ESPwn32: Hacking with ESP32 System-on-Chips

Romain Cayre, Damien Cauquil

EURECOM Sophia Antipolis

Quarkslab

Who are we ?

Romain Cayre, EURECOM

- maintainer of *Mirage*, a popular BLE swiss-army tool
- loves *cross-protocol attacks* (*Wazabee*)

Damien Cauquil, Quarkslab

- maintainer of *Btlejack*, another BLE swiss-army tool
- loves reversing stuff, including *embedded systems*

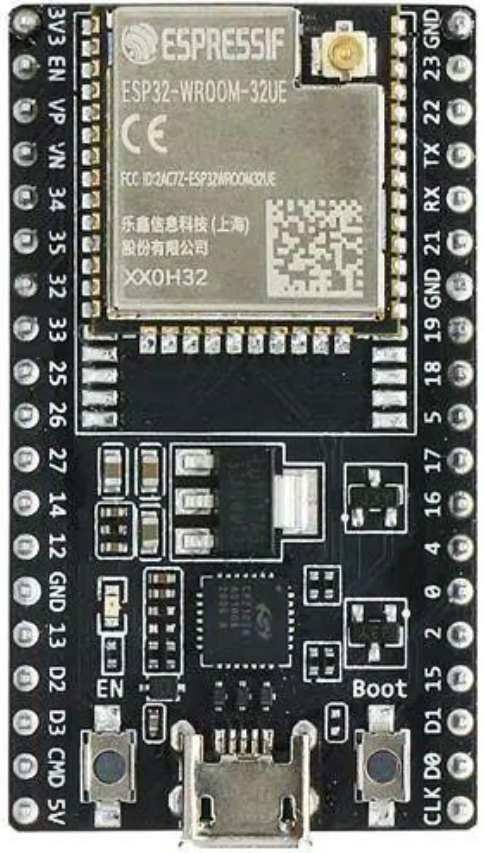
EURECOM Sophia Antipolis

Quarkslab

Introduction

Enter the ESP32 world !

- **Cheap** and **lightweight** SoCs
- Commonly used for **IoT devices**
- Provides **WiFi**, **Bluetooth Low Energy** / **Bluetooth BR/EDR**
- **Tensilica Xtensa** (ESP32, ESP32-S3) and **RISC-V** (ESP-C3)



Lots of questions ...

Is it possible to:

- **sniff** BLE communications ?
- **inject** an arbitrary BLE PDU ?
- **divert the radio PHY** to do *nasty* things ?
- support **other wireless protocols** ?
- turn any ESP32 into a **wireless hacking tool** ?

EURECOM Sophia Antipolis Quarkslab

ESP32 internals

ESP32 Internal ROMs

- 2 **specific ROM** regions
- These regions contain some **code and data**
- **Low-level API functions** to drive the BLE core
- **Problem**: how to hook these functions?

Hooking ROM functions

- ROM functions are called through `r_ip_funcs_p`
- `r_ip_funcs_p` is a **table of function pointers in RAM**

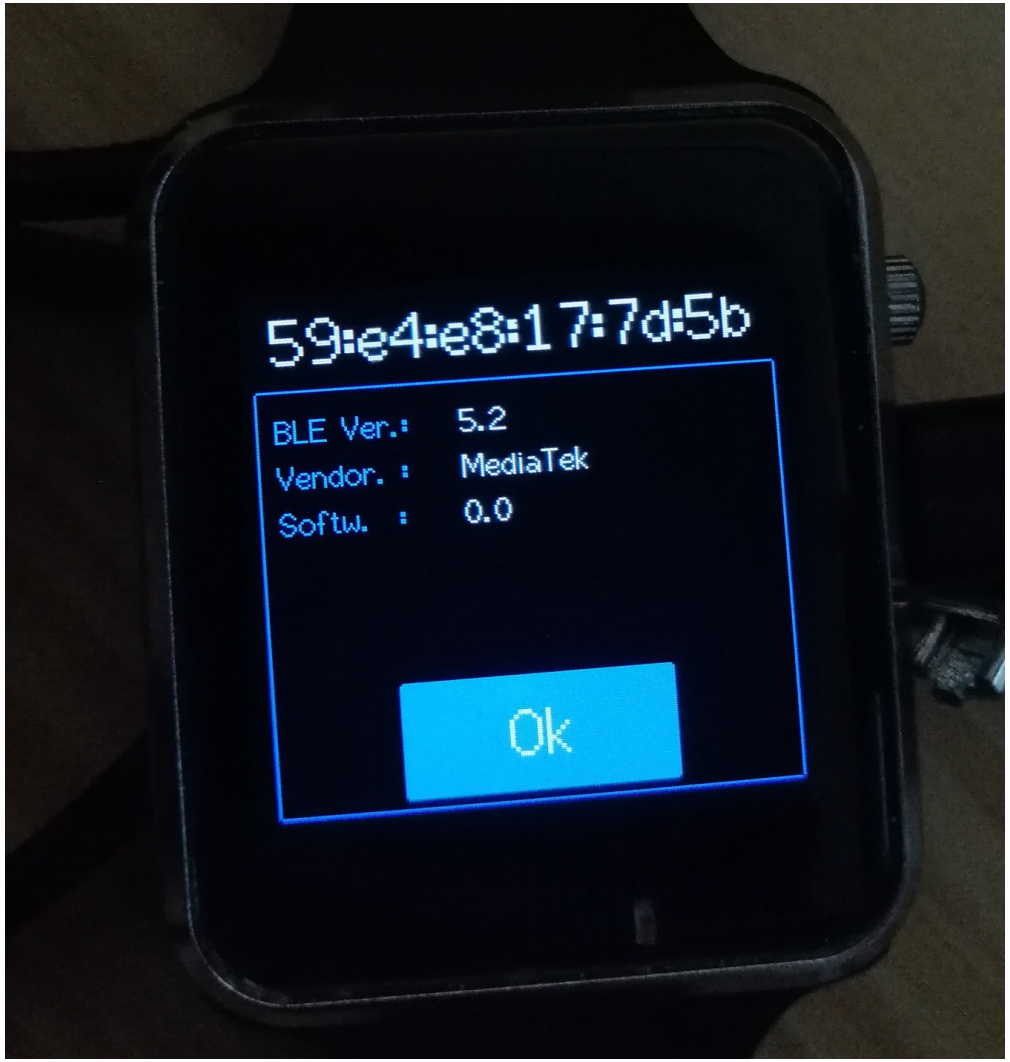
```
400ea86a 41 df e8      l32r      a4,->r_ip_funcs_p
400ea86d 48 04         l32i.n    a4=>r_ip_funcs_p,a4,0x0
400ea86f 42 d4 0a      addmi     a4,a4,0xa00
400ea872 42 24 2f      l32i      a4,a4,0xbc
400ea875 e0 04 00      callx8    a4
```

PDU sniffing & injection

- `r_lld_pdu_rx_handler()` : called whenever a **PDU is received**
- `r_lld_pdu_data_tx_push()` : used to **send a PDU**

LL_VERSION_IND injection

Remote BLE stack fingerprinting !



Hacking the physical layer

Cross-protocol attacks

Can ESP32 radio be diverted to interact with other protocols ?

- BLE uses **Gaussian Frequency Shift Keying** (GFSK) modulation…
- … like dozens of weak proprietary protocols !
  (**ANT**, Riitek, MosArt, Logitech Unifying, Microsoft…)
- **WazaBee:** equivalence between O-QPSK (802.15.4) and 2Mbps GFSK (BLE 2M) → **ESP32-S3 / ESP32-C3 only**

Cross-protocol attacks

We **control** the following **low level radio parameters**:

- CRC verification
- frequency
- datarate
- synchronization word
- whitening / dewhitening
- input and output bitstreams

Arbitrary reception primitive

Hook `r_llm_start_scan_en()` and **modify RF parameters**:

- force a specific frequency and disable channel hopping,
- divert access address as a synchronization word,
- force datarate,
- configure test format,
- disable whitening and CRC.

Reuse `r_lld_pdu_rx_handler()` hook to **extract packets**.

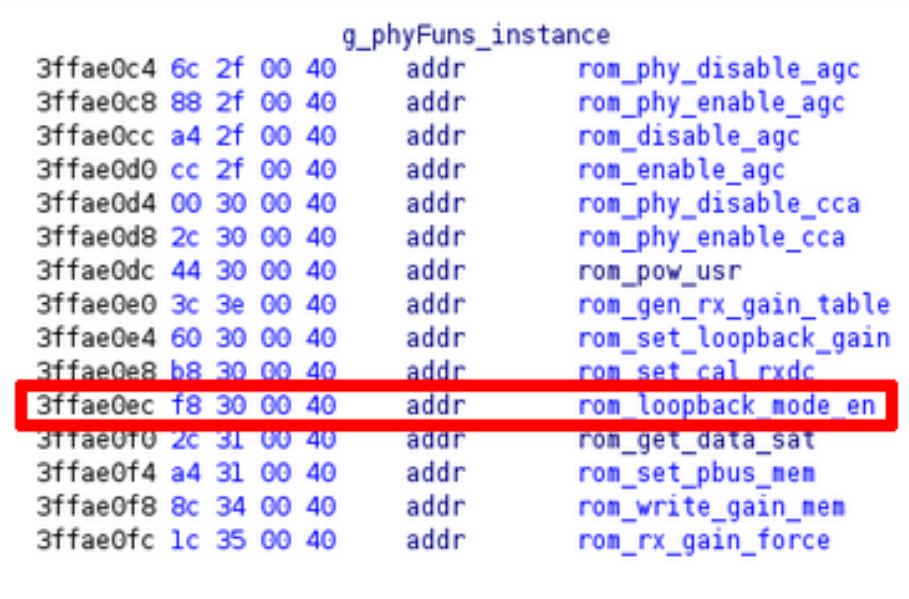
Arbitrary transmission primitive

- **Hook** `r_lld_pdu_tx_push` and **modify RF parameters**,
- **Find the TX buffer** in memory and write a packet (PIP attack),
- Start radio in **TX test mode**.

Demo time !

Transmitting arbitrary signals

Hooking PHY functions

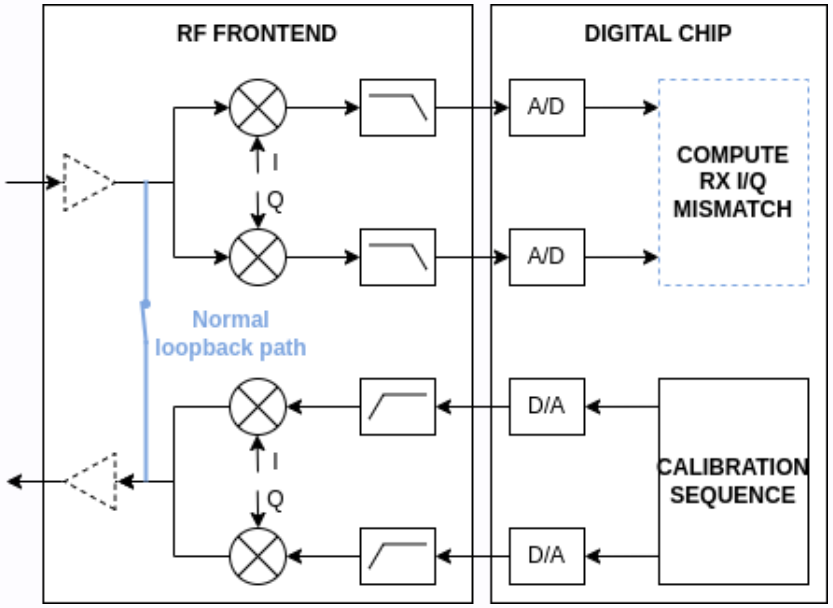


Low level RF functions are stored in a specific function pointers array: `g_phyFuns` .

➜ We can reuse the **same hooking technique**.

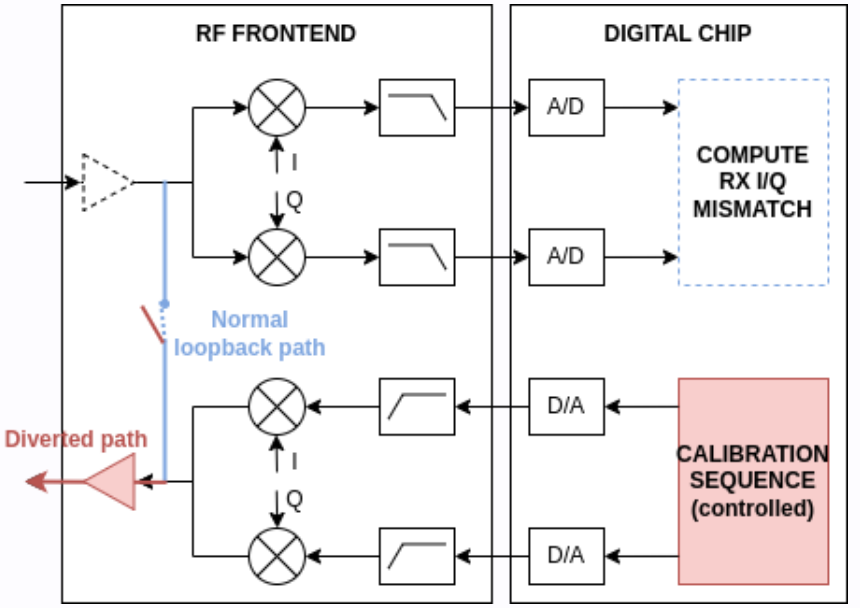
Calibration process

Imperfections corrected using **digital calibration technique**:



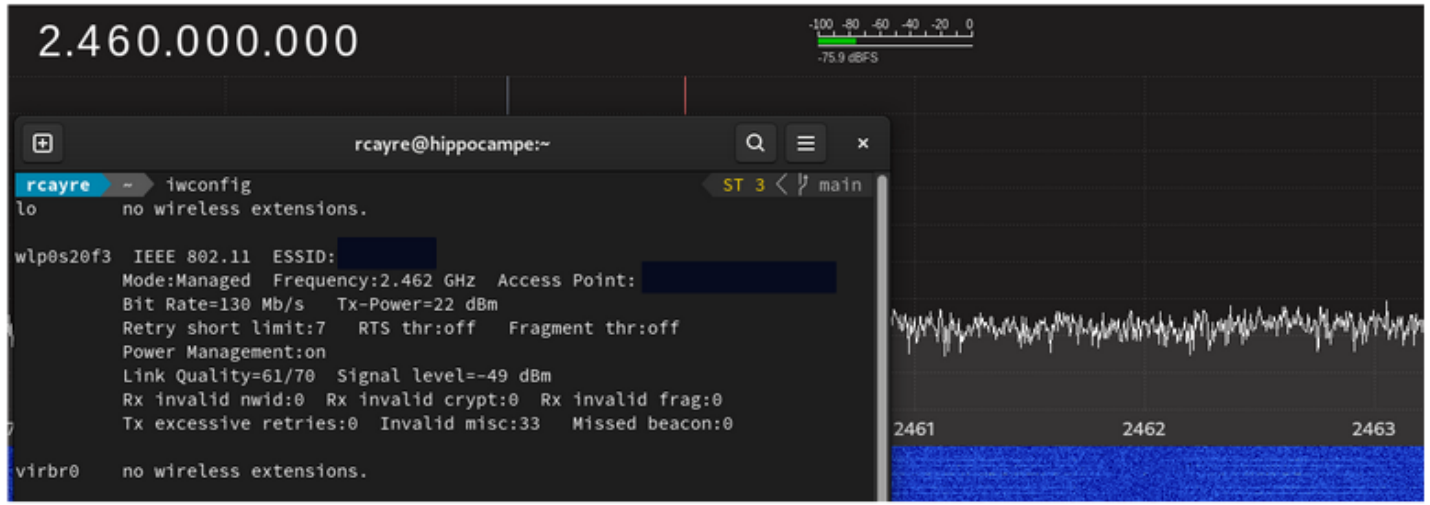
Loopback between TX and RX path to estimate and compensate I/Q mismatch.

Diverting calibration process

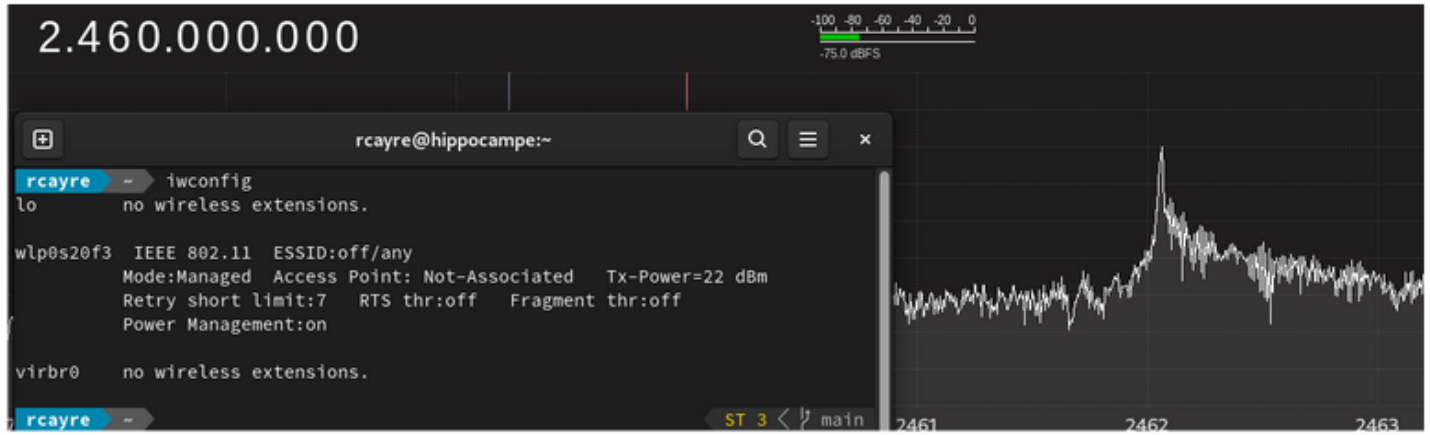


- Disable HW frequency control (`phy_dis_hw_set_freq`).
- Infinite loop when `rom_loopback_mode_en` is called.
- Call low level functions to control frequency and gain.

WiFi Jamming

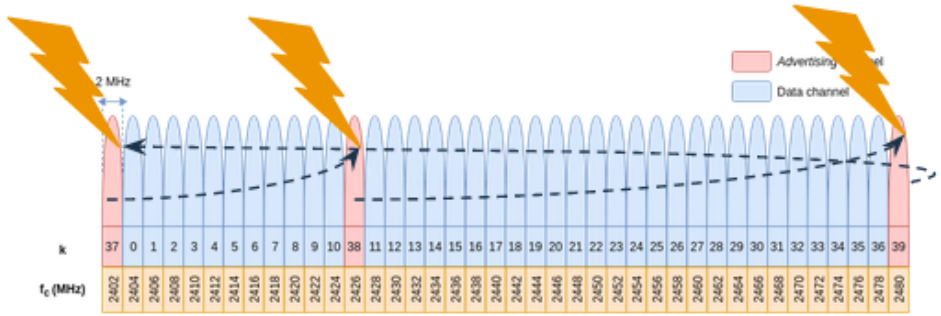


Jamming disabled

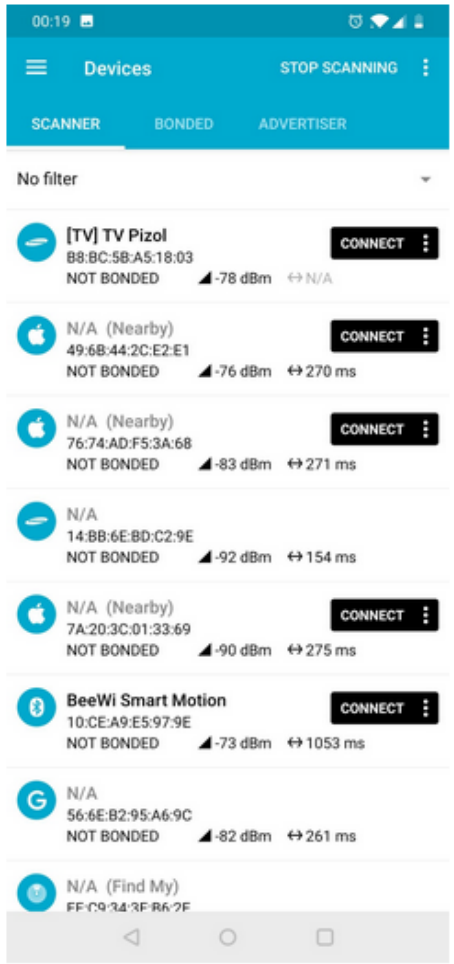


Jamming enabled

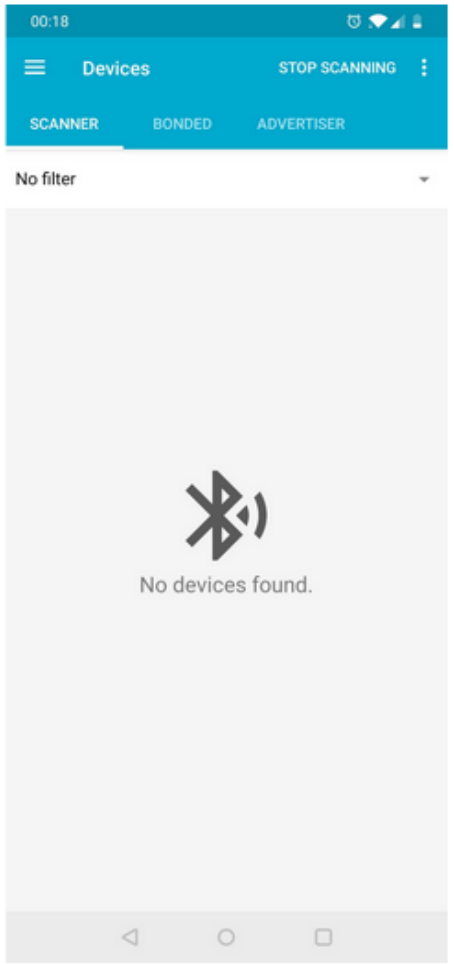
BLE Jamming




```
while (jammer) {
        // Set frequency to 2402 MHz (channel 37)
         set_chan_freq_sw_start(2,0,0);
        // Alter the parameters
        ram_start_tx_tone(1,0,10,0,0,0);

        // Set frequency to 2426 MHz (channel 38)
        set_chan_freq_sw_start(26,0,0);
        ram_start_tx_tone(1,0,10,0,0,0)

        // Set frequency to 2480 MHz (channel 39)
        set_chan_freq_sw_start(80,0,0);
        ram_start_tx_tone(1,0,10,0,0,0);

}
```



Jamming disabled



Jamming enabled

Takeaways

- **ESP32 BLE stack can be repurposed** to perform:
  - on the fly **BLE PDU monitoring, modification & injection**,
  - **cross-protocol** eavesdropping & injection,
  - **jam multiple channels** and establish a **covert channel**.

- **Risks related to the coexistence of wireless protocols**:
  - Attacker can **leverage similarities** in the physical layer,
  - **no security** or **security by obscurity**
  - large deployment of BLE devices ➔ **new attack surface**

Q/A time

EURECOM Sophia Antipolis

Quarkslab

Thank you !