



TECHNISCHE
UNIVERSITÄT
DARMSTADT

SECURITY ANALYSIS OF THE XIAOMI IOT ECOSYSTEM

DENNIS GIESE

Master Thesis

July 10, 2019

Secure Mobile Networking Lab
Department of Computer Science



Security Analysis of the Xiaomi IoT Ecosystem
Master Thesis
SEEMOO-MSc-0142

Submitted by Dennis Giese
Date of submission: July 10, 2019

Advisor: Prof. Dr.-Ing. Matthias Hollick
Supervisor: Jiska Classen and Prof. Guevara Noubir

Technische Universität Darmstadt
Department of Computer Science
Secure Mobile Networking Lab

ABSTRACT

Internet of Things (IoT) devices are gaining more and more importance in our daily lives. Through their deep integration they pose a potential risk for the user's privacy. In this thesis, I use reverse engineering methods to analyze the security of the Xioami IoT ecosystem and its devices. I implement a tool to emulate the Xiaomi cloud and analyze the cloud protocol. I use different, some unconventional, methods to extract the device firmware and get privileged access on the devices. The evaluation shows that, even though Xiaomi is a large IoT company, their cloud protocol and Software Development Kit (SDK) have serious flaws. Also, the actual vendors of the devices do not put much effort into device security. A slightly positive aspect of this is the fact that users can use the flaws to get full control over their own devices.

ZUSAMMENFASSUNG

IoT Geräte gewinnen mehr und mehr an Bedeutung in unserem täglichen Leben. Durch ihre tiefe Integration stellen sie für die Privatsphäre des Benutzers ein potentielles Risiko dar. In dieser Thesis untersuche ich mit Hilfe von "reverse engineering" die Sicherheit des Xiaomi IoT Ökosystems. Dazu habe ich eine Software entwickelt, die die Xiaomi Cloud emuliert und mir so die Analyse des Cloud Protokolls ermöglicht. Ich habe verschiedene, teils unkonventionelle, Verfahren verwendet um die Firmware aus den Geräten zu extrahieren und um dann priviligierte Zugriff auf diese zu bekommen. Die Evaluation meiner Analyse zeigt, dass Xiaomi, obwohl es ein großes und bekanntes Unternehmen im Bereich IoT ist, schwerwiegende Schwachstellen in ihrem Cloud Protokoll und ihrem SDK aufweist. Dazu kommt noch, dass die eigentlichen Hersteller der Geräte nicht viel Aufwand in die Absicherung dieser investieren. Ein positiver Aspekt ist jedoch, dass Benutzer diese Schwachstellen ausnutzen können, um die Kontrolle über ihre eigenen Geräte zu erhalten.

ACKNOWLEDGMENTS

I would like to thank Prof. Guevara Noubir and Jiska Classen for giving helpful advice while writing this thesis. Additionally, I am thankful to Daniel Wegemer for his support in the reverse-engineering of devices.

I am especially grateful to Prof. Matthias Hollick for his patience and advice.

I would like to thank Tristan Honscheid, Zachary Holbrook, Josef Biberstein, Marc Studlek and Sashank Narain for proofreading my thesis.

CONTENTS

I	INTRODUCTION	1
1	INTRODUCTION	3
1.1	Motivation: IoT in our daily life	3
1.2	Universal definition for IoT	3
1.3	Security incidents	4
1.4	Privacy concerns	5
1.5	IoT devices as Embedded devices	5
1.6	Requirements for a secure IoT system	7
1.7	Secure devices vs. user control	9
1.8	Focus on Xiaomi	9
1.9	Scope of this Thesis	10
2	RELATED WORK	13
II	CONTRIBUTIONS	17
3	XIAOMI ECOSYSTEM	19
3.1	Approach	19
3.2	Structure	20
3.3	Device API	22
3.4	App to Cloud API	25
3.5	Interception of cloud traffic: Dustcloud	26
3.6	Devices	27
4	REVERSE ENGINEERING METHODS	29
4.1	Network	29
4.2	Hardware	30
4.3	Software	33
4.4	Summary of tools	34
5	EVALUATION OF DEVICES	35
5.1	Comparison guideline	35
5.1.1	Hardware	35
5.1.2	Firmware security	35
5.1.3	Network	35
5.1.4	Data	36
5.2	Device analysis	36
5.2.1	Xiaomi Mi Robot Vacuum Cleaner Gen1	36
5.2.2	Roborock Vacuum Cleaner S50	45
5.2.3	Roborock Vacuum Cleaner S6/T6	47
5.2.4	Lumi Smart Home Gateway	52
5.2.5	Yeelight LED Strip	57
5.2.6	Yeelight Smart Light bulbs	59
5.2.7	Lumi Aqara Gateway Camera	60
5.3	Analysis results	62
5.4	Possible Attacks	62

III DISCUSSION AND CONCLUSIONS	65
6 DISCUSSION	67
6.1 The Used Approaches for Reverse Engineering	67
6.2 Security of the Xiaomi Ecosystem	68
6.3 Possibility of custom firmware	69
6.4 Future work	70
7 CONCLUSIONS	71
BIBLIOGRAPHY	73

LIST OF FIGURES

Figure 1	Overview of components of an IoT/embedded device	6
Figure 2	Simplified communication relations in the Mijia ecosystem	21
Figure 3	Detailed overview of components for the device communication	25
Figure 4	Example network setup to monitor IoT device traffic	30
Figure 5	Accessing the UART pins through venting holes in the case	31
Figure 6	Disassembled Mi Robot Vacuum Cleaner	37
Figure 7	Front side of the mainboard Printed Circuit Board (PCB) for the Mi Robot Vacuum Cleaner	38
Figure 8	Reverse Engineered Pinout of the R16 SOC in the Mi Robot Vacuum Cleaner	40
Figure 9	Sideview of the R16 SOC with dimensions from the R16 Datasheet [40]	41
Figure 10	Screenshot of IDA Pro inspecting the Strings of the SysUpdate binary	43
Figure 11	Method of soldering the UART pins on Mi Robot Vacuum Cleaner	44
Figure 12	Stored map (scaled) on Mi Robot Vacuum Cleaner	46
Figure 13	Disassembled Roborock Vacuum Cleaner S50	48
Figure 14	Front side of the mainboard PCB for Roborock Vacuum Cleaner S50	49
Figure 15	Setup of T6 Mainboard for Universal Asynchronous Receiver Transmitter (UART) connection	50
Figure 16	Front side of the PCB in Lumi Smart Home Gateway	54
Figure 17	Partition information extracted from SPI flash of Lumi Smart Home Gateway	55
Figure 18	Debug pins on PCB in Lumi Smart Home Gateway	56
Figure 19	Placeholders for device credentials in firmware of Lumi Smart Home Gateway	57
Figure 20	Matched functions between SDK libraries and unknown functions in firmware of Lumi Smart Home Gateway	58
Figure 21	Front side of the PCB in disassembled Yeelight LED Strip	59
Figure 22	Disassembled Yeelight Smart Lightbulb	60
Figure 23	Disassembled Aqara Gateway Camera with Back-(left) and Frontside PCB (right)	61

LIST OF TABLES

Table 1	Stored configuration settings for Mijia devices	22
Table 2	miIO protocol structure	23
Table 3	Mijia devices supported by region	27

Table 4	Overview over Mijia devices to be tested	28
Table 5	Overview over prepared tools for reverse engineering	34
Table 6	eMMC partition layout for Mi Robot Vacuum Cleaner	41
Table 7	Responsible processes running on Mi Robot Vacuum Cleaner . .	42
Table 8	Responsible processes running on Roborock T6 Vacuum Cleaner	50
Table 9	Reverse engineered format of Marvel 88MW30x firmware binaries	53
Table 10	Partitions on SPI flash of Aqara Gateway camera	62
Table 11	Hardware details of the analyzed devices	63
Table 12	Security details of the analyzed devices	64

LISTINGS

Listing 1	Example of a device entry of supported devices	19
Listing 2	Example of a message from the device to the cloud	23
Listing 3	Example of cloud command for installing an App firmware update	24
Listing 4	Example of cloud command for installing an MCU firmware update	24
Listing 5	Excerpt of a cloud response of the user's device list	26
Listing 6	Half of the obfuscated public key in AppProxy	51

ACRONYMS

ADB	Android Debug Bridge
API	Application Programming Interface
AWS	Amazon Web Services
BGA	Ball Grid Array
BLE	Bluetooth Low Energy
BSI	German Federal Office for Information Security
CBC	Cipher Block Chaining
CDN	Content Delivery Network
DID	Device ID
DKEY	Device Key
DNS	Domain Name System
ECC	Error-Correcting Code
EMI	Electromagnetic Interference
eMMC	Embedded Multimedia Card
FACT	Firmware Analysis and Comparison Tool
FDS	Xiaomi File Storage Service
GPIO	General-Purpose Input/Output
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
ISP	In-System Programming
IV	Initialization Vector
JSON	JavaScript Object Notation
JTAG	Joint Test Action Group

LIDAR	light detection and ranging
MAC	Medium Access Control
MCU	Microcontroller Unit
MITM	Man-in-the-middle
NDA	Non-Disclosure Agreement
NIST	US National Institute of Standards and Technology
NTP	Network Time Protocol
ODM	Original Design Manufacturer
OEM	Original Equipment Manufacturer
OS	Operating System
OTA	over-the-air
OTP	One-Time-Programmable
OWASP	Open Web Application Security Project
PCB	Printed Circuit Board
RPC	Remote Procedure Call
RTOS	Real-Time Operating System
RTSP	Real-Time Streaming Protocol
SD	Secure Digital
SDIO	Secure Digital Input Output
SDK	Software Development Kit
SOC	System-on-a-Chip
SOHO	Small Office / Home Office
SOP	Small Outline Package
SPI	Serial Peripheral Interface
SSH	Secure SHell
SWD	Serial Wire Debug
TLS	Transport Layer Security
TSOP	Thin Small Outline Package
UART	Universal Asynchronous Receiver Transmitter
URL	Uniform Resource Locator

USB Universal Serial Bus

WLAN Wireless Local Area Networking

Part I

INTRODUCTION

The first chapter of this part gives an introduction and a motivation to this thesis. This is followed in the second chapter by a presentation of related work found in the area of IoT device security.

INTRODUCTION

In this chapter, I will offer an introduction to the term Internet of Things (IoT) and embedded devices in general. I will discuss several security incidents involving IoT devices and provide recommendations for securing such systems. Next, I will introduce Xiaomi, a large IoT ecosystem vendor. Finally, I present the scope of this thesis.

1.1 MOTIVATION: IOT IN OUR DAILY LIFE

IoT devices are gaining more and more impact on our daily lives. For example, most consumer electronics like TVs, fridges, washing machines, smart speakers, surveillance cameras and even toothbrushes are equipped with some kind of smart functionality and Internet connectivity. A 2017 study by Consumer Technology Association (CTA) [10] estimated that there were at least 273 million IoT devices installed in US households at that time (excluding smartphones and wearable electronics).

Smart devices are also prevalent in new buildings. The lighting, heating and security systems of buildings of all sizes, from commercial high rises to small residential buildings, are often regulated by smart devices. There are a variety of different technologies, protocols and vendors for building automation. These building automation systems remain installed for extended periods of time, possibly forever, without being replaced or even updated.

Internet connectivity is gaining more importance in Industrial automation. In times of real-time production and tight supply management requirements, IoT in industrial settings is hailed as the answer to these challenges. These systems are often regarded as Industry 4.0 –the fourth Industrial Revolution [25].

To be able to plan and control the usage of the power grids and other infrastructure, there is a worldwide movement to deploy smart devices. In the US, over 72 million smart electricity meters were installed by December 2016 alone [12]. Most of these devices have to communicate back to the electricity providers.

One of the challenges of IoT is the wide fragmentation of ecosystems due to different chip vendors, ecosystems operators, and integrators [8]. This leads to a multitude of problems, including a lack of interoperability and security vulnerabilities. Challenges like this make this topic interesting for security researchers.

1.2 UNIVERSAL DEFINITION FOR IOT

There is no universally-accepted definition IoT and, in turn, which devices and systems fall under the term "IoT" and which do not. In many publications, smartphones are in-

cluded under the IoT umbrella, as they connect to the Internet and contain sensors and actuators. As the topic of IoT is regarded as new, national standardization authorities have only recently begun publishing definitions and technical documents addressing, for instance, security considerations or trust issues.

A comprehensive definition from the International Telecommunications Union (ITU) can be found in its July 2012 approved Recommendation ITU-T Y.2060, Overview of the Internet of Things, which defines IoT as "a global infrastructure for the Information Society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies" [31].

The US National Institute of Standards and Technology (NIST) writes in their 2016 Special Publication (SP) [58] that IoT devices, in general, are systems that involve computation, sensing, communication, and actuation. They form a connection between humans, non-human physical objects, and cyber objects, enabling monitoring, automation, and decision making. This definition is very open and enables us to dynamically decide if something belongs to IoT depending on a particular behavior.

The German Federal Office for Information Security (BSI), in their 2016 drafted security recommendation [48] defines IoT as systems, which have, in contrast to classical IT systems, additional smart functionality. These devices often have Internet access and can communicate over different media, e.g. wirelessly. The definition of the BSI is also very open and encompassing. Especially the transition line to industrial control systems (ICS) or traditional embedded systems is regarded as not static. The BSI regards IoT devices as specialized embedded devices.

For this thesis, I limit the scope of my devices to those that fit the definition of the BSI and exclude smartphones.

1.3 SECURITY INCIDENTS

Vulnerable IoT devices have led to numerous security incidents.

One of the most famous recent attacks was the "Mirai" botnet. In 2016 the Mirai malware was used to take over vulnerable IoT devices and subsequently attack multiple websites [7]. This botnet reached its peak in December 2016 with over 600,000 active devices at the same time. Leveraging hard-coded default passwords and open ports, the bot scans the Internet for vulnerable devices, in particular those with open ports 23 and 2323, which are used for telnet. Its primary targets were IP cameras, routers, and printers. Part of the vast success of Mirai was not only the insecure design of the devices by their vendors, but also the behavior of customers, who connected the devices directly to the Internet without firewalls. The anonymous developer of the "Carna" botnet, which mapped significant parts of the Internet using hacked devices in 2012 found over 500,000 accessible printers and over 1 million accessible IP cameras [6].

The Mirai botnet was, however, not the only malware which abused vulnerable IoT devices. One year before Mirai, in 2015, a malware named "BASHLITE" attacked and infected many devices [36]. It is estimated that BASHLITE infected over 1 million devices. The primary attack vector of this malware was the BASH vulnerability called "Shellshock", which was identified in September 2014.

In the preceding cases, the owners of the infected devices frequently did not notice that their devices had been hacked. However, there are also cases where vulnerable IoT devices have led to severe consequences for their owners: In 2014 the "SynoLocker" malware infected many Synology Network Attached Storage (NAS) devices and encrypted their users' data [53]. The malware then demanded ransom to decrypt the data again. The attack vector for the malware was a vulnerability in the firmware of the NAS and the fact that many owners made the devices accessible from the Internet. The same repeated in the beginning of 2019 where the malware "Cr1ptTor" infected vulnerable D-Link NAS devices [13].

1.4 PRIVACY CONCERNS

IoT devices, by their very nature, can collect much data. In the past, there have been many cases where IoT devices have put the privacy of their users at risk.

Smart TVs, for example, can monitor the user's behavior and can report this data to the vendor or to advertisers [18]. This has been found in European models of Smart TVs. In addition, these devices usually store this data locally on the device.

IP cameras have a huge impact on the privacy of the users, especially as users place cameras in private areas like their homes or even their bedrooms. If such a camera is improperly configured or vulnerable, unauthorized persons can watch the owner and listen to conversations. Since 2014 the website "INSECAM"¹ has scanned the Internet for open or misconfigured cameras and made them accessible. After starting their service, the operators of the website started to filter out private cameras. While the ethical aspect of this website is questionable, it still demonstrates the privacy problem and the fact that anyone can access that kind of cameras.

A particularly interesting case was published in June 2019 about second-hand Google's Nest cameras where the new owner could be monitored by the old owner, even though a factory reset according to the user manual was done [9]. As Google stated, the issue was a binding to a third-party cloud service was not correctly removed. In that case, the old owner could access video streams, see live video and control most functions of the camera remotely, without the new owner noticing it.

1.5 IOT DEVICES AS EMBEDDED DEVICES

IoT devices are technically a subgroup of embedded devices. Typical embedded devices are everywhere around us and can be found in consumer, building, industrial, automotive, medical, commercial and military applications. In comparison to usual general-purpose computers, embedded devices are designed for a specific task (e.g. to control some actuator depending on specific conditions).

¹ INSECAM <https://www.insecam.org/>

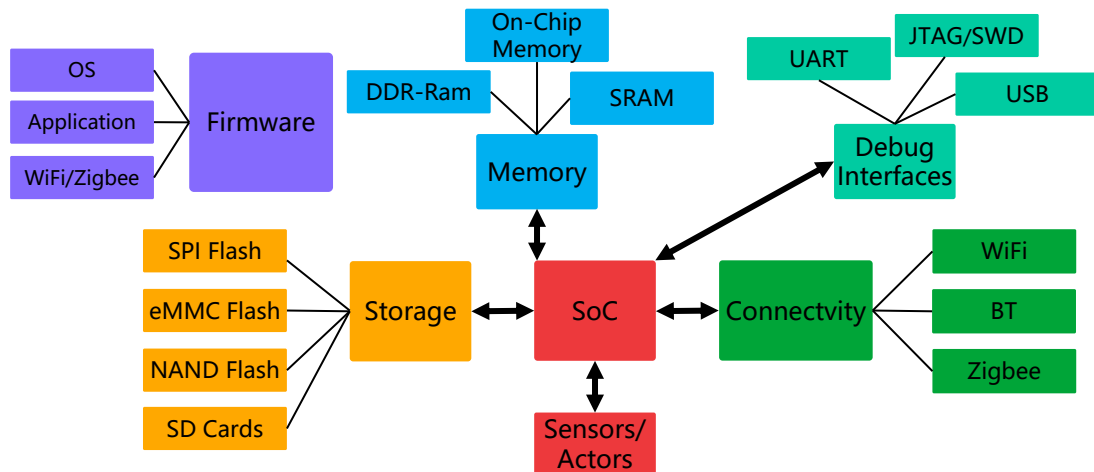


Figure 1: Overview of components of an IoT/embedded device

Figure 1 shows the typical components of embedded and IoT devices. The central component is the System-on-a-Chip (SOC). This SOC can come in different flavors and architectures.

Typical architectures for embedded systems are ARM, MIPS, Xtensa, PIC and Atmel AVR. The choice of a particular architecture depends mostly on the use-case and the developer's experience. For example, according to the OpenWRT supported hardware list², the majority of home user and Small Office / Home Office (SOHO) routers are based on MIPS. The majority of customer entertainment electronic, including most Smartphones, is based on ARM devices. Xtensa, mostly known for the ESP8266 and ESP32, can be found in devices like lightbulbs, switches or power plugs.

The SOC offers different interfaces: debug interfaces, storage interfaces, connectivity interfaces, and general purpose interfaces.

Debug interfaces are for example Joint Test Action Group (JTAG), Serial Wire Debug (SWD), Universal Asynchronous Receiver Transmitter (UART) or Universal Serial Bus (USB).

The JTAG interface is a standardized interface for testing and debugging. Often this interface is used to initially provision a firmware image for the SOC. SWD is a similar interface like JTAG, however it can be only found on ARM devices. SWD requires fewer wires than JTAG and supports ARM-specific features. Often, both interfaces share pins and either may be selected. Due to the importance of JTAG/SWD for the production process, it can be found on most Printed Circuit Boards (PCBs).

UART is a serial interface, which is used to print debug messages or as a command input.

² OpenWRT supported hardware list https://openwrt.org/toh/views/toh_available_864

The USB interface can play a dual role. It can be used as a General Purpose interface for peripherals (host mode) or as a way to connect to the SOC (device mode). For debug purposes, the device mode is used. Typical protocols which are used for the device mode are "fastboot", "adb" or some proprietary protocols. Examples for proprietary protocols are FEL for Allwinner [16] or Maskrom mode for Rockchip [45].

The SOC offers various interfaces for non-volatile storage. For small sized storage (<128 MByte) Serial Peripheral Interface (SPI) flash is often used. For higher amounts of storage, raw NAND, NOR, or Embedded Multimedia Card (eMMC) flash is used, depending on the speed and cost requirements. Also, SD cards can be used as storage.

One of the most distinguishing features of IoT devices, compared to other embedded devices, is their connectivity to the Internet or other devices. Most common is Wireless Local Area Networking (WLAN) or Bluetooth Low Energy (BLE) connectivity [29][42]. Some devices also offer connectivity via ZigBee [30]. Depending on the SOC, the radios can be integrated in the chip itself or on a separate module. These modules can be connected via the USB, SPI bus or even UART.

The functionality of the IoT/embedded device is provided by firmware. There are multiple kinds of firmware that can be stored on the device. The most important is the Operating System (OS) with the applications, but it is also common for separate firmware images for WLAN or BLE functionality to be packaged alongside.

1.6 REQUIREMENTS FOR A SECURE IOT SYSTEM

The BSI created base requirements for IoT devices. These requirements apply not only to the IoT device itself, but also on its operation. While the device security is the responsibility of the vendor, the actual operation is depending on the user [48]. Also, the requirements are mainly focused on the usage of IoT in the context of companies, and are, for example, mandatory for specific certifications. Still, the rules make sense for private consumers. The requirements can be summarized in a few categories: update mechanisms, authentication, network, services, and life-cycle.

It is required that devices have an update function and that vendors offer a process for these updates. Auto-update mechanisms must be enabled and new updates must be checked for on a regular basis, such as daily. If security patches are available, then they must be updated immediately. It must be verified that the updates are only coming from trustworthy sources [48][A3,A4].

The devices need to allow only authenticated access. The credentials must not be hard-coded and default passwords must be changed. It is recommended to use certificate-based authentication if possible [48][A2].

Regarding the network, the BSI guideline demands that the network traffic is limited to a minimum and is controlled. For example, routers and firewalls should only allow the necessary connections and detect suspicious traffic. If possible, IoT devices and sensors should operate in their own network segment, which is separated from different networks [48][A5]. As for the traffic itself, it is recommended to use encryp-

tion [48][A10]. Administration interfaces must especially be protected. Interestingly, the BSI recommends that companies avoid cloud-based devices.

As IoT devices might contain many services and applications, it is recommended to check if unnecessary services or interfaces are running and disable them, if possible. If this is not possible on the device, then it is necessary to block the services on the router/firewall [48][A13].

For the life-cycle of IoT devices, it is demanded that a secure provisioning and decommissioning mechanism is provided. For example, it must be ensured that the devices have been not manipulated before they are installed. The person responsible for the installation must be trustworthy and the steps need to be documented. The devices should not be connected to public or insecure networks before they have been fully installed and secured. In the case of the decommissioning, it is important to make sure that the devices are erased and that no sensitive information remains on the devices [48][A20].

While the BSI requirements provide generic advice, additional steps can be taken for a more secure IoT system.

In addition to the firmware update process, it might be also important to implement downgrade prevention. Older versions of firmware tend to have vulnerabilities that are fixed in future versions. If there is a way to initiate a downgrade of the firmware to an old, vulnerable, version, then the vulnerability can be abused again.

For the cloud environment, the establishment of secure identities on the cloud side and the client side is important. It needs to be ensured that the client is communicating only to the real cloud endpoints and not to an untrusted third party. Also, it is important for the cloud to correctly and securely identify the user and the devices. Here the risk is that a user can spoof his/her identity and take over devices of someone else. A spoofed device can send manipulated traffic to the cloud or even impersonate real devices.

Another helpful resource is the Open Web Application Security Project (OWASP) Embedded Application Security [39]. Its focus is more on embedded devices, however, most of the rules apply to IoT environments.

In addition to the recommendations from the BSI framework, it is required to secure the software on the device itself. For example, OWASP recommends implementing various software protections like buffer overflow, stack overflow, and injection prevention. Many devices receive input from untrusted sources, therefore potential vulnerabilities need to be closed or at least contained. One option is to use a secure-by-design approach in software development or the usage of hardening options in a particular programming language [39][E6].

Especially for IoT, privacy plays a big role due to the usage of these devices in private environments. Therefore it is important to limit the collection, storage, and sharing of personally identifiable information (PII) as well as other sensitive information, like credentials [39][E4,E9]. There are solutions available to ensure the protection of this information, like Trusted Execution Environments (TEE), however, it is still preferred if such information were not collected in the first place. In regard to the information it is also important to make sure that existing sensors cannot be abused.

A traditionally huge weakness in IoT devices is the availability of debug interfaces or backdoors. This applies to both physical debug interfaces and interfaces reachable over the network. The most dangerous is an open telnet port with hardcoded credentials, as seen before. There are recommendations of ensuring that no debug interfaces remain

active when the device gets deployed to the market. This can be done on a contract level (e.g. if a contractor develops the software) and needs to be verified before a product is mass-produced [39][E7].

As an additional security measure, the hardware can be protected against physical attacks [32]. There are many ways to achieve this. For example, the PCB can be encased in epoxy, which prevents probing attacks, modification of the circuits or desoldering of components (e.g. to read out the content of flash chips). Also, the design can be obfuscated, by removing the markings on the chip or by hiding vias inside of the PCB. While not designed as a security feature, the usage of Ball Grid Array (BGA) packaged chips can increase the difficulty for an attacker. For exposed applications, like doorbells, tamper-resistant designs can be helpful to prevent leakage of data (e.g. like the WLAN credentials of the user).

1.7 SECURE DEVICES VS. USER CONTROL

While secure devices are important to prevent attacks and potential leakage of user data, it also has some disadvantages.

A device which does not allow a user to execute custom firmware and has secured cloud interfaces is largely out of the control of the user. This means that the device is locked to one particular vendor. This situation can be problematic if the vendor goes out of business or decides not to support the product anymore. In 2016 this happened with Revolv after the company was acquired by Nest [52]. The Revolv smart home hub was left without a cloud connection and therefore became useless. Without custom firmware, the device would no longer be functional.

As most device vendors do not offer third party or even open source Application Programming Interface (API) connections, the user cannot decide to use an alternative ecosystem, like open-source home automation solutions. This makes the user highly dependent on the vendor.

For researchers, locked-down devices pose a challenge to analyze the vendors claims in regard to privacy and security.

1.8 FOCUS ON XIAOMI

The Xiaomi Corporation is a Chinese company founded in 2010. It is mostly known for its smartphones. In 2018, the company held a market share of around 10 percent in the smartphone market, which made them the 4th largest smartphone vendor worldwide [21].

In addition to smartphones, Xiaomi introduced in Spring 2016 their Smart Home IoT ecosystem, called Mijia. While most IoT accessory manufacturers have a narrow area of focus, Xiaomi IoT ecosystem supports a broad range of device types, including smart lightbulbs, sensors, cameras, vacuum cleaners, network speakers, electric scooters, smart toilet seats, and even washing machines. Initially, the ecosystem was focused on the mainland China market, where the company has a very high market penetration. However, today, their products are sold not only in Asia, but also in Europe and North America. The largest target market for Xiaomi after mainland China is India. The com-

pany claims to have the largest consumer IoT platform worldwide [3] with over 132 million connected devices in Q4 2018 [61].

Their devices might be deeply integrated with daily life and can collect a substantial amount of personal data. However, not all devices in Xiaomi's ecosystem are developed by the company. Whereas Xiaomi itself designs some devices, most IoT devices were developed by other companies and then integrated into Xiaomi's ecosystem. This results in different quality levels for software and designs. Xiaomi itself makes heavy use of Original Design Manufacturer (ODM) to develop products which are later branded under the Xiaomi IoT brand.

Xiaomi offers ecosystem integration as a service for device vendors. Traditionally, Xiaomi invested in IoT vendors and branded the products as theirs. Also, vendors can integrate their products into the Xiaomi ecosystem and benefit from Xiaomi's popularity and marketing. Many companies in the ecosystem sell devices under Xiaomi's name and therefore act as Original Equipment Manufacturers (OEMs). The usage of ODM and OEM is not unusual in the IoT device market [19]. Xiaomi offers a Software Development Kit (SDK)^{3,4}, which can be used by vendors to easily integrate their existing products into the ecosystem. This SDK supports multiple architectures and chipsets.

In November 2017, Xiaomi announced their Open IoT program at their developer conference (MIDC). For this, Xiaomi provides an ESP32 based module, which takes care of all the implementation of the Cloud API and communication [60]. Third-party developers can use simple UART commands to communicate with the Cloud, reducing the effort needed for software development.

Xiaomi has a partnership with other vendors and ecosystems. One example is their cooperation with IKEA, which enables the use of IKEA smart home products in combination with the Xiaomi IoT ecosystem [61].

1.9 SCOPE OF THIS THESIS

With the increasing usage of IoT devices, their security becomes a serious concern for security researchers. As stated earlier, there have been already many major security incidents involving IoT devices.

As Xiaomi has the largest ecosystem, in this thesis, I focus on devices which are integrated with their ecosystem and evaluate their security. My evaluation primarily focuses on IoT devices with a direct connection to the Internet. Typically these are devices which use WLAN, but they may have also other interfaces for connectivity. For better comparison, I will focus on devices which use the ARM architecture. To understand privacy implications I chose devices with privacy-related sensors, like light detection and ranging (LIDAR), microphones, or cameras. The actual server infrastructure of Xiaomi's

³ <https://GitHub.com/MiEcosystem>

⁴ <https://iot.mi.com>

ecosystem will be out of scope, as this data is typically out of control of the user as soon as it gets to the vendor. However, I will analyze the API between Device and Cloud and App to Cloud, to get an understanding of the data being transmitted.

Finally, I will analyze the possibility of rooting the evaluated devices. This should give the consumer and other researchers the chance to gain control over the devices, disconnect them from the cloud, and analyze their software.

RELATED WORK

The effort to gain root access on vacuum cleaning devices is not the first one published. In fact, there have been successful attacks by the company Checkpoint on LG IoT vacuum cleaners [26] in the first half of 2017. The researchers at Checkpoint used the Universal Asynchronous Receiver Transmitter (UART) to initially connect to the device and gained access on the actual filesystem using an undisclosed method through U-Boot. My assumption is that there a modification of the `cmdline` variable was done to start a `/bin/sh` shell by changing the `init` process. This method is a well-known way to bypass the login prompt of a device and is the default way for password recovery. It can be found in various mailing list discussions [56] and Linux tips sites [20]. After gaining access to the Operating System (OS) via UART, the researchers analyzed the binaries responsible for the communication to the smartphone app. In the second step, the smartphone app was analyzed. This required a recompilation of the app, as it had some protections in place. After recompiling, it was possible to intercept and modify the network traffic of the app. The researchers found insecurely implemented communication and authentication protocols. It was possible to change the identity of the user after the login process and therefore impersonate another user and take control of their devices. One of the features of that particular vacuum robot is access to the camera of the device (intended as part of a home alarm system) without the legitimate user knowing it. Checkpoint reported that vulnerability, and it was fixed by LG in at the end of September 2017. The information about the attack was released at the same time when my preparatory research for this master thesis was done. I took some ideas, like the risk to spy on the user, in consideration when developing my methods.

Ullrich et al. analyzed the security of Neato vacuum cleaning robots [57]. The researchers used a physical attack to gain access on the bootloader via a debug interface. After successfully using an exploit to get access to the QNX OS, they were able to analyze the binaries of the system. There they found multiple remotely exploitable vulnerabilities that affect the whole system's security.

Alrawi et al. [5] did a security analysis of over 45 different devices, including the accompanying mobile applications and cloud endpoints. They used different attack vectors, like access to Serial Peripheral Interface (SPI) flash, access via UART, and analysis of firmware updates. There was also a focus on active network-based attacks on the devices, like vulnerability scans using Nessus. The authors did not specify exactly which means were successful for each device; instead, more general approaches were described. While many vulnerabilities were found on devices of unknown vendors, many issues have been found for most mobile applications of the Internet of Things (IoT) devices. In particular, cryptography was implemented incorrectly, sensitive data was stored unprotected or the app claimed more privileges on the mobile phones than necessary. Also, the central cloud endpoints had configuration issues, especially Transport Layer Security

(TLS) issues. Interestingly, the authors faced challenges in the interception of Wireless Local Area Networking (WLAN) client-to-client communication. The solution for this problem was to use two different WLAN access points where the IoT device was connected to one access point, and the mobile phone with the app was connected to the other. In general, the idea and approaches of their work is similar to my thesis. However, the focus of my thesis is one particular IoT ecosystem. While the work is related to my thesis, it was published after most preparatory work was done by me.

Shwartz et al. released in late 2018 their paper about techniques and methods of reverse engineering of IoT devices [50]. The researchers described a workflow for the analysis of devices using inexpensive tools. The authors created a set of hardware and software tools for the reverse engineering process. These hardware tools included screwdrivers, a multimeter, a logic analyzer, a flash reader (CH341A) and two desktop computers. The authors tried to access the systems using UART connections. If the systems were protected with a login prompt and the U-Boot bootloader was uninterruptible by key-presses, fault injection attacks on the flash memory were used to stop the booting process. If the U-Boot shell was available, the authors used it to boot the system in single user mode and then replace the root password. As an alternative, the flash memory was dumped by desoldering the chip and reading it with the CH341A flash reader. Interestingly, the authors compared NAND flash in TSOP-48 package and SPI flash in SOIC-8 package, however, the author's tools are not suitable to read NAND flash at all. After gaining the firmware, the desktop computers were used in combination with powerful graphics adapters to crack the password hashes using the software JohnTheRipper and Hashcat. In total, 16 IoT devices, primarily surveillance cameras, were analyzed in the paper. The workflow of the paper contains many similar elements that I used for my research. While the methods might work especially well on devices like surveillance cameras, they are less effective for attacking more complex devices like vacuum cleaners or smart home gateways. The reason here is, that these devices are using NAND or Embedded Multimedia Card (eMMC) flash, quite often in Ball Grid Array (BGA) packages. This makes analysis of that kind of flash very difficult given the toolset of the authors.

In 2017 MWR Labs presented an attack on Amazon's Echo Dot [4]. After reverse engineering the debug interface of the device, which was hidden under the bottom cover, they were able to access the UART. After investigating the bootloader, they figured out that there is a way to boot the device from an external boot source. The external boot source could be attached using the exposed SD card pins and this source was preferred by the System-on-a-Chip (SOC) over the internal eMMC memory. After booting their own system, they extracted the contents of the internal eMMC and booted the system with `/bin/sh` as the `init` process. This enabled permanent changes to the file-system of the Echo device. Having root access on the device, they were able to listen to the microphone and stream the audio data to a remote device.

Attacks on Xiaomi ecosystem devices are not new. In fact, its surveillance cameras have been attacked by many people successfully. Davide Maggioni maintains a GitHub repository "yi-hack-v4"¹ with custom firmware for various "Yi" type cameras, which are

¹ <https://GitHub.com/TheCrypt0/yi-hack-v4>

operating in the Mijia ecosystem. The custom firmware can replace the cloud integration and adds features like Real-Time Streaming Protocol (RTSP) and Secure SHell (SSH). The aim of the project is to run the cameras stand-alone or in combination with open-source smart home solutions like Home Assistant [55]. Another project, named "Dafang-Hacks", is run by Elias Kotlyar², which focuses on "Dafang" type cameras, which are sold under various names, but also exist as a Mijia ecosystem version sold by Xiaomi [14]. Dafang cameras are based on the Ingenic T10/T20 SOC. The "Dafang-Hacks" project has the same aim as the "Yi-Hack" project, however it supports more smart home solutions. One of the first modifications of firmware was done by the GitHub user "samtap" and his repository "fang-hacks"³ in early 2017. His target device was "XiaoFang" branded, which are ARM-based cameras manufactured by the vendor "iSmart". With his firmware modifications, he completely disconnected the cameras from the cloud and offered a RTSP server instead. All three of these camera hacking projects used the same attack vector: automatic actions by specially prepared Secure Digital (SD) cards. All the vendors integrated functions in their bootloaders or firmware that scanned attached SD cards for special files. If these files existed, a particular firmware was installed or a shell script was executed from the SD card. By reverse engineering the binaries, the researchers found the triggers to execute a particular action. These triggers probably remain in the firmware for debugging or production reasons.

Sivaraman et al. [51] published in 2016 a paper discussing the risk of malicious smartphone apps attacking local smart home devices. A malicious app can scan the local network for known vulnerabilities of IoT devices without the user noticing it. Generally, nearly all apps have network privileges and it is very hard for the user to block them, in contrast to other privileges like contact list access or position access. The particularly interesting aspect of this attack is the fact that there are not many defenses against this attack as long as IoT devices are in the same network as normal end-user devices. This is usually the case, as a setup with separated networks is too complicated for the average user and routers have very poor protection against that kind of attacks. For this thesis, I considered evaluating the possibility of that attack on devices which I analyzed.

² <https://GitHub.com/EliasKotlyar/Xiaomi-Dafang-Hacks>

³ <https://GitHub.com/samtap/fang-hacks>

Part II

CONTRIBUTIONS

The contribution starts with a Xiaomi Ecosystem chapter, where I describe the design of the ecosystem and the Application Programming Interface (API). After this follows the Reverse Engineering methods chapter. Here I give an insight into different approaches to reverse engineer and modify IoT devices. The next chapter concentrates on evaluating the security and privacy of selected IoT devices in practice. Finally I discuss the relevance of found vulnerabilities on real world attacks.

XIAOMI ECOSYSTEM

This chapter describes the structure and function of the Xiaomi Cloud ecosystem. Also, the analyzed devices and their properties will be listed.

3.1 APPROACH

To investigate the structure of the Xiaomi Ecosystem, I followed two approaches: analysis of the Apps and reverse engineering of purchased devices.

As mentioned in the introduction, Xiaomi published on GitHub several repositories, which contain valuable information, mostly addressed to developers. A repository of particular interest is `miot-plugin-sdk`¹, which contains a debug version of the official "Xiaomi Home" app for Android. This debug version has some additional functions and is verbose with regard to log files. In addition, the behavior of the app is monitorable using Android's `logcat`². This significantly helped to understand the connections between the cloud infrastructure, the apps and the devices themselves.

The "Xiaomi Home" app retrieves at startup a list of supported devices from the servers. This list is in JavaScript Object Notation (JSON) format, and contains a structure of the supported devices in a particular region with some additional metadata. This metadata is helpful in understanding the functionality and features of the devices. An example of such device description can be found in listing 1. The field "pid" determines³ the connectivity feature of the device model, e.g., if the device is a Wireless Local Area Networking (WLAN) device, Zigbee device or Bluetooth Low Energy (BLE) device. Another interesting piece of information is the URL to the "icon_real" which offers a picture of the device, which is helpful to distinguish between unknown devices.

Listing 1: Example of a device entry of supported devices

```
1 ...
2 {
3     "model": "rockrobo.vacuum.v1",
4     "desc": "Mi home sweeping robot",
5     "subcategory_id": "113",
6     "pid": 0,
7     "rank": 0,
8     "pd_id": 85,
```

1 <https://GitHub.com/MiEcosystem/miot-plugin-sdk>

2 Logcat is a feature of Androids SDK, which enables easy access to logfiles while runtime. See for more information here: <https://developer.android.com/studio/debug/am-logcat>

3 The meaning of "pid" equals the field "iDevice.type" which is described here: <https://GitHub.com/MiEcosystem/miot-plugin-sdk/wiki/> under the point "Device"

```

9     "icon_on": "http://cdn.fds.api.xiaomi.com/miio.files/
        commonfile_png_3b83f4dd980b9b5398650dc92aa186a8.png",
10    "icon_real": "http://cdn.fds.api.xiaomi.com/miio.files/
        commonfile_png_430f65a6257b260001e0f7f15b4a6742.png",
11    "icon_guidepic": "",
12    "icon_bluetooth_pairing": "",
13    "icon_smartconfig_on": "http://cdn.fds.api.xiaomi.com/miio.files/
        commonfile_png_e3a5b726bb268111ce2ae3aa21d7b22f.png",
14    "icon_smartconfig_off": "http://cdn.fds.api.xiaomi.com/miio.files/
        commonfile_png_1dfffc4cba03df545c17876c2d0d3e4fd.png",
15    "icon_smartconfig_succ": "",
16    "gif_smartconfig": "",
17    "sc_type": 0,
18    "resolve_type": 0,
19    "voice_control": 0,
20    "voice_ctrl_ed": 1,
21    "bt_is_secure": 0,
22    "bt_gateway": 0,
23    "icon_336": "",
24    "fiveG_wifi": 0,
25    "ios_sc_visible": true,
26    ...
27    "inherit_id": 0,
28    "mesh_gateway": 0,
29    "member_cnt": 0,
30    ...
31 }
32 ...

```

3.2 STRUCTURE

A simplified structure of the Mijia ecosystem is shown in Figure 2

Xiaomi, as the owner of the miIO/Mijia ecosystem, is operating the cloud infrastructure. While it operates some of the servers in Mainland China itself, most of the actual infrastructure is run on Amazon Web Services (AWS). This can be discovered by resolving the Domain Name System (DNS) names, like `ot.io.mi.com` and then doing a reverse lookup of the returned IP addresses. Depending on the region, the number of returned server IP addresses varies. The cloud infrastructure consists of multiple components: the interface for the devices (`*.ot.io.mi.com`), the interfaces for the smartphone (`*.api.io.mi.com`) and a generic Content Delivery Network (CDN) (`cdn.*.fds.miimg.com`). The prefix defines the region. At the moment there are 6 real regions available: "cn" for Mainland China, "de" for Europe, "us" for the United States, "ru" for Russia, "iz" for India and "sg" for the rest of the world.

The central component of the Mijia ecosystem is the user's smartphone with the installed "Xiaomi Home" app. This app exists for Android and iOS and is developed by

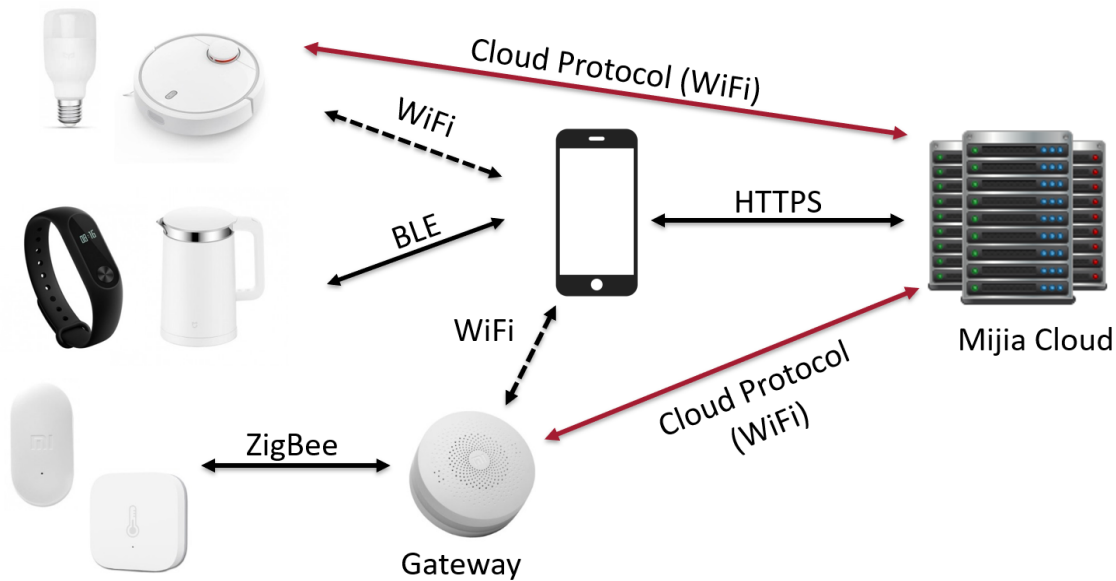


Figure 2: Simplified communication relations in the Mijia ecosystem

Xiaomi. The app is mandatory for the usage of the Mijia features; for most devices the app is the only user interface. The user authenticates here with his/her user ID. The user IDs are sequentially assigned. Also, the app is used to initially provision the WLAN devices and to communicate with the BLE devices. A provisioned device is bound to a particular user ID. While most devices will connect directly to the Mijia cloud after provisioning, there is still some communication between the app and the device directly. Mostly this is used when devices and the smart phone are in the same WLAN network, so it does not make sense to connect to the device via the cloud. This direct connection has the advantage of lower latency for commands/responses. If the smart phone and the device are not in the same WLAN network, then all communication is done via the cloud. While Xiaomi provides the app, the actual logic of the device is developed by the device vendors and is provided in the form of a plugin for the app.

The Internet of Things (IoT) devices themselves are the most important component from the user's perspective. These devices can have different brand names, and can be developed and manufactured by different vendors. As mentioned in the introduction, only a small number of the Xiaomi branded devices are actually developed by Xiaomi. Each device in the ecosystem has a unique Device ID (DID), a secret Device Key (DKEY) and Medium Access Control (MAC) address. This information is put on the device at the time of production and is never changed. Additionally, the model is defined by the "device model". This device model has the format "vendor.devicetype.model" and is therefore helpful to identify the actual vendor of a particular device. As soon as a device is provisioned, the device creates a so-called "device token", which is used instead of the DKEY as a secret used for communication with the app. Additionally, the device is connecting to the configured WLAN access-point and binds itself to the user ID. Every time a device gets unprovisioned, the provisioning information should be deleted and the device token regenerated. This token is also used by the Mijia cloud to keep track of a

Stored information on device	permanent?	usage
Device ID (DID)	✓	unique device identification
Device Key (DKEY)	✓	key for cloud communication
MAC Address	✓	device identification
Token	✗	key for app communication, identifier for provisioning
WLAN SSID	✗	connection to wifi
WLAN Password	✗	connection to wifi
User ID	✗	binding to a particular user
Country	✗	setting for a particular region
P2P ID (camera devices only)	✓	unique device ID for P2P

Table 1: Stored configuration settings for Mijia devices

particular provisioning state. Therefore, it is also stored by the cloud. The configuration information that is usually stored by every Mijia device can be found in Table 1. This constitutes the minimum information required to communicate with the Mijia cloud. It was empirically found, that the DID is not bound to a particular MAC address or region. Also, the device model is not initially set in the cloud and will be updated at the first connection. However, this device model is not synchronized between multiple regions, therefore it is possible that the same DID has multiple identities in different regions.

3.3 DEVICE API

The communication between the device and the Mijia cloud does not use Hypertext Transfer Protocol Secure (HTTPS) like other IoT ecosystems. Instead, Xiaomi created their own protocol, which is often referred to as "miIO". It supports UDP datagrams and TCP streams, and uses the same payload for both types. The protocol identifies each device by a unique DID. Each DID has its individual DKEY. The DKEY is used for the symmetric encryption for integrity and authentication. For encryption, AES256 in Cipher Block Chaining (CBC) mode is used. HMAC ensures integrity and authentication. As an additional feature, the protocol is also used for time synchronization, as every packet contains the Unix epoch time. This removes the need to synchronize the time using protocols like Network Time Protocol (NTP). The outer structure of the protocol is shown in Table 2. As mentioned before, there is also a communication path between the device and the "Xiaomi Home" app. The protocol used here is the same as the Mijia cloud communication, except that the device token is used.

The inner structure of the payload is in JSON format and is organized in packets. Each packet is identified by a packet ID. There are two main types of messages: commands and responses. Commands usually contain a "method" and "params." Responses usually only contain an array with the name "results." Usually, every command or response is confirmed by the recipient. There is a special kind of message type: heartbeats. Heartbeats are sent by the device every 15 or 30 seconds to keep the connection alive. They

	Byte 0,1	Byte 2,3	Byte 4,5,6,7	Byte 8,9,A,B	Byte C,D,E,F
Header	Magic:2131	Length	00 00 00 00	DID	epoch (big endian)
Checksum	Md5sum[Header + Key(Cloud)/Token(App) + Data(if exists)]				
Data	Encrypted Data (if exists, e.g., if not Ping/Pong or Hello message) - token = for cloud: key; for app: token - key = md5sum(token) - iv = md5sum(key+token) - cipher = AES(key, AES.MODE_CBC, iv, padded plaintext)				

Table 2: miIO protocol structure

have an empty data field. The protocol for the app-to-device communication has been reverse engineered in 2017 and published on GitHub under the name "python-miio" [67]. However, there was, to my knowledge, no public information available about the device to cloud communication until I published preliminary information which I gained in preparation of this thesis.

Listing 2: Example of a message from the device to the cloud

```

1 {
2   "id":136163637,
3   "params":{
4     "ap":{
5       "ssid":"myWifi",
6       "bssid":"F8:1A:67:CC:BB:AA",
7       "rssi":-30
8     },
9     "hw_ver":"Linux",
10    "life":82614,
11    "model":"rockrobo.vacuum.v1",
12    "netif":{
13      "localIp":"192.168.1.205",
14      "gw":"192.168.1.1",
15      "mask":"255.255.255.0"
16    },
17    "fw_ver":"3.3.9_003077",
18    "mac":"34:CE:00:AA:BB:DD",
19    "token":"xxx"
20  },
21  "partner_id":"",
22  "method":"_otc.info"
23 }

```

An important functionality of the cloud Application Programming Interface (API) is the ability to install firmware updates. In general, there are two kinds of firmware updates: so-called "app" updates and "mcu" updates. "App" updates are for the de-

vice component, which communicates with the cloud, e.g., the main System-on-a-Chip (SOC). An example of such a command can be seen in Listing 3. Notice that in addition to the Uniform Resource Locator (URL), the MD5 hash of the firmware update is also transmitted. The device can use this information to verify that the firmware update has not been tampered with, especially if it was transmitted over Hypertext Transfer Protocol (HTTP). "MCU" updates are for additional processors in the device, e.g., ZigBee controllers. In Listing 4 an example command is shown. In comparison to the "App" updates, the "MCU" update commands do not transmit an MD5 hash, therefore the device cannot verify the integrity of the firmware update that way.

Listing 3: Example of cloud command for installing an App firmware update

```

1 {
2   "method": "miIO.ota",
3   "params": {
4     "app_url": "http://cdn.cnbj0.fds.api.mi-img.com/miio_fw/upd_lumi.
      gateway.v3.bin?...",
5     "file_md5": "063df95bd...cf11e",
6     "install": "1",
7     "proc": "dnld install",
8     "mode": "normal"
9   },
10  "id": 123
11 }

```

Listing 4: Example of cloud command for installing an MCU firmware update

```

1 {
2   "method": "miIO.ota",
3   "params": {
4     "mcu_url": "http://cdn.cnbj0.fds.api.mi-img.com/miio_fw/mcu_lumi.
      gateway.v3.bin?...",
5     "install": "1",
6     "proc": "dnld install",
7     "mode": "normal"
8   },
9   "id": 123
10 }

```

A more detailed view of the communication relations from the perspective of the IoT device is shown in Figure 3. Here, the component for the communication between the device logic and the cloud is "miio_client". The "miio_client" is part of the Xiaomi Software Development Kit (SDK), which is distributed to vendors. To obtain the SDK, one must register as a vendor and sign a Non-Disclosure Agreement (NDA). Therefore, I do not have access to the SDK and can only do a black-box analysis. In the case of Linux-based devices, the configuration is stored in a file with the name "device.conf". The token is stored in the file "device.token". For RTOS based devices, the configuration is stored in a configuration partition. Apart from "miio_client", the application logic can

establish its own connection, e.g., to the Xiaomi File Storage Service (FDS) servers, in order to download new firmware.

In order to impersonate a device, the only information that is required is the DID and DKEY. Using this information, it is possible to impersonate different models in different regions at the same time. However, as soon as a device model was reported to the cloud, the model cannot be changed anymore. Interestingly, changing device models was possible until summer 2018, therefore it was possible to use extracted credentials to gain information about unknown models and to retrieve firmware updates.

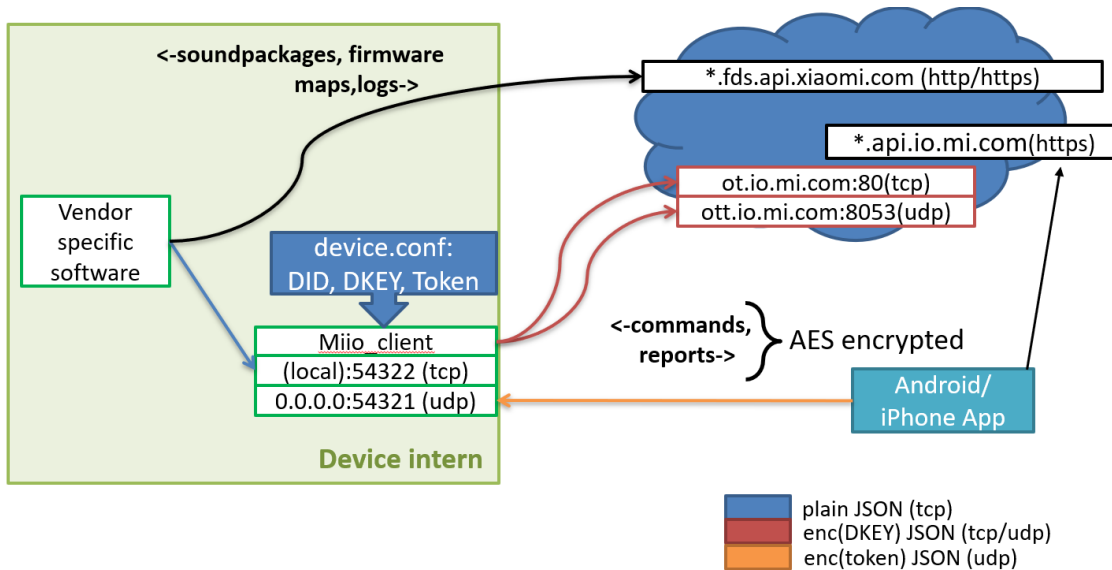


Figure 3: Detailed overview of components for the device communication

3.4 APP TO CLOUD API

The app to cloud communication differs from the device to cloud communication. Here, two different communication layers are used. The outside layer consists of a HTTPS connection. The inside layer is protected by additional AES encryption plus an HMAC, which is using a session key. The user authenticates in the app against the cloud using OAuth [24]. After the authentication, the user receives a service token, which is valid for 14 days. With this token, the user can send requests to the cloud. An interesting aspect here is that the service token is not revoked if the user password is changed.

The used protocol inside the payload is JSON Remote Procedure Call (RPC). The cloud endpoints have different resources to which the apps communicate. For example, the resource "api.io.mi.com/home/device_list" returns a list of all devices bound to the authenticated user, as can be seen in Listing 5. The interesting aspect is here, that the cloud is aware of the geographical location of a device. The source of this information is the "Xiaomi Home" app at the time of provisioning. As unprovisioned devices run

an open access-point for the initial configuration, the app needs localization permission in Android in order to find and to connect to this access-point. Granting the app this permission results in the behavior that the app is reporting its current position to the cloud. As the smart phone needs to be in proximity of the unprovisioned device, the cloud assumes that the smart phones position is equal to the device position. Except for the region "Mainland China", the information in the fields "longitude" and "latitude" is "o.o". However, this does not mean that the Xiaomi is still not collecting this information.

Listing 5: Excerpt of a cloud response of the user's device list

```

1  {
2    "message": "ok",
3    "result": {
4      "list": [
5        {
6          "did": "65981234",
7          "token": "abc...zzz",
8          "name": "Mi PlugMini",
9          "localip": "192.168.99.123",
10         "mac": "34:CE:00:AA:BB:CC",
11         "ssid": "IoT",
12         "bssid": "FA:1A:67:CC:DD:EE",
13         "model": "chuangmi.plug.m1",
14         "longitude": "-71.0872248",
15         "latitude": "42.33794500",
16         "adminFlag": 1,
17         "shareFlag": 0,
18         "permitLevel": 16,
19         "isOnline": true,
20         "desc": "Power plug on ",
21         "rssi": -47
22       }
23     ]
24   }
25 }

```

3.5 INTERCEPTION OF CLOUD TRAFFIC: DUSTCLOUD

After reverse engineering the first devices, I gained a better understanding of the protocol and realized that "python-miio" can also be used to parse and build device-to-cloud communication. This knowledge was used to build the tool Dustcloud⁴. Dustcloud is implemented in Python 3.6 and PHP. It can work in two modes: it can act as a cloud server emulator or as a proxy between the device and the cloud. As a cloud server, it can interact with the device and, for example, push updates to it. However, in this mode the complete communication to the real cloud is disconnected. In proxy mode, it

⁴ <https://github.com/dgiese/dustcloud>

Region name	Region prefix	supported models
Mainland China	"cn" or none	803
Global	"sg"	85
Europe	"de"	74
United States	"us"	49
Russia	"ru"	29
India	"i2"	8

Table 3: Mijia devices supported by region

may forward device messages to the cloud, but it can also change or even suppress commands (e.g., in the case of unwanted firmware updates). The advantage of Dustcloud is that it reads all the traffic in plaintext. Therefore it can be used to reverse engineer specific commands between the device and the cloud, in order to implement them in open source smart home software. However, for it to be able to operate, it is required that the DID and DKEY is known and configured in the tool. Additionally, a DNS redirection needs to be configured. The DNS names "ot.io.mi.com" and "ott.io.mi.com" need to be redirected to the IP address of the Dustcloud server. However, some devices block cloud traffic to local IP addresses. Therefore a public IP address needs to be used and redirected to the local server on the firewall. An example of the configuration can be found in the Wiki of the GitHub repository⁵.

3.6 DEVICES

In June 2019, the Mijia ecosystem supported more than 820 device models in all regions together. The individual numbers can be found in Table 3. While some device models overlap, it still presents an impressive number. For a closer analysis in this thesis, I selected a smaller number of device models based on the following characteristics: The device must be easy to ship due to the research taking place in Darmstadt (Germany) and Boston (US). Therefore, huge home appliances like fridges or televisions were excluded. The devices must also have WLAN connectivity as my primary interest is on devices which have a direct internet connection.

The devices selected for further analysis are shown in Table 4. In this table, the device is shown with its known name, which may differ from the official name. For better identification, the Mijia model and the region where the device was purchased is defined. As mentioned in the introduction, the brand of the device does not always disclose the actual developer/vendor; I included this information as well. For reference, there is also the approximate release date for each of the models. This release date might differ depending on regions. For some devices, the release date was not known and was therefore estimated. Finally, the approximate price is given. The price was rounded and is based on the cost at the time of the purchase. The purchases have been made from many different merchants, like gearbest.com, banggood.com, Amazon.com or even local

⁵ <https://github.com/dgiese/dustcloud/wiki/Preparations-on-the-Vacuum-Robots>

Device name	Region	Mijia model	Vendor	Release	Price (USD)
Aqara Gateway (Homekit)	CN	lumi.gateway.aqhm01	Lumi	Q2 2018	50
Aqara Gateway (Homekit)	US	lumi.gateway.aqhm02	Lumi	Q1 2019	50
Aqara Smart Home Gateway	TW	lumi.gateway.mitw01	Lumi	Q1 2018	35
Aqara Smart IP Camera	CN	lumi.camera.aq1	Lumi	Q4 2017	35
Lumi Smart Home Gateway	CN	lumi.gateway.v3	Lumi	Q3 2016	30
Philips Ceiling Lamp	CN	philips.light.ceiling	Yeelight	Q2 2017	70
Roborock S50	EU	roborock.vacuum.s5	Roborock	Q1 2018	400
Roborock S50	CN	roborock.vacuum.s5	Roborock	Q4 2017	350
Roborock T61	CN	roborock.vacuum.t6	Roborock	Q1 2019	450
Roborock S61	EU	roborock.vacuum.s6	Roborock	Q1 2019	550
Xiaomi Mi Vacuum Robot	CN	rockrobo.vacuum.v1	Roborock	Q4 2016	280
Xiaomi Mi WiFi Speaker	CN	xiaomi.wifispeaker.v1	Xiaomi	Q4 2016	85
Xiaomi WiFi Plug	CN	chuangmi.plug.m1	Chuangmi	Q2 2016	15
Yeelink Bedside lamp	CN	yeelink.light.bslamp1	Yeelight	Q4 2017	25
Yeelink Bedside lamp	TW	yeelink.light.bslamp1	Yeelight	Q1 2018	30
Yeelink Ceiling Lamp	CN	yeelink.light.ceiling1	Yeelight	Q3 2017	65
Yeelink Light Color	CN	yeelink.light.color1	Yeelight	Q4 2016	10
Yeelink Light Mono1	CN	yeelink.light.mono1	Yeelight	Q4 2016	10
Yeelink Light Strip	CN	yeelink.light.strip1	Yeelight	Q4 2016	15
Yeelink Smart White Bulb	EU	yeelink.light.ct2	Yeelight	Q2 2018	15
Yeelink Smart RGB Bulb	EU	yeelink.light.color2	Yeelight	Q2 2018	15

Table 4: Overview over Mijia devices to be tested

Xiaomi stores. From my experience, the prices are volatile, depending on promotions of the merchant websites.

REVERSE ENGINEERING METHODS

This chapter describes methods and approaches for reverse engineering of Internet of Things (IoT) devices in general. The methods are separated in different groups and will be discussed individually. Then a workflow for the reverse engineering of the devices will be introduced. In the next chapter, these methods will be used to attack IoT devices. There are many howtos, workflows [50] and checklists [33] available for testing of vulnerabilities and reverse engineering of devices. These are helpful to design a specialized workflow for a particular ecosystem.

The goal in reverse engineering is to get access to the Operating System (OS) and/or the firmware itself. There are three main approaches to achieve this: from the network, via the hardware and via the software. The approach over the network usually means less effort, but if it is not successful, there is the hardware approach. However, both approaches can be executed in parallel.

4.1 NETWORK

A key feature of IoT devices is the connectivity to the Internet. This offers an active and passive possibility to analyze the device.

As the device communicates with the Cloud or with local devices (e.g., Apps on smartphones or even other IoT devices in the local network), the traffic can be passively monitored. This method is always possible and does not require direct interaction with the device under test. Suitable tools are Wireshark¹, Kismet² or tcpdump³. The passive approach has the advantage that the risk of damaging the device is very low. It has been shown that it has advantages to have a separate router and Wireless Local Area Networking (WLAN) network for IoT devices, as it enables more control over the traffic. In particular, a WLAN router can be configured in a way that it blocks direct traffic between WLAN clients, so called "client-to-client" or "inter-client" traffic, and force the traffic sent through the router. That way it can be monitored or even modified. An example network setup is shown in Figure 4. Here all the IoT traffic can be captured and send to the monitoring computer. In addition, a firewall can block unwanted traffic towards the Internet. One option is to block Domain Name System (DNS) traffic and force the IoT devices to resolve host-names using the routers DNS resolver. That way it is possible to redirect traffic to different servers. Another option is also to use a software like sslstrip⁴ in combination with the firewall to intercept Transport Layer Security (TLS) traffic. This works in many cases as the TLS protocol is poorly implemented on many devices, if used at all. With this method, it is possible to intercept unencrypted

¹ <https://www.wireshark.org/>

² <https://www.kismetwireless.net/>

³ <https://www.tcpdump.org/>

⁴ <https://moxie.org/software/sslstrip/>

traffic between IoT device and cloud backend to understand the communication protocol or firmware update mechanisms. This method is especially successful for the retrieval of firmware update packages, which can then be analyzed without extracting the firmware from the devices using a different method. For my use cases an OpenWRT compatible router⁵ is sufficient for all the tasks above, especially as it can run on cheap routers and offers already a lot of pre-compiled packages.

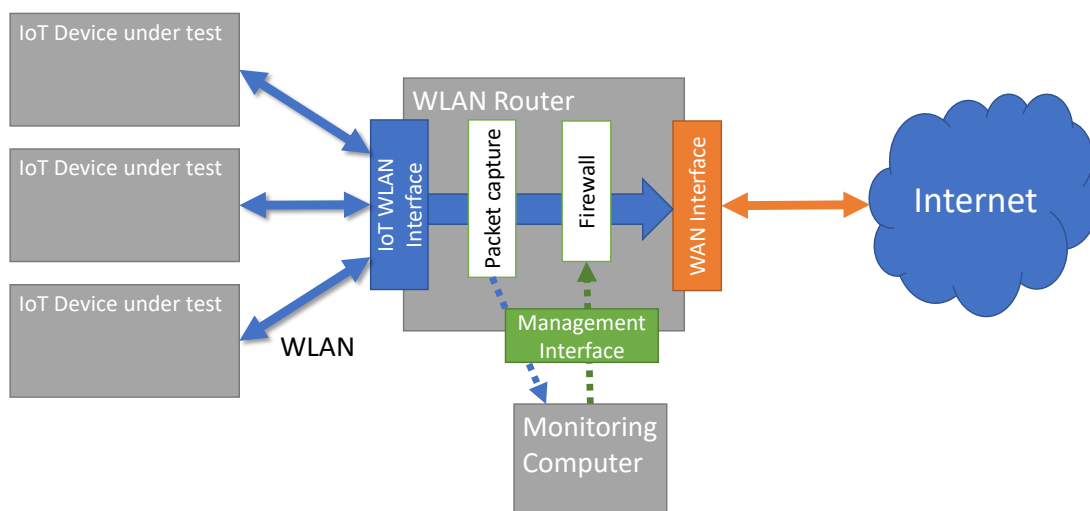


Figure 4: Example network setup to monitor IoT device traffic

A more active approach is the usage of port scanning tools or vulnerability scanners. The usual example for port scanning tools is `nmap`⁶. Example vulnerability scanners are `Nessus`⁷ or its open-source fork `OpenVAS`⁸. Using these tools, it is possible to probe a device for open ports and exposed services. The information about open ports and running services is helpful to assume the running OS type and version. Having this information enables the researcher to select already known vulnerabilities and attack the system from the network. One service, which is in particular of interest is `telnet`, which is in many cases vulnerable and actively attacked [6].

4.2 HARDWARE

The first step in the analysis of the hardware is to check the device for interfaces which are accessible without opening the devices. Especially interesting are externally accessible debug interfaces, or storage ports, like SD card slots. The advantage of this method is that potential warranty seals are not broken and that it is simple. One example for accessing a debug interface without opening the device is shown in Figure 5. Here the

⁵ <https://openwrt.org/>

⁶ <https://nmap.org/>

⁷ <https://www.tenable.com/products/nessus>

⁸ <http://www.openvas.org/>

Universal Asynchronous Receiver Transmitter (UART) pins were accessible thru holes in the case of the Xiaomi Router R3G using needles and paperclips. This was possible without breaking any warranty seals or leaving traces on the outside of the device.

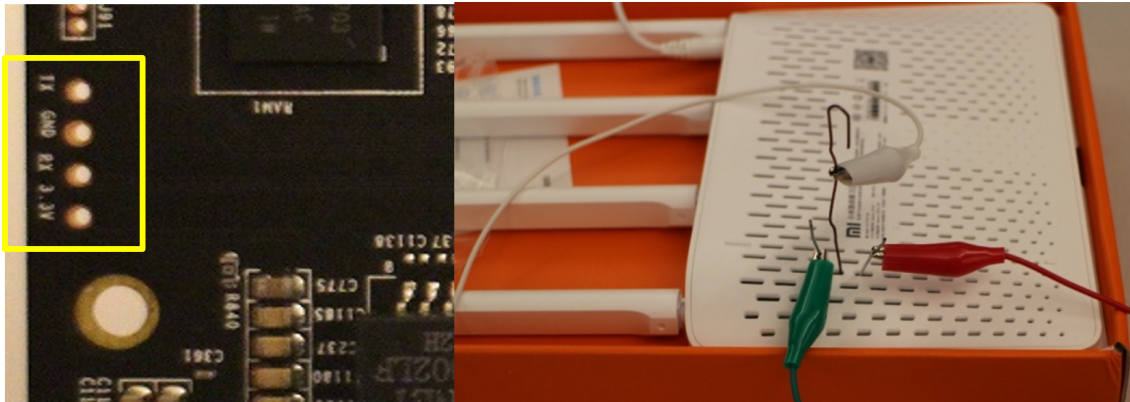


Figure 5: Accessing the UART pins through venting holes in the case

The next step requires opening the device. Depending on the design of the case, this might be difficult if the manufacturer used glue or one time clips, which break when the case is opened. Additionally, the manufacturer might have used warranty seals, so it becomes evident that the device has been opened. However, for the disassembly of most IoT devices the same tools can be used as for smart phones repairs, especially as IoT devices are often less complex. For example, the company iFixit offers various sets of tools which are suitable⁹. Upon opening, the Printed Circuit Board (PCB) can be inspected for debug interfaces. Also the used hardware can be analyzed to get a deeper understanding of the used architecture and components. Especially test pins on the PCB are of interest, as they can be probed manually or in an automated fashion, e.g., using tools like the JTAGulator [22]. In some cases, like in Figure 5 shown, they are directly visible and marked as such. For manual probing it is helpful to use a multimeter to measure the voltage of a testpin in relation to ground. Depending on the operating voltage of the System-on-a-Chip (SOC), an UART TX pin is pulled up to a voltage of 3.3 Volts or 5.0 Volts. A UART RX pin should be pulled down to a voltage of 0 Volts in relation to ground. In some cases it's possible to access Serial Peripheral Interface (SPI) flash without desoldering it using In-System Programming (ISP). Access to a SPI chip using ISP requires the ability to stop the SOC, e.g., by permanently triggering the reset pin. The method of opening the case can be regarded as relatively low-risk in regard to a potential destruction of the device. However, this does not apply for all devices.

If a UART or Universal Serial Bus (USB) port has been found but the access on the OS is blocked, a fault injection attack can be executed. As mentioned in the introduction, many SOCs have a fallback mode in case a valid system cannot be loaded. Additionally, the U-Boot bootloader stops by default in the integrated shell in case loading the kernel was not successful. This behavior can be exploited by corrupting the data lines of the flash and prevent loading of the Kernel. This method is safe for SPI and Embedded

⁹ <https://ifixit.com/Store/Tools/Toolkits>

Multimedia Card (eMMC) flash, but can cause corruptions on NAND flash, depending on the SOC. Typically, for eMMC flash, the DATo line is grounded. For SPI flash the CLK signal can be connected to the MISO signal. The usual method for NAND flash is to ground the IO-0 or IO-1 signals. Depending on the goal, the fault injection is executed before the device is powered on or shortly after U-Boot has been loaded.

If ISP imaging of the flash is not possible and no suitable debug interfaces are available, the next step would be to desolder the flash. Depending on the type of flash, the desoldering process is more or less complex. As this method is very common, there are how-to guides available online [44]. For most packages, a hot air station can be used to remove the chip. In case of NAND chips, especially in Thin Small Outline Package (TSOP)-48 or TSOP-56 packages, a longer application of hot air might be necessary, which poses a potential risk for surrounding chips and poses the risk of destroying the chip through thermal stress due to absorbed moisture [35]. A removal of moisture in a reflow-oven at a low temperature over multiple hours might be necessary. The alternative solution to hot air is to apply a low temperature soldering alloy to pin and then using a soldering iron to mix it with the existing solder. A commercially available product is produced by the company CHIP QUIK¹⁰, which also holds the patent for the method. The advantage of this method is that the melting temperature after the mix is much lower and therefore reduces the risk of damaging parts on the PCB. The melted solder can be removed by using a desoldering pump or desoldering wick. Unsoldering and later resoldering the same chip works good for Small Outline Package (SOP) and TSOP packages. However, the resoldering process for Ball Grid Array (BGA) chips is more complex, as it requires reballing and a very precise placement of the chip. Usually more sophisticated tools are required. Without these tools, there is a high risk to permanently damage the device.

Generally, when working with PCBs it is sometimes necessary to be able to solder very precisely. Therefore a stereo microscope is very useful, as it not only makes soldering easier but also enables a closer inspection of the individual components and traces on the PCB.

After removing the flash chip, the chip can be read out and its content analyzed. Depending on the type of flash, different tools can be used. For SPI flash, a Raspberry Pi can be used in combination with the software `flashrom` [43]. In case of eMMC flash, the Raspberry Pi supports also Secure Digital Input Output (SDIO) connections out of the box [47]. However, for raw NAND chips, a different tool is required, as the NAND support of the Raspberry Pi is unreliable due the high number of required pins. Therefore more specialized tools like `FlashcatUSB`¹¹ can be used. A special problem of NAND flash is that it does not contain a controller and therefore the NAND controller in the SOC is usually responsible to maintain wear-leveling and Error-Correcting Code (ECC) [11]. As the implementation of the ECC algorithm may differ between different SOCs, additional reverse engineering might be required in order to extract the data. An

¹⁰ <https://cdn-shop.adafruit.com/product-files/2660/SMD1.pdf>

¹¹ <https://www.embeddedcomputers.net/products/>

alternative is to use the exact same SOC, e.g., on a development board, to access the data.

4.3 SOFTWARE

For the software analysis, it is necessary to first retrieve the firmware. This can happen through the above described network or hardware approach. An additional method is the retrieval of a firmware update through an impersonated device identity. In the introduction, it was mentioned that the model name of a Mijia ecosystem device can be changed. The idea here is to extract Mijia cloud credentials from a cheap and insecure device and then to impersonate a more secure device. That way the firmware update can be retrieved and analyzed.

The first step of the software analysis is to identify the OS and the installed applications. This also includes the search for obvious vulnerabilities, back-doors, and potential methods to obtain root access. For a full OS it can be useful to manually root (e.g., by enabling Secure SHell (SSH) or telnet) the extracted image, and re-image it on the device again. That way also the hardware itself can be analyzed.

A more convenient way to analyze firmware is the usage of automated tools. One example is the Firmware Analysis and Comparison Tool (FACT), which allows automated analysis of the used OS, software, crypto libraries and potential vulnerabilities. FACT also enables the simple comparison of different firmware, even between different devices [65].

Another automatic tool for reverse engineering is the "firmware-mod-kit" [66]. This kit automatically unpacks known Linux firmware formats and offers the possibility to modify and repack the firmware again.

One problem with automatic tools is that they do not support bare-metal OSs like an Real-Time Operating System (RTOS). RTOS-based firmware require manual analysis. Here, disassembly and decompiling tools are very useful. One tool that I will use for this thesis is IDA Pro¹², however, after March 2019, the tool Ghidra¹³ is available and can be used. Ghidra and IDA Pro enable a deep analysis of binaries and offer also to patch binaries. An advantage of IDA Pro is the integrated multi-platform debugger [28].

The usual problem with a bare-metal OS is that the firmware does not contain any debug symbols. This makes the analysis complicated. A way to solve this problem is to get the Software Development Kit (SDK) of a particular SOC or even the ecosystem, and to compile an example firmware with debug symbols. Then the IDA Pro plugin `bindiff` can be used to map the known functions of the compiled example to the unknown functions in the device firmware. `bindiff` is a comparison tool for binary files, that can find similarities between disassembled code [68]. It can detect similar functions based on basic blocks and a call graph analysis. The gained information can be then even used

¹² <https://www.hex-rays.com/products/ida/>

¹³ <https://ghidra-sre.org/>

Tool name	Usage	Vendor
SEGGER J-Link EDU	JTAG debug probe	SEGGER
Raspberry Pi 3B	JTAG interface, SPI interface	Raspberry Pi Foundation
FlashcatUSB XPORT	Flash programmer (SPI,I2C)	Embedded Computers LLC
FlashcatUSB Mach	Flash programmer (NAND)	Embedded Computers LLC
eMMC Adapter	eMMC-to-SDIO adapter	exploitee.rs
Leica A60F	Stereo Microscope	Leica
Scienscope NZ	Stereo Microscope with camera	Scienscope
AS-5001	Reflow oven	SMTmax
AOYUE Int768	Hotair and soldering station	Aoyue
Agilent U1241A	Multimeter	Agilent
Xiaomi R3G Router	WLAN Router with OpenWRT	Xiaomi
IDA Pro 7.2	ARM Disassembler/Decompiler	Hex-rays
OpenOCD	On-Chip Debugger on Pi3B	Dominic Rath
HxD	Hexeditor	Maël Hörz

Table 5: Overview over prepared tools for reverse engineering

on other firmware of the same ecosystem.

For the modification of ARM binaries there is a helpful tool with Nexmon [49]. Nexmon is a firmware patching framework mainly developed for Broadcom/Cypress WiFi chips. However, it can be used to patch arbitrary ARM binaries. The advantage of Nexmon in comparison to manual patching is, that the patches can be written in C and the framework takes care of the compilation and linking. This enables the writing of more complex patches without much effort. However, in order to be able to use Nexmon, it is required to map the firmware binary and to identify interesting functions. As the binary was stripped of any debug information, the function names were unknown and need to be retrieved with the described method above.

4.4 SUMMARY OF TOOLS

In Table 5 the used tools are summarized. This table contains the most important software and hardware tools, however, no mechanical tools, like screwdrivers. While the toolset is powerful, there are some limitations, like the reliable soldering of BGA chips. For most cases even less expensive tools can be used, e.g., the reflow-oven and the stereo microscope are optional. The software IDA Pro can be subsidized by Ghidra.

EVALUATION OF DEVICES

In this chapter, devices of the Xiaomi ecosystem will be evaluated with regard to their security and privacy. For that, the methods of the previous chapter will be used. First, I define the metrics for the comparison of the individual devices. Finally, I analyze a selection of devices based on these metrics.

5.1 COMPARISON GUIDELINE

To compare the devices, I choose to use the following criteria to categorize each device.

5.1.1 *Hardware*

How tamper resistant is a device? If modified by a third party, how evident is the misuse? How likely is it that an attempt to tamper with device will render it inoperable? Tamper resistance and evidence is important for Internet of Things (IoT) devices, as especially used devices pose the risk of being manipulated. Additionally a tamper resistant device prevents supply-chain attacks. However, tamper resistance has also the disadvantage that it prevents the owner to modify his/her own device.

5.1.2 *Firmware security*

Is the firmware easily modifiable by a third party? Is secure boot or secure OS used? How sophisticated is the encryption on any firmware updates? Are the firmware updates signed? Is the integrity of the firmware update actually verified? Firmware updates are the best way to attack a system. In case of improperly implemented firmware update mechanisms there is a risk that a malicious firmware can be installed. The attack vectors here are attacks from the local network or Man-in-the-middle (MITM) attacks.

5.1.3 *Network*

Are ports open or services accessible? Is the network traffic encrypted? Is Transport Layer Security (TLS) correctly implemented? An IoT device is a potential target for hackers, therefore open and insecure services are posing a risk. The traffic between the device and the cloud should not be unencrypted, as a malicious third-party can read sensitive data or even modify the traffic. Even if the traffic is encrypted, it is important that the encryption is implemented correctly, e.g., the TLS certificates are checked.

5.1.4 Data

Are logfiles or other data stored locally? Is the data and configuration deleted after unprovisioning? Usually cloud devices do not require to store information, like usage or sensor data, on the local device. Also the devices should not store unnecessary logfiles. It is especially important that all sensitive data (e.g., credentials, logfiles) are deleted after the device is unprovisioned or a factory-reset was triggered. Otherwise the new owner of the device could extract that information and potentially abuse it.

5.2 DEVICE ANALYSIS

5.2.1 *Xiaomi Mi Robot Vacuum Cleaner Gen1*

The Xiaomi Mi Robot Vacuum cleaner was released in late 2016. While the device was designed for the market in mainland China, it was quickly imported unofficially by users in Europe and Russia in particular. One of the reasons for this was the very competitive price of 350 Euros compared to 800-1000 Euros for similar models from other companies. There was a huge interest of users to modify the device and change the sound language from Chinese to English or even other languages. First attempts date back to early 2017, mainly driven by users in the German forum roboter-forum.com¹, however, no-one was able to extract the firmware or more information for a long time. A global version of the model then was released in 2018, which was sold in the rest of Asia and Europe officially.

While the device was marketed under the brand Xiaomi, it was designed and manufactured by the Chinese company Roborock, which acts as the Original Design Manufacturer (ODM). Roborock received investments from Xiaomi, however, is still an independent company. The device was advertised to operate with 3 different processors, one of them a quad-core System-on-a-Chip (SOC).

For this thesis I purchased three devices: one for disassembly, one for testing a potential rooting method, and one device as a reference. More devices have been purchased later with one month delay between each other to investigate potential modifications of the hardware.

The further approaches on this device are documented in my [dustcloud-documentation](https://github.com/dgiese/dustcloud-documentation) repository under `rockrobo.vacuum.v1`².

First, a TCP port scan using `nmap` over all ports showed no open ports for the device. A UDP portscan revealed only the default `miIO` port `54321`. Therefore the next chosen approach was to analyze the USB connectivity of the device. It was observed, that the firmware updates are downloaded over an Hypertext Transfer Protocol Secure (HTTPS) connection. Intercepting this connection using a man-in-the-middle attack failed due to the correctly implemented TLS certificate verification on the device. The rest of the network traffic was encrypted. The only traffic that was observed was `miIO` cloud traffic

¹ <https://www.roboter-forum.com/index.php?thread/18752-das-system-hinter-dem-robo>

² <https://GitHub.com/dgiese/dustcloud-documentation/tree/master/rockrobo.vacuum.v1>

and some TLS traffic.

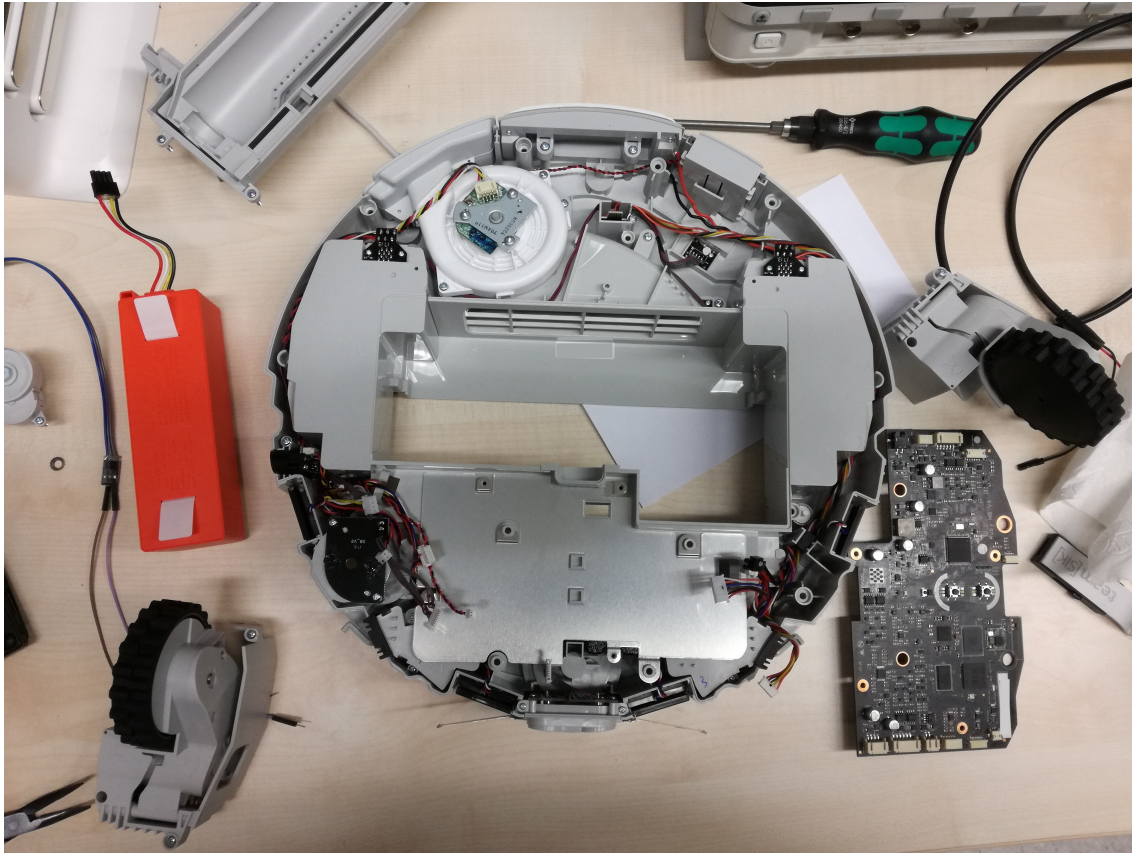


Figure 6: Disassembled Mi Robot Vacuum Cleaner

After disassembly of the device, the Printed Circuit Board (PCB) was analyzed to determine the components used for the device (as seen in Figure 6). The disassembly required to break multiple warranty stickers. The front side of the mainboard PCB can be seen in Figure 7.

It was discovered that the main processor is a Allwinner R16. This SOC is a special version of the Allwinner A33 produced by the Chinese company Allwinner [41]. The R16/A33 is a ARM Cortex-A7 Quad-Core which can be found in many devices like tablets or even the Nintendo NES Classic Edition [38]. The SOC has access to 512 MByte DDR3 RAM and 4 GByte eMMC flash. The flash is connected over Secure Digital Input Output (SDIO). The SOC, the RAM and the Embedded Multimedia Card (eMMC) flash are Ball Grid Array (BGA) chips. For connectivity there is a Realtek RTL8189ETV single-band Wireless Local Area Networking (WLAN) module, which is also connected to the SOC via a separate SDIO interface.

The main PCB contains an additional processor, which is responsible for the real-time tasks like sensor control. This processor is a STM STM32F103VCT6, which is a ARM Cortex-M3. As a sidenote: Another processor is built into the Lidar module, a Texas

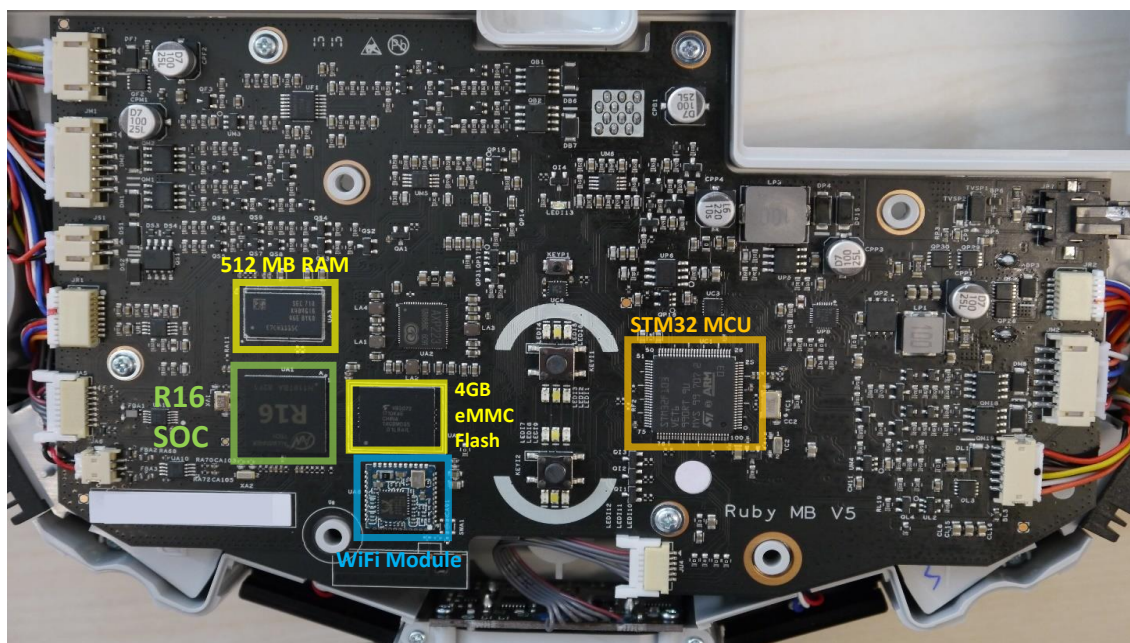


Figure 7: Front side of the mainboard PCB for the Mi Robot Vacuum Cleaner

Instruments TMS320F28026 (which is not on the main board). The main board is connected via cables to various components of the vacuum cleaner: to the tire motors, the fan, to the battery, to the power connectors on the back of the device, to a micro USB connector and to other sensors (ultrasonic sensor, compass, accelerometer, IR distance sensors and bump sensors).

On the back side of the main PCB there are many test-points, however, none were marked as obvious Universal Asynchronous Receiver Transmitter (UART) or Joint Test Action Group (JTAG) pins. At the time of the initial analysis those debug pins were unknown. Due to the large number of test-points (over 50), I decided to find an alternative way to root the device. The goal was to extract the firmware from the device without destroying the device.

While connecting to the USB port of the device, it was detected as an Android Debug Bridge (ADB) interface by the connected computer. However, interaction with the addb service on the vacuum cleaner was not successful as it appeared that additional authentication was required. Various tries to send malformed commands to the device were not successful.

As a last non destructive method it was planned to trigger the "FEL" mode on the device. The FEL mode is a low-level subroutine contained in the BootROM of Allwinner SOCs [16]. This mode is hardcoded in the SOC so it does not rely on any external flash media or ROM. According to the User Manual of the SOC the FEL mode can be triggered if one of this conditions are met: the "Boot-selection-pin" (BSP) is pulled to ground, no valid boot image is detected on the connected flash media or the FEL mode is triggered by software [41]. A similar attack invoking the FEL mode was used to ex-

tract the firmware of the Nintendo NES Classic console [38]. Due to the nondestructive approach and missing knowledge about the testpoints, the position of the BSP was unknown. All the physical buttons of the device were connected to the STM32, therefore the exact same approach like the Nintendo NES Classic console hack did not work. So the new idea was to prevent to the SOC to load a valid image from the eMMC flash. Due to BGA packaging, it was not possible to access the pins of the flash directly. However, after analysis of the traces on the PCB and chip pinout in the datasheet of the SOC, a different approach was considered: corrupt the data by grounding the data lines to the eMMC flash under the BGA chip. The user Florian Lentz, known under the nickname "Flole" developed a alternative app for the device and was discussing a similar approach to get access on the device in February 2017 [37], however, his assumptions were incorrect with regard to the circuitry and the usable tools.

Using the datasheet [40] and the observed traces, I created a tentative pinout for the chip. As seen in Figure 8 the data lines to the eMMC are at directly at the edge of the SOC. The datasheet of the R16 contains the dimensions of the SOC, which can be seen in an extract in Figure 9. The dimension A1 states the space between PCB and the chip, which is 0.3mm [40]. While this size is too small for an usual wire, one layer of aluminum foil would fit. This is, to my knowledge, the first documented use of aluminum foil to attack a SOC in this way. This attack is more risky as the usual fault-injection attacks on NAND chips in Thin Small Outline Package (TSOP) packaging.

Using the aluminum foil to shortcut the data pins after powering on the device, it was possible to corrupt the data in a way so that the SOC could not load a valid image from the eMMC. This led the SOC to fallback into the FEL mode. After connecting to the USB connection the FEL mode was successfully using the sunxi-tools³. Confirming the FEL mode, the aluminum foil was removed. The FEL mode itself does not allow to access the eMMC and also does not enable the DRAM. To be able to use the DRAM, it is required to enter the FES mode. The FES mode is an additional low-level USB interface on top of the FEL mode which enables to flash the device or to run images [16]. This mode can be enabled by loading a FES image, similar to a first-stage boot-loader, via USB onto the device. This FES images are usually only available in special firmware updates for devices, like smartphones, which are meant to be updated via USB. As no such updates are available for the vacuum robot, many different FES images of smartphones were tried, of which most did not work as they were hardware dependent. However, a FES image extracted from the firmware update package of the WINTOUCH Q75S smartphone, which is based on an Allwinner A33, did work and successfully initialized the FES mode. Using the modified U-Boot bootloader from the same smartphone, it was possible to enable the eMMC interface and extract the contents of the eMMC flash via the USB connection. Due to the used protocol, the extraction of the 4 GByte flash took 1 hour and was not stable. After multiple tries, which required power-cycles and repeated use of aluminum foil, a full image was successfully created.

The extracted partition layout can be found in Table 6. The device contains 3 copies of the operation system: one recovery copy, an active copy and a passive copy. The

³ <http://linux-sunxi.org/Sunxi-tools>

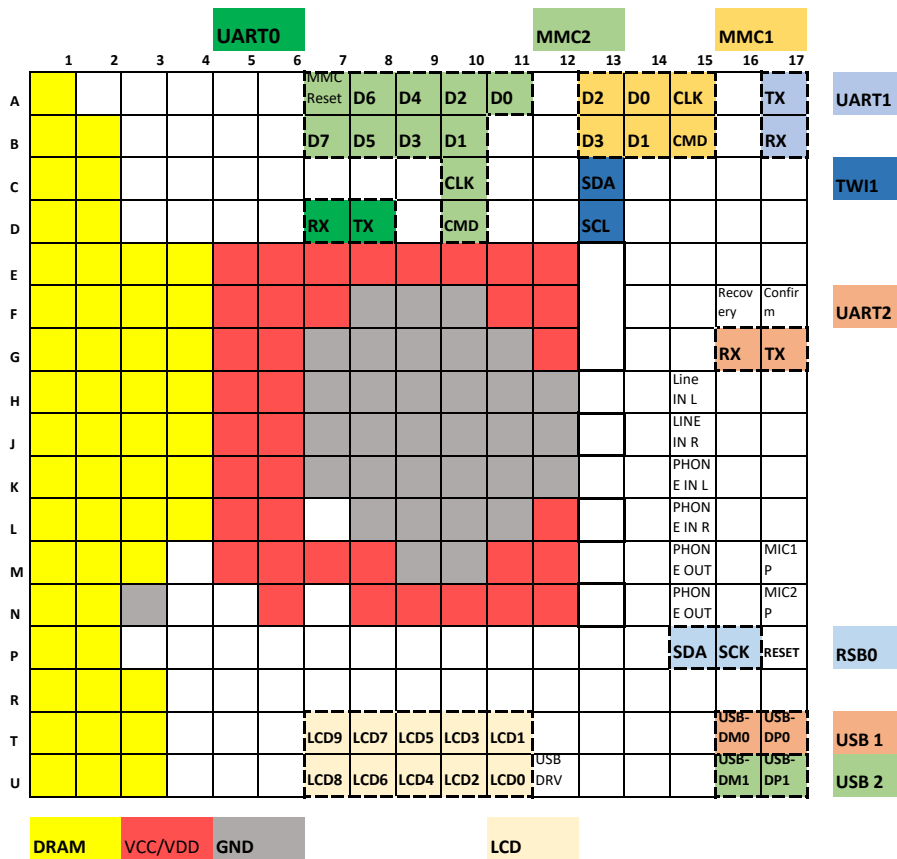


Figure 8: Reverse Engineered Pinout of the R16 SOC in the Mi Robot Vacuum Cleaner

recovery copy is never updated. The active and passive copy have the same contents. The operation system running on the device was Ubuntu 14.04.3 LTS and is stored in an EXT4 file-system. Most parts of the OS were not touched. Instead the developers added most of the custom software under the path `/opt`. There is an "app" partition, which is mounted as read-only under `/mnt/default`. This partition contains the device configuration and device key. Another partition, "UDISK" is mounted under `/mnt/data` and contains log-files and maps.

It was found that the SSH daemon is running but is blocked by an iptables rule. For navigation the open-source software `player`⁴ is used. Also the listening ports of `player` are blocked by iptables. The iptables rules are set by a custom `watchdog` process, which also starts all the other custom software. As a first step, the iptables rules were disabled and an "authorized_keys" file was added under `/root/.ssh`. The modified image was then written back using again the FEL mode. No special protections like SecureOS or SecureBoot were found. As the device did not verify the integrity of the file-system image, the modification was successful and Secure Shell (SSH) access was now possible. It was noticed that it is always possible to restore the original state of the Operating System (OS) by issuing a factory reset (by pressing the Home and reset button simulta-

⁴ <http://playerstage.sourceforge.net/>

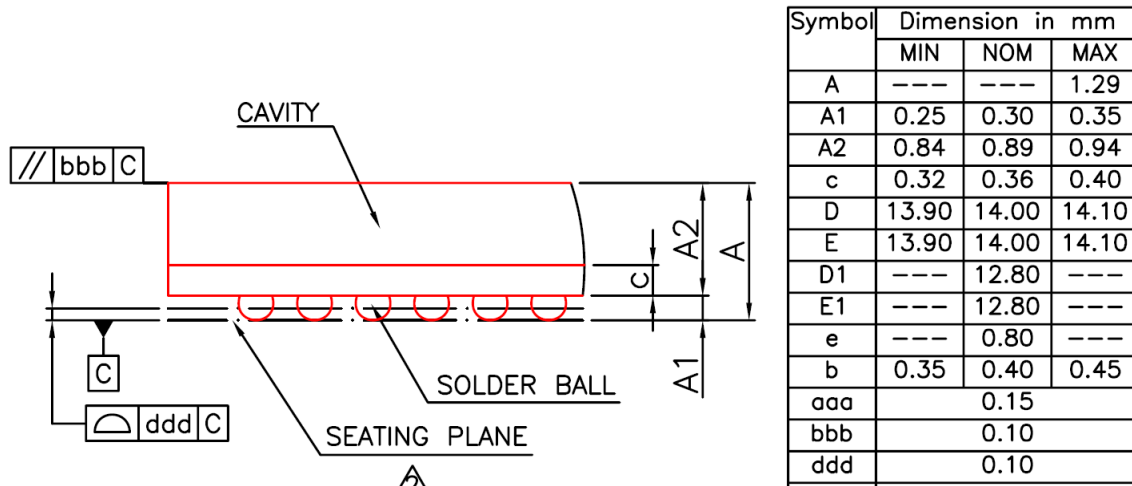


Figure 9: Sideview of the R16 SOC with dimensions from the R16 Datasheet [40]

Label	PartionID nand{}	Size in MByte	Start address
boot-res	a	8	0x00008000
env	b	16	0x0000c000
app	c	16	0x00014000
recovery	d	512	0x0001c000
system_a	e	512	0x0011c000
system_b	f	512	0x0021c000
Download	g	528	0x0031c000
reserve	h	16	0x00424000
UDISK	i	~1900	0x0042c000

Table 6: eMMC partition layout for Mi Robot Vacuum Cleaner

neously). The factory reset overwrites the active and passive copy of the OS with the recovery one, independently of any firmware update. No downgrade protection was implemented.

Having shell access on the system, it was now possible to identify the responsible processes. The identified processes and their functions can be found in Table 7. For a remote rooting method it is necessary to analyze the firmware update process. Therefore all the processes have been analyzed using the IDA Pro 7.0 disassembler and decompiler, and additionally using GDB.

The most important process is WatchDoge, as it sets up the environment and starts all other robot processes. WatchDoge also monitors the processes and restarts them in case of errors or unexpected behaviour. It also keeps a counter for occurred errors and restarted processes. If a particular threshold is reached, it sets a flag in the boot-loader

Process name	Function
WatchDoge	Watchdog, runs and monitors all other processes
miio_client	Connects to Mijia cloud, proxy for internal and external messages
AppProxy	Logic for cloud commands, controls robots behaviour
RoboController	Initializes sensors and functions on the robot
player	Navigation, communication to STM32
rrlogd	Collects logfiles and uploads them to the Mijia cloud
SysUpdate	Only active while system updates, handles update process

Table 7: Responsible processes running on Mi Robot Vacuum Cleaner

to switch to the secondary copy of the OS or even restore the device from recovery.

For communication between the internal components of the robot and the Mijia cloud, there is the `miio_client`. This software originating from Xiaomi's Mijia Software Development Kit (SDK) was not modified. This process is using the device configuration and key material to establish a secure connection to the Mijia cloud. Other than that, the `miio_client` is only responsible to set up the initial WLAN connection. It was noticed that the forwarded commands to the internal components do not contain an information whether the command was issued from the cloud (using the cloud key) or from the app (using the token). This is expected behaviour for that SDK version.

The cloud logic is implemented in the `AppProxy` process. This process contains all supported commands and triggers the corresponding actions. In particular it communicates with the `RoboController`, `player`, `rrlogd` and `SysUpdate` via named pipes. It is also responsible to upload map files and handling the sound packages. In case of firmware updates, it invokes the `SysUpdate` process and transfers the parameters. The supported commands of `AppProxy` can be found in my `dustcloud-documentation` repo⁵.

While the `RoboController`, `player` and `rrlogd` process are important for the function of the device, I believe that they are less relevant for a potential rooting possibility. The reason for this is, that this processes do not expose external interfaces and are invoked by other processes.

To understand the firmware update, the relevant processes have been further analyzed and monitored. It was seen, that the `AppProxy` receives the update command and starts the `SysUpdate` process. `SysUpdate` then uses the URL from the update command to download the firmware from Xiaomi's Content Delivery Network (CDN). For the download the CURL library⁶ is used and the TLS certificates are checked. After downloading the file, `SysUpdate` computes the MD5 checksum of the firmware file and compares it to the MD5 checksum in the update command. If it matches, the firmware file is decrypted and extracted to the download partition. In the next step, the default root password is

⁵ <https://GitHub.com/dgiese/dustcloud-documentation/blob/master/rockrobo.vacuum.v1/app-proxy-commands.txt>

⁶ <https://curl.haxx.se/>

replaced by a device specific one. The specific password is stored obfuscated in the file `/mnt/default/vinda` and is XOR'ed with the ASCII "7". After setting the password, the content of the download partition is written onto the passive copy of the OS. Then the device reboots into the updated, former passive copy, now active copy partition. Finally the former active copy is updated.

It was seen in the update process, that the firmware update packages were encrypted but not signed. That means that everyone who knows the encryption key can create new firmware packages or modify existing packages. As AppProxy and SysUpdate are not aware if the update command was received from the cloud or the app, this can be used to remotely push a firmware update onto the device and root it.

To find the encryption key and method, the SysUpdate binary was loaded into IDA Pro and a search for potential interesting strings was conducted. As shown in Figure 10, all the relevant strings for verification of the integrity and decryption were very close together. It was found that the open-source software CCrypt ⁷ was used to encrypt and decrypt the firmware image. CCrypt uses 256-bit Rijndael (AES) encryption. The hard-coded encryption key is "rockrobo". The decrypted firmware package contains a full EXT4 file-system.

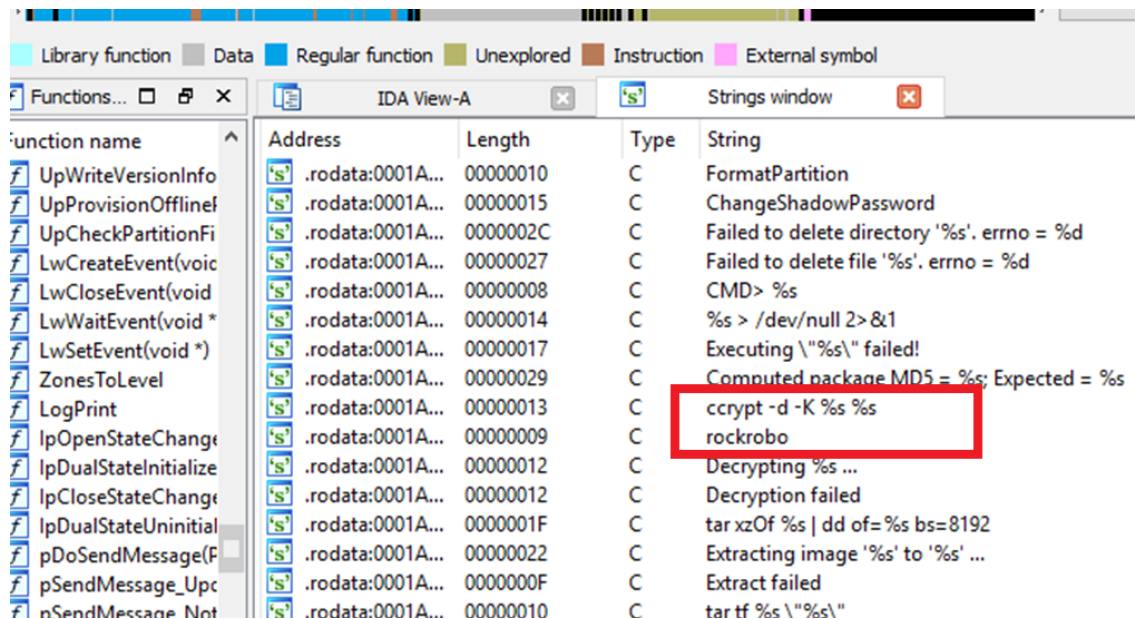


Figure 10: Screenshot of IDA Pro inspecting the Strings of the SysUpdate binary

Using the encryption key, it was now possible to decrypt existing firmware updates and create modified updates. As the same encryption method is used for the sound packages, the encryption key "rockrobo#23456" was found. Now it is possible to push a custom firmware and sound packages onto the vacuum robot using the token without opening the device. The token can be extracted from the Xiaomi Home app or

⁷ <http://ccrypt.sourceforge.net/>

directly queried from an unprovisioned device. This rooting method is applicable to all Mi Robot Vacuum Cleaners. The URLs of the firmware files were predictable and could be enumerated as soon as one URL is known. The reason for this is the file format "v11_00xxx.pkg" where "xxx" is the version number, e.g., "3096". So if the URL "https://cdn.awsbo.fds.api.mi-img.com/upd/pkg/v11_003096.pkg" is known, the number can be incremented and the version "v11_003132.pkg" can be found. In 2019 an additional, unique String was added before the .pkg extension, making it harder to guess the correct filename.

Having root access on the device made it possible to create a signal on the UART ports. Using that it was possible to identify the UART ports among the test points. Having access to UART simplifies debugging and enables easy interaction with the U-Boot boot-loader. Additionally, with the knowledge of the root password (which can be deobfuscated from the "vinda" file as mentioned earlier) access independent of the WLAN network is possible. Figure 11 shows the soldered UART connection. It has proved to be advantageous to additionally use hot glue to secure the soldered connections against physical stress and vibrations while operating the device.

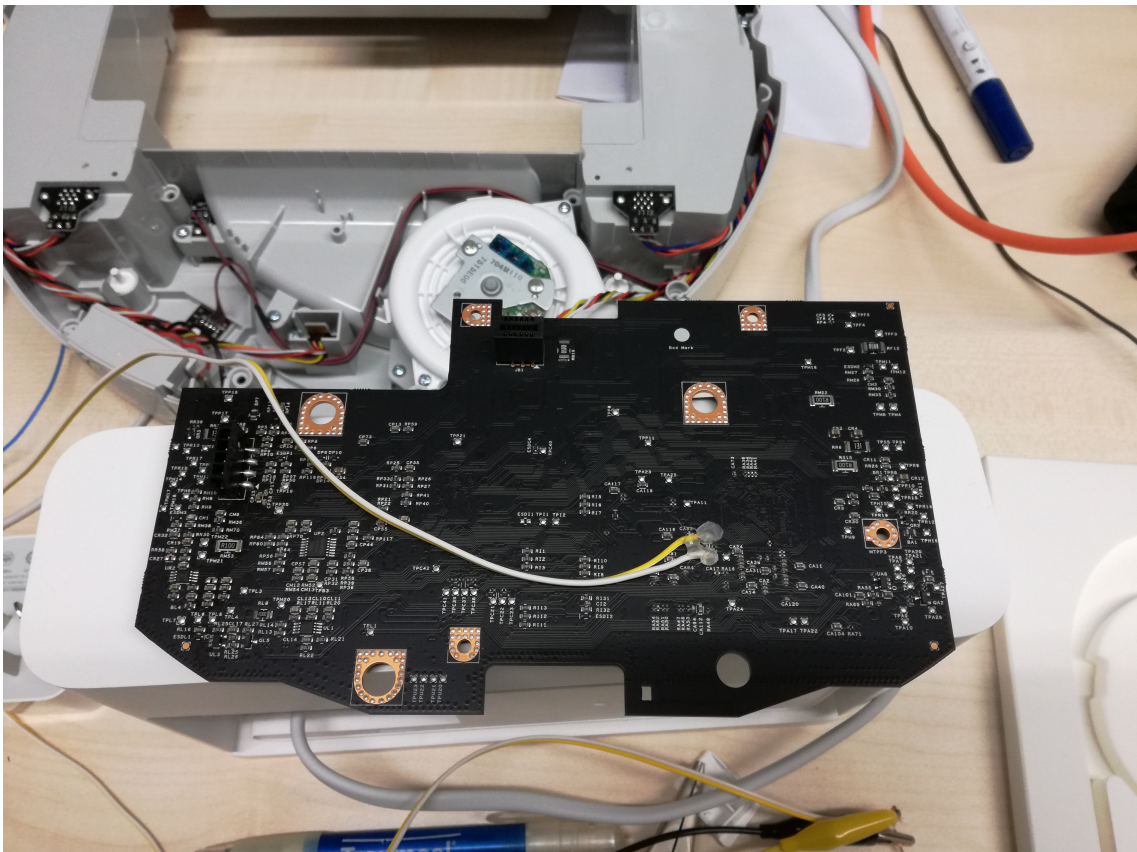


Figure 11: Method of soldering the UART pins on Mi Robot Vacuum Cleaner

After gaining root access, it was now possible to further analyze the data collected and stored on the device. The component of interest were the ultrasonic distance sensor

and the speaker, as they are potential possibilities to be used as microphones to snoop on the user. It was analyzed whether the receiving part of the ultrasonic distance sensor can be accessed as a microphone or if the whole module just acts as a switch. Due to the design of the distance module it is assumed that it acts as a switch. In addition it is connected to the STM32 via a normal General-Purpose Input/Output (GPIO), therefore it would be difficult to sample audio signals. The vacuum robot offers audio feedback using a speaker, which is connected to the R16 SOC. It was verified empirically and via the R16 user manual [41] that the integrated audio codec in the R16 does not offer the retasking feature of the audio output pins and should be not able to use the speaker as a microphone as defined in the "HD Audio" standard [23].

Multiple devices have been operated over multiple months and the collected data has been analyzed. The device creates while operating temporary maps, which are used for computing the navigation by the player process. These maps are bitmaps and each pixel represents an area of 5cm times 5cm. An example for such map can be found in Figure 12. The same maps are uploaded to the using TLS to the FDS cloud as they are used by the "Xiaomi Home" app to show the progress of the device while operation to the user.

To track the changes in firmware updates, the firmware versions starting version "3.3.6_003061" (Spring 2017) until "3.3.9_003468" (Spring 2019) have been collected. It was noticed that log-files and maps were stored unencrypted on the device and were not deleted in case of a factory reset. The logfiles contain WLAN credentials and other sensible information, like user ID and WLAN Medium Access Control (MAC) addresses. If a device is reset and sold to someone else, the new owner can access the maps and can extract the sensible information from the logfiles. In preparation of this thesis, this issue was reported to the manufacturer. Beginning with version "3.3.6_003514", which was released in September 2018, the log-files and maps were locally encrypted with a public key before they were uploaded into the cloud. Also the data partition was now erased while the factory reset. This happens however, not in a secure way.

To maintain permanent SSH access to the device, it is sufficient to upload a "authorized_keys" in /root/.ssh and to remove the firewall rules in /opt/roborock/watchdog/rrwatchdog.conf. I developed a tool to automatically build custom firmware updates. This tool is published on GitHub⁸.

5.2.2 Roborock Vacuum Cleaner S50

Roborock released in late Fall 2017 the Roborock Vacuum Cleaner S50. In contrast to the Xiaomi Mi Robot Vacuum Cleaner, this device was sold under the brand name Roborock. Therefore Roborock was the Original Equipment Manufacturer (OEM) itself.

The model was prepared from the beginning to be sold globally. In comparison to the Mi Robot Vacuum Cleaner it was more expensive, but offered a sweeping function.

⁸ <https://GitHub.com/dgiese/dustcloud/tree/master/devices/xiaomi.vacuum/firmwarebuilder>

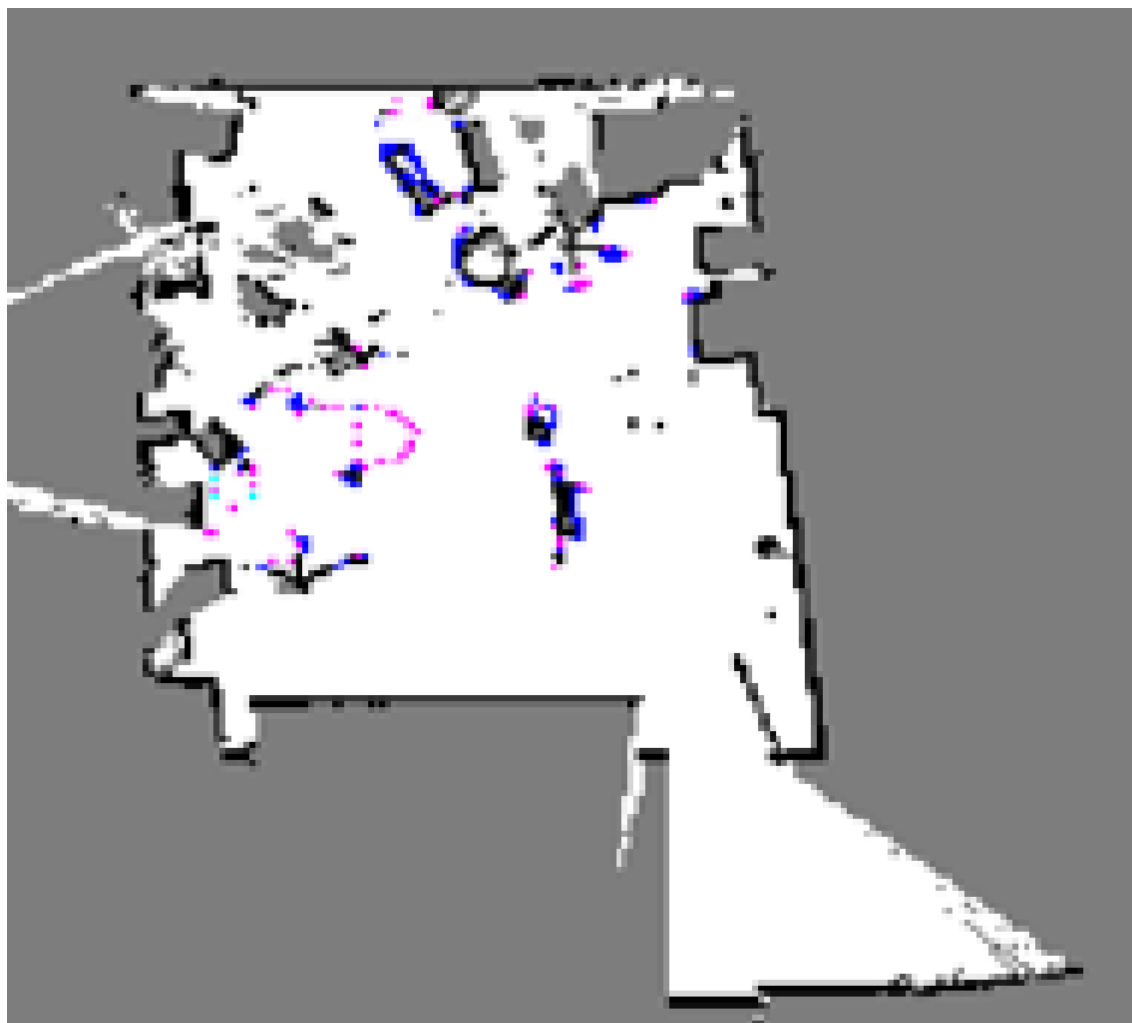


Figure 12: Stored map (scaled) on Mi Robot Vacuum Cleaner

Additionally Roborock claimed the S50 supports cleaning zones and an "goto" function. The company claimed, that this features were only possible on the S50 and not on the Xiaomi Mi Robot due to hardware changes.

For this thesis two devices have been purchased: one global version and one Chinese version. The idea behind the different versions was to understand what the differences between regions are.

At the time of release Roborock was not aware of any rooting method, because the research of this thesis was not published. This fact was confirmed by the company in a meeting in December 2018. Therefore the company reused most components and software of the Mi Robot Vacuum Cleaner. There were no countermeasures in place and even the firmware encryption keys stayed the same.

The pictures and additional supplemental material can be found in my dustcloud-documentation repository under `roborock.vacuum.s5`⁹.

The devices were disassembled similar to the Mi Robot Vacuum Cleaner. The disassembled device is shown in Figure 13. Apart from the new sweeping function, it was noticed, that the USB connection have been moved and that now a different charging method is used. The front side of the PCB is shown in Figure 14. Analyzing the PCB it was confirmed that the same platform as the previous model is used. So the main SOC is the Allwinner R16, with 512 MByte DDR3-RAM and 4 GByte eMMC flash. Also the software is nearly exactly the same as for the Mi Robot Vacuum Cleaner, except for some small changes in regard to changed GPIOs and one additional switch. So everything from the section before still applies, including the rooting methods.

One change which was noticed at disassembly was the fact that the part of the PCB is coated. However, it turned out as typical conformal coating, probably due to the fact that the device can operate with water. In that sense the coating was not a anti-tamper measurement.

The region of each device was defined by a variable in the config file `"/mnt/default-roborock.conf"`. This file was not protected, therefore a conversion from a Chinese model to a global model, and vice versa, was possible. The change of the region had an influence on the used cloud servers for the map and log upload.

In regard to the newly advertised functions, an analysis of the `p1ayer` binary showed, that the limitations of the older model in regard cleaning zones and the `"goto"` function are purely software based. After publishing this result in Summer 2018, the company added the features to the older models.

Similar to the Mi Robot, the firmware versions of the device have been collected from version `"3.3.9_001168"` (Spring 2018) until `"3.3.9_001748"` (Spring 2019). Similar to the previous model, Roborock introduced encrypted user-data and correctly implemented factory-reset in September 2018 with version `"3.3.9_001518"`. However, it is still possible to extract some information from the partition using forensic methods like imaging.

To maintain SSH access on the device, the same methods apply as for the Xiaomi Mi Vacuum Robot. Also my custom firmware update tool works for this model.

5.2.3 *Roborock Vacuum Cleaner S6/T6*

In April 2019 Roborock released the models S6 and T6 as successors for the S5 model series. While the hardware is the same for both models, the T6 model is only intended to be used in mainland china, while S6 is the global model. There is also a huge price difference between both models.

Compared to the S5 there are little changes: the brushes were redesigned and an additional reed contact for the water tank added. Additionally the tires have been replaced.

⁹ <https://GitHub.com/dgiese/dustcloud-documentation/tree/master/roborock.vacuum.s5>

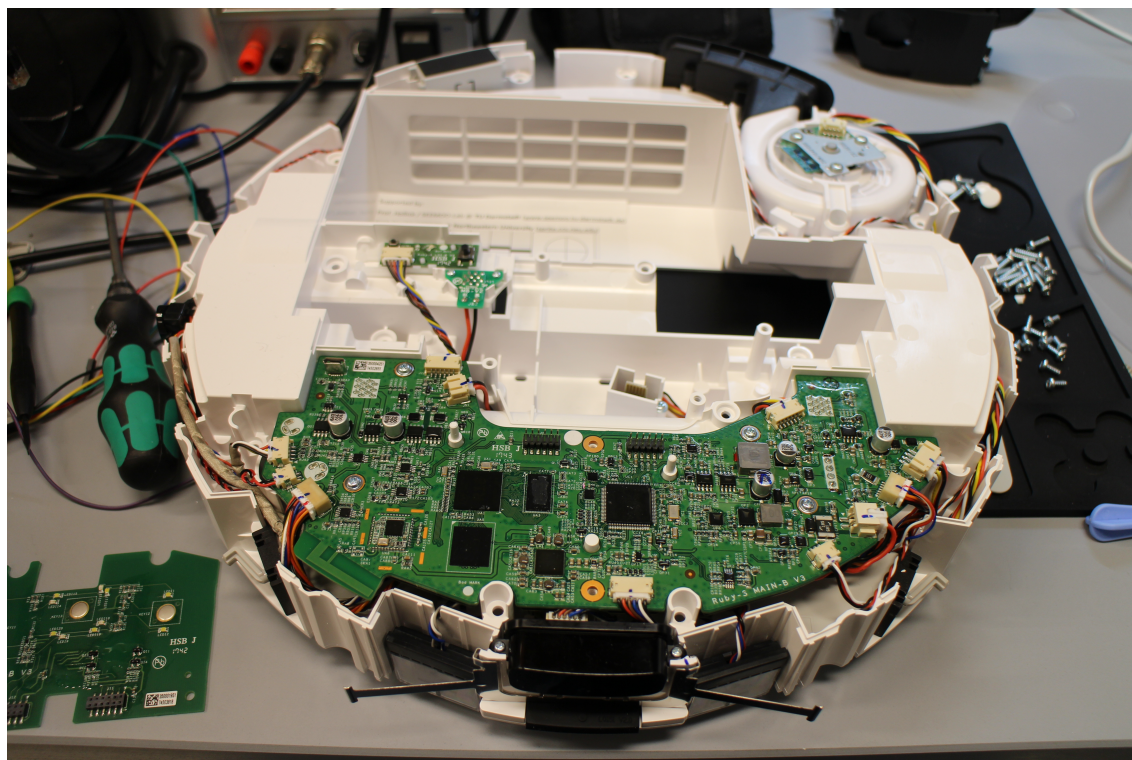


Figure 13: Disassembled Roborock Vacuum Cleaner S50

Feature-wise Roborock added multi-room cleaning and multi-floor maps. The company claimed also that the new model is quieter than the predecessor models.

This models have been designed after the rooting methods and attack vectors for the previous models have been published. Also as part of this thesis there has been a visit in December 2018 of the Rockrobo's headquarters in Beijing with consultations about security methods. Therefore it was expected that most of the existing vulnerabilities and rooting methods would be fixed.

After purchasing the T6 model and importing it from Mainland China, the device was disassembled and analyzed. Most of the case is exactly the same as for the S5 model series. There have been some minor changes to the PCB due to the new type of tires and the new water tank sensor. Additionally, there was a second filter in the fan exhaust. Other than that, the layout of the PCB and hardware specs remained the same.

The pictures and additional supplemental material can be found in my dustcloud-documentation repository under `roborock.vacuum.t6`¹⁰.

While the software analysis, it was noticed that the usual method to push a firmware onto a device did not work anymore. Also the firmware packages were now encrypted differently in comparison to previous models. For further analysis, a connection using

¹⁰ <https://GitHub.com/dgiese/dustcloud-documentation/tree/master/roborock.vacuum.t6>

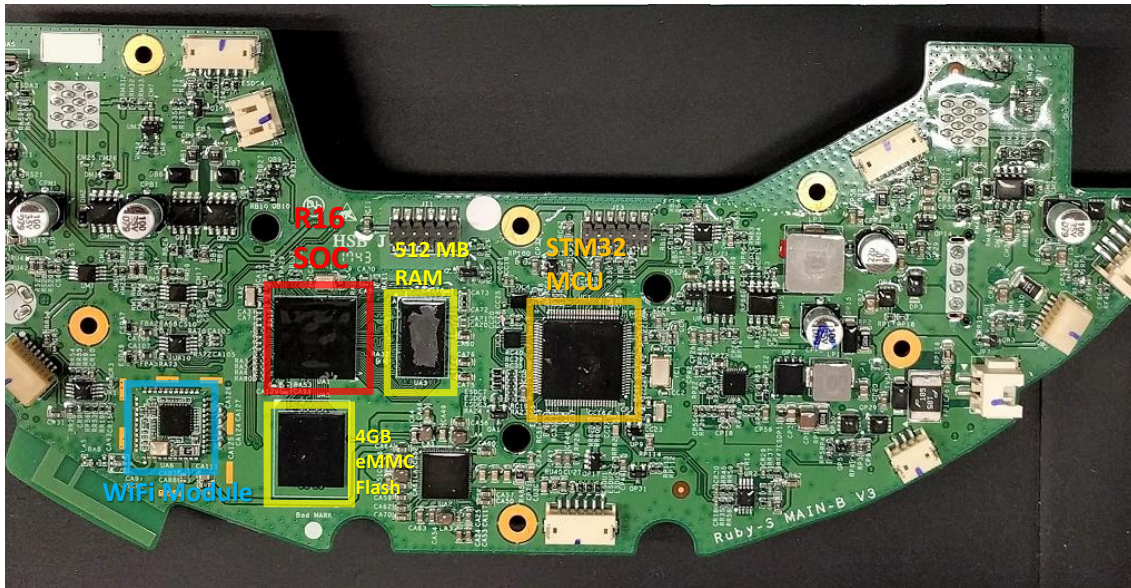


Figure 14: Front side of the mainboard PCB for Roborock Vacuum Cleaner S50

UART was established. The pins used for UART in S5 remained the same in S6/T6. Figure 15 shows the setup to connect to the UART pins. Due to the position on the bottom side of the PCB, it was required to remove the whole board from the case and power it directly over the battery.

It was assumed that the extraction of the obfuscated root password would still work over the U-Boot shell. Therefore the U-Boot autoboot was interrupted by continuously pressing the key "s" while powering on the device. This has given access to the U-Boot shell. The content of the `vinda` file was loaded into the memory and then printed out. The resulting string was XORed with ASCII "7" and the login with the user "root" was possible. However, it turned out, that the login shell was replaced by the custom binary `rr_login`. Therefore there is a risk that this method will be blocked in the future. As an alternative it was tried to change the parameter `cmdline` and change the `init` process to `/bin/sh`. This successfully lead to a temporary root shell after booting the device. Here the kernel armed the hardware watchdog of the R16 SOC and as the `WatchDoge` process was not started, the system rebooted after around 3 seconds. Usually the `WatchDoge` progress will regularly reset the watchdog timer by sending hearthbeats. As this does not happen, I needed to stop the watchdog completely. The Linux kernel has the special device file `/dev/watchdog` for the interaction with the watchdog and the watchdog can be disabled by sending the specific magic character "V" to that device file [54]. After booting the system into `/bin/sh` as the `init` process, the watchdog was successfully disabled, if the magic character has been written to the watchdog in time. After this, the root access was established the same way as for the previous models: removing the firewall rules blocking SSH and creating an `authorized_keys` file.

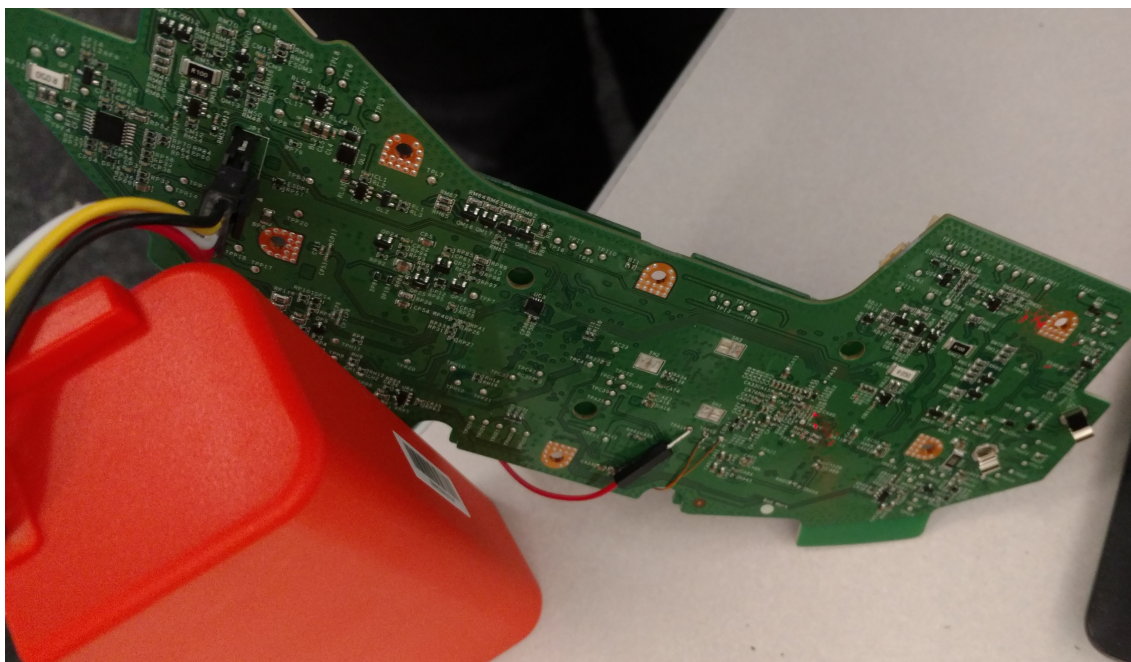


Figure 15: Setup of T6 Mainboard for UART connection

Process name	Function
WatchDoge	Watchdog, runs and monitors all other processes
miiio_client	Connects to Mijia cloud, proxy for internal and external messages
AppProxy	Logic for cloud commands, controls robots behaviour
RoboController	Initializes sensors and functions on the robot
rrloader	Navigation, communication to STM32
rrlogd	Collects logfiles and uploads them to the Mijia cloud
SysUpdate	Only active while system updates, handles update process

Table 8: Responsible processes running on Roborock T6 Vacuum Cleaner

With having SSH access, it was possible create a copy of the contents of the device by imaging the special device file `/dev/mmcblk0`. Also all other information of the device have been copied, e.g., the serial number and the volatile shared memory files in `/run/shm`. Also the processes were analyzed as shown in Table 8. It was noticed, that the `player` process was replaced by `rrloader`. Although the used libraries seem to be the same as for `player`, there is now some new functionality using `OpenCV` framework. It is assumed that `rrloader` is actually a modified `player`.

It was noticed, that some of the configuration files changed. In particular the file `roborock.conf` in `/mnt/default` contained now a new parameter field `"cpuid"`. In addition, there was a new file `roborock.conf.sign`. The binary of `AppProxy` was analyzed using `IDA Pro` to find the changes in relation to the `roborock.conf`. I saw, that before the file `roborock.conf` is parsed, its signature is verified with the file `roborock.conf.sign`.

The function responsible for this check, "rr_sec_verify_conf()" is imported by AppProxy from the library `librrlocale.so`. The actual 2048 bit RSA signature public key is stored obfuscated in the binary by splitting the public key in two parts: every second byte is deleted and is stored in a second string. One half of the public key is shown in Listing 6. By obfuscating the public key, it is not found if someone is searching for the string "`--BEGIN PUBLIC KEY--`". However, this method of obfuscation was not very effective, as I could reassemble the key quickly.

Listing 6: Half of the obfuscated public key in AppProxy

```

---EI ULCKY--\n"
"IBjNgqkGwBQFACQAIbgCQA3U+h+162yV2QYAIz7TBP2i6mhpZcgIXK8g1lNgUVoh\n"
"W0nUActG/QqhoK8Eb818G3A+wCs8kQgm+in1cwWtx4WMXqM1MW957RvavOKJi/mu\n"
"p1mMd4WSVkv/60IhybVpVr2LDYd2AmSPa4YnsBWMn08FvPNc/WU9erXnoCi6hPRL\n"
"QDQB---N ULCKY--"

```

If the check is successful, then the SOC serial number, which is retrieved by kernel Application Programming Interface (API), is compared with the field "cpuid". In case of a success, the rest of the file is parsed and the region settings are set (e.g., logging server or language settings). If the signature or cpuid check fails, then the device falls back to region settings of Mainland China. The region setting defines, whether users outside from Mainland China are being blocked or not. Roborock introduced with that effectively a region lock for their devices, in order to prevent the export and usage of Mainland China devices in the rest of the world. This region lock seems to be triggered if the "Xiaomi Home" app reports a different timezone to the vacuum than "Asia/Beijing". Interestingly Roborock implemented also an online whitelist, where the AppProxy process sends the serial number to the API endpoint `https://apiiot.roborock.com/fwapi/checkwl?sn=%s&cpuid=%s`. This API can also enforce a particular region on a device. However, by patching `librrlocale.so` it is possible to disable the signature check, modify the region and pass the region check.

In the next step I took a look at the firmware update mechanisms. I noticed, that `ccrypt` is not used anymore and instead `mbed TLS` is now used to decrypt the firmware image. In contrast to the previous models, the firmware is now also signed. The signature has a length of 256 bytes and is stored in the last 1028 bytes of the firmware file. The signature is verified using a 2048 RSA public key, which differs from the key used for the configuration files, however, uses the same obfuscation method. To extract the encryption key of the firmware, the function "mbedtls_aes_setkey_dec" was hooked using the IDA Pro debugger. This function is responsible to set the key and the Initialization Vector (IV) for the following decryption operation [64]. The now extracted encryption key and signature key were used to implement a firmware unpacking tool. This tool was also tested against sound packages, which decrypted them successfully. Having signed firmware packages and sound packages makes it impossible to create custom firmware using a similar way as in previous versions. An analysis of the `mio_client` revealed that Roborock modified the program in order to block over-the-air (OTA) update commands from the local API. Instead, only firmware update commands from the cloud are accepted.

Due to the changes in the firmware update process, my tools cannot build a custom firmware update anymore. Also it is not possible to even push the vendors firmware packages via the local API anymore. Therefore the user needs to disassemble the device and connect to UART in order to get SSH access. Same as before, the SSH access does not survive firmware updates and gets removed as the whole partition is overwritten. The only way to restore similar functionality, like for the earlier models, is to patch the public keys in the binary and to replace `miio_client` with an older version.

5.2.4 *Lumi Smart Home Gateway*

The Lumi Smart Home gateway was released in late 2016. It was the core of the Zigbee based products in the Mijia ecosystem. The device had three iterations, the version tested here is "lumi.gateway.v3". The device acted as a gateway between battery operated Zigbee sensors and the Mijia cloud. The connection to the cloud was established by WLAN. Additionally the device offered a webradio function, however, the radio stations were Chinese only. One reason for this is, that the device was never intended to be used outside of Mainland china and was therefore not advertised there.

A peculiarity of this device is, that the vendor Lumi did offer a local API, where some functions of the device could be controlled without interaction with the Mijia cloud [15]. This API was documented and used by many users to interact with the Zigbee sensors. However, this API still required that the gateway is connected to the Internet. Otherwise the local API would not be started. In addition, the local API needs to be enabled in the "Xiaomi Home" app, therefore a cloudless usage of the device is not possible [59].

From this model there have been 3 devices purchased. Here the idea was to destroy one devices in order to get all the necessary information to get access on the other devices.

My network scan revealed the usual `miIO` service on port 54321(udp) and additionally the local API port 9898(udp). However, both ports were not usable in order to get privileged access on the device. An analysis of the traffic revealed, that while the local API and Mijia traffic was encrypted, the firmware updates were not. The firmware updates for the device itself and for the subdevices(e.g., sensors) were sent in plain-text over HTTP. In order to prevent an actual firmware update, the download of the firmware has been blocked. Therefore the device remained on the older version.

One of the intercepted firmware files was "upd_lumi.gateway.v3.bin" with the MD5 "94571c41d164cc84a8a5ba249c44629e". This represents the version "1.4.1_150". Another intercepted file was "mcu_lumi.gateway.v3.bin", which represents the firmware for the Zigbee controller and will not be further analyzed here. Upon analysis of the file "upd_lumi.gateway.v3.bin" in HxD hex editor, I noticed, that the header contained "MRVL" as the file magic. I found that this header was used for Marvell 88MW300 and 88MW302 SOCs. A public version of the SDK was available¹¹. This SDK included example projects and the tools necessary to build a firmware. Especially the tool `axf2firmware` was in-

¹¹ <http://marvell-iot.GitHub.io/>

Byte	0-3	4-7	8-11	12-15	16-19
0x00000000	Magic	Magic	Timestamp	# of segments	entry address
	4D 52 56 4C	7B F1 9C 2E	FF BE A8 59	03 00 00 00	19 37 00 1F
	MRVL				0x1f003719
0x00000014	segment magic	offset in file	size of segment	mem addr	checksum
	02 00 00 00	C8 00 00 00	50 36 00 00	00 00 10 00	20 C8 51 7D
		0xc8	0x3650	0x100000	
0x00000028	segment magic	offset in file	size of segment	mem addr	checksum
	02 00 00 00	18 37 00 00	28 15 08 00	18 37 00 1F	0A 11 25 85
		0x3718	0x81528	0x1f003718	
0x0000003C	segment magic	offset in file	size of segment	mem addr	checksum
	02 00 00 00	40 4C 08 00	54 19 00 00	40 00 00 20	FB 5F ED 39
		0x84c40	0x1954	0x20000040	

Table 9: Reverse engineered format of Marvel 88MW30x firmware binaries

teresting, as it converts a binary from the ELF-format into the vendor specific firmware binary format. In this case the difficulty was that I had only the final firmware binary, but not the executable file with the debug symbols and memory allocations. It was required to reverse engineer `axf2firmware` in order to understand the firmware format. The result can be seen in Figure 9.

I created a parser for Marvel firmware, which converts the firmware binary format into ELF files. The parser creates the ELF header and reassembles the corresponding sections. This enables to load the binary into IDA Pro and further analyzing it. The parser is published on GitHub¹². With the ability to create and reverse the building process of the proprietary firmware format, I tried to modify the firmware. For that I changed the version number string in the ELF binary and created the proprietary firmware binary again. To test if the device verifies the integrity of the firmware, I redirected the Domain Name System (DNS) name `"cdn.cnbo.fds.api.mi-img.com"` to my own webserver and placed the modified binary under the same name as the original firmware. After triggering the firmware update from the "Xiaomi Home" app, I noticed, that the update was successfully installed and the "Xiaomi Home" app showed now my version number string. This meant, that the integrity of the firmware is not verified and therefore firmware files can be easily swapped. As the firmware is downloaded over plain-text HTTP, this offers a very good way to install a custom firmware. However, the limitation is here, that there must be a firmware update available for the device as the "Xiaomi Home" app does not allow updates for devices on the latest version.

In the next step, the device was disassembled carefully. For this 3 screws on the bottom of the device have to be removed. One of the screws had a small warranty sticker on it, which was damaged in the process of opening the device. The dangerous aspect of this particular device is, that it also contains the power supply, which is directly connected to the mains. Therefore there is a potential dangerous voltage of 110V/220V

¹² https://GitHub.com/dgiese/dustcloud-nexmon/tree/master/firmwaretools/Marvel_MW30x

on the power supply PCB. A picture of the opened device can be found in Figure 16. As seen here, the part with the actual SOC and the Flash memory is hidden under the Electromagnetic Interference (EMI) shield. This shield was soldered onto the module and could not be removed easily. Also on the PCB can be found a NXP JN5169 chip, which is a Zigbee Microcontroller Unit (MCU) [63].

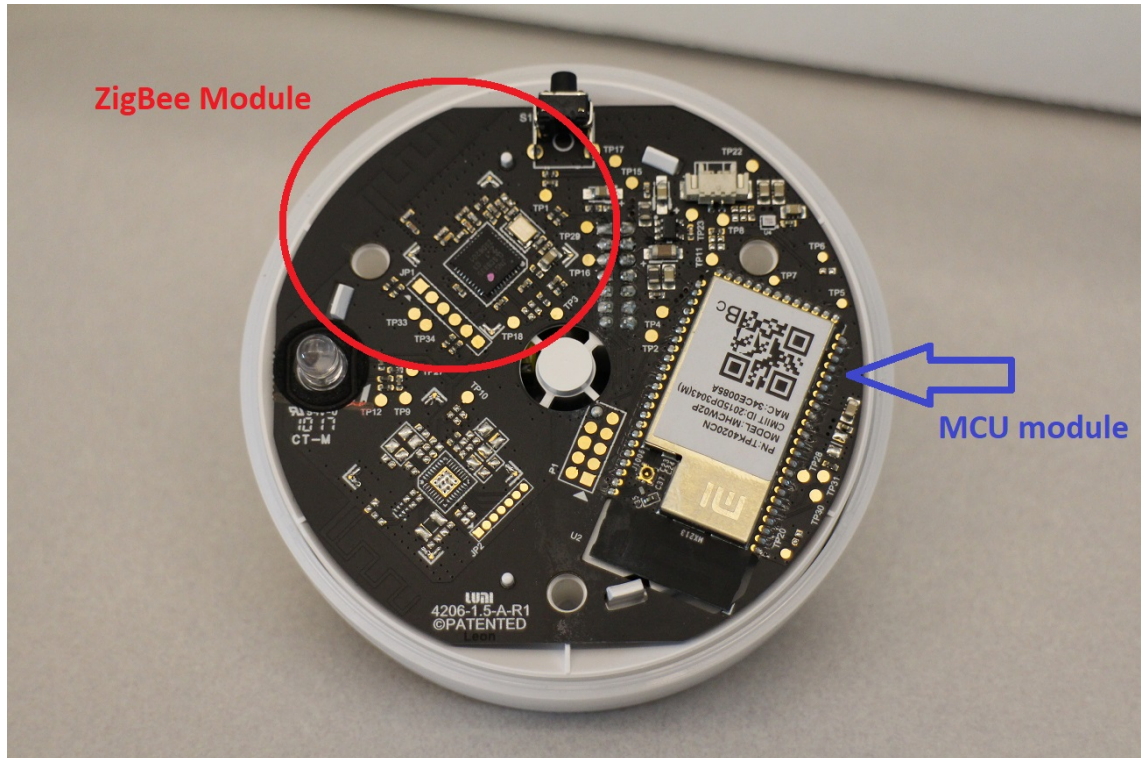


Figure 16: Front side of the PCB in Lumi Smart Home Gateway

To avoid thermal damages, the shielding was carefully removed using a rotary tool. In particular it is required to not cut too deep into the metal in order not to damage any of the parts inside the shielding. After removing the shielding, the parts inside the module have been analyzed. The SOC was identified as Marvell 88MW302, a Cortex-M4F processor. Using the datasheet of the chip [1] and the multimeter it was possible to map some of the pins to the outside test-points. The challenge here was the size of the chip which required a microscope in order to properly connect to the corresponding pins. Additionally the 16 MByte Serial Peripheral Interface (SPI) flash was accessible. I desoldered the flash chip and created an image. The partition layout was parsed and is shown in Figure 17. In contrast to the Roborock vacuum cleaners, here the device credentials have not been found on the flash. Therefore I assumed, that they are stored in the One-Time-Programmable (OTP) memory. 18 shows the PCB after analysis and with the mapped test-points. Especially important for the further research is the availability of the Serial Wire Debug (SWD) pins, as it enables direct memory access.

Having access to the UART enabled access to a shell on the device. The OS running on the device is a Real-Time Operating System (RTOS), which is containing every function in one single binary. While the OS supports some interesting commands, there is no

```

===partition table:
magic:0x54504d57
version:1
partition entry no:9
gen_level:0
crc:0x2830200f
===partition info:
device:0 gen_level:1 name:boot2 size:24576 start:0x0 type:0
device:0 gen_level:1 name:psm size:16384 start:0x6000 type:4
device:0 gen_level:1 name:appfw size:614400 start:0xa000 type:1
device:0 gen_level:1 name:userdata size:40960 start:0xa0000 type:6
device:0 gen_level:1 name:mcufw size:393216 start:0xaa000 type:5
device:0 gen_level:1 name:wififw size:196608 start:0x10a000 type:2
device:0 gen_level:1 name:wififw size:196608 start:0x13a000 type:2
device:0 gen_level:1 name:appfw size:614400 start:0x16a000 type:1
device:0 gen_level:1 name:musicfw size:14680064 start:0x200000 type:7

```

Figure 17: Partition information extracted from SPI flash of Lumi Smart Home Gateway

option to print the device’s cloud key. Only the last 2 bytes of the cloud key are shown. A sample output of the console can be found in my GitHub repo¹³.

To get more information about the memory layout, a Raspberry Pi was connected to the SWD of the gateway [2]. I used OpenOCD to get access to the memory of the device and extract memory dump. The required parameters for the connection were available in Marvell’s GitHub repository¹⁴. I found that the device ID, key and MAC are stored beginning at the address 0x2000037E. As shown in Figure 19, firmware binary itself contains at this address placeholders. This means, that the credentials are copied to this area after booting the device and it was confirmed that they are stored in the OTP memory. Also it was confirmed, that my firmware parsing tool actually parsed the firmware and assigned the segments correctly.

Having the firmware and the memory layout now enables to create a modified firmware. To create this firmware, Nexmon will be used. As a preparation the required functions needs to be identified and mapped.

To exfiltrate the device credentials without opening the device and send them to a remote server, some functions were required: `bin2hex`, `snprintf` and `http_get`. `bin2hex` is necessary to convert the credentials, which are stored in binary format, into readable hex representation. `snprintf` is used to build a string, like an Uniform Resource Locator (URL) with parameters. `http_get` makes the HTTP GET request using a given URL. My idea was to use parameters in HTTP GET requests to exfiltrate the credential information to my own webserver. The limitation here is that the device only supports HTTP and not HTTPS.

To get the positions of the above functions, I compiled the examples with debug-symbols from the Marvell SDK and used the IDA Pro plugin `bindiff`. I used it to match

¹³ <https://GitHub.com/dgiese/dustcloud-documentation/blob/master/lumi.gateway.v3/serialconsole-sample.log>

¹⁴ <https://GitHub.com/marvell-iot/ez-connect-lite/tree/master/sdk/tools/OpenOCD>

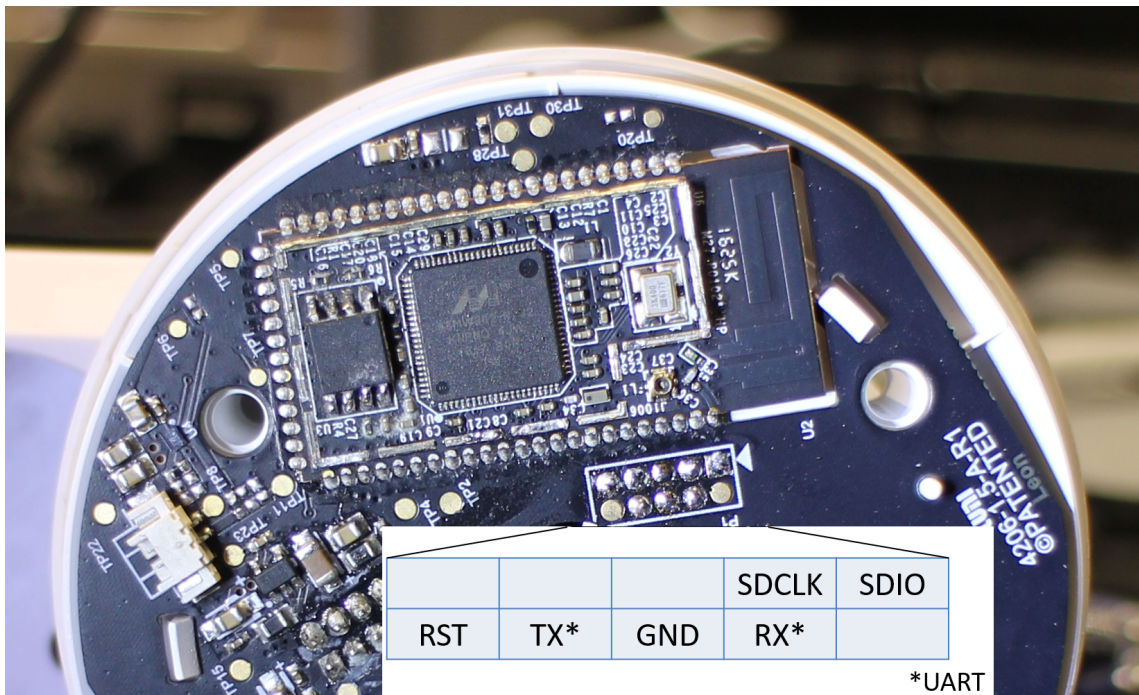


Figure 18: Debug pins on PCB in Lumi Smart Home Gateway

the known functions in the firmware binary with the ones in the compiled examples. Additionally I found compiled libraries of the miIO SDK in a GitHub repository¹⁵. The libraries contained the debug symbols of the basic miIO functionality. After comparing all available binaries and libraries with the gateway firmware using `bindiff`, I got many matches, as can be seen in Figure 20.

With all the found functions, I created a patch which read the credentials out of the memory and placed them as parameters in a HTTP GET request. I hooked this function at the `sprintf` call in `otu_timer_info`. I have chosen `otu_timer_info` as this function is called whenever a successful connection via UDP to the Mijia cloud is established. There is also a TCP variant of this function which is called `ott_timer_info`. The UDP function is tried at first and if it fails, then the TCP function is called. That way I can control from outside whether my patch is executed by blocking or allowing the UDP connection to the Mijia cloud. This is important in order to test the functionality of my patch without permanently disabling the device. The patch and a modified version of the Nexmon framework can be found in my "dustcloud-nexmon" repository¹⁶.

After I had all the information for my patched firmware, I created a binary with the exfiltration function. I modified the version number in my firmware to represent an older version in order to be able to trigger the update process from the "Xiaomi Home" app over and over again. I placed the binary on my webserver and triggered the update process. Due to my DNS redirection from before, the device downloaded and installed

¹⁵ <https://GitHub.com/caoguanqiu/midir>

¹⁶ <https://GitHub.com/dgiese/dustcloud-nexmon/>


```

IDA View-A | Hex View-1 | Structures | Enums | Imports | Exports
.
.text3:2000037C DCB 0xA
.text3:2000037D byte_2000037D DCB 0 ; DATA XREF: sub_1F006360+2fW
.text3:2000037D DCB 0 ; sub_1F00636C+3ETr ...
.text3:2000037E aTag_mac DCB "TAG_MAC",0 ; DATA XREF: sub_1F003FE0+18f0
.text3:2000037E DCB 0 ; sub_1F003FE0+1CTr ...
.text3:20000386 aTag_did DCB "TAG_DID",0 ; DATA XREF: sub_1F006664+60f0
.text3:20000386 DCB 0 ; .text2:off_1F0067B8f0 ...
.text3:2000038E aTag_key DCB "TAG_KEY",0 ; DATA XREF: sub_1F008E74+72f0
.text3:2000038E DCB 0 ; .text2:off_1F008F94f0 ...
.text3:20000396 DCD 0
.text3:2000039A byte_2000039A DCB 0 ; DATA XREF: sub_1F009070+B2Tr
.text3:2000039B byte_2000039B DCB 0 ; DATA XREF: sub_1F009070+B4Tr
.text3:2000039C byte_2000039C DCB 0 ; DATA XREF: sub_1F009070+B6Tr
.text3:2000039D byte_2000039D DCB 0 ; DATA XREF: sub_1F009070+B8Tr
.text3:2000039E DCW 0
.text3:200003A0 DCB 0
.text3:200003A1 DCB 0
.text3:200003A2 aTag_model DCB "TAG_MODEL",0
.text3:200003A3 DCD 0
.text3:200003B0 DCB 0

```

Figure 19: Placeholders for device credentials in firmware of Lumi Smart Home Gateway

my firmware. Shortly after that I received the credentials successfully.

The extracted credentials can be used to decrypt and modify the traffic between the device and the Mijia cloud using Dustcloud. This is especially helpful if the vendor Lumi implements HTTPS in the future or adds checks of the MD5 hash of the firmware.

In fact, Lumi implemented the MD5 hash verification in version "1.4.1_156", after I presented some preliminary findings of this thesis in the DEF CON 26 IoT Village in Summer 2018¹⁷.

As a sidenote: While analyzing the firmware, I found a unknown communication to a hardcoded IP address (166.70.53.160) located in Salt Lake City, USA. This was interesting due the fact that the gateway was only sold on the Chinese market. After reporting this to Xiaomi, Lumi responded that this was a remaining from some earlier experiments with other customers. However, this remains were existing for a duration of 1.5 years in the firmware.

5.2.5 Yeelight LED Strip

The Yeelight LED strip was released in Winter 2016 and was only marketed in mainland Chinese. Later in 2018 Yeelight released an official US version of the LED strip. The company Yeelight is one of the biggest members of the Xiaomi IoT ecosystem and also produces the Chinese "Philips" branded light products.

Similar to Lumi, Yeelight offers a local API for their light products [62]. In addition to the Xiaomi Home app support, Yeelight offers their own app for Android and iOS, which offers additional features.

¹⁷ http://dontvacuum.me/talks/DEFCON26-IoT-Village/DEFCON26-IoT-Village_How_to_Modify_Cortex_M_Firmware-Xiaomi.html

similarity	confid	chan	EA primary	name primary	EA secondary	name secondary	cor	algorithm
1.00	0.99	----	1F045B5C	sub_1F045B5C_1641	1F0449EC	zap_chunk_boundary		hash matching
1.00	0.99	----	1F068498	sub_1F068498_2769	1F067328	xz_uncompress_stream		hash matching
1.00	0.99	----	1F068460	sub_1F068460_2768	1F0672F0	xz_uncompress_init		hash matching
1.00	0.99	----	1F0684DC	sub_1F0684DC_2770	1F06736C	xz_uncompress_end		hash matching
1.00	0.99	----	1F068678	sub_1F068678_2779	1F067508	xz_dec_run		hash matching
1.00	0.99	----	1F06937A	sub_1F06937A_2785	1F06820A	xz_dec_lzma2_run		hash matching
1.00	0.99	----	1F06983E	sub_1F06983E_2787	1F0686CE	xz_dec_lzma2_reset		hash matching
1.00	0.99	----	1F0698A0	sub_1F0698A0_2788	1F068730	xz_dec_lzma2_end		hash matching
1.00	0.99	----	1F041D00	sub_1F041D00_1508	1F040B90	xiaomi_cli_system_conf		edges flowgraph ME
1.00	0.99	----	1F0455EC	sub_1F0455EC_1631	1F04447C	xiaomi_check_wifi_fwimage		hash matching
1.00	0.99	----	1F045408	sub_1F045408_1628	1F044298	xiaomi_check_user_fwimage		hash matching
1.00	0.99	----	1F0445C0	sub_1F0445C0_1606	1F043450	xiaomi_check_app_fwimage_forwa...		prime signature mat
1.00	0.99	----	1F0443DC	sub_1F0443DC_1605	1F04326C	xiaomi_check_app_fwimage		hash matching
1.00	0.99	----	1F0457C0	sub_1F0457C0_1633	1F044650	xiaomi_check_app_fsimage		hash matching
1.00	0.99	----	1F059A14	sub_1F059A14_2249	1F0588A4	xTimerGenericCommand		hash matching
1.00	0.99	----	1F0597E4	sub_1F0597E4_2241	1F058674	xTaskGetSchedulerState		hash matching
1.00	0.99	----	1F058B36	sub_1F058B36_2205	1F0579C6	xQueueGenericSend		hash matching
1.00	0.98	----	1F068E9C	sub_1F068E9C_2831	1F06AD2C	write_u32		MD index matching
1.00	0.99	----	1F066674	sub_1F066674_2675	1F065504	write_rx_buf		hash matching
1.00	0.99	----	1F04D924	sub_1F04D924_1873	1F04C7B4	wrapper_wlan_handle_rx_packet		hash matching
1.00	0.99	----	1F04D924	sub_1F04D924_1873	1F04C7B4	wrapper_wlan_handle_rx_packet		hash matching

Figure 20: Matched functions between SDK libraries and unknown functions in firmware of Lumi Smart Home Gateway

For the analysis I imported 3 LED strips from China, as by the time I analyzed the hardware they were not available officially in Europe and US.

A first network scan revealed the mi.I0 service on port 54321(udp). An analysis of the traffic revealed, that similar to the Lumi Smart Home Gateway, the firmware updates were not encrypted and instead transmitted in plain-text over HTTP. The router blocked the download of the firmware to prevent the update. The device remained on the older version.

A closer look at the received firmware "upd_yeelink.light.strip1.elf" (version 140.40, MD5 hash: 301890243fa6aae4a67825988679785a) revealed that this devices is also based on a Marvell MCU. The firmware was very similar to the firmware of the Lumi Smart Home Gateway, even the placeholders for the device credentials were at the same place.

I reused the Nexmon environment and created a slightly different patch for this device so that I could remap the interesting functions as they did not had the same addresses as the Lumi Smart Home gateway. I could use the tool Bindiff again and compared it against the firmware I created for the Lumi device. After successfully building a custom firmware image, I repeated the same steps as for the Lumi Smart Home gateway and got the same result. The DNS redirection worked and even though my custom firmware image had a different MD5 hash it was still successfully installed. This meant that the firmware verification was also not implemented in this device. After installing my firmware, I extracted the device credentials and could now connect the device to Dustcloud.

Due to a mistake in rewriting another patch, the device became unresponsive. Therefore I needed to open the device. Compared with the other devices, this device was harder to open as part of the case was glued and additionally plastic clips were used. In the end I was able to open the device, however this left some scratches and bent plastic.

Figure 21 shows the disassembled device. By probing the same pins as for the Lumi device, I got access on the SWD and could overwrite the firmware with the original one. While the device has 2 copies of the OS there is no procedure to do a factory reset to a non-broken OS. While looking for SWD, I found also the UART pins. It turned out that this device is using the Marvell 88MW300 MCU instead of the 88MW302. However, the only difference is the additional USB support of the 88MW302, otherwise these chips are compatible with regard to the software. Also, the LED Strip has only 4 MByte SPI flash in comparison to the 16 MByte of the Lumi Smart Home gateway.

The latest version I have from this device is version "1.4.2_0044", which was released in Spring 2018. Therefore this version does not have a fixed firmware verification.



Figure 21: Front side of the PCB in disassembled Yeelight LED Strip

5.2.6 Yeelight Smart Light bulbs

There are many different variations of Yeelight Smart Light bulbs. The first ones were released in Winter 2016 together with some other Yeelight products. Like the other products, their primary target market was mainland China.

For this analysis I used the Yeelight Mono Light Bulb. My first assumption was, that Yeelight is using the same platform for all their lighting products. Therefore I directly tried to trigger the firmware update and intercepted the firmware binary. I received the file "upd_yeelink.light.mono1.bin" (version 1.4.2_0055, MD5 hash: ffa341cf1e00bbc1ebca4470f64dc5cb). The firmware binary was again a Marvell firmware. I again used Bindiff to map the functions. This time I could match the firmware with the one of the Yeelight LED Strip.

After creating a custom firmware with Nexmon, I could successfully install it on the device and extract the device credentials.

To get an impression of the inside of the Light bulb, I tried to disassemble it and to get access to the PCB. While part of the device came apart, for the rest I needed to use a saw and a rotating tool in order to open it. The result is shown in Figure 22. The light is, in that state, not repairable or functional anymore. That means that accessing the debug pins is difficult, if not even impossible, without total destruction of the device.

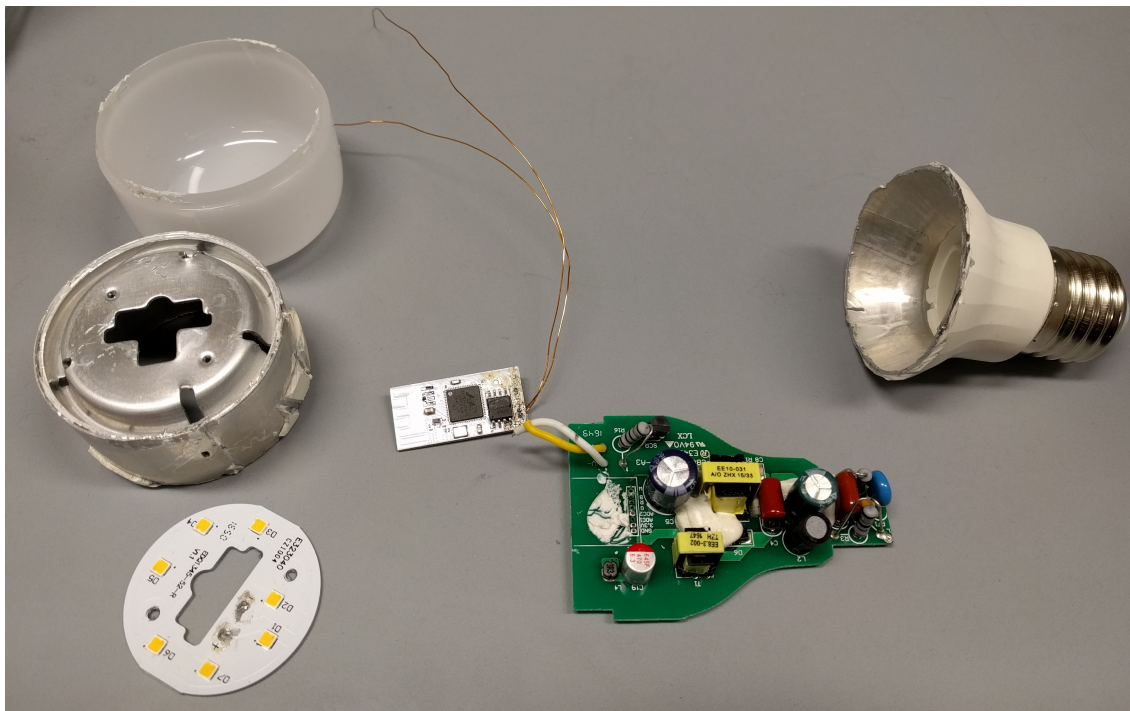


Figure 22: Disassembled Yeelight Smart Lightbulb

5.2.7 *Lumi Aqara Gateway Camera*

Towards the end of 2017 Lumi started their new brand "Aqara". As part of the new product series the Aqara Gateway camera was released. This device combines the function of a surveillance camera with the functionality of the Lumi Smart Home gateway. However, in contrast to the original Smart Home gateway, the Aqara Gateway camera does not support the local API functionality.

At first, the device was scanned for open ports. Surprisingly a telnet server was listening on port 23(tcp). However, a login was not possible, as the username and password were unknown. Additionally, as expected, the miIO service was on port 54321(udp). The intercepted network traffic showed that the firmware is again transmitted over plaintext Hypertext Transfer Protocol (HTTP). However this time the device seems to verify the firmware, therefore the DNS redirection method was not successful.

An analysis of the intercepted firmware update with binwalk showed a squashfs filesystem. However it contained only the application files, not a copy of the OS. In or-

der to obtain telnet access, it was necessary to disassemble the device and to unsolder the flash. Figure 23 shows the camera disassembled and the position of the UART pins. It was found that the camera uses a Hi3518-variant SOC and a 16 MByte SPI flash.

The information about the disassembly and the datasheets can be found on GitHub in my `dustcloud-documentation` repository¹⁸.

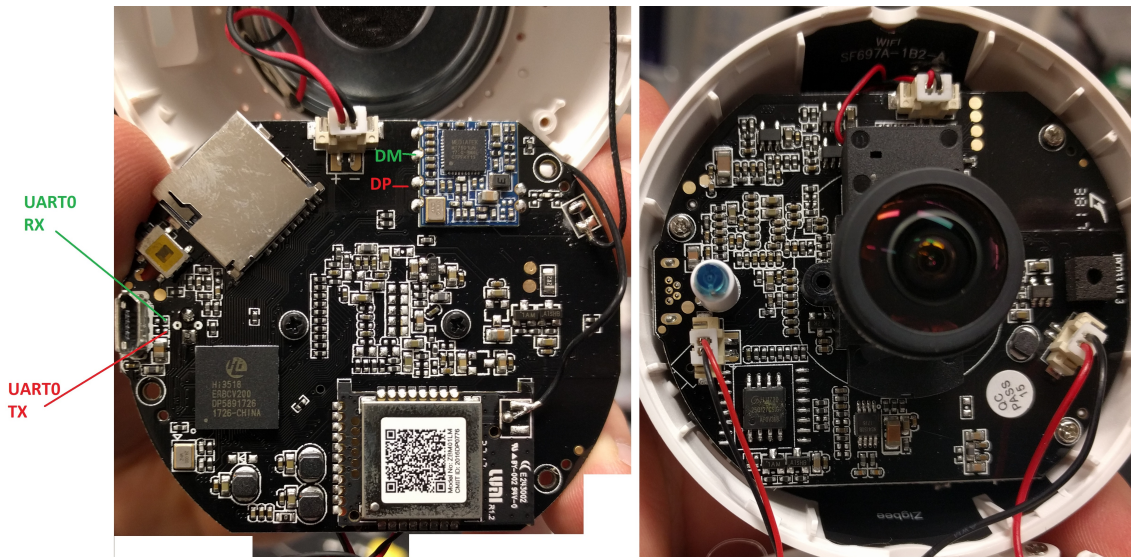


Figure 23: Disassembled Aqara Gateway Camera with Back-(left) and Frontside PCB (right)

After the SPI flash was unsoldered, it was imaged using the FlashcatUSB reader. The partitions on the flash can be found in Table 10. On the OS partition the shadow file was found. The developer of the camera used the SDK delivered with the Hi3518 SOC. The root password was encrypted using DEScrypt, which is highly insecure and shortens the password down to 8 characters. Using hashcat in combination with a NVIDIA Geforce 1080 graphics card, it was possible to brute-force the password. The resulting password was "lumi-201", however it is assumed that the original password was longer. Using this password the access via telnet was possible and the system could be modified.

After further analysis of the flash image, I found improperly deleted device credentials of the Lumi developers on the Userdata partition. This partition is using JFFS2 as the filesystem. JFFS2 is a flash optimized filesystem which does not delete files immediately, instead the "Garbage collector" removes them if necessary. The idea here is to avoid unnecessary writes on the flash [34]. However, this leads to the problem that deleted information is still present when, for example, the device is reset to factory settings.

The very interesting aspect of this device is, that the actual OS cannot be updated, as only one copy of it exists and its filesystem is also read-only. Instead, only the App partition can be changed via a firmware update. After I reported the vendor the flaw, Lumi released a firmware update in Fall 2018 where telnet server is being shutdown

¹⁸ <https://github.com/dgiese/dustcloud-documentation/tree/master/lumi.camera.aq1>

Partition	Usage	Filesystem	Size
mtdblock0	U-Boot bootloader	binary	256 KByte
mtdblock1	cmdline	binary	128 KByte
mtdblock2	Kernel	binary	1792 KByte
mtdblock3	OS	squashfs	6 MByte
mtdblock4	App0	squashfs	3 MByte
mtdblock5	App1	squashfs	3 MByte
mtdblock6	Userdata	JFFS2	1920 KByte

Table 10: Partitions on SPI flash of Aqara Gateway camera

after a few seconds by the software. Additionally the `/etc/shadow` file is replaced with a different version. However, here the vendor repeated the same mistake again and used DEScrypt for the password hashing. Therefore, I could also brute-force the new password.

5.3 ANALYSIS RESULTS

The hardware specifications I found in the previous reverse engineering can be found in Table 11. This table contains more devices that are mentioned in the previous sections due to the fact that many devices have a very similar behaviour and I wanted to give a broader overview for reasons of comparison. An assessment of security related features can be found in Table 12. Here the state of the security features is defined as of the moment of the initial analysis. Changes are mentioned in the corresponding footnotes.

Regarding the tables, it is evident that especially the Cortex-M based devices lack HTTPS or have HTTPS improperly implemented. Also, the Cortex-M based devices are often even lacking the verification of the firmware. It is assumed that this step was missing in a particular version of the SDK and was added as soon as some of my research was presented. All of the analyzed devices have at least one debug interface.

5.4 POSSIBLE ATTACKS

In this section I discuss possible attacks and threats on Xiaomi IoT devices using the information I got from the previous analysis.

In this chapter many security flaws of the device have been found. All, with exception of the open telnet service on the Aqara Camera, are not trivially abusable. One reason is, that the devices have been designed primarily for the usage in the Mijia Cloud ecosystem in mind. Therefore usual attack vectors, like web-interfaces, are missing. As these devices do not offer any direct interface, it does not make sense for the user to make them directly reachable from the Internet.

However, it does not mean that the devices are not attackable. In the unprovisioned state, everyone can connect to the devices as they are providing an open WLAN access-

Device name	SOC	Cores	OS	RAM	Flash
Aqara Gateway (Homekit)	NXP i.MX6ULL	ARM-Cortex A7	Yocto	256M	256M
Aqara Smart Home Gateway	NXP i.MX6ULL	ARM-Cortex A7	Yocto	256M	256M
Aqara Smart IP Camera	Hi3518EV200	ARM-Cortex A9	Linux	64M	16M
Lumi Smart Home Gateway	Marvell 88MW302	ARM Cortex M4F	RTOS	512K	16M
Philips Ceiling Lamp	Mediatek MT7697N	ARM Cortex M4F	Yeelight	352K	4M
Roborock S50	Allwinner R16	4x ARM Cortex A	Ubuntu 14.04	512M	4G
Roborock S6/T61	Allwinner R16	4x ARM Cortex A	Ubuntu 14.04	512M	4G
Xiaomi Mi Vacuum Robot	Allwinner R16	4x ARM Cortex A	Ubuntu 14.04	512M	4G
Xiaomi Mi WiFi Speaker	Amlogic-meson3	ARM Cortex A9	OpenWRT	128M	8G
Xiaomi WiFi Plug	Marvell 88MW300	ARM Cortex M4F	RTOS	512K	4M
Yeelink Bedside lamp	Mediatek MT7697N	ARM Cortex M4F	RTOS	352K	4M
Yeelink Ceiling Lamp	Mediatek MT7697N	ARM Cortex M4F	RTOS	352K	4M
Yeelink Light Color	Marvell 88MW300	ARM Cortex M4F	RTOS	512K	4M
Yeelink Light Mono1	Marvell 88MW300	ARM Cortex M4F	RTOS	512K	4M
Yeelink Light Strip	Marvell 88MW300	ARM Cortex M4F	RTOS	512K	4M
Yeelink Smart White Bulb	Marvell 88MW300	ARM Cortex M4F	RTOS	512K	4M
Yeelink Smart RGB Bulb	Marvell 88MW300	ARM Cortex M4F	RTOS	512K	4M

Table 11: Hardware details of the analyzed devices

point for configuration purposes. A malicious party could connect to the device in that state and flash a malicious firmware update. Nonetheless, as soon as a device is correctly provisioned, there are no trivial usable vulnerabilities, as the devices do not accept any command without the correct encryption by the token. This token is usually only known to the smart-phone of the owner.

Also my original planned attack using a malicious app, similar to Sivaraman et al. [51], does not make sense in this setting. But, in one scenario it is possible to attack the IoT devices in the local network: In case of a malicious application with root permissions on the owner's phone, it is possible to extract the tokens from the "Xiaomi Home" app and then attack the local devices.

The higher risk of attacks comes directly from the cloud. If there are unknown vulnerabilities in the Mijia cloud backend, then a huge amount of devices would be directly affected. This is especially critical for the Mijia cloud, as here the cloud can push firmware updates remotely on the devices.

Device name	Debug Interfaces			Firmware			Network		Physical		Data	
	UART	JTAG/SWD	Telnet/SSH	Encrypted	Signed	Verified	HTTPS	Certificate checked	Tamper resistant	Tamper evident	User data not on device	secure unprovisioning
Aqara Gateway (Homekit)	✓		✗	✗	✗	✓	✓	✓	✗	✓	✗	✗
Aqara Smart Home Gateway	✓		✗	✗	✗	✓	✓	✓	✗	✓	✗	✗
Aqara Smart IP Camera	✓ ¹	✗	✓ ²	✗	✗	✓	✗	✗	✗	✗	✗	✗
Lumi Smart Home Gateway	✓	✓	✗	✗	✗	✗ ³	✗	✗	✗	✓	✓	✓
Philips Ceiling Lamp	✓	✓	✗	✗	✗	✗	✓	✗	✗	✓	✓	✓
Roborock S50	✓		✓ ⁴	✓	✗	✓	✓	✓	✗	✓	✗	✗ ⁵
Roborock S6/T61	✓		✓ ⁴	✓	✓	✓	✓	✓	✗	✓	✗	✓ ⁶
Xiaomi Mi Vacuum Robot	✓		✓ ⁴	✓	✗	✓	✓	✓	✗	✓	✗	✗ ⁵
Xiaomi Mi WiFi Speaker	✓		✗	✗	✗	✓	✓	✓	✗	✗	✗	✗
Xiaomi WiFi Plug	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
Yeelink Bedside lamp	✓	✓	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓
Yeelink Ceiling Lamp	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
Yeelink Light Color	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
Yeelink Light Mono1	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
Yeelink Light Strip	✓	✓	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
Yeelink Smart White Bulb	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
Yeelink Smart RGB Bulb	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓

Table 12: Security details of the analyzed devices

¹ 0 Ohm resistors are missing

² disabled in later versions

³ verification added in later versions

⁴ blocked by iptables

⁵ formatting of the data partition introduced in later versions

⁶ data partition is formatted, however data partly restorable

Part III

DISCUSSION AND CONCLUSIONS

This chapter contains a discussion of the presented results as well as an overview of the future work. In addition, I finish this work with conclusions.

DISCUSSION

In this chapter I take a critical look at the results of my work. Section 6.1 discusses approaches used to reverse engineer the ecosystems. The results are discussed in Section 6.2. In Section 6.3 I discuss the results in relation to custom firmware. I conclude with a discussion of possible future work in Section 6.4.

6.1 THE USED APPROACHES FOR REVERSE ENGINEERING

The primary targets of my research were vacuum cleaning robots. The reason for this was based on the fact that these devices have much computational power for a vacuum cleaner and run a Linux Operating System (OS), which is, in comparison to a bare-metal OS, easier to understand and root access is more useful.

I collected the firmware updates for various devices, until I had a collection which spans over the time-frame of 2 years or even more, per device. This enabled me to track changes of new versions. This was also helpful in order to understand when which security measures and patches had been introduced.

Traditionally, the analysis of binary firmware requires a lot of experience and special tools like IDA Pro [28]. However, with the availability of automated tools and the publication of Ghidra [17], the entry level for such analysis is lowered. Still, in this thesis I mostly used the manual approach due to the lack of experience and overview in the field. However, with the gained knowledge from this research, I would do the analysis more efficiently next time.

The methods I used for my analysis are mostly low-cost and are already used in different contexts. Many of the laborious approaches can be avoided, if one has the experience and expertise for reliably desoldering and resoldering Ball Grid Array (BGA) chips. It has shown, that the firmware extraction from the flash chip was the most successful method to retrieve the firmware and potential other usual information, such as keys or factory tools. Also, my financial restriction led me to avoid potentially destroying the devices, which increased time consumption for the research. For a researcher without these restrictions the analysis of an Internet of Things (IoT) ecosystem is simpler and less time consuming. My assumption is, that for a malicious attacker the same would apply.

Still, my approaches have been successful and the number of permanently damaged devices was limited to Light bulbs and one Aqara camera. However I believe I will use more automated tools, like Firmware Analysis and Comparison Tool (FACT), in the future in order to create a more efficient workflow.

6.2 SECURITY OF THE XIAOMI ECOSYSTEM

After the analysis of the IoT devices, it became apparent that security engineering is not the top priority of the device vendors. All the Real-Time Operating System (RTOS) based devices had the same vulnerabilities. Some of the vendors released patched firmware, however only after the vulnerabilities had been reported. Also only the reported vulnerabilities have been closed.

The usage of the same Software Development Kit (SDK) leads to the problem that the same vulnerability can be present in a whole set of devices. In the case of the Mijia ecosystem, three vendors used mostly the same firmware, without custom modifications in regard to security. This shows a huge belief in the ability of the ecosystem provider to develop a secure SDK, which leads to a false security. Also one explanation could be that the developers are under pressure to develop a product as fast as possible. As long as no security incident is happening, the vendors do not see the necessity to improve security.

One extreme example of fixing flaws is the earlier shown case of the Aqara Camera. The camera runs an open telnet server and the password is hardcoded. The password itself is stored using the insecure hashing algorithm DEScript. After the report to the vendor, the developers did exactly the same mistake again and stored the new password using DEScript again. While at least this time the telnet server was disabled, it showed that the actual issue was not addressed. In case of the Aqara Camera, it is assumed that the SDK of the chip manufacturer was reused. The exact same SDK was part of the problem while the outbreak of the Mirai botnet [7].

In regard to the Linux based devices, many aspects are similar to the RTOS devices. One interesting observation is, that the logic of the Mijia SDK is slightly different between the platforms. For example the Linux based SDK verifies the MD5 hash of the firmware file, but does not differentiate between commands issued via the local Application Programming Interface (API) (using the token) and via the cloud API. This enabled the installation of custom firmware on all Linux based devices. In contrast to that, all the RTOS based firmware did not accept specific commands via the local API. Instead they did not verify the firmware MD5 hash. This could imply that the SDKs were developed by different teams.

Another interesting observation was that the cloud protocol had design flaws. In Section 3.3, I mentioned that the Microcontroller Unit (MCU) updates do not have an integrity checking method in form of an MD5 hash. In combination with the, earlier mentioned, missing verification of the System-on-a-Chip (SOC) firmware integrity, this observation is worrisome as firmwares can be modified using Man-in-the-middle (MITM) attacks. This is possible for devices which download firmware via Hypertext Transfer Protocol (HTTP) or do not verify Hypertext Transfer Protocol Secure (HTTPS) certificates correctly.

For a large IoT company like Xiaomi these kind of flaws should not be acceptable. However, if such a company has such flaws, one can make an assumption about how

many flaws less known IoT vendors have, as they have potentially less resources or interest to fix it.

In regard of the security, companies started to react. For example, Roborock started to close down flaws in their devices after me reporting issues and they got knowledge about it. All new models are not vulnerable anymore to the old attacks. In contrast to that, there are vendors who were aware of a vulnerability and did not properly close it over a long time.

One advantage of the Xiaomi ecosystem is that they are not meant for the direct user interaction. This means that they do not have the typical interfaces, like user-accessible webservers or APIs, which are most of the time the weakspot of IoT devices.

6.3 POSSIBILITY OF CUSTOM FIRMWARE

There are many open-source smart home software with huge communities, like home-assistant¹ or FHEM². Many user want to gain control over their smart home devices and control them independently from the vendor. This wish was also the reason what had driven me to do the research in this thesis.

While security flaws in IoT devices are usually considered bad, they are good out of the perspective of users who want to modify their devices. If the vendor does not offer an open interface to control a device, then modification of the devices might be the only alternative. An example here is the Wink Hub, which got rooted by many users and where a community was developed [46]. Additionally, modification of devices enables the user to add additional features to the device, for example, if the vendor stopped the support with functional updates, like happened with Roborock and the older versions of the vacuum cleaning robots.

Not all of the analyzed devices are equally suitable to be rooted. While it is possible to develop custom firmware for the Cortex-M based devices using Nexmon, it is too complicated for the average user. Tools like Dustcloud, which was mainly developed for this thesis, are suitable for power users or developers. Especially in the case of Dustcloud, it is important to know that it is not meant as a reliable solution for end-users. However, one specific set of the analyzed devices is also suitable for the average user: vacuum cleaning robots. The released tools in the Dustcloud suite enable users to create custom firmware and get Secure SHell (SSH) on their own vacuum cleaning robot. As a result of the research done in this thesis custom software for the control of the vacuum cleaning robots was developed. One example is valetudo which works autonomously on the vacuum cleaner and offers all the features of the "Xiaomi Home" app [27]. To simplify the creation of custom firmware for the vacuum cleaning robots, I created a public service³ which creates images automatically.

¹ <https://community.home-assistant.io/>

² <https://forum.fhem.de/>

³ <http://dustbuilder.dontvacuum.me>

6.4 FUTURE WORK

With every time I published rooting methods, some vendors reacted with closing down this ways. In particular, the Roborock vacuum robots became more and more locked down. Recently the company released the Xiaomi Mi Robot M1S, which does have a camera as an additional sensor. This device makes use or ARM TrustZone, encrypt its flash contents and uses a memory protection feature of the SOC in order to scramble the contents of the memory. The reverse engineering of this device would be difficult using only the methods presented in this work. However, as it is part of the Xiaomi ecosystem and likely Roborock reused part of the software, many learned information in this study could be potentially useful for the analysis of these new devices. Interestingly, the company told me, that the changes are a direct reaction on my research.

Also, due to the grow of the ecosystem, a shifted focus on other members of the ecosystem might be helpful. Especially the company "Viomi" releases very similar products like Roborock in the recent time. It could be interesting to compare two devices with the same features in the same ecosystem.

CONCLUSIONS

Large Internet of Things (IoT) ecosystems, which are operated by huge companies, are not automatically secure. However, due to the structure of the ecosystem, most of the flaws are not usable for mass-attacks in any users. In this research, I studied the security of the Xiaomi IoT ecosystem. In particular, my focus was on the security of the devices in the ecosystem. First I introduced the technical background with IoT devices as part of embedded systems. Then I discuss the requirements for an secure IoT system. I presented my approach to analyze the ecosystem. I introduced the structure and the corresponding Application Programming Interface (API) of Xiaomi's Mijia ecosystem. I built tools to parse firmware files. Also, I got a better understanding about the cloud protocol by developing a tool which proxied the cloud communication. Next I discussed different methods for the reverse engineering of the device, in particular based on network, on hardware, and on software. I defined criteria of interest for the following analysis and used the previously discussed methods to successfully attack the devices. For all devices there was a found to get root access or some other form of privileged access, like a modified bare-metal Operating System (OS). I published my findings, documentation, and tools on my GitHub repositories. Now other researchers can use the information for their research, possibly to analyze any kind of IoT or embedded device. In addition, users of the Xiaomi IoT ecosystem can modify their devices and hence decide for themselves if they want to opt-out from Xiaomi acting as their cloud provider or not.

The methods and tools developed in this thesis can be used to reverse engineer other Xiaomi ecosystem devices. This especially applies to the firmware parsing tools and the modified version of the Nexmon framework. However the use-case is not restricted to the Xiaomi ecosystem. Instead the methods can be used to analyze any kind of IoT or embedded device.

BIBLIOGRAPHY

- [1] *88MW300/302WLAN Microcontroller Datasheet*. v2.0. Marvell. May 2016.
- [2] Lady Ada. "Programming Microcontrollers using OpenOCD on a Raspberry Pi." In: *Adafruit Learning System* (Mar. 2016). URL: <https://learn.adafruit.com/programming-microcontrollers-using-openocd-on-raspberry-pi>.
- [3] *After crossing RMB 100B revenue milestone, a new journey for Xiaomi begins in 2018*. Feb. 2018. URL: <http://blog.mi.com/en/2018/02/07/after-crossing-rmb-100b-revenue-milestone-a-new-journey-for-xiaomi-begins-in-2018/>.
- [4] *Alexa, are you listening?* [Online; accessed 4. Jul. 2019]. Aug. 2017. URL: <https://labs.mwrinfosecurity.com/blog/alexa-are-you-listening>.
- [5] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. "Sok: Security evaluation of home-based iot deployments." In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. <https://doi.org/10.1109/SP>. 2019.
- [6] Anonymous. *Internet Census 2012: Port scanning/o using insecure embedded devices*. [Online archived; accessed 02. Jun. 2019]. 2013. URL: <https://web.archive.org/web/20151013010243/http://internetcensus2012.bitbucket.org/paper.html>.
- [7] Manos Antonakakis et al. "Understanding the Mirai Botnet." In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 1093–1110. ISBN: 978-1-931971-40-9. URL: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>.
- [8] Harald Bauer, Mark Patel, and Jan Veira. Tech. rep.
- [9] *Buyer Beware: Used Nest Cams Can Let People Spy on You*. [Online; accessed 3. Jul. 2019]. June 2019. URL: <https://thewirecutter.com/blog/used-nest-cams-can-let-people-spy-on-you>.
- [10] Consumer Technology Association (CTA). *CTA Consumer Technology Ownership and Market Potential Report*. 2017.
- [11] *Choosing the Right NAND*. [Online; accessed 4. Jul. 2019]. URL: <https://www.micron.com/products/nand-flash/choosing-the-right-nand>.
- [12] Adam Cooper. *IEI REPORT Electric Company Smart Meter Deployments: Foundation for A Smart Grid*. Tech. rep. The Edison Foundation, Oct. 2016, p. 2.
- [13] *CriptTor Ransomware Targeting NAS | D-Link*. [Online; accessed 02. Jun. 2019]. Mar. 2019. URL: <https://www.dlink.com/en/security-bulletin/cr1ptt0r-ransomware-targeting-nas>.
- [14] *EliasKotlyar/Xiaomi-Dafang-Hacks*. [Online; accessed 4. Jul. 2019]. June 2019. URL: <https://github.com/EliasKotlyar/Xiaomi-Dafang-Hacks/blob/master/README.md>.
- [15] *English manual · Issue #4 · louisZL/lumi-gateway-local-api*. [Online; accessed 4. Jul. 2019]. June 2017. URL: <https://github.com/louisZL/lumi-gateway-local-api>.

- [16] *FEL - linux-sunxi.org*. [Online; accessed 29. May 2019]. Mar. 2018. URL: <https://linux-sunxi.org/FEL>.
- [17] *Ghidra*. [Online; accessed 9. Jul. 2019]. Apr. 2019. URL: <https://www.nsa.gov/resources/everyone/ghidra>.
- [18] M. Ghiglieri and E. Tews. "A privacy protection system for HbbTV in Smart TVs." In: *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*. Jan. 2014, pp. 357–362. DOI: 10.1109/CCNC.2014.6866595.
- [19] A. Gilchrist. *IoT Security Issues*. De Gruyter, Incorporated, 2017. ISBN: 9781501505775. URL: <https://books.google.com/books?id=xipDDgAAQBAJ>.
- [20] Vivek Gite. *10 boot time parameters you should know about the Linux kernel*. [Online; accessed 29. May 2019]. Mar. 2006. URL: <https://www.cyberciti.biz/tips/10-boot-time-parameters-you-should-know-about-the-linux-kernel.html>.
- [21] *Global Smartphone Shipments Down 6.0% in Q3 2018 as the Leading Vendor and the Largest Market Face Challenges, According to IDC*. 2018. URL: <https://www.idc.com/getdoc.jsp?containerId=prUS44425818>.
- [22] J Grand. "JTAGulator: assisted discovery of on-chip debug interfaces." In: *DEFCON 21, Las Vegas*. 2013.
- [23] Mordechai Guri, Yosef Solewicz, Andrey Daidakulov, and Yuval Elovici. "SPEAKE(a)R: Turn Speakers to Microphones for Fun and Profit." In: *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. Vancouver, BC: USENIX Association, 2017. URL: <https://www.usenix.org/conference/woot17/workshop-program/presentation/guri>.
- [24] Dick Hardt. *The OAuth 2.0 Authorization Framework*. [Online; accessed 4. Jul. 2019]. Oct. 2012. URL: <https://tools.ietf.org/html/rfc6749>.
- [25] M. Hermann, T. Pentek, and B. Otto. "Design Principles for Industrie 4.0 Scenarios." In: *2016 49th Hawaii International Conference on System Sciences (HICSS)*. Jan. 2016, pp. 3928–3937. DOI: 10.1109/HICSS.2016.488.
- [26] *HomeHack: How Hackers Could Have Taken Control of LG's IoT Home Appliances - Check Point Software*. [Online; accessed 20. Jun. 2019]. Oct. 2017. URL: <https://blog.checkpoint.com/2017/10/26/homehack-how-hackers-could-have-taken-control-of-lgs-iot-home-appliances>.
- [27] *Hypfer/Valetudo*. [Online; accessed 4. Jul. 2019]. June 2019. URL: <https://github.com/Hypfer/Valetudo>.
- [28] *IDA Debugger: Overview*. [Online; accessed 9. Jul. 2019]. Feb. 2018. URL: <https://www.hex-rays.com/products/ida/debugger/index.shtml>.
- [29] "IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications." In: *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)* (Dec. 2016), pp. 1–3534. DOI: 10.1109/IEEESTD.2016.7786995.
- [30] "IEEE Standard for Low-Rate Wireless Networks." In: *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)* (Apr. 2016), pp. 1–709. DOI: 10.1109/IEEESTD.2016.7460875.

- [31] *ITU-T Y.2060 Overview of the Internet of things*. July 2012. URL: <http://handle.itu.int/11.1002/1000/11559>.
- [32] *IoT Security: Physical Layer Security for IoT PCBs*. [Online; accessed 29. May 2019]. May 2017. URL: <https://resources.altium.com/pcb-design-blog/iot-security-internet-of-things-pcbs>.
- [33] *IoT Testing Guides - OWASP*. [Online; accessed 20. Jun. 2019]. May 2016. URL: https://www.owasp.org/index.php/IoT_Testing_Guides.
- [34] *JFFS2*. [Online; accessed 8. Jul. 2019]. Oct. 2001. URL: <http://sourceware.org/jffs2/jffs2-html/node3.html>.
- [35] M. Kitano, A. Nishimura, S. Kawai, and K. Nishi. "Analysis of package cracking during reflow soldering process." In: *26th Annual Proceedings Reliability Physics Symposium 1988*. Apr. 1988, pp. 90–95. DOI: 10.1109/RELPHY.1988.23432.
- [36] A. Marzano, D. Alexander, O. Fonseca, E. Fazzion, C. Hoepers, K. Steding-Jessen, M. H. P. C. Chaves, Í. Cunha, D. Guedes, and W. Meira. "The Evolution of Bashlite and Mirai IoT Botnets." In: *2018 IEEE Symposium on Computers and Communications (ISCC)*. June 2018, pp. 00813–00818. DOI: 10.1109/ISCC.2018.8538636.
- [37] *Mi Robot vacuum • Robot Reviews*. [Online; accessed 2. Jul. 2019]. Feb. 2017. URL: <http://www.robotreviews.com/chat/viewtopic.php?f=6&t=19479&start=220>.
- [38] *Nintendo NES Classic Edition - linux-sunxi.org*. [Online; accessed 02. Jun. 2019]. Apr. 2019. URL: https://linux-sunxi.org/Nintendo_NES_Classic_Edition.
- [39] *OWASP Embedded Application Security - OWASP*. [Online; accessed 17. Jun. 2019]. Nov. 2018. URL: https://www.owasp.org/index.php/OWASP_Embedded_Application_Security#tab=Embedded_Top_10_Best_Practices.
- [40] *R16 Datasheet*. v1.4. Allwinner. June 2016.
- [41] *R16 User Manual*. v1.2. Allwinner. Jan. 2016.
- [42] *Radio Versions | Bluetooth Technology Website*. [Online; accessed 4. Jul. 2019]. June 2019. URL: <https://www.bluetooth.com/bluetooth-technology/radio-versions>.
- [43] *RaspberryPi - flashrom*. [Online; accessed 4. Jul. 2019]. Jan. 2018. URL: <https://www.flashrom.org/RaspberryPi>.
- [44] Fred Raynal. *Flash Dumping - Part I*. [Online; accessed 4. Jul. 2019]. Sept. 2017. URL: <https://blog.quarkslab.com/flash-dumping-part-i.html>.
- [45] *Rockusb - Rockchip open source Document*. [Online; accessed 2. Jul. 2019]. July 2018. URL: http://opensource.rock-chips.com/wiki_Rockusb.
- [46] *Root Your Hub - Wink@Home Wiki*. [Online; accessed 3. Jul. 2019]. Oct. 2015. URL: https://wiki.winkathome.net/Root_Your_Hub.
- [47] *SDIO at Raspberry Pi GPIO Pinout*. [Online; accessed 9. Jul. 2019]. URL: <https://pinout.xyz/pinout/sdio#>.
- [48] *SYS.4.4 Allgemeines IoT-Gerät*. Tech. rep. Bundesamt für Sicherheit in der Informationstechnik, 2018. URL: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKompendium/bausteine/SYS/SYS_4_4_Allgemeines_IoT-Gerät.html.

- [49] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. *Nexmon: The C-based Firmware Patching Framework*. 2017. URL: <https://nexmon.org>.
- [50] O. Schwartz, Y. Mathov, M. Bohadana, Y. Elovici, and Y. Oren. "Reverse Engineering IoT Devices: Effective Techniques and Methods." In: *IEEE Internet of Things Journal* 5.6 (Dec. 2018), pp. 4965–4976. ISSN: 2327-4662. DOI: 10.1109/JIOT.2018.2875240.
- [51] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. "Smart-Phones Attacking Smart-Homes." In: *WISEC*. 2016.
- [52] Nick Statt. "Nest is permanently disabling the Revolv smart home hub." In: *Verge* (Apr. 2016). URL: <https://www.theverge.com/2016/4/4/11362928/google-nest-revolv-shutdown-smart-home-products>.
- [53] Synology. *Important Information about Ransomware SynoLocker Threat*. [Online; accessed 02. Jun. 2019]. Aug. 2014. URL: <https://www.synology.com/en-global/security/advisory/SynoLocker>.
- [54] *The Linux Watchdog driver API*. [Online; accessed 15. Jun. 2019]. May 2007. URL: <https://www.kernel.org/doc/Documentation/watchdog/watchdog-api.txt>.
- [55] *TheCrypto/yi-hack-v4*. [Online; accessed 4. Jul. 2019]. May 2019. URL: <https://github.com/TheCrypto/yi-hack-v4/blob/master/README.md>.
- [56] *[U-Boot] change root password*. [Online; accessed 29. May 2019]. May 2010. URL: <https://lists.denx.de/pipermail/u-boot/2010-May/071166.html>.
- [57] "Vacuums in the Cloud: Analyzing Security in a Hardened IoT Ecosystem." In: *13th USENIX Workshop on Offensive Technologies (WOOT 19)*. Santa Clara, CA: USENIX Association, 2019. URL: <https://www.usenix.org/conference/woot19/presentation/ullrich>.
- [58] Jeffrey Voas. *NIST Special Publication 800-183: Network of 'Things'*. Tech. rep. National Institute of Standards and Technology(NIST), 2016. DOI: 10.6028/NIST.SP.800-183.
- [59] *Xiaomi Gateway (Aqara) - Domoticz*. [Online; accessed 4. Jul. 2019]. Mar. 2019. URL: [https://www.domoticz.com/wiki/Xiaomi_Gateway_\(Aqara\)](https://www.domoticz.com/wiki/Xiaomi_Gateway_(Aqara)).
- [60] *Xiaomi Unveils Their Very Own ESP32 Development Board, Module & SDK*. [Online; accessed 30. Jun. 2019]. Nov. 2017. URL: <https://www.cnx-software.com/2017/11/28/xiaomi-esp32-development-board>.
- [61] *Xiaomi and IKEA partner to bring smart connected homes to more users*. Nov. 2018. URL: <http://blog.mi.com/en/2018/11/28/news-xiaomi-and-ikea-partner-to-bring-smart-connected-homes-to-more-users/>.
- [62] Yeelight. *Developer Mode Yeelight*. [Online; accessed 8. Jul. 2019]. URL: https://www.yeelight.com/en_US/developer.
- [63] *Zigbee and IEEE 802.15.4 wireless microcontroller with 512 kB Flash, 32 kB RAM | NXP*. [Online; accessed 4. Jul. 2019]. URL: <https://www.nxp.com/products/wireless/proprietary-ieee-802.15.4-based/zigbee-and-ieee802.15.4-wireless-microcontroller-with-512-kb-flash-32-kb-ram:JN5169>.

- [64] *aes.h File Reference - API Documentation - mbed TLS*. [Online; accessed 1. Jun. 2019]. Mar. 2019. URL: https://tls.mbed.org/api/aes_8h.html.
- [65] *fkie-cad/FACT_core*. [Online; accessed 9. Jul. 2019]. June 2019. URL: https://github.com/fkie-cad/FACT_core.
- [66] *rampageX/firmware-mod-kit*. [Online; accessed 9. Jul. 2019]. Mar. 2019. URL: <https://github.com/rampageX/firmware-mod-kit>.
- [67] *rytilahti/python-miio*. [Online; accessed 4. Jul. 2019]. July 2019. URL: <https://github.com/rytilahti/python-miio>.
- [68] *zynamics.com - BinDiff*. [Online; accessed 4. Jul. 2019]. Apr. 2016. URL: <https://www.zynamics.com/bindiff.html>.

THESIS STATEMENT

pursuant to § 23 paragraph 7 of APB TU Darmstadt

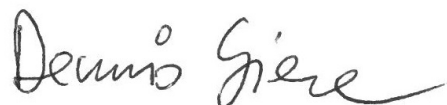
I herewith formally declare that I, Dennis Giese, have written the submitted Master Thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form. I am aware, that in case of an attempt at deception based on plagiarism (§ 38 Abs. 2 APB), the thesis would be graded with 5.0 and counted as one failed examination attempt. The thesis may only be repeated once. In the submitted thesis the written copies and the electronic version for archiving are identical in content.

ERKLÄRUNG ZUR ABSCHLUSSARBEIT

gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Dennis Giese, die vorliegende Master Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden. Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Boston, 10. Juli 2019



Dennis Giese