Mobile Hardware Security

Mobile Security 2022

Florian Draschbacher
florian.draschbacher@iaik.tugraz.at

Practicals

- Start now!
- Deadline 8th of June
- Questions?
  - Ask now
  - Send me an email

Introduction

Apple AirPlay Private Key Exposed, Opening Door to AirPort Express Emulators

Sunday April 10, 2011 11:11 pm PDT by Arnold Kim

Developer James Laird has reverse engineered the Airport Express private key and published an open source AirPort Express emulator called Shareport.

> *This program emulates an Airport Express for the purpose of streaming music from iTunes and compatible iPods. It implements a server for the Apple RAOP protocol.*

Previously, the private key was unknown, which meant that only Apple's Airport Express or official 3rd party solutions could wirelessly stream music from iTunes or equivalent. Many existing solutions such as Rogue Amoeba's Airfoil have long been able to stream music to AirPort Express or other AirPlay devices, but not the other way around. A Hacker News commenter illumin8 spells it out:

> ***Previously you could do this:***
> *iTunes -- stream to --> Apple Airport Express*
> *3rd party software -- stream to --> Apple Airport Express*

Now you can do this:
iTunes -- stream to --> 3rd party software/hardware

Now, it seems unlikely that any hardware manufacturers will use the unauthorized information to create AirPlay-compatible hardware products, especially when it is possible to be an officially licensed AirPlay partner. However, this does open the door to software solutions. iTunes music , for example, could be streamed to other Macs, non-Macs, customized consoles (Xbox 360), or mobile devices with the right software. The developer originally posted the key to the VideoLan developer mailing list in case there was interest in adding that feature to a future version of VLC.

Motivation

- **What?**
  Airplay key extracted from AirPort Express Firmware

- **Consequences**
  Unauthorized implementations of AirPlay receivers now possible

Source: macrumours.com

What’s this presentation about?

- Mobile Security is not just concerned with smartphones and their OS

- Many more devices that
  - Are highly connected (“Internet of Things”)
  - Contain or process sensitive information
  - Are not obviously computers to average consumers

- Mobile = Embedded computers
  - Embedded Linux
  - Microcontrollers

What’s this presentation about?

- Low-level mobile systems
  - Device interfaces and peripherals
  - Data and tamper protections

- Communication protocols
  - How is sensitive data exchanged?
  - How are these connections secured?
  - Ties back to smartphones!

What is sensitive data here?

- User Data
  - Passwords
  - Credentials
  - Activity logs
  - Location, …

- Device Data
  - Firmware (Security through obscurity!)
  - Burnt-in credentials
    - Protocol keys
    - Copyrighted material (games)
    - Algorithms, …

Scenarios

Microcontrollers

- Reduced computing environment
  - Low processing power, memory and storage capacity
  - No MMU = No real process separation
  - Low power consumption
  - Very fast boot
- Bare-bones firmware
  - Highly task-specific program or using some real-time OS
- Highly connected
  - Wifi, Bluetooth, USB, Ethernet
  - Serial, I2C, SPI, CAN
  - Debugger interface!

Embedded Computers (~ IoT devices)

- Bare-Bones OS on lightweight CPU
  - Mediocre processing power, memory, storage
  - MMU ➔ Capable of Process Separation
  - Higher power consumption, longer boot time
- Running fully-featured OS kernel or bare-bones OS
  - Embedded Linux
- Even higher degree of connectedness

Security-sensitive Embedded Applications

- Secure Elements / Enclaves
  - Smartphones, Laptops
- Controllers
  - Memory controllers, Keyboard controllers, …
- Access Control
  - Possession of some token as a factor for authentication
- Systems than involve DRM or some form of lock-down
  - Prevent unauthorized ecosystem access
- Lots of others, new device categories emerge all the time
  - Item Finders, Smart Locks, Drones, Smart Health devices…

Secure Elements / Enclaves

- Google Titan M2 (Google Pixel 6) Source: security.googleblog.com
  - RISC-V Microcontroller
  - Special Vulnerability Assessment
  - Connects to main SoC through SPI
  - Involved in boot process, file encryption, key management, device unlock, …

- Apple T2 Security Chip Source: Davidov et al.: Inside the Apple T2
  - Full-fledged additional ARMv8 SoC in Intel Mac computers
  - Runs bridgeOS kernel derived from iOS, same secure boot chain
  - Additional ARMv7 CPU acts as Secure Enclave Processor (SEP)
  - Connects to main CPU through USB-attached Ethernet port
  - Involved in boot process, file encryption, key management, device unlock,
    - Touch Bar, Speech Recognition, …

Controllers

- Many peripherals contain reprogrammable microcontrollers
  - Even some sensors are reprogrammable!

- Exploit Firmware Updates in USB Peripherals e.g. for keylogging

  Source: Maskiezicz et al.: Mouse Trap: Exploiting Firmware Updates in USB Peripherals

- SD Cards can be arbitrarily reprogrammed!

  Source: Huang et al.: On Hacking MicroSD Cards

- Multiple exploited reprogrammable modules of a system can collude
  - Wifi controller broadcasts keys logged by keyboard controller

    Source: 8051enthusiast.github.io

Access Control

Embedded devices are used for controlling access to (real-world) resources

- Smart Cards, USB Tokens
  - Use the embedded key material for solving some cryptographic challenge
  - E.g. Yubico Yubikey 5 Neo: Special security MC from Infineon Source: hexview.com
- Hardware Crypto Wallets
  - Store private keys for crypto ledgers on hardware device
  - E.g. Ledger Nano S: Secure Element + MCU for display and USB Source: saleemrachid.com
- Car Keys
  - Microcontroller in key fob communicates with car via simple radio protocol
  - Rolling Code System: Fresh key after every unlock, same algorithm in car and fob

DRM and Ecosystem Lockdown

- PS4 Controllers
  - Only allow gamers to use original or licensed controllers
  - Controllers contain MCU that performs handshake with PS4
  - Involves signing challenge with private key stored in controller firmware
  - Cortex-M3 ARM MCU

  Source: fail0verflow.com

- Apple (iOS) Lightning cables contain authentication chip
  - Only allow charging with official or licensed (MFi) cables
  - Not technically an MCU: EPROM

  Sources: nyansatan.github.io, techinsights.com

Low-Level Interfaces

Low-level Interfaces

- Even embedded devices usually do not consist of just the MCU/CPU
- Peripheral devices
  - External Storage
  - Sensors
  - Displays
  - Coprocessors
  - ...
- Also: MCU firmware needs to be debugged during development
- All of these can be used for physical attacks

Low-level Interface Protocols

Most common protocols:

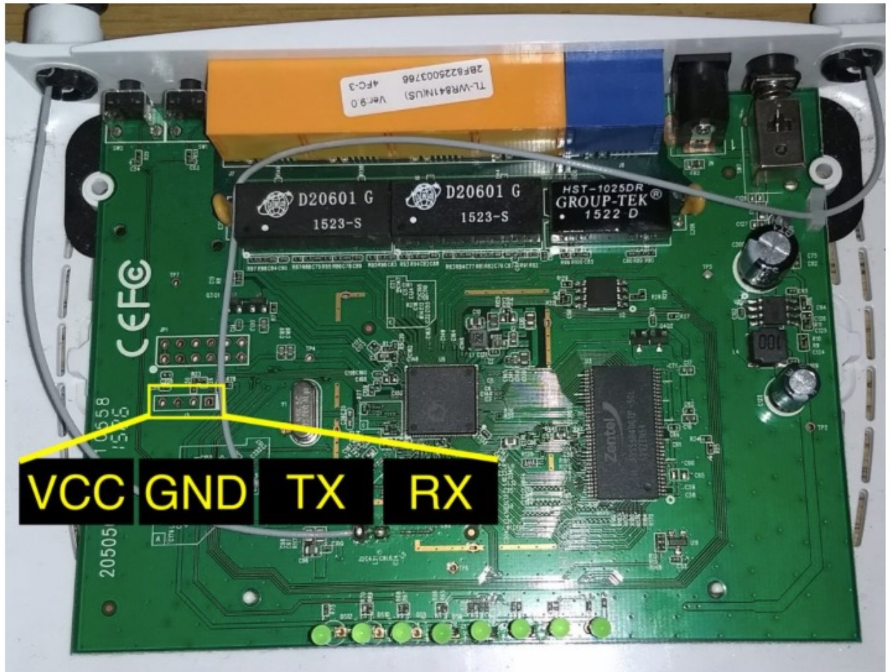
Protocol Name	Wires	Speed	Synchronous	Bus
Serial/UART	2 (RX, TX)	Low	No	No
I2C	2	Low	Yes	Yes
SPI	4+	High	Yes	Yes (1 select line per slave)

- Many more (device specific, vendor specific)
- Security was no concern during design of these protocols!
  - Easy to mount MITM attacks with some soldering

Exploiting Serial / UART

- Intercept all communication by just connecting additional RX line

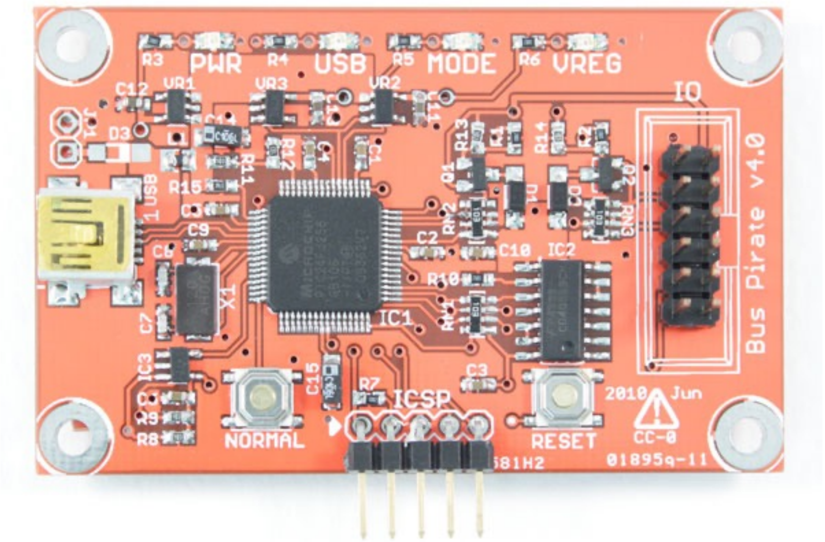
- Many devices have an unpopulated UART header
  - Debug logging
  - Sometimes even exposes root shell / bootloader shell!



Source: konukoii.com

Exploiting I2C

- Simple bus: All messages visible to all bus participants
  - They filter by the address contained in message
- Trivial to intercept
  - Just ignore address
- Dedicated hardware tools
  - Bus Pirate
  - Attify Badge



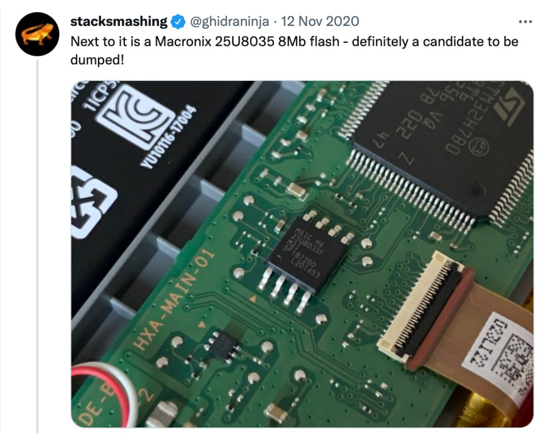
Picture: dangerousprototypes.com / CC BY-SA

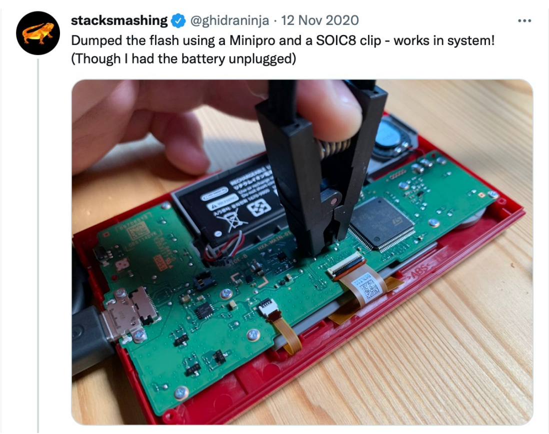
IAIK TU Graz

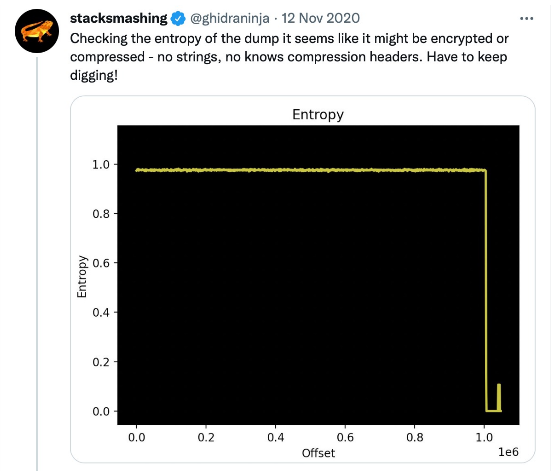
Exploiting SPI

- Intercept SPI communication between master (MCU) and slave
  - Gain insights into exchanged data

- Connect to SPI EEPROM directly to extract or modify its contents
  - May contain firmware!
  - Sometimes encrypted – We need access to the MCU!







Source: twitter.com/ghidraninja, also see [video]()

IAIK TU Graz

Debugging Interfaces (e.g. JTAG)

- Most MCUs and many CPUs have some low-level debugging interface
  - Single-step execution, inspect registers & memory, … during development

- Usually disabled for production
  - E.g. ARM Cortex-M: Firmware can disable SWD (~JTAG)
    - Can we simply flash a modified firmware?
  - Readout Protection (RDP): Prevent reading out flash contents (firmware)
    - Completely lock flash (even to MCU) while a debugger is connected

- Various physical attacks for working around these protections
  - Assemble flash content from incremental SRAM snapshots
    (Source: Obermaier et al.: Shedding too much Light on a Microcontroller's Firmware Protection)
  - Voltage Fault Injection to make MCU bootloader skip RDP check
    (Source: Bozzato et al.: Shaping the Glitch: Optimizing Voltage Fault Injection Attacks)

Cold Boot Attacks

Observation: RAM retains content for short duration after power loss

Can be exploited if

- We can remove the RAM and read it from another machine
- We can load another OS/FW that we have full control over
  - E.g. if bootloader is unlocked
- Mitigations: e.g. HW-based encryption, evicting keys from memory

Lots of other hardware-based side-channel attacks also affect mobile devices!

Tamper Detection & Prevention

Some devices include physical means to detect and prevent tampering

Tamper Prevention

- Use security screws
- Encapsulate PCB in chemical-resistant resin

Tamper Detection

- Sensors (Heat, Temperature, Light, Voltage, …)
- Switches that detect case opening

Higher Level Interfaces

High-Level Interfaces

- More sophisticated interfaces are available
  – Higher speeds
  – Wireless connections
  – More complex protocols
  – Some security mechanisms

- But still
  – More complex ➔ More prone to implementation flaws
  – Wireless or long-distance protocols ➔ Remote attacks

Wifi & Bluetooth

- Multiple ~remotely exploitable flaws have been uncovered
  - 2017: KRACK – Breaking WPA2 by forcing nonce reuse (Source: krackattacks.com)
    On some Linux and Android versions: Force all-zero encryption key!
  - 2021: BrakTooth – Flaws in BT stacks used by multiple vendors (Source: asset-group.github.io)
    Arbitary Code Execution on some IoT devices
- More generic attacks:
  - Relay attacks on Bluetooth (Low Energy) possible
  - Evil Twin attacks on open Wifi access points

Cellular Connections

- Particularly critical communication interface of many mobile devices
  - Mobile phones, cars, alarm systems, ATMs, …
  - Provides essential services to these devices
  - Also gets access to sensitive data from these devices

- Large number of influencing factors for design and operation
  - Regulatory bodies
  - Backwards compatibility
  - Cost-effectiveness
  - Security?

Cellular Communication Attacks

- State actors legally get access
  - Providers are legally required to collect data for authorities
  - Location profile, connection log, …
- IMSI Catchers
  - GSM clients blindly trust the cellular network
    - Mostly fixed in 3G (but GSM interoperability)
  - Fake cell network station that can collect IMSI identifiers
  - Acting as a MitM, it can dictate the encryption used for the connection
- Encryption flaws
  - Some legacy encryption (GSM) algorithms are broken
  - Still used in some countries!

MQTT (MQ Telemetry Transport)

- Simple publish-subscribe protocol for IoT devices, usually over TCP
- Star-shape topology: All communication routed via broker
- Popular in Smart Home gadgets

Problems

- Original version sent credentials in clear
  - Fixed by adding TLS layer
- Real-world MQTT brokers rarely (35%) even use password authentication
  Source: blog.avast.com
- Distinction between clients is the responsibility of broker implementation

Firmware

Embedded Firmware

- Usually either based on open-source OS kernel or custom implementation
  - Both options are interesting research targets!

- Open-source: Big impact for any vulnerability discovered
  - BadAlloc: Bug in FreeRTOS enabled RCE on millions of devices

    Source: msrc-blog.microsoft.com

- Custom implementation: Security usually not primary concern
  - Or no external security audit

Firmware Extraction

- Obtain firmware image from vendor website
  - Embedded Linux: Commonly squashfs root filesystem
- Dump from external EEPROM/Flash chip
  - Some devices run off of (micro) SD cards!

- Use `binwalk` for identifying image type
- Entropy can tell you about encryption

Reverse-Engineering Firmware

- Static analysis using e.g. open-source Ghidra tool
  - Support for many instruction-sets (ARM Cortex-A, Cortex-M, …)

- Embedded Linux:
  - Analyse init procedure, kernel modules, userspace libraries & programs
  - Device tree, configuration files

- Microcontroller:
  - Low-level firmware difficult to understand
    - Accesses to arbitary memory-mapped IO locations = HW registers
  - Construct memory region map from datasheet

Testing Firmware

In some cases, it is helpful to execute extracted firmware in a virtual device

- Embedded Linux
  - QEMU for virtualising CPU on a system / per-process level
  - chroot for running extracted rootfs (if same CPU architecture as host)
  - `LD_PRELOAD` for adding compatibility shims
- Microcontrollers
  - QEMU also supports common MCU architectures (e.g. Cortex-M3)
  - Needs definitions for virtual peripherals

Outlook

- 20.05.2022
  - Mobile Security Research
- 10.06.2022
  - Assignment 2 Presentations