Android Security Workshop

Eduardo Novella
(NowSecure)

Connecting next generation talent with the heavy duty industry to keep vehicles secure

June 20-24, 2022 | Michigan (USA)

$ whereis

Material

```
$ git clone https://github.com/nowsecure/cybertruckchallenge22.git
```

$ whoami

"I stay with problems longer"

- Mobile Security Research Engineer @ **NowSecure**
  - Focused on **Android** Reverse Engineering

- **Previously (**Reverse Engineering**)**
  - Android **mobile** security: cloud-based payments (HCE wallets), DRM and TEE solutions
  - **Embedded** security : smartcards, smart meters, Pay TV, HCE, routers, any hardened IoT dev
  - Crypto: side-channel & fault injection attacks (hw). Whitebox cryptography (sw)

- **Background**
  - **IT** : sw- and hw- security, crypto, embedded, networks
  - **CTF** player occasionally

- Personal @ enovella.github.io
  - Based in Europe (**ES**, UK, NL)
  - Chess player, swimmer and nature lover (soon to be father)

Outline

Main ideas

- ## Android Introduction

  Android Security Internals

  Automotive Android OS
  Threat Modeling & Bug Hunting

- ## Android Reverse Engineering

  Open-Source Mobile RE Tools

  Static Analysis
  Dynamic Analysis (Frida)
  Network Analysis

- ## Hands-On: Android Challenge

  Keyless Android app to wirelessly unlock vehicles with your mobile

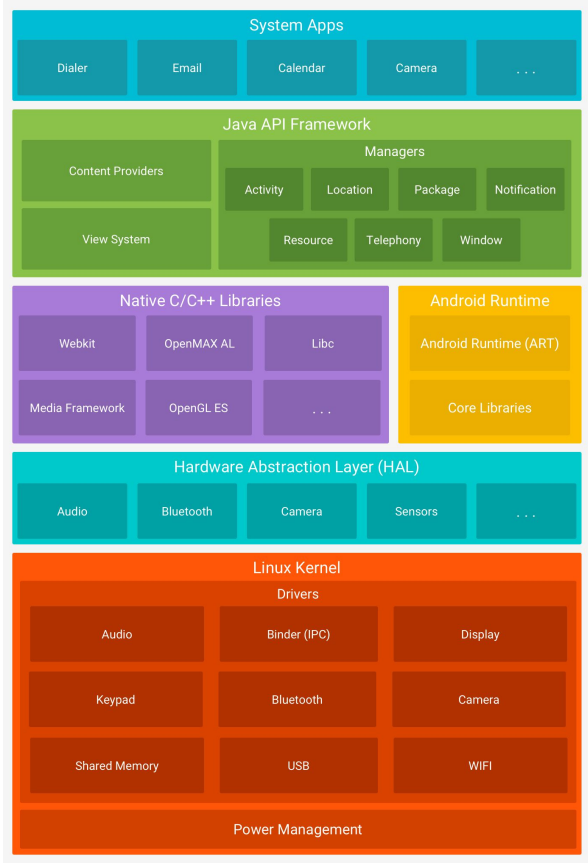
  “Mobile Keyless Remote System”

- ## Takeaways - QA

Android OS

Architecture

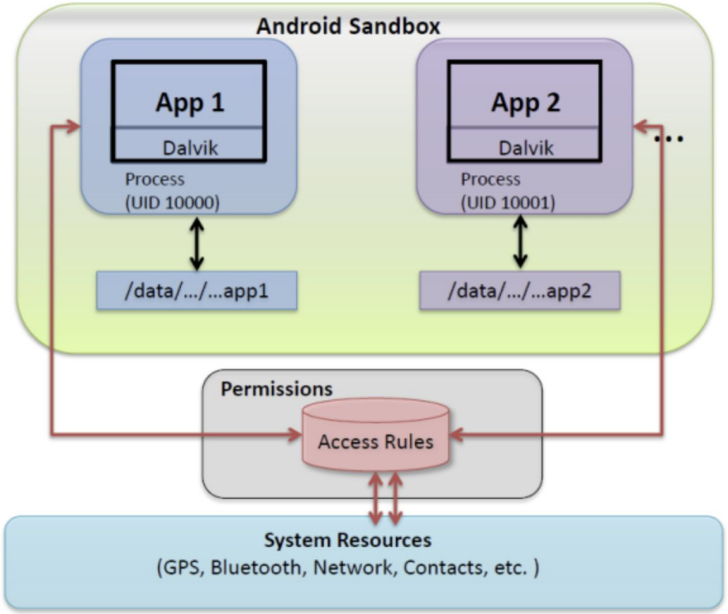
- Android OS developed by Google
  - Based on Linux (Open Source) with “Androidisms”
  - Components:
    - Linux Kernel
      - Binder - driver used for IPC
      - Native Userspace - init process - Zygote
    - Hardware Abstraction Layer (HAL)
    - Native core libraries (C/C++/Rust)
    - Android Runtime - Dalvik VM (jit) vs ART (aot)
    - Java API Framework
    - Applications
      - System Apps (RO partition mounted as /system)
      - User-installed Apps (RW partition mounted as /data)



Android Security Model

App Security

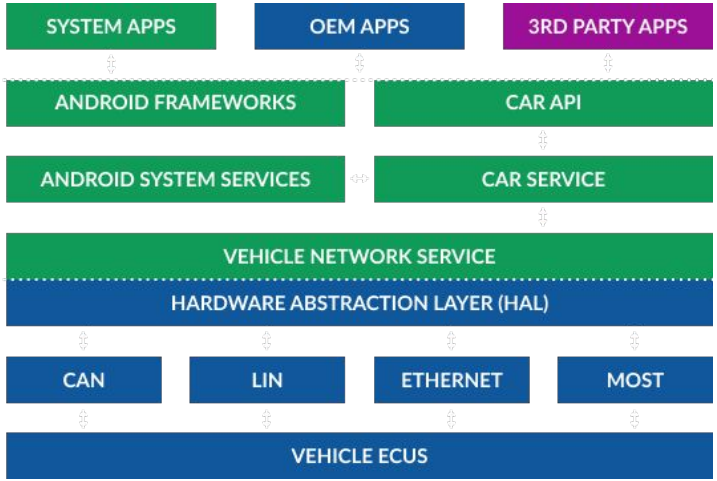
- Application Sandboxing
  - Each app operates in its own isolated environment
  - Unix-style permission model
  - Data directory */data/data/package-name-app/*
  - App data sharing via IPC (content providers)
  - UID (User Identity). Greater than 10000 for normal apps
  - Code signing inherited from Java JAR “same origin policy”
    - Each application signed with self-signed dev-certs
- Permissions
  - Defined `AndroidManifest.xml` inside APK
  - Run- and installation-time approval
  - Allow sms, microphone, network, gps, nfc,
- Components
  - Activity - UI screen
  - Broadcast receivers - snd/rcv data from/to apps
  - Content providers - enable sharing data between apps
  - Services - run in background



Automotive Android OS (AAOS)

Architecture

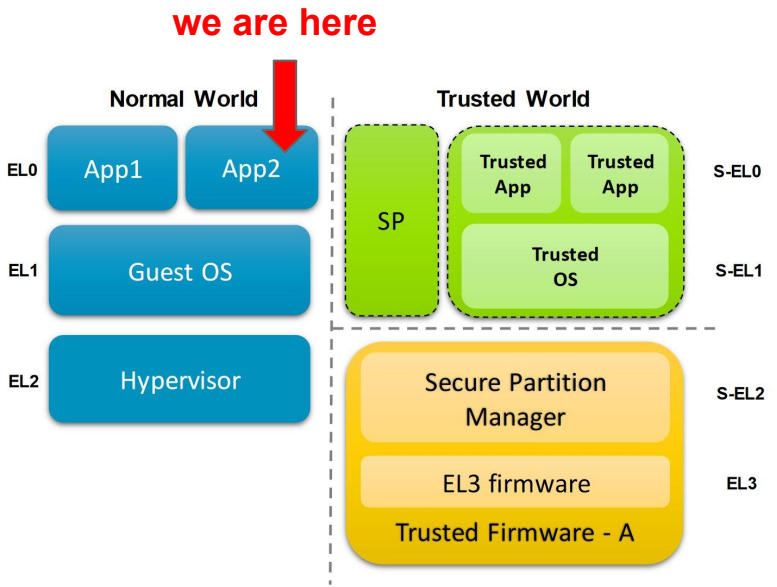
- Android Auto - App
  - A framework to connect your Android phone to cars
- Android Automotive OS (AAOS) - *"Android for Cars"*
  - Infotainment system built into cars by carmakers
  - Interface designed for car screens
  - Components
    - In-vehicle Infotainment (IVI) system
    - Google Automotive Services (GAS)
    - Vehicle Map Service (VMS)
    - Exterior View System (EVS)
    - Heating, ventilation & AC (HVAC)
  - OEM receives access to GAS via a partnership with Google



Android Security Model

Hardware Security

- ARM TrustZone - Trusted Execution Environment (TEE)
  - Hardware-enforced isolation built in SoC
  - Secure area of main processor
  - Isolate Normal- (NWd) and Secure- world (SWd)
  - Non-Secure and Secure state kept in HW reg
  - NWd ← Secure Monitor Call (SMC) → SWd
  - TEE OS executed right after BootROM
  - Hardware-backed KeyStore
  - Protect critical assets:
    - Crypto, TRNG, Biometrics, Payment, DRM, Boot Integrity
- Google Titan M Chip (Secure Element)
  - Separate secure chipset manufactured for *Pixel* devices
  - Tamper-resistant hardware against side channel attacks
  - Enforces Android Verified Boot (AVB)
  - Stronger KeyStore: Android "*StrongBox*" Keymaster
  - Side channels attacks - BH 2021



Android Security Model

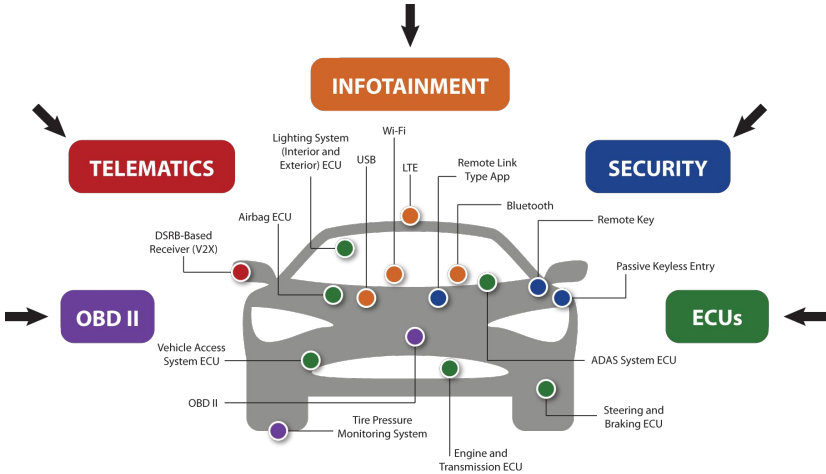
Device Security

- Bootloader
  - Unlocked
    - SuperSU - Magisk
  - Locked
    - Privilege escalation
      - Symlink/logic bugs
      - OEM Framework bugs
      - Kernel bugs
- Exploits
  - StageFright - Android 2.2 - 5.1.1
  - TowelRoot - Futex bug - Android devices w/ kernels 3.15.x
  - Pingpong - UAF in linux socket
  - Dirty Cow - Kernel race condition on Copy-on-Write (Cow)
  - Bluefrag - Bluetooth zeroclick RCE on Android 8/9
  - OEM backdoors - OnePlus “Angela”
  - Dirty Pipe - Android 12 kernel >= 5.10 (Pixel 6 - Samsung S22)

Threat Modeling

Attack Surface

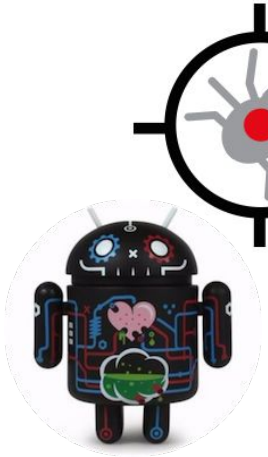
- Physical access
  - USB port (ADB). Developer Options enabled
  - Hardware ports for debugging purposes
  - Vendor proprietary apps
  - Kiosk escape
- Vendor's Applications
  - Identify critical assets within the app
    - IP, crypto, databases, shared pref
    - Proprietary protocols and crypto
  - Network protocols (MITM), tracking, GPS spoofing
  - Firmware updates
- Non-physical access
  - Wireless (WiFi, Bluetooth, NFC, LTE, Baseband)
  - Vulnerabilities on old Android OS
  - Web server accessible via browser



Android App Bug Hunting

Vulnerabilities

- Insecure connections (auth over network)
- Cryptography and Authentication
  - Hardcoded secrets, Oauth tokens
  - Plaintext databases
- Unprotected App Components (activity, content providers,...)
- Private File Access
  - Arbitrary File Read/Overwrite - Path Traversal - ACE
  - ZIP Path Traversal
  - SQLi / Path Traversal on exported content providers
- Android Deeplinks
  - XSS using WebViews
  - Open Redirect
  - Account Takeover
  - Sensitive Data Exposure
- More



MOBSF

Ostorlab

NowSecure™

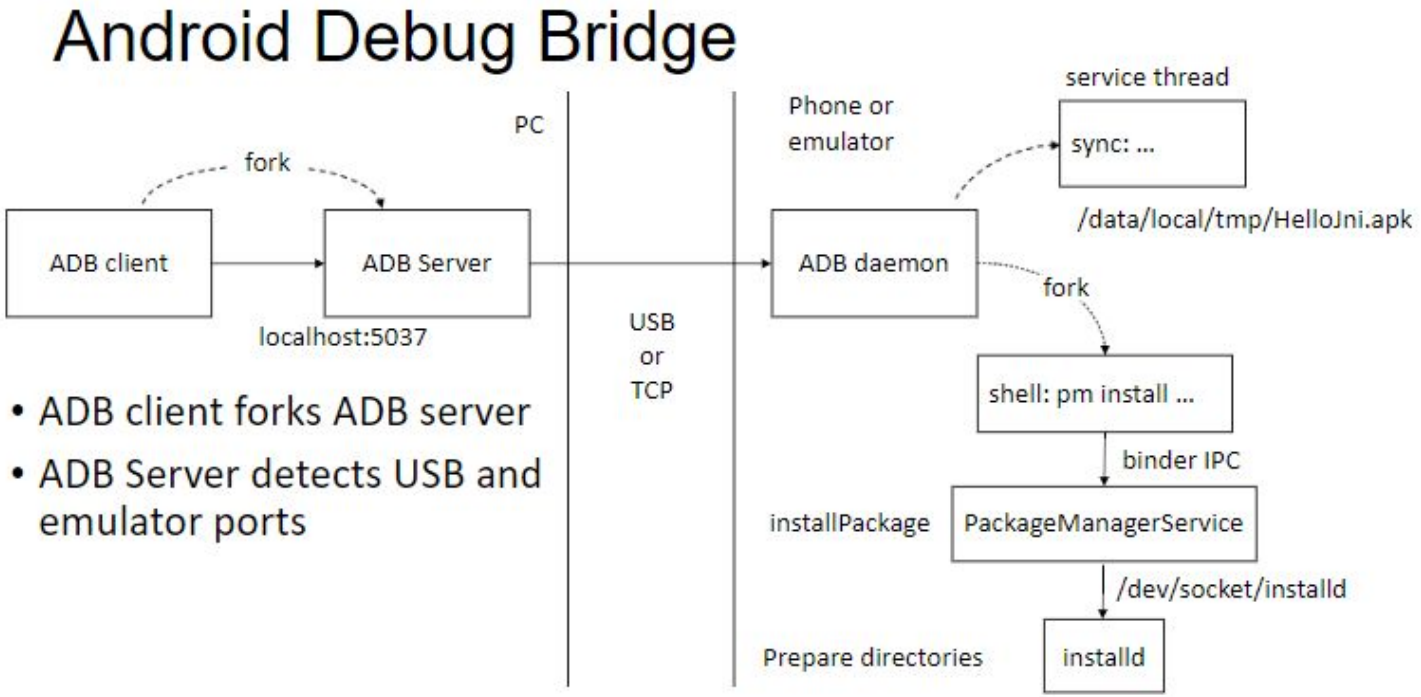
OVERSECURED

Semgrep

nuclei

ADB

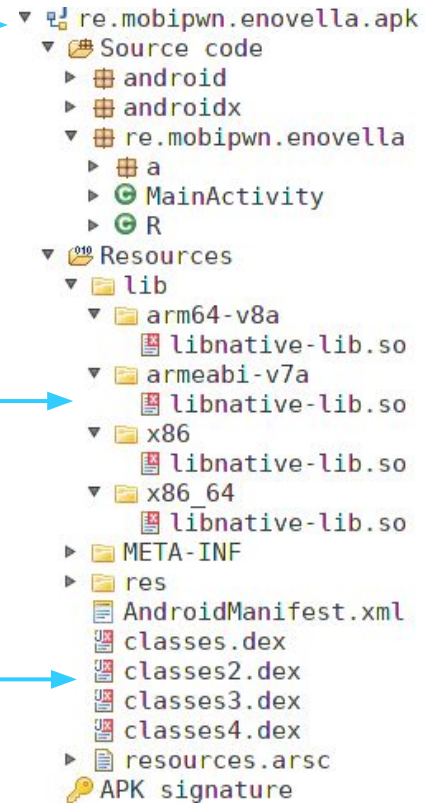
Android Debug Bridge - SDK Platform tools



APK

Android Application Packaging (APK)

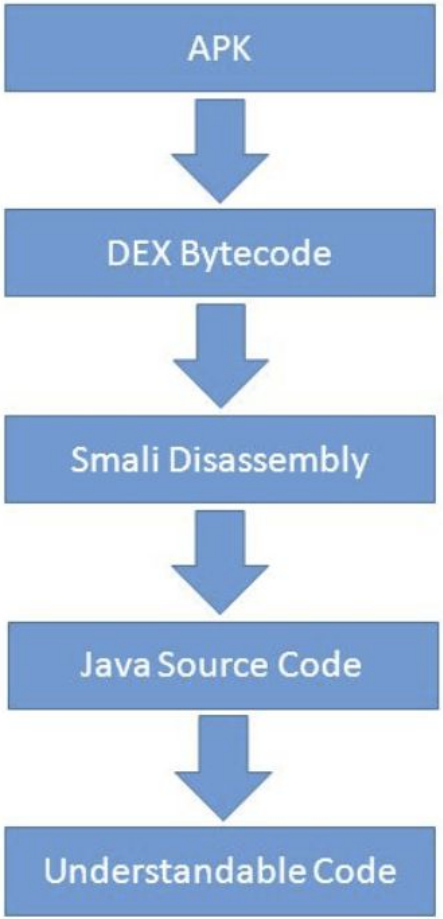
- **APK**
  - APK => ZIP
  - AndroidManifest.xml
  - Assets folder
  - Resources folder
- **Reverse Engineering**
  - APK
  - Native **code**
  - DEX byte**code**



Android RE

Static Analysis

- Static Analysis
  - Understand app logic
  - Find security bugs
  - Reveal critical assets
  - Discover spots to perform dynamic analysis
- Steps
  - Decompile binary code → Pseudo code (readable)
  - Navigate codebase & search for
    - strings, crypto keys, passwords, network traffic, ..
    - obfuscation
      - Rename variables, functions (if stripped)
  - Tamper with the app integrity
    - Intercept TLS/SSL traffic w/ certificate pinning
    - Include your modifications
      - enable logging
      - disable checks
      - GPS locations



Android Reverse Engineering

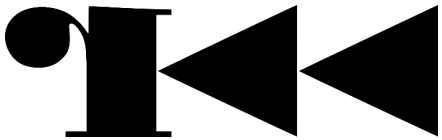
Tools

- Dalvik Bytecode → Smali assembly → Java (Kotlin)
  - **JADX**
  - Bytecode Viewer
  - JEB
  - **Apktool**
  - Baksmali/smali
- Native Binary code → Pseudocode
  - IDA Pro
  - **Radare2**
  - **Ghidra**
  - Binary Ninja
  - Hopper
- Dynamic Binary Instrumentation → Hooking
  - **Frida**
  - Xposed
- Source code
  - Android Studio + AVD emulators
  - VS Code

Android Reverse Engineering

Most powerful OSS tools

- JADX - DEX decompiler
- Ghidra - Native decompiler
- Radare2 - Unix-like reverse engineering framework
- Frida - Dynamic Binary Instrumentation
- R2Frida - The ultimate static analysis on dynamic steroids
- Apktool - APK RE tool
- Mitmproxy - An interactive HTTPS proxy



Android Reverse Engineering

Dynamic Analysis

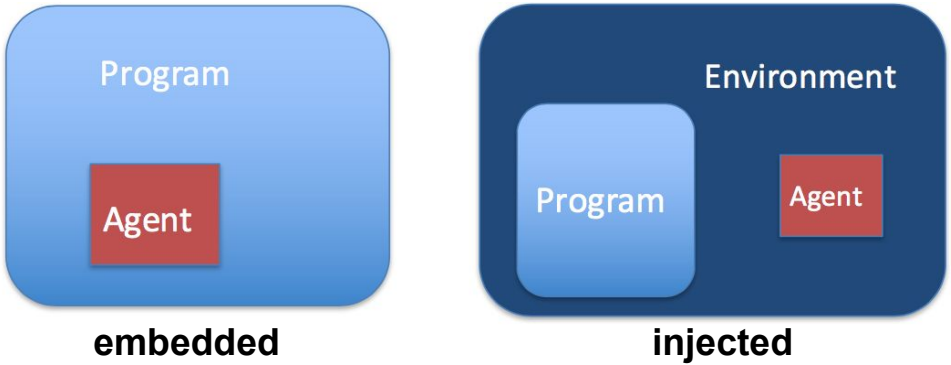
- Dynamic Binary Instrumentation (DBI) toolkit

"A method of analyzing the behavior of a binary application at runtime through the injection of instrumentation code"

  - Injects a JS V8 engine in your target app
  - Supports Linux, MacOS, Windows, Android, iOS, QNX, MIPS
  - Access process memory
  - Hook, trace, intercept functions
  - Change return values, variables, globals, function args,...
  - Call arbitrary functions from imported classes
  - Overwrite function implementations
  - Memory carving on the stack/heap
  - Bypass client-side security checks

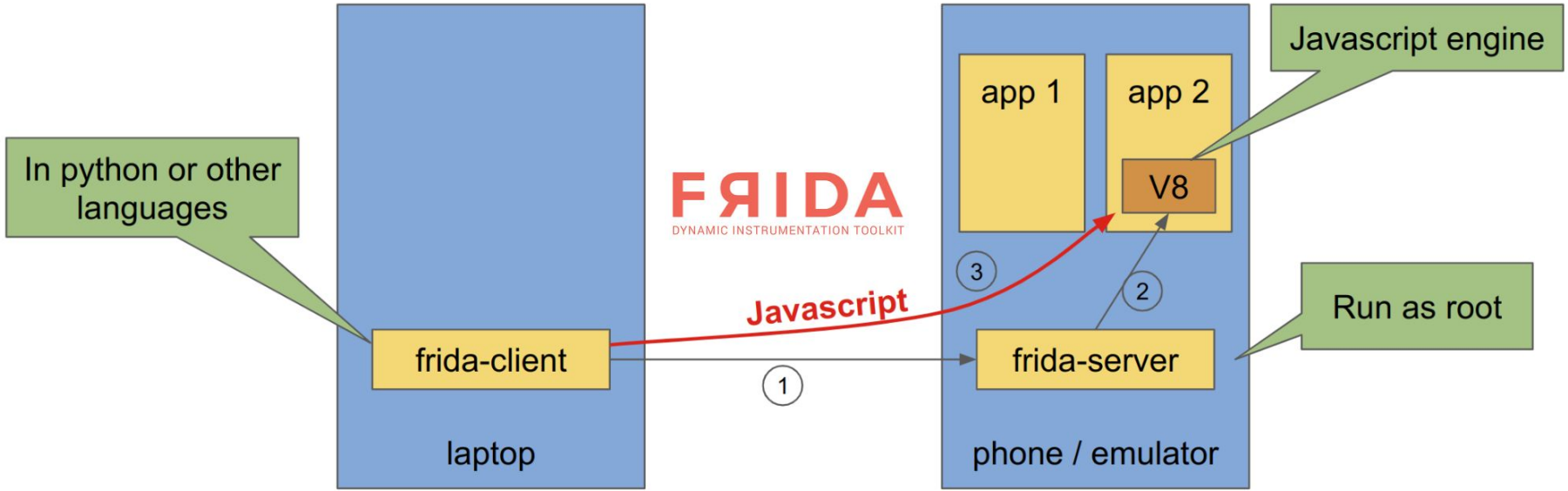
FRIDA

DYNAMIC INSTRUMENTATION TOOLKIT



Android Reverse Engineering

Process Injection via Frida



Source - MOBISEC

Android Reverse Engineering

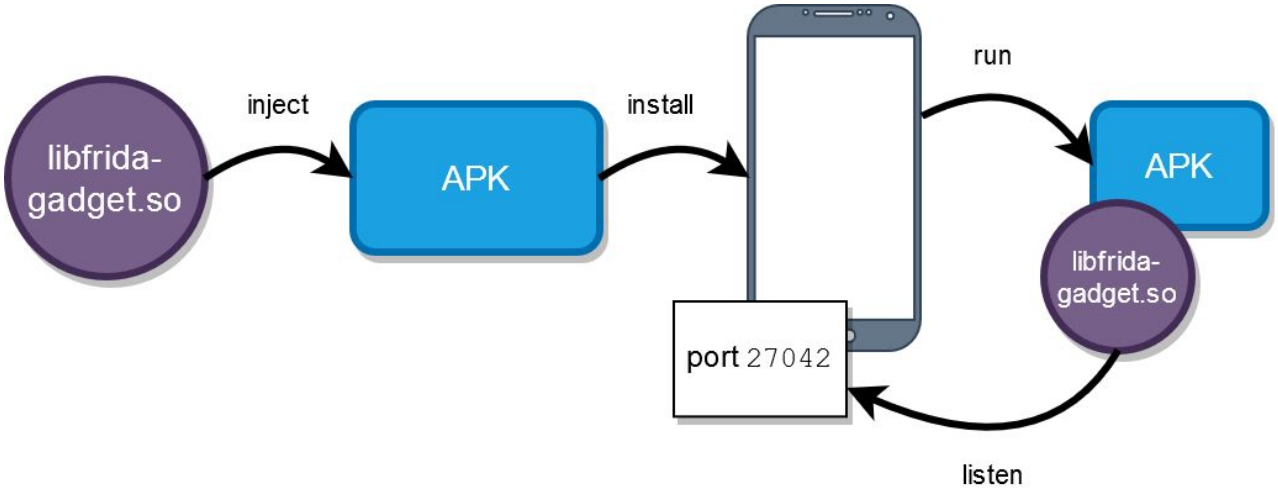
Frida setup

- **Launch Frida server on Android Emulator**
  - $ adb push frida-server-android-x86_64 /data/local/tmp/frida-server
  - $ adb shell
    - generic_x86_64:/ $ su
    - generic_x86_64:/ # cd /data/local/tmp/
    - generic_x86_64:/data/local/tmp # chmod +x frida-server
    - generic_x86_64:/data/local/tmp # ./frida-server -D
- **Spawn/attach to a process from host**
  - $ frida-ps -Uai
  - $ r2 frida://spawn/usb//org.nowsecure.cybertruck

Android Reverse Engineering

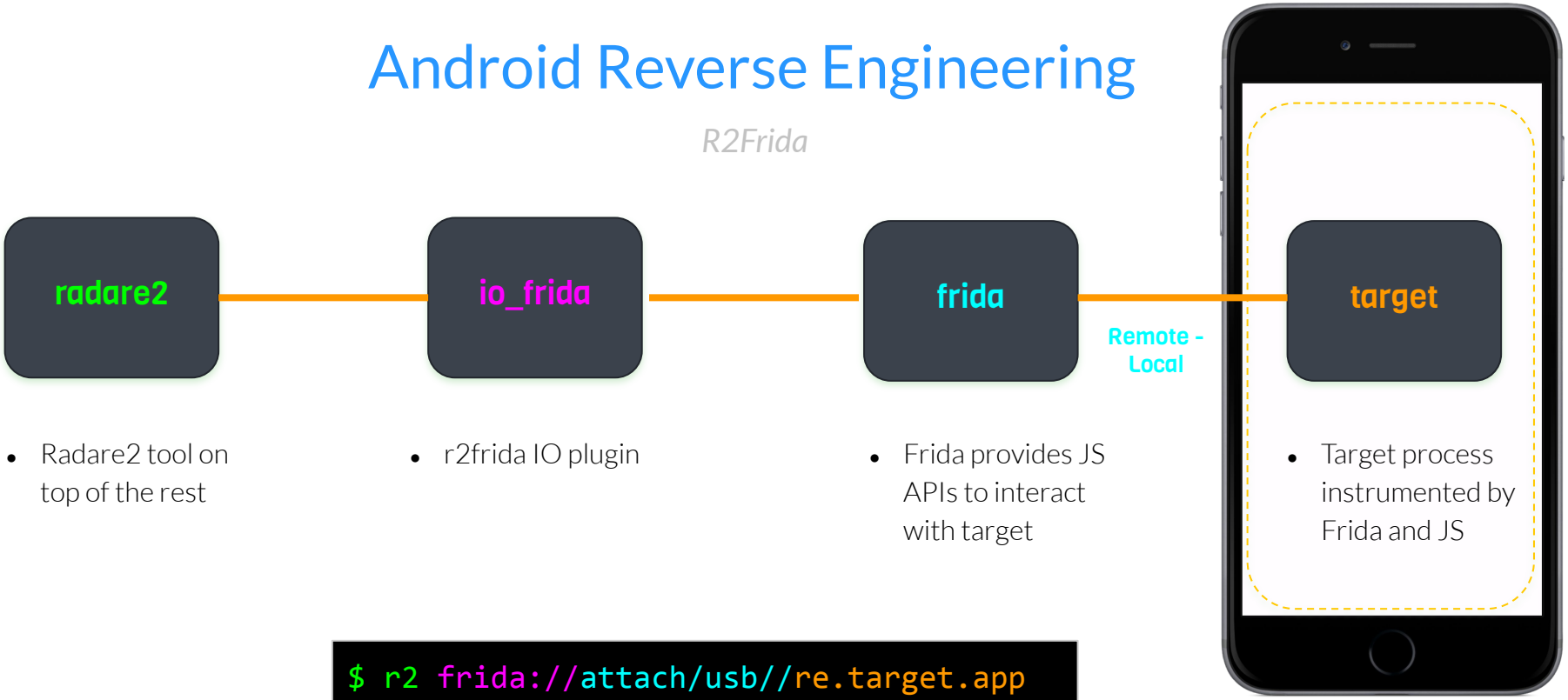
Frida Gadget Injection

- **Frida Gadget - Run on jailed devices without root privileges**
  - Repackage APK injecting a SO and loading it from Java



Android Reverse Engineering

R2Frida



- Radare2 tool on top of the rest
- r2frida IO plugin
- Frida provides JS APIs to interact with target
- Target process instrumented by Frida and JS

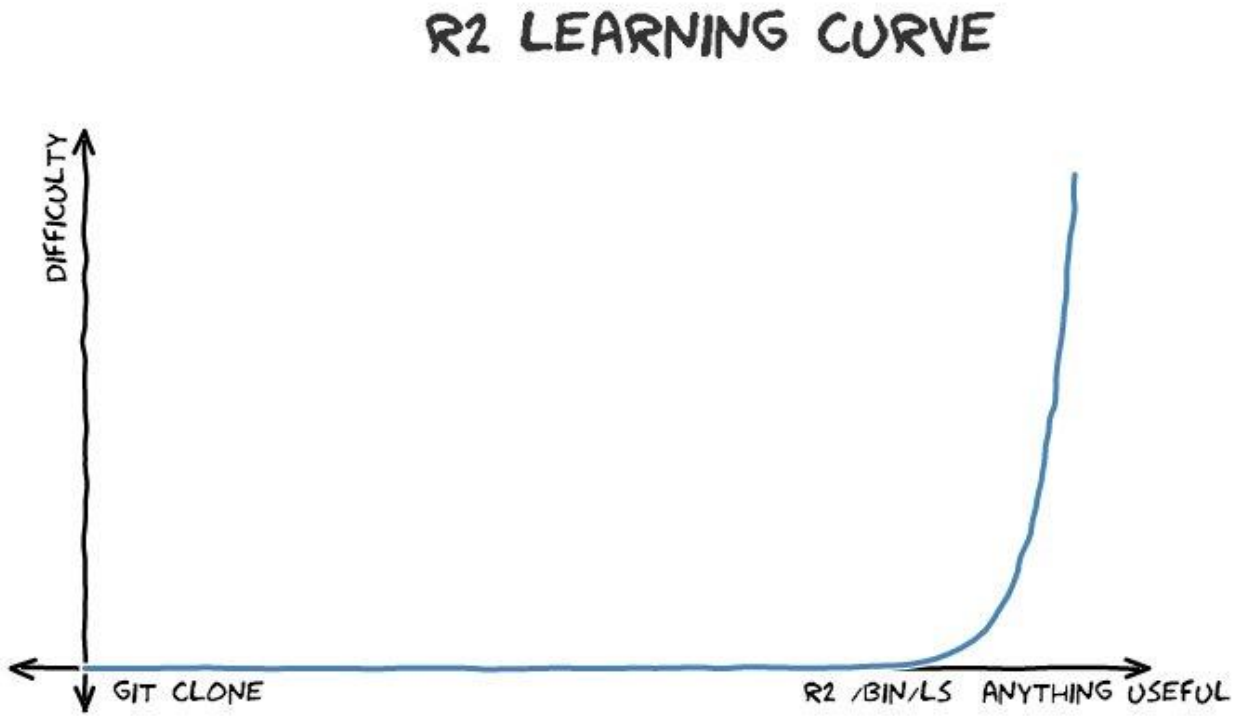
```
$ r2 frida://attach/usb//re.target.app
[0x00000000]> :?
```

Android Reverse Engineering

R2Frida

Android Reverse Engineering

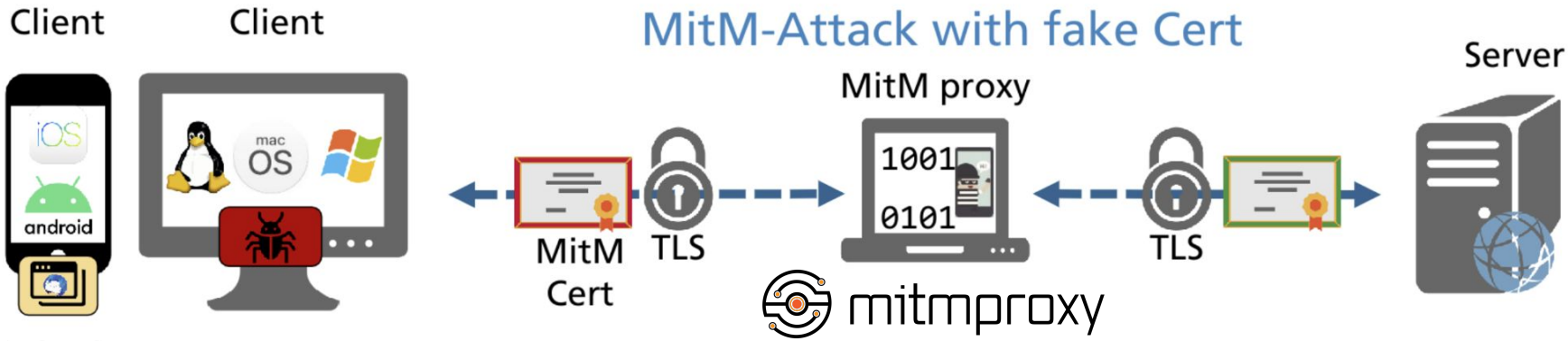
Radare2



Network Analysis

MITM

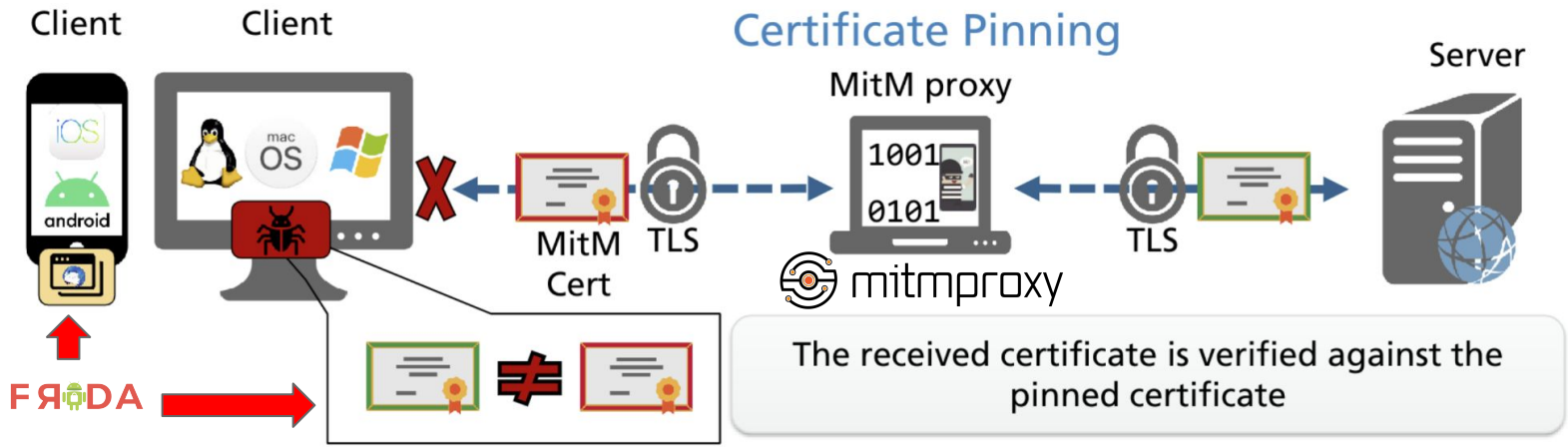
- Forwarding: regular / transparent proxy
  - Burp proxy / Mitmproxy
- Hooking: BoringSSL/OpenSSL read/write data into sockets before encryption
  - Frida-powered Fritap
- From >= Android 7.0, apps does not trust user-certs unless specified in Network Security Config (XML)
  - Adding self-signed certificate to system-certs will bypass this mitigation
    - Systemless root bypasses the read-only `/system` partitions (Magisk modules)



Network Analysis

MITM

- Certificate/ Public Key Pinning - Associate host name to an expected public key certificate
  - Proxy + Frida unpinning scripts
    - Hooking Java/Kotlin SDKs (Tool: Objection)
  - Frida-powered Fritap
    - Hooking TLS native APIs



CyberTruck Challenge App

Can you unlock this uncrackable car keyless system?

https://github.com/nowsecure/cybertruckchallenge22

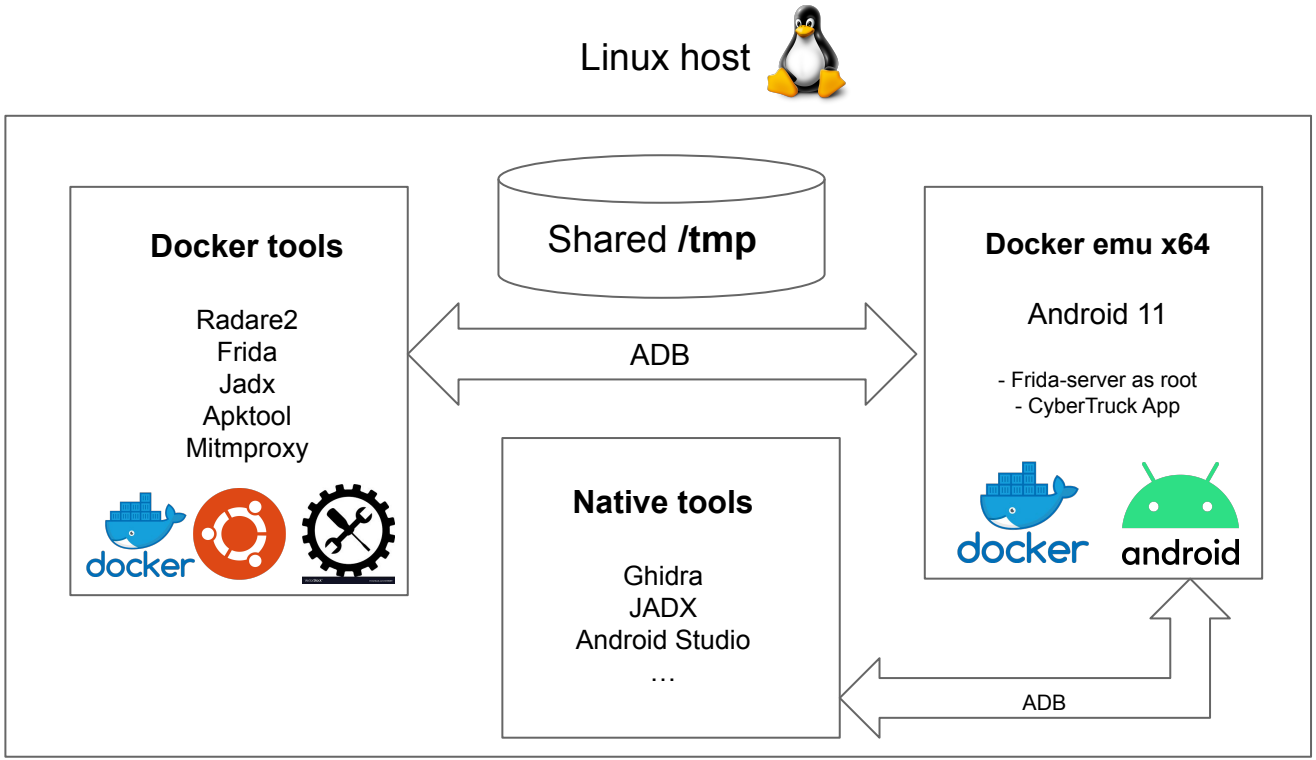
CyberTruck Challenge App

“Unlock your truck with your Android”

- **Android app capable of unlocking vehicles via bluetooth**
  - Material: https://github.com/nowsecure/cybertruckchallenge22
    - folder : **./apk/cybertruck19.apk**
  - Android challenge (3 static + 3 dynamic flags = **6 flags** in total)
  - Run the Android app in Android emulator (Dockerized) or rooted physical device
  - Enable the TamperProof switch if time left

CyberTruck Challenge Android Setup

"Unlock your truck with your Android"



CyberTruck Challenge Android Setup

"Unlock your truck with your Android"

- Material: https://github.com/nowsecure/cybertruckchallenge22:
  - $ git clone https://github.com/nowsecure/cybertruckchallenge22.git
  - $ cd docker
- **Docker Tools - Android RE**
  - Build: $ make build-local OR $ make build (if you're **away** from CyberNAS)
  - Run: $ make shell-local OR $ make shell (if you're **away** from CyberNAS)
- **Docker Emulator - Android 11 x64**
  - Build: $ make build-emu-local OR $ make build-emu (if you're **away** from CyberNAS)
  - Run: $ make shell-emu-local OR $ make shell-emu (if you're **away** from CyberNAS)

```
$ avdmanager create avd -n first_avd --abi google_apis/x86_64 -k "system-images;android-30;google_apis;x86_64"
$ emulator -avd first_avd -no-window -no-audio &   # Press enter if you got questions
$ adb devices
```

CyberTruck Challenge Android Setup

“Unlock your truck with your Android”

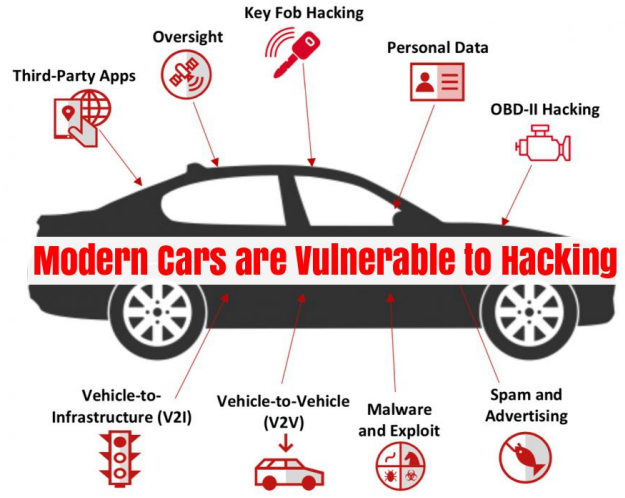
```
$ docker image ls
REPOSITORY                      TAG       IMAGE ID       CREATED          SIZE
cybertruck2:5000/cbtruck        latest    65dae343cf4c   13 hours ago     3.18GB
cybertruck2:5000/androidemu     latest    719db0146c62   11 months ago    5.67GB
```

Android Challenge

Let's play!

Takeaways

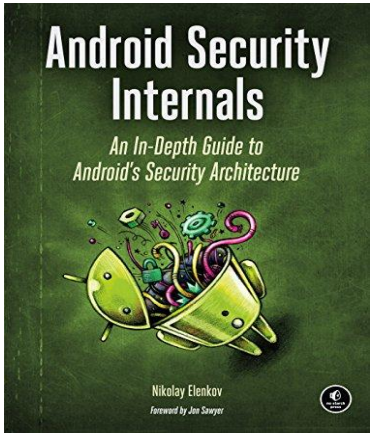
- Keep your software **up-to-date**
- Secure vehicles can be hard → Security by **obscurity** is not the solution
- Focus on the **design** and ensure **strong** key hierarchy → Client-side apps will be eventually compromised
- Follow security **guidelines** → OWASP MSTG
- **Minimum** privilege principle → Reduce the attack surface
- Do not **hardcode** secrets within your code → Use **encryption** at rest
- Employ hardened OS features→ **TrustZone** (TEE)
  - Use **hardware-backed** keystore instead of SW-based implementations to keep secrets
- Ensure proper **randomness** source → Use robust & secure **crypto**
- Implement multi-factor **authentication** (MFA)
- Protect IP → **Code hardening** (Enable ProGuard)
- Enforce certificate pinning to slow down MITM attacks
- Bug **bounty** your application before you got hacked
- Google security → **SafetyNet - Play Integrity API**

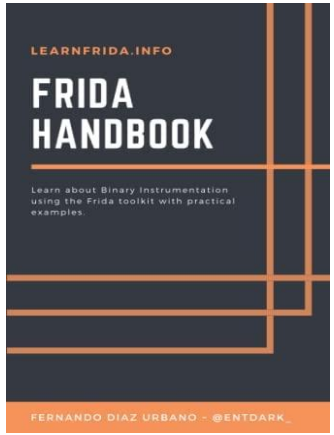


Links

Where to search

- Radare2 && Frida (NowSecure)
- The Mobile Security Testing Guide (MSTG)
- MOBISEC lectures
- Android App Reverse Engineering 101
- Awesome Frida && Frida CodeShare
- RedNaga Security - Awesome Mobile CTFs
- A bunch of mobile security blog posts on the Internet





THANK YOU!
Q&A

Eduardo Novella

Mobile Security Research Engineer

enovella@nowsecure.com

@NowSecureMobile

@enovella_

Special thanks to

@RomainKraft @fs0c131y @Hexploitable

for providing feedback on the crackme