



School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

Requirements to theses submitted in
the Faculty of Engineering

by

Andrew Jin-Meng Wong

Thesis submitted as a requirement for the degree of

Bachelor of Engineering in Computer Engineering

Submitted: August 2022

Supervisor: Prof. Richard Buckland

Student ID: z5206677

Topic ID: “Smart” Vacuum Cleaners

An Audit Into The Security and Integrity of IoT Systems

Abstract

Research Statement: How have manufacturers of IoT / smart home devices addressed the increasing concerns of digital privacy and product security

With the ever-growing adoption of convenient and user-friendly Internet of Things devices, more and more objects around us have made their way onto the internet, requiring connectivity to the web for one reason or another. Despite the unknown nature of communication and limited transparency of data, such privacy concerns are often overlooked in exchange for convenience. This paper audits the Roborock S6 robotic vacuum cleaner to assess its internal operations and network activity behaviour, as to investigate any potential vulnerabilities that may render the device unsafe or insecure.

A combination of dynamic and static binary analysis methods were performed to assess the security of the device, and network activity was inspected to verify the contents of network traffic. Investigation results revealed discrepancies in both the security of the product, and the privacy of user data pertaining to authentication credentials. Notably, a novel command injection exploit was proposed, and suggestions were made to better improve the device's security and privacy.

Acknowledgements

| sonder (noun)

the realisation that each random passerby is living a life as vivid and complex as your own

I would like to thank my family and friends for supporting me through my ventures and adventures at the University of New South Wales. This support was not solely just for this thesis adventure that I had decided to embark on, nor was it just from the wisdom or knowledge I learned or gained, but from every moment shared with others in the downtimes and pauses of my, and our sonderous lives.

Thank you to my peers and students from my tutorial classes who showed curiosity in my research, gave me a platform to share my passion, and a provided me a means to stay motivated. A special thank you to the friends who basically sat around with me doing nothing - yet made those moments so very memorable.

Thank you to my supervisor Lachlan and Prof. Richard Buckland for the various resources, tips, tricks and meaningful yet entertaining conversations about computers, and other things along those lines...

And of course, thank you to my family for their daily nagging to tell me to rest and go to sleep.

Contents

CHAPTER 1 INTRODUCTION	1
CHAPTER 2 A BACKGROUND ON IOT	3
THE CALL TO ACTION	3
ABOUT THE PRODUCT	3
CHAPTER 3 CURRENT STATE OF THE ART	4
BROAD SECURITY STUDY OF TUYA-BASED DEVICES	4
BROAD SECURITY STUDY OF XIAOMI-BASED DEVICES	5
SECURITY STUDY OF SMARTPHONE APPLICATIONS	5
ANALYSIS OF SIMILARITIES IN IOT FIRMWARES	6
SIDE-CHANNEL APPLICATION OF LIDAR SENSOR MEASUREMENTS	6
SHELL ACCESS VIA SIDELOADED MEDIA	7
SHELL ACCESS VIA BGA PIN SHORTING	7
HARDWARE BASED EXTRACTION OF FLASH MEMORY	8
COLD-BOOT ATTACK TO DUMP MEMORY STATE	8
CHAPTER 4 THREAT MODELLING	9
CHAPTER 5 WORK PERFORMED	10
SCOPE AND SUMMARY OF WORK	10
PRELIMINARY DEVICE ACCESS	11
DYNAMIC FIRMWARE ANALYSIS	12
STATIC FIRMWARE ANALYSIS	17
UPGRADE ANALYSIS (VERSION 02.29.02)	23
NETWORK ACTIVITY ANALYSIS	28
DEVICE ENTRY AND PERSISTENCE ANALYSIS	37
CHAPTER 6 DISCUSSION	42
COMMENTARY	42
RESPONSE OF OTHER MANUFACTURERS	45
CONCLUSION	46
FUTURE WORK	47
BIBLIOGRAPHY	48

List of Figures

FIGURE 1 - UART PIN LOCATIONS	11
FIGURE 2 - PASSWORD DECRYPTION OF THE VINDA FILE	11
FIGURE 3 - PROCESS LIST (v01.15.58)	13
FIGURE 4 - NETSTAT (v01.15.58).....	14
FIGURE 5 - IPTABLES (v01.15.58)	14
FIGURE 6 - IP6TABLES (v01.15.58)	14
FIGURE 7 - IFCONFIG (v01.15.58).....	14
FIGURE 8 - /ETC/PASSWD (v01.15.58)	15
FIGURE 9 - /ETC/SHADOW (v01.15.58)	15
FIGURE 10 - GENERATED SHA512 PASSWORD HASH	15
FIGURE 11 - UNDERSIDE OF THE CHARGING DOCK.....	15
FIGURE 12 - 2-WIRE BATTERY SHUTDOWN LOG	15
FIGURE 13 - SERIAL LOG DURING FACTORY RESET.....	16
FIGURE 14 - FIRMWARE DUMP COMMANDS	17
FIGURE 15 – FILE STRUCTURE OF MMCBLK0p11	18
FIGURE 16 - SCREENSHOT OF THE COMMENTREE TOOL.....	18
FIGURE 17 – BINDIFF COMPARISON OF NTPDATE (v01.15.58)	19
FIGURE 18 – APT-GET HISTORY.LOG FILE	19
FIGURE 19 – EXPOSED MICRO USB CONNECTOR ON THE ROBOROCK S6.....	20
FIGURE 20 – CUSTOM ADBD AUTH CHALLENGE FLOW	20
FIGURE 21 – ADBD LOCK RESET FLOW.....	21
FIGURE 22 – ADBD COMMAND INJECTION VULNERABILITY POC	21
FIGURE 23 - DISASSEMBLY OF THE ENCRYPTION ROUTINE IN RRLOGD (v01.15.58).....	22
FIGURE 24 – IPTABLES ALLOW RULE IN RRLOGD (v01.15.58).....	22
FIGURE 25 – OBSOLETE DECRYPTION ROUTINE IN SYSUPDATE.....	23
FIGURE 26 – IP6TABLES RESULTS (v02.29.02).....	24
FIGURE 27 – VERIFY__SHADOW FUNCTION ROUTINE	25
FIGURE 28 – SYSV CONFIGURATION SCRIPT (v02.29.02).....	26
FIGURE 29 – WATCHDOGE PROCESS ENFORCING IPTABLES.....	27

FIGURE 30 - ISOLATED NETWORK CONNECTION DIAGRAM	28
FIGURE 31 - CRYPTO FUNCTION HOOK SOURCE CODE	30
FIGURE 32 – NETWORK COMMUNICATION DIAGRAM	31
FIGURE 33 – PRIVACY POLICY EXCERPT	32
FIGURE 34 – EXPOSURE OF WIRELESS CREDENTIALS IN RRIOT_TUYA.LOG (FW: V02.29.02).....	32
FIGURE 35 - PLAIN-TEXT CREDENTIAL TRANSMISSION DURING PAIRING	33
FIGURE 36 – MQTT SERVER DATA HEATMAP	34
FIGURE 37 – MQTT SERVER HISTORICAL OVERVIEW	34
FIGURE 38 – CONTROL SERVER DATA HEATMAP	34
FIGURE 39 – FDS SERVER DATA HEATMAP	35
FIGURE 40 – FDS SERVER FLOW GRAPH	35
FIGURE 41 – GEOMAP OF DEVICE ACTIVITY TO XIAOMI FDS SERVERS	35
FIGURE 42 – MUD USAGE DIAGRAM	36
FIGURE 43 – MUD PROFILE SNIPPET (V02.29.02)	36
FIGURE 44 – RRWATCHDOGE.CONF WITH SSH ACCESS PATCH.....	38
FIGURE 45 – MIIO OTA PAYLOAD	39
FIGURE 46 – SILENT FAIL OF THE MIIO.OTA PAYLOAD	39
FIGURE 47 – ZEROTIER CONTROL PANEL.....	40
FIGURE 48 – SCREENSHOT OF CVEs ASSOCIATED WITH XIAOMI	45

List of Tables

TABLE 1 - THREAT MODEL MATRIX	9
TABLE 2 - ROOT PASSWORD EXTRACTION PROCEDURE	11
TABLE 3 - v01.15.58 SYSTEM FINGERPRINT	12
TABLE 4 - IMPORTANT PROCESSES (v01.15.58)	13
TABLE 5 - UNTOUCHED DIRECTORIES DURING VOLATILE ACTIONS	16
TABLE 6 – FIRMWARE PARTITION MAPPING	17
TABLE 7 – FIRMWARE UPGRADE CHANGELOG	23
TABLE 8 – WLANMGR ROUTINE 0x136e8 (v02.29.02)	27
TABLE 9 – COLLECTED NETWORK DATA (v02.29.02)	27
TABLE 10 - NETWORK EQUIPMENT LIST	28
TABLE 11 – COMPARISON OF DATA TRANSPARENCY METHODS	29
TABLE 12 – OVERVIEW OF NETWORK ENDPOINTS	30
TABLE 13 – OVERVIEW OF DEVICE ENTRY AND ACCESS METHODS	37

Table of Abbreviations

	Expansion	Meaning
C2	Command and Control	Remote action management
DUT	Device Under Test	Relating to the specific device being tested
eMMC	Embedded Multimedia Card	Onboard storage
HSTS	HTTP Strict Transport Security	Network security policy
IoT	Internet of Things	Classification of network-connected devices
IP	Internet Protocol	Network communication protocol
IPC	Inter-Process Communication	Data exchange between programs in a system
MAC	Media Access Control	Unique network device identifier
MITM	Man In The Middle	Intercepted communication
MQTT	Message Queue Telemetry Transport	Network communication protocol
NIC	Network Interface Card	Hardware to connect a device to a network
PII	Personal Identifiable Information	Data that could identify an individual
PoC	Proof of Concept	A demonstration to prove a concept / theory
SDK	Software Development Kit	Building blocks for software interoperability
SoC	System on Chip	An entire system integrated into a single chip
SSID	Service Set Identifier	Wi-Fi network name
SSL	Secure Sockets Layer	Network security protocol
TLS	Transport Layer Security	Network security protocol
UART	Universal Asynchronous Receiver/Transmitter	Hardware communication protocol
UGC	User Generated Content	Data created by the user
WEP	Wired Equivalent Privacy	Network security algorithm
WPA	Wi-Fi Protected Access	Network security algorithm

Chapter 1 | Introduction

Consumer grade Internet of Things (IoT) devices have become widely adopted with continuously growing demand. With demand growing by 12% each year (Research & Markets 2021), this AU\$130bn industry has cordially invited thousands of households to invest in smart devices such as light bulbs, fans, televisions and fridges. Given the abundance and affordability of these products, IoT devices have become an integral part of many homes, where 4 in 5 consumers would be more inclined to choose a property over another given the presence such technologies (Brown 2015).

Although convenient, these devices come with hidden costs and risks. Behind the seemingly ‘simple’, ‘smart’ and ‘secure’ product features that attract consumers lie a hidden complex network of services and devices, where functionality is often obscured and private. Without the transparency of what data is being sent, and of where that data is being sent to, consumers inevitably pay for convenience with not only their money but with their privacy and security (Miralem, Nejra et al. 2019)

Whilst manufacturers and vendors claim to be secure and/or confidential in how they treat UGC and PII, it is evident from various incidents that we cannot completely trust such claims. From leaked Facebook user data (Abrams 2021), to rumours of corporations monetising user data without consent (Jones 2017), there lies an equal need for consumers to understand the terms of service to which they agree to, but additionally for companies to be audited against those very same terms of service.

The infrastructural security and product security of IoT devices must also be scrutinised, given the rapid product lifecycle of IoT developments (Giese 2021). As security is often not a sellable feature in contrast to new products and most fallibly – convenience, proper and wholistic security precautions are often overlooked by companies who are more concerned with profits. Consequently, the prevalence of malicious actors in the cyberworld is alarming, where the overall lack of security awareness between consumers invites target devices to be easily accessed with default passwords or through unpatched vulnerabilities¹.

Given the black-box nature of IoT network communications where there is little transparency about the functionality and usage of IoT devices beyond their advertised description, there is a need to shed light unto the privacy and security of these devices. **This thesis aims to detail how manufacturers of IoT / smart home devices have addressed the increasing concerns of digital privacy and product security.** Specifically, we audit the Roborock S6 robotic vacuum cleaner to assess its internal operations and the nature of data that is transmitted, as to verify manufacturer claims, and investigate potential vulnerabilities that render the device insecure.

We first study further motivations behind auditing the privacy and security of IoT systems, then review existing research and methods that comprise the current state of the art of IoT security and privacy research. Finally, we detail the work performed in this thesis and discuss the contributions and conclusions, providing suggestions to further the security and privacy of IoT devices.

¹ <https://www.shodan.io/search?query=webcam>

Summary of Major Contributions

This thesis critically analyses the security and privacy of the Roborock S6’s firmware and network communications. A list of major contributions and actionable findings are as follows, by decreasing order of severity and importance.

Data Persistence

File persistence tests were conducted to test the retention of data during the following scenarios: firmware upgrade, factory reset, device disassociation (unpairing). It was observed that no data was cleared when a device was unpaired, raising concerns regarding data privacy. Methods were proposed to persist data during firmware upgrades and factory resets.

Privacy Policy

The privacy policy of the vacuum cleaner data was assessed and revealed that a statement regarding the locality of wireless credentials was non-compliant, as the credentials were found within uploaded log data.

Pairing Security

The pairing process of the device was observed and revealed that wireless credentials were transmitted in plain text over an unsafe medium (wireless network with open / no security), despite IoT ecosystem vendor guidelines to require a secured means of communication.

Product Security

Security assessments were performed on the programs in the Roborock S6 firmware to evaluate the security of the device. Whilst most programs were secure, a novel command injection vulnerability was discovered in the Android Debugging Bridge implementation. A proof of concept was created and disclosed to the vendor.

Upgrade analysis revealed that the vendor has made non-trivial effort to fortify their software against vulnerabilities and limit unauthorised access to the device.

Network Behaviour

The nature and content of network traffic generated by and received from the Roborock S6 was analysed to create a connection map of device communications, and a heatmap of network activity. A Manufacturer Usage Description profile (RFC 8520) was created for the device to better describe its expected traffic behaviour and provide a means to mitigate foreign traffic. The IPv6 capability of the device was also tested, drawing conclusions that possible IPv6 related issues were benign.

Chapter 2 | A Background on IoT

The Call to Action

The consumer market has experienced a large influx of IoT devices, largely attributed to the presence of IoT manufacturers who offer white-label partnerships with resellers to provide “custom” products. Through these partnerships, vendors buy into the IoT manufacturer’s ecosystem - namely the product itself, the companion smartphone application, and the cloud infrastructure supporting network communications - all without requiring vendors to possess any knowledge or understanding of how to design, develop nor manufacture the IoT products that they sell.

This raises concerns regarding the privacy and ownership of user data that is transmitted, as vendors themselves are often not in control of what information is transmitted nor of how that information is used – for example if the microphone data of a surveillance camera was used to determine advertised products related to the conversation. The lack of control over information is a potentially serious concern, as vulnerabilities within an IoT infrastructure would endanger customers from other vendors under the same infrastructure. Furthermore, the lifetime of a vendor business is not guaranteed. With the constant opening and sunseting of IoT vendors, the closure of the business from which an IoT product was purchased from might eventually render the device inoperable.

In the event that an IoT infrastructure suffers downtime or service instability, all white-labelled products too will also be affected. Great trust must be placed in the infrastructure’s availability and reliability. In conjunction with aforementioned privacy and security concerns, many concerned users have turned to internet-less and self-hosted automation systems such as *HomeAssistant* and *OpenHAB*. As evident in later reviewed works, concerns for privacy and security have been a driving force for developers and hackers to research and develop software to replace the internet-dependent stock software, effectively decoupling devices from vendor services.

About the Product

Beijing Roborock Technology Co., Ltd. (Roborock) is a Chinese company founded in Beijing that develops robotic cleaning appliances for households. In 2014, partnering with Xiaomi Corporation shortly after the opening of their business, the company released a line of both affordable and premium smart robotic vacuum cleaners, with their first iteration the “*Mi Home Robotic Vacuum Cleaner*” being released in Sep 2016. They have since released twelve other robotic vacuum cleaner models, each model offering new and improved features. Despite having released 13 different products, only one security vulnerability has been publicly disclosed², raising concern about the company’s security.

In June 2019, Roborock released their flagship Roborock S6 vacuum cleaner. Featuring an *Allwinner R16* SoC (ARM architecture), it is powered by either the Tuya Smart or Xiaomi Cloud infrastructure, both market leaders in the consumer IoT industry. Despite being released three years ago, the Roborock S6 vacuum cleaner is still widely popular and actively maintained by Roborock. This device will be the DUT (device under test) in this thesis.

² <https://global.roborock.com/pages/disclosure-security-vulnerability-on-tuya-iot-cloud>

Chapter 3 | Current State of the Art

Broad security study of Tuya-based devices

The security research group Vtrust (2018) analysed a line of white-labelled IoT product revisions based on the IoT manufacturer Tuya to identify common security vulnerabilities. Despite vendor claims of ‘military-grade security’, basic packet logging of network activity concluded that “*the analysis of the ‘smart’ devices using this basic platform is generally frightening*”, with “*serious [...] shortcomings*”. It was revealed that various PII, encryption keys and the device’s serial number (used to specify a device during remote commands) were insecurely transmitted over the network, allowing a user on the same wireless network to eavesdrop on the communication. Furthermore, during the initial setup and pairing of the IoT device, wireless credentials were also insecurely transmitted in plain text, allowing wireless network credentials to be observed.

Vtrust commented on the dangers of vendors selling white-label products, where anyone could become a so-called ‘IoT company’ regardless of whether they had “*in-depth technical knowledge of IoT or IT security*”. As a result of the hands-free approach to security and privacy for both direct and indirect customers of the IoT platform, concerns were raised regarding the ease of distributing maliciously modified devices, where firmware could be tampered with during any stage within the supply chain.

It is worthwhile to recognise that most custom firmware releases or “hardware hacks” originate from the desire to decouple hardware from online and official cloud services. These ventures effectually disconnect internet-reliant devices from the cloud, and limit their connectivity to a local server where communications are transparent and minimal.

As a result of many Tuya-powered devices sharing the widely popular *Espressif ESP8266 SoC*³, Vtrust was able to exploit discovered vulnerabilities on multiple products to perform over-the-air upgrades of custom firmware (e.g. *ESPHome*, *Tasmota*). An automated flashing tool (*tuya-convert*) was released, allowing consumers to easily integrate these devices with local home automation software such as *HomeAssistant*. As a result of Vtrust’s findings, the overall security posture of modern Tuya-powered devices has since improved⁴, with implementations of local flash memory encryption and firmware signing measures during over-the-air firmware upgrades.

Vtrust’s technical findings offer insights into methods of network-level security assessment highlighting how easily an individual could start their own IoT company, and the possibility of reselling devices with modified firmware with malicious intent. In this thesis we perform similar network security assessments through means of analysing packet captures to determine if data is weakly or insecurely transmitted.

³ <https://www.espressif.com/en/products/socs/esp8266>

⁴ <https://www.heise.de/newsticker/meldung/Smart-Home-Hack-Tuya-veroeffentlicht-Sicherheitsupdate-4292028.html>

Broad security study of Xiaomi-based devices

Giese (2019) performed a security assessment over a broad range of Xiaomi’s IoT products to examine the overall security of the Xiaomi ecosystem. Through different software injection and hardware fault injection techniques, Giese obtained shell access into various Xiaomi-powered devices. It was concluded that due to the enormous size of Xiaomi’s ecosystem, it was difficult to enforce global security policies between the different vendor-provided plugins that continued to support deprecated functions and APIs that were still being used by legacy devices. Out from this research, a *cloud emulator*⁵ was built, allowing for complete offline functionality and control over a large range of Xioami devices without requiring internet connectivity. This research also paved the way for other third-party, privacy-focused, vacuum cleaner remote applications to developed, such as *Valeduto*.

He concluded that Xiaomi indeed treats their security concerns seriously, given their quick responses to reported security incidents and vulnerability reports. In this thesis, we too will assess the security and privacy postures of IoT devices on the business-level.

It should be noted that Giese briefly assessed the security of the Roborock S6 vacuum cleaner in his study. Whilst Giese did perform a security analysis of the device under test, this thesis was performed as an independent study. With the exception of Giese’s work to obtain initial shell access, all other similar methods performed, findings and observations are coincidental. This thesis furthers previous studies as it additionally audits the state of privacy of the device.

Security study of smartphone applications

Jmaxxz (2016) investigated the security claims of a smart doorlock which had boasted in its bank-grade security, and superiority over conventional lock-and-key systems. These claimed were however invalidated, as flaws within the smartphone application were discovered which allowed control over the lock settings, amusingly only being protected by client-side checks. Consequently, modified request payloads containing elevated authorisation claims would be naively accepted by the server, allowing lock settings to be modified by a guest or other user. Furthermore, various debugging menus were present in the production version of the smartphone application, allowing certificate pinning protections to be subverted. In addition, the privacy of the user was also questioned, as it was observed that door lock events and other identifiable information were being transmitted to a logging endpoint.

The vulnerabilities in the smart doorlock’s own product security highlight the importance to verify any claims that manufacturers may advertise. This study serves as an excellent example of a failed access control system, where elementary methods of request tampering and hardcoded keys allow for arbitrary privileged control of a device. Subversion of HTTP Strict Transport Security (HSTS) and certificate pinning policies through system-wide tools⁶, per-application patching⁷ or accessible debug menus furthermore underlines that certificate pinning should not be relied upon to verify identity nor authority.

⁵ <https://github.com/dgiese/dustcloud>

⁶ <https://github.com/nabla-c0d3/ssl-kill-switch2>

⁷ <https://github.com/shroudedcode/apk-mitm>

Analysis of similarities in IoT firmwares

Costin, Zaddach et al. (2014) performed a broad static firmware analysis over a large number of firmware images to identify common patterns and similarities between product vendors. During the analysis of the 693 images, 38 new vulnerabilities were discovered, some of which were present in the majority of images. Many hardcoded keys and credentials were also discovered that could render the IoT device or its infrastructural service vulnerable. To facilitate the similarity analysis of firmware images, where per-byte analysis techniques are nonsensical, tools like *binwalk*, *ssdeep*, and *sdhash* were employed - which helped to facilitate file exploration relative to their file type and architecture. To compare versions of the same binary across different firmwares, a tool called *BinDiff* was used, which would compare the similarities and differences in assembly code and call graphs.

A large proportion of images shared similarities in code execution graphs, indicating that many vendors had simply reused and repurposed sample code (often available as part of the SDK from a SoC vendor or IoT framework). Whilst sample code itself is not often vulnerable, given the commonality of other vulnerabilities, concern is raised as to the vendor’s technical capability and understanding of IoT systems and of security. The tools and methods to perform this firmware study are transferable to the scope of this thesis, where static analysis of executable programs can be used to identify vulnerabilities or potential malicious modifications to existing software.

Side-channel application of LIDAR sensor measurements

As more and more IoT devices become online and sensor data is transmitted around the world, there are growing concerns to thoroughly investigate the extents of what data can be retrieved from the sensors. Given that the outputs of Light Detection and Ranging (LIDAR) sensors are reflected intensity values and distance measurements, Wei, Wang et al. (2015) developed a method to translate the intensity readings from the LIDAR sensor back into audio signals, when the LIDAR sensor was directed towards a surface near an audio source. This allowed speech to be identified from micro-vibrations within objects, raising concern regarding the privacy and confidentiality of conversations held within a sound-proof room.

This research has since been continued and tested on robot vacuum cleaners which too incorporate LIDAR sensors intended for spatial mapping. In the application of a robotic vacuum cleaner, light intensity values are considered a side-channel concern as those readings are not required for the operation of a vacuum cleaner. As general off-the-shelf LIDAR sensor units (capable of reading such light intensity values) are used within vacuum cleaners, this technique could be also applied to detect speech and sound (Sriram, Xiang et al. 2020). Despite the limitations of sampling light intensity values on a vacuum cleaner (i.e. accounting for the continuous rotation of the LIDAR sensor and audible noise floor as a result of the vacuum engine), a high classification accuracy of 91% was still achieved when extracting sensitive data from speech such as digits of a credit card.

Whilst this thesis will not pursue the exploration of sensor data analysis, these two studies offer potential future research areas on privacy concerns surrounding robot vacuum cleaners, as newer revisions of smart devices become continually equipped with more accurate and feature-rich sensors.

Shell access via sideloaded media

Often as a necessary preliminary step to further research, modification and integration of proprietary technologies, many device rooting methods (i.e ways to gain elevated access to a device) have been publicly disclosed on the internet. Commonly, devices which are not expected to have internet connectivity may provide offline firmware upgrade functionality by executing a script or booting from some form of removable flash memory such as a microSD or SD card. Kotlyar (2017) demonstrated the ability for the inexpensive *Xiaomi Dafang Camera* to boot into a custom alternate *u-boot* bootloader that was flashed onto a microSD card. Upon detection of a firmware-like storage medium, the device executed the contents of the microSD card, and booted into shell instead of the original entry-point script, effectively rooting the device. Kotlyar was then able to dump the firmware, later producing a custom firmware release that did not rely on the vendor’s cloud infrastructure.

Through the subversion of interrupting the default boot sequence, resultant shell access allowed for the development and release of decoupled software. Whilst the exact rooting steps are unlikely to be directly transferable to other devices, the idea of obtaining elevated access via sideloading techniques is an important method to investigate. Throughout the course of the thesis, we attempted to gain shell access via sideloading methods, but were unsuccessful.

Shell access via BGA pin shorting

For devices that do not automatically boot into removable media, methods have been discovered to force certain SoC’s to enter a recovery or fallback mode. Allwinner-based SoCs implement a mode known as “FEL” that can be entered by pulling a certain pin LOW during boot⁸, which allows device manufacturers to perform initial image flashing and bootloader configuration. For developers and hardware hackers, FEL mode allows users to modify the boot environment to execute a shell, allowing for further post-exploitation methods and firmware dumping / analysis.

It is noted that FEL mode can also be entered if the SoC fails to successfully launch the bootloader. Giese (2019) identified this fact and exploited the physical pin layout of the *Allwinner R16* BGA package, where the data pins connecting the SoC to the (e)MMC chips (where the bootloader is stored) were on the physical perimeter of the SoC. By sliding a piece of aluminium foil between the circuit board and the solder plane of the SoC, the electrically conductive aluminium foil could momentarily short the data pins long enough to cause the bootloader read operation to corrupt and fail, hence booting into FEL mode and eventually gaining shell access. This method is favourable when compared to pulling the FEL pin low during boot - as access to the FEL pin would require the desoldering and removal of the SoC from a circuit board - which can be tedious and prone to mistake and irreversible damage.

Through this hardware fault injection technique of shorting data pins during boot, Giese was able to successfully gain access to a shell on Roborock’s first robot vacuum cleaner (*Mi Robot Vacuum Cleaner*). On a different vacuum cleaner (the Roborock S7), Giese noted that test pad *TPA17* on the circuit board was connected to the SoC’s FEL pin - allowing FEL mode to be entered by usual means without needing to perform a hardware fault injection.

⁸ Generally triggered by pulling the *FEL pin* (LRADC0) LOW during boot

Hardware based extraction of flash memory

In situations where no provisions exist to programmatically extract stored data from a system (i.e. shell access to perform disk imaging), hardware devices known as flash programmers can be used; designed to read from and write data onto flash chips. Flash programmers incur a high cost overhead, as they are rather expensive and only work with specific models and/or types of flash chips; rendering it infeasible to own a specific flash programmer for every type of flash chip. Jimenez (2016) points out that a Raspberry Pi could be used as an affordable budget solution when paired with open-source flash programming software like *flashrom*.

It is noted that the process of hardware flash chip dumping is not feasible in the scope of this thesis due to resource and cost constraints of not possessing a suitable flash programmer, as well as the risk associated with hardware-based methods being possibly destructive with irreversible damage. This method of flash memory extraction was not required as other methods were successfully performed to obtain the firmware data of the device under test.

Cold-boot attack to dump memory state

Regarding prior investigations of smart robot vacuum cleaners, Ullrich, Classen et al. (2019) performed a security analysis on the *Neato BotVac Connected* robot. Through the combination of a cold-boot attack - where a system is rebooted without the volatile memory (i.e. RAM) being cleared - and the booting of a custom bootloader image, the memory state of the system's prior execution was able to be dumped and analysed. This memory dump is of significant value as it would contain the binaries of loaded programs as well as their application state. The proceeding analysis revealed major vulnerabilities and concerns in the vacuum cleaner and more alarmingly, in Neato's cloud infrastructure.

Whilst logs and coredumps were encrypted when transmitted to cloud servers, encryption keys were discovered to be hardcoded which nullified any assurances of encryption. Authentication and authorisation tokens were all encrypted with the same weak RSA key - which left the entire cloud infrastructure vulnerable to impersonated identities and access. Seemingly random generated keys were also discovered to be vulnerable, due to the keyspace for entropy being so short that the key was able to be bruteforced within reasonable time. Furthermore, an unauthenticated endpoint on the robot vacuum cleaner's remote port was found to be vulnerable to a buffer overflow, allowing remote code execution on the robot by anyone connected to the same wireless network.

The analysis of a system's memory state is beneficial to the security assessment of a product's firmware as static analysis techniques are unable to account for dynamic data such as response payloads from client-server communications. This method of memory extraction was not required as other simpler methods were successfully performed to obtain the firmware data of the device under test.

Chapter 4 | Threat Modelling

To qualify the observations of proceeding results, it is worthwhile to form threat scenario models, as to identify the different perspectives and their associated risks/concerns that will be assessed.

Table 1 - Threat model matrix

		TS0	TS1	TS2	TS3
		-	Physical (proximal)	Remote (proximal)	Remote (distal)
Concern	Physical Access	✓	✓		
	Remote Access		✓	✓	✓
	Data Ownership	✓			
	Data Visibility	✓	✓	✓	✓

Table 1 above forms an overview of the four threat scenarios analysed in this thesis.

In *TS0*, we analyse the implications of data visibility and data ownership in a scenario devoid of any malicious threat. This scenario is akin to a product owner who is wary of other parties holding data pertaining to them and wishes to seek transparency in the type and storage of data retained. The scenario additionally extends to a product owner who wishes to maximise the functionality of a device that they purchase and own – such as through improvements or various modifications.

In *TS1*, we assess the threat implications from parties who are within physical proximity of the device. This includes parties as part of the supply chain, second-hand sellers, and individuals who have either momentary, or prolonged physical access to the device. Concerns are raised regarding parties being able to data from the device or regaining control of the device after losing physical access.

In *TS2*, we inspect the ability for a remote party to monitor device communications, or otherwise gain control over a device, without needing physical access to the device at any time. Specifically, the remote party is nearby / within proximity of the device (either within wireless range or connected to a shared computer network).

In *TS3*, we analyse possibility and implications for a remote party to access the device, either through means of a backdoor (possibly planted from *TS1* / *TS2*), or through the vendor’s system themselves. We also assess the ramifications of gaining remote access to an internet-connected sensor-enabled device, however it should be noted that the scope of this thesis excludes the propagation of data in the cloud once received by the vendor.

Chapter 5 | Work Performed

Scope and Summary of Work

We begin our work by first defining the scope and extent to which the privacy and security assessment will be performed.

In investigating privacy concerns, we monitor the nature of wireless network activity from a powered off factory-reset Roborock S6 vacuum cleaner when where we pair (initialise), operate, and let the device idle. We observe the device’s behaviour and interaction to other devices on the same wireless network (LAN), as well as its communications to external servers (WAN). This is performed as to better understand the nature of network communications, such as data frequency, duration, size, destination, and content.

In investigating security concerns, we analyse the behaviour and configuration of the system, and identify points of potential compromise or modification that may allow a third-party to gain control of the device, or otherwise render the device insecure. We additionally compare a baseline version of the device firmware to its most recent (April 2022) as to draw insights into how the manufacturer (Roborock) has responded to both the security of the device, and the privacy of the user.

Whilst work and discussions may reference topics from the following: smartphone application communications and interactivity, internal cloud functionality and cloud endpoint vulnerabilities, and the propagation of cloud data - they are beyond the scope of assessment and were performed out of interest, or as aides to other discussion.

Throughout the course of investigation, findings relating privacy and security were not mutually exclusive, and often involved a discussion of both areas. As such, this chapter will be subdivided by work categories, and only briefly overview implications. Detailed privacy and security discussions will follow in the *Discussions* chapter.

Preliminary Device Access

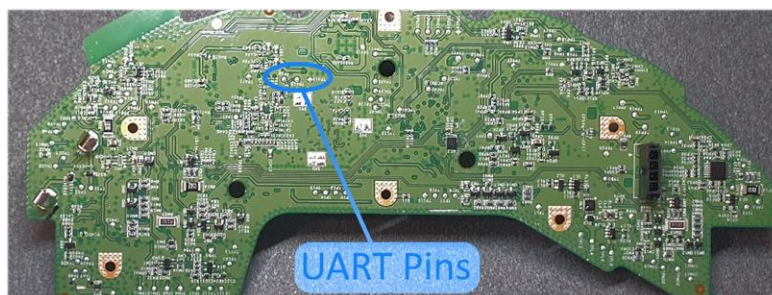


Figure 1 - UART pin locations

As discovered by Giese (2019), the Roborock S6 vacuum cleaner contains circuit board test pads that correspond to the *Allwinner R16* SoC’s configured serial pins, as seen above. In detail, *TPA8* is the device’s TX pin, *TPA15* is the device’s RX pin, and *TPA16* is ground. A USB to UART adapter can then be used to gain access to the serial interface

Once a serial connection was established (*baud rate* = 115200), functionality in the U-Boot bootloader firmware can be exploited to enter the bootloader’s shell mode, by means of sending multiple ‘s’ characters to interrupt the boot sequence⁹. Within the shell, Giese documented a series of instructions to extract the root password from a file called *vinda*, located inside the device’s eMMC flash. This file contained a 16-byte string, which when XOR’d with the byte *0x37*, results in the root password used to gain access to the device. It is noted that root shell access is obtainable without requiring the root password, however it is beneficial.

Table 2 - Root password extraction procedure

Step	Command	Description
1	<code>ext4load mmc 2:6 0 vind</code>	Load contents of <i>vinda</i> into memory position 0
2	<code>md 0 4</code>	Dump the first 4 words from memory position 0
3	-----	XOR values with <i>0x37</i>

```
sunxi#ext4load
ext4load - load binary file from a Ext4 filesystem

Usage:
ext4load <interface> <dev[:part]> [addr] [filename] [bytes]
- load binary file 'filename' from 'dev' on 'interface'
  to address 'addr' from ext4 filesystem
sunxi#ext4load mmc 2:6 0 vind
Loading file "vinda" from mmc device 2:6
16 bytes read
sunxi#md 0 4
00000000: 5b415243 51454346 54505042 525f5655  CRA[FCEQBPTUV_R
```

Figure 2 - Password decryption of the *vinda* file

⁹ https://github.com/allwinner-zh/bootloader/blob/master/u-boot-2011.09/board/sunxi/board_common.c#L843-L847

Dynamic Firmware Analysis

Device Fingerprinting

Upon gaining access to the shell, device fingerprinting was performed as to better understand the operating system, hardware feature set and software capability.

It is important to know that the device under test was manufactured in June 2020, one year after the official release of the Roborock S6 vacuum cleaner in June 2019. As a result, the recovery firmware (stored in *mncblk0p7*) is versioned 01.15.58 (25th March 2020). All firmware investigation processes and results collected in the proceeding sections were performed against version 01.15.58, until the upgrade analysis section on page 2323.

Table 3 below outlines the various commands and outputs used to identify the system information. Other necessary hardware information (such as storage and memory) is excluded from the table as they are officially listed on the Roborock product webpage¹⁰. Most notably, fingerprint results conclude that the system is running an ARM release of *Ubuntu 14.04.3 LTS*, with *libc* version 2.19 (released 2014). This finding aided the installation and execution of other software that was during the security and privacy assessment of the device under test.

Table 3 - v01.15.58 System Fingerprint

Command	Output
uname -a	Linux rockrobo 3.4.39 #1 SMP PREEMPT Wed Mar 25 20:47:59 CST 2020 armv7l armv7l armv7l GNU/Linux
ldd --version ldd	ldd (Ubuntu EGLIBC 2.19-0ubuntu6.6) 2.19
cat /etc/os-release	NAME="Ubuntu" VERSION=" 14.04.3 LTS , Trusty Tahr" ID=ubuntu ID_LIKE=debian PRETTY_NAME="Ubuntu 14.04.3 LTS" VERSION_ID="14.04" HOME_URL="http://www.ubuntu.com/" SUPPORT_URL="http://help.ubuntu.com/" BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/" ROBOROCK_VERSION= 3.5.4_1558
cat /etc/OS_VERSION	ro.product.device=MI1558_TANOS_MP_S2020032500REL_M3. 3.0_RELEASE_20200325-204847 ro.build.display.id=TANOS_MP_R16_RELEASE_ 20200325- 204847 ro.sys.cputype=R16.STM32.A3.G1 ro.build.version.release=1558 ro.build.date.utc=1585140527

¹⁰ <https://global.roborock.com/pages/roborock-s6>

Process Capability

An instance of *htop* - a process viewer utility¹¹ - was loaded on to the device to monitor the running processes as shown in Figure 3, and described in Table 4. Immediate observations revealed that all non-system processes were executed under root-level privileges, which raises device security concerns as a potential vulnerability in any of the executables may lead to system takeover.

It should be noted that it is not uncommon for embedded Linux systems to run processes under the root account during development as difficult IPC and communication port access issues (e.g. *udev* rules) can be bypassed whilst the product is being developed. If process privileges are not tightened for production or deployment releases however, vulnerabilities are formed regarding least-privilege security principles.

Given the nature of the device running an ARM version of Ubuntu, the execution of foreign binaries was tested successfully, confirming that there no software execution whitelist policies present in the system.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
995	root	0	-20	20704	3296	2720	S	0.0	0.6	0:00.16	RoboController
996	root	0	-20	20704	3296	2720	S	0.0	0.6	0:00.00	RoboController
1000	root	0	-20	25856	8340	3580	S	0.0	1.6	0:00.14	AppProxy
1012	root	0	-20	25856	8340	3580	S	0.0	1.6	0:00.17	AppProxy
1013	root	0	-20	25856	8340	3580	S	0.0	1.6	0:00.02	AppProxy
1014	root	0	-20	25856	8340	3580	S	0.0	1.6	0:00.00	AppProxy
1015	root	0	-20	25856	8340	3580	S	0.0	1.6	0:00.77	AppProxy
1016	root	0	-20	25856	8340	3580	S	0.0	1.6	0:00.02	AppProxy
1018	root	0	-20	25856	8340	3580	S	0.0	1.6	0:00.02	AppProxy
1026	root	0	-20	20704	3296	2720	S	0.0	0.6	0:00.00	RoboController
1027	root	0	-20	20704	3296	2720	S	0.0	0.6	0:00.00	RoboController
1028	root	0	-20	20704	3296	2720	S	0.0	0.6	0:00.00	RoboController
1030	root	0	-20	119M	35360	10192	S	0.0	6.9	0:00.02	rr_loader -d
1275	root	0	-20	9476	1584	1188	S	0.0	0.3	0:00.00	wlanmgr
1276	root	0	-20	9476	1584	1188	S	0.0	0.3	0:00.21	wlanmgr
1285	root	0	-20	2568	1404	1004	S	0.0	0.3	0:00.06	/bin/bash /opt/rockrobo/mio/mio_client_helper_nomqtt
1325	root	0	-20	1180	228	184	S	0.0	0.0	0:00.00	miio_recv_line
1349	root	0	-20	2724	1644	1080	S	0.0	0.3	0:00.14	/bin/bash /usr/bin/create_ap -c 11 -n wlan0 -g 192.168
1404	nobody	0	-20	2568	664	468	S	0.0	0.1	0:00.80	dnsmasq -C /tmp/create_ap.wlan0.conf.arN3AgFf/dnsmasq
1405	root	0	-20	2724	964	400	S	0.0	0.2	0:01.80	/bin/bash /usr/bin/create_ap -c 11 -n wlan0 -g 192.168
1406	root	0	-20	4364	1692	1384	S	0.0	0.3	0:00.04	hostapd /tmp/create_ap.wlan0.conf.arN3AgFf/hostapd.conf
1473	root	0	-20	13996	2592	1980	S	0.0	0.5	0:00.00	WatchDoge /opt/rockrobo/watchdog
1474	root	0	-20	13996	2592	1980	S	2.0	0.5	0:24.25	WatchDoge /opt/rockrobo/watchdog
1687	root	20	0	2480	1520	1192	S	0.0	0.3	0:00.21	-bash
3575	root	20	0	9202	2868	2196	S	0.0	0.6	0:00.34	sshd: root@notty
3600	root	20	0	1644	680	552	S	0.0	0.1	0:00.05	/usr/lib/openssh/sftp-server
9690	root	20	0	9292	2776	2092	S	0.0	0.5	0:02.06	sshd: root@pts/1

Figure 3 - Process list (v01.15.58)

Table 4 - Important processes (v01.15.58)

Program	Purpose
<i>AppProxy</i>	Central management
<i>RoboController</i>	Vacuum cleaner logic
<i>rr_loader</i>	Sensor and cleaning driver
<i>WatchDoge</i>	System health and process monitor
<i>rrlogd</i>	Device log manager
<i>rriot_tuya</i>	Tuya cloud communications bridge

¹¹ <https://github.com/htop-dev/htop>

Network Capability

A list of open ports and firewall rules were collected as shown in the figures below. Collected results revealed that ports were exposed on *tcp/6668* and *tcp/22* (SSH), with the SSH server listening to both IPv4 and IPv6 connections. As suggested in Figure 5, inbound IPv4 connections to the SSH server were dropped, however IPv6 connections were not (Figure 6). In effect, efforts to prevent SSH access may have been undermined due to the lack of IPv6 access control restrictions.

To verify this hypothesis, the vacuum cleaner was connected to a wireless network serving DHCPv6 leases from an *Orange Pi R1 Plus* device running *OpenWRT* (as the main network infrastructure did not support IPv6 – see *Test Infrastructure Setup*). Results from *ifconfig* refuted this theory, as the IPv6 address listed was prefixed with *fe80::*, which hints that the device did not request for a DHCPv6 lease – hence no IPv6 address was assigned to the device, rendering the device unreachable via IPv6.

```

root@rockrobo:~# netstat -nlt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:54322         0.0.0.0:*               LISTEN      991/mio_client
tcp        0      0 127.0.0.1:54323         0.0.0.0:*               LISTEN      991/mio_client
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1644/sshd
tcp        0      0 127.0.0.1:55551         0.0.0.0:*               LISTEN      998/rrriot_tuya
tcp        0      0 0.0.0.0:6668            0.0.0.0:*               LISTEN      998/rrriot_tuya
tcp6       0      0 :::22                   :::*                    LISTEN      1644/sshd

```

Figure 4 - *netstat* (v01.15.58)

```

root@rockrobo:~# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination
DROP udp -- anywhere udp dpt:6665
DROP tcp -- anywhere tcp dpt:6665
DROP tcp -- anywhere tcp dpt:ssh

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination

```

Figure 5 - *iptables* (v01.15.58)

```

root@rockrobo:~# ip6tables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination

```

Figure 6 - *ip6tables* (v01.15.58)

```

root@rockrobo:~# ifconfig
lo: Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:193 errors:0 dropped:0 overruns:0 frame:0
TX packets:193 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:16738 (16.7 KB) TX bytes:16738 (16.7 KB)

wlan0: Link encap:Ethernet HWaddr 64:90:c1:1d:24:c4
inet addr:192.168.2.206 Bcast:192.168.2.255 Mask:255.255.255.0
inet6 addr: fe80::6690:c1ff:fe1d:24c4/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:3635 errors:0 dropped:0 overruns:0 frame:0
TX packets:5052 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:628449 (628.4 KB) TX bytes:3576722 (3.5 MB)

```

Figure 7 - *ifconfig* (v01.15.58)

User Enumeration

No novel information was extracted from the `/etc/passwd` and `/etc/shadow` files, however it was confirmed that the password hash in the `/etc/shadow` file matched the root password located in the `vinda` file, as demonstrated in Figure 10. Upon inspection of `/etc/passwd~` file (a backup version of `/etc/passwd`), existence of a user called `ruby` was discovered with a home path set to `/home/ruby`, which existed as a blank directory in the file system - likely being a remnant from a previous firmware version.

```

root@rockrobo:~# ~/z5206677/bat /etc/passwd
File: /etc/passwd
Size: 1.1 KB
1 root:x:0:0:root:/root:/bin/bash
2 daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
3 bin:x:2:2:bin:/bin:/usr/sbin/nologin
4 sys:x:3:3:sys:/dev:/usr/sbin/nologin
5 sync:x:4:65534:sync:/bin:/bin/sync
6 games:x:5:60:games:/usr/games:/usr/sbin/nologin
7 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
8 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
9 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
10 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
11 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
12 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin

```

Figure 8 - `/etc/passwd` (v01.15.58)

```

root@rockrobo:~# ~/z5206677/bat /etc/shadow
File: /etc/shadow
Size: 679 B
1 root:$6$mpt0wWOW$DpR00/CdKUfpapA3rEGL/4m6WZ0
2 daemon:*:16652:0:99999:7:::
3 bin:*:16652:0:99999:7:::
4 sys:*:16652:0:99999:7:::
5 sync:*:16652:0:99999:7:::
6 games:*:16652:0:99999:7:::
7 man:*:16652:0:99999:7:::
8 lp:*:16652:0:99999:7:::
9 mail:*:16652:0:99999:7:::
10 news:*:16652:0:99999:7:::
11 uucp:*:16652:0:99999:7:::
12 proxy:*:16652:0:99999:7:::

```

Figure 9 - `/etc/shadow` (v01.15.58)

```

λ openssl passwd -6 -salt mpt0wWOW tevlqtrfuggcbahe
$6$mpt0wWOW$DpR00/CdKUfpapA3rEGL/4m6WZ0kRYC5LSaCJSYKj9iHuZp2PUzfolgrGvEHw5tMtRSYLBWSlonusy67027JF/

```

Figure 10 - Generated SHA512 password hash

Power Analysis

A power analysis was performed to determine how to charge the device's battery without requiring the charging dock's charging contacts, as it was difficult to keep the device in contact whilst performing other tests. Figure 11 illustrates the disassembly of the charging dock, which reveals the power leads that connect to the charging contacts. Measurement of the charging terminal voltages whilst loaded and unloaded revealed that dock's charge controller outputs $\sim 4.2\text{VDC}$ when there is no vacuum connected, and $\sim 20.4\text{VDC}$ when the vacuum is loaded (with an equivalent resistance of $3.7\text{ k}\Omega$)

It was noted that when the 4-wire battery was connected to the device with only the supply leads (+ve and -ve), the device would fail to remain powered on and shutdown after approximately 20 seconds, likely as a fail-safe mechanism as shown in Figure 12.



Figure 11 - Underside of the charging dock

```

Ubuntu 14.04.3 LTS rockrobo tty50
rockrobo login: wait-for-state stop/waiting
havedged: havedged Stopping due to signal 15

* Stopping rsync daemon rsync
* (not running)
* Asking all remaining processes to terminate...
* All processes ended within 1 seconds...
* Unmounting temporary filesystems...
* Deactivating swap...
* Unmounting local filesystems...
* Will now halt
[ 26.948171] [MCU_UART] sent ap poweroff event to mcu

```

Figure 12 - 2-wire battery shutdown log

Data Persistence

Temporary files were created in every directory of the filesystem as to investigate which file paths were untouched during the firmware upgrade, factory reset, and device disassociation (unpair the device via the smartphone application) procedures.

Table 5 - Untouched directories during volatile actions

Firmware Upgrade	Factory Reset	Disassociation
(mmcblk0p11) /mnt/reserve (mmcblk0p1) /mnt/data	(mmcblk0p11) /mnt/reserve	ALL

Where results for upgrade persistence and reset persistence were sensible, the results from device disassociation were alarming, as no data was removed from the device even after the device was deleted from the user’s account. Whilst it could be assumed that device disassociation was then followed by an immediate re-pair process by the same party, failure to follow this flow could potentially lead to PII and UGC being shared to another party if an unpaired device was given away.

Whilst statistical and calibration data (*mmcblk0p11*) are retained during firmware upgrades and factory resets, it can be noted from Figure 13 that user data (*mmcblk0p1*) and system partitions are securely wiped (block-writes rather than just files being unlinked in the partition) during the factory reset procedure, preventing data recovery tools like *photorec*¹² from recovering data.

```

flag a=2,flag b=4,will be recover system
play opt/rockrobo/resources/sounds/en/bl_recovery_bootfailed.wav
Loading file "opt/rockrobo/resources/sounds/en/bl_recovery_bootfailed.wav" from mmc device 2:7
195238 bytes read
sunxi codec request dma 0x5ebb47b4
rr_recovery_pre_check:716:found recovery num 3
Loading file "/boot/zImage" from mmc device 2:7
3882616 bytes read
part recovery valid
recovery from "recovery" to "system_a"
cover init begin
found recovery
set src start=0,src_size=0
found system_a
set dest start=645922816,dest_size=536870912
real_cover:141:total size=536870912,block=1048576

MMC read: dev # 2, block # 253952, count 8192 ... 8192 blocks read: OK

MMC write: dev # 2, block # 1302528, count 8192 ... 8192 blocks write: OK

```

Figure 13 - Serial log during factory reset

¹² <https://www.cgsecurity.org/wiki/PhotoRec>

Static Firmware Analysis

Firmware Extraction and Layout

To statically analyse the firmware of the device (as to provide a ‘offline’ access to the device’s system), a firmware dump was created with the `dd` utility via SSH. It is noted that the device had firewall rules in place which needed to be bypassed prior to connecting (as later explained). Following the commands from Figure 14, a set of eMMC partition dumps were created, which have been tabulated as shown in Table 6.



```
IP=10.10.10.8
for partition in `ssh root@$IP "ls /dev/mmcblk0?* -l" `
do
    ssh root@$IP "sudo dd if=$partition bs=1M | pv | dd of=$(basename $partition).img
done
```

Figure 14 - Firmware dump commands

Table 6 – Firmware partition mapping

Partition	Label	Size	Mount Point	Description
1	UDISK	1.5 GB	/mnt/data	User data
2	boot-res	8 MB		Bootloader resources
3	-	1 KB		(unknown)
4	-	-	-	(does not exist)
5	env	16 MB		Boot environment
6	app	64 MB	/mnt/default	Device data (read only)
7	recovery	512 MB		Stock firmware
8	system_a	512 MB	/	Firmware A
9	system_b	512 MB	/	Firmware B
10	download	528 MB	/mnt/updbuf	Firmware update storage
11	reserve	16 MB	/mnt/reserve	Device statistics

The UDISK partition contains UGC pertaining to map and cleaning data, in addition to device logs and device configurations (such as sound settings, clean scheduling, network settings).

The device contains two copies of the operating system firmware, labelled `system_a` and `system_b`. If the system fails to boot properly, a hardware watchdog will restart the device, and boot into the other partition. Should both partitions result in a failed boot, or a firmware reset is performed, the contents of the `recovery` partition (an old stock firmware version) will be flashed onto both `system_a` and `system_b`. It is noticed that the `recovery` partition is modifiable.

The `reserve` partition contains statistical data (officially termed a ‘blackbox’) storing the total number of cleans performed, bumper sensor clicks, hardware information, and error log events. The file structure of this partition is displayed in Figure 15 on the following page.

```
mmcb1k0p11 (reserve)
|   anonymousid1
|   blackbox.db
|   CompassBumper.cfg
|   counter
|   endpoint.bin
|   hwinfo
|   lds_calibration.txt
|   mcu_ready
|   RoboController.cfg
|   rrBkBox.csv
+---rriot
|   tuyu.json
|   try
```

Figure 15 – File structure of mmcb1k0p11

Commentree

 github.com/featherbear/commentree

A documentation tool was created and developed for this thesis to better mark important regions and annotate lines of plain-text files in the device firmware, which served beneficial in reviewing and analysing text content between research sessions. This tool was used to review and mark the configuration files and logs stored on the device’s filesystem, and additionally provided portability when performing research on different machines. A prototype version is available on GitHub, with plans to improve and complete it in the future.

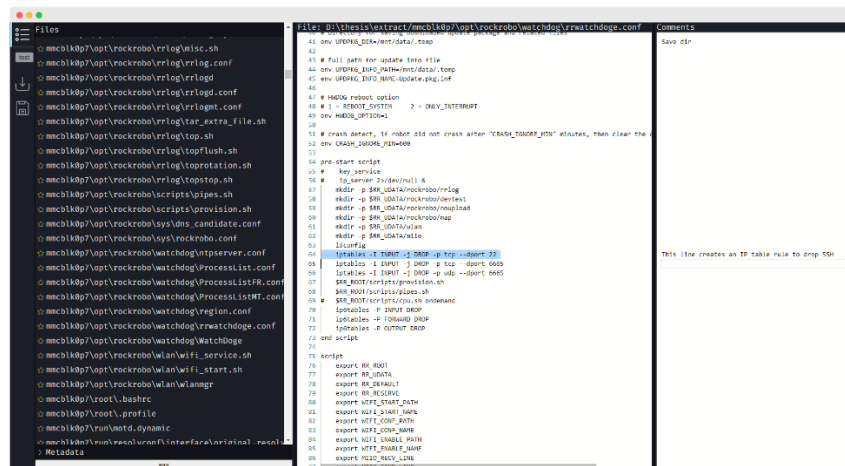


Figure 16 - Screenshot of the Commentree tool

Stock Ubuntu Comparison

As system reconnaissance (see *Device Fingerprinting*) indicated that Ubuntu 14.04.3 LTS was used as the firmware’s base image¹³, altered or modified binaries (such as one that has additional features or possible malicious functionality) could be identified through comparing the version in the base image against the device’s version. A byte-level MD5 hash comparison was performed for programs in the `/bin`, `/sbin`, `/usr/bin`, and `/usr/sbin` directories.

Results concluded that except for one program, all binaries completely matched the base image’s version, which indicates no sign of alteration or modification to existing programs. The binary whose MD5 hash differed¹⁴, `ntpd`, is responsible for retrieval and updating of the device’s time from a time server. When performing a function-level binary comparison with *BinDiff* (as proposed by Costin, Zaddach et al. (2014)), a low similarity ratio of 0.36 was produced as shown in Figure 17 – indicating a large change in program functionality.

Further binary analysis and cross-examination of the assembly call graphs however revealed that the version of `ntpd` on the device was only a stripped build of the base version (4.2.6p5@1.2349-o), built without public key cryptography support (provided by *OpenSSL*).

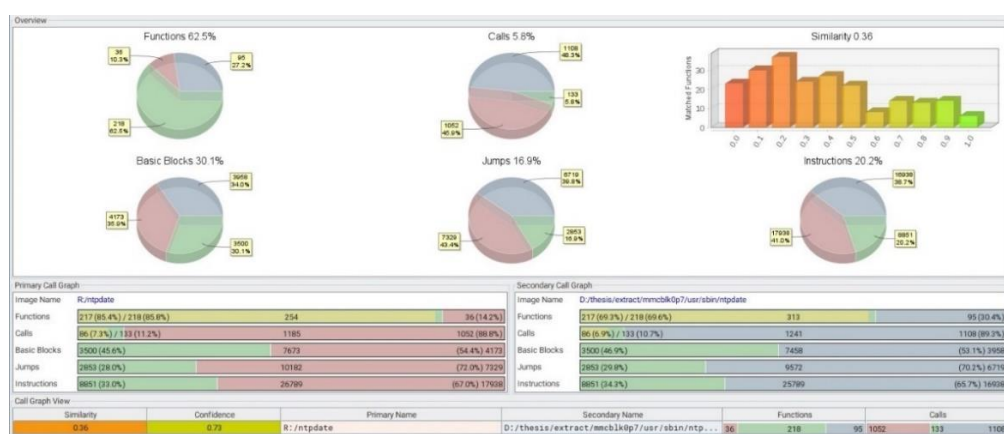


Figure 17 – BinDiff comparison of `ntpdate` (v01.15.58)

It was also noted that alongside the added vendor software in `/opt/rockrobo`, the firmware image contained the additional packages `rsync`, `ccrypt`, and `tcpdump`, however `rsync` and `tcpdump` had no usage calls in any program (as of version firmware 01.15.58).

```

Start-Date: 2016-01-25 11:18:05
Commandline: /usr/bin/apt-get install rsync
Install: rsync:armhf (3.1.0-2ubuntu0.2)
End-Date: 2016-01-25 11:18:11

Start-Date: 2016-04-05 12:30:59
Commandline: /usr/bin/apt-get install ccrypt
Install: ccrypt:armhf (1.10-4)
End-Date: 2016-04-05 12:31:01

Start-Date: 2016-04-25 09:58:29
Commandline: /usr/bin/apt-get install tcpdump
Install: tcpdump:armhf (4.5.1-2ubuntu1.2), libpcap0.8:armhf (1.5.3-2, automatic)
End-Date: 2016-04-25 09:58:33

```

Figure 18 – `apt-get history.log` file

¹³ <http://cdimage.ubuntu.com/ubuntu-base/releases/14.04/release/ubuntu-base-14.04.3-core-armhf.tar.gz>

¹⁴ base md5: 122890cbbaff8ca98f9664add64492bd | device md5: 006a0967281c9a061362086b638a21a4

ADB

ADB, short for Android Debugging Bridge, is a development and utility tool to communicate with an Android device, or any device that implements the server functionality. This tool allows for the management and transfer of files, installation of applications (on an Android device), and access to the device’s shell. In the Roborock S6’s firmware there is a custom version of the *adbd* binary that serves communications (via *FunctionFS*¹⁵) from the micro USB port located at the top of the vacuum cleaner, as visualised below.



Figure 19 – Exposed micro USB connector on the Roborock S6

The binary has additional functionality to perform system tests (the *uart_test* command) and flashing of the device (the *ruby_flash* command) without requiring the disassembly of the device to gain access to the programming pins or test pads.

```

SYS_PASSWD = /mnt/default/vinda := ABCD1234ABCD1234

# Get challenge
CHALLENGE $= adb shell [SYS_PASSWD]rockrobo dynamickey

# Generate response
ADB_PASSWD = generate(challenge, device_id)

# Perform command
adb shell [SYS_PASSWD][ADB_PASSWD] [COMMAND]

```

Figure 20 – Custom *adbd* auth challenge flow

Access to the ADB interface is restricted however, as a dynamic challenge / response auth process is required to issue *adb shell* commands. The authentication flow summarised in Figure 20 is as follows:

1. The user requests the challenge token, providing the 16-byte *vinda* password, followed by ‘*rockrobo dynamickey*’
2. The user generates the response¹⁶ based off the challenge token and the device’s ID
3. The user issues a command, providing the *vinda* password string, the response token, and the command they want to execute

¹⁵ <https://www.kernel.org/doc/Documentation/usb/functionfs.txt>

¹⁶ <https://featherbear.cc/UNSW-CSE-Thesis/posts/execs/usr-bin-adbd/#challenge-response-generation>

If the auth challenge succeeds, further custom access control implementations restrict the commands that can be executed, based off a value of a property named `adb_lock` in the read-only `/mnt/default/adb.conf` file. It is noted that the execution of any arbitrary command is only possible when the value is set to `0`, however this is never possible as the `adb` binary will reset the value to `1`, as shown in the assembly call graph below.

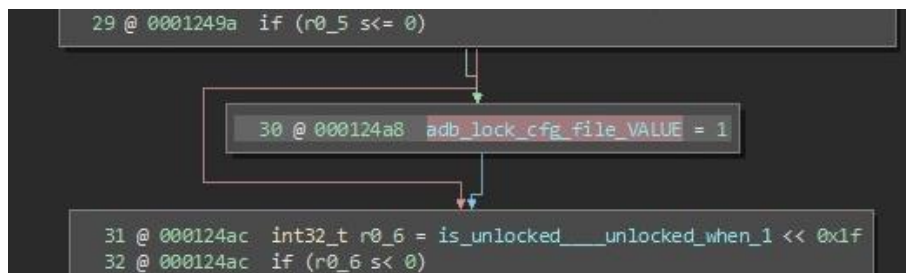


Figure 21 – adbd lock reset flow

A novel command injection vulnerability was discovered in the *adbd* binary which subverted the access restrictions, allowing any arbitrary command to be executed regardless of the current access level. Whilst the ampersand (&), semicolon (;), pipe (|) and backtick (`) characters are sanitised from the command string to prevent command chaining, failure to filter out the dollar sign (\$) character allows for command expansion to be performed via the following command.

```
adb shell [SYS PASSWD][ADB PASSWD] uart test $(COMMAND)
```

A proof of concept has been made available¹⁷. This vulnerability additionally exploits the fact that the `uart_test` command actually spawns a `/bin/sh` shell via a `libc system` library function call, which supports command expansions. Arbitrary command execution is obtained, as demonstrated by the proof of concept below. This exploit could be used to exfiltrate data from the system (such as map data and wireless credentials), write to the filesystem, or possibly gain SSH access (as later explained).

```
D:\thises\misc\adbd_launcher (master)
λ py -3 adBStart.py "uart-test $(cat $(base64 /etc/passwd))"
challenge='iUNs5vuhymIEJBRTiqsRtU' response='su-_71EB'
cmd='adb shell "CRA[FCEQBPTUV R$u- _71EB uart test $(cat $(base64 /etc/passwd))"'
cat: cm9vdDp04oJAMDPyb2900i9yb2900i9iaW4vYmFzaApNTIwNjY3Nzo6MDowOiwSLDovcm9vdDov:
cat: YmlLZDJlc2JcZGZGFblW9uQngMT0xOmRhZW1ibjovdXNyLSNiAe46L3VzcigzYmULZ25vbG9naW4KChj:
cat: YmULONG6MjoyOMjbjoYmUL0i9ic3lvc2Jpbj9ub2xzZ2LuNm5czp4O3M6MzpeXm6L2Rldjov:
cat: dXNyLSNiAe4vbm9sb2dpbgpzeW5jOng6ND02NTUzNDPzeW5joi9iaW46L2Jpbj9zeW5jCmdhbWVz:
cat: Ong6ND02MDpnYW1lczoZdXNyLSdhbWVzOi9ic3lvc2Jpbj9ub2xzZ2LuCmhbjpk4ojYGMT16bWFu:
cat: Oi92YXlYv2FjaGVubWVuOi9ic3lvc2Jpbj9ub2xzZ2LuCmxwOGNoZm03omxwOi92YXlYvc3Bvb2w:
cat: bHBkOi9ic3lvc2Jpbj9ub2xzZ2LuZmlhaHw6eDo4ogjg6bWFpbDovdmFuZGl2aHw6L3VzcigzYmUL:
cat: L25vbG9naW4KBmV3cmZp04kj0kTpuZXdzOi92YXlYvc3Bvb2wvbWV3czoZdXNyLSNiAe4vbm9sb2dp:
cat: bgp1dWNwong6MTA6MA6dXvjcdovdmFuZl3Nwb29sLSV1Y3A6L3VzcigzYmULZ25vbG9naW4KCkhj:
cat: eHEk6eDOzMoxZmcwcm94EtoYvYmUL0i9ic3lvc2Jpbj9ub2xzZ2LuCnd3dy1kYXRhOneg6MZm6Mz:
cat: M6MzpeXm6L2Rldjov
```

Figure 22 – *adbd* command injection vulnerability PoC

It is worthwhile to state the limitations of this exploit, as its success relies on the knowledge of the contents of the *vinda* file, and the device ID. Whilst the device ID is easily obtainable via viewing the USB device information, gaining access to the *vinda* file is non-trivial, and requires either the disassembly of the device to access the UART pins, or via another exploit.

Nevertheless, whilst this novel exploit does not provide a means to instantly gain control over a device, it provides a post-exploitation method to easily interface with the device over USB, should SSH or serial connections become inaccessible.

¹⁷ <https://featherbear.cc/UNSW-CSE-Thesis/poc/>

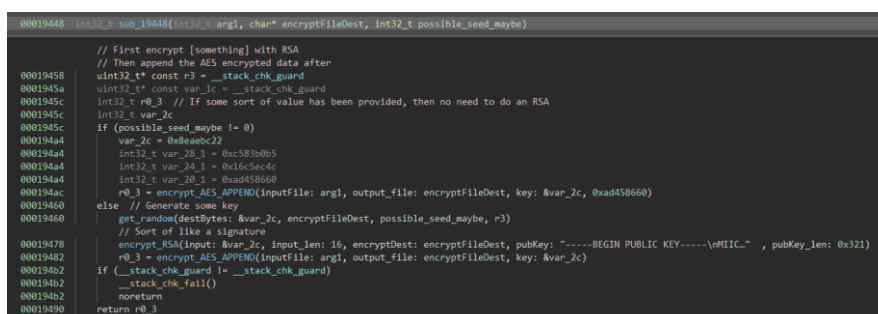
Device Logs (rrlogd)

The *rrlogd* binary is responsible for the management, rotation and uploading of logs to the Xiaomi File Data Service server¹⁸ (as determined by the device’s manufacture release).

Through both static analysis of the binary, and dynamic analysis of the filesystem, the following categories of log data was observed: application logs relating to vacuum cleaning functionality, application configuration, mapping data¹⁹, firmware upgrade logs, device hardware information, system lifecycle logs, running processes, network information and cleaning statistics. Newer versions of *rrlogd* (i.e. in the v02.29.02 firmware) also include the ability to upload network captures, as later explained (see *Network Capture*).

Before the logs are upload, they are compressed and encrypted with RSA + AES, as evident in Figure 23. Log files (see Appendix 1) are primarily sourced from the following directories:

- /mnt/data/rockrobo/rrlog/
- /dev/shm/
- /mnt/reserve/

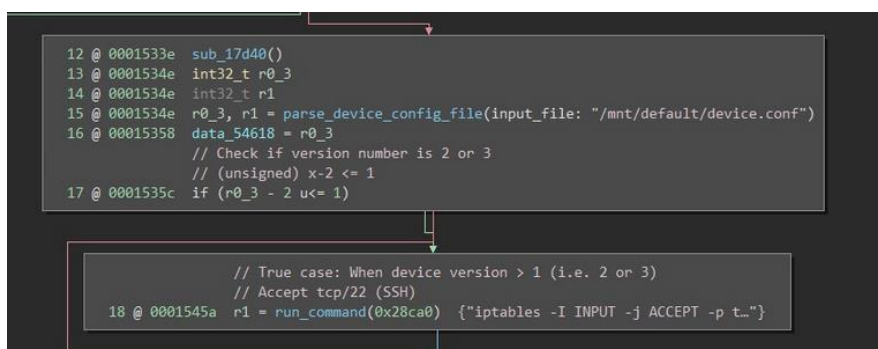


```

00019448  int32_t sub_19448(int32_t arg1, char* encryptFileDest, int32_t possible_seed_maybe)
// First encrypt [something] with RSA
// Then append the AES encrypted data after
00019458  uint32_t* const r3 = __stack_chk_guard
0001945a  uint32_t* const var_1c = __stack_chk_guard
0001945c  int32_t r0_3 // If some sort of value has been provided, then no need to do an RSA
0001945e  int32_t var_2c
0001945f  if (possible_seed_maybe != 0)
00019461  var_2c = 0x8eaebc22
00019463  int32_t var_2b_1 = 0xc581b0b5
00019465  int32_t var_2a_1 = 0x1e5e0e4e
00019467  int32_t var_20_1 = 0xad5866d8
00019469  r0_3 = encrypt_AES_APPEND(inputFile: arg1, output_file: encryptFileDest, key: &var_2c, 0xad5866d8)
0001946b  else // Generate some key
0001946d  get_random(destBytes: &var_2c, encryptFileDest, possible_seed_maybe, r3)
0001946f  // Sort of like a signature
00019471  encrypt_RSA(input: &var_2c, input_len: 16, encryptDest: encryptFileDest, pubKey: "-----BEGIN PUBLIC KEY-----\nMIIc", pubKey_len: 0x321)
00019473  r0_3 = encrypt_AES_APPEND(inputFile: arg1, output_file: encryptFileDest, key: &var_2c)
00019475  if (__stack_chk_guard != __stack_chk_guard)
00019477  __stack_chk_fail()
00019479  noretun
00019481  return r0_3

```

Figure 23 - Disassembly of the encryption routine in *rrlogd* (v01.15.58)



```

12 @ 0001533e  sub_17d40()
13 @ 0001534e  int32_t r0_3
14 @ 0001534e  int32_t r1
15 @ 0001534e  r0_3, r1 = parse_device_config_file(input_file: "/mnt/default/device.conf")
16 @ 00015358  data_54618 = r0_3
// Check if version number is 2 or 3
// (unsigned) x-2 <= 1
17 @ 0001535c  if (r0_3 - 2 u<= 1)
// True case: When device version > 1 (i.e. 2 or 3)
// Accept tcp/22 (SSH)
18 @ 0001545a  r1 = run_command(0x28ca0) {"iptables -I INPUT -j ACCEPT -p tcp"}

```

Figure 24 – iptables allow rule in *rrlogd* (v01.15.58)

It was curiously noted that *rrlogd* implemented functionality to potentially unblock inbound SSH connections depending on the device model. However the specific DUT (Roborock S6) would not satisfy the required conditions and so was unaffected.

¹⁸ <http://docs.api.xiaomi.com/en/fds/>

¹⁹ Determined to be stored as in the *RRMapFile* format.

See <https://github.com/marcelrv/XiaomiRobotVacuumProtocol/blob/master/RRMapFile/RRFileFormat.md>

Upgrade Analysis (Version 02.29.02)

Whilst upgrades are a means to add additional features or improve the performance of existing functions, upgrades additionally assess a company’s response to security vulnerabilities and privacy concerns. It is rather uncommon for vendors to include internal system changes, or detailed security notes in upgrade changelogs as this information will not be of any use to common end-users. Independent research must therefore be performed to produce a system changelog that addresses security and/or privacy concerns.

The DUT was upgraded from v01.15.58 (25th March 2020) to v02.29.02 (28th April 2022), with firmware images being dumped between the incremental upgrades. Static firmware analysis was then performed to compare the changes in the filesystem between versions and has been collated in the table below. In this section of the thesis, the base firmware (v01.15.58) will be compared against the latest version (v02.29.02) to best discern Roborock’s response to security and privacy concerns throughout the product’s life.

Table 7 – Firmware upgrade changelog

Firmware	Official Changelog	Unofficial System Changelog
01.17.08 (17th April 2020)	- Supports multi-floor map saving and robot knows which floor it is - Update to new structured SLAM algorithm to make map more reliable - Support customised room cleaning sequence - Support no-mop zone	<i>iptables</i> enforcement to drop SSH <i>rrlogd</i> <i>WatchDoge</i> - Utilities change to <i>busybox</i> - SSH server changed to <i>dropbear</i> <i>rriot_rr</i> added (but not enabled)
01.19.98 (9th June 2020)	- Improvise Wi-Fi Easy Connect - Overall improvements - Bug fixes - UX fixes	Serial handler changed to <i>rr_login</i>
01.20.76 (23rd June 2020)	- Obstacle avoidance enhancements - Bug fixes and UI optimisation	-
...	...	...
02.29.02 (28th April 2022)	Optimized the quick mapping experience	<i>rriot_rr</i> enabled

Firmware Images

A security assessment of the firmware upgrade procedure was beyond the scope of this thesis, however it is worthwhile to mention that upgrade packages are encrypted, as observed when intercepted upgrade packages were not trivially extractable. Brief analysis of the *SysUpdate* binary indicate that packages are additionally signed to prevent unauthorised firmware upgrade files. Whilst a subroutine (as annotated below) indicates that files may be encrypted with *ccrypt*, this routine is deprecated given that *ccrypt* is removed in later firmware versions.

```

00013d34  int32_t DecryptFile(char* file)
00013d58      uint32_t* const var_1c = __stack_chk_guard
00013d5e      void var_11c
00013d5e      memset(&var_11c, 0, 0x100, __stack_chk_guard)
00013d76      __sprintf_chk(&var_11c, 1, 0x100, "ccrypt -d -K %s %s", "rockrobo", file)
00013d92      log(0x67, main: "DecryptFile", data_5e504 & 0x800, format: "Decrypting %s ...", file, "rockrobo")
00013d9c      int32_t r0_3 = InvokeSystem(&var_11c, 0, 1)
00013da2      if (r0_3 == 0)
00013db8          log(0x6b, main: "DecryptFile", data_5e504 & 0x2000, format: "Decryption failed")
00013dca      if (__stack_chk_guard != __stack_chk_guard)
00013dca          __stack_chk_fail()
00013dca      noreturn
00013dc8      return r0_3

```

Figure 25 – Obsolete decryption routine in *SysUpdate*

Broad System Changes

A filesystem comparison between the base firmware and latest firmware revealed a system migration towards an embedded system design, where functionality is stripped and unused tools are removed from the firmware. In comparison to the base firmware (10680 files totalling 242 MB), a 60% reduction in filesystem size was observed (1976 files totalling 98 MB).

Most noticeably, many utilities were replaced with a stripped-back *busybox* distribution (v1.24.1), commonly used in embedded Linux systems to decrease firmware image size. Ubuntu-like and Debian-like files and folder structures (including the *apt-get* and *dpkg* package managers) were additionally removed in later firmware versions. Whilst the removal of package managers does not prevent foreign binaries from being loaded and executed, it does significantly increase the time required to execute foreign binaries.

It was also noted that the *rsync* and *ccrypt* binaries previously found in base firmware were removed, however the added *tcpdump* package remained.

MD5 hashes were calculated for the binaries in the latest firmware and were compared against the base firmware (see *Stock Ubuntu Comparison*) to determine if files were changed. All shared binaries (ignoring programs replaced with *busybox*) in the */bin*, */sbin* and */usr/sbin* directories matched, indicating that no changes exist. Whilst some binaries in the */usr/bin* directory were modified, functional analysis comparisons concluded that only performance changes were made.

We now outline the non-trivial changes noticed between the base firmware and latest firmware.

IPv6 Routing

As previously assessed during the dynamic firmware analysis of the Roborock S6 (see *Network Capability*), no *ip6tables* rules were applied in the base firmware – however as the device did not request nor assign itself an IPv6 address (other than its link-local address), access to exposed ports on the device via IPv6 were denied. Despite the device being unreachable via IPv6, newer firmware versions explicitly prevent IPv6 traffic (in both directions) by enforcing *DROP* rules to all network chains, as shown in the program output below.

```
> ip6tables -L
Chain INPUT (policy DROP)
target    prot opt source                destination

Chain FORWARD (policy DROP)
target    prot opt source                destination

Chain OUTPUT (policy DROP)
target    prot opt source                destination
>
```

Figure 26 – *ip6tables* results (v02.29.02)

Authentication Flow Modification

In the base firmware, device authentication from a terminal interface (such as through SSH or serial) was managed through the standard *pam_unix.so* module, which would utilise the authentication information within the */etc/passwd* and */etc/shadow* files. It was noted that the root password was identical to the decrypted value of the *vinda* file contents in the device data partition (see *Firmware Extraction and Layout*).

Newer firmware versions (as of firmware version 01.19.98, released 9th June 2020) however no longer use the standard module to authenticate login requests, and instead use a custom authentication routine called *verify_shadow* located in the vendor’s *libuart_api.so* library. As visible in the disassembly below, the presence of a */mnt/default/shadow* file is noted – whose purpose likely mirrors the */etc/shadow* file (to store password hashes). The presence of */mnt/default/shadow.sign* is also noted, used in an RSA signature check to verify the integrity of */mnt/default/shadow*. It is inferred that modification to the *root* password is difficult without knowledge of the RSA key used to perform the signature signing.

This authentication flow modification does not apply to all authentication interfaces on the system, as the */bin/login* and *su* binaries still utilise the standard Unix authentication. Only programs which specifically use the *libuart_api.so* library (i.e. vendor software) are affected by this authentication flow modification.

```

00016b62  int32_t r3 = *__stack_chk_guard
00016b6a  int32_t r0_1
00016b6a  if (load_libmbedtls(arg1, arg2, arg3, r3) == 0)
00016b8c  int32_t r0_2 = load_libcrypt()
00016b90  if (r0_2 == 0)
00016be0  int32_t r0_7 = access("/mnt/default/shadow", r0_2) // Check if file exist
00016be6  if (r0_7 == 0)
00016c16  void* shadowFilePath = fopen("/mnt/default/shadow", &data_19fd4)
00016dd0  if (shadowFilePath == 0)
00016dd0  if (*uart_api_print_level > 2)
00016dea  __printf_chk(1, "[%8u]<t%lu><E>open shadow fail\n", rua_ms_now(), pthread_self())
00016dd0  goto label_16b76
00016c26  void var_2ac
00016c26  void* var_2b8_1 = &var_2ac
00016c36  void var_22c
00016c36  memset(&var_22c, r0_7, 0x200)
00016c40  memset(var_2b8_1, r0_7, 0x80)
00016c58  void* r0_13
00016c58  while (true)
00016c58  r0_13 = fgetspent(shadowFilePath)
00016c5c  uint32_t r3_8 = *uart_api_print_level
00016d0a  int32_t var_2c8
00016d0a  if (r0_13 == 0)
00016d0a  if (r3_8 > 2)
00016d62  __printf_chk(1, "[%8u]<t%lu><E>SHA256 pass\n", rua_ms_now(), pthread_self(), var_2c8)
00016d0e  endspent()
00016d14  fclose(shadowFilePath)
00016d20  r0_1 = RSA_verify("/mnt/default/shadow", "/mnt/default/shadow.sign")
00016d26  if (r0_1 == 0)

```

Figure 27 – *verify_shadow* function routine

The DUT specifically used during this thesis however did not contain the *shadow* or *shadow.sign* file, likely due to the authentication flow changes not yet propagating through the manufacture and initial device flashing process. Consequently all authentication methods in firmware versions which utilised the *verify_shadow* routine (serial, SSH, ADB) would always fail, as the missing files would trigger early exit conditions.

It is noted that the manufacture date (June 2020) of the Roborock S6 vacuum cleaner specifically used during the thesis was unideal, as it coincides with the release month of firmware version 01.19.98, where this authentication flow modification was implemented. This raises uncertainty regarding how the vendor may have modified the filesystem. As the base firmware was versioned in March, it is assumed that the DUT has the filesystem structure of a device manufactured prior to June, and hence prior to the authentication flow modification.

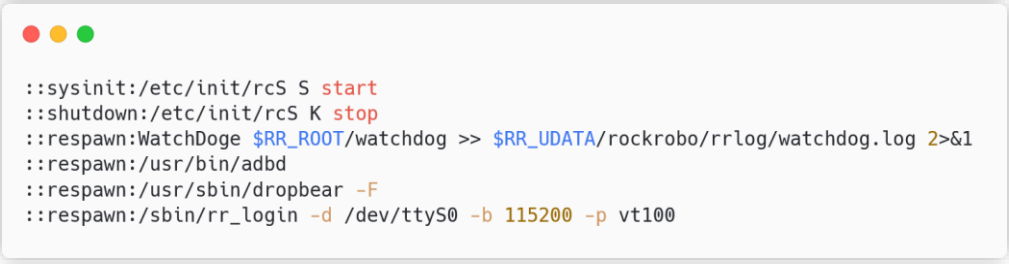
It would be possible to patch the *libuart_api.so* binary to always return a successful verification result, however this was not tested as it would require greater effort as compared to other trivial methods to gain access.

Serial Access

Later firmware versions replaced the original serial handler `/sbin/getty` with a custom implementation named `rr_login`. Similar to the patched SSH interface, this binary restricted serial access (see Appendix 2) to only the root user, and utilised the `verify_shadow` authentication flow - which would always fail with the DUT.

As consequence to the serial login always failing because of the missing `shadow` and `shadow.sign` files, the following steps were developed to regain access to the console by replacing the serial handler in the `/etc/inittab` file (see Figure 28).

-
1. Boot into the `u-boot` debug shell by sending ‘s’
 2. Overwrite the `init` entry point to start `/bin/bash`
 - `setenv setargs_mmc ${setargs_mmc} init=/bin/bash`
 3. Resume system boot with the `boot` command
 4. Disable the hardware watchdog
 - `echo V > /dev/watchdog`
 5. Edit the `/etc/inittab` file
 - Remove `::respawn:/sbin/rr_login -d /dev/ttyS0 -b 115200 -p vt100`
 - Append `ttyS0::respawn:/bin/login`
 6. Reboot the system with the `reboot` command
-



```

::sysinit:/etc/init/rcS S start
::shutdown:/etc/init/rcS K stop
::respawn:WatchDoge $RR_ROOT/watchdog >> $RR_UDATA/rockrobo/rrlog/watchdog.log 2>&1
::respawn:/usr/bin/adbd
::respawn:/usr/sbin/dropbear -F
::respawn:/sbin/rr_login -d /dev/ttyS0 -b 115200 -p vt100

```

Figure 28 – SysV configuration script (v02.29.02)

Upon modification of `/etc/inittab`, serial access was restored allowing access to the device with the original root password.

SSH Access

In the base firmware, a stock *OpenSSH* server was exposed on `tcp/22` on both IPv4 and IPv6 addresses (albeit no IPv6 connection was able to be established), the upgraded firmware revealed that the SSH server was replaced with *dropbear* (v2013.60), a compact SSH server that is commonly used in embedded Linux system. Notably, this *dropbear* binary was modified to limit access solely to the root user and implemented the aforementioned `verify_shadow` authentication flow. The standard Unix authentication flow can be restored by replacing the *dropbear* binary with a stock or alternate server binary.

It was also noted that the *dropbear* binary only offers two legacy key exchange algorithms, *diffie-hellman-group1-sha1* and *diffie-hellman-group14-sha1*, both which are considered to be weak by modern cryptography standards and may be vulnerable to the attacks like *Logjam* (Adrian, Bhargavan et al. 2015).

A binary analysis of the *WatchDoge* and *rrlogd* binaries in the latest firmware reveal that extra functionality was implemented to further enforce SSH access restrictions (as previously established in the *Network Capability* section), as evident in Figure 29, where the very first functional instruction was to call `iptables -I INPUT -j DROP -p tcp --dport 22`.

```

WatchDoge.bndb (ELF Graph)
int32_t main(int32_t argc, char** argv, char** envp)

00013f18 main:
0 @ 00013f34 uint32_t* const var_2c = __stack_chk_guard
1 @ 00013f36 void var_ac
2 @ 00013f36 memset(&var_ac, 0, 0x80, __stack_chk_guard)
3 @ 00013f42 call_system("iptables -I INPUT -j DROP -p tcp..." )

```

Figure 29 – WatchDoge process enforcing iptables

Network Capture

Static analysis of the updated *rrlogd* binary in the latest firmware revealed new IPC signal handling behaviour. When the `MSG_LOG_DEBUG_ENABLE` signal was received, a function in the *wlanmgr* process is called, whose behaviour is as described below.

Table 8 – wlanmgr routine 0x136e8 (v02.29.02)

Signal	Action	Description
0	<code>rm -rf /mnt/data/debug</code>	Delete debug files
1	<code>tcpdump -i any -s 0 -C %lu -W %d -Z root -w %s/%s/%s &</code>	Perform packet capture
2	<code>killall tcpdump</code>	Stop packet capture
3	<code>/opt/rockrobo/wlan/wifi_debug_collect.sh</code>	Collect other network information

Most notably, the *wlanmgr* process was observed to be able to create network packet captures via *tcpdump*. When *rrlogd* receives the `MSG_LOG_DEBUG_UPLOAD_DATA` signal, the packet capture dump along with other files (as referenced by the `wlan_debug_collect.sh` script) are uploaded to the log servers. The table below details the content of uploaded data.

Table 9 – Collected network data (v02.29.02)

Filename	Source	Description
<code>resolv.conf</code>	<code>/etc/resolv.conf</code>	DNS nameserver configuration
<code>netstat.txt</code>	<code>netstat -anp</code>	List of all sockets and related processes
<code>ifconfig.txt</code>	<code>ifconfig</code>	Overview of network interfaces
<code>network_packet.pcap</code>	<code>(wlanmgr)</code>	Packet captures

As the network packet capture is performed using *tcpdump*, only TCP packets are captured within the dump file, and does not include any UDP traffic. It should also be noted that the visibility of network traffic is limited to the traffic broadcasted by the access point, as only a passive network capture is performed.

Network Activity Analysis

This section covers the security and privacy assessments pertaining to network traffic and device communications. Network packet captures were performed during the research period, capturing network activity during the following scenarios and events:

- Device is uninitialised – Perform pairing and initial setup
- Device is initialised – Perform cleaning
- Device is initialised – Perform firmware upgrade
- Device is initialised – Device idle

Test Infrastructure Setup

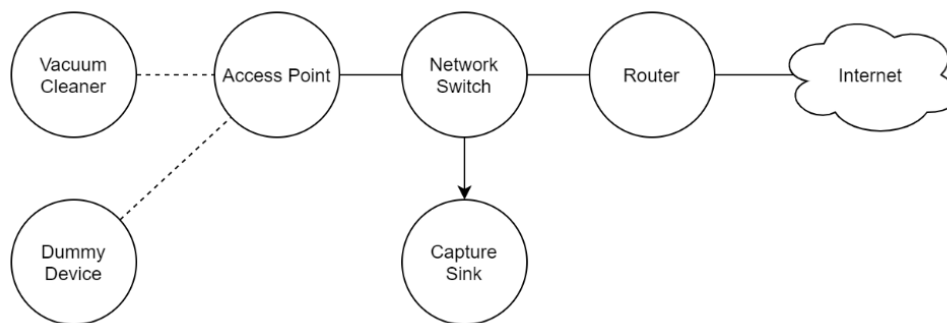


Figure 30 - Isolated network connection diagram

Table 10 - Network equipment list

Label	Device	Purpose
Vacuum Cleaner	Roborock S6	(Device Under Test)
Dummy Device	Lenovo M93p Tiny	Simulate network traffic
Access Point	Ubiquiti UniFi UAP	Provide Wi-Fi connectivity
Network Switch	TP-Link TL-SG105E	Network expansion, port mirror
Capture Sink	Mac Mini	Port mirror
Router	Routerboard RB1200	Network gateway

An isolated network (disconnected from personal devices) was set up to securely monitor the network traffic of the vacuum cleaner without external influences. The Roborock S6 vacuum cleaner was connected to a WPA2-PSK secured wireless network (via the access point), and network activity was *port-mirrored* to a capture sink for packet capturing purposes.

Port-mirroring is a network observability function to copy network traffic flowing through a switched port to another port, often to allow for the transparent monitoring of data without requiring a physical network tap. Given the nature of network switches only forwarding data to the required destination port (compared to a network hub which broadcasts data to all connected ports/clients), port mirroring allows for the traffic of the wireless access point (and consequently the vacuum cleaner) to be monitored. As access points function as network hubs, the port-mirroring of the access point effectively provides a means to view all the packets that the vacuum cleaner itself can see.

Due to the port mirroring functionality limitations specific to the network switch used during this thesis (*TP-Link TL-SG105E*), modifications to the capture sink’s NIC required to only permit unidirectional data transmission from the switch to the capture sink, as to effectively disconnect the capture sink from the network whilst still receiving port mirrored traffic.

As the device may exhibit different behaviour under a sterile environment (no other devices connected that produce network activity), a “dummy device” was connected to the same wireless network to simulate common traffic with the *nping* utility.

Packet captures were performed in several batches over several months under the previously mentioned test scenarios, with most captures being performed whilst the device was idle - as it would best reveal any network activity patterns. Packet captures were performed on both firmware versions 01.15.58 and 02.29.02.

Data Transparency Preparation

Given the encrypted nature of network communications present on the device, steps must be taken to decrypt or otherwise transparently observe the encapsulated payload or message. Before exploring the actions taken in this thesis to meaningfully observe the network traffic, we first overview common issues faced by developers and other security professionals when dealing with analysis of encrypted network traffic.

Table 11 – Comparison of data transparency methods

	SSLKEY LOGFILE	MITM (e.g. Burp Suite)	Frida	Manual Patching
(Straight-forward) System-level configuration possible	✓	✓		
Always respected by application			✓	✓
Non-HTTP TCP traffic support		✓	✓	✓
UDP traffic support			✓	✓
Application-level crypto support			✓	✓
Requires access to the binary			✓	✓
Difficulty	Easy	Medium	Hard	Even Harder

Certain programs and web browser such as Firefox and Chrome implement a development feature where SSL / TLS session secrets can be stored in a file (via the *SSLKEYLOGFILE* environment variable²⁰). This file can then be used to aid packet capture analysis tools such as Wireshark to decrypt encrypted SSL / TLS sessions, and consequently view the unencrypted payloads. Whilst seemingly useful, this method is not protocol agnostic and can only be used to decrypt website traffic.

Embedded systems such as the Roborock S6 may include software that do not communicate over HTTP(S) – in fact this is often the case as the adoption of MQTT or custom protocols are becoming more prevalent in IoT systems (Mishra and Kertesz 2020). There is also no guarantee that all applications will respect the presence of the *SSLKEYLOGFILE* variable.

²⁰ https://firefox-source-docs.mozilla.org/security/nss/legacy/key_log_format/index.html

As observed during the static firmware analysis, binaries of the DUT implement application-level encryption, and hence do not rely on SSL / TLS encryption to secure communications. Even if SSL / TLS encryption could be stripped, this method does not provide any means to decrypt application-level encryption. This limitation also exists in MITM solutions such as *Burp Suite*, *mitmproxy* and other associated utilities²¹ that only aid in SSL / TLS decryption.

Dynamic instrumentation frameworks like *Frida*²² exist to solve the inability to decrypt application-level encryption, by instead hooking into the program’s function calls. Through function hooking, unencrypted payloads can be obtained by intercepting the pre-encryption and post-decryption stages. The utilisation of Frida was not pursued due during the thesis due to initial technical issues and time constraints.

The modifiable nature of binary files in the filesystem instead allowed for the injection of crafted ARM assembly code that relayed the pre-encrypted / post-decrypted payloads over the network to an arbitrarily defined address *10.251.252.253:28422 (UDP)*, as seen in the figure below. By transmitting the payload data over the network, payloads were also captured in the packet capture, which consequently simplified the process of correlating network traffic.

```

/* Send test to 10.251.252.253:28422 */
int hook(char* data, size_t len) {
    int sock = create_udp4_connection(IPV4_ADDR(10, 251, 252, 253), 28422);
    send(sock, (void*) data, len, 0);
}

```

Figure 31 - Crypto function hook source code

It was also noted that certain encrypted traffic (such as the upload of log data) could be studied by simply viewing the underlying log files within the filesystem.

Overview of Network Endpoints

The table below summarises the endpoints that the Roborock S6 vacuum cleaner connects to and is provided to give context to the upcoming observations and results.

Table 12 – Overview of network endpoints

Endpoint	Protocol	Description	Used in 01.15.58	Used in 02.29.02
ms.tuya.eu.com	MQTT	Inbound requests		✓
m2.tuya.eu.com	MQTT	Inbound requests	✓	
a2.tuya.eu.com	HTTPS	Outbound requests	✓	✓
awsde0.fds.api.xiaomi.com	FDS ²³	Logs upload	✓	✓
xx.ot.io.mi.com	HTTP	(unknown)	✓	
xx.ott.io.mi.com	HTTP	(unknown)	✓	

²¹ Tools exist to strip HSTS certificate pinning mechanisms, that would otherwise prevent MITM techniques. See <https://github.com/shroudedcode/apk-mitm>

²² <https://frida.re/>

²³ <http://docs.api.xiaomi.com/en/fds/>

Figure 32 below visualises the nature of dataflows between the device and external endpoints and displays the inter-process communication flow between relevant processes on the device.

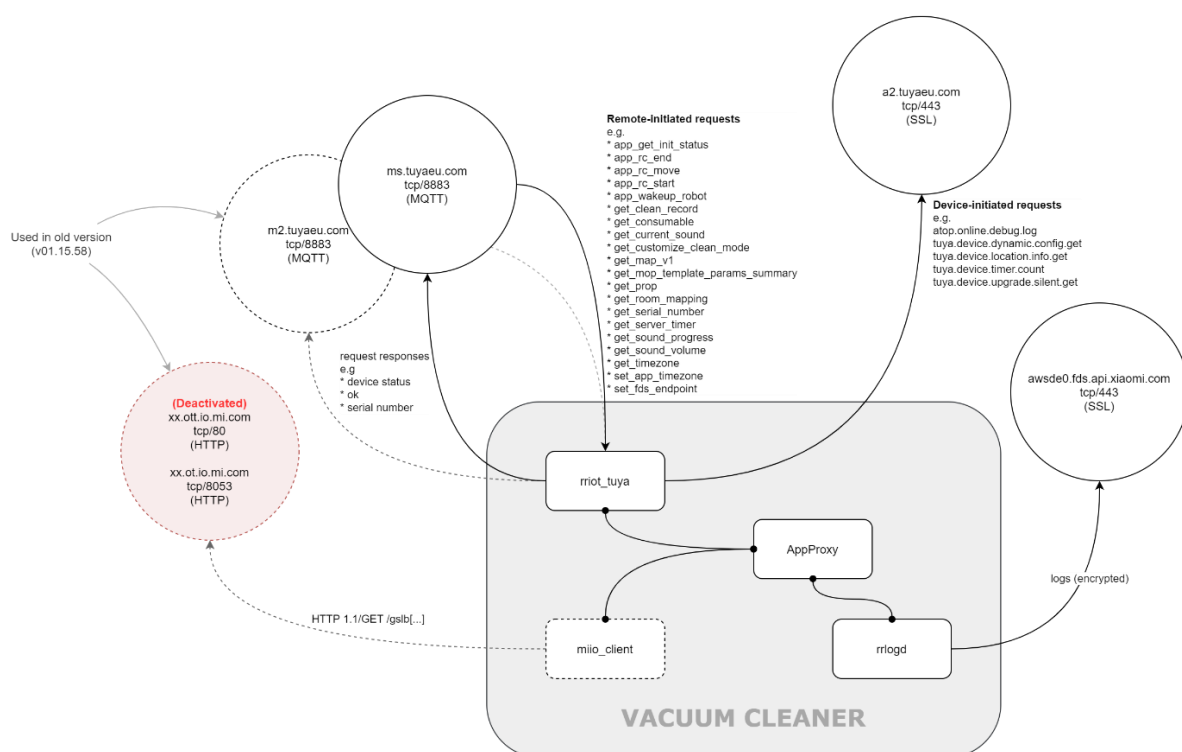


Figure 32 – Network communication diagram

Network Content Analysis

Exploration

The MQTT servers (*m2.tuya.eu.com*, *ms.tuya.eu.com*) are responsible for the requests sent by the server to the device, such as status checks and queries for device settings. Commands sent by the smartphone companion application to remotely navigate the Roborock S6 are also delivered through the MQTT protocol (as labelled by the *app_rc_move* request). Payloads are packed as JSON for both requests and replies.

The *a2.tuya.eu.com* endpoint is responsible for requests initiated by the device and set to the server. These requests include firmware update checks and configuration update polls and are also packed in the JSON format.

As previously mentioned in the static analysis of the *Device Logs* (*rrlogd* binary and further explored during its upgrade analysis), logs are compressed and secured with RSA + AES before being uploaded to the Xiaomi File Storage Service (FDS) server (*awsde0.fds.api.xiaomi.com*). These logs included application config and runtime data, device data, system lifecycle data, cleaning statistics and network capture data (as seen in v02.29.02).

In version 01.15.58 of the device firmware, HTTP GET requests were issued to the *xx.ot.io.mi.com* and *xx.ott.io.mi.com* endpoints, however these endpoints appear to be deactivated as they produced no meaningful response (HTTP Error 400).

Privacy Policy Violation

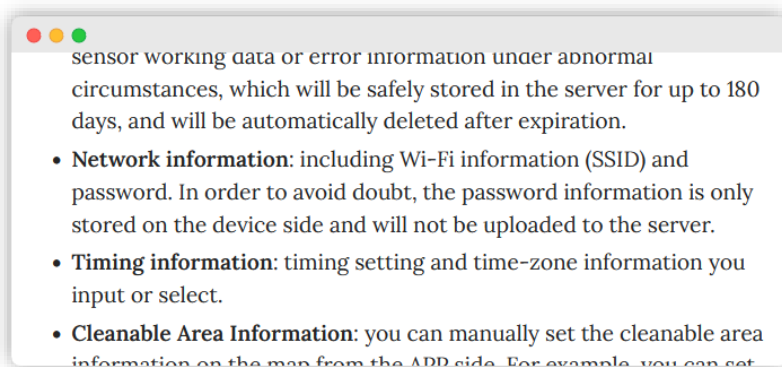


Figure 33 – Privacy policy excerpt

As shown in Figure 33, the privacy policy (effective 30th April 2019) for vacuum cleaner data in the Android version of Roborock’s smartphone application (app version 3.2.48) states that the “password [...] is only stored on the device” – however the screenshot below showing contents of the uploaded `rriot_tuya.log` file contradict the statement. Despite firmware version 02.29.02 being released 28th April 2022, the wireless network name and password are clearly visible within the log file.

It was noted that whilst a newer dated privacy policy (12th November 2021) was found on the vendor’s website²⁴, the privacy policy scope only addressed ‘Email Subscriptions’ and not of the privacy of data on, or of the vacuum cleaner.

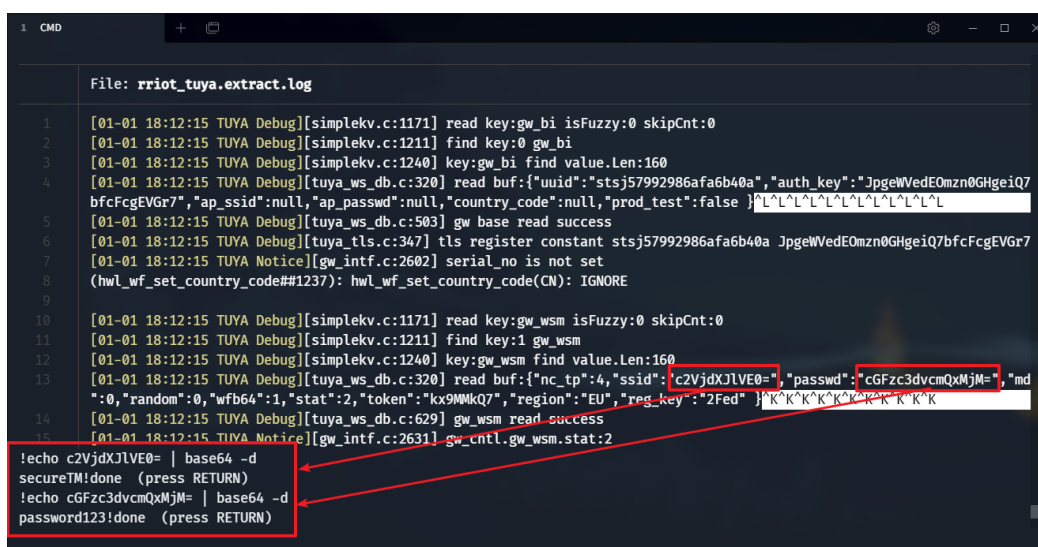


Figure 34 – Exposure of wireless credentials in `rriot_tuya.log`
(FW: v02.29.02)

²⁴ <https://global.roborock.com/pages/privacy-policy>

Pairing Traffic

When the Roborock S6 is uninitialised / factory-reset, the device enters Access Point mode, and broadcasts an SSID named *roborock-vacuum-s6_miapXXXX*, where *XXXX* is replaced with the last four characters of the device’s MAC address. The companion smartphone app will then connect to this access point and send the configuration frames to continue the pairing process. It was noted that the network was not secured with any passphrase, and consequently has no WEP / WPA security protecting transmissions. External parties can easily monitor the traffic of open networks, even without needing to join the network (given possession of a wireless adapter that supports promiscuous monitoring).

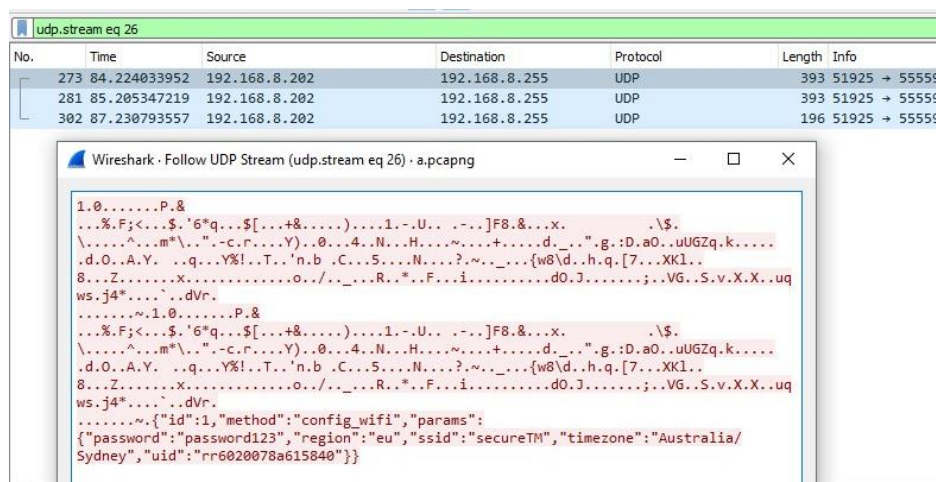


Figure 35 - Plain-text credential transmission during pairing

Network activity captured during the device pairing action revealed that the JSON-encoded configuration payload (containing the wireless credentials) was transmitted from the smartphone application to the robot over plain-text, as visible in Figure 35. Here, the SSID *secureTM*, and password *password123* are visible to anyone monitoring the network traffic. This observation of the plain-text transmission of wireless credentials violates the IoT ecosystem’s official security guidelines (Tuya Smart 2020), which outline the requirement for a product to “use AES encryption to transmit [...] Wi-Fi information”, and is synonymous with previous security and privacy studies on devices using the Tuya IoT ecosystem (Vtrust 2018).

Network Behaviour Analysis

Local Traffic

A high number of local traffic requests was observed being emitted by the DUT, albeit small in volume (< 3MB). The following local network behaviour was observed:

- Every 5 minutes a DHCP lease request was issued by the device
- Every 5 seconds the *rriot_tuya* process issued a *Tuya Discovery Packet*
 - Broadcast to *udp/6667* containing the device identifier and IP address

Specifically for firmware v01.15.58, the following additional behaviour was observed:

- Every 5 minutes an SSDP poll was issued by the device
 - This is an artifact of the operating system effectively running Ubuntu
- Every 10 seconds the *mio_client* process issued a request to *xx.ott.io.mi.com*
- Every second the *mio_client* process issued 2 requests to *xx.ot.io.mi.com*

External Traffic

The following traffic reports are based off network captures whilst the device was not in operation (idle) to determine network behaviour patterns. External traffic is broken down by endpoint to better characterise each individual process, and further broken down into hourly segments with times labelled in reference to Australian Eastern Standard Time (GMT+10).

Inbound Requests (ms.tuyaau.com / m2.tuyaau.com)

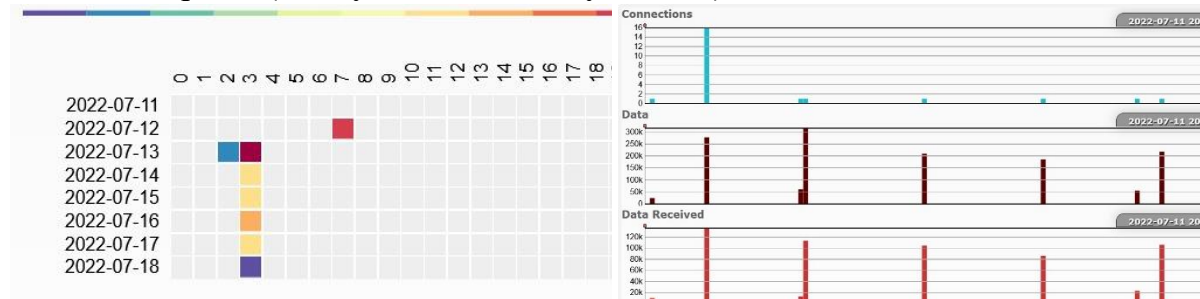


Figure 36 – MQTT server data heatmap

Figure 37 – MQTT server historical overview

The network heatmap above indicates increased network activity around 3am every day, however during these peaks, at most, only 300 KB of data was transferred. Application logs from `rriot_tuya` reveal that the increased activity is a result of the program timing out and reconnecting daily at 3am. A small packet was transmitted every minute; however, it was determined to be an MQTT keep-alive packet.

Outbound Requests (a2.tuyaau.com)

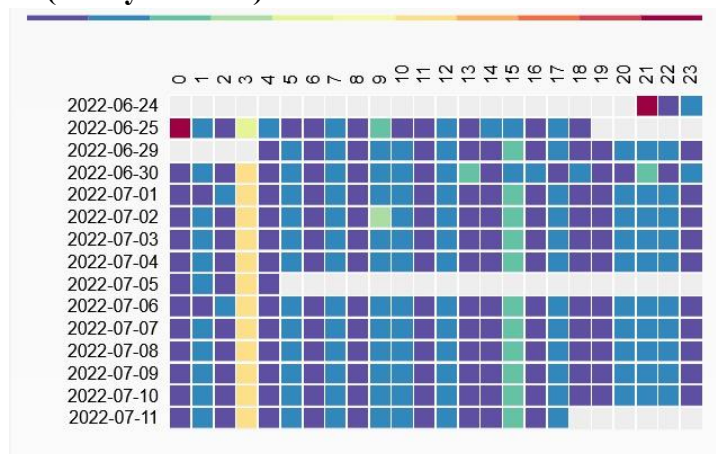


Figure 38 – Control server data heatmap

The `rriot_tuya` process exhibits more behaviour when communicating to the control server, evident in the increased dataflow counts in the figure above. Whilst increased dataflow is observed, total average hourly bandwidth does not exceed 10 KB, with peak hourly consumption of 20 KB at 3am every day. It was observed that a `tuya.device.timer.count` request was emitted every 25 minutes likely as an uptime poll, which aids in explaining the above heatmap. When the device reconnects to the MQTT server at 3am, upgrade checks and configuration polls are emitted, explaining the coincident activity.

Logs (awsde0.fds.api.xiaomi.com)

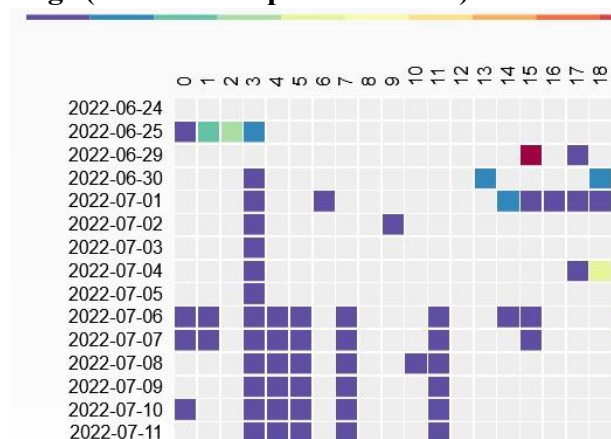


Figure 39 – FDS server data heatmap

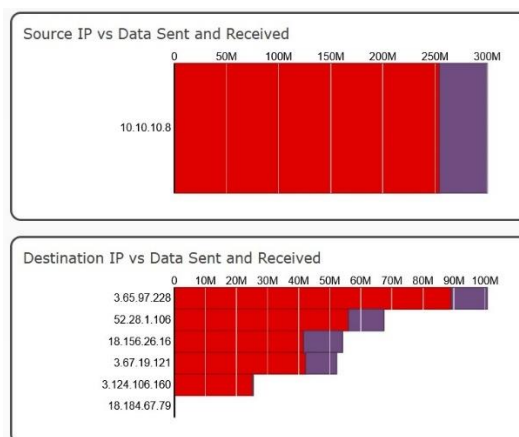


Figure 40 – FDS server flow graph

Inspection of the `rrlogd` process revealed that it did not transmit nor receive data from the FDS server unless logs were being uploaded. The behaviour of somewhat regular network activity (as visible in the 2022-07-06 to 2022-07-11 timeframe) can be attributed to the log sizes growing and reaching the threshold limit which triggers the logs to be uploaded. Likewise, when the MQTT connection is re-established, the increased log activity triggers logs to be uploaded, hence why all services incur increased activity at 3am. It was noted that the FDS servers which the device uploaded logs to were situated in Germany and the United States as visualised in Figure 41 below.

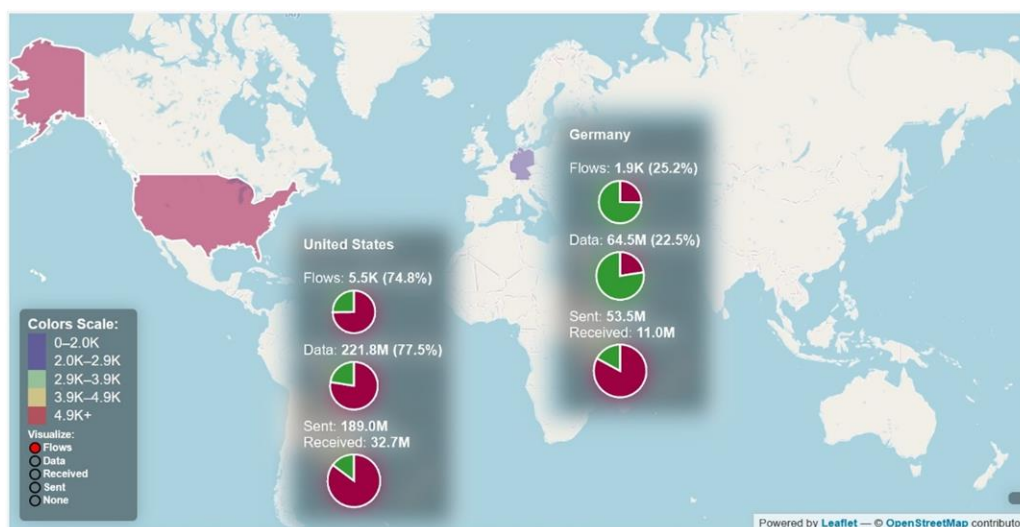


Figure 41 – Geomap of device activity to Xiaomi FDS servers

Device Docking

It was noted that network activity (both flow count and traffic volume) would increase when the vacuum returned to the charging dock after cleaning, or when manually docked. This was in accordance with a configuration parameter `ONLY_UPLOAD_ONDOCK=1` found in the `rrlog.conf` file.

Manufacturer Usage Description (RFC 8520)

How MUD works



Figure 42 – MUD usage diagram

Drafted in 2016, and published in 2019, the Manufacturer Usage Description (MUD) specification provided mechanisms for a networked device to advertise its expected network activity and behaviour. The supporting network infrastructure can then make use of this MUD profile (as outlined in Figure 42) to determine whether certain network traffic should be blocked or allowed at the switching level. For example – traffic emitted by a device to *example.com:8890/tcp* can be dropped if the device’s MUD profile does not contain a definition for traffic flow to *example.com* via *tcp/8890* – which can potentially mitigate foreign processes on a device from reaching out to the internet.

Whilst communication to distinct ports and hostnames can be controlled via RFC8520, there is no ability to perform deep packet inspection – payloads sharing the same connection cannot be differentiated. Consequently, this protocol can only be used to protect foreign and unidentified traffic connections and should not be relied upon to protect a network or device from all network threats (such as vendor C2, MITM and spoofing attempts).

As Roborock has not released MUD profiles for the Roborock S6, a set has been created (Hamza, Ranathunga et al. 2018) from the network traffic captured from firmware versions 01.15.58 and 02.29.02; and is publicly offered²⁵ to promote the adoption of RFC8520. An excerpt of the generated MUD profile is provided in Figure 43.

```

{
  "name": "from-ipv4-roboreck-s6",
  "type": "ipv4-acl-type",
  "aces": {
    "ace": [ {
      "name": "from-ipv4-roboreck-s6-0",
      "matches": {
        "ipv4": {
          "protocol": 6,
          "ietf-acl-dns:dst-dnsname": "a2.tuya.eu.com"
        },
        "tcp": {
          "destination-port": {
            "operator": "eq",
            "port": 443
          },
          "ietf-mud:direction-initiated": "from-device"
        }
      },
      "actions": {
        "forwarding": "accept"
      }
    } ]
  }
}

```

Figure 43 – MUD profile snippet (v02.29.02)

²⁵ <https://featherbear.cc/UNSW-CSE-Thesis/mud>

Device Entry and Persistence Analysis

We now detail methods to grant local access to, remote access to, or otherwise root the Roborock S6 vacuum cleaner to provide additional functionality and or capability. This section covers the practical methods and building blocks that a malicious actor may use, however discussions regarding security and privacy implications will be held until the chapter on Discussions.

Table 13 below is provided to summarise the proceeding content.

Table 13 – Overview of device entry and access methods

	Serial (UART)	USB (ADB)	SSH	OTA (MiIO)	Backdoor
Requires vinda		✓			
Requires Modifications			✓		✓
Requires Physical Access	✓	✓			
Fortified[†]	✓	✓	✓	✓	
Upgrade Resistant[#]		✓			

[†]Fortified: as describing if the vendor has implemented changes over the lifetime of the product to prevent or otherwise restrict access

[#]Upgrade resistant: as describing if an entry method will continue to work immediately after an upgrade is performed

Device Entry

Serial (UART)

The device’s serial console (see *Preliminary Device Access*) is likely the first point of entry to gain remote access. This method however requires physical access and disassembly of the device, as the UART pins are located on the device’s circuit board located within the device. Familiarity and the confidence to touch electrical circuits are also required, in addition to the possession a serial device interface (i.e. a USB to UART adapter).

In newer firmware versions where *rr_login* is used as the serial handler, additional work must be performed to gain root access, due to the *verify_shadow* authentication flow. After the initial connection however, the serial handler can be modified to use the old */bin/login* handler, which utilises the old Unix authentication method.

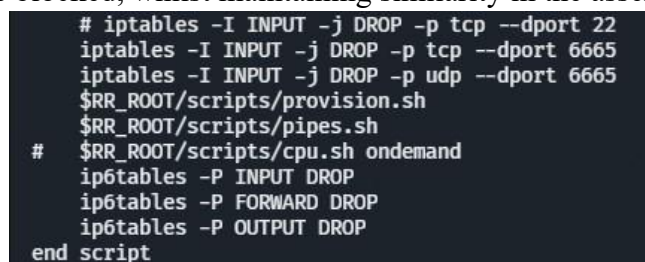
USB (ADB)

As mentioned previously (see *ADB*), access to the device via the ADB port is restricted due a custom authentication challenge, and further access restrictions even after authentication. Where the proposed novel exploit can be performed to remotely execute commands, an alternate method exists where the custom *adbd* is simply replaced with a fully functional version, bypassing all added authentication stages. The exploit method however remains resilient to upgrades (until patched by the vendor).

Whilst access to the ADB port is simple and quick (the micro USB port is located underneath the removable lid of the device, both methods (command injection, binary replacement) require prior access to the device – to either gain knowledge of the *vinda* content, or to access a shell.

SSH

In legacy firmware versions, the *rrwatchdog.conf* configuration file could be modified to nullify the offending *iptables* command, as shown in Figure 44. However, in newer firmware versions where the modified *dropbear* SSH server is used, the *iptables* drop command is present in multiple locations (*S04wdgenv*, *WatchDoge*, *rrwatchdog.conf*, *rrlogd*) and consequently each file must be patched to permit SSH access. In patching the *WatchDoge* and *rrlogd* binaries, calls can be simply nullified by replacing the instructions with *NOP* instructions, or by replacing the string ‘22’ with a spurious value like ‘27’, as to cause the wrong TCP port to be blocked, whilst maintaining similarity in the assembly code execution.



```
# iptables -I INPUT -j DROP -p tcp --dport 22
iptables -I INPUT -j DROP -p tcp --dport 6665
iptables -I INPUT -j DROP -p udp --dport 6665
$RR_ROOT/scripts/provision.sh
$RR_ROOT/scripts/pipes.sh
# $RR_ROOT/scripts/cpu.sh ondemand
ip6tables -P INPUT DROP
ip6tables -P FORWARD DROP
ip6tables -P OUTPUT DROP
end script
```

Figure 44 – *rrwatchdog.conf* with SSH access patch

Access to the SSH server is heavily reliant on the ability to manipulate the filesystem, and hence requires prior access to the device. It is beneficial to replace the *dropbear* binary with an *OpenSSH* server implementation, to allow non-root user authentication whilst supporting more secure key exchange algorithms (see *SSH Access*) and possibly supporting file transfers via SFTP. It is also worthwhile to create an additional user on the device, as to provide an alternate backup login account.

OTA (MiIO)

Prior to November 2019, Roborock S6 devices supported over-the-air firmware upgrades via the MiIO protocol²⁶, where a packet could be transmitted to the device containing instructions to upgrade the firmware, as visualised in Figure 45. The device would then fetch the firmware and execute the associated setup scripts. The ability to control the firmware URL to a user-provided package provided the potential to remotely root, or otherwise gain control over the device without requiring physical access and/or the disassembly of the device.

The MiIO OTA rooting method has limited use as only devices manufactured within four months of the product’s release (June 2019) were supported. Consequently, this method was not applicable to the DUT, as it was manufactured after the method was disabled, however a downgrade of the *miio_client* and *SysUpdate* binaries confirmed the past exploitability of this method (using *miio_client* version 3.3.9). Figure 46 visualises the assembly code graph of *miio_client* version 3.5.4, where modifications were made to discard the *miIO.ota* payload and cause the process to follow the silent fail path in red.

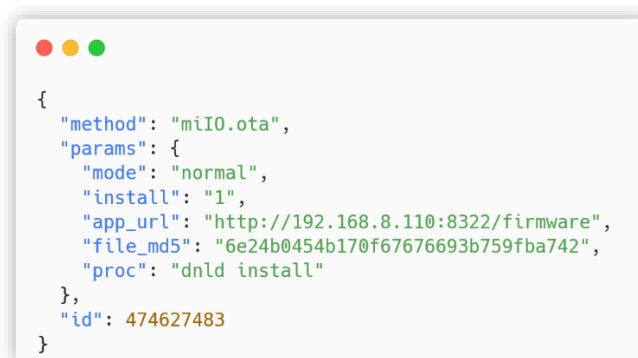


Figure 45 – MiIO OTA payload

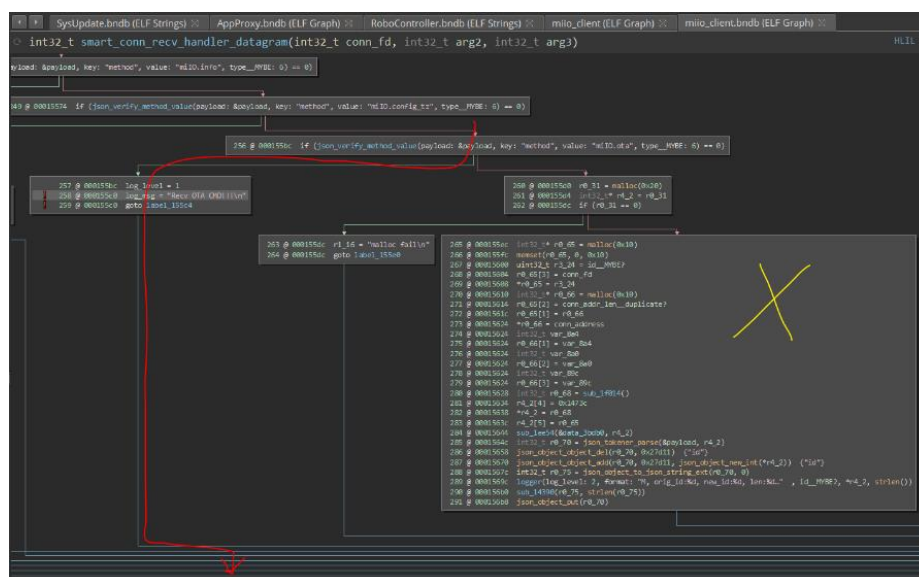


Figure 46 – Silent fail of the miIO.ota payload

²⁶ <https://github.com/marcelrv/XiaomiRobotVacuumProtocol/blob/master/miIO-ota.md>

Persistence

Remote Access Persistence (Backdoor)

In benefit of the device’s network stack capability, a virtual private network (VPN) utility or software defined network (SDN) tool such as *ZeroTier*²⁷ can be installed, allowing remote communication with the device through standard IP networking, gaining access to local device services such as SSH. A proof of concept has been made available²⁸.

Other remote access methods such as a reverse shell can also be established given the freedom of software and hardware support. Figure 47 below provides insights into the ability to create private ad-hoc networks despite a remote network topology.

It is worthwhile to note that the implementation of the RFC8520 protocol (see *Manufacturer Usage Description*) would aid in preventing these remote access methods from working.

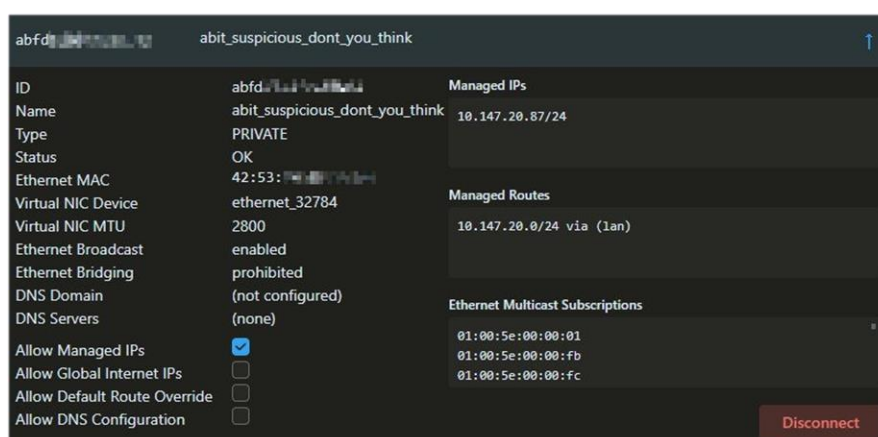


Figure 47 – ZeroTier control panel

²⁷ <https://www.zerotier.com>

²⁸ <https://featherbear.cc/UNSW-CSE-Thesis/poc/>

Reset Persistence

During the factory reset procedure (reset pin or boot failure), the *recovery* partition is flashed onto both *system_a* and *system_b* partitions. As this partition is modifiable, changes to the partition will be propagated to the system partitions after a factory reset. Modifications can therefore be performed by mounting the *recovery* partition while the system is live. A proof of concept has been made available²⁹.

Candidate changes may include enabling SSH access, adding a backdoor user, remote access persistence patches, and the storage of additional tools like *wget*, *curl*, *gdb*, and *strace*.

Upgrade Persistence

During the firmware upgrade procedure, the system is updated in the following manner:

1. Download the update to *UDISK*
2. Extract update to *download*
3. Unmount *system_a* / *system_b*
4. Flash download to *system_a* / *system_b*
5. Boot into *system_a* / *system_b*
6. Flash download to *system_b* / *system_a*
7. Boot into *system_a* / *system_b*

Notably, both *system_a* and *system_b* partitions are completely overwritten, which would discard any changes or patches made on the device, inclusive of all previously stated methods.

A new method is proposed to achieve upgrade persistence, allowing modifications and other rooting artifacts to persist between upgrades. By hooking into the post-extraction stage of the firmware upgrade process (after step 2) and manipulating the contents of the *download* partition, upgrade-persistent changes can be performed – however this procedure is time-sensitive as the interception must occur between the start of the firmware extraction process (step 2) and the start of the image flashing process (step 4).

It is noted that interception tasks should complete as fast as possible, to best ensure that all changes will propagate to the system partitions during flashing. Where multiple changes are desired to be retained between updates, large sized files and time-bound functions can be offloaded onto the *recovery* or *reserve* partitions (see *Data Persistence*), where they can be processed during the runtime of the upgraded system. This offloading technique can dramatically decrease the number of required steps to perform upgrade persistence patching to a single step (i.e. creating a boot entry-point that calls the offloaded scripts)

Various techniques can be used to write to the *download* partition, such as a service worker, crontab or scheduled task – however one must be mindful of incurred CPU load should the technique incur a ‘busy-wait’. Binary patching of the *SysUpdate* could be performed as an alternative to a timed task and guarantee successful modification, however this method is complicated and likely prone to failure should the vendor perform unexpected updates to the binary. It should also be noted that upgrade persistence patching techniques must also handle future firmware upgrades, and thus must self-replicate its functionality.

²⁹ <https://featherbear.cc/UNSW-CSE-Thesis/poc/>

Chapter 6 | Discussion

Commentary

In discussing the opinions on how manufacturers of IoT / smart home device have addressed the increasing concerns of digital privacy and product security, we comment on the collected results and findings, with reference to the threat scenarios previously defined in Chapter 4.

For ease of access, a summary of the threat scenario is provided below:

- TS0 – Concerned with the visibility and ownership of data and the device
- TS1 – Concerned with physical (proximal) threats e.g. supply chains, physical access
- TS2 – Concerned with remote (proximal) threats e.g. monitoring, device takeover
- TS3 – Concerned with remote (distal) threats e.g. backdoors, cloud services, the vendor

System Design

Regarding the embedded system design, Roborock’s decision to strip down their original Ubuntu Core based system to a more standard embedded Linux system significantly reduces the attack surface that may be exploited in scenario *TS2*, whereby the reduction in running processes consequently generate less network traffic that can be observed by an actor monitoring the network. Regarding *TS1* and *TS3*, the reduced set of available software and the omission of a package manager on the device additionally increases the effort required to sideload and run potentially malicious applications.

It is however emphasised that malicious intent is not prevented, but only slowed down. As such, in the case of *TS1* where periods extended access is possible (such as a supply chain attacks, or a malicious reseller), the device can still be modified to plant remote access persistence and grant an actor remote access even after possession of the device has been released.

Process Privilege Level

Regarding the privilege level of all processes running under the root account, security concerns are naturally raised for *TS1*, *TS2* and *TS3*, whereby a single vulnerability in any root-owned process can lead to system takeover. This threat is most prominent in *TS3*, as a vulnerability that lies in the cloud service communications would provide the means for an actor to gain control over any affected device over the internet. It is again noted that embedded system applications often run with system-level access (i.e. root) or in a root-less environment where all processes are effectively elevated – as it often mitigates hardware integration issues. Extreme care must therefore be taken when developing and securing such programs, however such attention to detail is difficult.

Device Access

Efforts to restrict access to the device via the serial terminal, ADB port and SSH shell are largely beneficial, as the restrictions significantly impede the ability to gain control of the device. In the case of *TS1* – ADB communications are restricted because of Roborock’s custom access control implementations and requires knowledge of a device-specific secret that can only be retrieved through tedious disassembly of the device, which is also required to access the UART pins that serve the serial terminal.

The long period of time required to disassemble, modify and reassemble the device severely decreases the capability for an actor to perform the modification over a number of devices. Large-scale modifications through supply chain attacks are only profitable between the stages of the flashing of the eMMC storage and the assembly of the device.

In the case of *TS2*, access to the device via SSH is prevented due to *iptables* rules. In later versions of the firmware, this restriction is enforced as observed through additional calls *iptables* from the *WatchDoge* and *rrlogd* binaries. Should the server be accessible for some reason, knowledge of the root password is still required which is unobtainable remotely.

It is noted that in the case of *TS0*, the security fortifications serve as hindrance to a device owner wanting to study or tinker with the device. The inability to use the ADB port and SSH server force an owner to disassemble the device and establish a UART connection, which may likely be out of technical ability for many owners who purchase this robot vacuum cleaner. Authentication flow modifications may potentially completely break access functionality for devices with outdated filesystem layouts, as was experienced with our device under test.

Modifiable Recovery Partition

Regarding the ability to modify the *recovery* partition, whilst useful under *TS0* (i.e. as a hardware hobbyist) to store software tools and maintain access between factory resets, security concerns are raised in the case of *TS1* – where the ability to modify the *recovery* partition raises the concern of backdoors being planted during the supply chain, or from a previous owner. Backdoors could eventually lead to the compromise of owner’s data, and of the owner’s network to which the targeted device is connected to. It is unlikely that an actor with only momentary physical access to the device will be able to exploit the reset persistence, due to the time required to disassemble and reassemble the device.

It is recommended that the *recovery* partition should be marked as ‘permanent read-only’ (Western Digital 2017) on the hardware eMMC level, as there is no need for the partition to be modified once the initial recovery image is flashed. Hardware write-protection provides the best method of data integrity as software-level write protection controls can be subverted (such as removing the ‘*ro*’ parameter from the Linux mount options).

Should the partition need to be modifiable for some reason, provisions to verify the authenticity of the filesystem should be enforced, such as some sort of signature verification or asymmetric encryption. Hardware security features like RPMB (Replay-Protected Memory Block) could serve to be beneficial, where writes to the storage medium must be paired with an authentication key that could be stored in an SE (Secure Element).

Data Retention

Whilst data in the *UDISK* (user data) partition is cleared securely during a factory reset, it is not cleared during a disassociation event (when the device is removed from a user’s account), despite the device acknowledging the event and entering access point mode. Under *TS0* and *TS1*, privacy concerns are raised – as an actor in possession of a recently disassociated device may be able to extract UGC and PII, inclusive of LIDAR mapping data, network credentials and network dumps. It would be advised for future firmware updates to delete and effectively factory reset the device when device disassociation is performed.

Pairing Security

The plain-text transmission of wireless credentials during initial device pairing raises concern for *TS2*, as anyone nearby who is monitoring wireless traffic will be able to eavesdrop and intercept the wireless credentials. The respective wireless network could then be joined using the intercepted credentials, allowing further access and enumeration into a victim’s network. Alarming, as there is no passphrase for the pairing access point – the wireless credentials within the pair request payload can be intercepted without even requiring the actor to join the same network, due to the behaviour of network traffic in an ‘Open’ wireless network.

It is recommended that the wireless network broadcasted during the device’s access point mode be secured with some wireless network security protocol, such as WPA2. Furthermore the pairing request should be encrypted, as stated in the Tuya security guidelines (2020). Whilst this specific privacy and security concern is only applicable during the pairing of the device, assumptions should not be made regarding the likelihood nor presence of a malicious actor nearby.

Encryption of Logs

Regarding the confidentiality of transmitted data, logs remain secure against *TS2*, even when the device is placed in an adversarial network condition such as a MITM proxy, or where TLS / SSL decryption is present – as logs are encrypted on the application-level. Whilst possible to visualise the flow of data and knowledge of network activity, the underlying data is ultimately protected with no way to view the decrypted contents without knowledge of the private key.

The application-encrypted logs are however ineffective against *TS1* and *TS3*, as the log sources are simply located within the filesystem. Access to any means to read the contents of files (whether it be serial, SSH, SFTP, ADB, reverse SSH or similar backdoor) will result in the ability to access log files before they are encrypted. In the case of the vendor, they possess the private decryption key, and hence will have unfiltered access to decrypt the log data.

In the case of *TS0*, the encryption of logs (and other transmitted data) is similarly trivial to a user wanting to view the nature and content of network traffic; however sufficient skill and technical knowledge is required to navigate a Linux filesystem, and optionally manually patch or dynamically instrument a process to hook into the pre-encryption or post-decryption stages.

Packet Logging

With the added implementation of the `MSG_LOG_DEBUG_*` signal in `rrlogd`, and the ability to perform a `tcpdump` packet capture in `wlanmgr`, privacy and security concerns are raised under *TS3*, as the activity of other devices on the same wireless network can be captured. As residential network traffic is often large and verbose, a great amount of flow detail can be obtained regarding the access to websites, frequency of website visits, the number of devices on a network, the types of devices on a network, and information about the network (such as the IP address). Whilst geographical lookups of IP address are often inaccurate, knowledge of the approximate region that the targeted device is in may aid in further exploitation.

Privacy of Uploaded Data

Concerns surrounding *TS0* and *TS3* are raised regarding the use of, and necessity to upload all the files from *rrlogd*. Whilst mapping data and packet logging data (as previously discussed) may be beneficial to the vendor, as to improve the cleaning functionality of the product, great trust must be placed in believing that the data is not misused or abused.

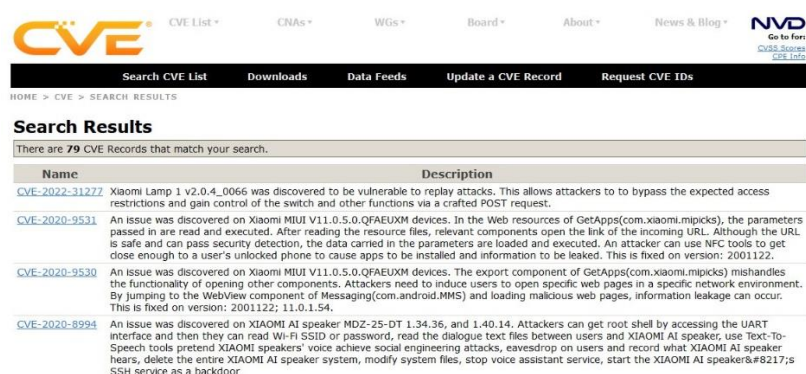
As discovered, log data was found to contain wireless credentials - despite the privacy policy stating that data of that type would not be kept remotely. Consequently, the need to verify company statements against their actions is stressed, with better transparency (and somehow confirmation) regarding the use of data. The flexibility to control the type of collected data is also desired, where privacy-minded owners can choose to opt-out / opt-in of certain log types.

Response of Other Manufacturers

It is noted that whilst the Roborock S6 vacuum cleaner faces several privacy and security concerns, the company has made considerable effort to fortify their product against potential malicious threats. Privacy optimisations such as decreased log verbosity and application-level encryption was observed. Likewise, security fortifications such as overflow detection, signature verification and access control restrictions were noted. Additional steps can be taken by Roborock as a company however, to further improve their digital privacy and product security.

We turn to Xiaomi and Tuya for comparison, to investigate how other companies have addressed the increasing concerns for digital privacy and product security. Both Xiaomi and Tuya are large IoT ecosystem vendors who lease their infrastructure to white-label vendors and OEMs (like Roborock) as a subscription. Naturally, these ecosystem vendors are much larger in employee count, as both Xiaomi and Tuya have their own multi-staffed security teams.

As a result of their larger business (in both popularity, profit and employee count), these companies publicly promote the security research of their products and offer a bug bounty to incentivise research. Consequently, a high number of security vulnerabilities are discovered (as illustrated in Figure 48), allowing these companies to constantly issue security patches and fixes to better protect their products.



The screenshot shows the CVE website interface. At the top, there are navigation links: CVE List, CNAs, WGs, Board, About, News & Blog, and NVD. Below these is a search bar and a navigation menu with links: Search CVE List, Downloads, Data Feeds, Update a CVE Record, and Request CVE IDs. The main content area is titled 'Search Results' and indicates 'There are 79 CVE Records that match your search.' Below this is a table with two columns: 'Name' and 'Description'. The table lists four CVEs related to Xiaomi:

Name	Description
CVE-2022-31277	Xiaomi Lamp 1 v2.0.4_0066 was discovered to be vulnerable to replay attacks. This allows attackers to bypass the expected access restrictions and gain control of the switch and other functions via a crafted POST request.
CVE-2020-9531	An issue was discovered on Xiaomi MIUI V11.0.5.0.QFAEUXM devices. In the Web resources of GetApps(com.xiaomi.mipicks), the parameters passed in are read and executed. After reading the resource files, relevant components open the link of the incoming URL. Although the URL is safe and can pass security detection, the data carried in the parameters are loaded and executed. An attacker can use NFC tools to get close enough to a user's unlocked phone to cause apps to be installed and information to be leaked. This is fixed on version: 2001122.
CVE-2020-9530	An issue was discovered on Xiaomi MIUI V11.0.5.0.QFAEUXM devices. The export component of GetApps(com.xiaomi.mipicks) mishandles the functionality of opening other components. Attackers need to induce users to open specific web pages in a specific network environment. By jumping to the WebView component of Messaging(com.android.messaging) and loading malicious web pages, information leakage can occur. This is fixed on version: 2001122; 11.0.1.54.
CVE-2020-8994	An issue was discovered on XIAOMI AI speaker MDZ-25-DT 1.34.36, and 1.40.14. Attackers can get root shell by accessing the UART interface and then they can read Wi-Fi SSID or password, read the dialogue text files between users and XIAOMI AI speaker, use Text-To-Speech tools pretend XIAOMI speakers' voice achieve social engineering attacks, eavesdrop on users and record what XIAOMI AI speaker hears, delete the entire XIAOMI AI speaker system, modify system files, stop voice assistant service, start the XIAOMI AI speaker's #8217;s SSH service as a backdoor.

Figure 48 – Screenshot of CVEs associated with Xiaomi

Whilst it is noted that Roborock is a small company, it is intriguing to see that only one security vulnerability was disclosed despite having released 13 different products since the company's inception in 2014. Whilst it is not mandatory for a company to publicly disclose their security vulnerabilities, it can conversely negatively illustrate the company's security posture, as customers may be led to believe that the company is hiding its issues.

Both Xiaomi’s³⁰ and Tuya’s³¹ security teams have additionally released white-papers regarding security minimums and guidelines for products that utilise their infrastructure. Despite the Roborock S6 vacuum cleaner utilising the Tuya infrastructure (through `rriot_tuya`), the failure to encrypt the pairing traffic as outlined in the security guidelines raise concern to whether Tuya (and other IoT ecosystem vendors) perform security compliance checks on their white-label partners and OEMs before allowing a partner product to be verified and released.

We end our discussion involving other manufacturers by commenting on the adoption of the Manufacturer Usage Description protocol (RFC 8520), or rather why it hasn’t been widely adopted within the IoT and networking industry. Currently RFC 8520 does not seem to be adopted by any large IoT vendor, nor network equipment manufacturer. Despite Cisco spearheading the push for the use of MUD³², only their Catalyst line of network switches support the ‘Network Access Device’ role used in the MUD process.

It is likely that there is no incentive for IoT vendors to release MUD profiles of their devices, nor is there an incentive for networking equipment manufacturers to support RFC 8520 as there is no recent sign of activate development on any MUD-related projects. Furthermore, whilst UNSW’s Internet of Thing Research Group (EE&T)³³ has contributed multiple MUD profiles for a variety of IoT devices, no other shared repository of MUD profiles exist – which further discourages companies from investing time and effort into implementing the specification. MUD profiles for the Roborock S6 were generated and are publicly available in the hopes of supporting the widespread adoption of RFC 8502.

Conclusion

Through the firmware and network analysis from multiple firmware versions of the Roborock S6 vacuum cleaner, we conclude that Roborock has made efforts to secure their products and respect the privacy of their customers. Specifically, firmware upgrades incorporated changes to the ways in which device authentication and remote access was established, as to increase the difficulty and time required for local and remote threats to gain access to the device. Furthermore, in response to adversarial network conditions where a wireless network may be insecure, the confidentiality of transmitted log data was kept through application-level encryption that would remain given the presence of TLS / SSL decryption.

In the context of IoT and smart home device manufacturers in general, a ‘shift-left’ mentality to security research is encouraged – evident in the Xiaomi and Tuya each releasing security papers and offering bug bounty programs to promote the disclosure and reporting of vulnerabilities, as to patch and better protect their products.

Whilst these companies have made significant improvements to their product’s digital privacy and product security, further work is required to address the vulnerabilities and concerns raised in this thesis. Notably, IoT ecosystem vendors like Tuya need to perform (or improve) compliance checking procedures to ensure that their white-label vendors and OEM clients are in accordance with the security policies they released. Product vendors need to additionally review and verify their own policies (i.e. privacy policy) to ensure that their products are in

³⁰ <https://github.com/MiSecurity/Cyber-Security-Baseline-for-Consumer-Internet-of-Things>

³¹ <https://images.tuyacn.com/smart/docs/TuyaSmart-WhitePaper-Intl.pdf>

³² <https://developer.cisco.com/docs/mud/#!/what-is-mud/what-is-mud>

³³ <https://iotanalytics.unsw.edu.au/mudprofiles>

accordance with their own policies even during upgrades, and that data is securely deleted in all expected scenarios. Furthermore, selective control over the nature and type of collected data should be given to the user. Modifications to a device’s storage should additionally be locked down, where write-access to base data and recovery firmware should be restricted unless mandatory.

In summary, manufacturers of IoT / smart home devices have addressed to privacy and security concerns by reacting with positive improvements and fixes, however better care must be taken to wholistically improve their privacy and security posture. The contributions offered from this thesis sought to critically analyse the security and privacy of the device, as to provide suggestions to ultimately improve the state of the art of IoT security and privacy research.

Limitations and Future Work

Whilst a substantial amount of work was performed during this thesis, it is reiterated that the DUT used in the study was manufactured in June 2020, one year after the official release of the Roborock S6 vacuum cleaner in 2019. Consequently, the scope of security and privacy assessment is limited to firmware versions from 2020 – 2022, notably failing to support the MiIO OTA exploit procedure present early firmware versions. Furthermore, the DUT’s manufacture month coincides date of the firmware version which used the migrated authentication flow, providing uncertainty as to whether later manufactured devices contained a modified filesystem structure that included files missing from the DUT’s filesystem.

It is also worth mentioning that the results collected and observations made from this study may differ from future replication studies that assess the same Roborock S6 device, as variances in the product’s region setting and cloud provider (Tuya in our case) may generate different network traffic and device behaviour.

Further topics of research are presented below which were beyond the scope of the study, or otherwise follow the study.

- Resilience of applications towards MITM and HSTS certificate pinning bypasses
- Security and strength of the asymmetric keys used
- Security and/or privacy assessment of the smartphone application
- Security and/or privacy assessment of the STM32 co-processor
- Side-channel analysis of sensor data
- Fuzzing of program binaries on the device to find further vulnerabilities
- Dynamic instrumentation of program binaries using Frida
- Upgrade analysis of new firmware versions after v02.29.02
- Analysis of other Tuya-integrated vacuum cleaners to find similar vulnerabilities
- Detailed study of the adoption of RFC 8520

Bibliography

Abrams, L. (2021) 533 million Facebook users’ phone numbers leaked on hacker forum.

Adrian, D., et al. (2015). Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. 22nd ACM Conference on Computer and Communications Security.

Brown, R. (2015) Smart homes can pay off when it's time to sell.

Costin, A., et al. (2014). A Large-Scale Analysis of the Security of Embedded Firmwares, USENIX Association: 95--110.

Giese, D. (2019). Security Analysis of the Xiaomi IoT Ecosystem.

Giese, D. (2021). Smart Home Security & Privacy.

Hamza, A., et al. (2018). Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles. Budapest, Hungary, ACM Sigcomm workshop on IoT Security and Privacy.

Jimenez, J. C. (2016, 2016-06-08). "Practical Reverse Engineering - Dumping the Flash." from <https://jcjc-dev.com/2016/06/08/reversing-huawei-4-dumping-flash/> ,.

Jmaxxz (2016). Backdooring the Frontdoor, DEF CON.

Jones, R. (2017) Roomba's Next Big Step Is Selling Maps of Your Home to the Highest Bidder.

Kotlyar, E. (2017). Xiaomi DaFang Hacks.

Miralem, M., et al. (2019). About the Connectivity of Xiaomi Internet-of-Things Smart Home Devices. 2019 XXVII International Conference on Information, Communication and Automation Technologies (ICAT): 1-6.

Mishra, B. and A. Kertesz (2020). "The Use of MQTT in M2M and IoT Systems: A Survey." IEEE Access **8**: 201071-201086.

Research & Markets (2021) Insights on the Smart Homes Global Market to 2026 - Featuring ABB, Acuity Brands and Emerson Electric Among Others.

Sriram, S., et al. (2020). LidarPhone: Acoustic Eavesdropping Using a Lidar Sensor: Poster Abstract. Proceedings of the 18th Conference on Embedded Networked Sensor Systems, Association for Computing Machinery: 701–702 , numpages = 702.

Tuya Smart (2020). Tuya Smart White Paper on Information Security & Compliance.

Ullrich, F., et al. (2019). Vacuums in the Cloud: Analyzing Security in a Hardened IoT Ecosystem, USENIX Association.

Vtrust, M. S. (2018). Smart Home - Smart Hack - Wie der Weg ins digitale Zuhause zum Spaziergang wird.

Wei, T., et al. (2015). Acoustic Eavesdropping through Wireless Vibrometry. Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, Association for Computing Machinery: 130–141 , numpages = 112.

Western Digital (2017). A detailed overview of the different security methods one can use in an eMMC storage device.

Appendix

Appendix 1 – rrlogd log scope excerpt (v01.15.58)

Archive	Contents
varlog.tar.gz (tar_extra_file.sh)	/var/log/upstart
	/var/log/boot.log
	/var/log/bootdmesg
	/var/log/dmesg
	/var/log/faillog
	/var/log/kern.log
	/var/log/lastlog
	/var/log/rr_try_mount.log
	/var/log/syslog
misc.gz (misc.sh)	date
	/dev/jiffies
	/proc/interrupts
	/proc/softirqs
	dmesg
	/proc/meminfo
	/proc/vmstat
	/proc/slabinfo
	/proc/zoneinfo
	/proc/pagetypeinfo
	/sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state
	/sys/devices/system/cpu/cpu1/cpufreq/stats/time_in_state
	/sys/devices/system/cpu/cpu2/cpufreq/stats/time_in_state
	/sys/devices/system/cpu/cpu3/cpufreq/stats/time_in_state
	df -h
	lsuf /
	lsuf /dev
	lsuf /tmp
	lsuf /run
	lsuf /run/lock

	lsof /run/shm
	lsof /mnt/updbuf
	lsof /mnt/data
	lsof /mnt/reserve
	lsof /mnt/default
	/sys/devices/platform/uart.0/ctrl_info
	/sys/devices/platform/uart.0/status
	/sys/devices/platform/uart.1/ctrl_info
	/sys/devices/platform/uart.1/status
	/sys/devices/platform/uart.2/ctrl_info
	/sys/devices/platform/uart.2/status
watchdog.gz	watchdog.log
rrlog.gz	rrlog.log
miio.gz	miio.log
SLAMMAP.tar.gz	*.ppm
SYSUPD_normal_updater.tar.gz	SYSUPD_updater_pid*.log
varlog.tar.gz	varlog.tar.gz (itself)
mt_test.tar.ss.gz	/mnt/data/rockrobo/Mt*
	/mnt/data/rockrobo/mt*
uarttest.tar.ss.gz	/mnt/data/rockrobo/noupload/uart_test*
boot_reason.gz	boot_reason
crashlog.gz	crashlog

Appendix 2 - rr_login authentication loop (v02.29.02)

