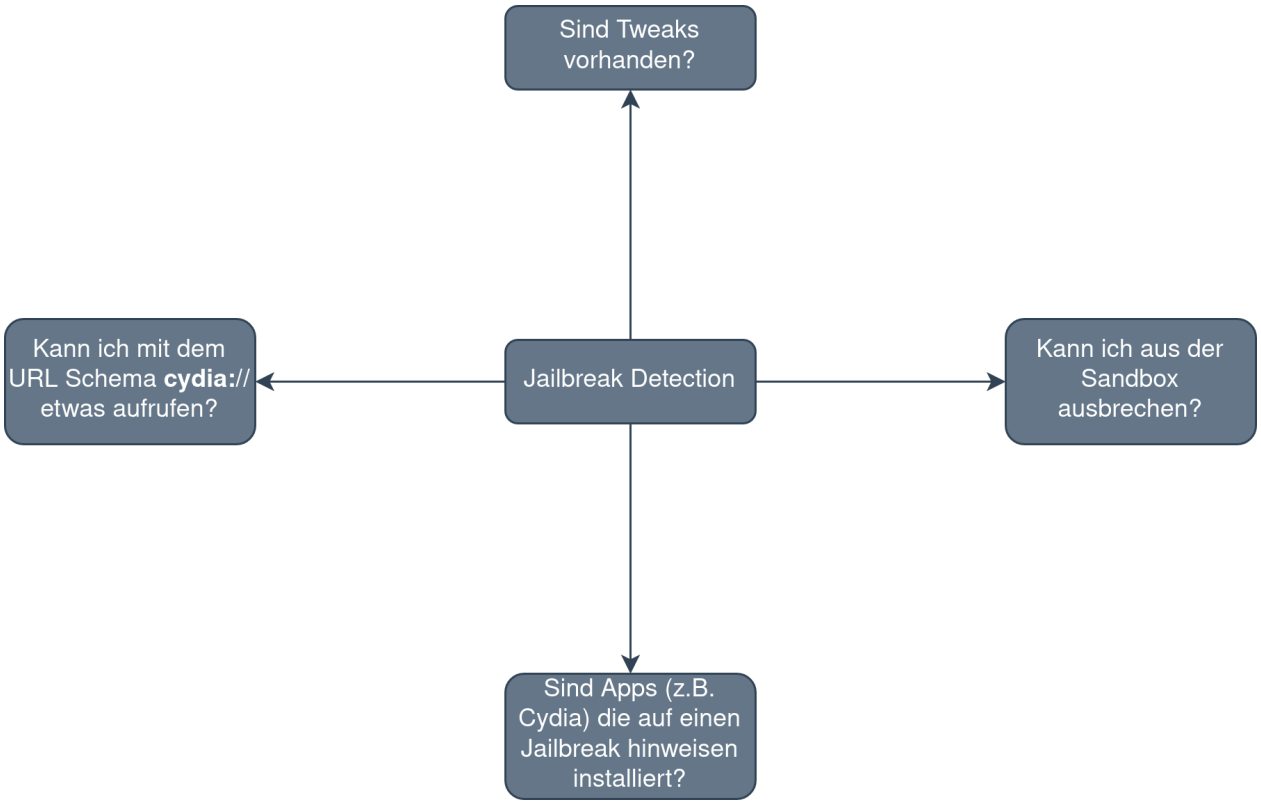
Beispiele für Jailbreak Tools & Exploits

- Tools
  - Checkra1n (14.0 - 14.8.1): Semi-Tethered
  - Taurine (14.0 - 14.3): Semi-Untethered
  - Unc0ver (14.0 - 14.8): Semi-Untethered
- Exploits
  - Checkm8 (CVE-2019-8900)
  - Ivac entry use-after-free (CVE-2021-1782)
  - Pattern-f's closed source exploit (CVE-2021-30883)
  - Cicuta_virosa (CVE-2021-1782)

Brainstorming zur Jailbreak Detection

Worauf würden Sie achten, wenn Sie eine Jailbreak Detection implementieren müssten?

Brainstorming zur Jailbreak Detection - Mindmap



Beispiele für Jailbreak Detection in Apps

- Dateien/Apps prüfen (z. B. Cydia in `/Applications`, Paths mit `isReadableFile` testen)
- Dateisystem-Prüfungen (z. B. versuchen, in `/privat`-Verzeichnis zu schreiben)
- Überprüfung der Plattformfunktionalität (z. B. URL schemes - `canOpenURL`, `bin/sh` Check mit `system`-Funktion)

Detection Mechanismen

- Root Detection
- Hook Detection
- Emulator & Custom Firmware Detection
- Jailbreak Detection
- Bootloader Unlock Detection
- Remote Attestation

Bootloader Unlock Detection

- Standard Android-Geräte haben bei Auslieferung (immer) geschlossenen Bootloader
- In letzten Jahren ist es notwendig gewesen, dass Bootloader aufgemacht werden muss, um Gerät zu rooten
  - Davor haben oftmals Exploits ermöglicht, Gerät ohne offenen Bootloader zu rooten

Brainstorming zur Bootloader Unlock Detection

Worauf würden Sie achten, wenn Sie eine Bootloader Unlock Detection implementieren müssten?

Tests der Bootloader Unlock Detection

- Nicht direkt möglich, Status zu überprüfen
- Indirekt kann darauf, z. B., via Custom Firmware Detection geschlossen werden
- Googles SafetyNet API ermöglicht über „Hardware Attestation“, Rückschlüsse auf Status des Geräts zu ziehen

Detection Mechanismen

- Root Detection
- Hook Detection
- Emulator & Custom Firmware Detection
- Jailbreak Detection
- Bootloader Unlock Detection
- Remote Attestation

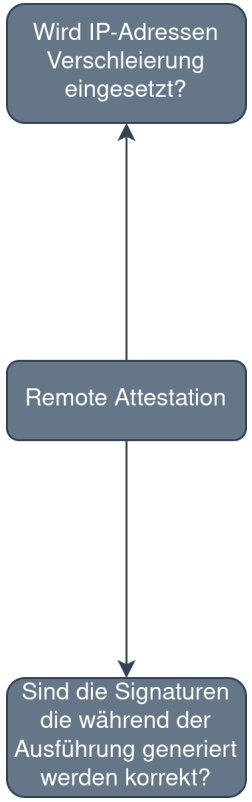
Remote Attestation

- Bei Remote Attestation wird extern überprüft, ob Applikation/Gerät in sicherem Zustand
- Erfordert Internet-Anbindung
- Sehr kritisch in anbetracht von Privacy
  - Daten werden immer zu einem externen Server geschickt

Brainstorming zur Remote Attestation

Worauf würden Sie achten, wenn Sie eine Remote Attestation implementieren müssten?

Brainstorming zur Remote Attestation - Mindmap



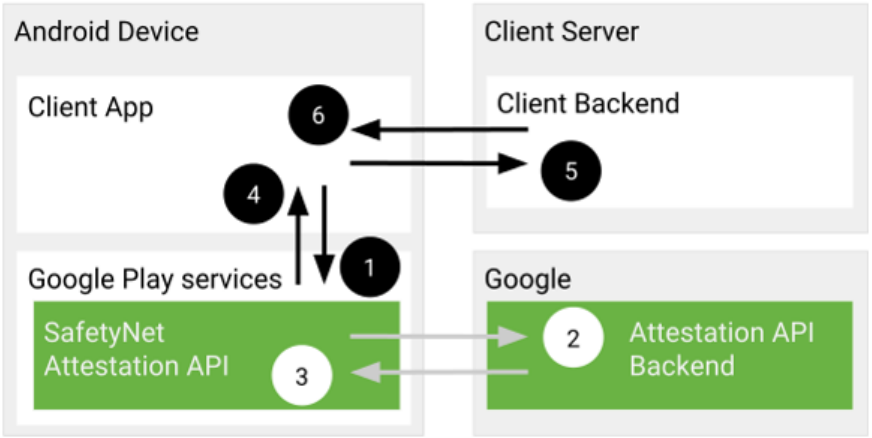
Aufbau der Remote Attestation 1/3

- Eine der kompliziertesten Techniken, die man umsetzen kann
- Je nach Hersteller wird unterschiedlich attestiert, ob Gerät in Ordnung
  - Sind gewisse Abläufe durchgeführt worden
  - Sind signierte Daten valide (SafetyNet API)
  - etc.

Aufbau der Remote Attestation 2/3

- Googles SafetyNet API ermöglicht über „Hardware Attestation“, Rückschlüsse auf Status des Geräts
  - In TEE wird ein „Attestation Signing Key“ bei der Produktion hinterlegt, der beim Öffnen des Bootloaders unzugänglich gemacht wird (Prünster et al. (2019))
  - Ohne „Attestation Signing Key“ schlägt Attestierung fehl

Aufbau der Remote Attestation 3/3



(Vergleiche `https://developer.android.com/images/training/safetynet/attestation-protocol.png`)

Zusammenfassung

- Environment Checks um sicherzustellen, dass sich Gerät in "sicherem" Zustand befindet
- Beispiele für Environment Checks
  - Root Detection
  - Hook Detection
  - Jailbreak Detection
- Schutz von persönlichen Daten oder vor Angriffen auf Compile-Zeit Schutzmechanismen
- Umsetzung durch Prüfen verschiedenster Geräteeigenschaften

Literaturverzeichnis 1/2

- Feng Liu, Ke-Sheng Liu, Chao Chang, und Yan Wang. Research on the Technology of iOS Jailbreak. In *2016 Sixth International Conference on Instrumentation Measurement, Computer, Communication and Control (IMCCC)*, Seiten 644–647, 2016. doi: 10.1109/IMCCC.2016.178
- Ansgar Kellner, Micha Horlboge, Konrad Rieck, und Christian Wressnegger. False Sense of Security: A Study on the Effectivity of Jailbreak Detection in Banking Apps. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, Seiten 1–14, 2019. doi: 10.1109/EuroSP.2019.00011

Literaturverzeichnis 2/2

- Amin Aenurahman Ali, Niken Cahyani, und Erwin Jadied. Digital Forensic Analysis on iDevice: Jailbreak iOS 12.1.1 as a Case Study. *Indonesia Journal on Computing (Indo-JC)*, 4(2):205–218, Sep. 2019. doi: 10.34818/INDOJC.2019.4.2.349
- Bernd Prünster, Gerald Palfinger, und Christian Kollmann. Fides: Unleashing the Full Potential of Remote Attestation. In *ICETE (2)*, Seiten 314–321, 2019

Vielen Dank!

https://establishing-security.at/