Abstract

What was done, what was found, what are the main conclucions

Keywords:

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 did 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](#_Toc110817845)

[Chapter 2 | A Background on IoT 2](#_Toc110817846)

[The Call to Action 2](#_Toc110817847)

[About Roborock 2](#_Toc110817848)

[Chapter 3 | Current State of the Art 3](#_Toc110817849)

[Broad security study of Tuya-based devices 3](#_Toc110817850)

[Broad security study of Xiaomi-based devices 4](#_Toc110817851)

[Security study of smartphone applications 4](#_Toc110817852)

[Analysis of similarities in IoT firmwares 5](#_Toc110817853)

[Side-channel application of LIDAR sensor measurements 5](#_Toc110817854)

[Shell access via sideloaded media 6](#_Toc110817855)

[Shell access via BGA pin shorting 6](#_Toc110817856)

[Hardware based extraction of flash memory 7](#_Toc110817857)

[Cold-boot attack to dump memory state 7](#_Toc110817858)

[Chapter 4 | Threat Modelling 8](#_Toc110817859)

[Chapter 5 | Work Performed 9](#_Toc110817860)

[Scope and Summary of Work 9](#_Toc110817861)

[Preliminary Device Access 10](#_Toc110817862)

[Dynamic Firmware Analysis 11](#_Toc110817863)

[Static Firmware Analysis 19](#_Toc110817864)

[Network Activity Analysis 27](#_Toc110817865)

[OTA Rooting 34](#_Toc110817866)

[Device Entry Analysis 34](#_Toc110817867)

[Chapter 6 | Discussion 36](#_Toc110817868)

[Summary of Contributions 36](#_Toc110817869)

[Commentary 36](#_Toc110817870)

[Future Work 37](#_Toc110817871)

[Bibliography 38](#_Toc110817872)

List of Figures

[Figure 1 - UART pin locations 10](#_Toc110817698)

[Figure 2 - Password decryption of the vinda file 10](#_Toc110817699)

[Figure 3 - Process list (v01.15.58) 11](#_Toc110817700)

[Figure 4 - netstat (v01.15.58) 12](#_Toc110817701)

[Figure 5 - iptables (v01.15.58) 12](#_Toc110817702)

[Figure 6 - ip6tables (v01.15.58) 13](#_Toc110817703)

[Figure 7 - ifconfig (v01.15.58) 13](#_Toc110817704)

[Figure 8 - /etc/passwd (v01.15.58) 14](#_Toc110817705)

[Figure 9 - /etc/shadow (v01.15.58) 14](#_Toc110817706)

[Figure 10 - Serial log during factory reset 15](#_Toc110817707)

[Figure 11 - Exposed underside of the charging dock 17](#_Toc110817708)

[Figure 12 - 2-wire battery shutdown log 17](#_Toc110817709)

[Figure 13 - Firmware dump commands 19](#_Toc110817710)

[Figure 14 - Screenshot of the Commentree tool 20](#_Toc110817711)

[Figure 15 - Isolated network connection diagram 27](#_Toc110817712)

[Figure 16 - Crypto function hook source code 28](#_Toc110817713)

[Figure 17 - Plain-text credential transmission during pairing 33](#_Toc110817714)

List of Tables

[Table 1 - Threat Model Matrix 8](#_Toc110817725)

[Table 2 - Root password extraction 10](#_Toc110817726)

[Table 3 - Partition Mapping 19](#_Toc110817727)

[Table 4 - Network equipment list 27](#_Toc110817728)

Table of Abbreviations

|  |  |  |
| --- | --- | --- |
|  | **Expansion** | **Meaning** |
| **DUT** | Device Under Test | Relating to a device being tested |
| **eMMC** | Embedded Multimedia Card | Onboard storage |
| **IoT** | Internet of Things | Classification of network-connected devices |
| **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 |
| **NIC** | Network Interface Card | Hardware to connect a device to a network |
| **PII** | Personal Identifiable Information | Data that could identify an individual |
| **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 |
| **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 demands for such devices 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. Giving 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 if the former were to have 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 the general populus lies 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 lies evident from various incidents that we cannot completely trust such claims. From involuntary exposure of 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 and high return on investments. 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 vulnerabilities[[1]](#footnote-1).

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. We then review existing research and methods that comprise the current state of the art. Finally, we detail the contributions from this thesis and discuss conclusions.

# 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.

Concerns are raised 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. This concern is potentially serious, as vulnerabilities within an IoT infrastructure would imply that devices and data from customers of vendor products would too be vulnerable. Furthermore, the lifetime of a vendor business is not guaranteed. With the constant opening and sunsetting of IoT vendors, the closure of the business from which an IoT product was purchased from might result in the eventual inoperability of the said device.

In the event that an IoT infrastructure suffers downtime or service instability, all white-labelled products too will also be inherently 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 Roborock

Roborock is a Chinese company founded in Beijing that develops robotic cleaning appliances for households. In 2014, partnering with Xiaomi 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 - such as the addition of mopping functionality and improved spatial object detection via LIDAR technologies.

In June 2019, Roborock released their flagship Roborock S6 vacuum cleaner, boasting in its reduced operating noise levels and improved cleaning performance. Featuring an Allwinner R16 SoC (ARM architecture), it is powered by either the Tuya Smart or Xiaomi Cloud infrastructure, both being 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.

# 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 from 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 are raised regarding the ease of possibly 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[[2]](#footnote-2)*, Vtrust was able to exploit discovered vulnerabilities on multiple products to perform over-the-air upgrades of custom firmware (e.g. [*ESPhome*](https://esphome.io/), [*Tasmota*](https://tasmota.github.io/docs/)). An automated flashing tool (*[tuya-convert](https://github.com/ct-Open-Source/tuya-convert)*) was released, allowing consumers to easily integrate these devices with local home automation software such as [*HomeAssistant*](https://www.home-assistant.io/). As a result of Vtrust’s findings, the overall security posture of modern Tuya-powered devices has since improved[[3]](#footnote-3), 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 will perform network security assessments to identify if similar findings are discovered regarding weak network security.

## 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*](https://github.com/dgiese/dustcloud)*[[4]](#footnote-4)* 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*](https://github.com/Hypfer/Valetudo).

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 tools[[5]](#footnote-5), per-application patching[[6]](#footnote-6) or accessible debug menus furthermore underlines that certificate pinning should not be relied upon to verify identity nor authority.

## Analysis of similarities in IoT firmwares

Costin, Zaddach et al. (2014) performed a broad static analysis over a large number of firmware images to identify common patterns and similarities between firmwares of different product vendors. During the analysis of the 693 images, 38 new vulnerabilities were discovered, some which were present in the majority of images. A large number of 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*](https://github.com/ReFirmLabs/binwalk), [*ssdeep*](https://github.com/ssdeep-project/ssdeep), and [*sdhash*](https://github.com/sdhash/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*](https://www.zynamics.com/software.html) was used, which would compare similarities and differences in disassembled code.

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, access to a shell 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 (however unsuccessfully) to gain shell access via sideloading methods.

## 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 boot[[7]](#footnote-7), 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.

## 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*](https://www.flashrom.org/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. Supposed randomly 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threat** | | **TS0** | **TS1** | **TS2** | **TS3** |
| - | Physical (proximal) | Remote (proximal) | Remote (distal) |
| **Concern** | **Physical Access** | // | x |  |  |
| **Remote Access** |  | x | x | x |
| **Data Ownership** | x |  |  |  |
| **Data Visibility** | x | x | x | // |

Table 1 - Threat model matrix

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 other parties extracting data from the device or gaining remote control to 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, it was observed that the findings observed did not cause discussion specific to areas of only privacy or security, but often both. 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_7_|) chapter.

## Preliminary Device Access

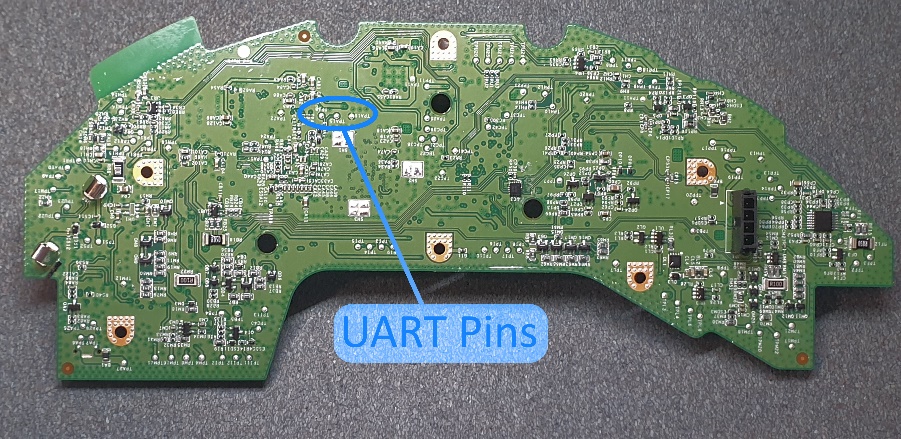


Figure 1 - UART pin locations

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 sequence[[8]](#footnote-8). 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

|  |  |  |
| --- | --- | --- |
| **Step** | **Command** | **Description** |
| **1** | ext4load mmc 2:6 0 vinda | Load contents of vinda into memory position 0 |
| **2** | md 0 4 | Dump the first 4 words from memory position 0 |
| **3** | ------------------------ | XOR values with 0x37 |

Table 2 - Root password extraction

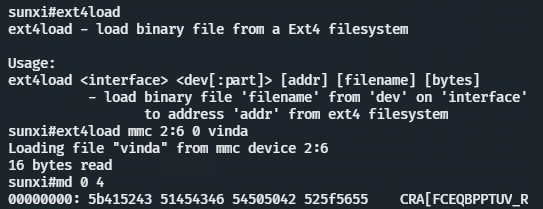


Figure 2 - Password decryption of the vinda file

## 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, which is one year after the official release of the Roborock S6 vacuum cleaner in June 2019. As a result, the stock / recovery firmware (as stored in mmcblk0p7) is versioned 01.15.58 (released 25th March 2020). All firmware investigations and results collected in this thesis were performed against version 01.15.58 until the upgrade analysis section on page 22.

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 webpage[[9]](#footnote-9). 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.

|  |  |
| --- | --- |
| **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 |

Table 3 - v01.15.58 System Fingerprint

### Process Capability

An instance of htop - a process viewer utility[[10]](#footnote-10) - was loaded on to the device to monitor the running processes as shown in Figure 3. 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.

Graphical user interface, text

Description automatically generated

Figure 3 - Process list (v01.15.58)

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.

### 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*](https://openwrt.org/) (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, rending the device unreachable via IPv6

A screenshot of a computer

Description automatically generated with medium confidence

Figure 4 - netstat (v01.15.58)

|  |  |
| --- | --- |
| Figure 5 - iptables (v01.15.58) | Figure 6 - ip6tables (v01.15.58) |

Text

Description automatically generated

Figure 7 - ifconfig (v01.15.58)

### User Enumeration

No additional 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 (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.

|  |  |
| --- | --- |
| Figure 8 - /etc/passwd (v01.15.58) | Graphical user interface, text, website  Description automatically generated  Figure 9 - /etc/shadow (v01.15.58) |

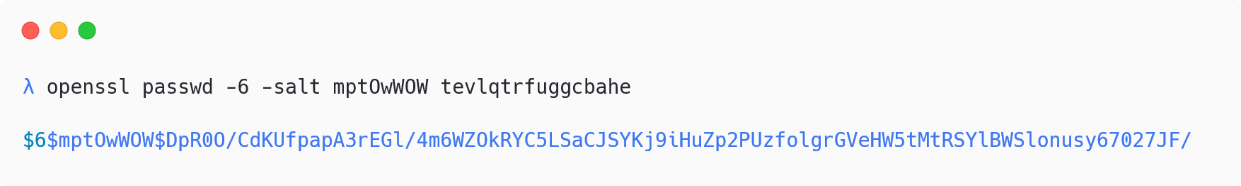


Figure 10 - Generated SHA512 password hash

### Log File Analysis

TODO;

### 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. Whilst results for upgrade persistence and reset persistence were sensible, the results from device disassociation were alarming, as no data was removed after a device was deleted from a 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 the device under test is passed around.

Whilst statistical and calibration data (mmcblk0p11) are retained during firmware upgrades and factory resets, it can be noted from Figure 11 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[[11]](#footnote-11) from recovering data.

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

Table 4 - Directories untouched during volatile actions



Figure 11 - Serial log during factory reset

### Power Analysis

A power analysis was performed in order 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 3 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 ~4.2VDC when there is no vacuum connected, and ~20.4VDC when the vacuum is loaded (with an equivalent resistance of 3.7 kΩ)

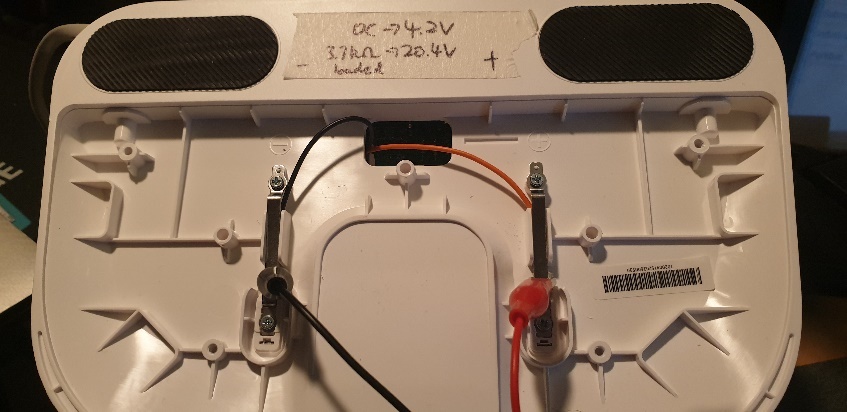


Figure 12 - Exposed underside of the charging dock

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

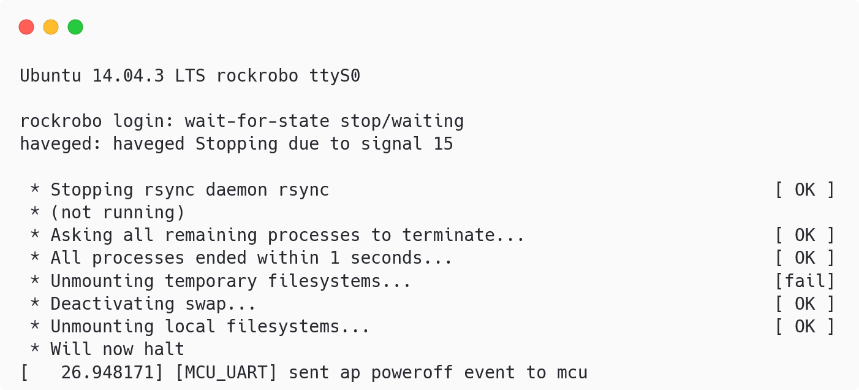


Figure 13 - 2-wire battery shutdown log

### Filesystem Modification

Backdoor persistence (since network capability, and process capability)

rrwatchdoge.conf

iptables

boot / cron

Reset persistence

Upgrade persistence

<https://featherbear.cc/UNSW-CSE-Thesis/posts/achieving-upgrade-persistence/>

* iptables -D INPUT -p tcp -m tcp --dport 22 -j DROP

## 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 3, a set of eMMC partition dumps were created, which have been tabulated as shown in Table 3   
Table 3 - Partition Mappingbelow.



Figure 14 - Firmware dump commands

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Partition** | **Label** | **Size** | **Mount Point** | **Description** |
| **1** | UDISK | 1.5 GB | /mnt/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 |

Table 5 - Partition Mapping

It should be noted that the device contains two copies of the latest operating system firmware, labelled system\_a and system\_b. If the system fails to boot properly, a hardware watchdog will reboot 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.

mmcblk0p11

| mcu\_ready

| try

| hwinfo

| anonymousid1

| lds\_calibration.txt

| counter

| CompassBumper.cfg

| blackbox.db

| rrBkBox.csv

| endpoint.bin

| RoboController.cfg

\---rriot

tuya.json

### Commentree

Graphical user interface, text

Description automatically generated

Figure 15 - Screenshot of the Commentree tool

### Firmware Analysis (Version 01.15.58)

#### Secret, Configuration, Logs Search

<https://featherbear.cc/UNSW-CSE-Thesis/posts/logviews/mmcblk0p7-var-log-apt-history.log/>

AppProxy

RoboController

rr\_loader

wlanmgr

WatchDoge

Rrlogd

Rriot\_tuya

#### rrlogd

<https://featherbear.cc/UNSW-CSE-Thesis/posts/execs/opt-rockrobo-rrlog-rrlogd-v01.15.58/>

* Device data
* Application config
* Application logs
* SLAM (map)
* Running processes
* Wireless configuration
* Packet capture
* Blackbox (statistics)

TODO: Get actual list of files?

#### Stock Ubuntu Comparison

<https://featherbear.cc/UNSW-CSE-Thesis/posts/files-in-device-that-are-missing-from-base/>

<https://featherbear.cc/UNSW-CSE-Thesis/posts/diff-inspection/>

<https://featherbear.cc/UNSW-CSE-Thesis/posts/files-in-device-that-are-missing-from-base/>

<https://featherbear.cc/UNSW-CSE-Thesis/posts/file-system-comparison/>

#### miio

https://featherbear.cc/UNSW-CSE-Thesis/posts/logviews/mnt-data-rockrobo-rrlog-miio.log/

### Upgrade Analysis (Version 02.29.98)

#### Software Changelong

> 01.17.08 (17th April 2020)

\* /opt/rockrobo/watchdog/WatchDoge iptables drop SSH

\* /opt/rockrobo/rrlog/rrlogd iptables drop SSH

\* Utilities change to busybox

\* SSH server changed to dropbear

\* /opt/rockrobo/rriot/rriot\_rr added (but not enabled)

> 01.19.98 (9th June 2020)

\* Serial handler changed to /sbin/rr\_login

> 01.20.76 (23rd June 2020)

-

> probably more updates (got locked out)

> 02.29.02 (28th April 2022)

\* /opt/rockrobo/rriot/rriot\_rr enabled

#### Firmware Images

Firmware is encrypted and signed

#### IPv6

Ip6tables

SSH access dropped by rrlogd and WatchDoge

#### Software Changes

<https://featherbear.cc/UNSW-CSE-Thesis/posts/versioning-and-fingerprinting-v02.29.02/>

<https://featherbear.cc/UNSW-CSE-Thesis/posts/opt-roborock-md5-hashes/>

Busybox

Dropbear

#### rr\_login

<https://featherbear.cc/UNSW-CSE-Thesis/posts/execs/sbin-rr_login/>

getty-> rr\_login

root only login

Graphical user interface

Description automatically generated

#### rrlogd

<https://featherbear.cc/UNSW-CSE-Thesis/posts/execs/opt-rockrobo-rrlog-rrlogd-v02.29.02/>

#### Authentication Flow

No longer use vinda

Upgrade -> shadow  
New devices are harder to modify, need to modify the shadow AND shadow.sign file (RO in live system)  
Of course can patch the binary

<https://featherbear.cc/UNSW-CSE-Thesis/posts/v02.29.02-firmware-size-comparison/>

<https://featherbear.cc/UNSW-CSE-Thesis/posts/v02.29.02-accessing-the-shell/>

#### rriot\_rr

/dev/shm/.migrate\_to\_rriot

Instructed by AppProxy

#### Wlanmgr

<https://featherbear.cc/UNSW-CSE-Thesis/posts/upgrade-notes/>

<https://featherbear.cc/UNSW-CSE-Thesis/posts/tcpdump-usage-v02.29.02/>

tcpdump

### ADB

<https://featherbear.cc/UNSW-CSE-Thesis/posts/13-07-2022-progress-update/>

<https://featherbear.cc/UNSW-CSE-Thesis/posts/vinda.c/>

<https://featherbear.cc/UNSW-CSE-Thesis/posts/execs/usr-bin-adbd/>

Command injection vulnerability exists within the modified adbd binary

**What’s modified?**

Interface to perform uart\_test and ruby\_flash

Authenticated access to adb shell

Dynamic challenge/response

Requires knowledge of vinda, device ID

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\*]

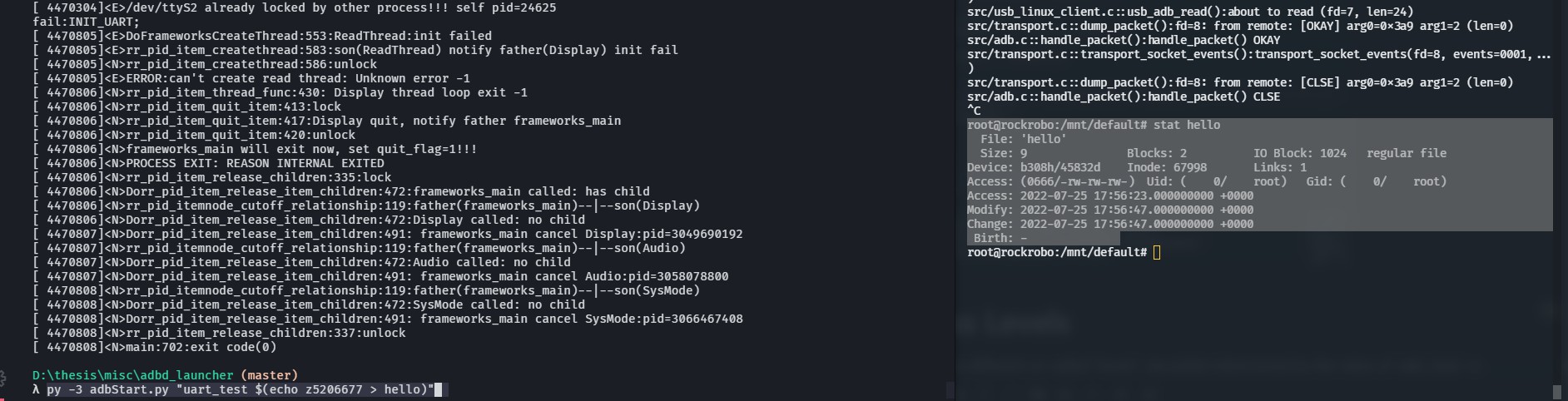
* The modified binary has some sort of access level implementation
  + Depends on value in /mnt/default/adb.conf (RO)
* Arbitrary command execution when access level = 0
  + But the app also resets this value to 1
  + &, ;, |, ` characters are also forbidden

😢 No arbitrary command execution…

$> py -3 adbStart "whoami"

src/rr\_ruby.c::adb\_check\_unlock\_level1():not support /adb shell sys\_passwd#adb\_passwd whoami in level 1

RCE via command substitution

Text

Description automatically generated

Avoiding the forbidden characters (&, ;, |, `) we can exploit command substitution and redirections to inject commands.

Allows us to write to the filesystem

Or read from the filesystem too!

Text

Description automatically generated

POC breakdown (Where it falls apart)

Still need to authenticate before RCE possible

Still need knowledge of the /mnt/default/vinda file

Need to physically open the device at least once

Screws. Lots of them.

At least, provides a way to issue commands even when adb\_lock != 0

USB protocol is more common and accessible to people

SSH access might stop working / be blocked (spoilers)

Serial access might stop working / be blocked (spoilers)

## 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

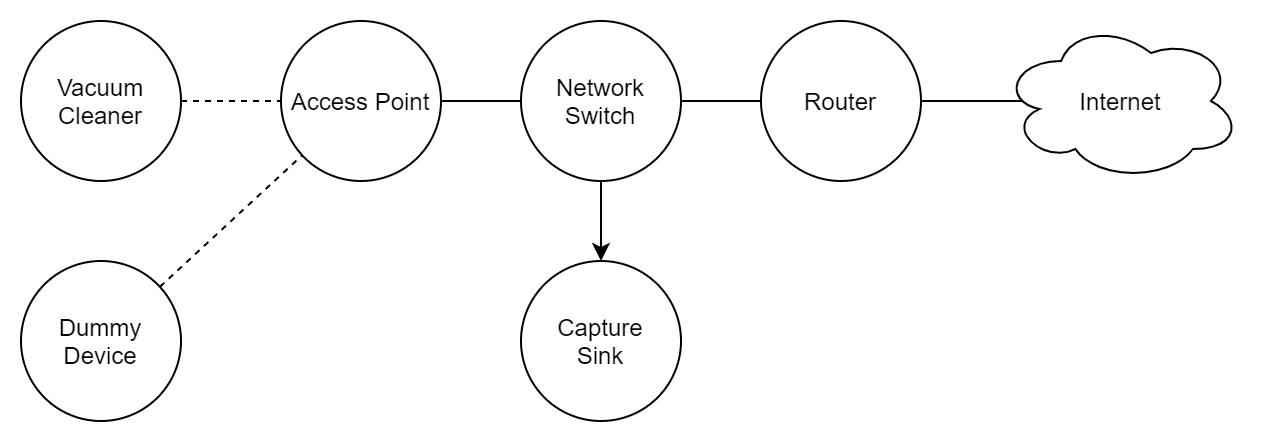


Figure 16 - Isolated network connection diagram

|  |  |  |
| --- | --- | --- |
| **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 |

Table 6 - Network equipment list

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.

It should be pointed out that given the port mirroring functionality limitations of the network switch used in this thesis (TP-Link TL-SG105E), modifications to the capture sink’s NIC settings were performed 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 the majority of captures being performed whilst the device was idle - as it would best reveal any patterns of network activity.

### Binary Patching Preparation

As consequence to having the ability to modify and executed modified binary files

Frida - https://featherbear.cc/UNSW-CSE-Thesis/posts/frida-objection-https-without-mitm/

<https://featherbear.cc/UNSW-CSE-Thesis/posts/pre-encryption-and-post-decryption-hooks/>



Figure 17 - Crypto function hook source code

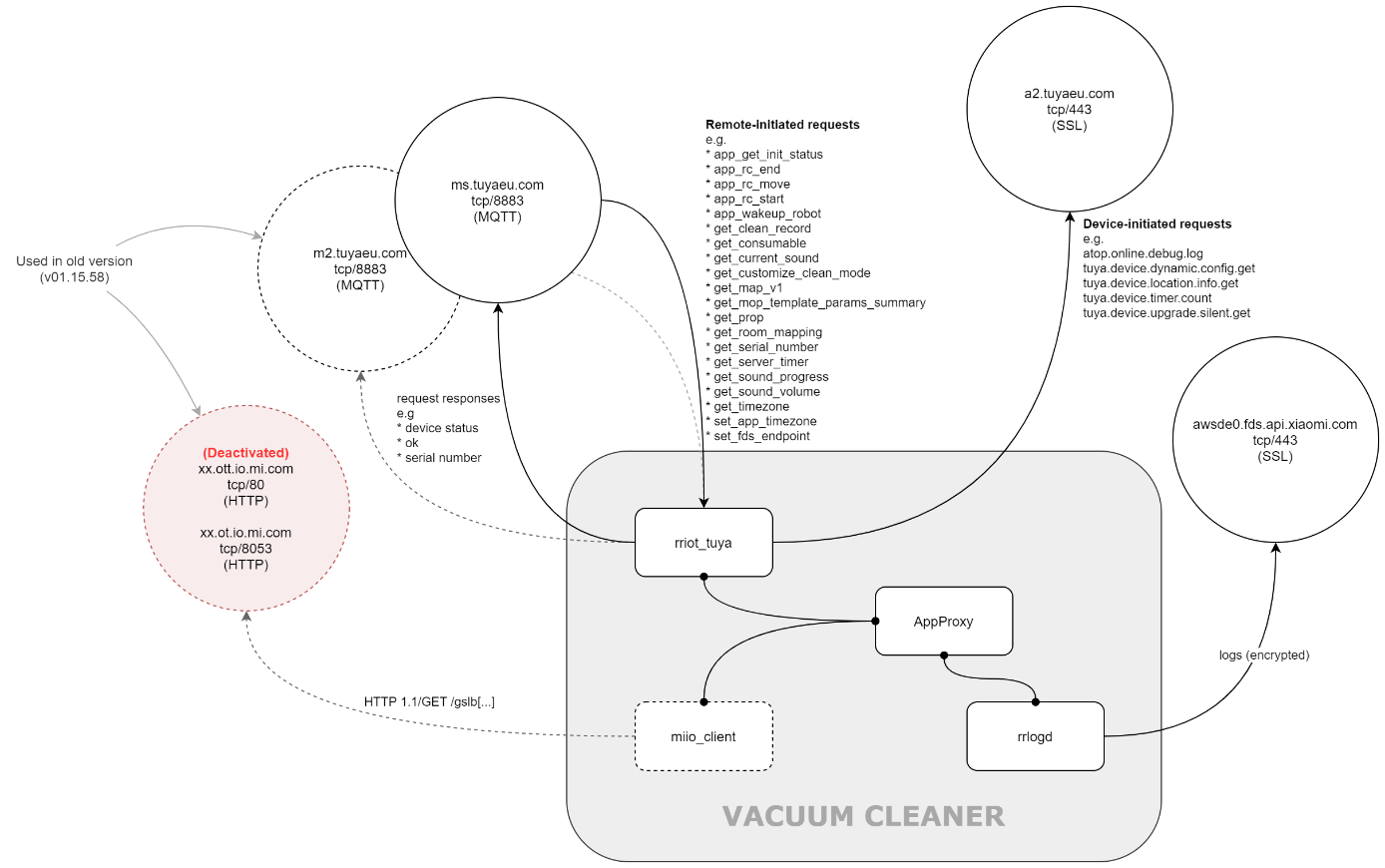
### Cleaning of Ground Truth

Device traffic (STP, etc)

### Observation of Packet Capture

<https://featherbear.cc/UNSW-CSE-Thesis/posts/packet-analysis/>

#### Network Capture Data



What data?  
Binary Patch + Log data

Data is compressed and encrypted

* /mnt/data/rrlog/\*\*
* /dev/shm/\*\*
* /mnt/reserve/...
* App logs
* Updater logs
* ‘Anonymous’ statistics
* wlanmgr tcpdump

Privacy Policy, available on the Roborock smartphone

Website shows different

<https://featherbear.cc/UNSW-CSE-Thesis/posts/privacy-policy-from-apk/>

Effective: 30th April 2019

Cleaning-related information ... last 20 items will be saved by your device and server...

...stored in the server for up to 180 days, ... automatically deleted after expiration.

Network information: ...the password information is only stored on the device side...

... will not be uploaded to the server.

Timing information... Cleanable Area Information... Other information: For example, ....

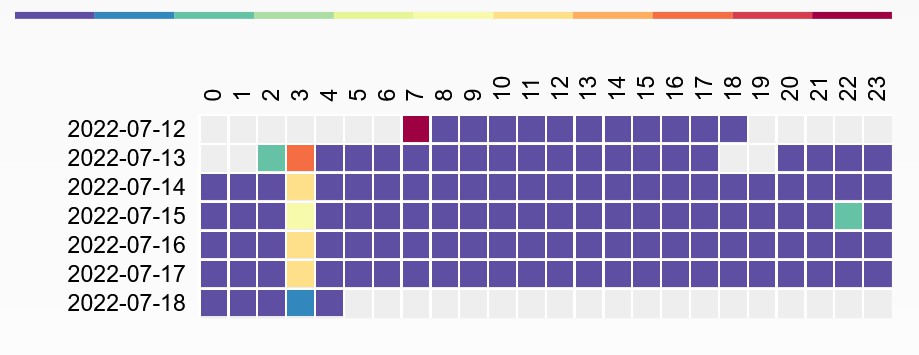
Text

Description automatically generated

Above: Log data uploaded by rrlogd | FW: v02.29.02 (28th April 2022)

(TODO: Put this in the network analysis section?)

#### Behaviour



| **Endpoint** | **Protocol** | **Description** |
| --- | --- | --- |
| m2.tuyaeu.com | MQTT | Inbound requests |
| a2.tuyaeu.com | HTTPS | Outbound requests |
| awsde0.fds.api.xiaomi.com | [FDS](http://docs.api.xiaomi.com/en/fds/) | Logs upload |

Network Geomap (FW v02.29.02) (exc FDS) (1 week)Graphical user interface, application

Description automatically generated

Observations

Local traffic - DHCP (5min), Tuya Discovery (5s)

Connections to America, Germany, China

America, Germany - AWS - fds, a2, ms, m2

China - Mi IO Cloud (v01.15.58 only)

Increased network activity at 3am

3am AEDT is 12am in Beijing

Connections are being established - possible timeout/reconnect

Changes

New FW uses m2.tuyaeu.com instead of ms.tuyaeu.com

New FW no longer polls xx.ot[t].io.mi.com

#### 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 line with a configuration parameter ONLY\_UPLOAD\_ONDOCK=1 found within the rrlog.conf file.

#### 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).

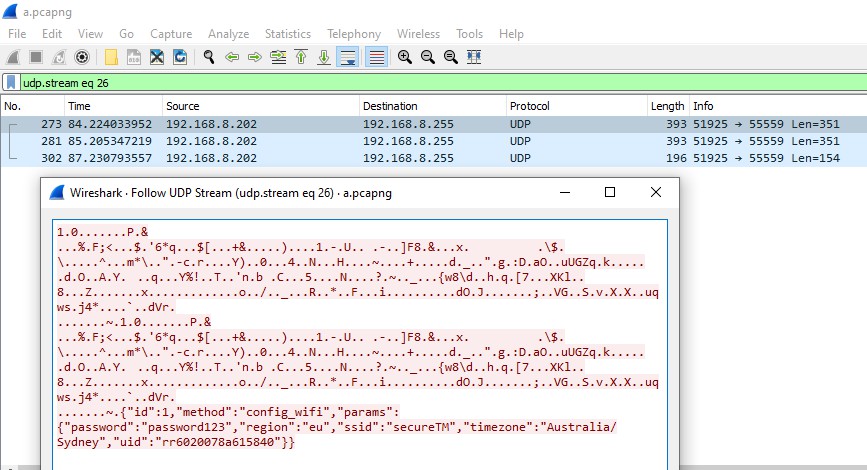


Figure 18 - Plain-text credential transmission during pairing

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 7. 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).

### MUD

https://featherbear.cc/UNSW-CSE-Thesis/posts/mud/

<https://github.com/ayyoob/mudgee>

MUD profiles for versions: v01.15.18, v02.29.02

Show flow table

\*\*Limitations: Baseline ground-truth

\*\*There are likely other ports and addresses that were not identified during the packet captures

\*\*Limitations: Payload packing

\*\*MUD files do not perform deep packet inspection, nor does it decode SSL traffic. Limited to blocking unexpected addresses and ports.

## OTA Rooting

<https://featherbear.cc/UNSW-CSE-Thesis/posts/13-07-2022-progress-update/>

<https://featherbear.cc/UNSW-CSE-Thesis/posts/ota-updates-blocked-as-of-late-2019/>

## Device Entry Analysis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Serial (UART) | USB (ADB) | SSH | MiIO OTA |
| Requires Root Password | // | y | y | NO |
| Requires Initial Disassembly | |  |  | NO |
| Patched / Restricted | y | y | y | y |

shadow and shadow.sign

Text

Description automatically generated

[(1)](https://featherbear.cc/UNSW-CSE-Thesis/posts/upgraded-accessing-the-shell/) Enter bootloader and force entrypoint to a shell  
[(2)](https://featherbear.cc/UNSW-CSE-Thesis/posts/upgrade-notes/#notice) Patch /etc/inittab to revert back to a normal login shell

# Chapter 6 | Discussion

## Summary of Contributions

## Commentary

Answer Research Questions

Support and Defend answers with results

Explain

* Conflicting results
* Unexpected findings
* Discrepancies with other research

State limitations of the study

State importance of findings

Establish newness

Good

* Restrict SSH
  + Upgrade also limit
* AuthN and AuthZ for all interfaces?
* SSH server, but blocked
* Logs encrypted (good but also bad)

Ipv6

Processes as root – udev… takeover

Recovery parttion modifiabile

* Could encrypt

Pairing - RSA 1024 / ECB / PKCS1 + AES / CBC / PKCS7

https://featherbear.cc/UNSW-CSE-Thesis/posts/disclosures/

* Stock firmware was last 2020
  + New firmware may do better with pairing
  + Can’t verify without getting another device manufactured later

## Future Work

Announce further research

* MITM / HSTS
* OTA
* Mobile app
* Fuzz other things

# Bibliography

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

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.

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.

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.

1. <https://www.shodan.io/search?query=webcam> [↑](#footnote-ref-1)
2. <https://www.espressif.com/en/products/socs/esp8266> [↑](#footnote-ref-2)
3. <https://www.heise.de/newsticker/meldung/Smart-Home-Hack-Tuya-veroeffentlicht-Sicherheitsupdate-4292028.html>  
    [↑](#footnote-ref-3)
4. <https://github.com/dgiese/dustcloud> [↑](#footnote-ref-4)
5. <https://github.com/nabla-c0d3/ssl-kill-switch2> [↑](#footnote-ref-5)
6. <https://github.com/shroudedcode/apk-mitm> [↑](#footnote-ref-6)
7. Generally triggered by pulling the [*FEL pin*](https://linux-sunxi.org/images/b/b3/R16_Datasheet_V1.4_(1).pdf) (LRADC0) LOW during boot [↑](#footnote-ref-7)
8. <https://github.com/allwinner-zh/bootloader/blob/master/u-boot-2011.09/board/sunxi/board_common.c#L843-L847> [↑](#footnote-ref-8)
9. <https://global.roborock.com/pages/roborock-s6> [↑](#footnote-ref-9)
10. <https://github.com/htop-dev/htop> [↑](#footnote-ref-10)
11. <https://www.cgsecurity.org/wiki/PhotoRec> [↑](#footnote-ref-11)