An Exploration into Bluetooth RF Fingerprinting and Protocol Fuzzing

Mr. Artis Rušiņš (LV), Mr. Eduards Blumbergs (LV)

Cyberchess 2023

$ About us

Artis Rušiņš
Scientific assistant at EDI (Elektronikas un Datorzinātņu institūts)
PhD student at University of Latvia

Eduards Blumbergs
Computing Systems Analyst at IMCS UL AI Lab. (LU Matemātikas un informātikas institūts)
PhD student at University of Latvia

$ Agenda

{01}
Context and motivation

{02}
Background

{03}
Gathering dataset & RF fingerprints

{04}
Moving up the stack

{05}
Fuzzing

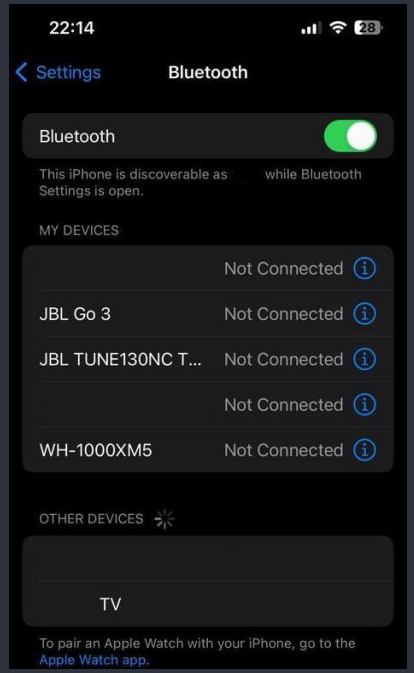
{06}
Windows crash

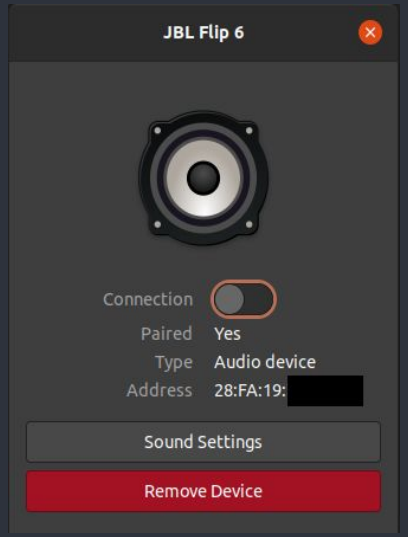
$ Context and motivation

1. 4.9 billion units shipped in 2022 and forecasted to reach 7.6 billion units in 2027 [1]
2. Privacy and surveillance risks of sensors carried on person (location, sensor data, connection metadata, etc.)
3. A complex protocol saddled with compatibility baggage (Interesting protocol level firmware vulnerabilities)
4. Hard or impossible to distribute updates
5. Existing QA and certification does not test for malicious data
6. Firmware and driver may be able to circumvent device OS security

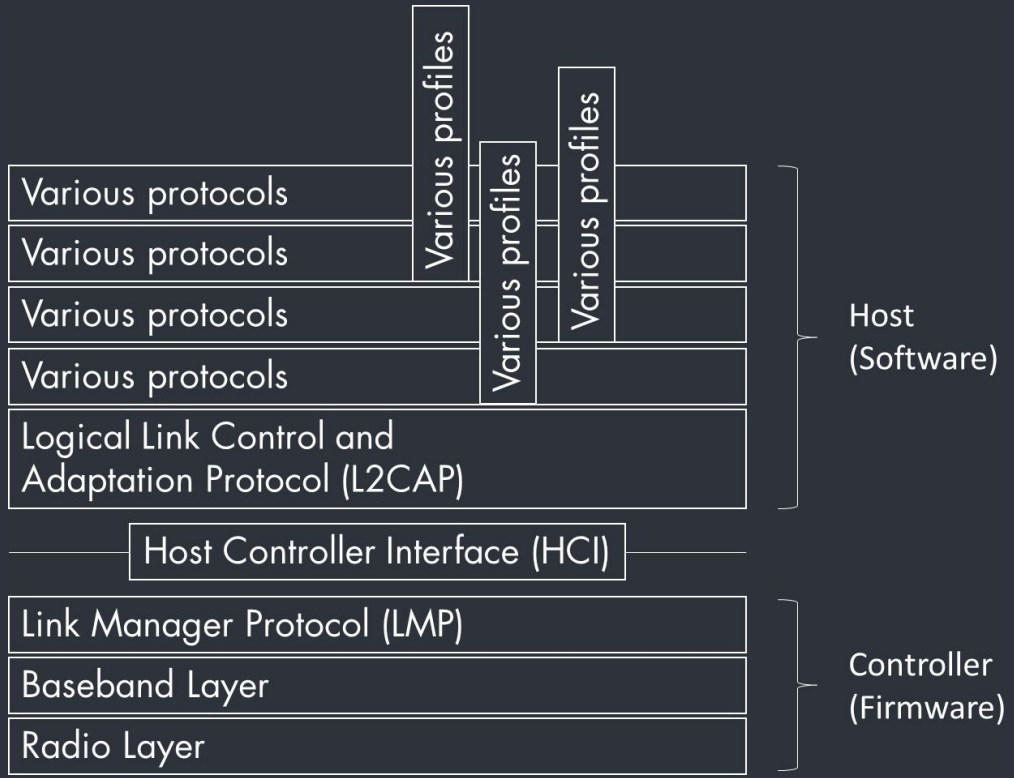
$ Context and motivation

1. The user-friendly name (changeable by the user)
2. MAC (BD_ADDR) (mostly randomized, changes every few minutes)
3. Every radio chipset has quantifiable unique features (slight changes off ideal standard defined way to communicate). Can not be changed



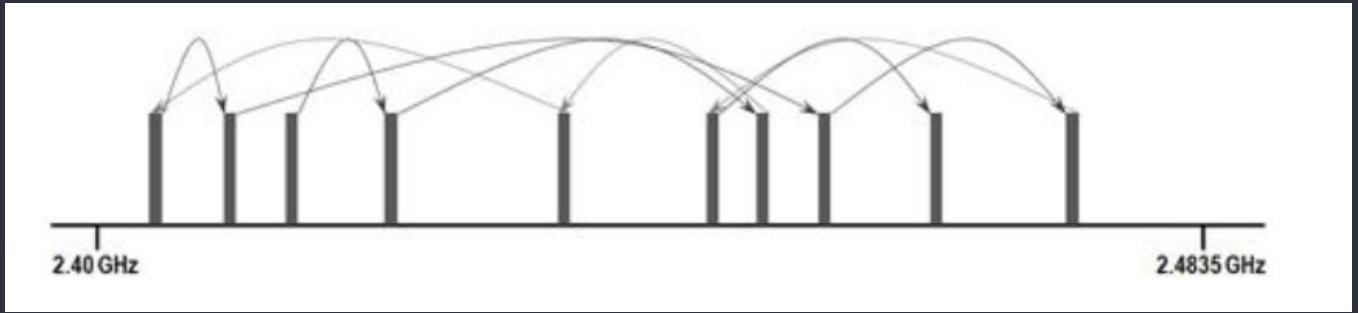


$ Bluetooth Stack [Simplified]



$ Background

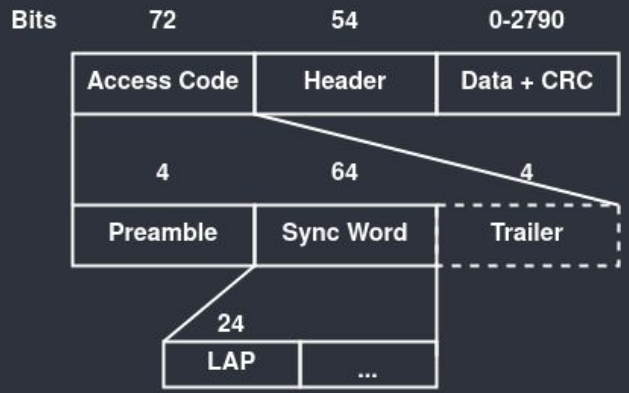
1. ISM band 2.402 GHz – 2.480 GHz
   (same as WiFi, ZigBee, Microwave owens …)
2. 79 channels, each 1 MHz of bandwidth for Bluetooth Classic
3. 40 channels, each 2 MHz of bandwidth for Bluetooth Low Energy
4. Frequency hopping spread spectrum (FHSS)



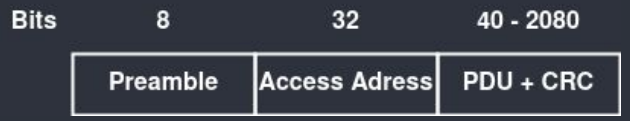
[2]

$ Background

Bluetooth classic packet



BLE packet



LAP 0x9E8B33 or AA 0x8E89BED6 used for discovering devices in range

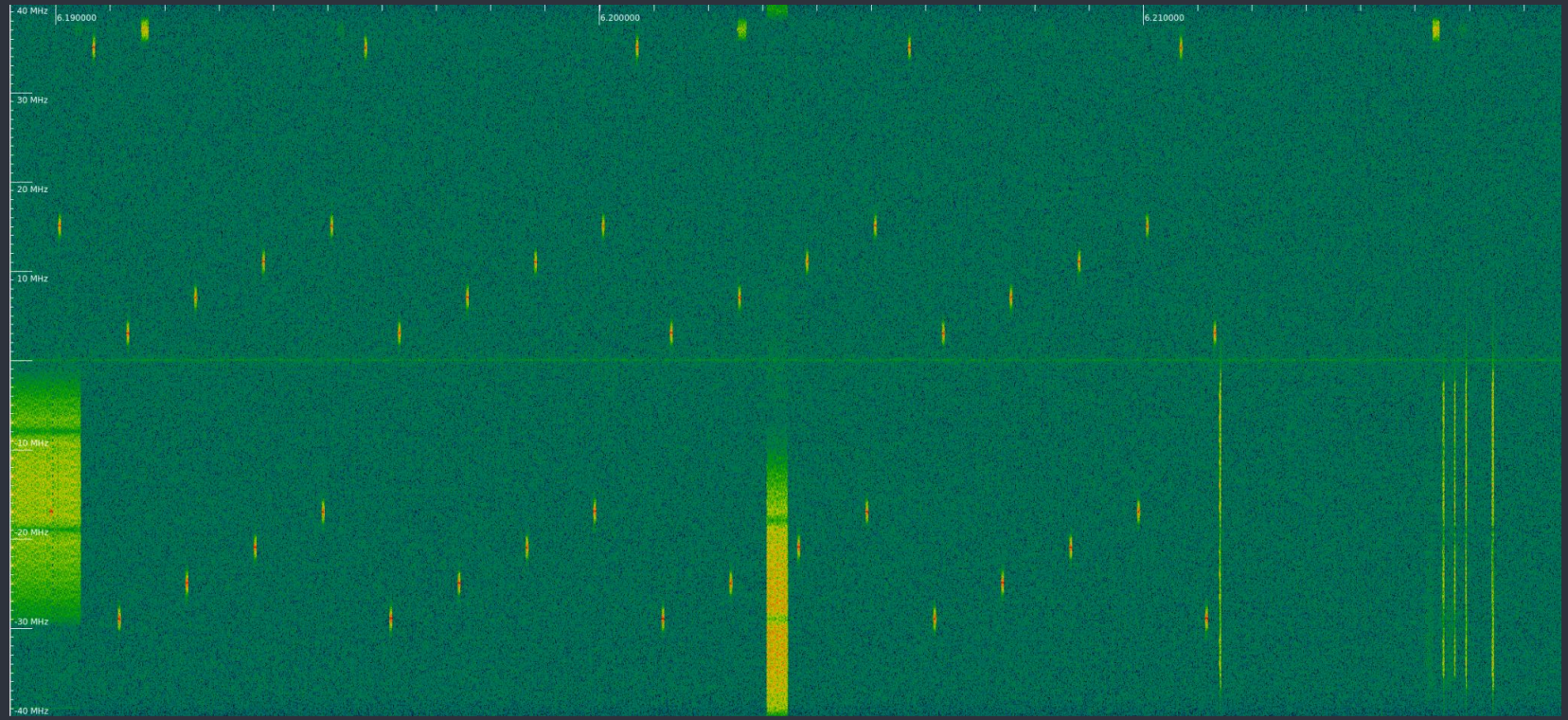
$ Background

1. We can NOT reliably identify device by its user friendly name
2. We can NOT reliably identify device by its BD_ADDR
3. We can try to identify device by its RF fingerprint?

$ Background

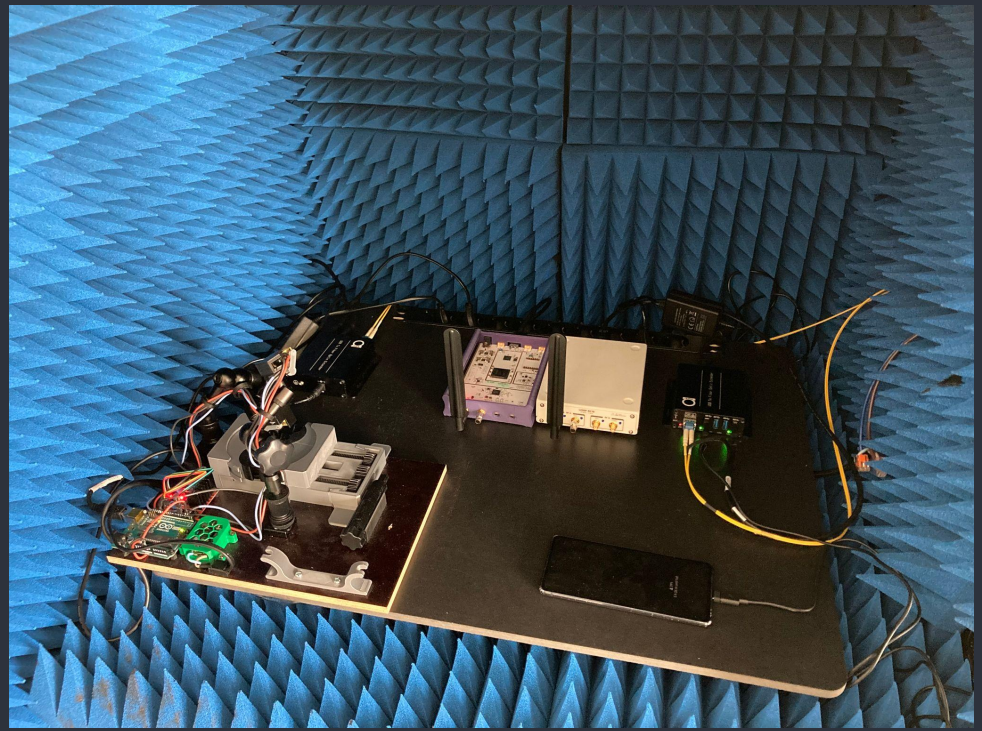
1. We can NOT use Bluetooth dongles / Bluetooth hcitool
2. We CAN use Software Defined Radio (SDR)

$ Background



[3]

$ Gathering dataset



1. 2x USRP B210
2. Android smartphone (master)
3. Device under test (DUT)
4. Robot-hand
5. Everything is optically decoupled

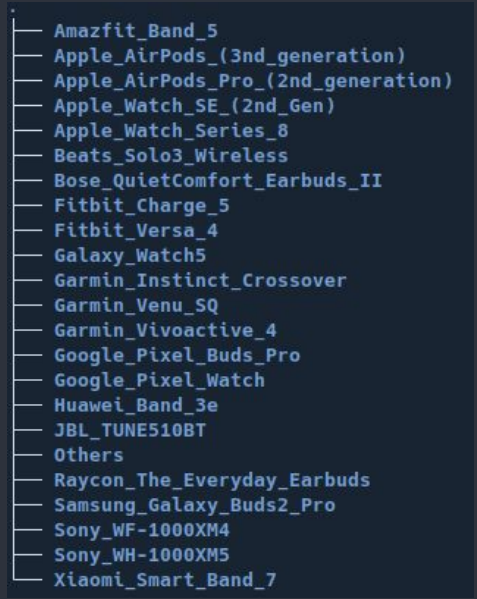
$ Gathering dataset

1. Turn on DUT in pairing mode
2. Close Faraday cage
3. Start SDR recording
4. Run Android Debug Bridge (ADB)[5] script
    a. Turns on Android Bluetooth scanning
    b. Connects to DUT
    c. Exchange data (play a song, interact with specific app, etc.)
    d. Turn off Android Bluetooth
5. End SDR recording
6. Write metadata

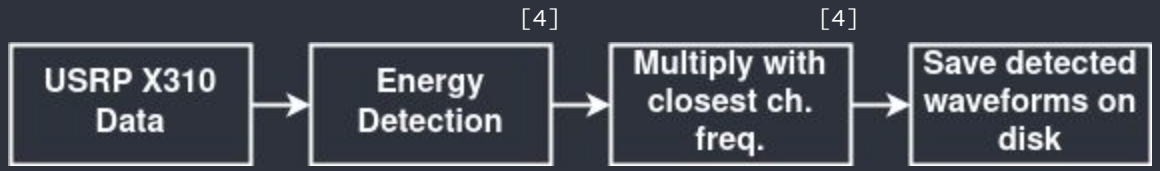
1 experiment = ~23 GB of data

$ Gathering dataset

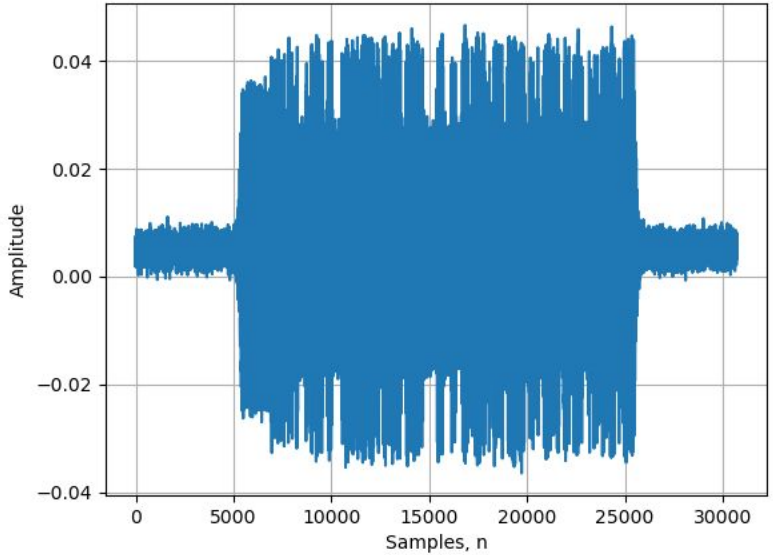
1. Bluetooth device radio recording dataset in isolated environment
2. Metadata: radio parameters, event description, timestamps of connection phases

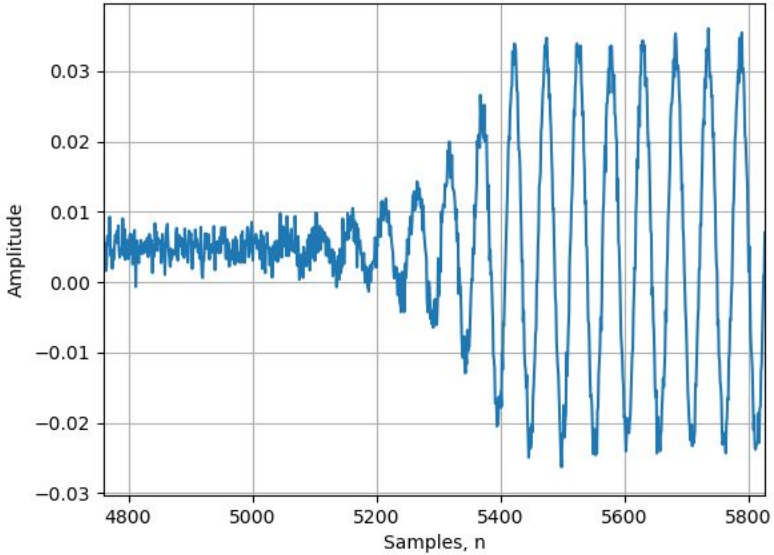


$ Gathering dataset

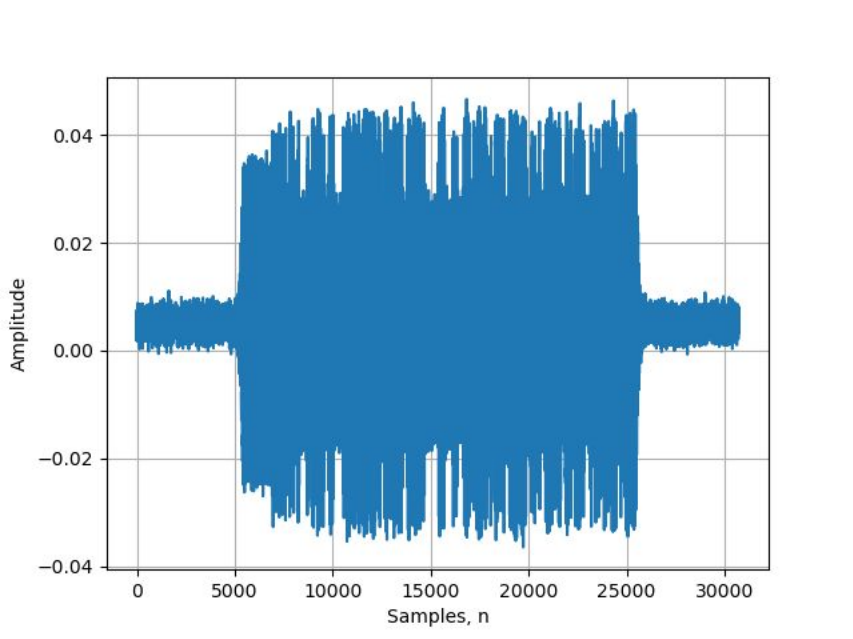


$ Gathering dataset

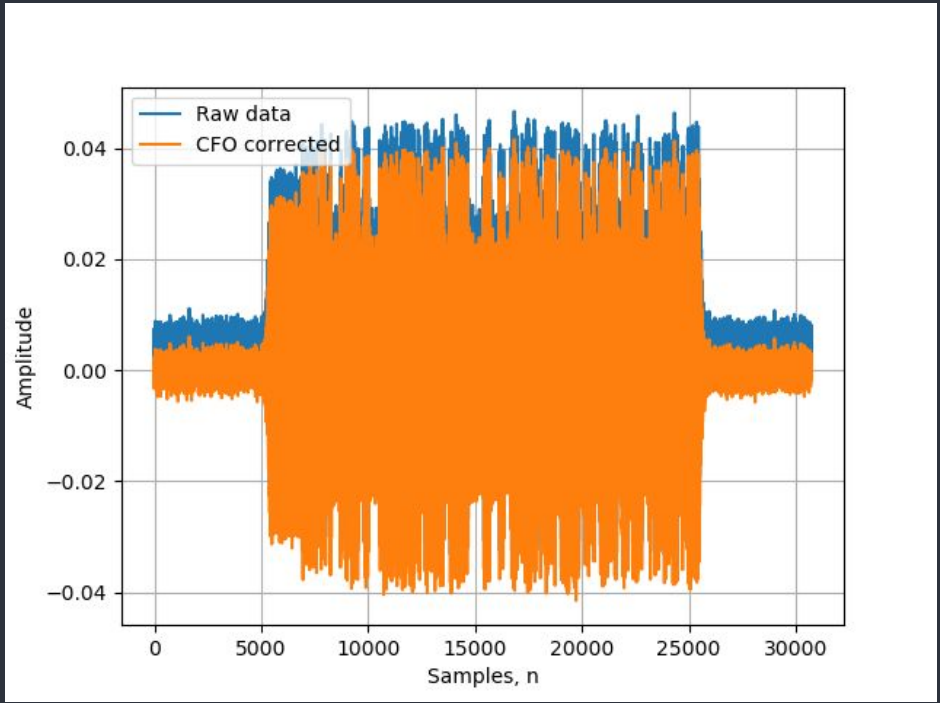




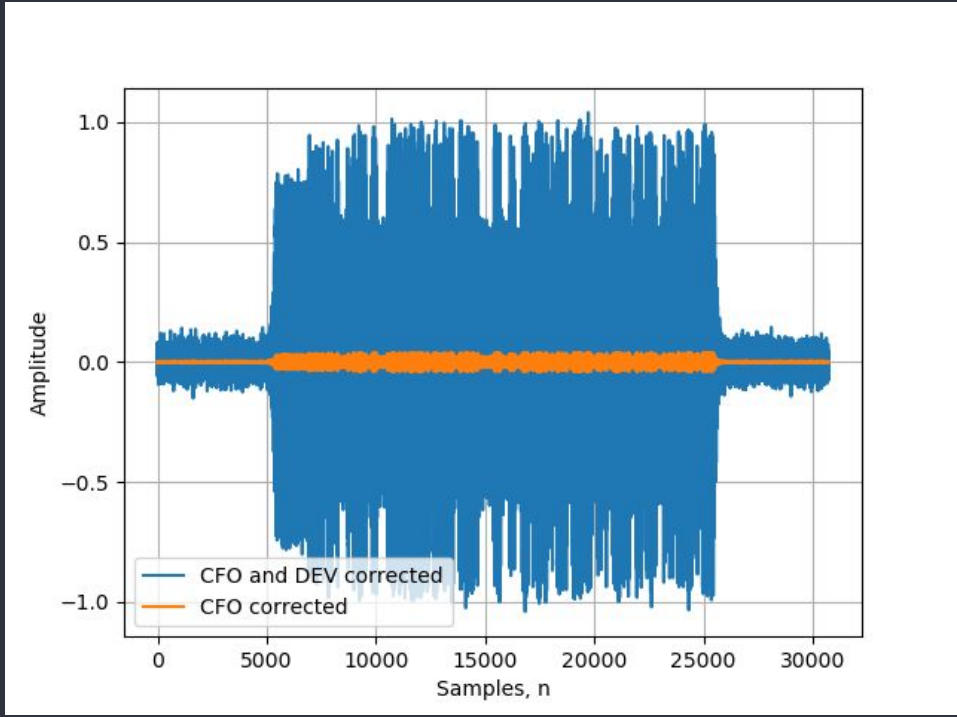
$ RF Fingerprints



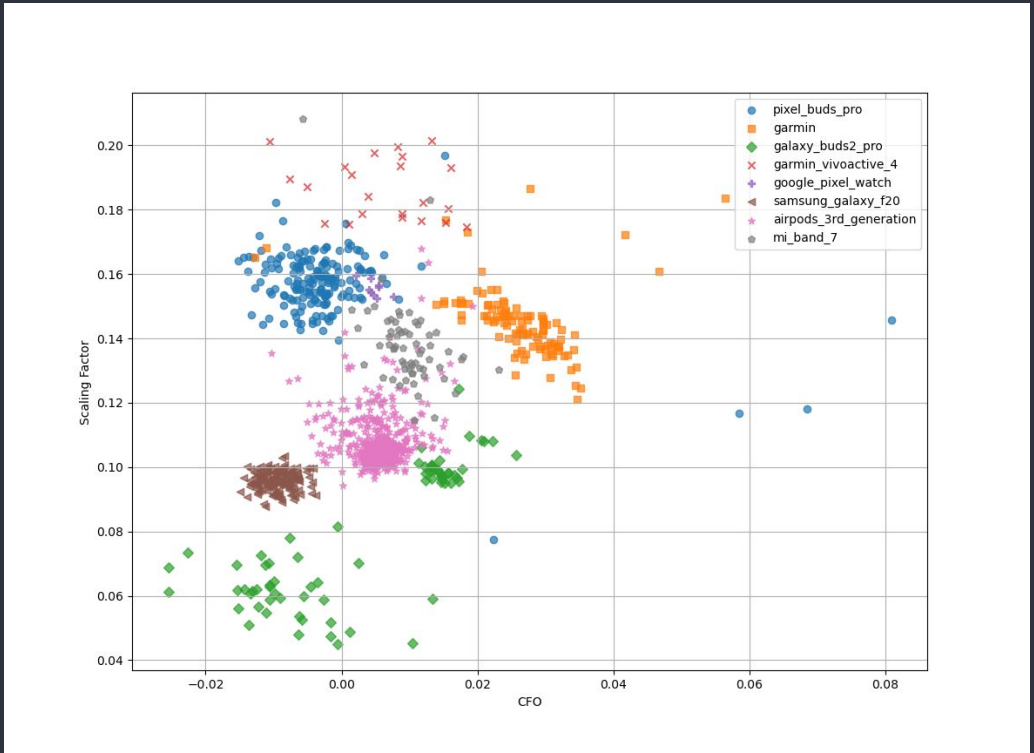
$ RF Fingerprints



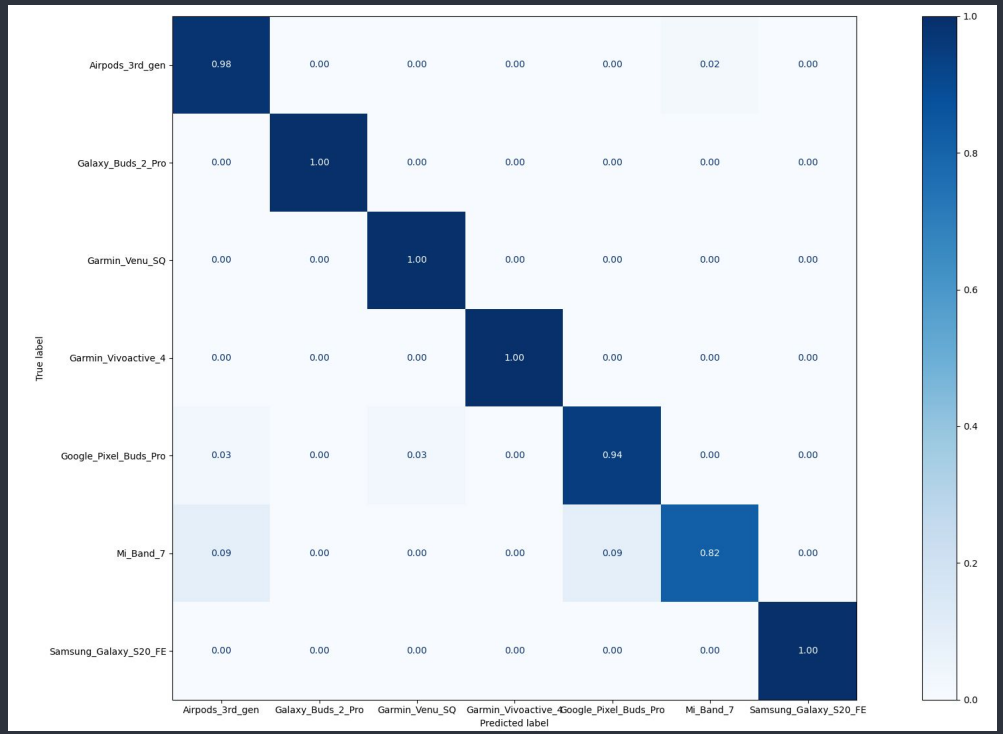
$ RF Fingerprints



$ RF Fingerprints



$ RF Fingerprints



KNN accuracy of 0.97 in ideal conditions

$ RF Fingerprints

Demo:

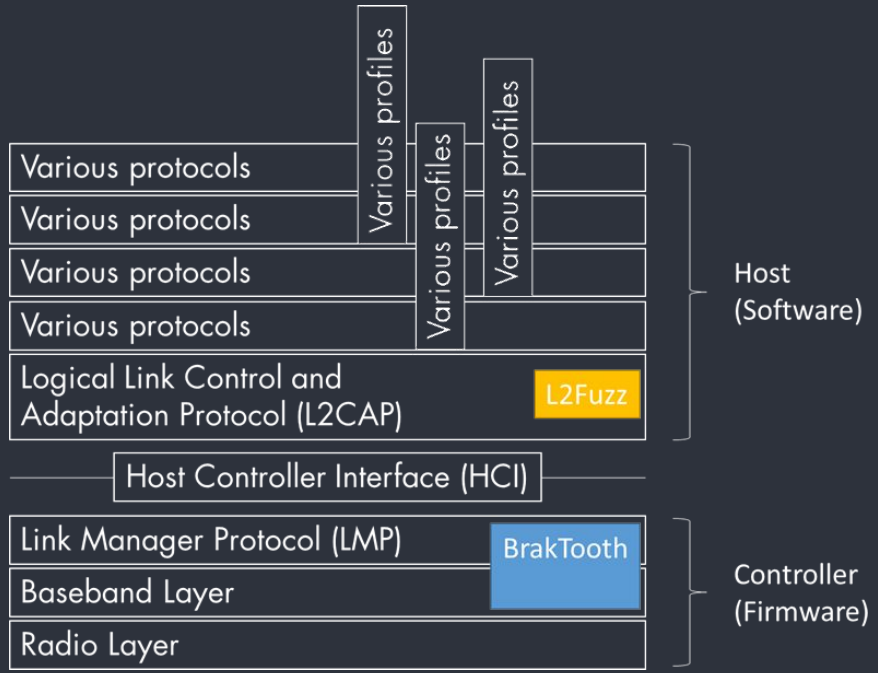
1. HackRF with modified Bluetooth sniffer[6]
2. Individual waveforms
3. RF Fingerprints of captured data

$ RF Fingerprints

Potential use cases:

1. Tracking
    a. Obvious privacy issues
2. Secure facilities
    a. Can we detect untrustworthy devices?
3. Cyber-attack by identification of a specific Bluetooth device model / hardware version / Bluetooth version to exploit device-specific vulnerabilities

$ Moving up the stack



The Bluetooth stack layers, each with its own set of responsibilities

- multiplexing of logical transports
- logical / physical transport management
- connection negotiation and management
- operation of physical channels

$ Bluetooth Implementations

1. Many embedded chip producers and closed source implementations.

2. Out of the 30 implementations tested, 25 had vulnerabilities detectable by fuzzing[7].

$ Bluetooth Fuzzing

- Fuzz testing is a technique that uses random or semi-random test data to probe well-behaving protocol for new vulnerabilities or unexpected behavior.
- Bluetooth has a stateful data protocol with multi-stage connection interactions.
- Limitation: Purely random fuzzing risks state disruption and lessens meaningful vulnerability detection.
- Usual fuzzing outcome is device crash, disruption or deadlock.

$ Fuzzing for Vulnerabilities

1. Memory Corruption (overflows, use-after-free).
2. Protocol Implementation Flaws (weak encryption, non-compliant pairing).
3. Denial of Service (exhaust of resources, crashes).
4. Information Disclosure (disclosed keys or personal data).
5. Undocumented Features (unknown added functionality of unclear quality, data gathering).

$ Dynamic Fuzzing Approaches

1. Instrumentation and Debugging - additional monitoring of the program under test (gather data at the time of the execution).

2. Emulation - used for vulnerability detection in large closed codebases (closed firmware analysis).

$ Notable Fuzzing Tools

L2fuzz (2022) is an easily manageable Python fuzzer for the software (driver) parts. Aiming at the L2CAP layer and covering 19 states of the protocol it can fuzz Protocol Service Multiplexer (PSM) Port 1 (SDP) and can be used with a standard BT dongle and Linux OS.

BrakTooth (2021) is a firmware fuzzer, mostly for the BT Classic audio and automotive hands-free systems, aiming at the Link Manager and Baseband. Most effective exploits are the Invalid-Timing-Accuracy (overload by sending malformed timing data and repeatedly reconnecting), and the Invalid-Setup-Complete (disrupting audio by sending incorrect setup information)[8]. It requires ESP-WROVER-KIT devkit with a modified firmware.

$ Video / Demo

For the purpose of simplify the testing process we have made a few python scripts that:

- Reads **blue_hydra** database and lists BD_ADDResses, names.
- Launch any of 24 known **BrakTooth** fuzzing exploits (PoCs).
- Launch **BrakTooth** fuzzer GUI.
- Launch modified **L2fuzz** fuzzer to fuzz L2CAP PSM Port 1.
- Launch modified **L2fuzz** fuzzer to list profiles the SDP reports.

The device that will be tested with L2fuzz is an unmodified TP-Link Bluetooth 5.0 Nano USB adapter UB500 connected to the Windows 11 laptop. Due to the randomness of the data, a crash can not be guaranteed, but we will try! ;)

$ Windows Crashes [multiple]

Preconditions: BD_ADDR, device NOT paired, latest TP-Link (Realtek) driver.

Outcome: BSOD (W10, W11) or complete freeze (W8.1).

Possible error codes observed (depending on the fuzz sent):

- **PAGE_FAULT_IN_NONPAGED_AREA** (invalid memory access in RtkBtfilter.sys that is a Realtek Bluetooth driver used in TP-Link Bluetooth 5.0 Nano USB Adapter UB500). E.g. Operation triggering the crash: a MOV instruction (`mov [rax+r9-8], rdx`) located at the relative offset `0x76ff0`.
- **SYSTEM_THREAD_EXCEPTION_NOT_HANDLED** (caused by BTHAvrcp.sys related to Bluetooth Audio/Video Remote Control Profile that is required to play Bluetooth audio on Windows).
- **WDF_VIOLATION** (caused by USBXHCI.SYS when the Bluetooth device is connected via USB3 causing the host controller driver to violate Windows Driver Framework rules).
- **DRIVER_IRQL_NOT_LESS_OR_EQUAL** (caused by Wdf01000.sys, a software component that helps run other drivers, tried to access memory it wasn't supposed to, breaking the system's rules.)

$ Prevention

- Acceptable Bluetooth device list (types, tested models).
- Organizational inventory policy - periodic scanning intervals.
- Policy for Bluetooth usage in a hostile environment.
- Pairing only in a secure location (not the IT store).
- If possible, disable unneeded profiles and services.
- Turn BT off when not using (better than non-discoverable mode).
- Maintain software versions.

$ References

1. F. Laricchia. Global Bluetooth device shipments worldwide from 2015 to 2027. https://www.statista.com/statistics/1220933/global-bluetooth-device-shipment-forecast/ Accessed: 12.06.2023.
2. IoTSSC Bluetooth. https://www.inf.ed.ac.uk/teaching/courses/iotssc/slides/Lecture 4-Bluetooth.pdf Accessed 02.10.2023
3. Miek. Inspectrum. https://github.com/miek/inspectrum Accessed: 23.09.2023.
4. Sandia National Laboratories. gr-fhss utils. https://github.com/sandialabs/gr-fhss utils Accessed: 26.04.2023.
5. Android Debug Bridge (adb). https://developer.android.com/tools/adb Accessed: 23.09.2023
6. Mike Ryan. ice9-bluetooth-sniffer.https://github.com/mikeryan/ice9-bluetooth-sniffer Accessed: 26.04.2023.
7. A. Takanen, J. DeMott, C. Miller, and A. Kettunen. Fuzzing for Software Security Testing and Quality Assurance Second Edition. Artech House, 2018, p.27.
8. BrakTooth Hunting in the Car Hacker's Wonderland https://rollingpwn.github.io/Braktooth-IVI-Report/ Accessed: 02.10.2023.

$ Q/A

Artis Rušiņš
artis.rusins@edi.lv

Eduards Blumbergs
eduards.blumbergs@lu.lv

$ Acknowledgement

This research is funded by the Latvian Council of Science, project “Automated wireless security analysis of wearable devices” (WearSec), project No. lzp-2020/1-0395