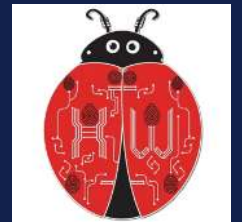


Connected Car Hacking training

03. In-Vehicle Infotainment (IVI) hacking

Hardware.io USA 2026



Quarkslab

In-Vehicle Infotainment (IVI)



- ▶ In-Vehicle Infotainment (IVI) unit is the ECU in charge of most of the **multimedia** and **customer-oriented features** inside the vehicle
- ▶ Primary purpose is to enhance the driving experience by offering a wide range of features:
 - ▶ **Navigation**
 - ▶ **Entertainment**
 - ▶ **Connected applications**
 - ▶ **Interaction with driver/passenger smartphones**
- ▶ This ECU has the **widest range of attack surfaces**, being a target of choice



Illustration: [link]

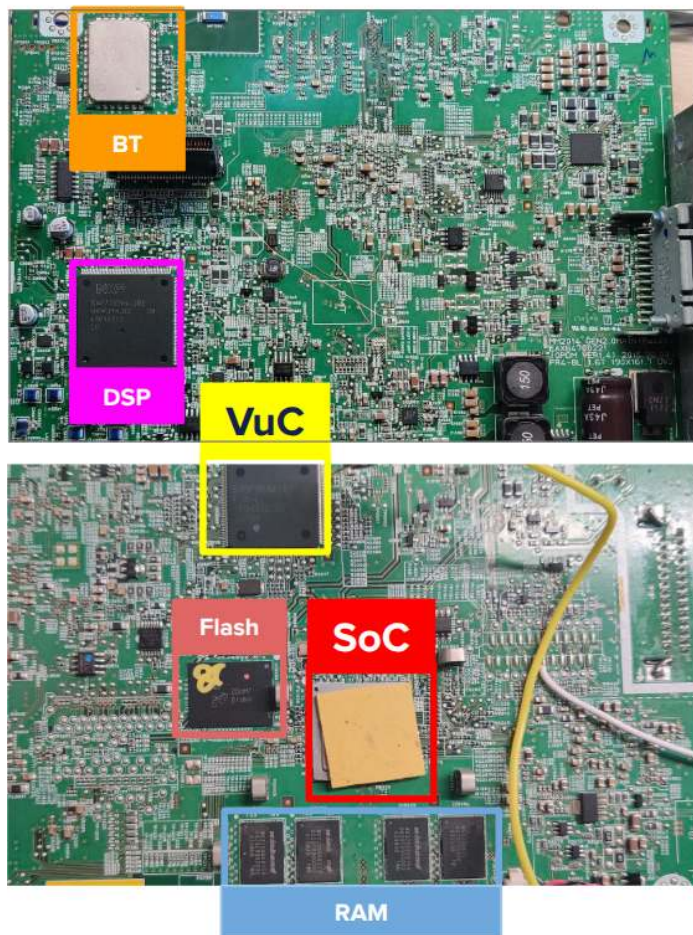
Key elements of an **IVI** are:

- ▶ **Dedicated vehicle microcontroller:** or **VuC** (Vehicle uC), which interacts with the on-board networks like CAN and handles diagnostic requests
- ▶ **Main System on Chip (SoC):** runs the operating system (OS)
- ▶ **External Flash memory** and **RAM:** for the SoC
- ▶ **Digital Signal Processor (DSP):** process audio signals
- ▶ **External connectivity chip:** for wireless connectivity and positioning (GNSS, Wi-Fi, Bluetooth, ...)
- ▶ Some IVI have **integrated display**, others are separated in 2 ECUs, connected with **LVDS** cable (**Low Voltage Differential Signal**)

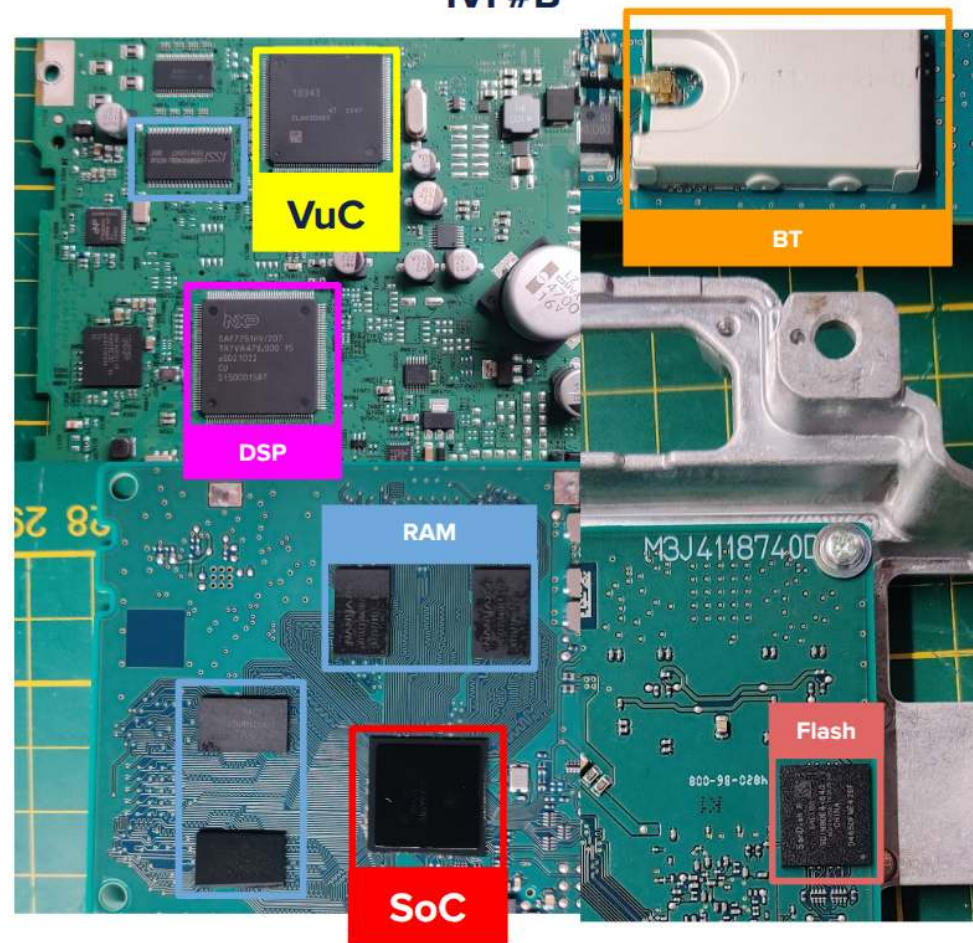
Hardware analysis



IVI #A



IVI #B



IVI firmware extraction

Embedded firmwares

To fully assess an IVI at least 2 firmwares need to be analysed:

- ▶ **VuC** with its **embedded firmware** for in-vehicle communications
- ▶ **SoC** running an **operating system**, which manages wireless/USB connectivity and system updates

Main operating system used in IVI are:

- ▶ **Android**
- ▶ **QNX**
- ▶ **Automotive Grade Linux**



QNX



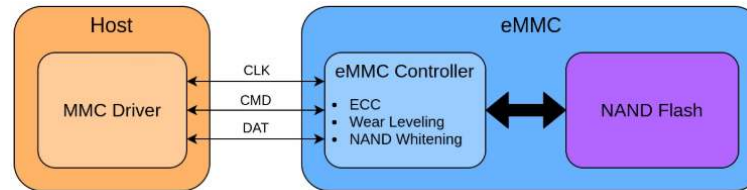
The operating system is commonly stored on an **eMMC** memory chip, with a transition to **UFS** memory on newest IVIs

Firmware storage - eMMC



To store an Operating System, **eMMC chips** are often used.

eMMC have a dedicated controller that handles the inner **NAND Flash** memory and performs wear levelling and error correction code of this memory.



The various pins of an eMMC are:

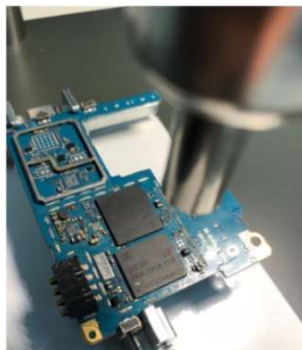
- ▶ **CMD**: command line
- ▶ **CLK**: clock line
- ▶ **DAT[0-7]**: data lines
- ▶ **VCC**: input voltage for the flash storage (1.8V and/or 3.3V)
- ▶ **VCCQ**: input voltage for the flash controller (1.8V and/or 3.3V)

Firmware storage - eMMC extraction

eMMC uses the same protocol as **SD-card**, and could operate in 1-bit, 4-bit or 8-bit mode.

Content of an **eMMC** could be extracted by removing the chip from the PCB using **chip-off** technique and a memory programmer.

However, by finding pins **CMD**, **CLK** and **DAT[0-4]**, it is possible to solder them onto an SD adapter and read and write it using an SD-card reader. This technique is useful to keep the device working and inject/extract data and binaries. To avoid conflict on the **CLK** line, the main chip of the device need to be **halted** or **unpowered**.



Secure Digital Input Output

Secure Digital Input Output, or **SDIO**, is a synchronous protocol enabling high-speed data transfer with peripheral devices.

It requires a **command** (CMD), a **clock** (CLK) line and **several data lines** (DATx) to support 8-bit, 4-bit or 1-bit data bus.

It provides faster speeds than SPI and I2C, making it ideal for high bandwidth applications.

This protocol is used by SD cards and eMMC memory chips.

Illustration: [\[link\]](#), [\[link\]](#)

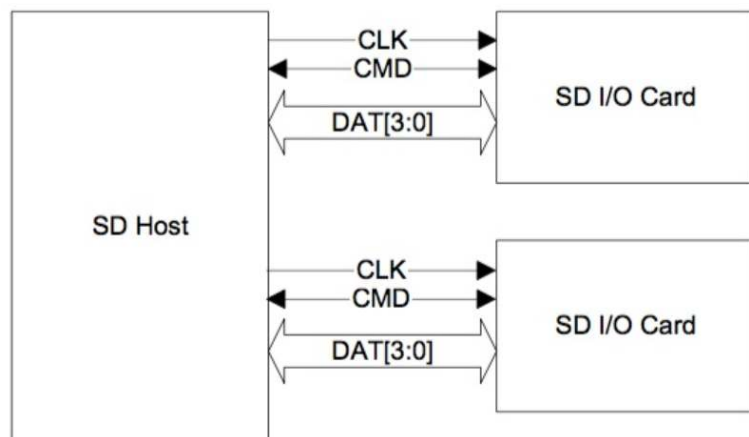
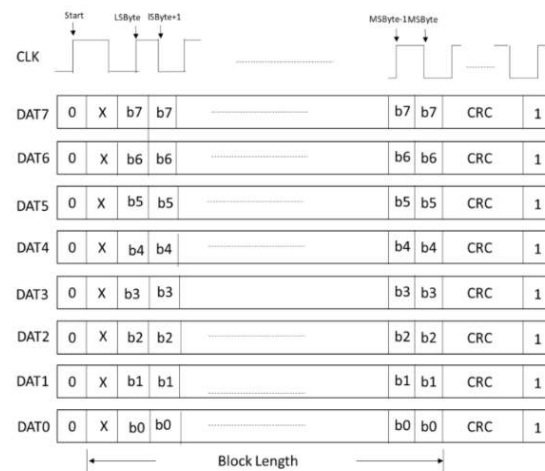


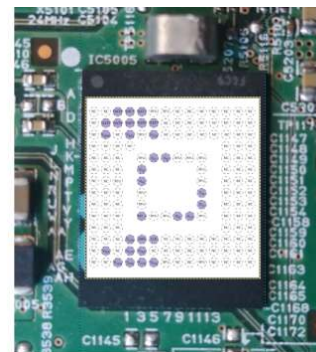
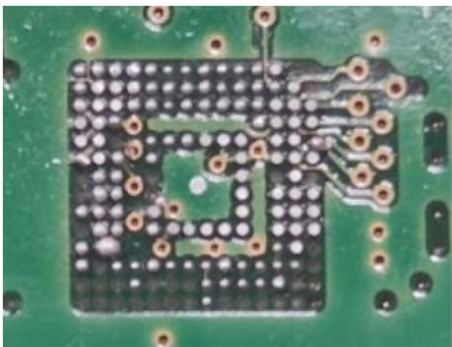
Figure 2-1 Signal connection to two 4-bit SDIO cards



Firmware storage - eMMC pins identification

To find the required pins to perform the extraction using a SD-card reader, there are several techniques:

- ▶ **Removing the eMMC** on a second device, then find vias, debug pins or components to solder to
- ▶ Searching for **array of pull-up/pull-down resistors** around the eMMC, which are good candidates for **DAT** pins
- ▶ **Aligning pictures** of the two sides of the PCB then **overlaying the pins** of a spare eMMC over the real one to visually find candidate traces

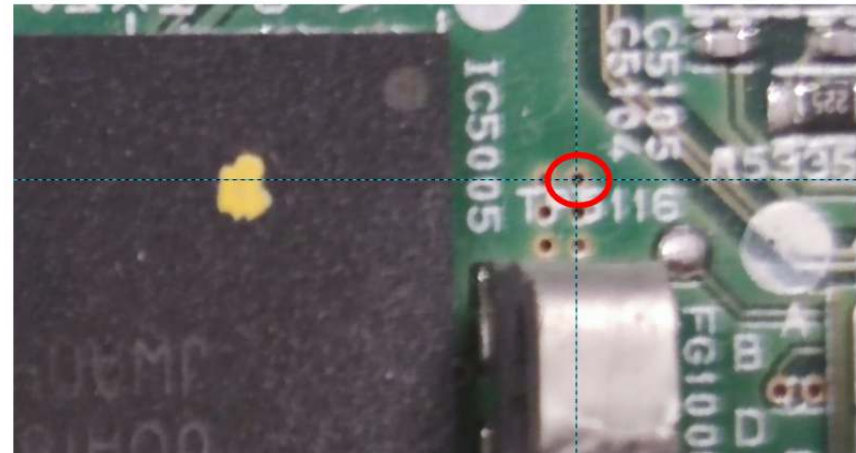
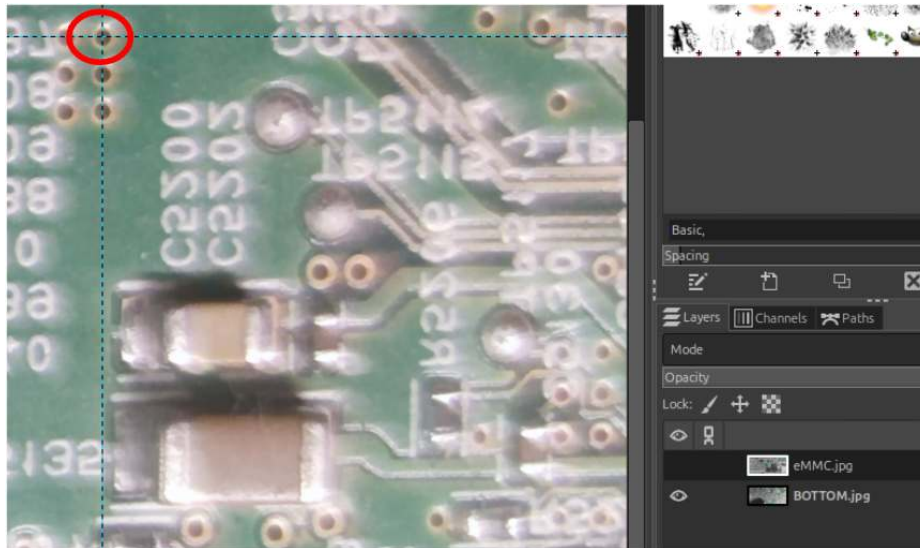


Firmware storage - eMMC pins identification



To overlay and align images, we will use **GIMP** and its **unified transform tool**

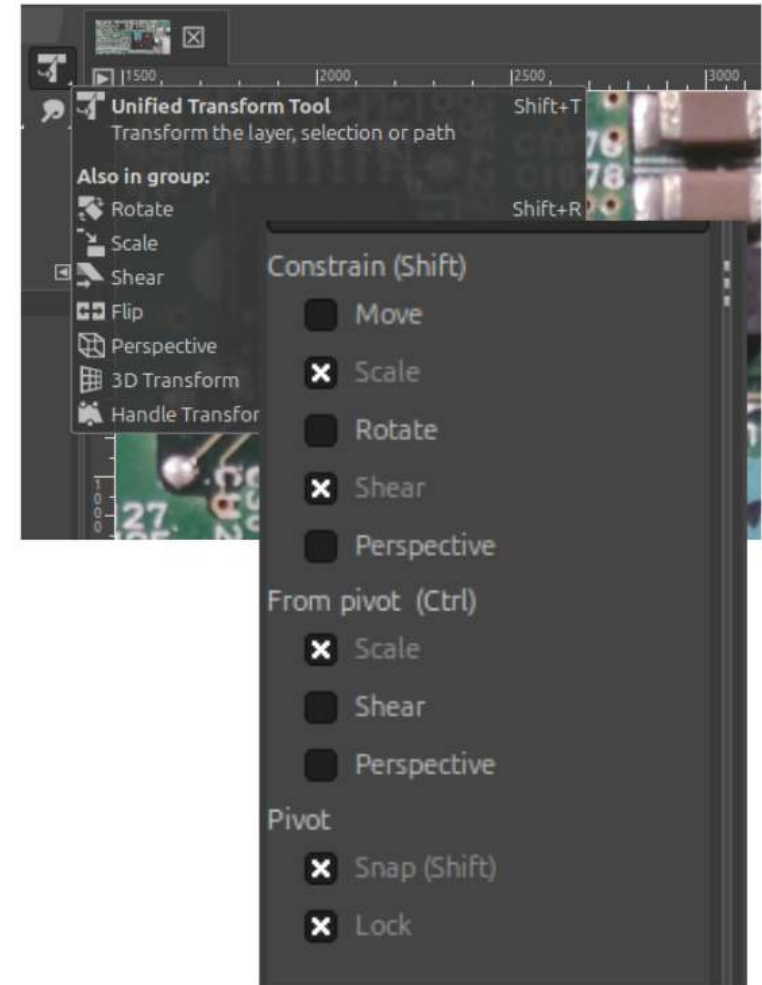
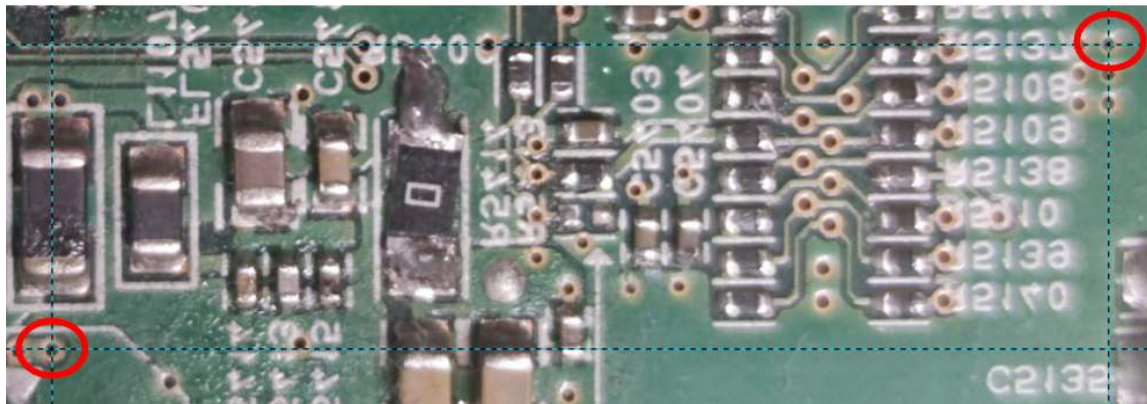
- ▶ **1.** Open one of the picture of the PCB and add the second one in a new layer
- ▶ **2.** Using **guides**, mark a **1st reference** (an easily identifiable vias for example) on the bottom layer
- ▶ **3.** **Align the upper layer** on this first mark



Firmware storage - eMMC pins identification

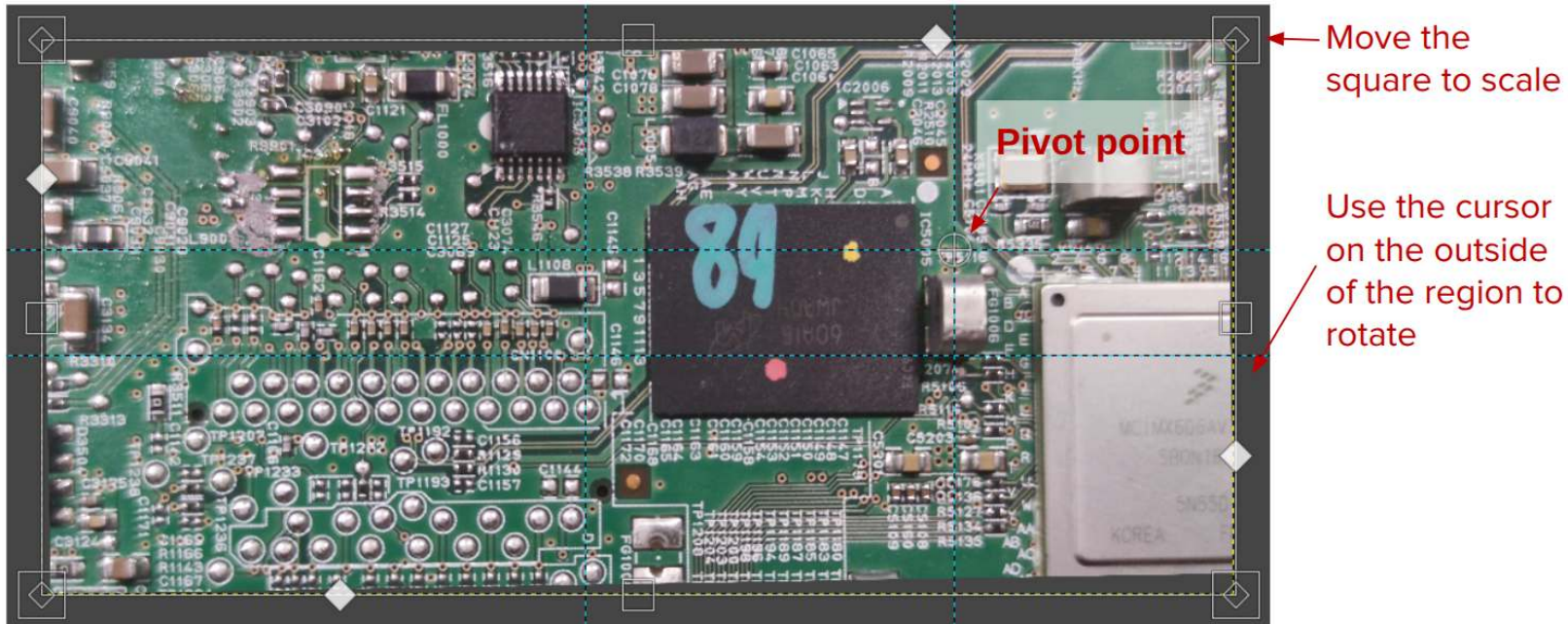


- ▶ 4. On the bottom image, **select a second reference point**
- ▶ 5. Open the **Unified Transform Tool**
- ▶ 6. Check **Pivot > Lock**, **Pivot > Snap** and **From Pivot > Scale**



Firmware storage - eMMC pins identification

- ▶ 7. Click on the upper image to activate the **Unified Transform Tool**
- ▶ 8. Move the **pivot point** (crosshair) on the **first reference point**
- ▶ 9. **Scale** and **rotate** the image to match the **second reference point**





🧪 Lab 1 - Locating eMMC pins



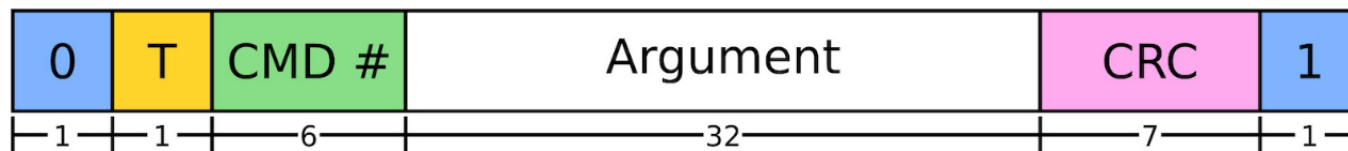
Lab 1 - Locating eMMC pins

For this lab, you'll practice visual identification of the **PCB** of an **IVI** to find components or test points that could be used to extract the **eMMC** memory

- ▶ Complete challenges **eMMC identification** to **eMMC pin identification 2** in **Infotainment - firmware extraction**

Firmware storage - eMMC pins identification - initialisation

- ▶ It is also possible to find candidates using a **logic analyser**
- ▶ In 1-byte mode, only **CMD**, **CLK** and **DAT0** are required
- ▶ During **initialisation**, **DAT0** is the only data line active
- ▶ Commands sent through the **CMD** line use the following structure:



🧪 Lab 2 - Identifying eMMC pins



Lab 2 - Identifying eMMC CMD/CLK/DAT0

- ▶ A **logic analyser capture** was performed on test points around an eMMC during boot
- ▶ Complete challenge **Signal analysis** from **Infotainment - firmware extraction**

Firmware storage - eMMC extraction - disabling auto-mount

- ▶ Using an **SD-card reader**, it is possible to mount the eMMC on your computer
- ▶ Beware of the **voltage** of the eMMC, check the datasheet, some only work with **1.8V**, requiring special adapter
- ▶ Also, it is recommended to **disable auto-mount**, to avoid writing metadata on the eMMC, which can trigger secure boot protection
- ▶ On Debian, type:

```
$ sudo systemctl stop udisks2  
$ sudo systemctl disable udisks2
```





Firmware storage - eMMC extraction - dump

- ▶ Use **dmesg** command to find the path to the eMMC (**/dev/sdX**, **/dev/mmcblkX**)
- ▶ The last number is the **partition**, omitting it will dump the whole content of the eMMC

```
$ sudo dmesg
[18989.012121] scsi 0:0:0:0: Direct-Access Multiple Card Reader 1.00 PQ: 0 ANSI: 0
[18989.012421] sd 0:0:0:0: Attached scsi generic sg0 type 0
[18989.232914] sd 0:0:0:0: [sda] 15204352 512-byte logical blocks: (7.78 GB)
[18989.233345] sd 0:0:0:0: [sda] Write Protect is off
[18989.233348] sd 0:0:0:0: [sda] Mode Sense: 03 00 00 00
[18989.233728] sd 0:0:0:0: [sda] No Caching mode page found
[18989.233733] sd 0:0:0:0: [sda] Assuming drive cache: write through
[18989.236785] sda: sda1 sda2 sda3 \ [18989.237038] sd 0:0:0:0: [sda] Attached SCSI removable disk
[18989.525477] sda: sda1 sda2 sda3 <- 3 partitions

$ sudo dd if=/dev/sda of=./dump.bin bs=512 status=progress <- Full dump
```



Firmware storage - eMMC extraction - partial

- Using **fdisk**, you can list the various partitions to select a target one to speed up the dumping process

Listing all partitions:

```
$ sudo fdisk -l
```

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/sda1		8192	532479	524288	256M	c	W95 FAT32 (LBA)
/dev/sda2		532480	6846463	6313984	3G	83	Linux
/dev/sda3		6846464	7434239	587776	287M	83	Linux
/dev/sda4		7434240	31115263	23681024	11,3G	83	Linux

To dump partition #3 using /dev/sda3:

```
$ sudo dd if=/dev/sda3 of=./dump.bin bs=512 status=progress
```

To dump partition #3 with block address

```
$ sudo dd if=/dev/sda of=./dump.bin bs=512 status=progress seek=6846464 count=587776
```



Firmware - exploring dump

- ▶ You can quickly mount dumped partition with command `losetup`:

```
$ sudo losetup -fP mydump.bin
```

- ▶ To mount it as **read-only**, add `-r`:

```
$ sudo losetup -fP -r mydump.bin
```

- ▶ To find the `/dev/loop` where the dump is mounted:

```
$ lsblk
```

- ▶ To remove the mounted partition from the `/dev/loop`:

```
$ sudo losetup -d /dev/loopXX
```

🧪 Lab 3 - Dumping the eMMC



Lab 3 - Dumping the eMMC

In this lab, you'll extract one partition of the **Infotainment unit** memory

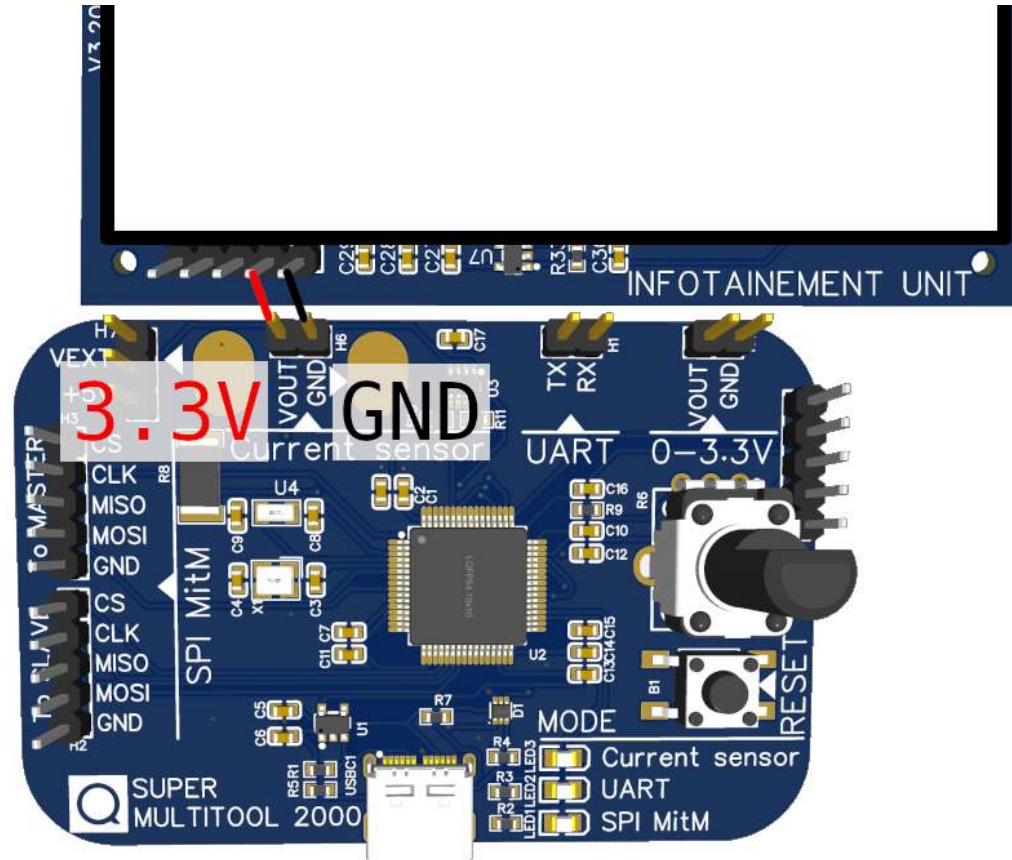
- ▶ Using the provided **SD card adapter**, dump the content of the **third partition** of the **eMMC**
- ▶ To do so:
 - ▶ Unplug the **10-pin ribbon cable**
 - ▶ Connect the **SD card adapter**
 - ▶ Power the **Infotainment** using the **Super Multitool 2000** as shown on next slide
- ▶ Complete the remaining **Infotainment - firmware extraction** challenges



⚗ Lab 3 - Dumping the eMMC



Lab 3 - Dumping the eMMC

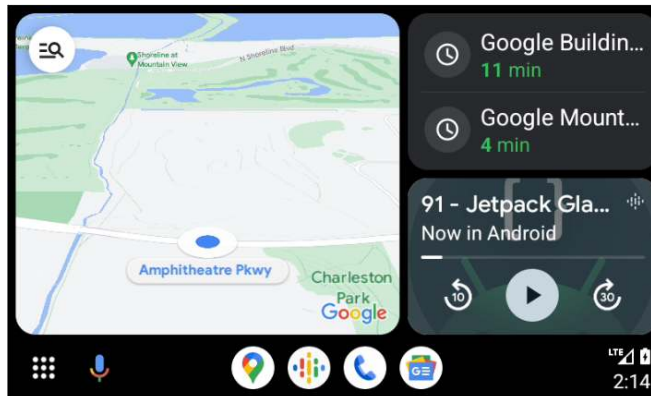


Infotainment main OS



- ▶ Infotainment unit commonly runs **Linux based distributions**, like:
 - ▶ **QNX**
 - ▶ **Automotive Grade Linux**
 - ▶ **Android Automotive OS**
- ▶ For this training, we will focus on **Android Automotive OS** as it has some interesting specifics

Illustration: [\[link\]](#)



Android 101



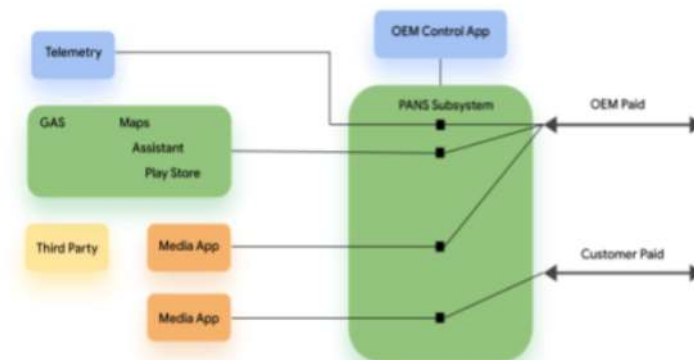
Android Automotive specifics

- ▶ In **Android Automotive**, applications are **car-optimized** for safety in road usage
- ▶ It handles **interaction** with some part of the vehicle, including:
 - ▶ HVAC
 - ▶ Instrument cluster display (navigation, music)
 - ▶ Battery charging
 - ▶ Locking system
 - ▶ Telematic Control Unit
- ▶ It can support some base Android services, known as **Google Android Automotive Services (GAS)**, like the Play Store, Maps...
- ▶ If it offers different features, it is based on the **Android Open Source Project (AOSP)**

Per-Application Network Selection

- ▶ **Per-Application Network Selection** is an API that is specific to **Android Automotive**
- ▶ It allows network management to route data traffic of **OEM-allowed application** to **OEM network**
- ▶ It ensures **critical applications** could be connected using an **OEM-paid data plan** and other application to use user-paid data plan
- ▶ It expands **NetworkCapabilities** by adding **NET_CAPABILITY_OEM_PAID** and **NET_CAPABILITY_OEM_PRIVATE**

Illustration: [\[link\]](#)



*actual deployments may vary by OEM

Android key concepts



- ▶ **Android** runs applications in a **sandboxed environment** (per-application UID)
- ▶ It has **permission-based** access control
- ▶ Applications can communicate through **Inter-Process Communication (IPC)**
- ▶ Core packages, applications and services are signed using **keys**, granting specific privileges
- ▶ The key that signed core packages is known as the **platform key**
- ▶ **ADB** is a tool allowing shell access to an Android device, that is normally disabled in production
- ▶ An Android device is set in **user build** for production and **userdebug** during development



- ▶ In **Android**, Applications and Services are provided in **APKs**, which are **ZIP** archives
- ▶ Key element of an APK is its **AndroidManifest.xml**, which defines application behavior, components, permissions...
- ▶ **JADX** allows to unpack **APK** and analysis **Java/Kotlin** code
- ▶ An APK could also integrate **native libraries** which are C/C++ compiled code developed for specific uses
- ▶ By default, **applications are restricted**, unless proper permissions are set (Internet access, read/write access to shared folder...)
- ▶ Basic APK has no privilege, **unless signed by the platform key**, being then considered as system application



- ▶ The **Android Manifest** is an XML file describing the behavior of an APK and its permissions
- ▶ Some **permissions** could be dangerous from a security point of view
- ▶ It also declares the **BroadcastReceivers**, for **Intents** that could be used to interact with the application



Android Manifest - example

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.app">
    <!-- Permissions -->
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:label="ExampleApp"
        android:usesCleartextTraffic="true"
        android:theme="@style/Theme.App">
        <!-- Receiver for system BOOT event -->
        <receiver android:name=".BootReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
        <!-- Receiver for a custom Intent -->
        <receiver android:name=".CustomReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="com.example.CUSTOM_ACTION" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

XML



Android's quick win: Intents

- ▶ For **Inter-Process Communication** application could use **Intents**
- ▶ Incorrectly secured **Intents** could allow non-privileged applications to execute commands or alter the behavior of the target application
- ▶ Exposed **BroadcastReceivers** can be found by looking at the various **AndroidManifest.xml**

Sending Intent:

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");
startActivity(sendIntent);
```

Java

Receiving Intent:

```
MyBroadcastReceiver myBroadcastReceiver = new
MyBroadcastReceiver();
IntentFilter filter = new
IntentFilter("com.example.snippets.ACTION_UPDATE_DATA");
ContextCompat.registerReceiver(context, myBroadcastReceiver,
filter, receiverFlags);
```

Java



Android's quick win: Services & AIDL

- ▶ **Inter-Process Communication** and **Services** can have **AIDL** interfaces
- ▶ They will expose a set of functions that can be executed through **ADB** access or by **third-party APKs**
- ▶ As the **AIDL** file will not be available in the compiled application, look for switch-case statements

AIDL example:

```
// IRemoteService.aidl
package com.example.android;

/** Example service interface */
interface IRemoteService {
    /** Request the process ID of this service. */
    int getPid();

    /** Demonstrates some basic types that you can use as parameters */
    int basicTypes(int anInt, byteArray[] aByteArray);
}
```



Android's quick win: Services & AIDL

Compiled result:

```
package com.example.android;

public interface IRemoteService extends IInterface {
    public static class Default implements IRemoteService {
        @Override
        public boolean onTransact(int i, Parcel parcel, Parcel parcel2, int i2) throws RemoteException {
            switch (i) {
                case 1:
                    int getPId_ = getPId();
                    parcel2.writeNoException();
                    parcel2.writeInt(getPId_);
                    return true;
                case 2:
                    int i1 = parcel.readInt();
                    byte[] bArr = parcel.createByteArray();
                    int basicTypes_ = basicTypes(i1, bArr);
                    parcel2.writeNoException();
                    parcel2.writeInt(basicTypes_);
            }
        }
    }
}
```

Java



Android's quick win: `getRuntime`

- ▶ From the **System** class, command **getRuntime** executes the provided command on the system
- ▶ A **privileged** application will have **privileged** rights for execution
- ▶ Searching for **insecure calls** to `GetRuntime` is always recommended

```
Process process = Runtime.getRuntime().exec("cp myFile /var/data/tmp/%s", target_name);
```

Java

Getting deeper



If you're new to Android and want to get deeper in, look at the great [Hextree.io](https://www.hextree.io) courses, Android ones are free !



Engineering mode

Engineer mode



Engineer modes are often present in IVI, allowing after-sales **analysis/debugging** of units

- ▶ It could also be used to **trigger/modify specific features** of the IVI
- ▶ To activate this mode, it may be required to:
 - ▶ Perform specific pattern on the touch screen
 - ▶ Insert an USB key with a specific file/folder
 - ▶ Use UDS commands
 - ▶ ...



Illustration: [link], [link]

Finding how to activate engineer mode

- ▶ For many IVI, method to activate engineer mode can be found on **forums/websites**
- ▶ On **Android-based IVI**, search for **ClickListener** or **onLongClickListener** usage or **Intent** which could be named with **engineer** or **debug**
- ▶ On some devices, using a **UDS WriteDataByIdentifier** request could activate such mode

Illustration: [link]



🧪 Lab 4 - Accessing Engineer Menu



Lab 4 - Accessing Engineer Menu

For this lab, you'll reverse engineer an **APK** running on the **Infotainment unit** to find how to activate the **engineer mode**

- ▶ Complete challenges **Infotainment - Engineer Mode**

Analysing USB interface

Scanning supported devices



- ▶ **IVI** may support various **USB devices**, scanning them may give us hints on some capabilities (keyboard, mass storage, usb to ethernet support, ...)
- ▶ Well known device for this task is the **Facedancer21** board and **umap2**, which are old but reliable tools
- ▶ Using the **Facedancer** Python library, it is simple to emulate USB devices to assess IVI with compatible hardware, like the ones below
- ▶ We recommend using board like **Cynthion** or the **Hydradancer**, which offer higher data rate and flexibility than the Facedancer21 and the GreatFET

Illustration: [\[link\]](#), [\[link\]](#)

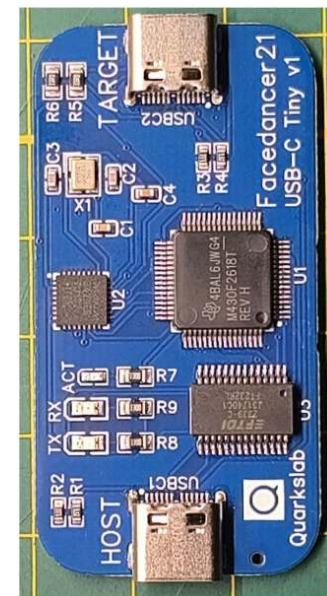


Scanning supported devices

- ▶ **umap2** from NCC group is the go-to tool to scan supported USB devices using a **Facedancer21**
- ▶ Connect the **HOST** port of the **Facedancer21** to your computer and the **TARGET** one to the device under test

To simply scan a USB host, we can use the following command:

```
$ umap2scan -P fd:/dev/ttyUSB0
[.. redacted for readability ..]
[INFO ] have been waiting long enough (over 6 secs.), disconnect
[INFO ] [Max342xPhy] Disconnected device SmartcardDevice
[INFO ] [Max342xPhy] Disconnect called when already disconnected
[ALWAYS] -----
[ALWAYS] Found 2 supported device(s):
[ALWAYS] 1. audio
[ALWAYS] 2. cdc_acm
```





- ▶ With the Facedancer 3.0 library, it is possible to **emulate a USB device**
- ▶ Working with this feature could help to **scan for supported characteristic**, find non-filtered **Vendor ID** and **Product ID** ...
- ▶ For instance, we can emulate a **mass storage device** and **display live content written in clusters** to follow dump logs without having to remove a USB stick between various attempts of an attack
- ▶ Facedancer's Github repository holds several [examples](#) to jump into USB emulation



Emulating devices - example

```
from facedancer import *
from facedancer import main

@use_inner_classes_automatically
class HackRF(USBDevice): """ Device that emulates a HackRF enough to appear in ``hackrf_info``. """
    product_string      : str = "HackRF One (Emulated)"
    manufacturer_string : str = "Great Scott Gadgets"
    vendor_id           : int = 0x1d50
    product_id          : int = 0x6089

    class DefaultConfiguration(USBConfiguration):
        class DefaultInterface(USBInterface):
            pass

    @vendor_request_handler(number=14, direction=USBDirection.IN)
    @to_device
    def handle_control_request_14(self, request):
        request.reply([2])

    @vendor_request_handler(number=15, direction=USBDirection.IN)
    @to_device
    def handle_get_version_request(self, request):
        request.reply(b"Sekret Facedancer Version")
```

Python



Emulating devices - example

Host

```
$ export BACKEND=goodfet    # For Facedancer 21 support
$ python3 ./hackrf-info.py # From the Facedancer examples
INFO | __init__ | Starting emulation, press 'Control-C' to disconnect and exit.
[...]
```

Target

```
$ sudo dmesg -w
[12779.953944] usb 1-1.2: new full-speed USB device number 54 using xhci_hcd
[12781.105002] usb 1-1.2: New USB device found, idVendor=1d50, idProduct=6089, bcdDevice= 0.00
[12781.105008] usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[12781.105011] usb 1-1.2: Product: HackRF One (Emulated)
[12781.105013] usb 1-1.2: Manufacturer: Great Scott Gadgets
[12781.105016] usb 1-1.2: SerialNumber: S/N 3420E
```

Lab 5 - Scanning and emulating USB device

In this lab, you'll scan USB devices supported by the **Infotainment unit** and emulate a **mass storage** device to retrieve logs

- ▶ You will need to disconnect the on-board **10Base-T1S** adapter of the **Infotainment unit** to use the **Facedancer21**
- ▶ Remove the **10-pin** ribbon cable, unplug the **jumper** then put the **ribbon cable back**
- ▶ The **Facedancer** is listed as **FTDI FT232R UART adapter** on your virtual machine
- ▶ Complete challenges **Infotainment - USB**



Tooling IVI and bypassing secure boot



- ▶ Having **read/write access to the eMMC**, we can modify its content to **add/patch binaries** or **modify configuration files**
- ▶ Correctly secured IVI will implement a **secure boot** to ensure the authenticity of the OS
- ▶ But ... it has to be done in a proper way
- ▶ Not all partition could be **signed**, as at least one partition must be alterable to **store persistent/user data**
- ▶ What if this partition is **executable** or store **binaries/configuration** used by the system ?
- ▶ Analysing **logs** may give some hints on how the **secure boot** is implemented

Standard secure boot process

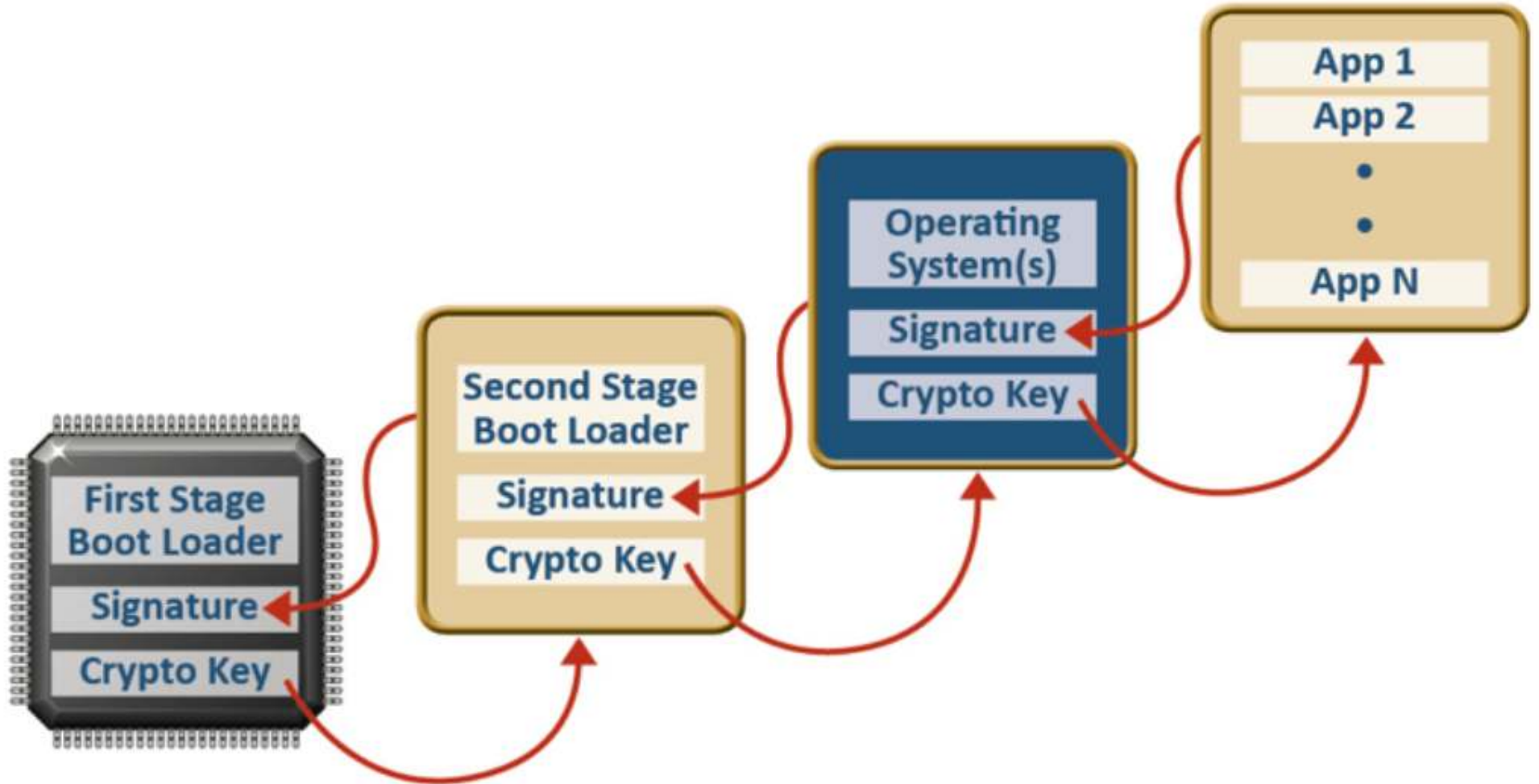


Illustration: [\[link\]](#)

Bypassing secure boot



- ▶ If correctly done, a **secure boot** is a robust **protection against firmware alteration**
- ▶ Anyway, there are techniques to explore to overcome this
- ▶ Look for **1st HW/SW generation**, sometimes secure boot is not implemented yet
- ▶ **Fault injection** is a useful technique to attack secure boot, but it requires time
- ▶ Depending on the main chip used in the **SoC**, known **BootROM vulnerability** could be exploited

⚗ Lab 6 - Bypassing vulnerable secure boot



Lab 6 - Bypassing vulnerable secure boot

In this lab you'll exploit a **vulnerable design** we found in an **Infotainment unit** to bypass the **secure boot**

- ▶ Complete challenges **Infotainment - Bypassing secure boot**



Instrumenting the firmware

- ▶ If the **Infotainment unit** filesystem is writable, it could be possible to **instrument** it, adding binaries or altering configuration files
- ▶ Otherwise, if a **shell** access is present, finding a **writable folder** with **executable right** could be an alternative
- ▶ You may need to compile useful binaries for the target architecture or look for repository like [The Real Saumil's GitHub: static-arm-bins](#)
- ▶ For the training, we will use [Frida](#) to hook to a process and change its execution flow

- ▶ **Frida** is a renowned **dynamic instrumentation toolkit**
- ▶ It allows to inspect and modify applications **while they run**
- ▶ It's a **multi-platform** application
- ▶ It injects **QuickJS** into the target process

FRIDA

Frida - installation



- ▶ On the target device install the the of **frida-server** for the corresponding **architecture**
- ▶ On you computer, you'll need **frida** and **frida-tools** that match the major and minor version of **frida-server**
- ▶ By default, **frida-server** listens on the local interface only, add `-l 0.0.0.0` to expose port **27049**



Frida - usage

```
$ frida-ps -H 192.168.222.1
```

```
PID  Name
-----
11541  frida-server
11468  frida_target
1305   systemd
```

```
$ frida -H 192.168.222.1 frida_target
```

```

  ____
 / _ |  Frida 17.9.10 - A world-class dynamic instrumentation toolkit
| (| |
> _ |  Commands:
/_/ |_|    help      -> Displays the help system
. . . .    object?   -> Display information about 'object'
. . . .    exit/quit -> Exit
. . . .
. . . .    More info at https://frida.re/docs/home/
. . . .
. . . .    Connected to 192.168.222.1 (id=socket@192.168.222.1)
```

```
[Remote::frida_target ]->
```



Frida - commands example

- ▶ **ptr**: converts a number into a **NativePointer**

```
args[0] = ptr(0x1337) # Set the 1st argument of a hooked function to 0x1337  
var p = ptr(0x1234)  
p.readU8() # Read 1 byte at address 0x1234
```

- ▶ **Process.enumerateModules()**: show loaded modules

```
Process.enumerateModules()[0] # Show information about the main program
```

- ▶ **Interceptor.attach(address, callbacks)**: hook a function and intercept its arguments and return value

```
var my_target_function_address = ptr(0xdeadbeef)  
Interceptor.attach(my_target_function_address, { onEnter(args) {}, onLeave(retval) {} })
```



- ▶ When called, **Frida** displays the **command line interface**
- ▶ **Javascript hooks** can be used to automate complex functions
- ▶ A hook is loaded by using argument **-l**

```
$ frida -H 192.168.222.1 my_process -l my_hook.js
```

🧪 Lab 7 - Binary instrumentation



Lab 7 - Binary instrumentation

This lab is a short introduction to **Frida**, where you'll hook to a running process to retrieve the flag



Due to software constraints, the **Infotainment unit** could only run outdated version of **frida-server**, so the real target will be the **Telematic Control Unit**

You'll mimic flashing **frida-server** on the **Infotainment unit** using the eMMC adapter, then using **Frida** to retrieve the flag

- ▶ Complete challenges **Infotainment - Binary instrumentation** challenges

Extracting Personally Identifiable Information

IVI: a personally identifiable information goldmine



- ▶ IVIs **store a lot of data**, including:
 - ▶ Phonebook address
 - ▶ Navigation history
 - ▶ Paired phone information
 - ▶ Logs of call sent/received
 - ▶ ...



Illustration: [link]

- ▶ Modern **IVI** are connected to **internal/external camera**, pictures may be retrieved
- ▶ If there is some compliance with the **GDPR**, logs may still contain **privacy related data**



IVI: a personally identifiable information goldmine

Database Structure

Browse Data

Edit Pragmas

Execu

Table: sysDB

	bj_ns	obj_id	obj_size	obj_type
	32	Filter	Filter	Filter
1	45232	0	4	0 BLOB
2	45232	9	9	2 BLOB
3	45232	2	4	2 BLOB
4	45232	3	10	3 VW BT 2279
5	45232	13	4	0 BLOB

Edit Database Cell

Mode: Binary

00f0 00 00 10 00 03 01 00 75 00 01 00 00 01 06 00 00

.....u.....

0100 00 d4 fb 8e 9a 7d fe 11 00 00 00 47 65 6f 72 67

.....}.....Georg

0110 65 e2 80 99 73 20 69 50 68 6f 6e 65 0a 00 00 00

e...s iPhone...

0120 41 70 70 6c 65 20 49 6e 63 2e 0a 00 00 00 69 50

Apple Inc.....iP

0130 68 6f 6e 65 31 32 2c 31 1c 00 00 00 56 65 72 73

hone12,1....Vers

0140 69 6f 6e 20 31 36 2e 33 2e 31 20 28 42 75 69 6c

ion 16.3.1 (Buil

0150 64 20 32 30 44 36 37 29 10 00 00 00 a9 4a bd 72

d 20D67).....J.r

0160 b9 7f f1 5e 60 4f 85 5a b7 66 05 d4 06 01 3f 15

...^0.Z.f....?.

0170 10 00 04 00 00 00 07 00 00 00 01 00 00 00 02 01

.....

0180 02 00 00 00 06 01 08 00 00 00 03 01 10 00 00 00

.....

1 BEGIN:PROFILE

2 X-TSD-SLOT:4

3 X-TSD-MACADDRESS:90b123d2c60a

4 X-TSD-NAME:Galaxy A51

5 X-TSD-LRU:442

6 X-TSD-SORTORDER-LASTNAME-COMMA-FIRSTNAME:

7 X-TSD-PICTURE-VISIBLE:

8 X-TSD-PRIVATEPROFILE:

9 END:PROFILE

10 BEGIN:VCARD

11 VERSION:3.0

12 FN:PAU

13 X-TSD-ORIGIN:ME

14 N:;JOHN;;;;

15 TEL;TYPE=CELL:+32453271723

16 END:VCARD

17 BEGIN:VCARD

18 VERSION:3.0

19 FN:John, Doe

20 X-TSD-ORIGIN:ME

21 N:;John;;;;

22 TEL;TYPE=CELL:5749

23 END:VCARD

```
14:56:00.878 [Warn] [iMX6.Organizer.CallStack] CallStack::addCall. Reached the maximal nr of calls. Call details, normalNumber: 4 origin: 0 ent
14:56:00.408 [Info] [J5e.Radio.AUDIO] IMX Input DMA: iMaxAnn1 = 0x0000
14:56:00.886 [Warn] [iMX6.Organizer.CallStack] CallStack::addCall. Reached the maximal nr of calls. Call details, normalNumber: 4586359 origin:
14:56:00.409 [Info] [J5e.Radio.AUDIO] IMX Input DMA: iMaxAnn2 = 0x0000
14:56:00.895 [Info] [iMX6.Navi.tsd.nav.core.navlocresolver.GeoPositionResolver] resolved country: France
14:56:00.411 [Info] [J5e.Radio.AUDIO] IMX Input DMA: iMaxAnn3 = 0x0000
14:56:00.909 [Info] [iMX6.Navi.tsd.nav.core.navlocresolver.GeoPositionResolver] resolved city: Marseille
14:56:00.412 [Info] [J5e.Radio.FmSingle-ALL] ESC[0mQuali - Antenne 1|F: 103200kHz||PI: -2|QS: 0|QA: 0|QSmin:30|RDS:0|Seek:0ESC[0m
14:56:00.909 [Info] [iMX6.Navi.tsd.nav.core.navlocresolver.GeoPositionResolver] cityRefinement: Bouches-du-Rhône, Provence-Alpes-Côte d'Azur
```

Collecting personal data

- ▶ Some IVI dump logs on **USB mass storage** device or **SD card**
- ▶ A specific folder or file may be required, or an activation through the engineer menu
- ▶ Trying **USB to Ethernet adapter** on USB port may provide access to **exposed services**
- ▶ It is common to have a USB **Vendor ID** or **Product ID** filtered for **USB to Ethernet adapter**, but using an **ASIX AX88772B** device it can be altered
- ▶ Scanning UDS **ReadDataByIdentifier** may also provide sensitive data





🧪 Lab 8 - Export personal data



Lab 8 - Export personal data

Exploiting **Engineer Mode** and **USB emulation**, you'll practice on recovering **Personally Identifiable data** stored on the IVI

- ▶ Complete challenges **Infotainment - Personally Identifiable Data**

Thank you

Contact information

Email:

contact@quarkslab.com

Phone:

+33 1 58 30 81 51

Website:

<https://www.quarkslab.com>