FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Smart Home Devices Vulnerability Analysis
Student:	Bc. Martin Šutovský
Supervisor:	Ing. Jiří Dostál, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security
Department:	Department of Information Security
Validity:	until the end of summer semester 2023/2024

Instructions

With the increasing popularity and availability of Smart home appliances, the number of IoT (Internet of Things) devices has grown significantly in recent years. These devices allow users to effectively control and remotely monitor their homes. However, there is little knowledge of the cybersecurity maturity of such devices. They can introduce a significant security risk, as they often take control of essential home building blocks. The main goal of this thesis is to analyze the security state of smart home devices and determine security issues/challenges in the home automation appliance industry.

1) Perform a review of the smart home devices market. Classify the most common devices into up to 5 classes (e.g., a smart plug, smart lock, white goods, ...).
Propose criteria (e.g., RF technology, number of interfaces, ...) and select devices from each class for the subsequent analysis.
2) Analyze selected devices from a cybersecurity physical and remote access point of view.
3) Based on the previous analysis, identify the most severe issues.
4) Compare your results with the existing rankings of IoT cybersecurity issues, e.g., OWASP IoT Top Ten.
5) Discuss the results and propose generally applicable mitigation and remediation measures.

Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 15 January 2023 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master’s thesis

Smart Home Devices Vulnerability Analysis

Bc. Martin Šutovský

Department of Information Security
Supervisor: Ing. Jiří Dostál, Ph.D.

May 4, 2023

Acknowledgements

I want to thank my supervisor for the guidance and advices during the work on the thesis. I wanna thank my colleagues at Accenture for their pricless hints. And I also want to thank my family and girlfriend for their support during my studies.

Declaration

In V Praze on May 4, 2023 . . . .

Citation of this thesis

Šutovský, Martin. *Smart Home Devices Vulnerability Analysis.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Abstrakt

Dostupnosť IoT zariadení za posledné roky vzrástla a uživatelia implementujú čoraz častejšie smart home prvky do domov. Popularita týchto zariadení vychádza z použitia nových technologií a protokol, ktoré uľahčujú control domácnosti. Smart home je jedna z kategórii IoT zariadení, kedže smart home ekosystém spája všetky zariadenia do jednej siete. Bezpečnosť súčasných IoT zariadení ale ešte nebola dostatočne preskúmaná. Preto je dôležité vyhodnotiť bezpečnosť smart home zariadení, ktoré kontroluje zásadné a citlivé aspekty domácnosti. Posledný populárny zoznam IoT zraniteľnosti vyšiel v roku 2018. Cieľom práce je vyhodnotiť bezpečnosť smart home zariadení a porovnať výsledky s existujucími štúdiami. Práca obsahuje návrh metodológiu pre rýchlu identifikáciu IoT zraniteľností vo forme minipentestu. Vybrané kategórie smart home zariadení boli chytré kamery (Tapo C200, Tapo C320WS), chytré kontrolné jednotky (Tesla Zigbee Hub), chytré zámky (Danalock, Danapad) a chytré senzory (Tesla smart smoke detector). Dokopy bolo analyzovaných šesť zariadení - dve kamery, jedna kontrolná jednotka, dve zariadenia pre chytré zámky a jeden sensor. Analýza odhalila zraniteľnosti, ktoré boli porovnané s OWASP IoT Top 10 a s výsledkami súčasne prebiehajúceho výskumu, ktorý navrhuje nový zoznam najčastejších IoT zraniteľností.

Klíčová slova IoT, Smart Home, bezpečnostná analýza, OWASP, modely hrozieb

Abstract

Smart home device availability has risen in the last few years, and users widely implemented smart home elements into their houses. The popularity of these devices stems from the usage of new technologies and protocols, which make the control of the house more convenient. The smart home is one of the categories of IoT devices, as the smart home ecosystem connects all devices into one network. Nevertheless, the security evaluation in IoT devices remains unknown. Therefore, it is vital to examine the security state of smart home devices, which controls crucial and sensitive aspects of a house. The last popular overview of IoT devices was released in 2018. The thesis aims to evaluate the security of smart home devices and compare the results with existing overviews. A methodology for the quick identification of vulnerabilities in IoT devices was proposed in the form of a minipentest. The selected categories of smart home devices were smart cameras, smart locks, smart hubs, and smart sensors. In total, six devices were analyzed - 2 smart cameras (Tapo C200, Tapo C320WS), one smart hub (Tesla Zigbee Hub), two smart lock appliances (Danalock, Danapad), and one smart smoke (Tesla smart smoke detector). The analysis uncovered vulnerabilities compared with OWASP IoT Top 10 and with simultaneous research proposing a new list of top IoT vulnerabilities.

Keywords IoT, Smart Home, Security Analysis, OWASP, threat model

Contents

List of Figures

List of Tables

Introduction

Over the past decade, IoT devices have been on a surge. Its usage has been widespread, from medical appliances to the automotive industry. The smart home is another category where IoT devices are implemented increasingly. A smart home allows users to remotely or effortlessly control many functions, from lights to temperature to door unlocking. However, it has become apparent that the security of IoT devices is only sometimes adequately implemented, and sometimes, they contain vulnerabilities that can be trivially exploited. The smart home poses an even greater risk since compromising any of the devices can threaten the safety of users in their homes.

The topic was chosen because there has yet to be an overview of vulnerabilities in contemporary smart home devices, and it is essential to identify common threats as developers can use such knowledge to enhance the security of their devices. Therefore, the thesis aims to practically review the state of current IoT security, propose a methodology for IoT security testing and confront the results with an existing list of vulnerabilities. The analysis will focus on selected devices from hardware, firmware, and wireless communication perspective. The analysis consists of the following:

- Gather information about the device
- Model potential threats
- Describe the vulnerabilities and exploits
- Document the output and compare the results with the existing overview of IoT vulnerabilities

The structure of the thesis is as follows: the first chapter introduces the concept of IoT and smart homes and identifies possible threats. The second chapter defines methodology, technologies, and protocols. The next chapter reviews the market, selects categories of devices with high value for adversaries, and selects devices for testing. The fifth chapter analyzes the selected device based on a previously defined methodology. The sixth chapter compares results with the existing list of IoT vulnerabilities. The final chapter concludes the thesis.

Chapter 1

State-of-the-art

1.1 IoT

Technological advancements in semiconductor technology in the last years have made the size of chips and microcontrollers increasingly compact. Such improvements have resulted in implementing modern technology into a broader spectrum of devices, leading to the rise of Internet of things (IoT). As a result, vendors implemented modern technologies and protocols into devices such as household compliances, cars, medical devices, or embedded devices. The inception of IoT dates back to 1980 when a modified soda drink machine that was connected to the internet and could tell the number of drinks and temperature was introduced [1]. In 1990, a remotely controlled toaster was developed as another example of proof of concept for IoT. In the following ten years, researchers developed a system based on Radio Frequency Identification (RFID) to identify devices [2]. Since then, the rise has been rapid. IoT is a concept of a global network based on machine-to-machine communication. IoT allows an autonomous and secure connection between real-world devices. These devices can exchange data for performing defined actions or providing information [3]. IoT comprises devices, sensors, cloud infrastructure, and network communication protocols. Users can remotely control or receive information from various devices, making life easier and, to some extent, even more ecological and economical. IoT can be described from a high-level perspective as a five-layer architecture [1]:

- **Coding layer**
- **Perception layer**
- **Network layer**
- **Middleware layer**
- **Application layer**

- **Business layer**

The coding layer defines the identification of each device - in forms such as MAC address or unique ID. Each device contains a set of sensors or actuators in the perception layer. Sensors gather information from the environment, and actuators execute actions affecting the physical world. *The perception layer* sends information from sensors or data from executed actions to the network layer. *The network layer* is responsible for the transmission of data - it is sometimes called *the transmission layer.* Network and radio protocols are defined in this layer - such as WiFi, Zigbee, or 5G. *The middleware layer* contains a system where the data from sensors and actuators are processed. The system can be, for example, a cloud computing system. *The application layer* manages applications and devices based on information and type of application [1]. Previous works additionally defined *the business layer.* However, *the application layer* already contains the functionality of *the business layer.*

As the market of IoT is proliferating, there are four significant domains of application [2]:

- **Industrial Internet of Things (IIoT)**
- **Internet of Medical Things (IoMT)**
- **Smart Cities**
- **Smart Homes**

1.1.1 Industrial IoT

IIoT implements modern technologies in production lines. The base for industrial IoT is machines that communicate with each other. The goal is to create a network where machines can share data to optimize workload and distribute tasks effectively. Also, in IIoT, effective data sharing can prevent failures, provide real-time production data, and ensure constant production [2].

1.1.2 Medical IoT

The main goal of medical IoT devices is to provide a constant flow of information. This information can cover sleep patterns, vital data such as blood pressure or heart rate, or physical activity. They also should allow remote configuration to adjust functionality based on data analysis. A doctor can adjust a pacemaker based on patients' vital information and prevent severe medical conditions [2]. In this category, security is a vital aspect of design as a possible security incident can lead, in the worst case, to the endangerment of a patient's life.

1.1.3 Smart cities

The smart city category defines IoT's very influential application in society. With the growth of urbanization, citizens and building fill the cities rapidly. Advanced technologies in IoT can help manage urbanization. As a simple example, IoT can help with traffic optimization by leveraging data from a sensor on traffic lights. Another example can intelligent garbage collection. Sensors placed into garbage bins can help determine how much each bin is filled and optimize the garbage collection road [2].

1.1.4 Smart Home

There is an extensive scale of IoT products for the smart home - usually labeled as smart devices - ranging from smart thermostats to smart TVs or blinds. As the mentioned example of remotely controlled toasters, houses and household systems can also implement technological advancements of IoT. The options are virtually limitless.

1.2 Threats

The growth of IoT devices on the market has been extremely rapid in the last few years. Estimates show that in 2025, there are going to be around 75.44 billion devices available [4]. While the growth is remarkable, it is worth considering what it means. Manufacturers will make new devices as fast as possible and focus on functionality. Which in turn could result in neglected security in design. At the same time, compared to other cybersecurity areas, IoT security gets less attention than web security. For web security, there is a list of the most common vulnerabilities every year, whereas, for IoT security, the latest list made by organizations such as OWASP was released in 2018. The list is the following [5]:

- **Weak, Guessable, or Hardcoded Passwords** mean using easily brute-forced, publicly available, or unchanged credentials. Includes backdoors in firmware or software, which can grant access to the system.

- **Insecure Network Services** is running insecure network services on the devices. More severe are even services exposed to the internet, meaning an attacker does not have to be on the same network to access and possibly exploit them. Insecure Ecosystem Interfaces include insecure web, backend API, or cloud/mobile interfaces outside the device, which could allow compromising the device or related components.

- **The lack of a Secure Update Mechanism** allows an attacker to exploit already identified vulnerabilities because the manufacturer does not have a mechanism for updating and fixing the vulnerabilities. Insufficient update mechanisms might include a lack of firmware validation or insecure delivery.

- **The use of Insecure or Outdated Components** occurs when a device is using insecure or deprecated software components or libraries. If the components or libraries contain a vulnerability, an attacker could exploit it to access the device.
- **Insufficient Privacy Protection** means insufficient protection of personal information stored on a device.
- **Insecure Data Transfer and Storage** allows for sniffing or gaining sensitive data in any other way while they are being transferred or accessing them from an insecure storage mechanism.
- **Lack of Device Management** is a lack of security support for deployed devices in production.
- **Insecure Default Settings** occur when the manufacturer ships a device or a system with insecure default settings. The device might stay insecure because users are not guaranteed to change the default settings or know how to change them appropriately.
- **A lack of Physical Hardening** means a lack of physical hardening measures in the device. If the attacker can access the board, they can gain sensitive information to help with further attacks.

Such thought is not only a prediction, but in many cases, it is reality. Vulnerabilities in IoT devices occur often, and they are often severe as well. To mention a few, in August 2021, researchers disclosed vulnerability in Realtek SDK. This vulnerability could result in unauthenticated remote command execution [6]. Another example is a series of security vulnerabilities in multiple smart home hubs, which could lead to remote code execution, data leaks, or man-in-the-middle attacks [7]. Even though researchers responsibly disclosed discovered vulnerabilities, there are many others, which are exploited in the wild before even researchers discover them. Botnet RedGoBot was exploiting the previously mentioned vulnerability in Realtek SDK to gain control [8]. Another of the examples can be the Mirai botnet, which was in its functionality simple but still managed to wreak havoc. Mirai botnet exploited weak and hardcoded credentials in IoT devices, compromising many IoT devices [9]. Security is a crucial concern as IoT becomes increasingly intertwined with everyday life. As mentioned in the previous section, a security incident can have a more significant impact than in other segments of the information technology industry. A vulnerability in IoT can not only jeopardize users' privacy, but it can represent actual harm to the users - ranging from medical devices to heavy industrial machinery. Security concerns also pose an issue for manufacturers. An adversary can gain knowledge about the internal network or database of the manufacturer and access users' information, leading to dangerous reputational damage. The interconnectivity of IoT devices has other security implications, as compromising one device can lead to compromising others in the same network [2]. That does not only mean compromising other IoT devices but also other systems in the network. An insecure IoT device connected to a corporate network can provide a pivot point for an adversary to compromise other devices or systems. IoT security differs from traditional IT security; most obviously, it differs in the ecosystem. Typically, the IoT ecosystem comprises embedded devices or sensors, cloud infrastructure, and mobile apps - all communication through various protocols. Such an ecosystem contains many attack vectors, and securing all of them is challenging [10].

An example of such an ecosystem is a smart home system controlling lights.

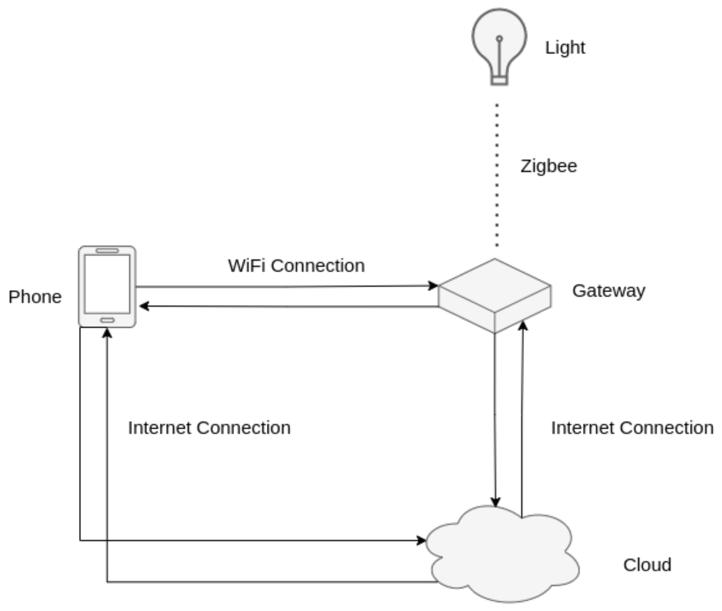


Figure 1.1: IoT Ecosystem Example

In this example, the gateway controls the lightbulb via ZigBee protocol. Gateway communicates with users' phones and the cloud at the same time. The examples assume the phone and gateway must exist on the same WiFi. User is logged in with their credentials to the gateway. API requests from the user to the gateway can change the lightbulb status. If the user sends the correct API request, the gateway will send a ZigBee message to the lightbulb, which will, in turn, act as desired. Even a simple example like this can contain multiple possible threats. For example, an attacker can capture and replay a ZigBee message. If a replay attack works, an attacker can control the lightbulb with the correct command. Another option is to explore network communication between the phone and gateway. If there is no authentication between both devices, an attacker can send the same HTTP request to the gateway. Alternatively, the gateway has its credentials to the cloud. An attacker can extract firmware and memory from the gateway and then access the cloud, posing as the gateway.

Setting security objectives of IoT can help evaluate security mechanisms implemented in IoT devices. From the manufacturer's perspective, it can help solve potential security issues during design and development, and for security researchers, it can help identify potential attack vectors. In [2], the proposed security objectives are the following:

- **Identification**
- **Authentication, authorization**
- **Integrity**
- **Confidentiality**
- **Privacy**
- **Availability**
- **Non-repudiation**

Identification ensures the identification of other devices or participants in the network - not only identifying participant presence but distinguishing between malicious and benign. *Authentication and authorization* mean that before the device allows another device or user to access restricted resources, it first verifies identity and access rights. The data and messages cannot be modified, altered, or destroyed during transmission, which is what *integrity* is. *Confidentiality* is protecting confidential data from unauthorized disclosure. To ensure *privacy* is to protect the personal information of individuals correctly. *Availability* ensures that the system and its services are available whenever required. Finally, *non-repudiation* ensures that some entities can deny the existence of transition [2].

Identifying security threats for any system is vital to any threat assessment. As mentioned before, the IoT ecosystem can be split into layers. Since each layer can use various technologies and protocols, they also contain various threats. Some of the threats for each layer are listed in the following [11]:

- **Coding layer:** node capturing, node spoofing
- **Sensing layer:** malicious code injection, side-channel attacks, booting vulnerabilities

- **Network layer:** denial-of-service, routing attacks
- **Middleware layer:** Man-in-the-middle (MITM), SQL injections
- **Application layer:** sniffing attacks, access control attacks

Chapter 2

Methodology

2.1 Penetration testing methodology

The penetration testing methodology defines a structure for testing the security of the chosen target. Any relevant analysis should use some defined penetration testing methodology as a base for more thorough research. Many penetration testing methodologies exist - such as the more popular Penetration Test Execution Standard (PTES) or Open Worldwide Application Security Project (OWASP). IoT security testing differs from regular penetration testing, for example, web penetration testing. It differs because IoT consists of many technologies and protocols. It is an important fact to keep in mind when choosing the correct methodology. Popular PTES consists of the following steps [12]:

- **Pre-engagement Interactions**
- **Intelligence Gathering**
- **Threat Modeling**
- **Vulnerability Analysis**
- **Exploitation**
- **Post Exploitation**
- **Reporting**

PTES defines each step more abstractly. IoT smart home security has many possible attack vectors, and addressing these vectors in a defined methodology is essential.

2.2 CVSS

When the vulnerability is discovered, it also needs to be adequately evaluated. Some vulnerabilities are less severe than others, and quantitative evaluation helps identify the most severe ones. One of the standard scoring systems is the Common Vulnerability Scoring System (CVSS). It is a framework for quantitative measurement of vulnerabilities [13]. It counts the measure based on the factors such as the complexity of the attack, if it requires high privileges, or if an attacker needs to be close to the target. The CVSS generates three types of score: *base score*, *temporal score* and *enviroment score*. The base score ranges from zero to ten. The CVSS has two main versions: version v2 and v3. The discovered vulnerabilities will be rated with CVSS v3 base score during the security analysis an.

2.3 OWASP methodology

OWASP is a non-profit organization working to improve security. It is a source of information for developers and technologists to secure mainly the web [14]. It offers penetration testing methodologies for web, mobile apps, or IoT. They also list the top 10 vulnerabilities in each information technology security segment. However, only some of these lists are up-to-date, as the primary domain for OWASP is web security. Nevertheless, the methodology for IoT security can serve as a strong base for research. Defining extensive methodology allows for research to be precise and comprehensive. It also helps identify attack vectors that researchers can examine. OWASP published IoT device penetration testing methodology, which serves precisely this purpose. In the IoT device penetration testing methodology, the following attack vectors are defined [15]:

- **Hardware**
- **Firmware**
- **Network**
- **Wireless communication**
- **Mobile and web application**
- **Cloud API's**

Each of the vectors mentioned should be tested for vulnerabilities or for information about a device that reveals the workings of a system. Testing **hardware vector** means opening the device and examining the printed circuit board (PCB) and implemented chips. From a hardware perspective, an adversary can try various attacks. To mention less complex ones, chips on PCB can use internal communication protocols such as Universal asynchronous receiver-transmitter (UART) protocol or Serial Peripheral Interface (SPI) protocol. If the adversary can communicate with chips using mentioned protocols, they can either discover some crucial information about the system or get direct control over the device. From another perspective, an adversary can try to extract firmware from these components if a PCB contains Electrically Erasable Programmable Read-Only Memory (EEPROM) or flash memory [15]. For **firmware testing**, an adversary has to have access to the firmware. They can obtain it from hardware components and protocols or network communication. Accessing firmware means accessing the code or filesystem present on the device. Some microcontrollers might not run an operating system (OS) with the filesystem. However, they are still running code that might contain vulnerabilities or have sensitive information such as keys, certificates, or credentials. Furthermore, if an adversary can also write firmware to memory, they can modify it to perform malicious actions or add a backdoor [15].

In a typical design, devices communicate through a network, for example, through WiFi. Some devices run a web interface for configuration, or various network services run on a device. Therefore, the network vector is vital to test. If the running services are outdated or vulnerable, an adversary might use them to gain access [15]. From this perspective, IoT testing seems more like regular penetration testing of network services.

Wireless communication vector is another approach to testing IoT devices. In the IoT ecosystem, devices often use wireless protocols to transfer data or to communicate with other devices. Attacks can range from sniffing, modifying, or jamming packets to exploiting protocol implementation directly in the device [15].

In an IoT ecosystem, users can often control the system through an application on their mobile phones. **Mobile application vectors** can uncover vulnerabilities to exploit mobile apps, help discover the system's inner workings, or disclose API keys or credentials to the cloud [15].

Cloud vector targets the control mechanism for the whole ecosystem. If the attacker gains access to the cloud, they can virtually control any device connected to that cloud. In a simple example, an adversary can gain users' credentials to the cloud and therefore has complete control [15].

In this thesis, a modified methodology of OWASP IoT methodology will be used. As the goal is to test multiple devices, the methodology will be modified to focus on *hardware vector*, *firmware vector*, and *wireless communication vector* - namely, Bluetooth Low Energy (BLE). The following sections will describe these vectors, technologies, and protocols discovered in the research.

2.4 Hardware chain of trust

The very first act is to examine the chips and components. To test IoT devices from a hardware respective, a researcher examines the PCB of the device. It allows an understanding of how the board works and its functionality. Identification of board functionality involves reading out the chip description and discovering what kind of chip it is. After identifying each chip on the board, a researcher can look for **PCB pins**.

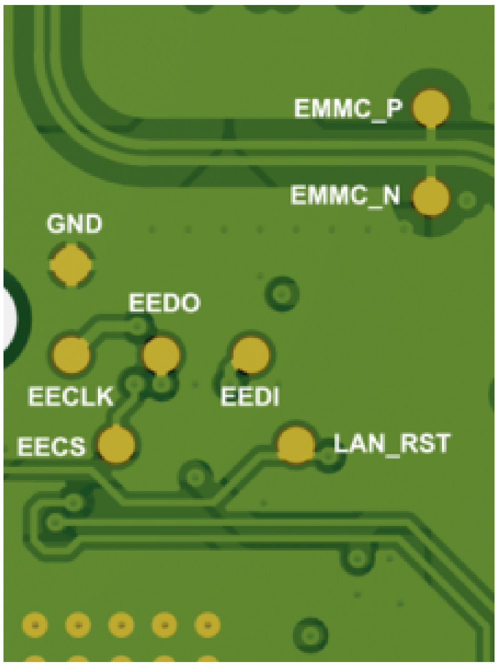


Figure 2.1: PCB Pin [16]

They serve for serial communication with some components of the board. There might be some labels for these ports, which help identify their protocol. If no label exists, their functionality can be identified with a logic analyzer (like Salea, for example) or with a voltmeter. To identify protocol or functionality with a voltmeter, a researcher can use the resistance mode of the voltmeter to find out where the connection from the pin leads. It can lead to one of the pins on the chip. If the datasheet for the chip is available, it is trivial to find the pin assignment from the datasheet and determine functionality. After the protocol identification, a researcher can try to communicate through a pin to gain information about the system and get direct control over the device. In the following paragraphs, standard communication protocols will be described.

2.4.1 UART

Universal Asynchronous Receiver-Transmitter (UART) protocol is a serial protocol. UART is used for serial data transmission. Serial protocol means the data are transferred between components one bit at a time [10]. It is one of the most common serial interfaces used. As an asynchronous protocol, communicating components must agree on clock speed or baud rate. The baud rate sets the maximum number of bits that components can transfer by UART protocol in a second. The typical baud rate for UART is 115200 bits per second [17]. UART typically uses 3 or 4 wires. The first three wires are ground (GND), transmit (TX), and receive (RX). The optional fourth wire is Voltage (VCC), which sets the voltage level for the logical one. A typical reference voltage for UART is 3.3V [17].

2.4.2 SWD

Serial-wire debug (SWD) is a two-pin interface designed for communication with ARM processors. ARM processors are increasingly popular in IoT devices, and therefore, SWD is one of the standard pins on PCBs. SWD offers bidirectional data connection. SWD allows control and debugs of the processor, which means that with SWD, an attacker can control the flow of the processor [10]. SWD requires two pins. SWDIO pin transfer data between devices bidirectionally, and SWDCLK transfers information about the clock. Therefore, it provides a separate clock connection and allows data transfer synchronously [18].

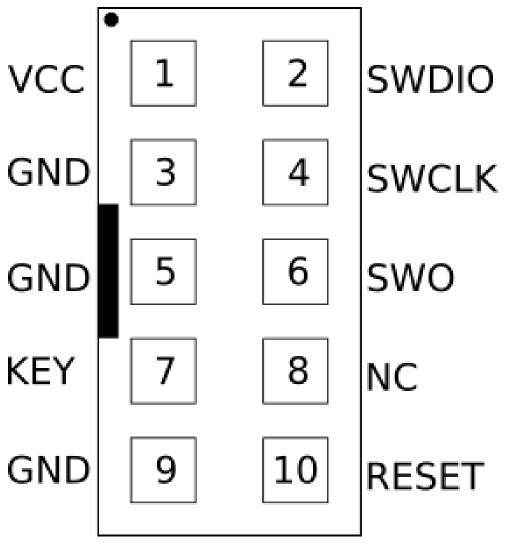


Figure 2.2: SWD Pins [19]

2.5 Firmware

Firmware links hardware components to the main software layer of the device. It is software providing communication and control over hardware components. Often, firmware first boots up the operating system of the device. Then it provides specific services for communication with hardware [10]. The range of operating systems on devices varies. They might run complex operating systems (such as Windows 10 IoT Core) or less complex like Blackberry QNX. However, bare-metal IoT devices do not have a typical operating system; instead, they execute assembly instructions on hardware without the overlay of the operating system [10]s . In firmware analysis, firmware from devices has to be acquired first. A simple way is to extract firmware from the hardware debug interfaces, flash memory, or EEPROM. Another approach is to obtain firmware by sniffing out network packets from an over-the-air update. If a device is updated remotely by downloading firmware files from a web server, sniffing out the communication can lead to obtaining firmware.

If the filesystem of an operating system is a part of the firmware, it can be extracted and analyzed. Alternatively, obtained firmware can then be reverse-engineered to uncover software vulnerabilities inside. Software vulnerabilities range from remote code execution through buffer overflow to hardcoded credentials and secrets. The Unix-based filesystem can have files like passwd or shadow, from which hashes can be extracted and cracked.

IoT devices support various computer architectures, but ARM architecture is prevailing. The following section describes ARM architecture.

2.5.1 ARM architecture

ARM is a reduced instruction set computing (RISC) architecture. It exists in 32-bit and 64-bit modes. One of the differences between, for example, x86 architecture is that ARM is a load-store architecture. It means that performing some operations directly on memory is impossible. The processor first loads value from the memory address into a register modifies it, and then writes it back into the memory address. Another feature of ARM is the possibility of adding a coprocessor, which can execute specific functions like encryption. ARM instructions can be executed in three modes: ARM mode, Thumb-1 mode, and Thumb-2 mode. ARM mode has 4-byte instructions, Thumb-1 uses 2-byte instructions, and Thumb-2 can use either 2-byte or 4-byte instructions. During execution, a processor can switch between the ARM mode and Thumb mode. When BLX or BX instructions are executed, the least significant bit defines whether the operating mode will be Thumb or ARM - value 0 represents ARM mode, and value 1 represents Thumb mode [20]. ARM uses 16 basic registers. Some of them have special functionality: R7 is used for the syscall number, R11 is the frame pointer, R13 is the stack pointer (SP), R14 is the link register (LR), and R15 is the program counter (PC). The link register is GPR, which holds the return address when executing the function [20]. ARM does not use instructions like CALL or RET. Instead, it uses branching instruction with linking. Branching with linking means that before jumping to the target address, the processor will save the return address to the LR register. ARM also has an interrupt vector table. It usually starts at address 0x00. The first value at address 0x00 is the initial value of the stack pointer. Next, it contains handling functions for events like a reset [20].

2.6 Network

In the IoT ecosystem, users often use their phones to receive sensor data or control devices. However, devices can use various protocols, and smartphones do not support them. Therefore, smartphones often communicate with gateway devices in the ecosystem through a network connection, i.e., WiFi. Gateway devices can send and accept HTTP/HTTPS connections. Based on the request, it can communicate with other devices in an ecosystem through its preferred protocol. Such configuration makes the gateway act like a standard HTTP server. From a security perspective, discovering how the gateway processes the information can lead to exciting results and investigating whether it runs other, possibly insecure services.

2.7 Wireless communication

Wireless protocols allow devices to communicate and transmit data to each other. Manufacturers often use short-range radio technologies, meaning the devices usually have low-powered transmitters and receivers. Implementation of protocols and technologies is often up to the manufacturer; therefore, security parameters and secure implementation are also in the manufacturer's hands. Testing wireless communication is, therefore, an option to gain control over devices or force them to perform some action [10]. Typically, testing wireless communication involves specialized hardware, either software-defined radio (SDR) or a specialized sniffer. Sniffing can help uncover sensitive data being transmitted between devices. More interesting attacks involve replay attacks or jamming attacks. In a replay attack, an adversary tries to record a wireless signal transmitted to the destination device and then replay the same signal later, hoping that the device will perform the same action. The device should include a mechanism to discard repeated messages, such as adding an increment to the end of a message, to prevent replay attacks. The following section will describe one of the standard technologies used in IoT: Bluetooth Low Energy (BLE).

2.7.1 Bluetooth Low Energy

Bluetooth Low Energy (BLE) was introduced in Bluetooth 4.0 core specification. One of its perks is low-power consumption. It transmits small of data efficiently; it does not use power constantly but receives and transmits data only when required. BLE uses 40 channels and frequencies from 2400 to 2483.5 MHz - classical Bluetooth uses 79 channels [10].

BLE devices operate in two modes: *broadcasting* and *communication* modes. A device in broadcasting mode sends advertisement packets, indicating its presence to nearby devices. Communication mode serves for data transmission between devices [21].

BLE defines roles for devices: *central* or *peripheral*. In communication mode, the peripheral device uses sensors or actuators, and the central device reads data from sensors or instructs the peripheral device to perform some action through the actuator. In broadcasting mode, peripheral devices send advertising packets, and central devices listen to them and initiate the pairing procedure [21].

A peripheral device uses Generic Attribute Profile (GAP) protocol to provide a central device to read data and write instructions for commands. GAP is a service framework defining reading, discovering, and formatting characteristics. It also defines a way to store data on a peripheral device [21].

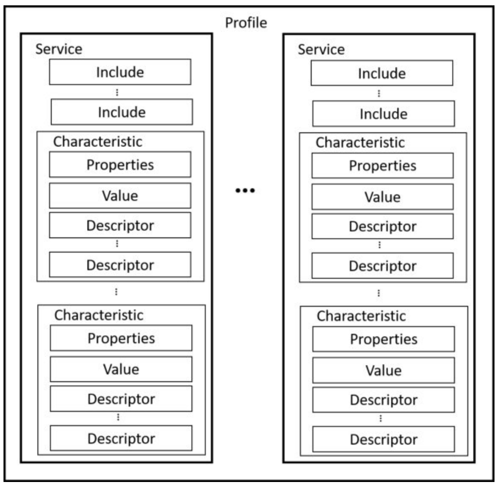


Figure 2.3: GATT hierarchy example [22]

The topmost layer defines the profile, which consists of one or more services. Service contains characteristics or references to other services. Services and characteristics store data, which are used for reading a value from a sensor or writing a value, after which a device will perform some action. Services can be categorized as primary and secondary. Primary services define the main functionality of the device. Secondary services define additional functionalities that reference primary service or perform additional action for primary service[21].

2.8 Proposed methodology

As the thesis aims to confront the security of contemporary IoT devices, following all steps from OWASP methodology would take much time, and it is not feasible to perform a complete analysis for every device. Therefore, a new penetration testing methodology is proposed for the thesis - **minipentest**. A minipentest is a less comprehensive penetration test to gain control over the target device as fast as possible. The methodology is based on PTES, aiming to examine the target device quickly and identify exploitable vulnerabilities. Minipentest consists of the following steps:

1. **Information gathering**
   - Analysis of the target device and collect information about its functionality, technologies, and services
2. **Threat modeling**
   - Based on the analysis, describe possible threats which can lead to vulnerabilities for the target device
3. **Vulnerability analysis and exploitation**
   - Use information from the analysis to identify vulnerabilities and propose exploits
4. **Reporting**
   - Report the results in the form of discovered vulnerabilities and mitigations

Furthermore, the methodology should also be specified for IoT. The IoT methodology selects the domain of interest from the OWASP Methodology. The main focus will be on hardware and firmware security; network and wireless communication security will also be examined. Only simpler technologies, such as Bluetooth Low Energy, will be examined in wireless communication. The methodology for IoT used in the thesis is defined as follows:

1. *Information gathering*
   - Examine the PCB from the perspective of possible hardware interfaces and identify them
   - Acquire the firmware through the hardware interface or network

  - Analyze the network communication
  - Analyze the wireless communication, namely BLE

2. *Threat modeling*
   - Model threats based on the technologies and functionality of a target device

3. *Vulnerability analysis and exploitation*
   - Analyze the information from the first point and identify the vulnerabilities stemming
   - Propose an attack exploiting the vulnerabilities

4. *Reporting*
   - Summarize the discovered vulnerabilities and their impacts
   - Propose mitigations for the vulnerabilities

Chapter 3

Smart Home Devices

The smart home is a category of IoT ecosystems in which IoT concepts and technologies connect appliances to a unified network. As IoT devices have become more available, smart homes have rapidly risen. This chapter identifies the most valuable categories of smart home devices and selects devices for testing based on proposed parameters.

The goal of a smart home is to connect appliances and machines on one network, from which the user can perform control and read data remotely. Smart home networks can contain devices ranging from white goods such as washing machines or fridges through entertainment systems like television to lights or temperature sensors. Compromising each device can pose a risk and at least discomfort to the user. However, some categories have a higher value to an adversary. These categories can lead to accessing the user's house, disrupt privacy, or pose a safety risk.

As smart home networks can use multiple protocols for communication, the typical setup has a gateway that can use various protocols to communicate with end devices. The gateway communicates with the users, providing sensor data and ways to execute actions remotely. So the gateway is a desirable target for an adversary because compromising it can provide complete control over the smart home.

Smart locks are one of the main categories where security should be part of the design. One of the goals of a smart home is to reduce the need to carry items like keys every day. Smart locks are becoming more popular, as they can be unlocked through the phone instead of using the key. Users can open the doors even when they are not close to the house because they can unlock doors virtually anywhere with the correct network design. However, mentioned functionality can also be a huge security risk. Compromising the security of smart locks can effectively allow an adversary to enter the house physically.

Among other features, a smart home monitors the house remotely through cameras. The camera can connect to the network and become part of the ecosystem. They can be installed inside the house and monitor different rooms, or they can be placed outside. The risk is, however, apparent. Gaining control over the camera can have a significant impact on privacy and can also cause significant damage to the manufacturer. For example, EZVIV IoT cameras contained a chain of vulnerabilities that could lead to complete control of the device [23].

The vital part of a smart home is sensors. They can range from temperature and humidity sensors to motion detectors. Some sensors, like temperature detectors, provide data that are not valuable to an adversary. Motion sensors, however, can be another goal in an attack on a smart home once the attacker gains access to the house. Furthermore, vulnerabilities in sensors like smoke detectors can cause significant property damage or even put life at risk. The security of sensors should have the great attention of manufacturers.

Mentioned categories of smart home devices are valuable targets for the attacker; therefore, evaluating their security is essential. The final list of categories is following:

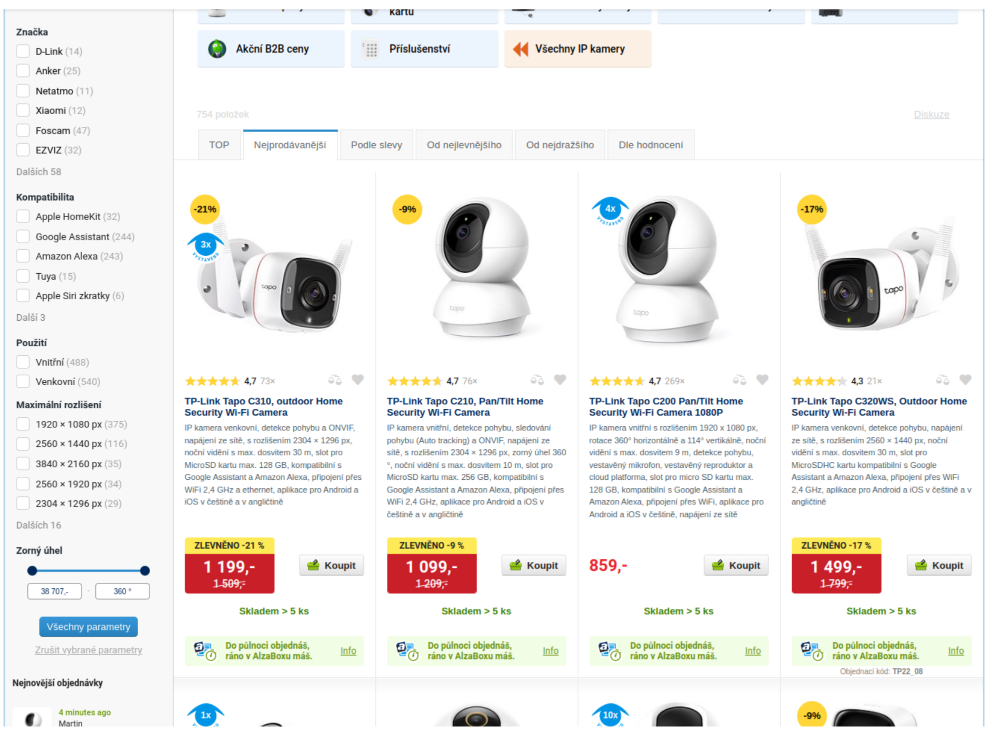
- **smart sensors**
- **smart cameras**
- **smart locks**
- **smart sensors**

For the thesis, devices from selected categories will be analyzed. The following section describes the selection parameters and identifies specific devices.

3.1 Smart cameras

The selected smart camera should offer the functionality of control over Wi-Fi, reasonable resolution, and additional features which would make it attractive for the users. As a first task, the manufacturer should be selected. After looking through the list of products from popular shopping websites (alza and czc), TP-Link is a popular manufacturer.

TP-Link is a widespread manufacturer of network devices such as cameras, routers, and switches. It offers a wide range of smart cameras. From the list of available devices, Tapo C200 and Tapo C320WS are viable candidates. Tapo C200 is a home security camera with rotation both vertically and horizontally. It supports Google Assistant and Amazon Alex and is controlled over Wi-Fi [24]. Tapo C320WS is an outdoor security camera with a resolution of 2560x1440 pixels. It has built-in night vision and support for Google Assistant and Amazon Alexa [25] The Tapo C200 and Tapo C320WS will be tested in the following chapters.



4,7 73×
TP-Link Tapo C310, outdoor Home Security Wi-Fi Camera
IP kamera venkovní, detekce pohybu a ONVIF, napájení ze sítě, s rozlišením 2304 × 1296 px, noční vidění s max. dosvitem 30 m, slot pro MicroSD kartu max. 128 GB, kompatibilní s Google Assistant a Amazon Alexa, připojení přes WiFi 2,4 GHz a ethernet, aplikace pro Android a iOS v češtině a v angličtině
ZLEVNĚNO -21 % 1 199,- ~~1 509,-~~
Skladem > 5 ks
Do půlnoci objednáš, ráno v AlzaBoxu máš.

4,7 76×
TP-Link Tapo C210, Pan/Tilt Home Security Wi-Fi Camera
IP kamera vnitřní, detekce pohybu, sledování pohybu (Auto tracking) a ONVIF, napájení ze sítě, s rozlišením 2304 × 1296 px, zorný úhel 360 °, noční vidění s max. dosvitem 10 m, slot pro MicroSD kartu max. 256 GB, kompatibilní s Google Assistant a Amazon Alexa, připojení přes WiFi 2,4 GHz, aplikace pro Android a iOS v češtině a v angličtině
ZLEVNĚNO -9 % 1 099,- ~~1 209,-~~
Skladem > 5 ks
Do půlnoci objednáš, ráno v AlzaBoxu máš.

4,7 269×
TP-Link Tapo C200 Pan/Tilt Home Security Wi-Fi Camera 1080P
IP kamera vnitřní s rozlišením 1920 x 1080 px, rotace 360° horizontálně a 114° vertikálně, noční vidění s max. dosvitem 9 m, detekce pohybu, vestavěný mikrofon, vestavěný reproduktor a cloud platforma, slot pro micro SD kartu max. 128 GB, kompatibilní s Google Assistant a Amazon Alexa, připojení přes WiFi, aplikace pro Android a iOS v češtině, napájení ze sítě
859,-
Skladem > 5 ks
Do půlnoci objednáš, ráno v AlzaBoxu máš.

4,3 21×
TP-Link Tapo C320WS, Outdoor Home Security Wi-Fi Camera
IP kamera venkovní, detekce pohybu, napájení ze sítě, s rozlišením 2560 × 1440 px, noční vidění s max. dosvitem 30 m, slot pro MicroSDHC kartu kompatibilní s Google Assistant a Amazon Alexa, připojení přes WiFi 2,4 GHz, aplikace pro Android a iOS v češtině a v angličtině
ZLEVNĚNO -17 % 1 499,- ~~1 799,-~~
Skladem > 5 ks
Do půlnoci objednáš, ráno v AlzaBoxu máš.
Objednací kód: TP22_08

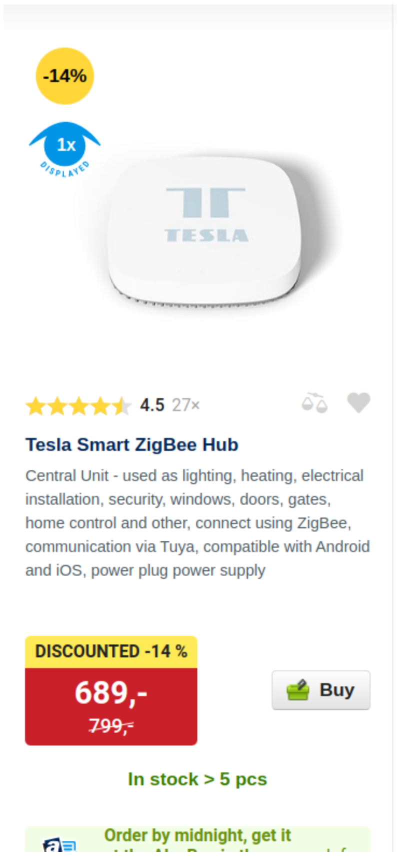
Figure 3.1: Market with smart cameras

3.2 Smart locks

Among smart locks, a reputable manufacturer is Danalock. Danalock offers a smart lock [26], which is installed on the door, and also it offers keypad [27], which can be placed for example, outside of the house and by entering the correct pin, it will unlock the door. They approach the security of their devices very seriously, and generally, there is very little publicly available information about the security incidents of Danalock [28]. It is, therefore, a fascinating target for testing. The analysis will test both the smart lock and keypad from Danalock.

3.3 Smart gateways

A gateway acts as a central unit for the smart home network. It is a part of a network that connects all devices. For more technical users, there are ways how to build a gateway from scratch, but for many users, it is more convenient to buy an already-built device. The market offers many choices for consumers. Among popular brands are Tesla and Immax. Looking through online shopping websites, Immax and Tesla cost reasonably.



Tesla Smart ZigBee Hub

Central Unit - used as lighting, heating, electrical installation, security, windows, doors, gates, home control and other, connect using ZigBee, communication via Tuya, compatible with Android and iOS, power plug power supply

DISCOUNTED -14 %
689,-
~~799,-~~

Buy

In stock > 5 pcs

Order by midnight, get it

Figure 3.2: Tesla ZigBee Hub

While the gateway from Immax supports Zigbee and Bluetooth, the reviews for Tesla Hub are more optimistic, even though it supports only Zigbee. The TesSmart Home Devices Vulnerability Analysisla brand is more popular than Immax so the average user could prefer a more know manufacturer. Therefore, in the thesis, Tesla Smart Zigbee Hub will be analyzed.

3.4 Smart sensor

A sensor should contain functionality that can advance the house's or the user's safety. It could be, for example, a smoke detector or motion detector. A compromised smoke detector can pose a bigger risk than a compromised motion detector, as a smoke detector can fail to report an ongoing fire and can cause damage. There is an available smoke detector from the same brand as the previously mentioned gateway. It communicates through Zigbee. From a security perspective, choosing the same manufacturer can help with common mistakes in manufactured devices and instruct the manufacturer to avoid such mistakes. In the thesis, the tesla smart smoke detector will be analyzed

Chapter 4

Analysis

4.1 Smart Cameras

This section describes the analysis of smart cameras Tapo C3200 and Tapo C320WS.

4.1.1 TP-Link Tapo C200

Tapo C200 is a smart home camera with horizontal and vertical rotation. It uses Wi-Fi to communicate with the user's phone. Tapo C200 has Full HD resolution and allows even night vision.

Figure 4.1: TAPO C200 [24]

4.1.1.1 Information gathering

Tapo C200 is a smart camera that is reasonably simple to disassemble. The PCB after disassembly is shown in 4.2.

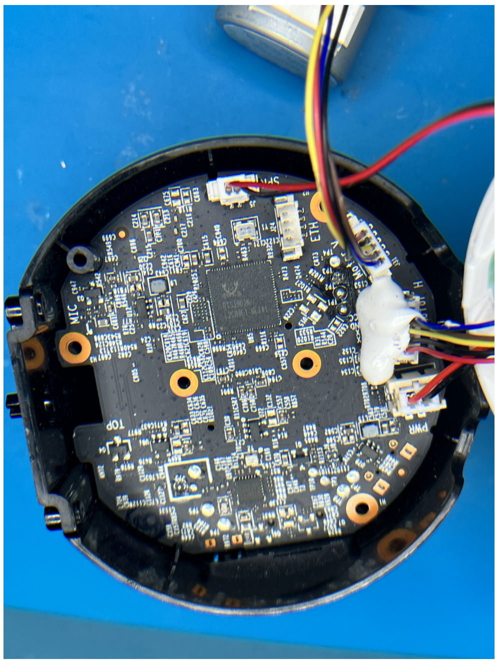


Figure 4.2: Printed circuit board

The main chip on the board is Realtek RTS3903N SoC. This chip acts as the central processor of the board. After a close look at the PCB, there are a lot of pins. Most hardware communication protocols have a ground point (GND) somewhere close, so looking for the ground is a good start. A voltmeter in resistance mode can help with ground identification. In resistance mode, the voltmeter will send a small current to one of the probes. If there is a connection between pins, a resistor inside the voltmeter will trigger a small current. An even better approach is setting the voltmeter to sound mode, which will beep after detecting the connection.

From voltmeter analysis, the pin in 4.3 is ground. The next step is determining the other pins and possibly the protocol used. An obvious way is using a voltmeter again. This time, the voltmeter should measure voltage between pins. The discovered ground can serve as a ground reference. Measuring with a voltmeter shows around 3.3 volts on all three points. That does not tell much, but it clearly shows it is worth exploring more. Connecting the pins to the logic analyzer is a more straightforward way. A logic analyzer can convert a signal from pins to a digital signal and help identify incoming communication. A few seconds after start, the third pin gets very noisy -

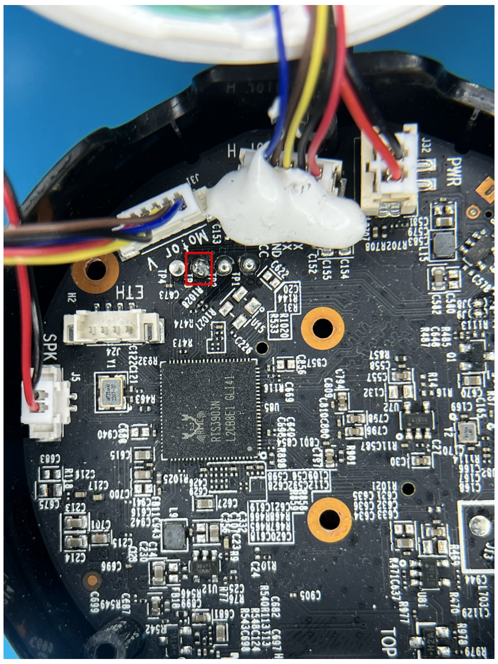


Figure 4.3: Ground pin

some communication is taking place on that pin. Assuming the interface is UART, the third pin could be the transmit pin (TX). If the third pin is the UART TX pin, Salea can decode the signal into readable. First, the baud rate of communication needs to be identified. Baudrate is the highest number of transmitted bytes in a second. Therefore, finding the shortest time interval between the change between logical one and logical zero can help identify the baud rate, as 4.4 shows.

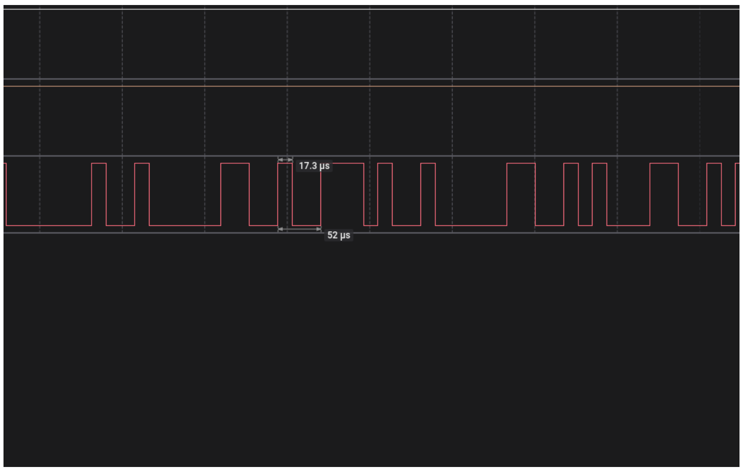
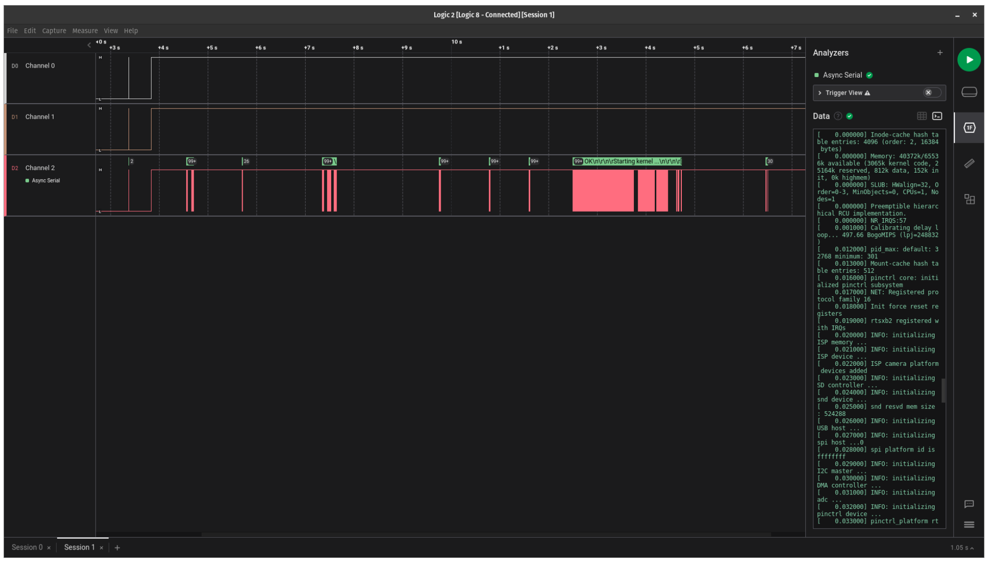


Figure 4.4: Baudrate

In this case, the shortest interval is 17.3 microseconds, 0.0000173. The baud rate would be inverted value of 17.3 microseconds, which results in 57803 bits per second. Looking through standard baud rates for UART, a comparable value is 57600. After decoding the communication with the baud rate set to 57600, there are visible ASCII characters in the Salea software, as shown in 4.5.


Logic 2 [Logic 8 - Connected] [Session 1]
File Edit Capture Measure View Help
Channel 0
Channel 1
Channel 2
Async Serial
OK\r\n\r\nStarting kernel ...\r\n\r\n
Analyzers
Async Serial
Trigger View
Data
Session 0
Session 1
1.05 s

```
[    0.000000] Inode-cache hash ta
ble entries: 4096 (order: 2, 16384
 bytes)
[    0.000000] Memory: 40372k/6553
6k available (3065k kernel code, 2
5164k reserved, 812k data, 152k in
it, 0k highmem)
[    0.000000] SLUB: HWalign=32, O
rder=0-3, MinObjects=0, CPUs=1, No
des=1
[    0.000000] Preemptible hierarc
hical RCU implementation.
[    0.000000] NR_IRQS:57
[    0.001000] Calibrating delay l
oop... 497.66 BogoMIPS (lpj=248832
)
[    0.012000] pid_max: default: 3
2768 minimum: 301
[    0.013000] Mount-cache hash ta
ble entries: 512
[    0.016000] pinctrl core: initi
alized pinctrl subsystem
[    0.017000] NET: Registered pro
tocol family 16
[    0.018000] Init force reset re
gisters
[    0.019000] rtsxb2 registered w
ith IRQs
[    0.020000] INFO: initializing
ISP memory ...
[    0.021000] INFO: initializing
ISP device ...
[    0.022000] ISP camera platform
 devices added
[    0.023000] INFO: initializing
SD controller ...
[    0.024000] INFO: initializing
snd device ...
[    0.025000] snd resvd mem size
: 524288
[    0.026000] INFO: initializing
USB host ...
[    0.027000] INFO: initializing
spi host ...0
[    0.028000] spi platform id is
ffffffff
[    0.029000] INFO: initializing
I2C master ...
[    0.030000] INFO: initializing
DMA controller ...
[    0.031000] INFO: initializing
adc ...
[    0.032000] INFO: initializing
pinctrl device ...
[    0.033000] pinctrl_platform rt
```

Figure 4.5: Data transferred through UART

The next step is correctly identifying the RX pin. It is safe to assume that the third pin is the TX pin for UART. RX can be either of the remaining two, so both can be tested. After trying both pins, the second is working correctly.

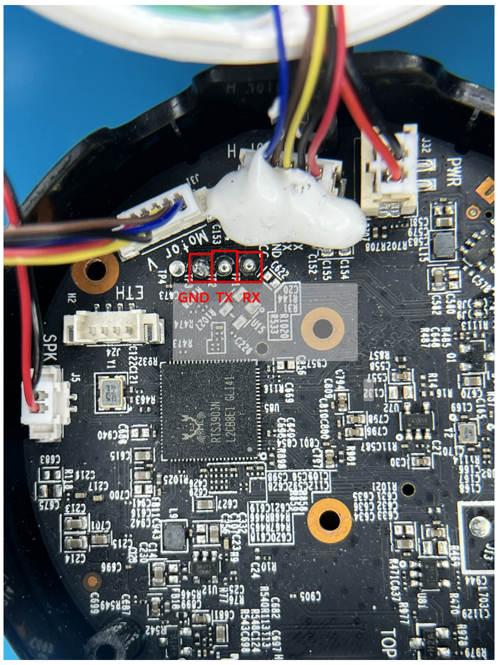
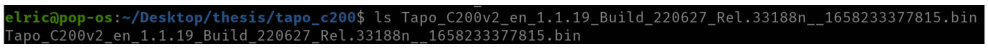


Figure 4.6: UART ports

During the research, TP-Link released an update for their cameras. The update includes software changes, which can introduce some software bugs. The next step is to obtain the latest firmware. One of the ways to obtain it is to capture the request from the communication between the phone and the camera. A typical approach is to use Burpsuite, a tool used for web application testing. It can act as a proxy through which communication between target devices goes.

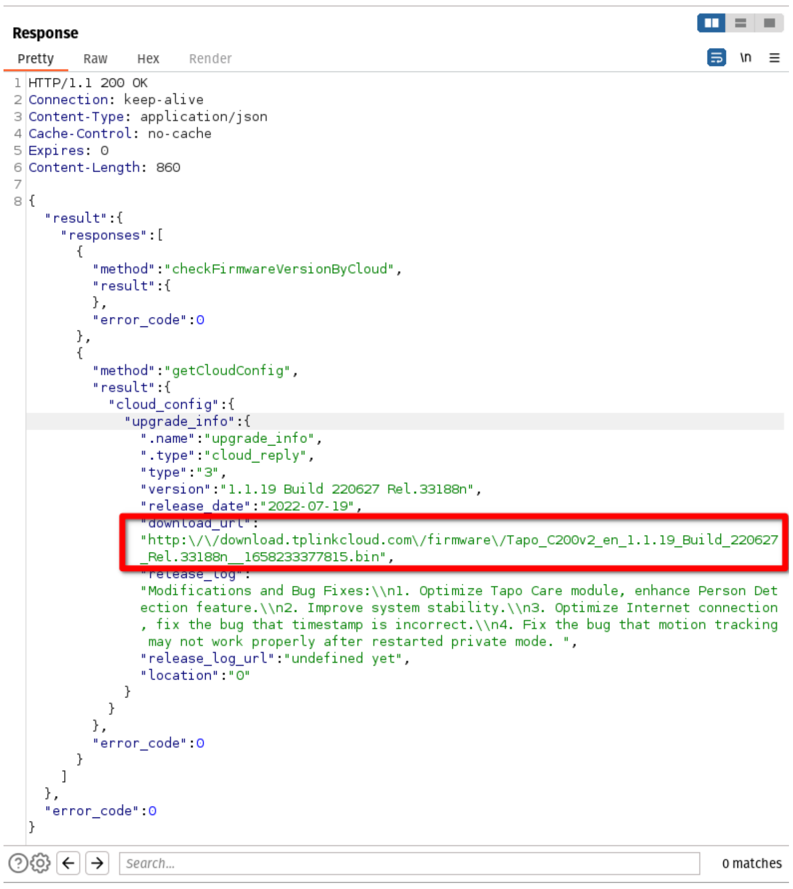
After examining the request for firmware check, the camera responds with JSON containing the URL for new firmware. From that URL, new firmware can be downloaded, as in 4.8.



```
elric@pop-os:~/Desktop/thesis/tapo_c200$ ls Tapo_C200v2_en_1.1.19_Build_220627_Rel.33188n__1658233377815.bin
Tapo_C200v2_en_1.1.19_Build_220627_Rel.33188n__1658233377815.bin
```

Figure 4.8: Firmware file

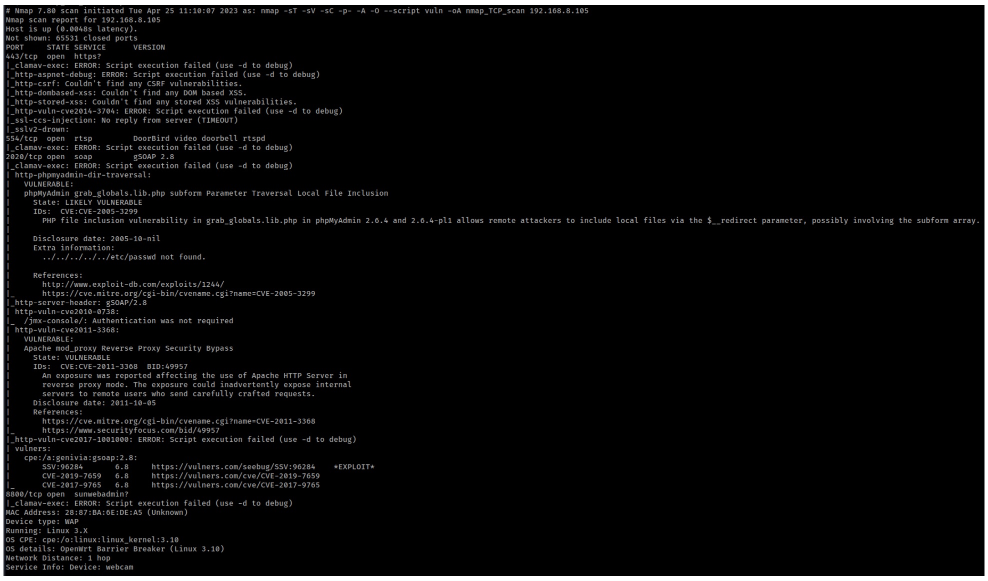
From the network perspective, the camera can be viewed as a target of network penetration testing. It can have services running on different ports, which can contain vulnerabilities. Port scanning is the first step of network analysis. There are various tools for port scanning; a typical one is Nmap. First, TCP ports will be scanned as shown in 4.9.



```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Cache-Control: no-cache
Expires: 0
Content-Length: 860

{
  "result":{
    "responses":[
      {
        "method":"checkFirmwareVersionByCloud",
        "result":{
        },
        "error_code":0
      },
      {
        "method":"getCloudConfig",
        "result":{
          "cloud_config":{
            "upgrade_info":{
              ".name":"upgrade_info",
              ".type":"cloud_reply",
              "type":"3",
              "version":"1.1.19 Build 220627 Rel.33188n",
              "release_date":"2022-07-19",
              "download_url":
              "http:\/\/download.tplinkcloud.com\/firmware\/Tapo_C200v2_en_1.1.19_Build_220627
              _Rel.33188n__1658233377815.bin",
              "release_log":
              "Modifications and Bug Fixes:\\n1. Optimize Tapo Care module, enhance Person Det
              ection feature.\\n2. Improve system stability.\\n3. Optimize Internet connection
              , fix the bug that timestamp is incorrect.\\n4. Fix the bug that motion tracking
               may not work properly after restarted private mode. ",
              "release_log_url":"undefined yet",
              "location":"0"
            }
          }
        },
        "error_code":0
      }
    ]
  },
  "error_code":0
}
```

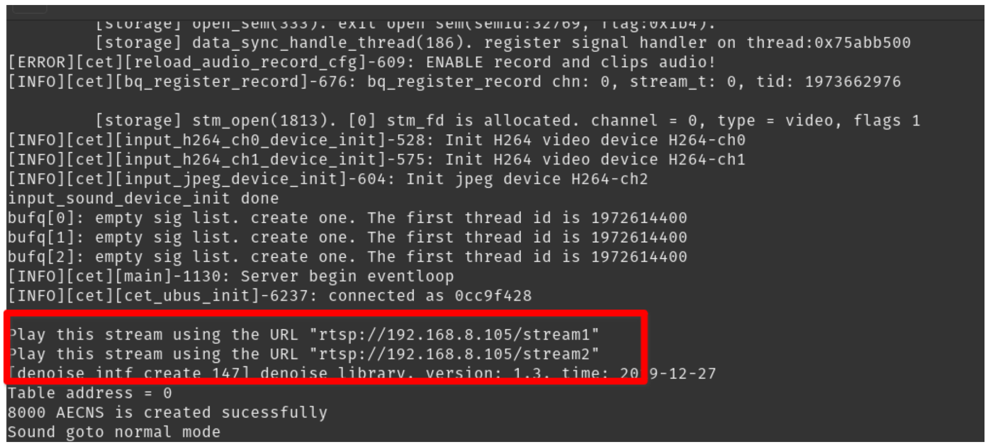
Figure 4.7: Firmware update

There are four open TCP ports - 443,554,2020 and 8800. The critical ports are 443 and 554. Port 443 is the webserver used for communication with the phone. Port 554 is Real-Time Publish-Subscribe (RTPS) protocol, which streams the camera image to the user. To access the stream, a user has to access the stream on a specific URL and with username credentials. Looking back at the UART port, it prints out a logging message while booting up after rebooting. One of the messages is RTPS addresses (4.10).



```
# Nmap 7.80 scan initiated Tue Apr 25 11:10:07 2023 as: nmap -sT -sV -sC -p- -A -O --script vuln -oA nmap_TCP_scan 192.168.8.105
Nmap scan report for 192.168.8.105
Host is up (0.0048s latency).
Not shown: 65531 closed ports
PORT     STATE SERVICE      VERSION
443/tcp  open  https?
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
|_http-aspnet-debug: ERROR: Script execution failed (use -d to debug)
|_http-csrf: Couldn't find any CSRF vulnerabilities.
|_http-dombased-xss: Couldn't find any DOM based XSS.
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
|_http-vuln-cve2014-3704: ERROR: Script execution failed (use -d to debug)
|_ssl-ccs-injection: No reply from server (TIMEOUT)
|_sslv2-drown:
554/tcp  open  rtsp         DoorBird video doorbell rtspd
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
2020/tcp open  soap         gSOAP 2.8
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
| http-phpmyadmin-dir-traversal:
|   VULNERABLE:
|   phpMyAdmin grab_globals.lib.php subform Parameter Traversal Local File Inclusion
|     State: LIKELY VULNERABLE
|     IDs:  CVE:CVE-2005-3299
|       PHP file inclusion vulnerability in grab_globals.lib.php in phpMyAdmin 2.6.4 and 2.6.4-pl1 allows remote attackers to include local files via the $__redirect parameter, possibly involving the subform array.
|
|     Disclosure date: 2005-10-nil
|     Extra information:
|       ../../../../../etc/passwd not found.
|
|     References:
|       http://www.exploit-db.com/exploits/1244/
|_      https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3299
|_http-server-header: gSOAP/2.8
| http-vuln-cve2010-0738:
|_  /jmx-console/: Authentication was not required
| http-vuln-cve2011-3368:
|   VULNERABLE:
|   Apache mod_proxy Reverse Proxy Security Bypass
|     State: VULNERABLE
|     IDs:  CVE:CVE-2011-3368  BID:49957
|       An exposure was reported affecting the use of Apache HTTP Server in
|       reverse proxy mode. The exposure could inadvertently expose internal
|       servers to remote users who send carefully crafted requests.
|     Disclosure date: 2011-10-05
|     References:
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3368
|_      https://www.securityfocus.com/bid/49957
|_http-vuln-cve2017-1001000: ERROR: Script execution failed (use -d to debug)
| vulners:
|   cpe:/a:genivia:gsoap:2.8:
|       SSV:96284       6.8     https://vulners.com/seebug/SSV:96284    *EXPLOIT*
|       CVE-2019-7659   6.8     https://vulners.com/cve/CVE-2019-7659
|_      CVE-2017-9765   6.8     https://vulners.com/cve/CVE-2017-9765
8800/tcp open  sunwebadmin?
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
MAC Address: 28:87:BA:6E:DE:A5 (Unknown)
Device type: WAP
Running: Linux 3.X
OS CPE: cpe:/o:linux:linux_kernel:3.10
OS details: OpenWrt Barrier Breaker (Linux 3.10)
Network Distance: 1 hop
Service Info: Device: webcam
```

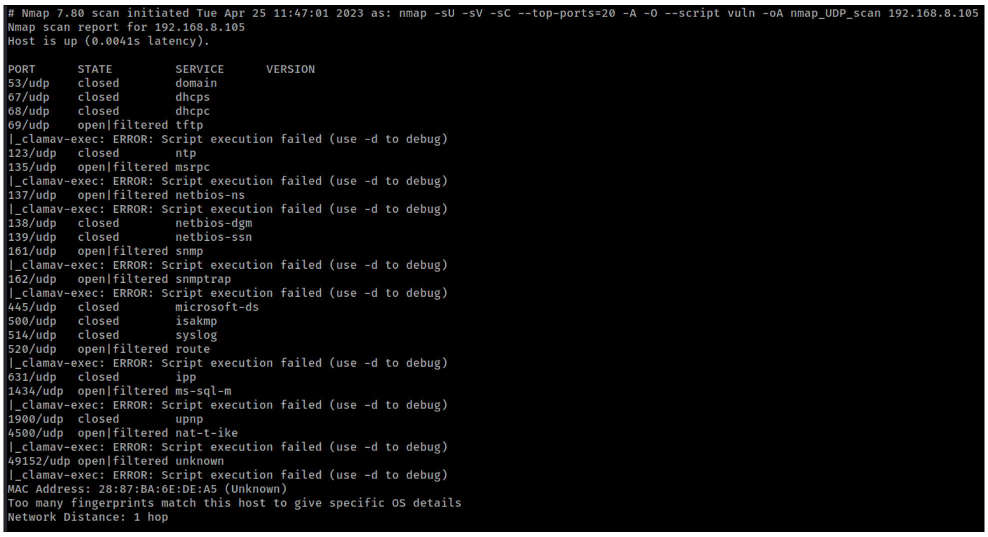
Figure 4.9: TCP scan



```
        [storage] open_sem(333). exit open sem(semid:32769, flag:0x1b4).
        [storage] data_sync_handle_thread(186). register signal handler on thread:0x75abb500
[ERROR][cet][reload_audio_record_cfg]-609: ENABLE record and clips audio!
[INFO][cet][bq_register_record]-676: bq_register_record chn: 0, stream_t: 0, tid: 1973662976

        [storage] stm_open(1813). [0] stm_fd is allocated. channel = 0, type = video, flags 1
[INFO][cet][input_h264_ch0_device_init]-528: Init H264 video device H264-ch0
[INFO][cet][input_h264_ch1_device_init]-575: Init H264 video device H264-ch1
[INFO][cet][input_jpeg_device_init]-604: Init jpeg device H264-ch2
input_sound_device_init done
bufq[0]: empty sig list. create one. The first thread id is 1972614400
bufq[1]: empty sig list. create one. The first thread id is 1972614400
bufq[2]: empty sig list. create one. The first thread id is 1972614400
[INFO][cet][main]-1130: Server begin eventloop
[INFO][cet][cet_ubus_init]-6237: connected as 0cc9f428

Play this stream using the URL "rtsp://192.168.8.105/stream1"
Play this stream using the URL "rtsp://192.168.8.105/stream2"
[denoise_intf_create_147] denoise library, version: 1.3, time: 20 9-12-27
Table address = 0
8000 AECNS is created sucessfully
Sound goto normal mode
```

Figure 4.10: RTPS addresses

It is also vital to examine UDP ports. Nmap can also scan UDP ports; however, in this case, reviewing all possible UDP ports would take a long time, so it is more effective to check only standard ports. In this, the first 20 common ports will be scanned. The Nmap command for scanning is the following:



```
# Nmap 7.80 scan initiated Tue Apr 25 11:47:01 2023 as: nmap -sU -sV -sC --top-ports=20 -A -O --script vuln -oA nmap_UDP_scan 192.168.8.105
Nmap scan report for 192.168.8.105
Host is up (0.0041s latency).

PORT      STATE         SERVICE      VERSION
53/udp    closed        domain
67/udp    closed        dhcps
68/udp    closed        dhcpc
69/udp    open|filtered tftp
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
123/udp   closed        ntp
135/udp   open|filtered msrpc
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
137/udp   open|filtered netbios-ns
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
138/udp   closed        netbios-dgm
139/udp   closed        netbios-ssn
161/udp   open|filtered snmp
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
162/udp   open|filtered snmptrap
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
445/udp   closed        microsoft-ds
500/udp   closed        isakmp
514/udp   closed        syslog
520/udp   open|filtered route
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
631/udp   closed        ipp
1434/udp  open|filtered ms-sql-m
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
1900/udp  closed        upnp
4500/udp  open|filtered nat-t-ike
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
49152/udp open|filtered unknown
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
MAC Address: 28:87:BA:6E:DE:A5 (Unknown)
Too many fingerprints match this host to give specific OS details
Network Distance: 1 hop
```

Figure 4.11: UDP scan

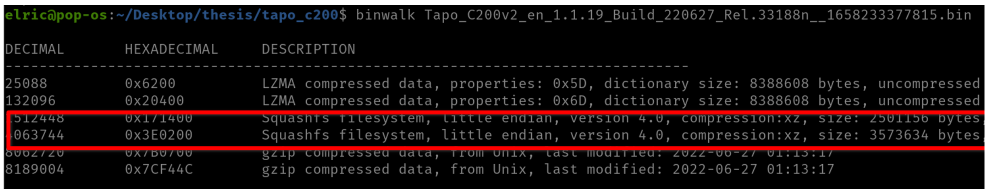
4.1.1.2 Threat modeling

There are a few layers of threats that can compromise the device. As the UART interface is available in the camera, it can access the shell or root shell, which gives an adversary complete control over the device. Another threat is the availability of firmware. The firmware analysis can uncover software vulnerabilities, ranging from information disclosure to remote code execution, or the firmware might contain some hard-coded secrets, which can then compromise the device. The leading service responsible for communication with the phone is the HTTPS server on port 443. The misconfigurations in the server can lead to typical web application vulnerabilities. An adversary can access the user account or read files through local file inclusion (LFI) if the session management is misconfigured. Furthermore, the camera contains multiple TCP/UDP ports. If these ports run vulnerable or misconfigured services, the vulnerabilities can even allow the attacker to get remote access.

4.1.1.3 Vulnerability analysis and exploitation

During the analysis, the UART interface was identified on the board. A device called a TTL-to-USB adapter can help with communication through this interface. The next step is examining the functionality of the UART interface. After pressing enter, the interface shows a login prompt. It would be fair to assume that the user is the *root*. Therefore, some default passwords can be tested. Unfortunately, neither of the typical passwords like *root*, *toor*, or *123456* does not work. However, after searching for some information about the camera, one of the GitHub pages contains the password. The password is *slprealtek*. The username **root** and password **slprealtek** allows accessing the root shell. However, TP-Link released an update for the camera during the research. After updating the device, a password for the UART shell does not work anymore, but for the newly bought camera, the update is not a part of the default build, so it is possible to use the UART shell still.

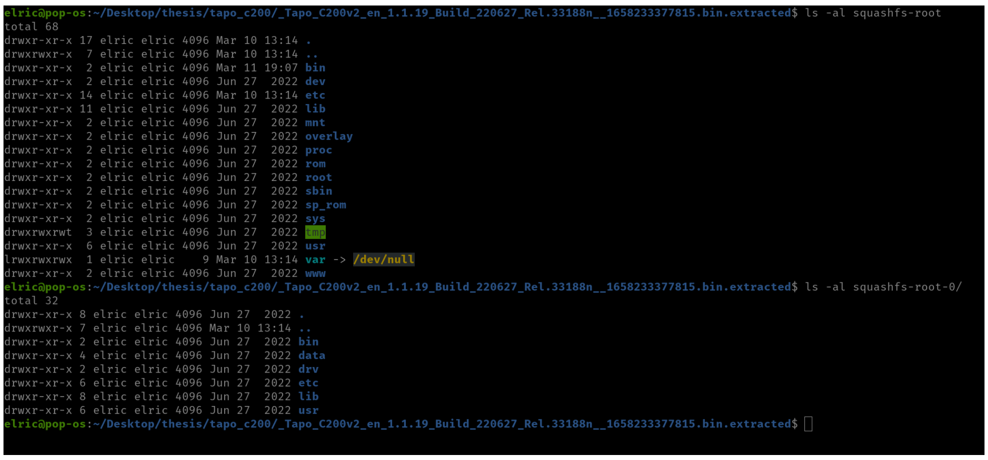
New firmware is packed as a binary file. It can be either firmware for the bare-metal processor, which is stored in memory or the instructions will be executed directly on the processor after reset. It can also be packed with firmware containing a filesystem. The UART interface shows that the camera has an entire Linux filesystem, it is worth exploring this option. Binwalk is a tool that can detect files and structures inside the binary file. It can help see what information the firmware contains.



```
elric@pop-os:~/Desktop/thesis/tapo_c200$ binwalk Tapo_C200v2_en_1.1.19_Build_220627_Rel.33188n__1658233377815.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
25088         0x6200          LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed
132096        0x20400         LZMA compressed data, properties: 0x6D, dictionary size: 8388608 bytes, uncompressed
512448        0x171400        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 2501156 bytes
063744        0x3E0200        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 3573634 bytes
8062720       0x7B0700        gzip compressed data, from Unix, last modified: 2022-06-27 01:13:17
8189004       0x7CF44C        gzip compressed data, from Unix, last modified: 2022-06-27 01:13:17
```

Figure 4.12: Parsing the content of binary file

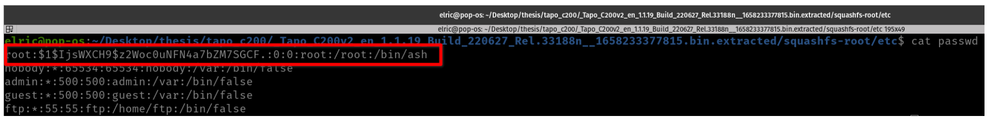
Binwalk successfully managed to identify the filesystems inside the binary. Firmware contains two squashfs filesystems. There are a few reasons why there could be two filesystems. A simple explanation could be to have two filesystems; one has read/write permissions, while the second is read-only. But the implementation details are unnecessary; what matters is the information they contain. Binwalk can extract the data from a binary file for further examination. The data are extracted to the local directory. Inside, there are squashfs-root and squashfs-root-0, as in 4.13.



```
elric@pop-os:~/Desktop/thesis/tapo_c200/_Tapo_C200v2_en_1.1.19_Build_220627_Rel.33188n__1658233377815.bin.extracted$ ls -al squashfs-root
total 68
drwxr-xr-x 17 elric elric 4096 Mar 10 13:14 .
drwxrwxr-x  7 elric elric 4096 Mar 10 13:14 ..
drwxr-xr-x  2 elric elric 4096 Mar 11 19:07 bin
drwxr-xr-x  2 elric elric 4096 Jun 27  2022 dev
drwxr-xr-x 14 elric elric 4096 Mar 10 13:14 etc
drwxr-xr-x 11 elric elric 4096 Jun 27  2022 lib
drwxr-xr-x  2 elric elric 4096 Jun 27  2022 mnt
drwxr-xr-x  2 elric elric 4096 Jun 27  2022 overlay
drwxr-xr-x  2 elric elric 4096 Jun 27  2022 proc
drwxr-xr-x  2 elric elric 4096 Jun 27  2022 rom
drwxr-xr-x  2 elric elric 4096 Jun 27  2022 root
drwxr-xr-x  2 elric elric 4096 Jun 27  2022 sbin
drwxr-xr-x  2 elric elric 4096 Jun 27  2022 sp_rom
drwxr-xr-x  2 elric elric 4096 Jun 27  2022 sys
drwxrwxrwt  3 elric elric 4096 Jun 27  2022 tmp
drwxr-xr-x  6 elric elric 4096 Jun 27  2022 usr
lrwxrwxrwx  1 elric elric    9 Mar 10 13:14 var -> /dev/null
drwxr-xr-x  2 elric elric 4096 Jun 27  2022 www
elric@pop-os:~/Desktop/thesis/tapo_c200/_Tapo_C200v2_en_1.1.19_Build_220627_Rel.33188n__1658233377815.bin.extracted$ ls -al squashfs-root-0/
total 32
drwxr-xr-x 8 elric elric 4096 Jun 27  2022 .
drwxrwxr-x 7 elric elric 4096 Mar 10 13:14 ..
drwxr-xr-x 2 elric elric 4096 Jun 27  2022 bin
drwxr-xr-x 4 elric elric 4096 Jun 27  2022 data
drwxr-xr-x 2 elric elric 4096 Jun 27  2022 drv
drwxr-xr-x 6 elric elric 4096 Jun 27  2022 etc
drwxr-xr-x 8 elric elric 4096 Jun 27  2022 lib
drwxr-xr-x 6 elric elric 4096 Jun 27  2022 usr
elric@pop-os:~/Desktop/thesis/tapo_c200/_Tapo_C200v2_en_1.1.19_Build_220627_Rel.33188n__1658233377815.bin.extracted$ 
```

Figure 4.13: Filesystems in the firmware

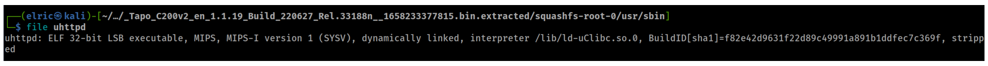
The first one contains more data and includes all standard Linux directories; it is safe to assume it is the primary filesystem. It contains *passwd* and *shadow* files, which contain root hash.



```
elric@pop-os: ~/Desktop/thesis/tapo_c200/_Tapo_C200v2_en_1.1.19_Build_220627_Rel.33188n__1658233377815.bin.extracted/squashfs-root/etc
elric@pop-os: ~/Desktop/thesis/tapo_c200/_Tapo_C200v2_en_1.1.19_Build_220627_Rel.33188n__1658233377815.bin.extracted/squashfs-root/etc 195x49
elric@pop-os:~/Desktop/thesis/tapo_c200/_Tapo_C200v2_en_1.1.19_Build_220627_Rel.33188n__1658233377815.bin.extracted/squashfs-root/etc$ cat passwd
root:$1$IjsWXCH9$z2Woc0uNFN4a7bZM7SGCF.:0:0:root:/root:/bin/ash
nobody:*:65534:65534:nobody:/var:/bin/false
admin:*:500:500:admin:/var:/bin/false
guest:*:500:500:guest:/var:/bin/false
ftp:*:55:55:ftp:/home/ftp:/bin/false
```

Figure 4.14: Hashes in passwd file

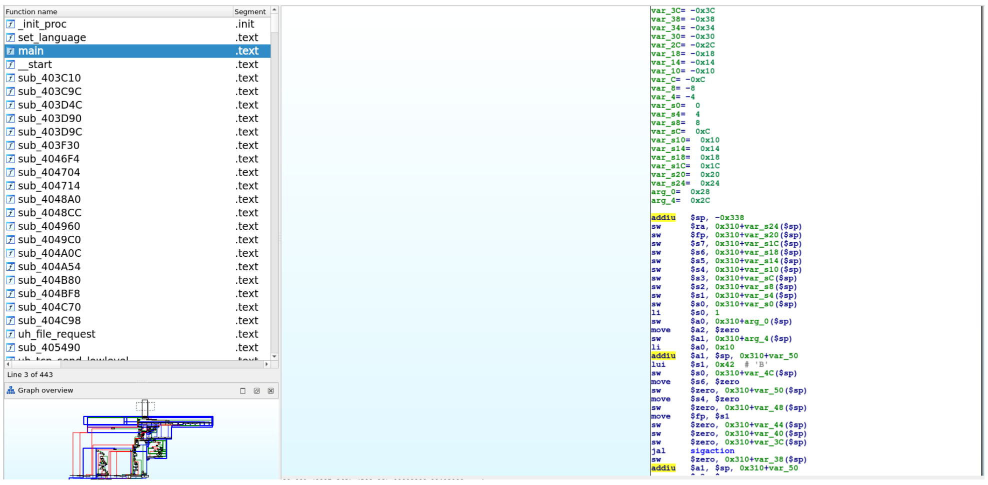
The credentials can allow accessing the UART shell. The next step is to look for intriguing binaries. Going through extracted firmware, a binary *uhttpd* located in squashfs-root-0 is a binary running the web server on the camera. The web server acts as the primary communication source between phone and camera functionalities. It is also the main input point for the user so that it can be used as an input point for possible software vulnerabilities.



```
┌──(elric㉿kali)-[~/…/_Tapo_C200v2_en_1.1.19_Build_220627_Rel.33188n__1658233377815.bin.extracted/squashfs-root-0/usr/sbin]
└─$ file uhttpd
uhttpd: ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0, BuildID[sha1]=f82e42d9631f22d89c49991a891b1ddfec7c369f, stripp
ed
```

Figure 4.15: Tapo C200 uhttpd binary

The binary is MIPS binary. MIPS is RISC architecture that is very simple and scalable [29]. The obvious step is to try to reverse engineer it and see what it does and how it works. IDA is a standard tool for reversing engineering; IDA Pro 8.2 is used in this analysis.



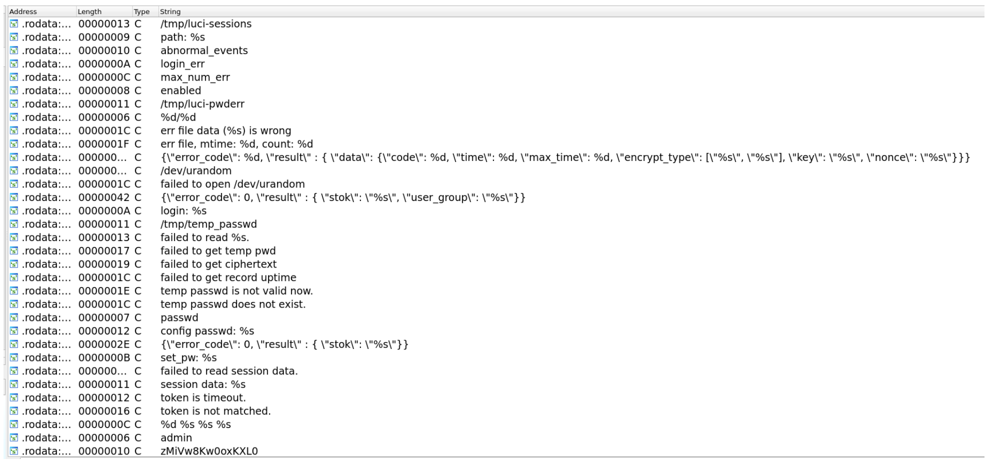
Function name	Segment
_init_proc	.init
set_language	.text
main	.text
__start	.text
sub_403C10	.text
sub_403C9C	.text
sub_403D4C	.text
sub_403D90	.text
sub_403D9C	.text
sub_403F30	.text
sub_4046F4	.text
sub_404704	.text
sub_404714	.text
sub_4048A0	.text
sub_4048CC	.text
sub_404960	.text
sub_4049C0	.text
sub_404A0C	.text
sub_404A54	.text
sub_404B80	.text
sub_404BF8	.text
sub_404C70	.text
sub_404C98	.text
uh_file_request	.text
sub_405490	.text

Line 3 of 443

```
var_3C= -0x3C
var_38= -0x38
var_34= -0x34
var_30= -0x30
var_2C= -0x2C
var_18= -0x18
var_14= -0x14
var_10= -0x10
var_C= -0xC
var_8= -8
var_4= -4
var_s0=  0
var_s4=  4
var_s8=  8
var_sC=  0xC
var_s10=  0x10
var_s14=  0x14
var_s18=  0x18
var_s1C=  0x1C
var_s20=  0x20
var_s24=  0x24
arg_0=  0x28
arg_4=  0x2C

addiu   $sp, -0x338
sw      $ra, 0x310+var_s24($sp)
sw      $fp, 0x310+var_s20($sp)
sw      $s7, 0x310+var_s1C($sp)
sw      $s6, 0x310+var_s18($sp)
sw      $s5, 0x310+var_s14($sp)
sw      $s4, 0x310+var_s10($sp)
sw      $s3, 0x310+var_sC($sp)
sw      $s2, 0x310+var_s8($sp)
sw      $s1, 0x310+var_s4($sp)
sw      $s0, 0x310+var_s0($sp)
li      $s0, 1
sw      $a0, 0x310+arg_0($sp)
move    $a2, $zero
sw      $a1, 0x310+arg_4($sp)
li      $a0, 0x10
addiu   $a1, $sp, 0x310+var_50
lui     $s1, 0x42  # 'B'
sw      $s0, 0x310+var_4C($sp)
move    $s6, $zero
sw      $zero, 0x310+var_50($sp)
move    $s4, $zero
sw      $zero, 0x310+var_48($sp)
move    $fp, $s1
sw      $zero, 0x310+var_44($sp)
sw      $zero, 0x310+var_40($sp)
sw      $zero, 0x310+var_3C($sp)
jal     sigaction
sw      $zero, 0x310+var_38($sp)
addiu   $a1, $sp, 0x310+var_50
```

Figure 4.16: Main function in uhttpd binary

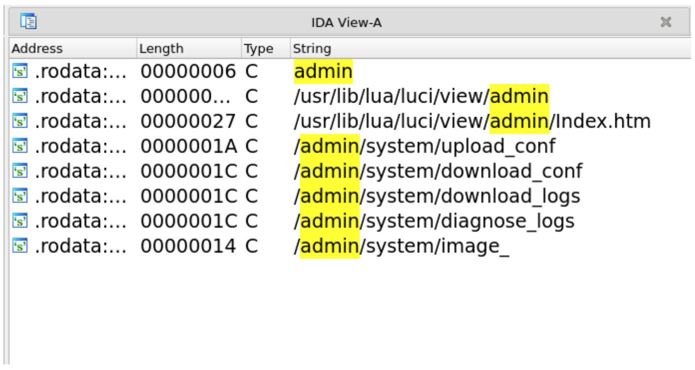
From a first look, many functions already have names, so the binary contains some debug information, which makes reverse engineering more accessible. The binary can use strings as printable characters, which can either help uncover some functionality or discover some helpful information. IDA can look for them inside the binary and print them out.



Address	Length	Type	String
.rodata:...	00000013	C	/tmp/luci-sessions
.rodata:...	00000009	C	path: %s
.rodata:...	00000010	C	abnormal_events
.rodata:...	0000000A	C	login_err
.rodata:...	0000000C	C	max_num_err
.rodata:...	00000008	C	enabled
.rodata:...	00000011	C	/tmp/luci-pwderr
.rodata:...	00000006	C	%d/%d
.rodata:...	0000001C	C	err file data (%s) is wrong
.rodata:...	0000001F	C	err file, mtime: %d, count: %d
.rodata:...	000000...	C	{\"error_code\": %d, \"result\" : { \"data\": {\"code\": %d, \"time\": %d, \"max_time\": %d, \"encrypt_type\": [\"%s\", \"%s\"], \"key\": \"%s\", \"nonce\": \"%s\"}}}
.rodata:...	000000...	C	/dev/urandom
.rodata:...	0000001C	C	failed to open /dev/urandom
.rodata:...	00000042	C	{\"error_code\": 0, \"result\" : { \"stok\": \"%s\", \"user_group\": \"%s\"}}
.rodata:...	0000000A	C	login: %s
.rodata:...	00000011	C	/tmp/temp_passwd
.rodata:...	00000013	C	failed to read %s.
.rodata:...	00000017	C	failed to get temp pwd
.rodata:...	00000019	C	failed to get ciphertext
.rodata:...	0000001C	C	failed to get record uptime
.rodata:...	0000001E	C	temp passwd is not valid now.
.rodata:...	0000001C	C	temp passwd does not exist.
.rodata:...	00000007	C	passwd
.rodata:...	00000012	C	config passwd: %s
.rodata:...	0000002E	C	{\"error_code\": 0, \"result\" : { \"stok\": \"%s\"}}
.rodata:...	0000000B	C	set_pw: %s
.rodata:...	000000...	C	failed to read session data.
.rodata:...	00000011	C	session data: %s
.rodata:...	00000012	C	token is timeout.
.rodata:...	00000016	C	token is not matched.
.rodata:...	0000000C	C	%d %s %s %s
.rodata:...	00000006	C	admin
.rodata:...	00000010	C	zMiVw8Kw0oxKXL0

Figure 4.17: Strings in uhttpd binary

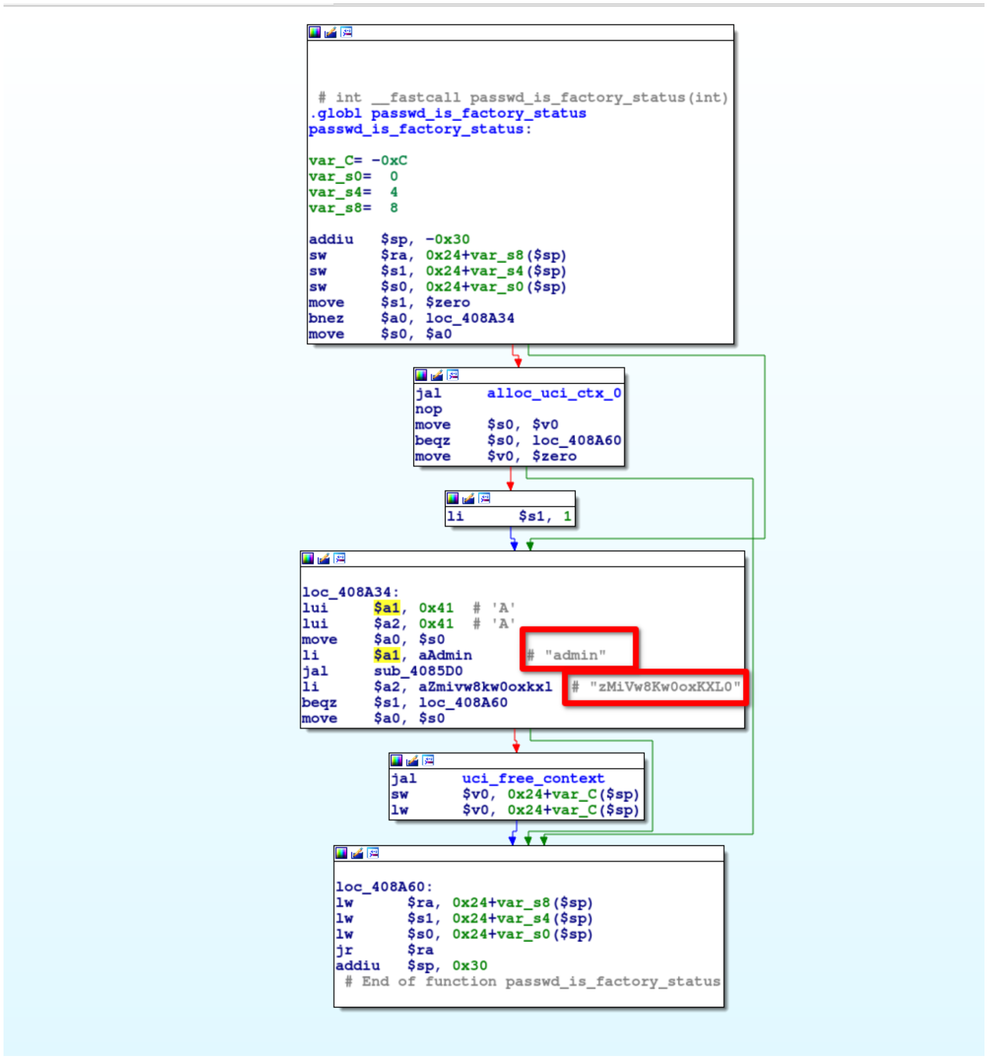
There are a lot of strings, so it makes sense to look only for interesting ones. One of the valuable strings can be some hardcoded credentials. Therefore, a common approach is to look for strings such as *admin* or *root*.



```
IDA View-A
Address       Length      Type  String
.rodata:...   00000006    C     admin
.rodata:...   000000...   C     /usr/lib/lua/luci/view/admin
.rodata:...   00000027    C     /usr/lib/lua/luci/view/admin/Index.htm
.rodata:...   0000001A    C     /admin/system/upload_conf
.rodata:...   0000001C    C     /admin/system/download_conf
.rodata:...   0000001C    C     /admin/system/download_logs
.rodata:...   0000001C    C     /admin/system/diagnose_logs
.rodata:...   00000014    C     /admin/system/image_
```

Figure 4.18: Admin strings in the binary

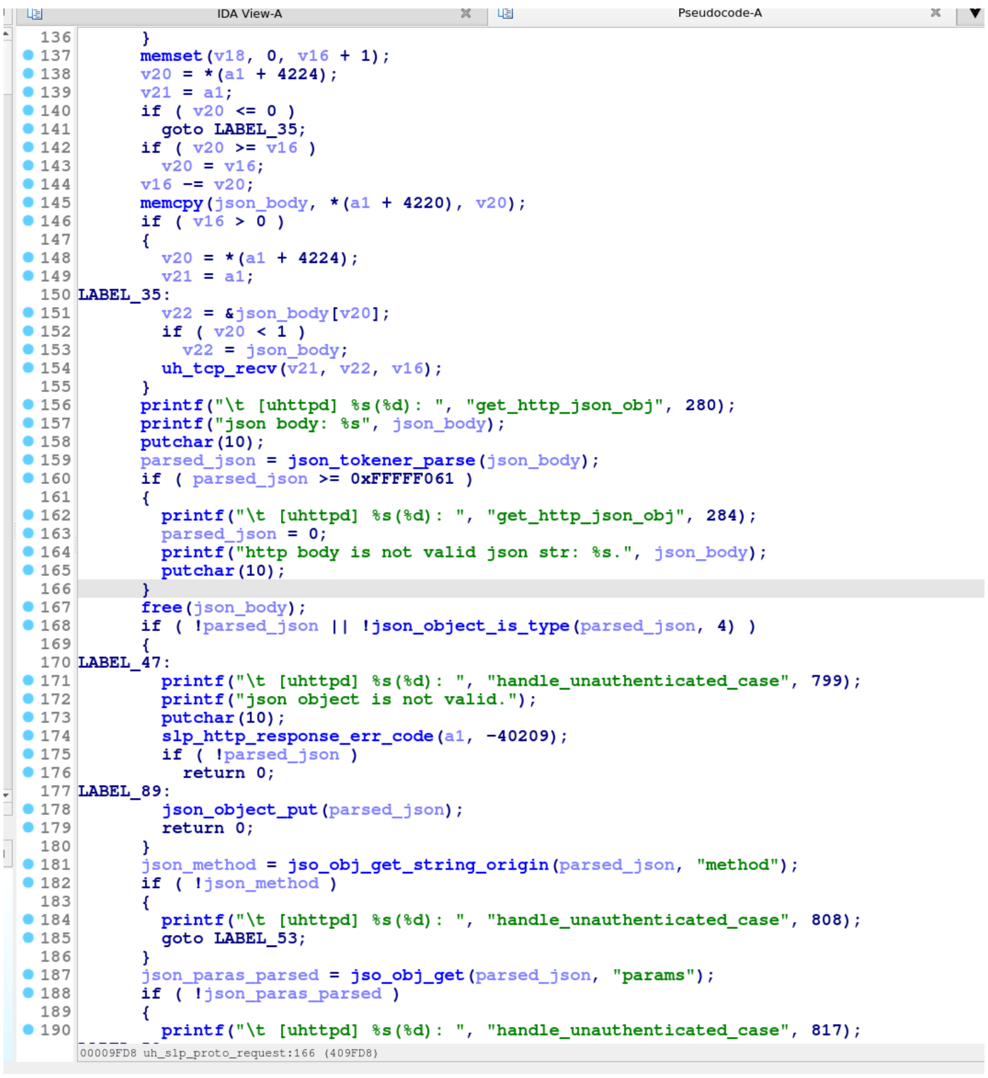
The first occurrence looks interesting, and after exploring it more, the following function references the string.



```
 # int __fastcall passwd_is_factory_status(int)
.globl passwd_is_factory_status
passwd_is_factory_status:

var_C= -0xC
var_s0=  0
var_s4=  4
var_s8=  8

addiu   $sp, -0x30
sw      $ra, 0x24+var_s8($sp)
sw      $s1, 0x24+var_s4($sp)
sw      $s0, 0x24+var_s0($sp)
move    $s1, $zero
bnez    $a0, loc_408A34
move    $s0, $a0

jal     alloc_uci_ctx_0
nop
move    $s0, $v0
beqz    $s0, loc_408A60
move    $v0, $zero

li      $s1, 1

loc_408A34:
lui     $a1, 0x41   # 'A'
lui     $a2, 0x41   # 'A'
move    $a0, $s0
li      $a1, aAdmin          # "admin"
jal     sub_4085D0
li      $a2, aZmivw8kw0oxkxl # "zMiVw8Kw0oxKXL0"
beqz    $s1, loc_408A60
move    $a0, $s0

jal     uci_free_context
sw      $v0, 0x24+var_C($sp)
lw      $v0, 0x24+var_C($sp)

loc_408A60:
lw      $ra, 0x24+var_s8($sp)
lw      $s1, 0x24+var_s4($sp)
lw      $s0, 0x24+var_s0($sp)
jr      $ra
addiu   $sp, 0x30
 # End of function passwd_is_factory_status
```

Figure 4.19: Factory reset function

After further examination, the credentials in the function are default credentials for the camera when it is unpaired. Another interesting function is *uh_slp_proto_request*. It is responsible for processing the requests coming from the user. IDA can decode instructions, and based on the architecture, it can create pseudocode for the binary.

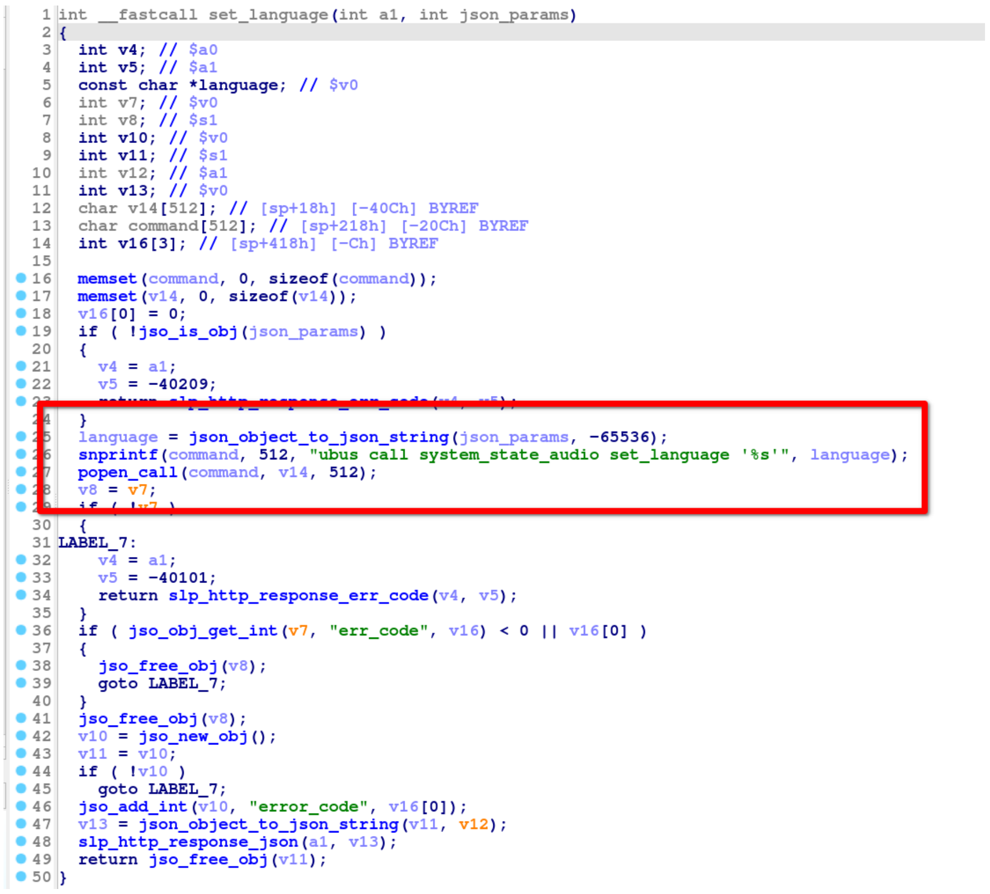


```
      }
      memset(v18, 0, v16 + 1);
      v20 = *(a1 + 4224);
      v21 = a1;
      if ( v20 <= 0 )
        goto LABEL_35;
      if ( v20 >= v16 )
        v20 = v16;
      v16 -= v20;
      memcpy(json_body, *(a1 + 4220), v20);
      if ( v16 > 0 )
      {
        v20 = *(a1 + 4224);
        v21 = a1;
LABEL_35:
        v22 = &json_body[v20];
        if ( v20 < 1 )
          v22 = json_body;
        uh_tcp_recv(v21, v22, v16);
      }
      printf("\t [uhttpd] %s(%d): ", "get_http_json_obj", 280);
      printf("json body: %s", json_body);
      putchar(10);
      parsed_json = json_tokener_parse(json_body);
      if ( parsed_json >= 0xFFFFF061 )
      {
        printf("\t [uhttpd] %s(%d): ", "get_http_json_obj", 284);
        parsed_json = 0;
        printf("http body is not valid json str: %s.", json_body);
        putchar(10);
      }
      free(json_body);
      if ( !parsed_json || !json_object_is_type(parsed_json, 4) )
      {
LABEL_47:
        printf("\t [uhttpd] %s(%d): ", "handle_unauthenticated_case", 799);
        printf("json object is not valid.");
        putchar(10);
        slp_http_response_err_code(a1, -40209);
        if ( !parsed_json )
          return 0;
LABEL_89:
        json_object_put(parsed_json);
        return 0;
      }
      json_method = jso_obj_get_string_origin(parsed_json, "method");
      if ( !json_method )
      {
        printf("\t [uhttpd] %s(%d): ", "handle_unauthenticated_case", 808);
        goto LABEL_53;
      }
      json_paras_parsed = jso_obj_get(parsed_json, "params");
      if ( !json_paras_parsed )
      {
        printf("\t [uhttpd] %s(%d): ", "handle_unauthenticated_case", 817);
```

```
00009FD8 uh_slp_proto_request:166 (409FD8)
```

Figure 4.20: Parsing request data in JSON

The camera and the phone send the data in the form of JavaScript Object Notation (JSON). JSON data in requests have two primary keys - method and params. The former defines an action to perform; the latter contains parameters for the activity. Supported actions include login or setLanguage. A more interesting method is setLanguage. When the camera receives a request with a method set to string setLanguage, it will look for a parameter with name language. If the parameter is present, it will call the function set_language. The reason why the function is interesting is the calling of popen. Popen is a function that will run a system command supplied as a string. It is worth exploring because it might lead to code execution.



```
int __fastcall set_language(int a1, int json_params)
{
  int v4; // $a0
  int v5; // $a1
  const char *language; // $v0
  int v7; // $v0
  int v8; // $s1
  int v10; // $v0
  int v11; // $s1
  int v12; // $a1
  int v13; // $v0
  char v14[512]; // [sp+18h] [-40Ch] BYREF
  char command[512]; // [sp+218h] [-20Ch] BYREF
  int v16[3]; // [sp+418h] [-Ch] BYREF

  memset(command, 0, sizeof(command));
  memset(v14, 0, sizeof(v14));
  v16[0] = 0;
  if ( !jso_is_obj(json_params) )
  {
    v4 = a1;
    v5 = -40209;
    return slp_http_response_err_code(v4, v5);
  }
  language = json_object_to_json_string(json_params, -65536);
  snprintf(command, 512, "ubus call system_state_audio set_language '%s'", language);
  popen_call(command, v14, 512);
  v8 = v7;
  if ( !v7 )
  {
LABEL_7:
    v4 = a1;
    v5 = -40101;
    return slp_http_response_err_code(v4, v5);
  }
  if ( jso_obj_get_int(v7, "err_code", v16) < 0 || v16[0] )
  {
    jso_free_obj(v8);
    goto LABEL_7;
  }
  jso_free_obj(v8);
  v10 = jso_new_obj();
  v11 = v10;
  if ( !v10 )
    goto LABEL_7;
  jso_add_int(v10, "error_code", v16[0]);
  v13 = json_object_to_json_string(v11, v12);
  slp_http_response_json(a1, v13);
  return jso_free_obj(v11);
}
```

Figure 4.21: set_language function

As shown in 4.21, the language parameter is converted to the string and used as a parameter in the command. The language is placed between single quotes, which should prevent simple code injection. The only way to gain command execution is to insert a single quote inside the parameter and escape the containment of single quotes. However, before calling the set_language function, the code checks whether the language parameter contains a single quote with the function strchr. A quote can be encoded as 39 in the decimal number system, and strchr looks for that number inside the parameter. The control with strchr prevents an adversary from injecting the commands and gaining command execution.

From UDP scan, an attractive port is port 69, also a trivial FTP server. It is a much simpler version of FTP, which only allows data download and upload. After trying to connect to the port, the connection is successful.

4.1.1.4 Reporting

The identified vulnerabilties in Tapo C200 are summarized in the following table:

Vulnerability	**CVSS**	**Mitigation**
UART shell with weak password	5.4	Disable UART shell and set password to the cryptographically strong one
Insecure firmware download	3.2	Secure firmware download through mutual TLS
Filesystem with critical files	3.2	Encryption of firmware
TFTP server	Informational	Disable unnecessary service

Table 4.1: Table of vulnerabities in Tapo C200

4.1.2 TP-Link Tapo C320WS

Tapo C320WS is an outdoor security camera with night vision up to 30 meters. It has a resolution of 2560x1440 pixels and supports Google Assistant and Amazon Alexa.

Figure 4.22: TAPO C320WS [25]

4.1.2.1 Information gathering

The first task is to disassemble the camera. It has no unique defense mechanism against disassembling, so it is relatively easy to access the PCBs.

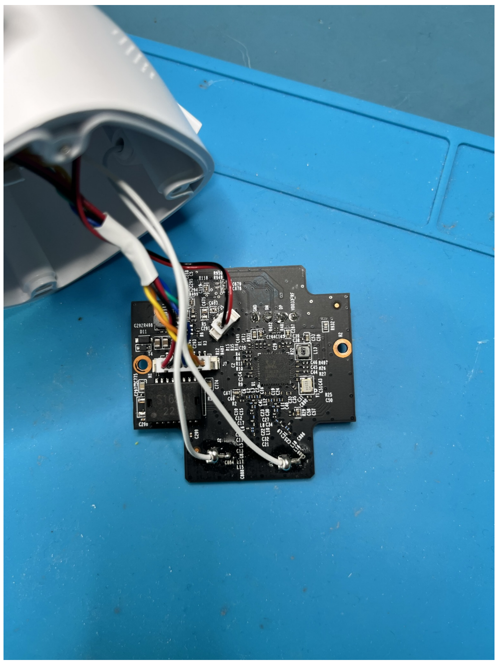


Figure 4.23: First PCB

There are two separate PCBs. The camera contains two main chips - RTL8192EU and STAR SSC337. The former is a single-chip integrating LWAn and network USB interface, the latter ARM-based System-on-chip (SoC) processing image from the camera. The only labels describing functionality are D- and D+ on one PCB and DN and DP on the other. These are USB ports. There are, however, additional pins that could have more functionality. The first task is to find ground pins. Ground pins can be identified with a voltmeter using resistance mode.

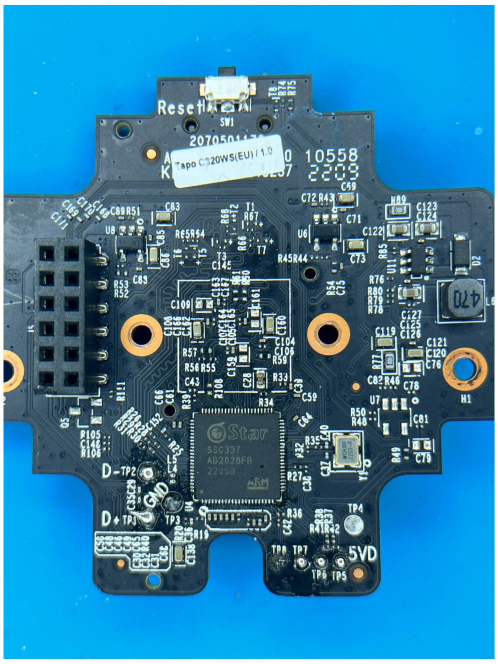


Figure 4.24: Second PCB

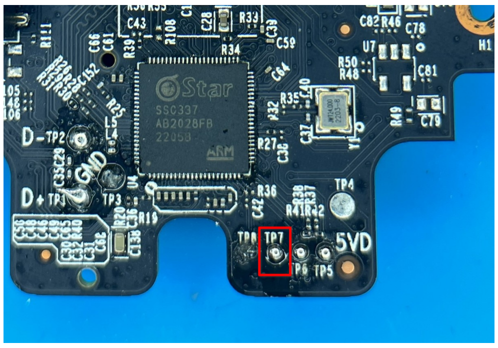


Figure 4.25: Ground pin

There are three additional pins close to the ground pin. The next step is to measure the voltage on the other pins and identify possible protocols. All pins have around 3.3 volts, which could mean UART based on the previous research in Tapo C200. A logic analyzer can definitely identify whether the suspected ports are actually UART. Connecting ports to the Salea logic analyzer will result in the output as shown in 4.26.

From the behavior of the signal, it looks like UART. However, decoding requires knowledge of the baud rate. The shortest interval of logic one is 8.8 microseconds. Therefore, the inverse value is around 113636 bits per second.

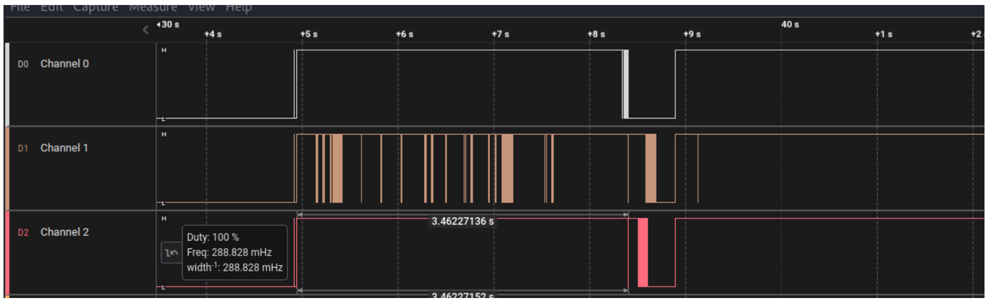
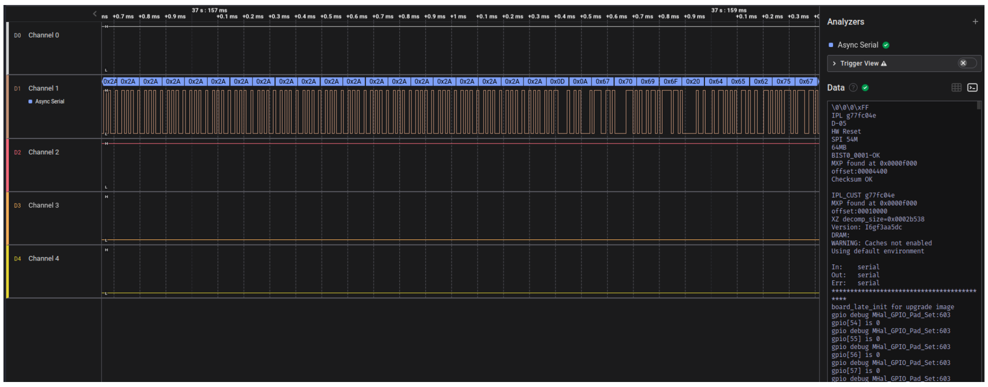


Figure 4.26: Digital signal decoding

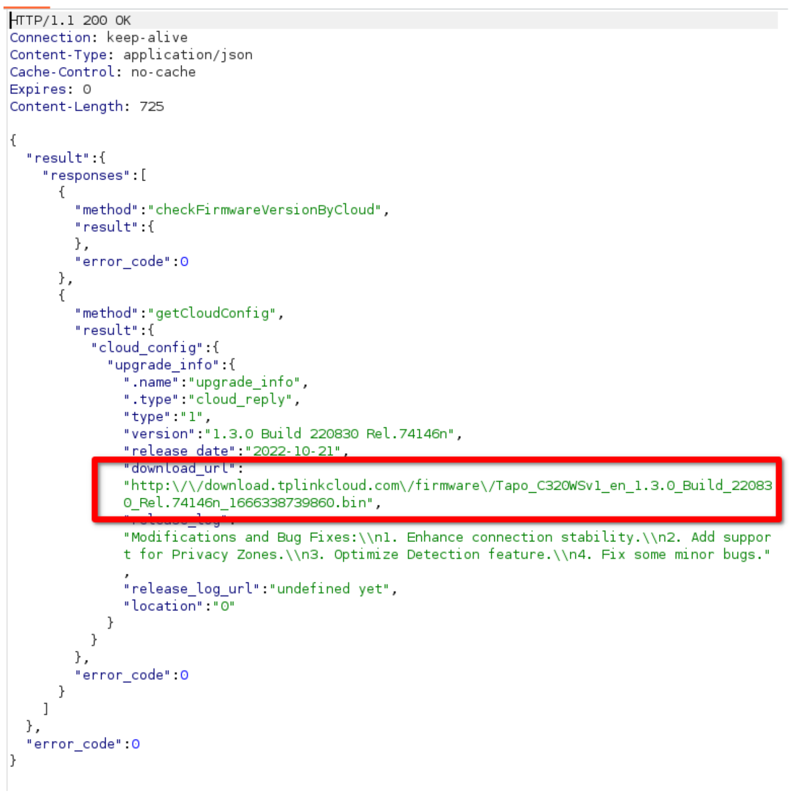
The value is very close to 115200, which is one of the most common baud rates for UART. Decoding the communication with the baud rate set to 115200 will allow seeing transferred ASCII characters.



```
\0\0\0\xFF
IPL g77fc04e
D-05
HW Reset
SPI 54M
64MB
BIST0_0001-OK
MXP found at 0x0000f000
offset:00004400
Checksum OK

IPL_CUST g77fc04e
MXP found at 0x0000f000
offset:00010000
XZ decomp_size=0x0002b538
Version: I6gf3aa5dc
DRAM:
WARNING: Caches not enabled
Using default environment

In:    serial
Out:   serial
Err:   serial
***************************************
****
board_late_init for upgrade image
gpio debug MHal_GPIO_Pad_Set:603
gpio[54] is 0
gpio debug MHal_GPIO_Pad_Set:603
gpio[55] is 0
gpio debug MHal_GPIO_Pad_Set:603
gpio[56] is 0
gpio debug MHal_GPIO_Pad_Set:603
gpio[57] is 0
gpio debug MHal_GPIO_Pad_Set:603
```

Figure 4.27: UART decoding

As with the previous camera, a new update was released during the research. Intercepting the update and downloading the firmware file was still possible.



```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Cache-Control: no-cache
Expires: 0
Content-Length: 725

{
  "result":{
    "responses":[
      {
        "method":"checkFirmwareVersionByCloud",
        "result":{
        },
        "error_code":0
      },
      {
        "method":"getCloudConfig",
        "result":{
          "cloud_config":{
            "upgrade_info":{
              ".name":"upgrade_info",
              ".type":"cloud_reply",
              "type":"1",
              "version":"1.3.0 Build 220830 Rel.74146n",
              "release_date":"2022-10-21",
              "download_url":
              "http:\/\/download.tplinkcloud.com\/firmware\/Tapo_C320WSv1_en_1.3.0_Build_22083
              0_Rel.74146n_1666338739860.bin",
              "release_log":
              "Modifications and Bug Fixes:\\n1. Enhance connection stability.\\n2. Add suppor
              t for Privacy Zones.\\n3. Optimize Detection feature.\\n4. Fix some minor bugs."
              ,
              "release_log_url":"undefined yet",
              "location":"0"
            }
          }
        },
        "error_code":0
      }
    ]
  },
  "error_code":0
}
```

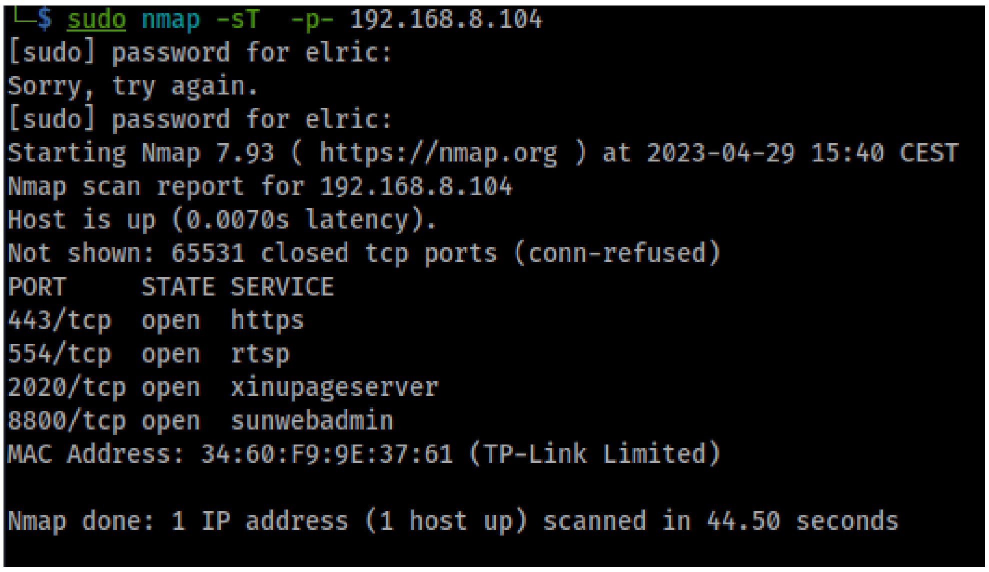
Figure 4.28: Firmware update in JSON

The camera is running HTTPS server to communicate with the user, but it is not the only service running on the camera, and it is worth exploring what other services it offers. For port scanning, the Nmap command can be used.

The important port is 443 and 554. Port 2020 just redirects everything to port 443, and port 8800 does not respond to HTTP or HTTPS requests. The port 554 Real-time Streaming Protocol (RSTP), which transfers camera video stream to the phone. However, the video streams are not accessible without credentials.

4.1.2.2 Threat modeling

Tapo C320WS has very similar functionality from the threat perspective as the Tapo C200. With disclosed firmware, an adversary can reverse engineer it and discover software vulnerabilities, or as seen in the previous section, can read the whole filesystem with critical files like *passwd*. A UART interface is also available, which can give access to the shell and allow it to read memory or write to the memory.



```
└─$ sudo nmap -sT  -p- 192.168.8.104
[sudo] password for elric:
Sorry, try again.
[sudo] password for elric:
Starting Nmap 7.93 ( https://nmap.org ) at 2023-04-29 15:40 CEST
Nmap scan report for 192.168.8.104
Host is up (0.0070s latency).
Not shown: 65531 closed tcp ports (conn-refused)
PORT     STATE SERVICE
443/tcp  open  https
554/tcp  open  rtsp
2020/tcp open  xinupageserver
8800/tcp open  sunwebadmin
MAC Address: 34:60:F9:9E:37:61 (TP-Link Limited)

Nmap done: 1 IP address (1 host up) scanned in 44.50 seconds
```

Figure 4.29: TCP scan

4.1.2.3 Vulnerability analysis and exploitation

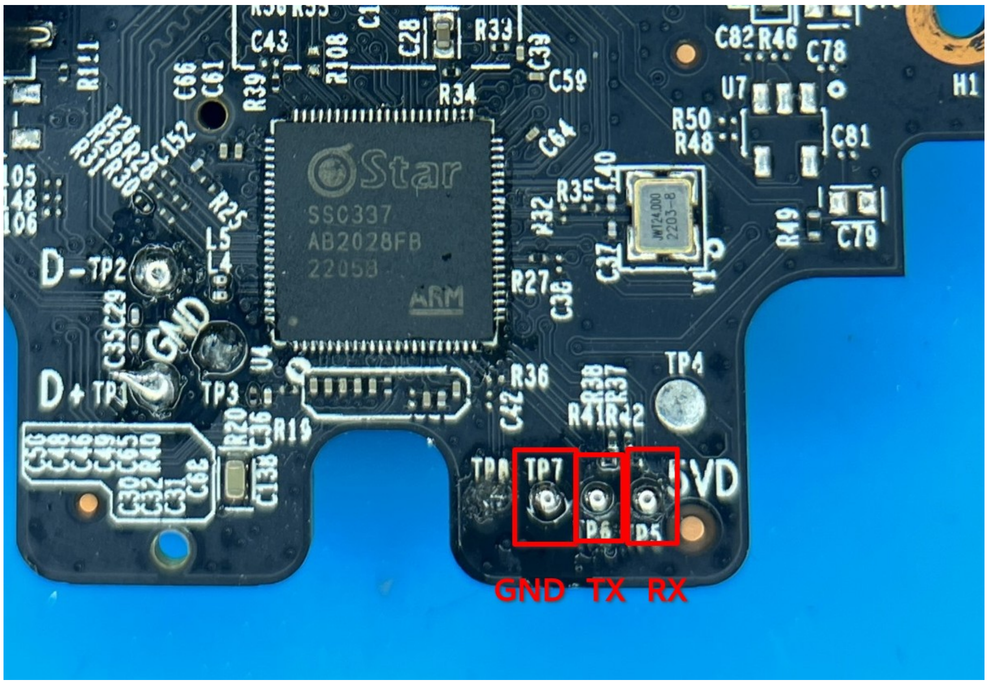
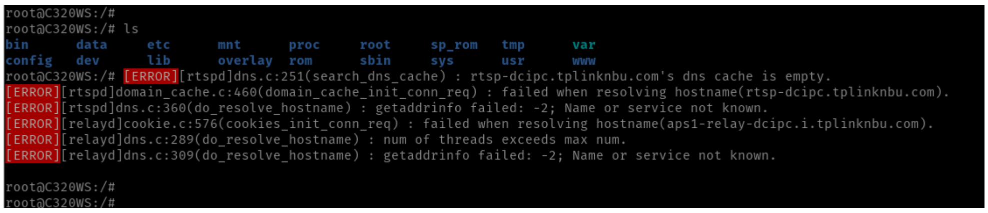


Figure 4.30: UART interface

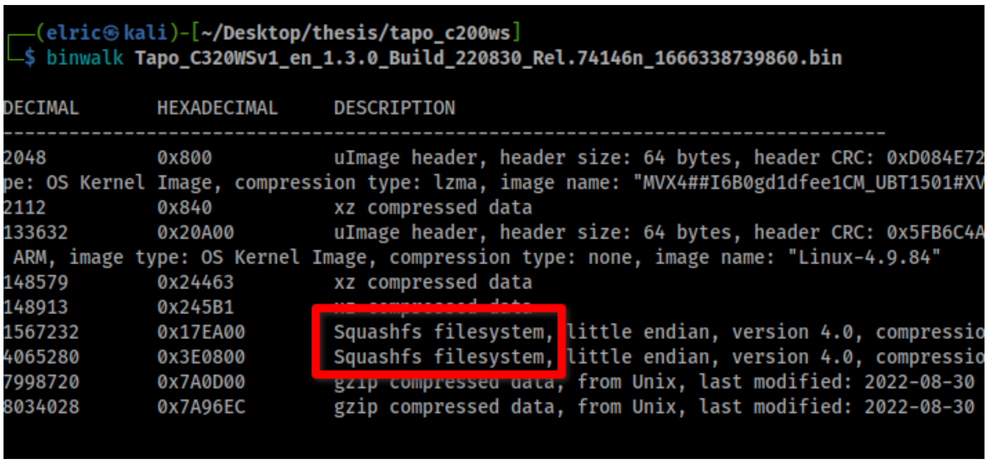
To communicate with UART, a TTL-to-USB adapter is used. After connecting the adapter to the computer, a root shell is prompted without a password.



```
root@C320WS:/#
root@C320WS:/# ls
bin      data     etc      mnt      proc     root     sp_rom   tmp      var
config   dev      lib      overlay  rom      sbin     sys      usr      www
root@C320WS:/# [ERROR][rtspd]dns.c:251(search_dns_cache) : rtsp-dcipc.tplinknbu.com's dns cache is empty.
[ERROR][rtspd]domain_cache.c:460(domain_cache_init_conn_req) : failed when resolving hostname(rtsp-dcipc.tplinknbu.com).
[ERROR][rtspd]dns.c:360(do_resolve_hostname) : getaddrinfo failed: -2; Name or service not known.
[ERROR][relayd]cookie.c:576(cookies_init_conn_req) : failed when resolving hostname(aps1-relay-dcipc.i.tplinknbu.com).
[ERROR][relayd]dns.c:289(do_resolve_hostname) : num of threads exceeds max num.
[ERROR][relayd]dns.c:309(do_resolve_hostname) : getaddrinfo failed: -2; Name or service not known.

root@C320WS:/#
root@C320WS:/#
```

Figure 4.31: UART shell

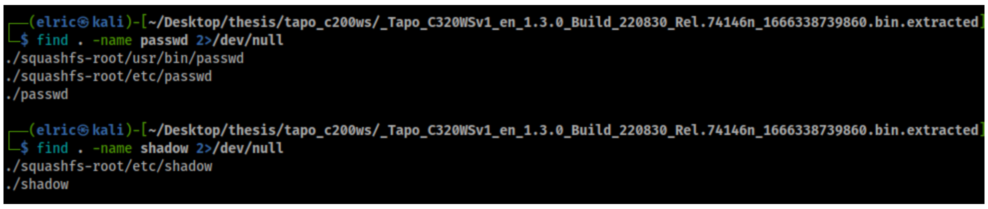
The firmware update is he binary file that can contain a filesystem, which might be similar to the previous camera. Binwalk can help to analyze the file further. The firmware contains two filesystems, which can uncover some valuable information. They can be extracted through binwalk.



```
┌──(elric㉿kali)-[~/Desktop/thesis/tapo_c200ws]
└─$ binwalk Tapo_C320WSv1_en_1.3.0_Build_220830_Rel.74146n_1666338739860.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
2048          0x800           uImage header, header size: 64 bytes, header CRC: 0xD084E72
pe: OS Kernel Image, compression type: lzma, image name: "MVX4##I6B0gd1dfee1CM_UBT1501#XV
2112          0x840           xz compressed data
133632        0x20A00         uImage header, header size: 64 bytes, header CRC: 0x5FB6C4A
 ARM, image type: OS Kernel Image, compression type: none, image name: "Linux-4.9.84"
148579        0x24463         xz compressed data
148913        0x245B1         xz compressed data
1567232       0x17EA00        Squashfs filesystem, little endian, version 4.0, compressio
4065280       0x3E0800        Squashfs filesystem, little endian, version 4.0, compressio
7998720       0x7A0D00        gzip compressed data, from Unix, last modified: 2022-08-30
8034028       0x7A96EC        gzip compressed data, from Unix, last modified: 2022-08-30
```

Figure 4.32: Binwalk analysis

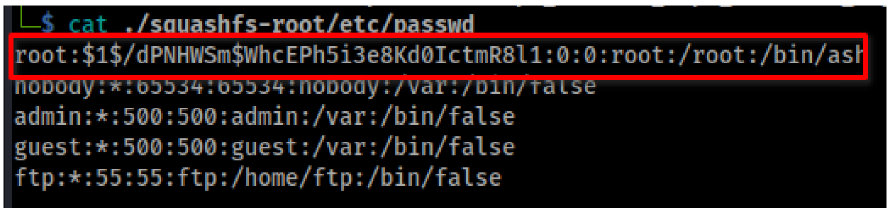
There are two filesystems - squashfs-root and squashfs-root-0. As both are Linux filesystems, they might contain *passwd* or *shadow* files.



```
┌──(elric㉿kali)-[~/Desktop/thesis/tapo_c200ws/_Tapo_C320WSv1_en_1.3.0_Build_220830_Rel.74146n_1666338739860.bin.extracted]
└─$ find . -name passwd 2>/dev/null
./squashfs-root/usr/bin/passwd
./squashfs-root/etc/passwd
./passwd

┌──(elric㉿kali)-[~/Desktop/thesis/tapo_c200ws/_Tapo_C320WSv1_en_1.3.0_Build_220830_Rel.74146n_1666338739860.bin.extracted]
└─$ find . -name shadow 2>/dev/null
./squashfs-root/etc/shadow
./shadow
```

Figure 4.33: Filesystems in firmware

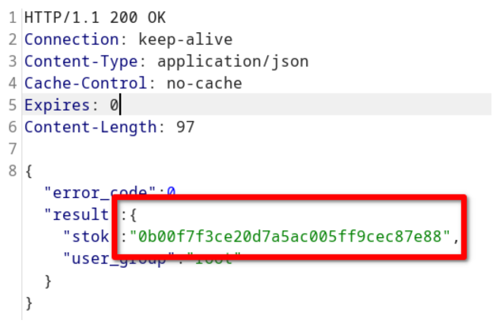
Filesystem contains a *passwd* file, which does hold the hash of the root.



```
└─$ cat ./squashfs-root/etc/passwd
root:$1$/dPNHWSm$WhcEPh5i3e8Kd0IctmR8l1:0:0:root:/root:/bin/ash
nobody:*:65534:65534:nobody:/var:/bin/false
admin:*:500:500:admin:/var:/bin/false
guest:*:500:500:guest:/var:/bin/false
ftp:*:55:55:ftp:/home/ftp:/bin/false
```

Figure 4.34: Hashes in passwd file

Similar to the previous camera, the main point for communication is the HTTPS server running on port 443. The binary for the service is located at squashfs-root/usr/sbin/uhttpd. The structure of the HTTPS server is very similar to the Tapo C200. The user authenticates through JSON request with his username and password. The server returns a response with a token named stok.



```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Cache-Control: no-cache
Expires: 0
Content-Length: 97

{
  "error_code":0,
  "result":{
    "stok":"0b00f7f3ce20d7a5ac005ff9cec87e88",
    "user_group":"root"
  }
}
```

Figure 4.35: Authentication token

To receive further data from the camera, the user sends a POST request to URL /stok=/ds, where stok is set to a token received in the login response. The available methods are the same as in Tapo C200.

4.1.2.4 Reporting

The following table includes the vulnerabilities in Tapo C320WS:

Vulnerability	**CVSS**	**Mitigation**
UART shell without password	6.4	Set the password for UART or disable it entirely
Insecure firmware download	3.2	Secure firmware download through mutual TLS
Filesystem with critical files	3.2	Encryption of firmware

Table 4.2: Table of vulnerabities in Tapo C320WS

4.2 Smart Gateways

This section describes the analysis of the smart gateway Tesla Zigbee Hub.

4.2.1 Tesla Zigbee Hub

Tesla Zigbee Hub is a control gateway for the smart home ecosystem, allowing control through Zigbee. It communicates through the Tuya network and can be controlled through Android and iOS phones.

4.2.1.1 Information gathering

After disassembling, the hub consists of one PCB, as in 4.36.

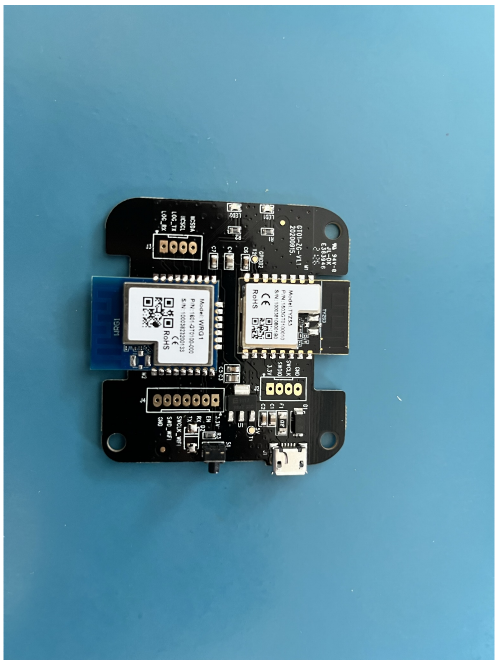


Figure 4.36: Printed circuit board

It contains two main modules - the first is labeled as WRG1, and the second is labeled as TYZS3. WRG1 is a Tuya Wi-Fi module, and TYZS3 is an embedded Zigbee module made by Tuya. Searching these names in searching engine will result in a datasheet for both modules. The main chips on both modules are covered with metal foil, which can be removed.

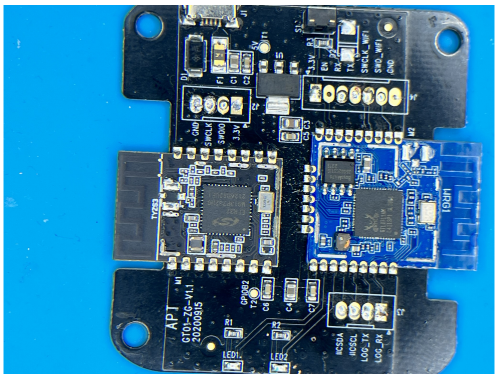


Figure 4.37: Uncovered modules

After foil removal, the main chips for both modules can be identified. The WRG1 contains RTL8711AM, an ARM-based low-powered single chip for WLAN communication. The TYZS3 module has an EFR32 Multi-Gecko System-on-chip (SoC) capable of radio communication. The device contains labeled hardware pins, which makes analysis easier.

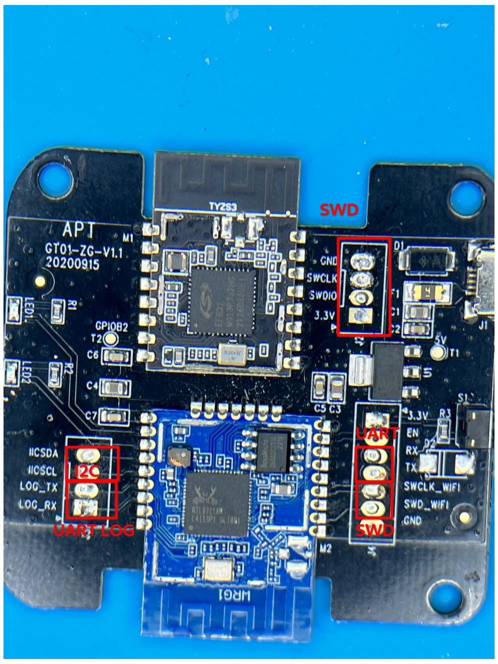
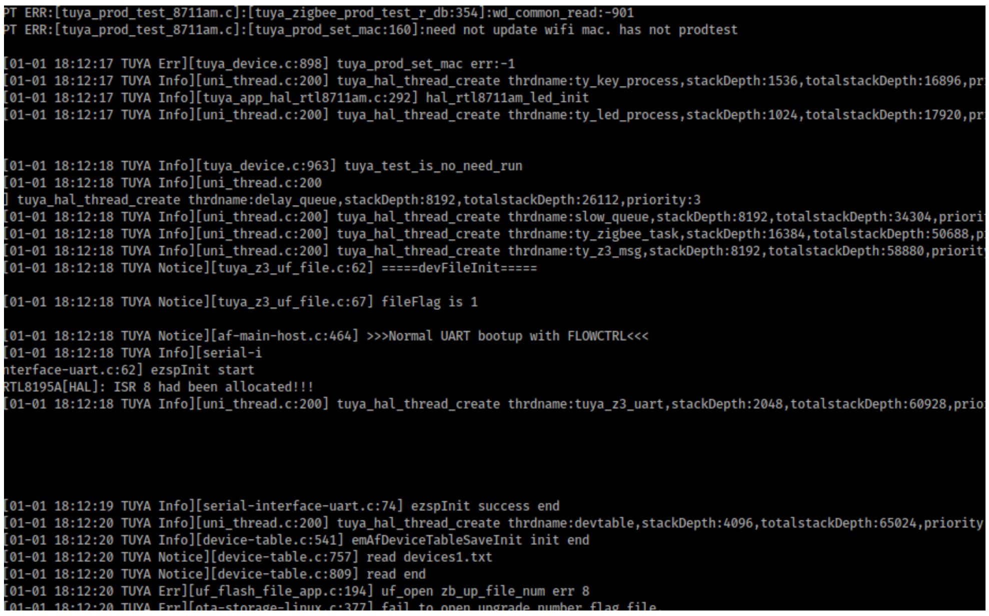


Figure 4.38: Marked interfaces

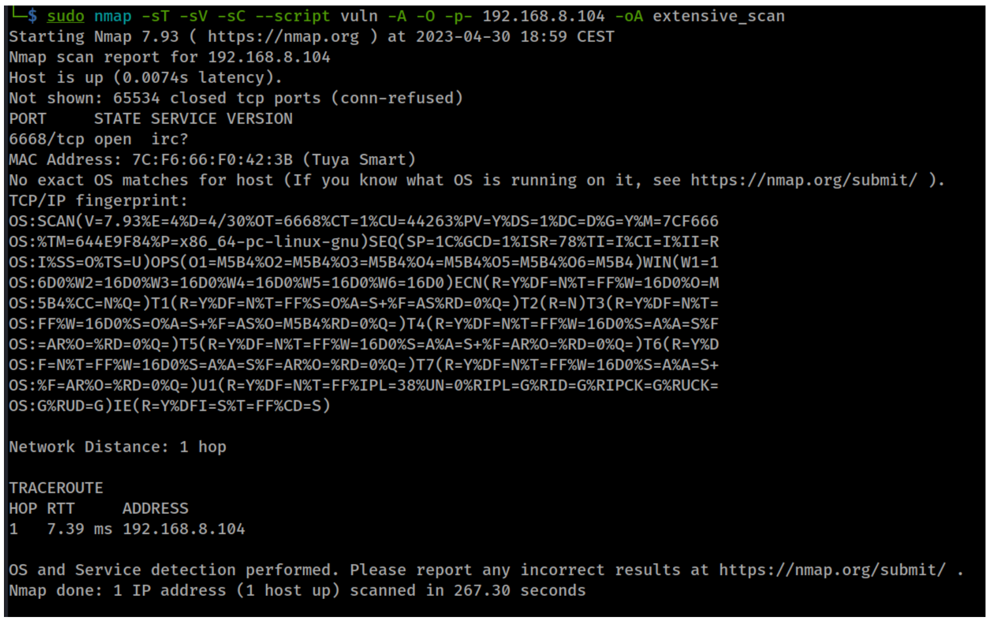
WRG1 contains more interfaces: *UART for logging*, *Serial-Wire Debug (SWD)*, *I2C*, and *UART*. TYZS3 contains only SWD. Of mentioned interfaces, the most interesting are UART and SWD. SWD allows to control CPU and directly debug instructions that are being executed. The UART for logging has RX pin disabled, meaning it only prints out system information, which can be helpful for further knowledge.

The part of configuration is connecting the hub to local Wi-Fi. Therefore, network scan is the next step. The first task is to determine the IP address. One of the options is to scan the entire network, where the gateway is connected, and identify which IP address belongs to it. The scan shows that IP address 192.168.8.104 belongs to the gateway. The next step is to scan all the ports and identify which could lead to further access.



```
PT ERR:[tuya_prod_test_8711am.c]:[tuya_zigbee_prod_test_r_db:354]:wd_common_read:-901
PT ERR:[tuya_prod_test_8711am.c]:[tuya_prod_set_mac:160]:need not update wifi mac. has not prodtest

[01-01 18:12:17 TUYA Err][tuya_device.c:898] tuya_prod_set_mac err:-1
[01-01 18:12:17 TUYA Info][uni_thread.c:200] tuya_hal_thread_create thrdname:ty_key_process,stackDepth:1536,totalstackDepth:16896,pr
[01-01 18:12:17 TUYA Info][tuya_app_hal_rtl8711am.c:292] hal_rtl8711am_led_init
[01-01 18:12:17 TUYA Info][uni_thread.c:200] tuya_hal_thread_create thrdname:ty_led_process,stackDepth:1024,totalstackDepth:17920,pr


[01-01 18:12:18 TUYA Info][tuya_device.c:963] tuya_test_is_no_need_run
[01-01 18:12:18 TUYA Info][uni_thread.c:200
] tuya_hal_thread_create thrdname:delay_queue,stackDepth:8192,totalstackDepth:26112,priority:3
[01-01 18:12:18 TUYA Info][uni_thread.c:200] tuya_hal_thread_create thrdname:slow_queue,stackDepth:8192,totalstackDepth:34304,priori
[01-01 18:12:18 TUYA Info][uni_thread.c:200] tuya_hal_thread_create thrdname:ty_zigbee_task,stackDepth:16384,totalstackDepth:50688,p
[01-01 18:12:18 TUYA Info][uni_thread.c:200] tuya_hal_thread_create thrdname:ty_z3_msg,stackDepth:8192,totalstackDepth:58880,priorit
[01-01 18:12:18 TUYA Notice][tuya_z3_uf_file.c:62] =====devFileInit=====

[01-01 18:12:18 TUYA Notice][tuya_z3_uf_file.c:67] fileFlag is 1

[01-01 18:12:18 TUYA Notice][af-main-host.c:464] >>>Normal UART bootup with FLOWCTRL<<<
[01-01 18:12:18 TUYA Info][serial-i
nterface-uart.c:62] ezspInit start
RTL8195A[HAL]: ISR 8 had been allocated!!!
[01-01 18:12:18 TUYA Info][uni_thread.c:200] tuya_hal_thread_create thrdname:tuya_z3_uart,stackDepth:2048,totalstackDepth:60928,prio





[01-01 18:12:19 TUYA Info][serial-interface-uart.c:74] ezspInit success end
[01-01 18:12:20 TUYA Info][uni_thread.c:200] tuya_hal_thread_create thrdname:devtable,stackDepth:4096,totalstackDepth:65024,priority
[01-01 18:12:20 TUYA Info][device-table.c:541] emAfDeviceTableSaveInit init end
[01-01 18:12:20 TUYA Notice][device-table.c:757] read devices1.txt
[01-01 18:12:20 TUYA Notice][device-table.c:809] read end
[01-01 18:12:20 TUYA Err][uf_flash_file_app.c:194] uf_open zb_up_file_num err 8
[01-01 18:12:20 TUYA Err][ota-storage-linux.c:377] fail to open upgrade number flag file.
```

Figure 4.39: UART interface



```
└─$ sudo nmap -sT -sV -sC --script vuln -A -O -p- 192.168.8.104 -oA extensive_scan
Starting Nmap 7.93 ( https://nmap.org ) at 2023-04-30 18:59 CEST
Nmap scan report for 192.168.8.104
Host is up (0.0074s latency).
Not shown: 65534 closed tcp ports (conn-refused)
PORT     STATE SERVICE VERSION
6668/tcp open  irc?
MAC Address: 7C:F6:66:F0:42:3B (Tuya Smart)
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.93%E=4%D=4/30%OT=6668%CT=1%CU=44263%PV=Y%DS=1%DC=D%G=Y%M=7CF666
OS:%TM=644E9F84%P=x86_64-pc-linux-gnu)SEQ(SP=1C%GCD=1%ISR=78%TI=I%CI=I%II=R
OS:I%SS=O%TS=U)OPS(O1=M5B4%O2=M5B4%O3=M5B4%O4=M5B4%O5=M5B4%O6=M5B4)WIN(W1=1
OS:6D0%W2=16D0%W3=16D0%W4=16D0%W5=16D0%W6=16D0)ECN(R=Y%DF=N%T=FF%W=16D0%O=M
OS:5B4%CC=N%Q=)T1(R=Y%DF=N%T=FF%S=O%A=S+%F=AS%RD=0%Q=)T2(R=N)T3(R=Y%DF=N%T=
OS:FF%W=16D0%S=O%A=S+%F=AS%O=M5B4%RD=0%Q=)T4(R=Y%DF=N%T=FF%W=16D0%S=A%A=S%F
OS:=AR%O=%RD=0%Q=)T5(R=Y%DF=N%T=FF%W=16D0%S=A%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%D
OS:F=N%T=FF%W=16D0%S=A%A=S%F=AR%O=%RD=0%Q=)T7(R=Y%DF=N%T=FF%W=16D0%S=A%A=S+
OS:%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=FF%IPL=38%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=
OS:G%RUD=G)IE(R=Y%DFI=S%T=FF%CD=S)

Network Distance: 1 hop

TRACEROUTE
HOP RTT     ADDRESS
1   7.39 ms 192.168.8.104

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 267.30 seconds
```

Figure 4.40: TCP scan

From TCP ports, the only opened port is *6668*. The port does not contain any obvious functionality and does not support an HTTP connection. Examining network traffic from the phone, it also does not connect to that port during communication. The port may be used for gateway communication with the cloud. However, as the cloud is outside the scope of the analysis, this port will not be examined further. The next step is to examine UDP ports. Upon inspecting them, all of the UDP ports are closed.

4.2.1.2 Threat modeling

The PCB contains few hardware interfaces, which can lead to vulnerabilities. An adversary can have a shell if the UART is enabled, as with smart cameras. Another interface is SWD, which can control CPU, read, or write memory. An adversary can exploit the control over the instructions or read sensitive parts of memory. Furthermore, the services running on the hub can be vulnerable or misconfigured.

4.2.1.3 Vulnerability analysis and exploitation

One of the identified interfaces in analysis was SWD. The SWD port controls the processor in real-time, giving complete control over the device. In this case, an adversary can control Zigbee or Wi-Fi communication. To access SWD, an adversary can use, for example, a JLink debugger.

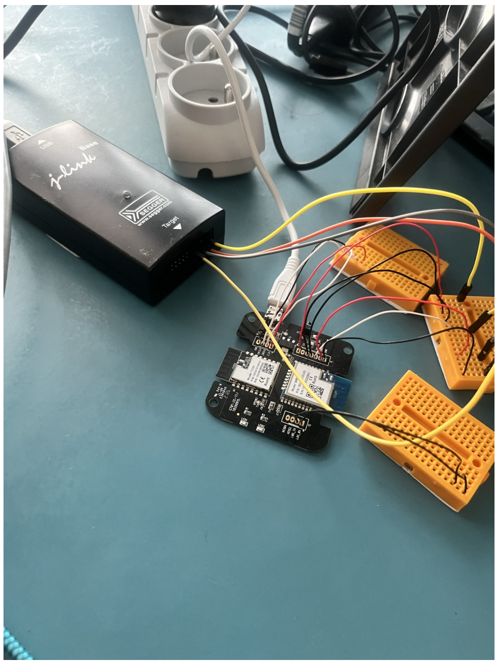
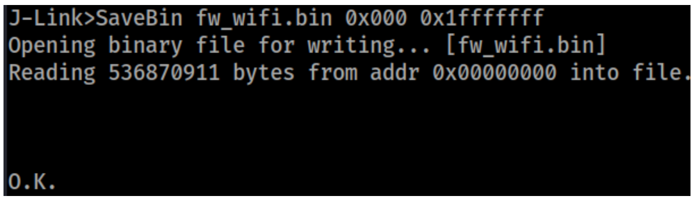


Figure 4.41: Connecting to SWD

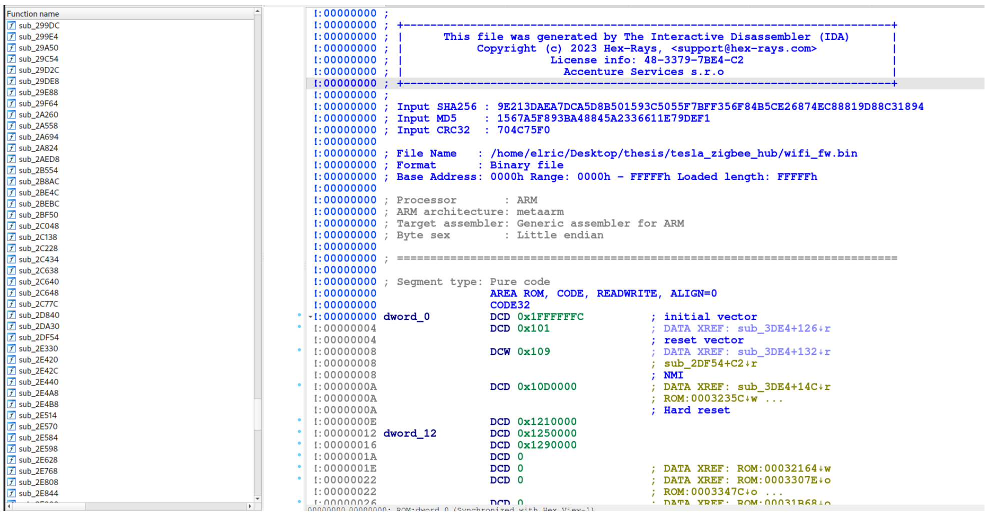
One of the options is to run the JLink GDB server and connect to the processor through IDA. IDA allows connecting to remote GDB servers and debugging in real time. To connect to SWD, an adversary has to know the type of CPU and frequency of the CPU. For example, the WRG1 documentation specifies that this module uses ARM cortex-m3 [30]. After searching for the typical frequency for an ARM cortex-m3 processor, the results say that the typical frequency is 72 MHz. After connecting JLink to SWD ports and JLink to the computer, JLink can connect to SWD and start GDB server, which allows debugging processor in real-time. The next step is to connect IDA to the remote process. After a successful connection, an adversary can start debugging the processor and control the flow. An adversary can debug the processor in real-time or download the firmware through SWD, as SWD allows to write or read memory. The documentation for RTL8711AM is available online and it describes the memory map of the modules - the focus will be on WRG1. The code section of memory is located from **0x0000000** to **0x1fffffff**. The memory can be downloaded through JLink.



```
J-Link>SaveBin fw_wifi.bin 0x000 0x1fffffff
Opening binary file for writing... [fw_wifi.bin]
Reading 536870911 bytes from addr 0x00000000 into file.



O.K.
```

Figure 4.42: Reading memory through SWD

The code section can be loaded into IDA and analyzed.

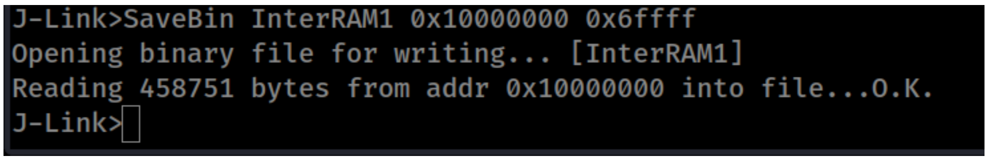


```
Function name
sub_299DC
sub_299E4
sub_29A50
sub_29C54
sub_29D2C
sub_29DE8
sub_29E88
sub_29F64
sub_2A260
sub_2A558
sub_2A694
sub_2A824
sub_2AED8
sub_2B554
sub_2B8AC
sub_2BE4C
sub_2BEBC
sub_2BF50
sub_2C048
sub_2C138
sub_2C228
sub_2C434
sub_2C638
sub_2C640
sub_2C648
sub_2C77C
sub_2D840
sub_2DA30
sub_2DF54
sub_2E330
sub_2E420
sub_2E42C
sub_2E440
sub_2E4A8
sub_2E4B8
sub_2E514
sub_2E570
sub_2E584
sub_2E598
sub_2E628
sub_2E768
sub_2E808
sub_2E844
```

```
I:00000000 ;
I:00000000 ; +-------------------------------------------------------------------------+
I:00000000 ; |      This file was generated by The Interactive Disassembler (IDA)      |
I:00000000 ; |          Copyright (c) 2023 Hex-Rays, <support@hex-rays.com>           |
I:00000000 ; |                     License info: 48-3379-7BE4-C2                      |
I:00000000 ; |                       Accenture Services s.r.o                         |
I:00000000 ; +-------------------------------------------------------------------------+
I:00000000 ;
I:00000000 ; Input SHA256 : 9E213DAEA7DCA5D8B501593C5055F7BFF356F84B5CE26874EC88819D88C31894
I:00000000 ; Input MD5    : 1567A5F893BA48845A2336611E79DEF1
I:00000000 ; Input CRC32  : 704C75F0
I:00000000
I:00000000 ; File Name   : /home/elric/Desktop/thesis/tesla_zigbee_hub/wifi_fw.bin
I:00000000 ; Format      : Binary file
I:00000000 ; Base Address: 0000h Range: 0000h - FFFFFh Loaded length: FFFFFh
I:00000000
I:00000000 ; Processor       : ARM
I:00000000 ; ARM architecture: metaarm
I:00000000 ; Target assembler: Generic assembler for ARM
I:00000000 ; Byte sex        : Little endian
I:00000000
I:00000000 ; ===========================================================================
I:00000000
I:00000000 ; Segment type: Pure code
I:00000000                 AREA ROM, CODE, READWRITE, ALIGN=0
I:00000000                 CODE32
I:00000000 dword_0         DCD 0x1FFFFFFC          ; initial vector
I:00000004                 DCD 0x101               ; DATA XREF: sub_3DE4+126↓r
I:00000004                                         ; reset vector
I:00000008                 DCW 0x109               ; DATA XREF: sub_3DE4+132↓r
I:00000008                                         ; sub_2DF54+C2↓r
I:00000008                                         ; NMI
I:0000000A                 DCD 0x10D0000           ; DATA XREF: sub_3DE4+14C↓r
I:0000000A                                         ; ROM:0003235C↓w ...
I:0000000A                                         ; Hard reset
I:0000000E                 DCD 0x1210000
I:00000012 dword_12        DCD 0x1250000
I:00000016                 DCD 0x1290000
I:0000001A                 DCD 0
I:0000001E                 DCD 0                   ; DATA XREF: ROM:00032164↓w
I:00000022                 DCD 0                   ; DATA XREF: ROM:0003307E↓o
I:00000022                                         ; ROM:0003347C↓o ...
I:00000026                 DCD 0                   ; DATA XREF: ROM:00031B68↓o
00000000 00000000: ROM:dword_0 (Synchronized with Hex View-1)
```

Figure 4.43: Firmware analysis in IDA

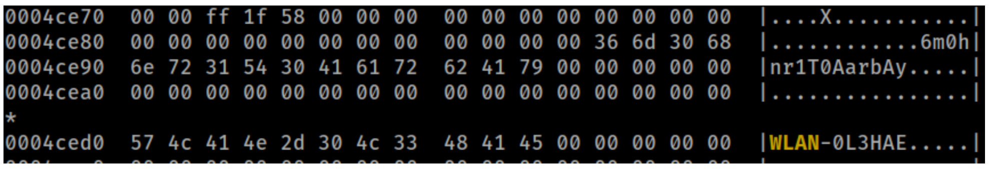
During the gateway configuration, a user sets up a Wi-Fi name and password to allow the gateway to connect to the network. The WRG1 module contains a section of memory called Inter-RAM, where the firmware stores the data it uses during execution in the code section. It would be fair to assume that somewhere in the code section, the device will run instructions that connect the device to the Wi-Fi. As the parameters for connection will be stored in the Inter-RAM, an adversary can read the Inter-RAM and locate the credentials. The Wi-Fi name and password will be used simultaneously to be placed close together in the memory. Therefore, after locating the Wi-Fi name in the RAM section, a string located closely will be the password. The RAM section can be saved into the file with the command *SaveBin*.



```
J-Link>SaveBin InterRAM1 0x10000000 0x6ffff
Opening binary file for writing... [InterRAM1]
Reading 458751 bytes from addr 0x10000000 into file...O.K.
J-Link>
```

Figure 4.44: Saving RAM through SWD

If an adversary knows the Wi-Fi name, they can locate it inside the file.



```
0004ce70  00 00 ff 1f 58 00 00 00  00 00 00 00 00 00 00 00  |....X...........|
0004ce80  00 00 00 00 00 00 00 00  00 00 00 00 36 6d 30 68  |............6m0h|
0004ce90  6e 72 31 54 30 41 61 72  62 41 79 00 00 00 00 00  |nr1T0AarbAy.....|
0004cea0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
0004ced0  57 4c 41 4e 2d 30 4c 33  48 41 45 00 00 00 00 00  |WLAN-0L3HAE.....|
```

Figure 4.45: Wi-Fi password in the RAM

4.2.1.4 Reporting

This subsection summarizes the vulnerabilities in Tesla Zigbee Hub. The results are presented in the following table:

Vulnerability	CVSS	Mitigation
Accesible SWD port	5.3	Disable unnecessary interfaces in production
Wi-Fi password accesible in the memory	6.0	Restrict memory access for hardware interface

Table 4.3: Table of vulnerabities in Tesla Zigbee Hub

4.3 Smart Locks

In this section, Danalock V3 and Dapanad V3 will be analyzed.

4.3.1 Danalock

Danalock is a smart lock using Bluetooth Low Energy (BLE) or Zigbee. It allows control of the lock remotely and shares access between users.

4.3.1.1 Information gathering

The disassembling requires more effort for the smart lock. After removing the case, the device contains one PCB.

The smart lock contains one chip - N52832. The full name of the chip is *nRF52832*, which is Bluetooth 5.3 SoC capable of connecting with Bluetooth Low Energy. At first sight, there is little information about the structure of PCB. There are no labels and six visible testing points. The N52832 has an available document containing much interesting information. Among other things, it includes pin mapping, which describes the functionality of each pin on the chip. One of the ways to determine functionality is to use a voltmeter

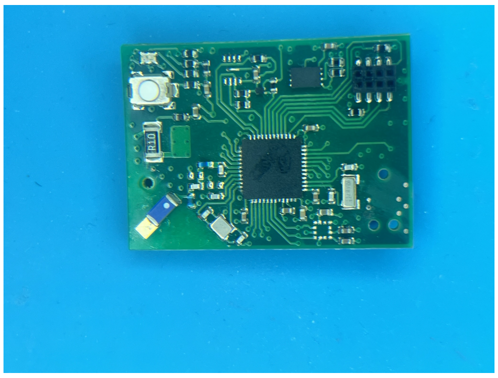


Figure 4.46: Printed circuit board

and assign each testing point to the chip's pin. Neither pins may be connected to the chip, but it can contain some hidden functionality if they are.

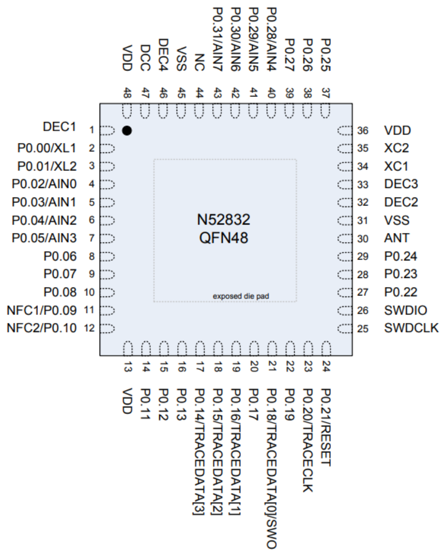


Figure 4.47: N52832 Pinout [31]

Voltmeter can help to identify SWD pins on the board. However, the testing points only contain *SWDIO*, *SWDCLK*, and *GND*. There is no VCC pin on the pins. One way to approach this is to find a voltage pin with the same voltage level as the highest value of SWDIO or SWDCLK can gain. Another way is to find a VCC pin anyway. VCC from the chip can lead to other places on the board, and one of the candidates is the stack of pins.

After further testing, the VCC pin is present on one of the pins on the pin

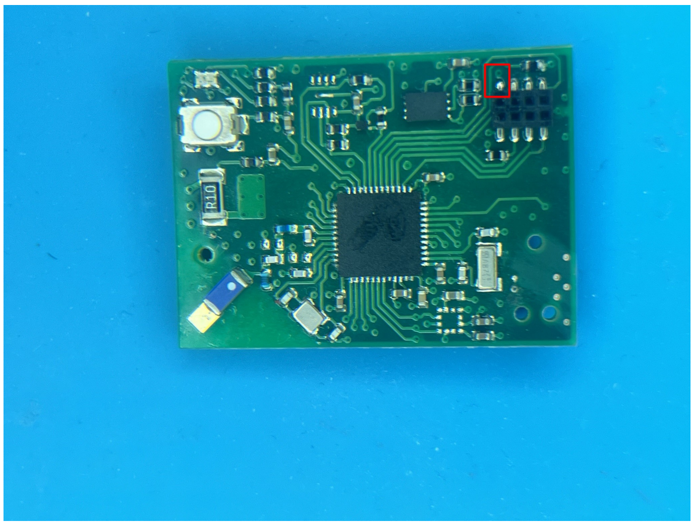


Figure 4.48: VCC pin

stack.

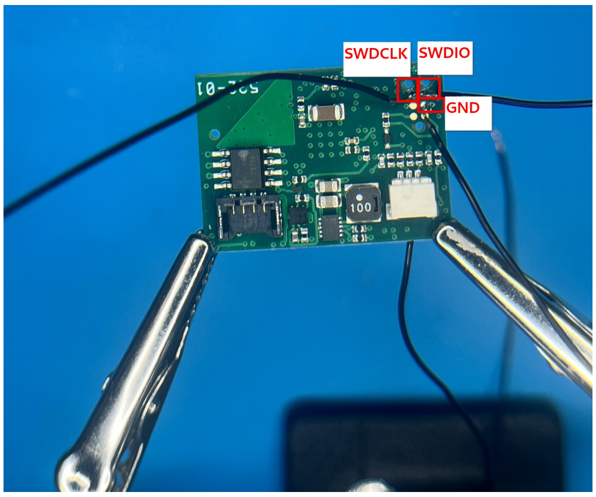
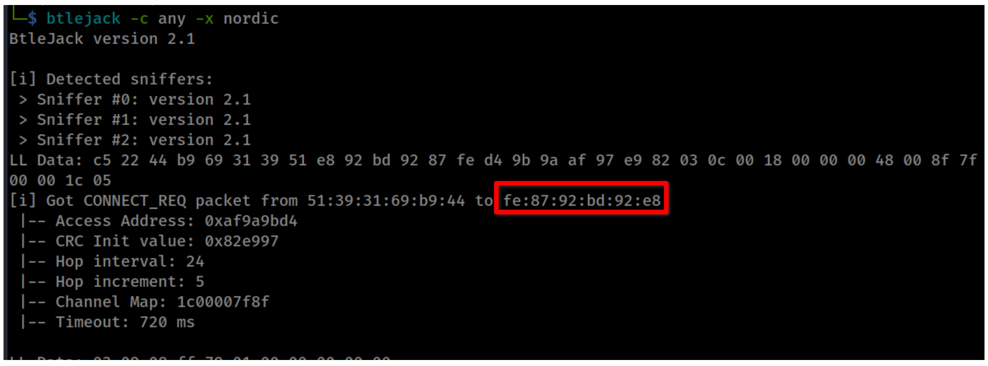


Figure 4.49: SWD interface

The BLE communication is a vital part of the functionality, therefore, it is crucial to examine. The first step is to identify the Bluetooth address of the device. One approach is to try to sniff for incoming connections, which will capture BLE addresses, among other information. Danalock uses BLE to communicate with the phone and send the lock and unlock commands. Such a tool is, for example, *Btlejack*.

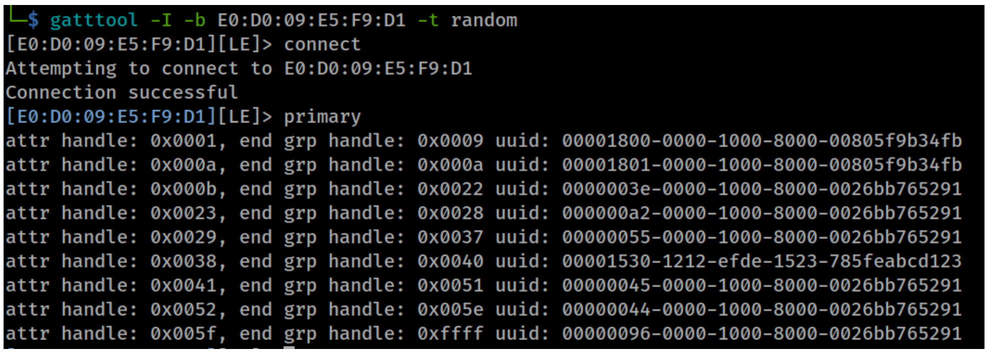
After trying to send a lock command from the phone, *Btlejack* will capture the request and print out information. Right now, the important part is that



```
└─$ btlejack -c any -x nordic
BtleJack version 2.1

[i] Detected sniffers:
 > Sniffer #0: version 2.1
 > Sniffer #1: version 2.1
 > Sniffer #2: version 2.1
LL Data: c5 22 44 b9 69 31 39 51 e8 92 bd 92 87 fe d4 9b 9a af 97 e9 82 03 0c 00 18 00 00 00 48 00 8f 7f
00 00 1c 05
[i] Got CONNECT_REQ packet from 51:39:31:69:b9:44 to fe:87:92:bd:92:e8
 |-- Access Address: 0xaf9a9bd4
 |-- CRC Init value: 0x82e997
 |-- Hop interval: 24
 |-- Hop increment: 5
 |-- Channel Map: 1c00007f8f
 |-- Timeout: 720 ms
```

Figure 4.50: Btlejack sniffing

Danalock has Bluetooth address fe:87:92:bd:92:e8. Another helpful hint is that the phone connects to the smart lock, meaning Danalock acts as a peripheral device and, therefore, can contain GATT service. An adversary can read or write data to the critical parts if the GATT is misconfigured. *Gattool* is a script that allows connecting to and exploring the GATT service.

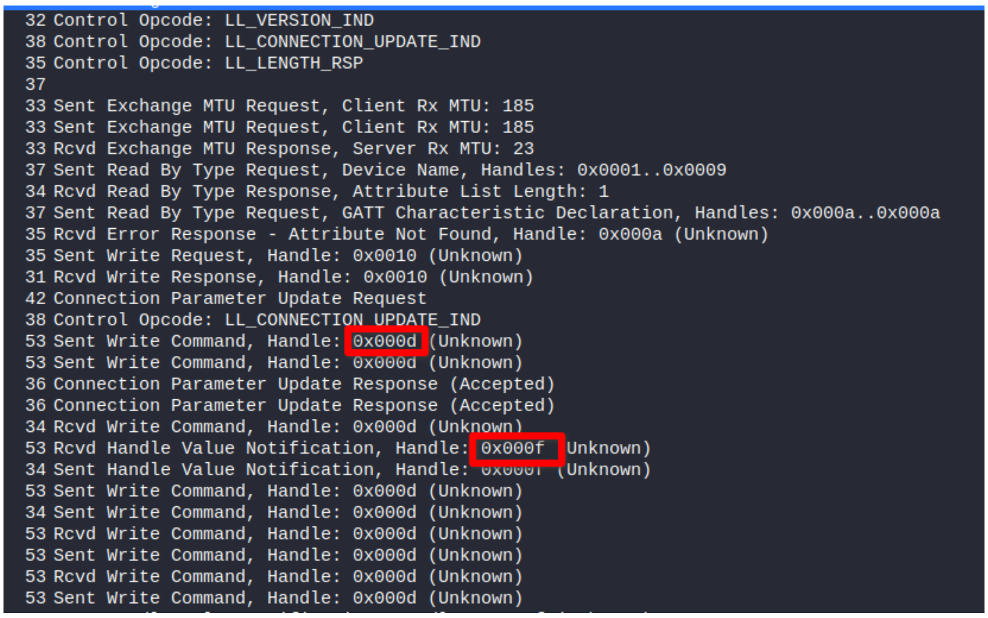


```
└─$ gatttool -I -b E0:D0:09:E5:F9:D1 -t random
[E0:D0:09:E5:F9:D1][LE]> connect
Attempting to connect to E0:D0:09:E5:F9:D1
Connection successful
[E0:D0:09:E5:F9:D1][LE]> primary
attr handle: 0x0001, end grp handle: 0x0009 uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle: 0x000a, end grp handle: 0x000a uuid: 00001801-0000-1000-8000-00805f9b34fb
attr handle: 0x000b, end grp handle: 0x0022 uuid: 0000003e-0000-1000-8000-0026bb765291
attr handle: 0x0023, end grp handle: 0x0028 uuid: 000000a2-0000-1000-8000-0026bb765291
attr handle: 0x0029, end grp handle: 0x0037 uuid: 00000055-0000-1000-8000-0026bb765291
attr handle: 0x0038, end grp handle: 0x0040 uuid: 00001530-1212-efde-1523-785feabcd123
attr handle: 0x0041, end grp handle: 0x0051 uuid: 00000045-0000-1000-8000-0026bb765291
attr handle: 0x0052, end grp handle: 0x005e uuid: 00000044-0000-1000-8000-0026bb765291
attr handle: 0x005f, end grp handle: 0xffff uuid: 00000096-0000-1000-8000-0026bb765291
```

Figure 4.51: Gattool connection

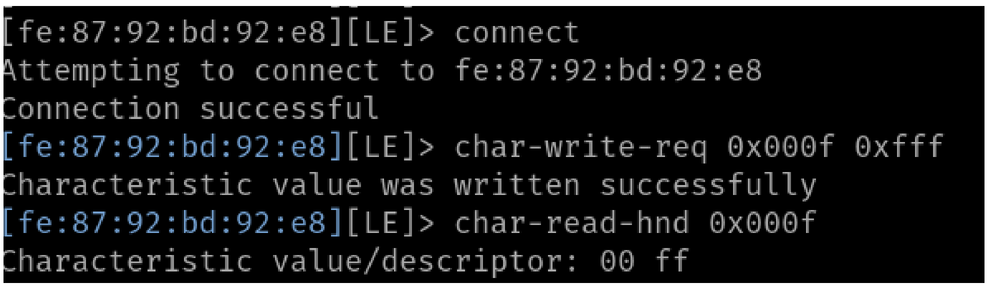
There are three primary services, two of which are standard BLE services, and the third is custom service. The first two services start with 00001800 - a common prefix for routine services. The third one begins with *bce90001*, which is not included in common prefixes for BLE services. Therefore, the main characteristics are **0x000d** and **0x000f**. As the Btlejack also allows capturing the communication and saving it to a *pcap* file, these two handles are often present.

From the Wireshark output, it would be fair to assume that 0x000d is a handle for writing data and 0x000f serves for reading the data out. Using the Gattool, reading or writing to the mentioned handles is possible.



```
32 Control Opcode: LL_VERSION_IND
38 Control Opcode: LL_CONNECTION_UPDATE_IND
35 Control Opcode: LL_LENGTH_RSP
37
33 Sent Exchange MTU Request, Client Rx MTU: 185
33 Sent Exchange MTU Request, Client Rx MTU: 185
33 Rcvd Exchange MTU Response, Server Rx MTU: 23
37 Sent Read By Type Request, Device Name, Handles: 0x0001..0x0009
34 Rcvd Read By Type Response, Attribute List Length: 1
37 Sent Read By Type Request, GATT Characteristic Declaration, Handles: 0x000a..0x000a
35 Rcvd Error Response - Attribute Not Found, Handle: 0x000a (Unknown)
35 Sent Write Request, Handle: 0x0010 (Unknown)
31 Rcvd Write Response, Handle: 0x0010 (Unknown)
42 Connection Parameter Update Request
38 Control Opcode: LL_CONNECTION_UPDATE_IND
53 Sent Write Command, Handle: 0x000d (Unknown)
53 Sent Write Command, Handle: 0x000d (Unknown)
36 Connection Parameter Update Response (Accepted)
36 Connection Parameter Update Response (Accepted)
34 Rcvd Write Command, Handle: 0x000d (Unknown)
53 Rcvd Handle Value Notification, Handle: 0x000f (Unknown)
34 Sent Handle Value Notification, Handle: 0x000f (Unknown)
53 Sent Write Command, Handle: 0x000d (Unknown)
34 Sent Write Command, Handle: 0x000d (Unknown)
53 Rcvd Write Command, Handle: 0x000d (Unknown)
53 Sent Write Command, Handle: 0x000d (Unknown)
53 Rcvd Write Command, Handle: 0x000d (Unknown)
53 Sent Write Command, Handle: 0x000d (Unknown)
```

Figure 4.52: Wireshark capture

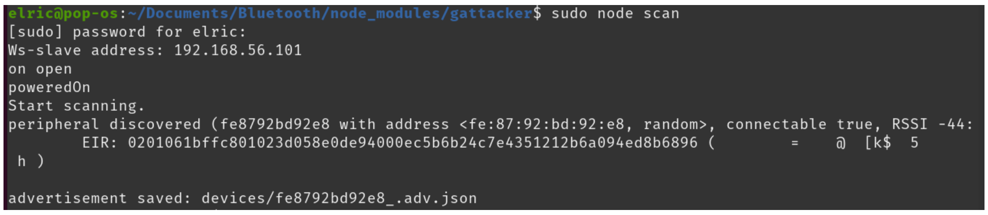


```
[fe:87:92:bd:92:e8][LE]> connect
Attempting to connect to fe:87:92:bd:92:e8
Connection successful
[fe:87:92:bd:92:e8][LE]> char-write-req 0x000f 0xfff
Characteristic value was written successfully
[fe:87:92:bd:92:e8][LE]> char-read-hnd 0x000f
Characteristic value/descriptor: 00 ff
```

Figure 4.53: Write request in Gattool

But, the data written into these handles are encrypted and change with every communication. The first step is to capture advertising packets, which are later used to advertise malicious peripheral devices. The next step is to try to perform an MITM attack. Such an attack allows further analysis and a way to modify incoming data. One such tool for MITM is a gattacker. It will enable using two Bluetooth cards as peripheral and central and inspect the traffic incoming from target devices. In the testing setup, a virtual machine with one Bluetooth card will act as a peripheral, and native Linux with a second Bluetooth card will work as central.

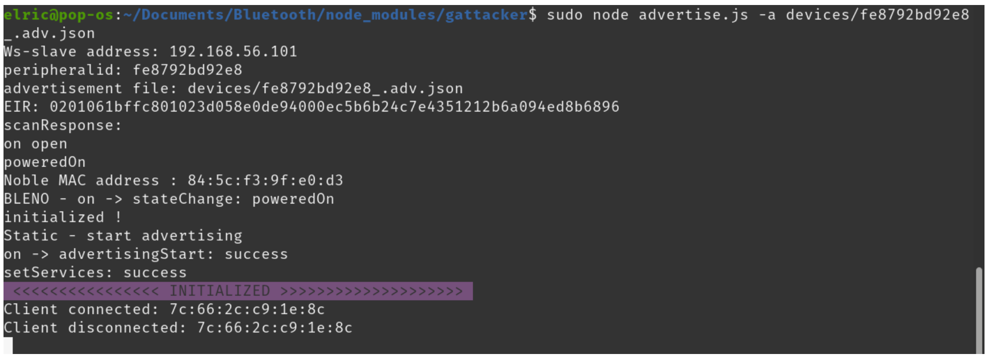
The second step is to scan the GATT service and its structure. The



```
elric@pop-os:~/Documents/Bluetooth/node_modules/gattacker$ sudo node scan
[sudo] password for elric:
Ws-slave address: 192.168.56.101
on open
poweredOn
Start scanning.
peripheral discovered (fe8792bd92e8 with address <fe:87:92:bd:92:e8, random>, connectable true, RSSI -44:
        EIR: 0201061bffc801023d058e0de94000ec5b6b24c7e4351212b6a094ed8b6896 (        =    @  [k$  5
 h )

advertisement saved: devices/fe8792bd92e8_.adv.json
```

Figure 4.54: Gattacker scan

scanned GATT service is saved into a JSON file. All this information are then used to spoof the central and peripheral device and try to sniff the communication.



```
elric@pop-os:~/Documents/Bluetooth/node_modules/gattacker$ sudo node advertise.js -a devices/fe8792bd92e8
_.adv.json
Ws-slave address: 192.168.56.101
peripheralid: fe8792bd92e8
advertisement file: devices/fe8792bd92e8_.adv.json
EIR: 0201061bffc801023d058e0de94000ec5b6b24c7e4351212b6a094ed8b6896
scanResponse:
on open
poweredOn
Noble MAC address : 84:5c:f3:9f:e0:d3
BLENO - on -> stateChange: poweredOn
initialized !
Static - start advertising
on -> advertisingStart: success
setServices: success
 <<<<<<<<<<<<<<<< INITIALIZED >>>>>>>>>>>>>>>>>>>>
Client connected: 7c:66:2c:c9:1e:8c
Client disconnected: 7c:66:2c:c9:1e:8c
```

Figure 4.55: MITM in Gattacker

The scanned GATT service is saved into a JSON file. All this information are then used to spoof the central and peripheral device and try to sniff the communication. However, the MITM is impossible because whenever the user wants to unlock or lock, the devices create a new connection and perform a pairing procedure.

4.3.1.2 Threat modeling

The only identified hardware interface is **SWD**. That still can be dangerous; at the very least, it allows an adversary to read the firmware and modify registers in the CPU. The BLE communication is the most necessary to secure, as the interface controls the locking mechanism. Danalock acts as a peripheral device, meaning it runs the GATT service. The GATT service is used for data exchange between peripheral and central. It should be secured not to allow unauthenticated users to read or write commands for unlocking or locking, or the GATT should be used in a fashion where even writing data into GATT services does not pose a risk. This can be achieved by encrypting the commands and including a counter in the message.

4.3.1.3 Vulnerability analysis and exploitation

The SWD interface allows the control of the CPU, reading and writing to the memory. To try to communicate to SWD with JLink, for example, an adversary needs to know the processor type and frequency. All this information is available in the document: the processor type of ARM Cortex M4 and its frequency is 64 MHz. From here, an adversary can try to connect to the chip and control its functionality. All this information allows us to communicate with SWD.

The next step is to connect to the GDB server with IDA and control the flow of the CPU. Besides debugging in real-time, an adversary can download the code and try to reverse-engineer it. However, downloading the whole memory is very time-consuming and inefficient. The better approach is to download and analyze only the code section of memory. Such an approach requires knowledge of the memory map, also available in the datasheet.

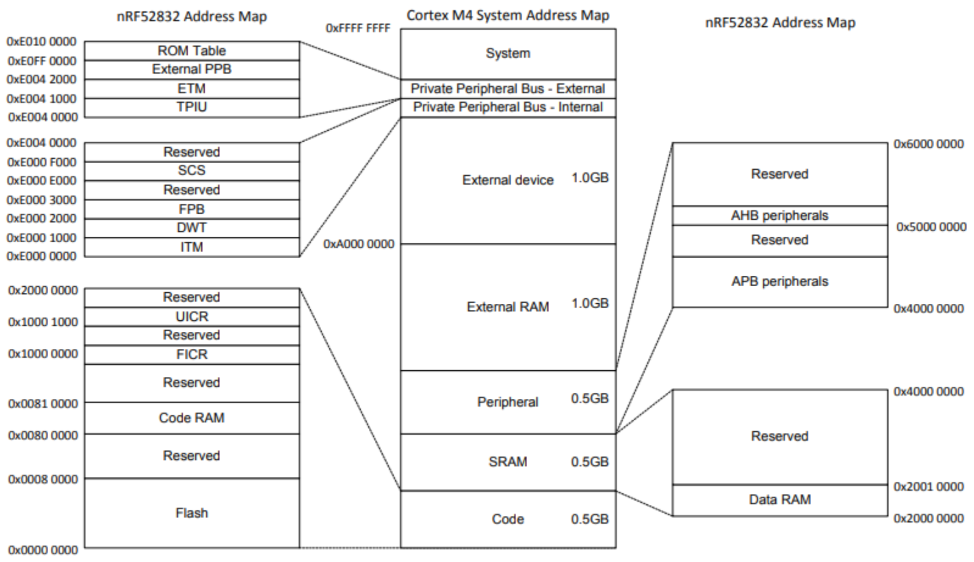
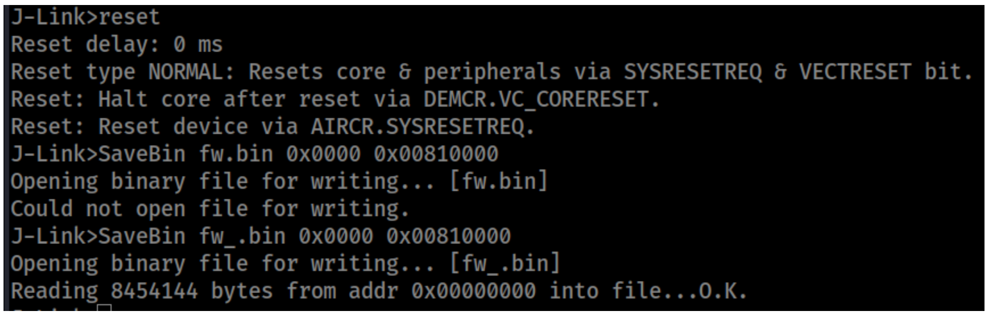


Figure 4.56: Memory map of N52832 [31]

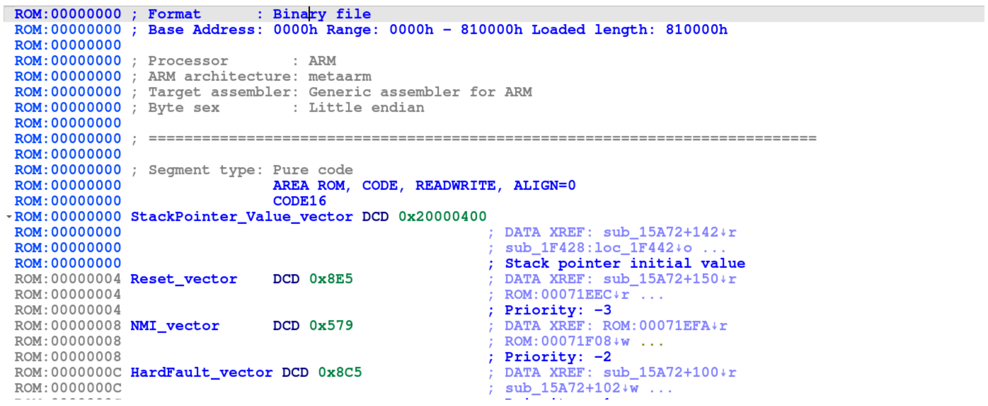
The valuable part of memory is from address **0x0000** to **0x00810000**. JLink also offers a command-line utility for SWD. Running JLinkExe will prompt the connection setup and command line, where an adversary can control the CPU and send SWD commands. To save the memory to the file, an adversary can use the *SaveBin* command.

The downloaded firmware can be further analyzed with IDA.



```
J-Link>reset
Reset delay: 0 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
Reset: Halt core after reset via DEMCR.VC_CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
J-Link>SaveBin fw.bin 0x0000 0x00810000
Opening binary file for writing... [fw.bin]
Could not open file for writing.
J-Link>SaveBin fw_.bin 0x0000 0x00810000
Opening binary file for writing... [fw_.bin]
Reading 8454144 bytes from addr 0x00000000 into file...O.K.
```

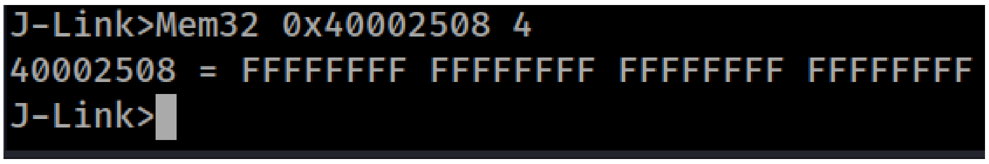
Figure 4.57: Firmware download through SWD



```
ROM:00000000 ; Format      : Binary file
ROM:00000000 ; Base Address: 0000h Range: 0000h - 810000h Loaded length: 810000h
ROM:00000000
ROM:00000000 ; Processor        : ARM
ROM:00000000 ; ARM architecture: metaarm
ROM:00000000 ; Target assembler: Generic assembler for ARM
ROM:00000000 ; Byte sex         : Little endian
ROM:00000000
ROM:00000000 ; ===========================================================================
ROM:00000000
ROM:00000000 ; Segment type: Pure code
ROM:00000000                 AREA ROM, CODE, READWRITE, ALIGN=0
ROM:00000000                 CODE16
ROM:00000000 StackPointer_Value_vector DCD 0x20000400
ROM:00000000                                         ; DATA XREF: sub_15A72+142↓r
ROM:00000000                                         ; sub_1F428:loc_1F442↓o ...
ROM:00000000                                         ; Stack pointer initial value
ROM:00000004 Reset_vector    DCD 0x8E5               ; DATA XREF: sub_15A72+150↓r
ROM:00000004                                         ; ROM:00071EEC↓r ...
ROM:00000004                                         ; Priority: -3
ROM:00000008 NMI_vector      DCD 0x579               ; DATA XREF: ROM:00071EFA↓r
ROM:00000008                                         ; ROM:00071F08↓w ...
ROM:00000008                                         ; Priority: -2
ROM:0000000C HardFault_vector DCD 0x8C5              ; DATA XREF: sub_15A72+100↓r
ROM:0000000C                                         ; sub_15A72+102↓w ...
```

Figure 4.58: Firmware analysis in IDA

In ARM binaries, an interrupt vector table is present from address 0x00. This table contains memory addresses for various interrupt events, such as reset. The address 0x00 has an initial stack pointer (SP) value. With access to the documentation and processor, additional interfaces can also be analyzed. Access to the running CPU makes it possible to read the memory. The chip also contains UART, and the adversary can look for the part of the code where UART is being handled. In the memory, the UART configuration starts at *0x40002000*. The configuration for the pins lies between *0x40002508* and *0x40002514*. If there is a value of 0xffffff between these addresses, the UART is disabled and has no mapping to any pins.



```
J-Link>Mem32 0x40002508 4
40002508 = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
J-Link>
```

Figure 4.59: UART configuration

4.3.1.4 Reporting

The following table summarizes the results of security analysis of Danalock:

Vulnerability	CVSS	Mitigation
SWD interface	3.2	Disable unnecessary hardware interfaces

Table 4.4: Table of vulnerabities in Danalock

4.3.2 Danapad

Danapad is an additional device that controls the lock with a keypad. The user setups the pin code, and after typing the correct pin, it can lock or unlock the smart lock.

4.3.2.1 Information gathering

Danapad contains one PCB, which has all the needed functionality. There is a keypad from one side, and from the other is a chip and other components.

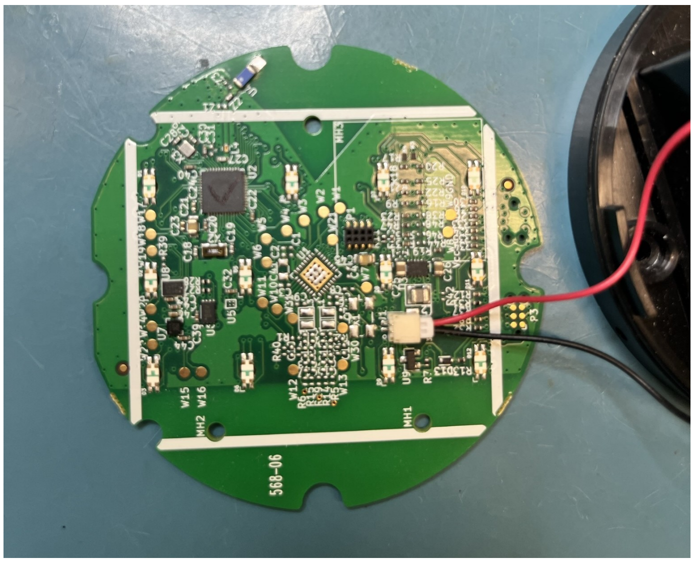


Figure 4.60: Printed circuit board

The device contains the same chip as Danalock - *nRF52832*. It also has multiple pins, many with labels. But they do not tell much about the functionality. But, with access to the datasheet, it is again possible to enumerate the pins with the voltmeter and pin assignment from the datasheet.

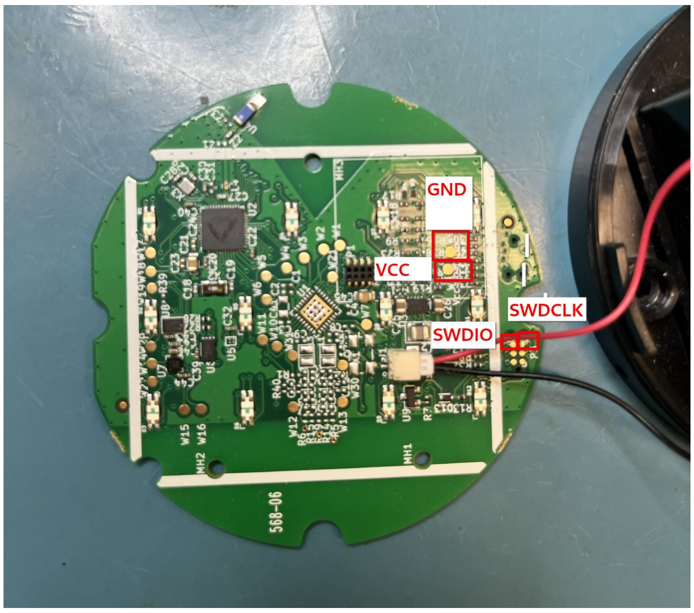
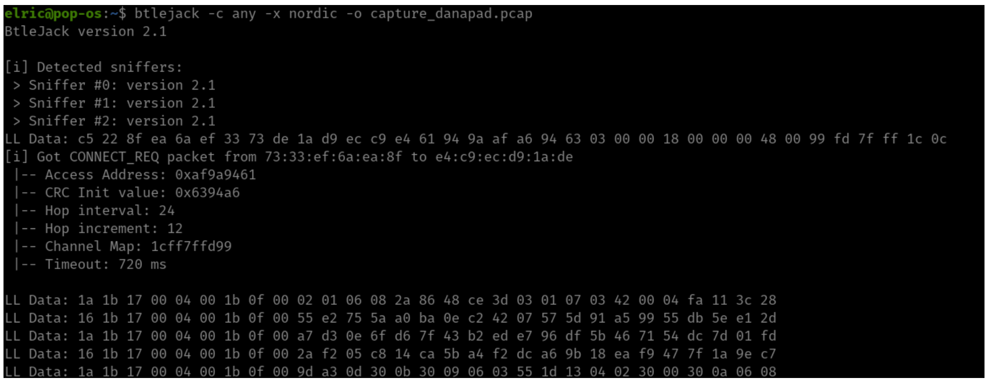


Figure 4.61: SWD interface

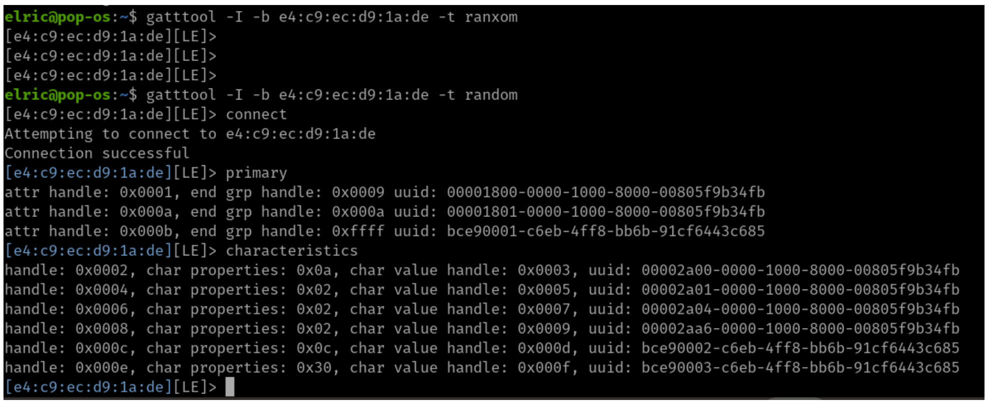
Danapad sends the BLE commands between a phone or Danalock. It offers exciting options to analyze communication and possibly gain control over it. The first is to identify the Bluetooth address. The address can be identified while sniffing the communication during pairing - for example, with Btlejack.



```
elric@pop-os:~$ btlejack -c any -x nordic -o capture_danapad.pcap
BtleJack version 2.1

[i] Detected sniffers:
 > Sniffer #0: version 2.1
 > Sniffer #1: version 2.1
 > Sniffer #2: version 2.1
LL Data: c5 22 8f ea 6a ef 33 73 de 1a d9 ec c9 e4 61 94 9a af a6 94 63 03 00 00 18 00 00 00 48 00 99 fd 7f ff 1c 0c
[i] Got CONNECT_REQ packet from 73:33:ef:6a:ea:8f to e4:c9:ec:d9:1a:de
 |-- Access Address: 0xaf9a9461
 |-- CRC Init value: 0x6394a6
 |-- Hop interval: 24
 |-- Hop increment: 12
 |-- Channel Map: 1cff7ffd99
 |-- Timeout: 720 ms

LL Data: 1a 1b 17 00 04 00 1b 0f 00 02 01 06 08 2a 86 48 ce 3d 03 01 07 03 42 00 04 fa 11 3c 28
LL Data: 16 1b 17 00 04 00 1b 0f 00 55 e2 75 5a a0 ba 0e c2 42 07 57 5d 91 a5 99 55 db 5e e1 2d
LL Data: 1a 1b 17 00 04 00 1b 0f 00 a7 d3 0e 6f d6 7f 43 b2 ed e7 96 df 5b 46 71 54 dc 7d 01 fd
LL Data: 16 1b 17 00 04 00 1b 0f 00 2a f2 05 c8 14 ca 5b a4 f2 dc a6 9b 18 ea f9 47 7f 1a 9e c7
LL Data: 1a 1b 17 00 04 00 1b 0f 00 9d a3 0d 30 0b 30 09 06 03 55 1d 13 04 02 30 00 30 0a 06 08
```

Figure 4.62: Btlejack sniffing

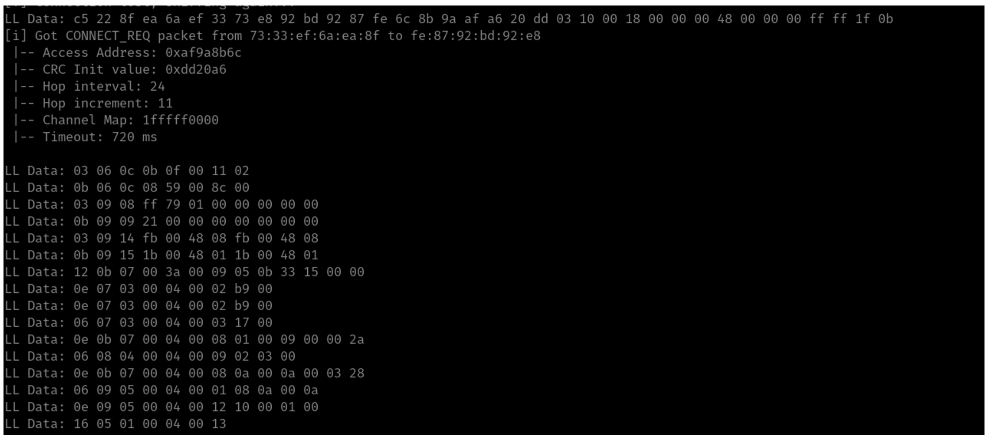
The Bluetooth of Danapad is e4:c9:ec:d9:1a:de. From the nature of communication, Danapad is a peripheral device. Therefore, the next step is to enumerate the GATT service.



```
elric@pop-os:~$ gatttool -I -b e4:c9:ec:d9:1a:de -t ranxom
[e4:c9:ec:d9:1a:de][LE]>
[e4:c9:ec:d9:1a:de][LE]>
[e4:c9:ec:d9:1a:de][LE]>
elric@pop-os:~$ gatttool -I -b e4:c9:ec:d9:1a:de -t random
[e4:c9:ec:d9:1a:de][LE]> connect
Attempting to connect to e4:c9:ec:d9:1a:de
Connection successful
[e4:c9:ec:d9:1a:de][LE]> primary
attr handle: 0x0001, end grp handle: 0x0009 uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle: 0x000a, end grp handle: 0x000a uuid: 00001801-0000-1000-8000-00805f9b34fb
attr handle: 0x000b, end grp handle: 0xffff uuid: bce90001-c6eb-4ff8-bb6b-91cf6443c685
[e4:c9:ec:d9:1a:de][LE]> characteristics
handle: 0x0002, char properties: 0x0a, char value handle: 0x0003, uuid: 00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0004, char properties: 0x02, char value handle: 0x0005, uuid: 00002a01-0000-1000-8000-00805f9b34fb
handle: 0x0006, char properties: 0x02, char value handle: 0x0007, uuid: 00002a04-0000-1000-8000-00805f9b34fb
handle: 0x0008, char properties: 0x02, char value handle: 0x0009, uuid: 00002aa6-0000-1000-8000-00805f9b34fb
handle: 0x000c, char properties: 0x0c, char value handle: 0x000d, uuid: bce90002-c6eb-4ff8-bb6b-91cf6443c685
handle: 0x000e, char properties: 0x30, char value handle: 0x000f, uuid: bce90003-c6eb-4ff8-bb6b-91cf6443c685
[e4:c9:ec:d9:1a:de][LE]>
```

Figure 4.63: Gattool connection

The structure of the GATT service is very similar to Danalock. The difference, however, is that while setting up the pin code, it is not saved into Danapad but rather to Danalock. From the wireshark communication, it is again clear that 0x00d serves for writing the data and 0x000f for reading the data.



```
LL Data: c5 22 8f ea 6a ef 33 73 e8 92 bd 92 87 fe 6c 8b 9a af a6 20 dd 03 10 00 18 00 00 00 48 00 00 00 ff ff 1f 0b
[i] Got CONNECT_REQ packet from 73:33:ef:6a:ea:8f to fe:87:92:bd:92:e8
 |-- Access Address: 0xaf9a8b6c
 |-- CRC Init value: 0xdd20a6
 |-- Hop interval: 24
 |-- Hop increment: 11
 |-- Channel Map: 1fffff0000
 |-- Timeout: 720 ms

LL Data: 03 06 0c 0b 0f 00 11 02
LL Data: 0b 06 0c 08 59 00 8c 00
LL Data: 03 09 08 ff 79 01 00 00 00 00 00
LL Data: 0b 09 09 21 00 00 00 00 00 00 00
LL Data: 03 09 14 fb 00 48 08 fb 00 48 08
LL Data: 0b 09 15 1b 00 48 01 1b 00 48 01
LL Data: 12 0b 07 00 3a 00 09 05 0b 33 15 00 00
LL Data: 0e 07 03 00 04 00 02 b9 00
LL Data: 0e 07 03 00 04 00 02 b9 00
LL Data: 06 07 03 00 04 00 03 17 00
LL Data: 0e 0b 07 00 04 00 08 01 00 09 00 00 2a
LL Data: 06 08 04 00 04 00 09 02 03 00
LL Data: 0e 0b 07 00 04 00 08 0a 00 0a 00 03 28
LL Data: 06 09 05 00 04 00 01 08 0a 00 0a
LL Data: 0e 09 05 00 04 00 12 10 00 01 00
LL Data: 16 05 01 00 04 00 13
```

Figure 4.64: Setting up pin through BLE

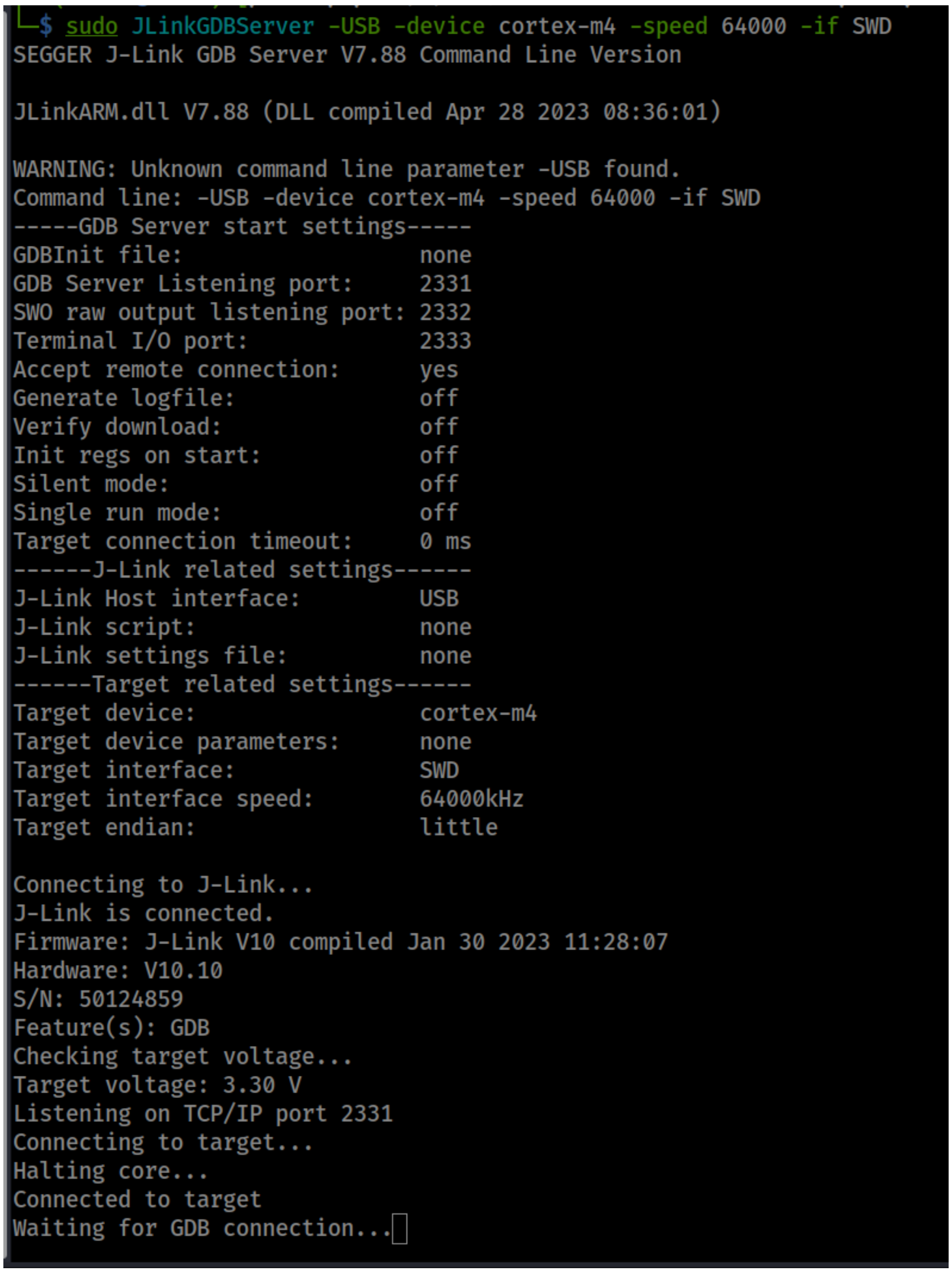
At the same time, a typed pin from Danapad is sent to Danalock through BLE.The previous chapter defined the problem with MITM of Danalock. Due to the same reason, this analysis, it is not possible to capture the communication and replay the code, nor is it possible to inject the pin code into Danalock without further research and advanced attacks.

4.3.2.2 Threat modeling

Danapad serves as the middleman between the user and the lock. The user enters the pin, and if correct, Danapad locks or unlocks the Danalock. With the SWD interface present, if the PIN is saved into the Danapad, an adversary can read the correct pin through the interface. As the Danapad can be placed outside the house, it can allow the attacker to access the house. Or an adversary can find the function which sends the unlock command and, through SWD, opens the lock. BLE can also be a threat. An adversary can try spoofing the unlock communication, or if the GATT service contains the correct PIN, they can read out the PIN through GATT.

4.3.2.3 Vulnerability analysis and exploitation

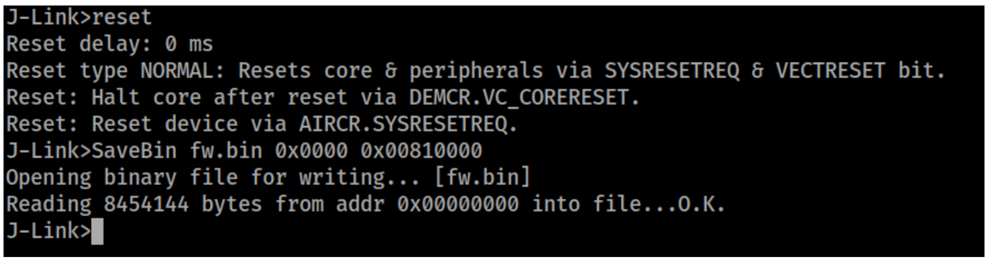
Same as in previous sections, Danapad also supports **SWD**. As the chip is the same as the Danalock, the same procedure can be used to connect to the CPU. The processor type and frequency stay the same.



```
└─$ sudo JLinkGDBServer -USB -device cortex-m4 -speed 64000 -if SWD
SEGGER J-Link GDB Server V7.88 Command Line Version

JLinkARM.dll V7.88 (DLL compiled Apr 28 2023 08:36:01)

WARNING: Unknown command line parameter -USB found.
Command line: -USB -device cortex-m4 -speed 64000 -if SWD
-----GDB Server start settings-----
GDBInit file:                  none
GDB Server Listening port:     2331
SWO raw output listening port: 2332
Terminal I/O port:             2333
Accept remote connection:      yes
Generate logfile:              off
Verify download:               off
Init regs on start:            off
Silent mode:                   off
Single run mode:               off
Target connection timeout:     0 ms
------J-Link related settings------
J-Link Host interface:         USB
J-Link script:                 none
J-Link settings file:          none
------Target related settings------
Target device:                 cortex-m4
Target device parameters:      none
Target interface:              SWD
Target interface speed:        64000kHz
Target endian:                 little

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V10 compiled Jan 30 2023 11:28:07
Hardware: V10.10
S/N: 50124859
Feature(s): GDB
Checking target voltage...
Target voltage: 3.30 V
Listening on TCP/IP port 2331
Connecting to target...
Halting core...
Connected to target
Waiting for GDB connection...
```

Figure 4.65: Connecting to SWD with J-Link

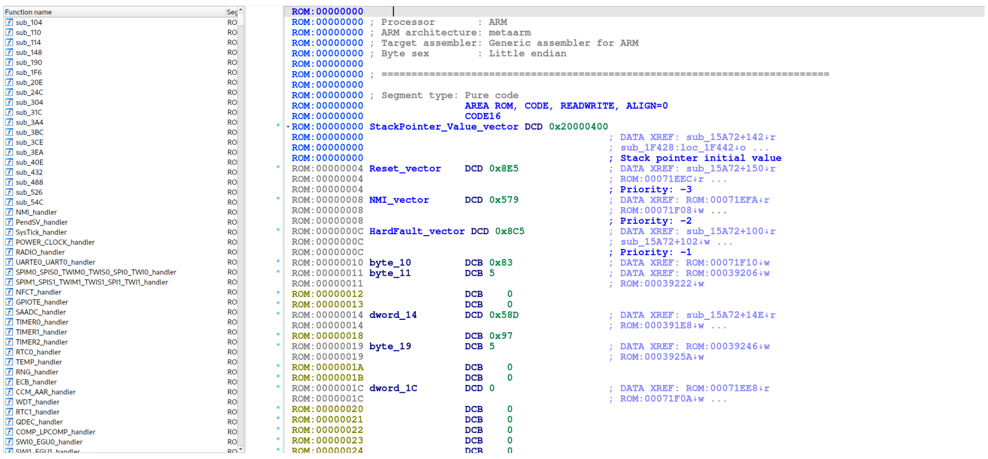
From now, it is again possible to control the CPU with SWD. With SWD, it is possible to download the data from memory. The datasheet contains information on where the code section of memory is located, and such information can be used to extract that memory.



```
J-Link>reset
Reset delay: 0 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
Reset: Halt core after reset via DEMCR.VC_CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
J-Link>SaveBin fw.bin 0x0000 0x00810000
Opening binary file for writing... [fw.bin]
Reading 8454144 bytes from addr 0x00000000 into file...O.K.
J-Link>
```

Figure 4.66: Firmware download through SWD

The memory can be again loaded into IDA and analyzed.

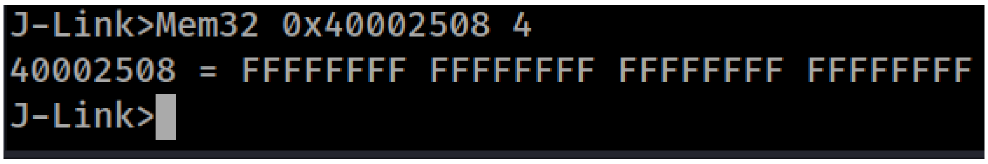


```
Function name                                   Seg
sub_104                                         RO
sub_110                                         RO
sub_114                                         RO
sub_148                                         RO
sub_190                                         RO
sub_1F6                                         RO
sub_20E                                         RO
sub_24C                                         RO
sub_304                                         RO
sub_31C                                         RO
sub_3A4                                         RO
sub_3BC                                         RO
sub_3CE                                         RO
sub_3EA                                         RO
sub_40E                                         RO
sub_432                                         RO
sub_488                                         RO
sub_526                                         RO
sub_54C                                         RO
NMI_handler                                     RO
PendSV_handler                                  RO
SysTick_handler                                 RO
POWER_CLOCK_handler                             RO
RADIO_handler                                   RO
UARTE0_UART0_handler                            RO
SPIM0_SPIS0_TWIM0_TWIS0_SPI0_TWI0_handler       RO
SPIM1_SPIS1_TWIM1_TWIS1_SPI1_TWI1_handler       RO
NFCT_handler                                    RO
GPIOTE_handler                                  RO
SAADC_handler                                   RO
TIMER0_handler                                  RO
TIMER1_handler                                  RO
TIMER2_handler                                  RO
RTC0_handler                                    RO
TEMP_handler                                    RO
RNG_handler                                     RO
ECB_handler                                     RO
CCM_AAR_handler                                 RO
WDT_handler                                     RO
RTC1_handler                                    RO
QDEC_handler                                    RO
COMP_LPCOMP_handler                             RO
SWI0_EGU0_handler                               RO
SWI1_EGU1_handler                               RO
```

```
ROM:00000000
ROM:00000000 ; Processor       : ARM
ROM:00000000 ; ARM architecture: metaarm
ROM:00000000 ; Target assembler: Generic assembler for ARM
ROM:00000000 ; Byte sex        : Little endian
ROM:00000000
ROM:00000000 ; ==================================================================
ROM:00000000
ROM:00000000 ; Segment type: Pure code
ROM:00000000                 AREA ROM, CODE, READWRITE, ALIGN=0
ROM:00000000                 CODE16
ROM:00000000 StackPointer_Value_vector DCD 0x20000400
ROM:00000000                                                ; DATA XREF: sub_15A72+142↓r
ROM:00000000                                                ; sub_1F428:loc_1F442↓o ...
ROM:00000000                                                ; Stack pointer initial value
ROM:00000004 Reset_vector    DCD 0x8E5                      ; DATA XREF: sub_15A72+150↓r
ROM:00000004                                                ; ROM:00071EEC↓r ...
ROM:00000004                                                ; Priority: -3
ROM:00000008 NMI_vector      DCD 0x579                      ; DATA XREF: ROM:00071EFA↓r
ROM:00000008                                                ; ROM:00071F08↓w ...
ROM:00000008                                                ; Priority: -2
ROM:0000000C HardFault_vector DCD 0x8C5                     ; DATA XREF: sub_15A72+100↓r
ROM:0000000C                                                ; sub_15A72+102↓w ...
ROM:0000000C                                                ; Priority: -1
ROM:00000010 byte_10         DCB 0x83                       ; DATA XREF: ROM:00071F10↓w
ROM:00000011 byte_11         DCB 5                          ; DATA XREF: ROM:00039206↓w
ROM:00000011                                                ; ROM:00039222↓w
ROM:00000012                 DCB    0
ROM:00000013                 DCB    0
ROM:00000014 dword_14        DCD 0x58D                      ; DATA XREF: sub_15A72+14E↓r
ROM:00000014                                                ; ROM:000391E8↓w ...
ROM:00000018                 DCB 0x97
ROM:00000019 byte_19         DCB 5                          ; DATA XREF: ROM:00039246↓w
ROM:00000019                                                ; ROM:0003925A↓w
ROM:0000001A                 DCB    0
ROM:0000001B                 DCB    0
ROM:0000001C dword_1C        DCD 0                          ; DATA XREF: ROM:00071EE8↓r
ROM:0000001C                                                ; ROM:00071F0A↓w ...
ROM:00000020                 DCB    0
ROM:00000021                 DCB    0
ROM:00000022                 DCB    0
ROM:00000023                 DCB    0
ROM:00000024                 DCB    0
```

Figure 4.67: Firmware analysis in IDA

As in the previous case, the configuration of UART is on the address 0x40002000. To see if UART has mapped to some of the available pins, the memory from 0x40002508 to 0x40002514.



```
J-Link>Mem32 0x40002508 4
40002508 = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
J-Link>
```

Figure 4.68: UART configuration

Like Danalock, the UART has no mapping to the pins and is, therefore, disabled.

4.3.2.4 Reporting

The identified vulnerabilties in Danapad are summarized in the following table:

Vulnerability	**CVSS**	**Mitigation**
SWD interface	3.2	Disable unnecessary hardware interfaces

Table 4.5: Table of vulnerabities in Danapad

4.4 Smart Sensors

In this section, Tesla smart smoke sensor will be described.

4.4.1 Tesla Smart Smoke Sensor

The smoke detector is a smart home sensor compatible with the smart Zigbee hub mentioned in the previous sections. It uses Zigbee for communication and has a very low consumption of energy.

4.4.1.1 Information gathering

The smoke detector consists of only one PCB.

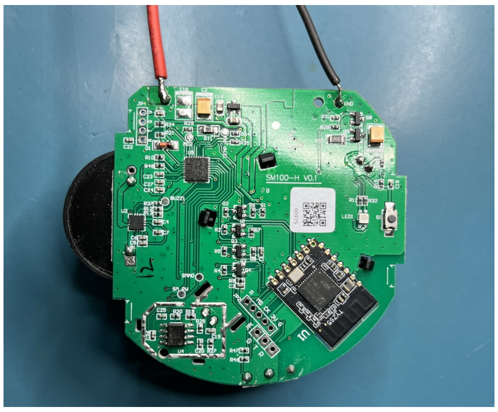


Figure 4.69: Printed circuit board

The main module is *TYZS5is.*, which is a low-powered Zigbee module. The main chip of the module is EFR32 - the Mighty Gecko Multiprotocol SoC. It contains a few pins, which are all labeled. Two could be UART based on the labels; the third could be the Joint Test Action Group (JTAG), a protocol for verifying designs and testing PCBs.

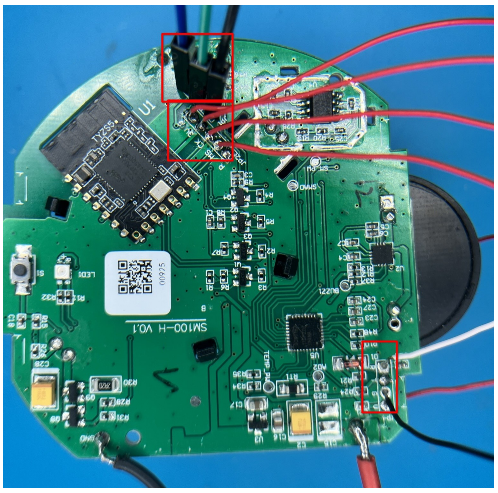
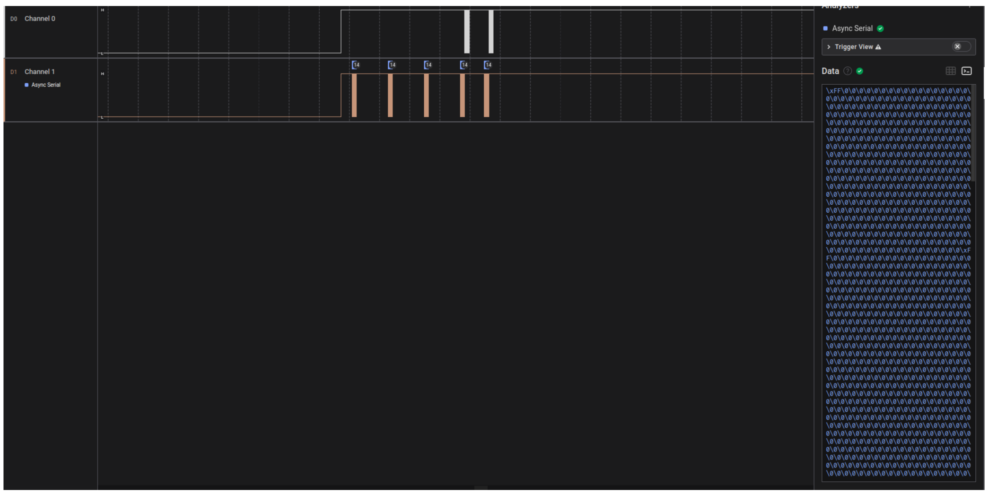


Figure 4.70: Marked interface in Tesla smoke sensor

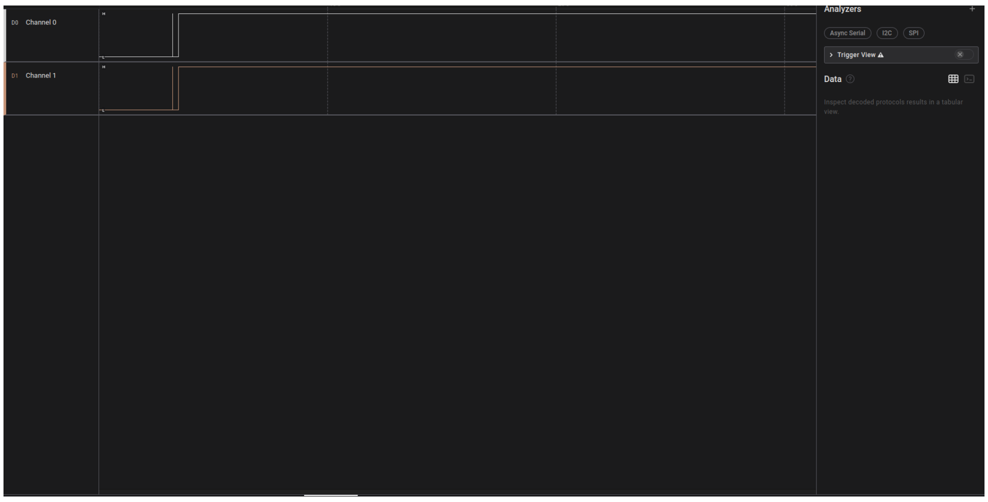
However, neither of the supposed UART interfaces has some communication. In the first case, some data are transferred, but the RX pin does not respond to any data, which could mean it is disabled.



```
\xFF\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\xF
F\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
```

Figure 4.71: First UART interface

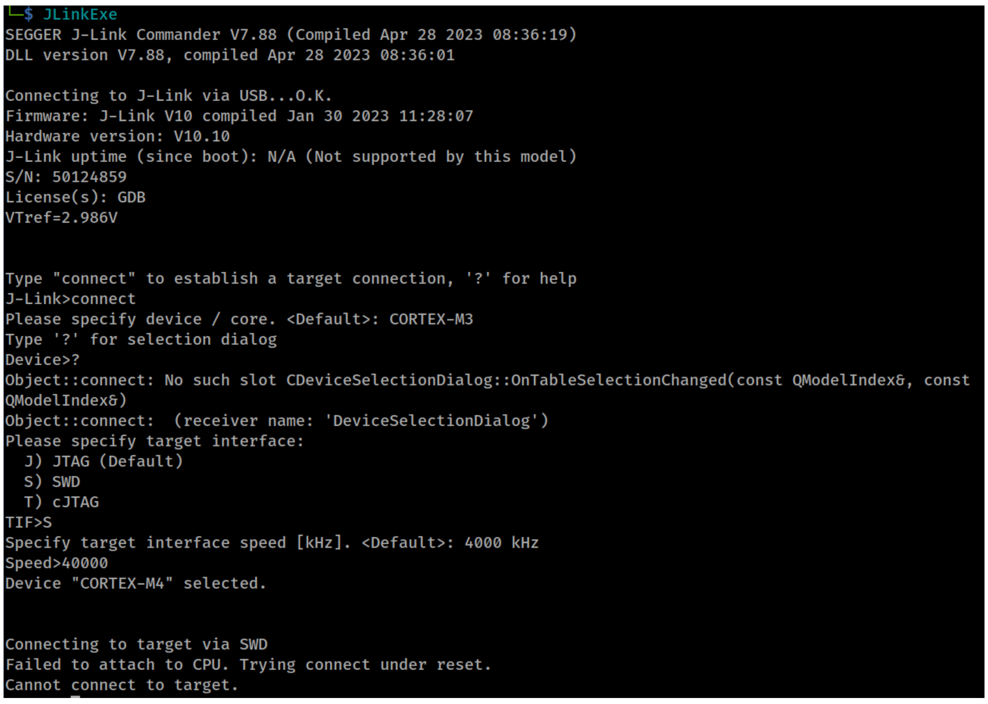
The second UART interface does not transfer data, which could mean the whole interface is disabled.



Inspect decoded protocols results in a tabular view.

Figure 4.72: Second UART interface

Typically, in the JTAG interface, there are five pins - Test Clock (TCK), Test Mode Select (TMS), Test Data Input (TDI), Test Data Output (TDO), Test Reset (TRST) [32]. However, on the PCB, there are labels only for reset pin (R), Mode select (MS), and clock (CK). The remaining pins might be located elsewhere on the PCB, so one approach is to try to find them. Another method is to use a voltmeter and identify where the pins lead. With the technique with a voltmeter, the MS and CK pins show to the TYZS5 module. For the module, the datasheet is available online. The datasheet indicates that the MS is an SWDIO pin and CK is an SWDCLK pin. The next step is identifying the processor type and frequency - all the information is available in the datasheet. The processor is ARM Cortex-M4, and the frequency is 40 MHz.



```
└─$ JLinkExe
SEGGER J-Link Commander V7.88 (Compiled Apr 28 2023 08:36:19)
DLL version V7.88, compiled Apr 28 2023 08:36:01

Connecting to J-Link via USB...O.K.
Firmware: J-Link V10 compiled Jan 30 2023 11:28:07
Hardware version: V10.10
J-Link uptime (since boot): N/A (Not supported by this model)
S/N: 50124859
License(s): GDB
VTref=2.986V


Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: CORTEX-M3
Type '?' for selection dialog
Device>?
Object::connect: No such slot CDeviceSelectionDialog::OnTableSelectionChanged(const QModelIndex&, const
QModelIndex&)
Object::connect:  (receiver name: 'DeviceSelectionDialog')
Please specify target interface:
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>S
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>40000
Device "CORTEX-M4" selected.


Connecting to target via SWD
Failed to attach to CPU. Trying connect under reset.
Cannot connect to target.
```

Figure 4.73: SWD connection failed

However, the connection fails after connecting with JLink to SWD, which can mean that the SWD and both UARTs are disabled in the production version.

Chapter 5

Discussion

The previous chapters analyzed the security of chosen smart home devices. Four of them contained vulnerabilities. In most cases, the initial vector was hardware interfaces. This chapter will compare the results to existing overviews and rankings of IoT vulnerabilities. The most notorious publicly available list of IoT vulnerabilities is OWASP's Top 10 IoT. In this list, the most common vulnerabilities are [5]:

- Weak, Guessable, or Hardcoded Passwords
- Insecure Network Services
- The lack of a Secure Update Mechanism
- The use of Insecure or Outdated Components
- Insufficient Privacy Protection
- Insecure Data Transfer and Storage
- Lack of Device Management
- Insecure Default Settings
- A lack of Physical Hardening

In all of the analyzed devices, *the lack of physical hardening* was an issue because, in all cases, it was trivial to access the PCB and analyze the board. In the category of smart cameras, both of them contained opened UART ports. Both cameras contained *insecure default settings.* In Tapo C200, the password for UART was easy to find from the public information, which can be classified as *a weak password.* Tapo C320WS had a UART shell accessible without a password, which means *no device management* exists. During the research, TP-Link released an update for the cameras, which was, however, easy to intercept, and an adversary can download the update and analyze it even before the device is updated. This shows *the lack of a secure update mechanism*. The firmware update contained an easily extractible filesystem, which contained files like *shadow* and *passwd*, resulting from *insecure data storage*. The network analysis showed an open TFPT port in Tapo C200, which did not lead to exploitation during the analysis, but it is a bad practice because it might be exploited with further research.

The smart gateway has easily accessible SWD ports for Zigbee and Wi-Fi modules, which shows the *lack of device management*. As a result, an adversary can then exploit insecure data storage, as the SWD allows reading the memory, and the password for Wi-Fi is stored in the RAM section.

The smart locks also had an accessible SWD port, which was more challenging to find but still manageable due to the *lack of device management*. The attacker can read the memory and write to the memory.

The following table summarizes the comparison between discovered vulnerabilities and OWASP IoT Top 10 and its mitigations:

OWASP IoT Top 10	**Vulnerability**	**Mitigation**
The lack of physical hardening	Easy access to PCB	Device design, where disassemble can damage PCB
The lack of device management	Open interfaces such as SWD or UART	Disable interfaces or restrict their functionality
The weak passwords	Weak password or no password used for UART	Password for critical interface should be cryptographically strong
The lack of a secure update mechanism	Intercepted update of firmware	Update mechanism uses secure mechanism like mTLS
The insecure data storage	Credentials stored in the accessible parts of memory or firmware	Store credentials in secure or encrypted part of memory

Table 5.1: Table of comparison between vulnerabilities and OWASP IoT Top 10

While writing this thesis, research took place simultaneously—the research aimed to classify IoT vulnerabilities and propose a new list of the most common vulnerabilities. As the list consists of research on IoT vulnerabilities from recent years, the results from this thesis will also be compared to the newly proposed list. The the thesis identified the following vulnerabilities as the most common in IoT [33]:

- Overflow
- Improper use of memory
- Access control problem
- Execution of malicious code
- Problematic password management
- Problematic authentication/session handling
- Improper input validation
- Denial of service
- Problematic cryptography/certificate manipulation
- Insecure design/design flaw

In the category of smart cameras, the UART interfaces can be classified as an *access control problem*, as they are easily accessible and either use weak passwords or no password in the default state. *Problematic password management* leads to accessible hashes in filesystems.

The SWD interfaces in smart gateways can be classified as *insecure design*, as the interface is not disabled in the production version. The access control problem allows the attacker to read the Wi-Fi password from memory through the SWD interface.

The smart locks also suffer from *insecure design*, as the SWD port on Danalock and Danapad is accessible.

One of the leading security issues is insecure design or lack of device management. In most cases, it meant access to hardware interfaces and protocols, which provided control over the device or the possibility to further exploitation. As a possible solution, manufacturers can disable the interfaces not needed in production devices or secure them enough, making it challenging for an adversary to access them. For the UART interface, they can be either disabled or protected with a strong password. In the case of the SWD interface, there can be mitigations in the form of restricted access to the memory or disabling it entirely if it is not needed. As for the password to the UART shell in Tapo C200, the mitigation of weak passwords is to use cryptographically strong passwords accessible only to vendors who need them. The insecure update mechanism in TP-Link cameras allows an adversary to intercept communication between the phone and target devices, and therefore, they can download and analyze the firmware. The update mechanism can be secured by receiving the information about the update from the cloud and downloading the update from the cloud. That way, an adversary has a more complicated scenario to intercept the communication. Protocols such as Mutual TLS (mTLS) can be implemented to make the interception even more challenging. A lack of physical hardening allows an adversary to access and examine the PCBs of the device. It is difficult for IoT devices to harden devices, so it would be challenging to access PCB. Nevertheless, it is possible to implement a secure mechanism against manipulation and access to the PCBs. Furthermore, the passwords for critical services should be stored securely and hashed with a robust hashing algorithm. The following table summarizes the compared vulnerabilities and its mitigations:

IoT List	**Vulnerability**	**Mitigation**
Access control problem	Insecure settings in UART/readable RAM through SWD	Restrict fuctionality or implement strong password protection
Problematic password management	Extractable user hashes from firmware	Usage of encrypted firmware
Insecure design	Accessible SWD interface	Disable unnecessary interface in production

Table 5.2: Table of common vulnerabilities

Conclusion

The goal of the thesis was mainly to analyze the current security state in smart home devices and compare the results with the existing list of IoT vulnerabilities. Chapter 4 introduced the categorization of smart home devices based on the potential value for an adversary and the selection of the devices from the categories. The selected devices were analyzed in Chapter 5. Chapter 6 compares the results with OWASP IoT Top 10 and a list of IoT vulnerabilities from Exploring Vulnerabilities of the Internet of Things Devices [5]. The analysis uncovered some common vulnerabilities among smart home devices - a lack of physical hardening, insecure default settings, secure update mechanisms, device management, or insecure data storage. The vulnerabilities allowed an attacker either to compromise the device or to access the information, which helps to analyze the device further. The analysis followed the structure of minipentest, which aims to explore a target quickly and uncover relatively easy exploitation. The research results in a review of the current state of smart home devices. The goals of the thesis are, therefore, fulfilled. The mitigations of vulnerabilities can serve as a fundamental source of information for manufacturers and customers. The thesis results can provide information for manufacturers to avoid implementing vulnerabilities into their devices, which are easy to exploit. Security researchers can also benefit from the results, as the methods in the analysis can help with small-scale penetration testing of IoT devices. However, it is still necessary to examine devices from selected categories more deeply, as it may uncover complex but severe vulnerabilities. The future progress of the research is to analyze the devices more deeply and publish the full results as an article.

Bibliography

[1] Farooq, M. U.; Waseem, M.; et al. A review on internet of things (IoT). *International journal of computer applications*, volume 113, no. 1, 2015: pp. 1–7.

[2] Schiller, E.; Aidoo, A.; et al. Landscape of IoT security. *Computer Science Review*, volume 44, 2022: p. 100467, ISSN 1574-0137, doi: https://doi.org/10.1016/j.cosrev.2022.100467. Available from: `https://www.sciencedirect.com/science/article/pii/S1574013722000120`

[3] Khan, R.; Khan, S. U.; et al. Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges. In *2012 10th International Conference on Frontiers of Information Technology*, 2012, pp. 257–260, doi:10.1109/FIT.2012.53.

[4] Zhou, W.; Jia, Y.; et al. The effect of iot new features on security and privacy: New threats, existing solutions, and challenges yet to be solved. *IEEE Internet of things Journal*, volume 6, no. 2, 2018: pp. 1606–1616.

[5] OWASP Top 10 IoT 2018. 2018, `https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf` [Accessed: (21.4 2023)].

[6] NIST. NVD - CVE-2021-35394. 2021, `https://nvd.nist.gov/vuln/detail/CVE-2021-35394`[Accessed: (26.4 2023)].

[7] Milan Franik, M. C. Serious flaws found in multiple smart home hubs: Is your device among them? — WeLiveSecurity. 2020, `https://www.welivesecurity.com/2020/04/22/serious-flaws-smart-home-hubs-is-your-device-among-them/`[Accessed: (26.4 2023)].

[8] Malware exploited critical Realtek SDK vulnerability. 2023, `https://www.quorumcyber.com/threat-intelligence/malware-`

exploited-critical-realtek-sdk-vulnerability-in-millions-of-attacks/[Accessed: (26.4 2023)].

[9] Antonakakis, M.; April, T.; et al. Understanding the mirai botnet. In *26th {USENIX} security symposium ({USENIX} Security 17)*, 2017, pp. 1093–1110.

[10] Chantzis, F.; Stais, I.; et al. *Practical IoT Hacking: The Definitive Guide to Attacking the Internet of Things*. No Starch Press, 2021, ISBN 9781718500914. Available from: https://books.google.cz/books?id=GbYEEAAAQBAJ

[11] Hassija, V.; Chamola, V.; et al. A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures. *IEEE Access*, volume 7, 2019: pp. 82721–82743, doi:10.1109/ACCESS.2019.2924045.

[12] The Penetration Testing Execution Standard. 2014, http://www.pentest-standard.org/index.php/Main_Page [Accessed: (21.4 2023)].

[13] NVD - Vulnerability Metrics. Available from: https://nvd.nist.gov/vuln-metrics/cvss

[14] OWASP Foundation, the Open Source Foundation for Application Security — OWASP Foundation. 2023, https://owasp.org/ [Accessed: (21.4 2023)].

[15] Chougule, S. IoT Device Penetration Testing. 2019, https://owasp.org/www-chapter-pune/meetups/2019/August/IoT_Device_Pentest_by_Shubham_Chougule.pdf [Accessed: 21.4 2023].

[16] Team, E. PCB test points. 2022, https://www.pcbdirectory.com/community/what-are-pcb-test-points[Accessed: (21.4 2023)].

[17] van Woudenberg, J.; O'Flynn, C. *The Hardware Hacking Handbook: Breaking Embedded Security with Hardware Attacks*. No Starch Press, 2021, ISBN 9781593278748. Available from: https://books.google.cz/books?id=DEqatAEACAAJ

[18] Documentation – Arm Developer. 2023, https://developer.arm.com/documentation/ihi0031/a/The-Serial-Wire-Debug-Port--SW-DP-/Introduction-to-the-ARM-Serial-Wire-Debug--SWD--protocol[Accessed: (21.4 2023)].

[19] ARM Programming. https://learn.sparkfun.com/tutorials/arm-programming/jtag-and-swd[Accessed: (21.4 2023)].

[20] Dang, B.; Gazet, A.; et al. *Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation.* Wiley, 2014, ISBN 9781118787311. Available from: `https://books.google.cz/books?id=YSrbAgAAQBAJ`

[21] Martin, Š. *Bezpečnostní analỳza bezklíčového vstupu Tesla Model 3.* B.S. thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2021.

[22] Leonardi, L.; Patti, G.; et al. Multi-Hop Real-Time Communications Over Bluetooth Low Energy Industrial Wireless Mesh Networks. *IEEE Access*, volume PP, 05 2018: pp. 1–1, doi:10.1109/ACCESS.2018.2834479.

[23] Staff, D. R. Popular IOT cameras need patching to fend off catastrophic attacks. Sep 2022. Available from: `https://www.darkreading.com/attacks-breaches/popular-iot-cameras-patching-catastrophic-attacks`

[24] Available from: `https://www.alza.cz/tp-link-tapo-c200-d5703789.htm`

[25] Available from: `https://www.alza.cz/tp-link-tapo-c320ws-outdoor-home-security-wi-fi-camera-d7269261.htm`

[26] Available from: `https://www.alza.cz/danalock-v3-chytry-zamek-bez-cylindricke-vlozky-bluetooth-d5088429.htm`

[27] Available from: `https://www.alza.cz/danalock-danapad-v3-elektronicky-kodovy-zamek-d5525556.htm`

[28] Wireless Smart Lock Makers React to security scares. Aug 2018. Available from: `https://www.nordicsemi.com/News/2018/08/Wireless-smart-lock-makers-react-to-security-scares`

[29] MIPS architecture - a streamlined, highly scalable RISC architecture. Aug 2022. Available from: `https://www.mips.com/products/architectures/`

[30] WRG1 module Datasheet. Available from: `https://developer.tuya.com/en/docs/iot/wrg1-datasheet?id=K97rig6mscj8e`

[31] Available from: `https://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.4.pdf`

[32] Mohieldin, S. Hardware hacking 101: Introduction to JTAG. May 2021. Available from: `https://www.riverloopsecurity.com/blog/2021/05/hw-101-jtag/`

[33] Tropková, Z. *Exploring Vulnerabilities of the Internet of Things Devices.* Master's thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2023.

Appendix A

Acronyms

IoT Internet of Things

RFID Radio Frequency Identification

IIoT Industrial Internet of Things

IoMT Internet of Medical Things

MiTM Man-in-the-middle

PTES Penetration Test Execution Standard

OWASP Open Worldwide Application Security Project

PCB Printed circuit board

UART Universal asynchronous receiver-transmitter

SPI Serial Peripheral Interface

EEPROM Electrically Erasable Programmable Read-Only Memory

OS Operating system

BLE Bluetooth Low Energy

SWD Serial Wire Debug

RISC Reduced instruction set computing

SDR Software-defined radio

GAP Generic Attribute Profile

RTPS Real-Time Publish-Subscribe

LFI Local File Inclusion

JSON	JavaScript Object Notation
SP	Stack Pointer
JTAG	Joint Test Action Group
TCK	Test Clock
TMS	Test Mode Select
TDI	Test Data Input
TDO	Test Data Output
TRST	Test Reset
mTLS	Mutual TLS
SOC	System-on-chip

Appendix B

Contents of enclosed CD

- `readme.txt` .... the file with CD contents description
- `artifacts` .... the directory with artefacts from analysis
  - `Tapo_C200` .... the directory with artefacts for Tapo C200
    - `nmap_TCP_scan.nmap` .... output of Nmap scan of TCP ports
    - `nmap_UDP_scan.nmap` .... output of Nmap scan of UDP ports
    - `passwd` .... passwd file from Tapo C200 filesystem
    - `uhttpd` .... binary running HTTPS server on Tapo C200
  - `Tapo_C320WS` .... the directory with artefacts for Tapo C320WS
    - `nmap_TCP_scan.nmap` .... output of Nmap scan of TCP ports
    - `nmap_UDP_scan.nmap` .... output of Nmap scan of UDP ports
    - `passwd` .... passwd file from Tapo C320WS filesystem
    - `uhttpd` .... binary running HTTPS server on Tapo C320WS
  - `Tesla_Zigbee_Hub` .. the directory with artefacts for Tesla Zigbee Hub
    - `firmware_wrg1.bin` .... firmware for WRG1 module
  - `Danalock` .... the directory with artefacts for Danalock
    - `GATT_service_enumeration.json` .. Enumeration of GATT service
    - `firmware.bin Firmware from Danalock`
  - `Danapad` .... the directory with artefacts for Danapad
    - `firmware.bin Firmware from Danapad`
- `src` .... the directory of source codes
  - `thesis` .... the directory of LaTeXsource codes of the thesis
- `text` .... the thesis text directory
  - `thesis.pdf` .... text práce ve formátu PDF